# Mobile Application and Cloud Computing

## Questions, Answers & Theory

### ANDROID:

- Describe what an Activity is
- Describe the life cycle of an Activity
- Describe the Fragments with their life cycle
- Describe the differences between Fragment and Activity
- Describe the main features of Android Service components and the difference between Bound and Started Service in Android
- Describe the main features of the Services
- Bound Service and Started Service
- Difference between Activity and Service
- What is an Intent?
- Describe what an Intent Service is
- Intent filter
- Describe the function of the Broadcast Receiver in Android
- Describe how to get multithreading
- Describe the function of AsyncTask in Android
- Discussing security in Android
- Describe the permission
- Describe the concept of "Filter" and "Permission" in Android
- Content Providers

### WEB-API:

- Describe RPC and its structure
- Describe REST-like
- Describe static, dynamic and AJAX technologies
- Describe XML-RPC and JSON-RPC
- Describe the XML-RPC protocol
- Describe JSON-RPC and give some examples where it can be used

### PHP

- Describe what a superglobal variable is in PHP
- Describe the properties and use of $_SESSION in PHP
- Describe the main features and use of cookies in PHP

### CLOUD COMPUTING

- Describing Virtualization in the Context of Cloud Computing
- Describe the Cloud Computing and the types of services offered
- Describe the main features of SaaS
- Describe the pros and cons of PaaS and give an example
- Describe the main features of IaaS
- Describe what a hypervisor is

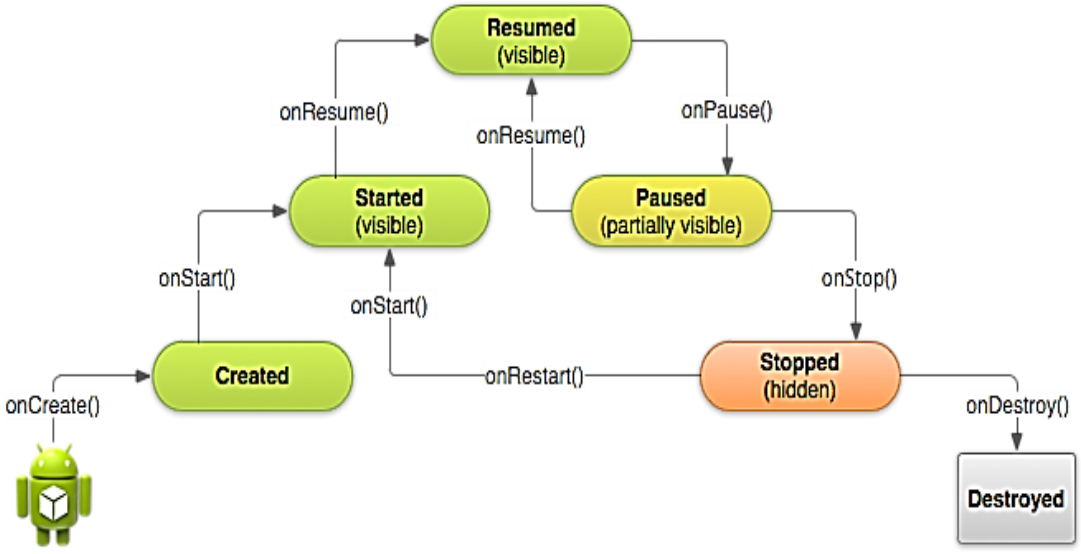
# ANDROID

## Describe what an Activity is

An Activity is a software component that provides a screen with which user can interact to do something. It runs inside the main thread (UI Thread) and should be reactive to user's input. Activities are managed by the Activity Manager. An Android application is composed of at least one Activity. An Activity can be created by the Launcher when the corresponding icon is pushed, or it can be created by another Activity or by another software component.

## Describe the life cycle of an Activity

An Activity is created by the Launcher when the corresponding icon is pushed. It can be also created by another Activity or by another software component. Activities are managed through a back stack. When the activity is created it is pushed on the top of the stack and it is active. When another activity is started, the previous activity remains below in the stack until the current activity is finished. An activity can be in 3 different states: ***running***, ***pause***, ***stopped***. A running activity is in foreground and has the focus, so the user can interact with it. A paused activity is partially visible but has not the focus: this can happen when another activity is started but it does not fill the entire screen. An activity is stopped when it is completely obscured by another one. Paused and stopped activities can be killed by the system when it needs resources. When an activity is created, the ***onCreate()*** method is called: it initializes all the views, data, etc., and is followed by the ***onStart()*** method. This method is called before the activity becomes visible, and when the activity is restarted after the stopped state (***onRestart()*** → ***onStart()***). The ***onResume()*** method is called when after the ***onStart()*** or when the activity is resumed from the pause state. The ***onPause()*** method is called before another activity is resumed. It is used to commit unsaved changes and to stop threads, animations, etc. It must be very fast because the other activity cannot be resumed until the current one returns. The ***onStop()*** method is called when the activity is no longer visible to the user. If the activity is going to be resumed, this method is followed by the ***onRestart()***, else if the activity is going to be destroyed the ***onDestroy()*** method is called.



## Describe the Fragments with their life cycle

Fragment is a part of an activity, which contributes its own UI to that activity. Fragment can be thought like a sub activity. Whereas the complete screen with which user interacts is called as activity. An activity act as a glue for multiple fragments. A fragment has its own layout and its own behaviour with its own lifecycle callbacks. You can add or remove fragments in an activity while the activity is running. You can combine multiple fragments in a single activity to build a multi-pane UI. A fragment can be used in multiple activities. The fragment life cycle is closely related to the lifecycle of its host activity. When the activity is paused, all the fragments available in the activity will also be stopped.
The lifecycle of the activity in which the fragment lives directly affects the lifecycle of the fragment, such that each lifecycle callback for the activity results in a similar callback for each fragment. For example, when the activity receives onPause(), each fragment in the activity receives onPause(). Fragments have a few extra lifecycle callbacks, however, that handle unique interaction with the activity to perform actions such as build and destroy the fragment's UI. These additional callback methods are:

- ***onAttach()*** called when the fragment has been associated with the activity (the Activity is passed in here).
- ***onCreateView()*** called to create the view hierarchy associated with the fragment.
- onActivityCreated() called when the activity's onCreate() method has returned.
- ***onDestroyView()*** called when the view hierarchy associated with the fragment is being removed.
- ***onDetach()*** called when the fragment is being disassociated from the activity.

## Describe the differences between Fragment and Activity

La differenza più significativa nel ciclo di vita tra un'activity e un fragment è il modo in cui uno è memorizzato nel suo rispettivo back stack. Un'activity viene inserita in una pila posteriore di activity che viene gestita dal sistema quando è ferma, per impostazione predefinita (in modo che l'utente possa tornare ad essa con il pulsante "Indietro"). Tuttavia, un fragment viene inserito in una pila posteriore gestita dall'activity dell'host solo quando si richiede esplicitamente che l'istanza sia salvata richiamando ***addToBackStack()*** durante una transazione che rimuove il fragment. Altrimenti, gestire il ciclo di vita del fragment è molto simile alla gestione del ciclo di vita dell'activity. Quindi, le stesse pratiche per la gestione del ciclo di vita dell'activity si applicano anche ai fragment. Quello che bisogna anche capire, però, è come la vita dell'activity influisce sulla vita del fragment.

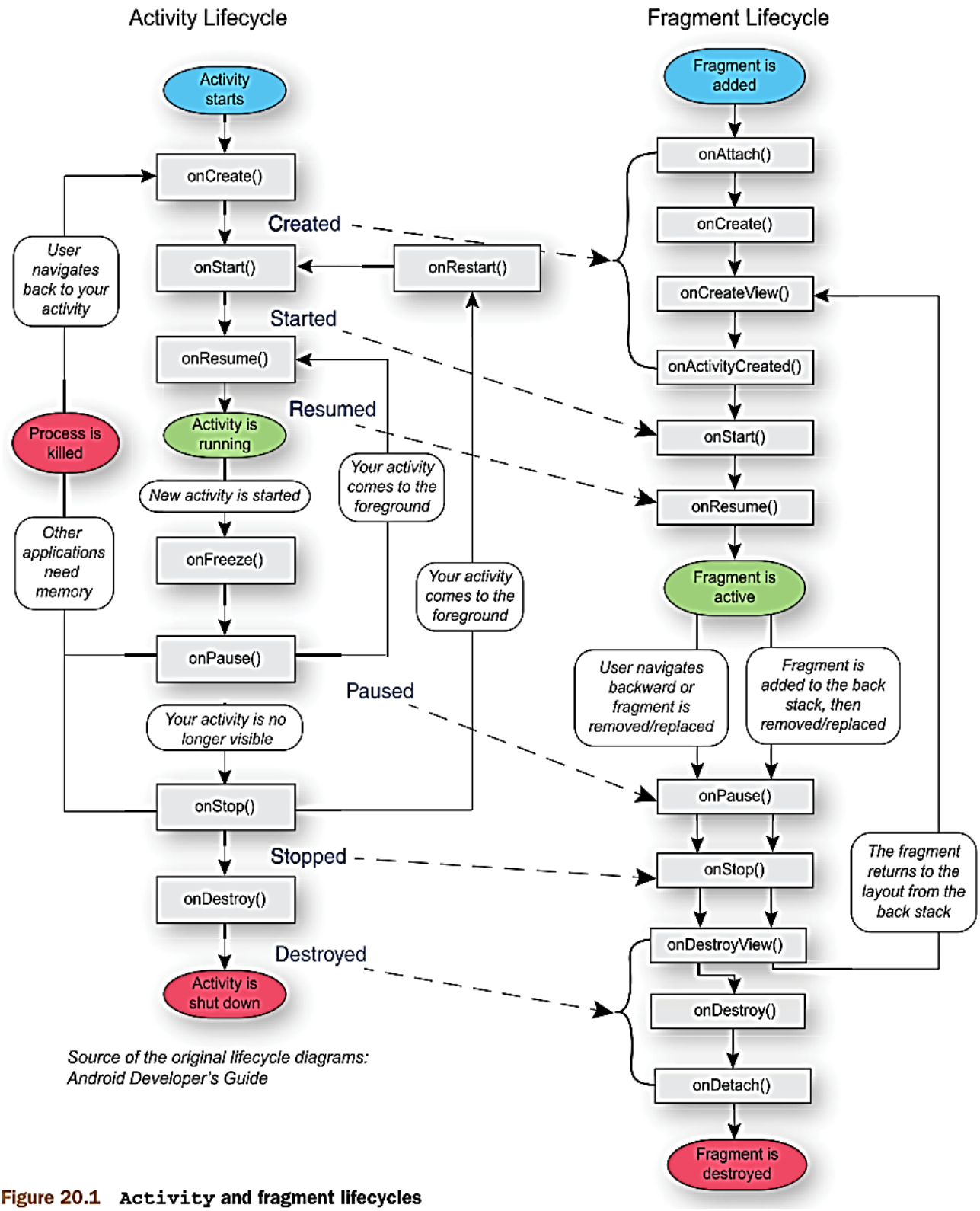



**Figure 20.1** **`Activity` and fragment lifecycles**

## Describe the main features of Android Service components and the difference between Bound and Started Service in Android

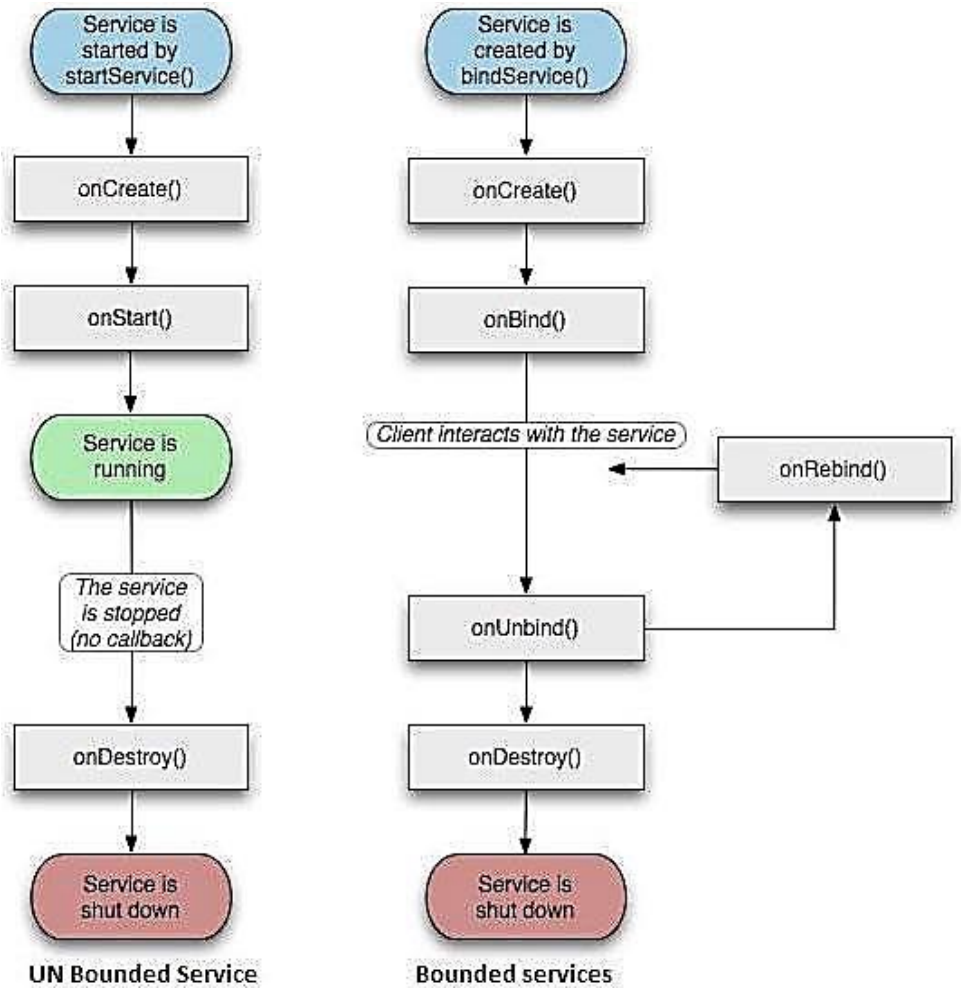
A Service is an application component that can perform long-running operations in the background and does not provide a user interface. Another application component can start a service and it will continue to run in the background even if the user switches to another application. Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC). For example, a service might handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background. A service can essentially take two forms:

- **Started Service**: a service is "started" when an application component (such as an activity) starts it by calling ***startService()***. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed. Usually, a started service performs a single operation and does not return a result to the caller. For example, it might download or upload a file over the network. When the operation is done, the service should stop itself.
- **Bound Service**: a service is "bound" when an application component binds to it by calling ***bindService()***. A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC). A bound service runs only if another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

## Describe the main features of the Services

Services are software components that perform long background operations, such as downloading files from the internet or playing music in background. They do not require a user interface. Services lifecycle is independent from that of the component that starts them, so they can continue running even if the starting component is finished. There are two types of services: **Started Services** and **Bound Services**. In both the service to be started is identified via an intent: it is recommended to use explicit intents because if two or more services respond to the same action, the system selects one of them at random. Started Services are used when something must be done in background and they do not provide results. They are started with ***startService().*** The ***onBind()*** method must return null. When a started service has completed his task, it should call the ***stopSelf()*** method; it can be also stopped by another component via the ***stopService()*** method. Services have a high priority in the system, so they are among the last processes to be terminated when the system needs resources. To run a service in foreground we can call ***startForeground():*** this method shows a notification while the service is in this state. Because it runs in foreground, it is less likely that the system kills the service. Bound Services are used to set up a connection with a service so that methods can be called. They are started with ***bindService()*** and are like started services, but they permit interaction with the component that launched them. A bound service allows the launching component to interact with and receive results from the service. Multiple components can bind to a service simultaneously. To unbind the service, a component calls ***unbindService()*** method: when all the bound clients unbind from the service, it is terminated by the system. A bound service can also be launched with a call to

***startService()*** and then components can bind to it with ***bindService().*** In this case the service will not terminate when all the clients have unbound from it.

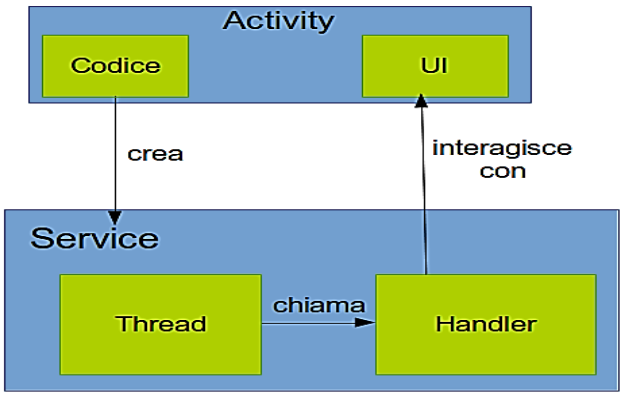



## Bound Service and Started Service

A service is started when an application component (such as an activity) calls startService(). After it's started, a service can run in the background indefinitely, even if the component that started it is destroyed. Usually, a started service performs a single operation and does not return a result to the caller. For example, it can download or upload a file over the network. When the operation is complete, the service should stop itself. A service is bound when an application component binds to it by calling bindService(). A bound service offers a client-server interface that allows components to interact with the service, send requests, receive results, and even do so across processes with interprocess communication (IPC). A bound service runs only if another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

## Difference between Activity and Service

An **Activity** is a software component that represents a single screen with a user interface. It runs in the UI Thread and should be reactive to user's input.
A **Service** is a software component that runs in background to perform long running operations. It does not provide a user interface and by default it runs in the same thread as the component that launched it. To process long running operations, it should create a separate working thread; this can be simplified by using IntentService class.




## What is an Intent?

An Intent is a messaging object that can be used to request an action from another app component. It facilitates communication between app components, but it can be used also to start an Activity, Service and delivering a broadcast. Intents can be explicit, if the specify the component to start, or implicit if the only specify the action to be performed.

## Describe what an Intent Service is

A service runs in the main thread, so if it must do a long running operation it should create a new thread that performs the task. **Intent Services** simplify this operation. **IntentService** is a subclass of Service and it sets up a working thread for handling asynchronous background tasks. The IntentService processes one request at a time in a FIFO order. Once the service has executed all the requests, it exits. To process a request, the ***onHandleIntent()*** method is called, so the programmer must implement it. An IntentService is a service, so it must be declared in the manifest file.

## Intent filter

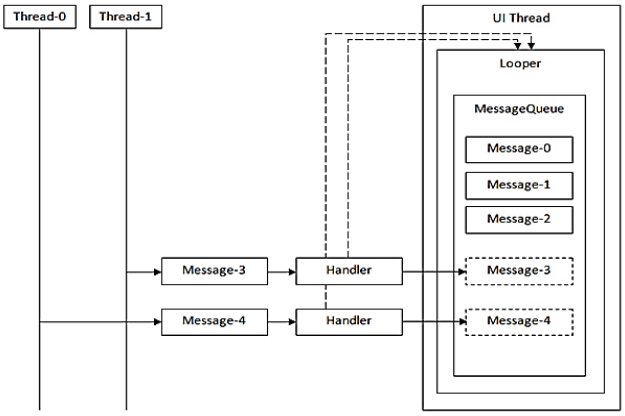
Android OS uses filters to pinpoint the set of Activities, Services, and Broadcast receivers that can handle the Intent with help of specified set of action, categories, data scheme associated with an Intent. You will use ***<intent-filter>*** element in the manifest file to list down actions, categories and data types associated with any activity, service, or broadcast receiver

## Describe the function of the Broadcast Receiver in Android

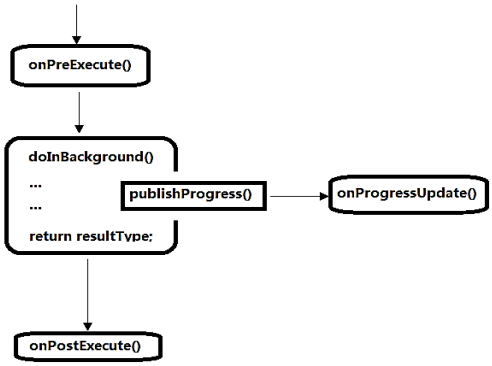
A broadcast receiver is a component which allows you to register for system or application events. All registered receivers for an event are notified by the Android runtime once this event happens. Part of the registration implementation involves the creation of intent filters to indicate the specific broadcast intents the receiver is required to listen for. This is achieved by referencing the action string of the broadcast intent. When a matching broadcast is detected, the ***onReceive()*** method of the broadcast receiver is called, at which point the method has 5 seconds within which to perform any necessary tasks before returning. It is important to note that a broadcast receiver does not need to be running all the time. Broadcast intents are Intent objects that are broadcast via a call to the ***sendBroadcast()***, ***sendStickyBroadcast()*** or ***sendOrderedBroadcast()*** methods of the Activity class (the latter being used when results are required from the broadcast). In addition to providing a messaging and event system between application components, broadcast intents are also used by the Android system to notify interested applications about key system events (such as the external power supply or headphones being connected or disconnected).

## Describe how to get multithreading

When the main activity of an application is started, it is executed on the main thread (UI Thread). This is responsible for dispatching events to the views inside the screen. The UI Thread should be very reactive to the user's input so long running operations should not be executed on it. There are several ways to implement multithreading. The first one is to use standard thread programming, extending the Thread class and overriding the ***run()*** method. If the working thread needs to update the UI it has to communicate with the main thread through a message-based mechanism: the UI Thread creates an internal Handler object and the working thread uses it to obtain an empty message and to send back messages to the main thread. The Handler allows to send Message and Runnable objects. Another way to implement multithreading is by using the AsyncTask class. It simplifies the interaction between the main thread and the working thread: it allows to perform operations in background (***doInBackground()*** method) and publish the results on the UI Thread (***onPostExecute()*** method). We can also perform background operations with IntentServices or using the Volley framework, that simplifies HTTP requests.




*Metodo 1*




*Metodo 2 (AsyncTask)*

## Describe the function of AsyncTask in Android

AsyncTask is an abstract Android class which helps the Android applications to handle the Main UI thread in efficient way. AsyncTask allows us to perform long lasting tasks/background operations and show the result on the UI thread without affecting the main thread, so that it remains responsive. The main methods are:

- ***onPreExecute()*** is invoked on the UI thread before the task is executed. It is generally used to create dialog box or progress bar and carry out initialization.
- ***doInBackground()*** invoked on the background thread immediately after onPreExecute() finishes executing. Generally used to perform background computation that can take a long time. The parameters of the asynchronous task are passed to this step.
- ***onPostExecute(Result)*** invoked on the UI thread after the background computation finishes. The result of the background computation is passed to this step as a parameter.

## Discussing security in Android

The first security measure adopted by Android is the use of sandboxes. Each application runs in an isolated sandbox provided by a virtual machine and is assigned a unique Linux user ID. Any data stored by an application will be assigned that application's user ID, so only the user ID assigned to that app can access them. Each app has access only to the components it really needs. It is possible to share data with other application through permissions. The app requests a set of permissions at installation time, and the user must grant either all or none of such permissions. Starting with Android 6.0 permissions are granted at runtime. Android security has also weaknesses: the market is open; apps are easy to reverse engineer, so the repackaging for malware injection is facilitated; the inter component communication is vulnerable, so apps may unintentionally expose sensitive interfaces; malicious apps can invoke native code through Java JNI to leverage corruption memory bugs.

## Describe the "Permissions"

Android is a privilege-separated operating system, in which each app runs with a distinct system identity (Linux user ID and group ID). Parts of the system are also separated into distinct identities. Linux thereby isolates apps from each other and from the system. Because each Android app operates in a process sandbox, apps must explicitly request access to resources and data outside their sandbox. They request this access by declaring the permissions they need for additional capabilities not provided by the basic sandbox, in the manifest. Depending on how sensitive the area is, the system may grant the permission automatically, or may prompt the user to approve or reject the request. If the device is running Android 6.0 (API level 23) or higher, and the app's targetSdkversion is 23 or higher, the app requests permissions from the user at run-time. The user can revoke the permissions at any time, so the app needs to check whether it has the permissions every time it runs. If the device is running Android 5.1 (API level 22) or lower, or the app's targetSdkVersion is 22 or lower, the system asks the user to grant the permissions when the user installs the app. If you add a new permission to an updated version of the app, the system asks the user to grant that permission when the user updates the app. Once the user installs the app, the only way they can revoke the permission is by uninstalling the app.

## Describe the concept of "Filter" and "Permission" in Android

An **intent filter** is an expression in an app manifest file that specifies the type of intents that the component would like to receive. Declaring an intent filter for an activity, we make it possible for other apps to directly start that activity with a certain kind of intent. So, when we create an implicit intent, the system finds the appropriate component to start by comparing the content of the intent to the intent filters declared in the manifest file of other apps on the device. Likewise, if we do not declare any intent for an activity, then it can be started only with an explicit intent. Each Android application runs in a process sandbox, that provides access to a limited number of system resources. Therefore, applications must explicitly share resources and data. They do this by declaring the permissions they need for additional capabilities not provided by the basic sandbox. The access to a resource is restricted using a permission. User should declare the use of the permission in the manifest file and grant all permissions at installation time. Starting with Android 6.0 permissions are granted at runtime immediately before an app needs it.

## Content Provider

**Content providers** can help an application manage access to data stored by itself, stored by other apps, and provide a way to share data with other apps. They encapsulate the data and provide mechanisms for defining data security. Content providers are the standard interface that connects data in one process with code running in another process.

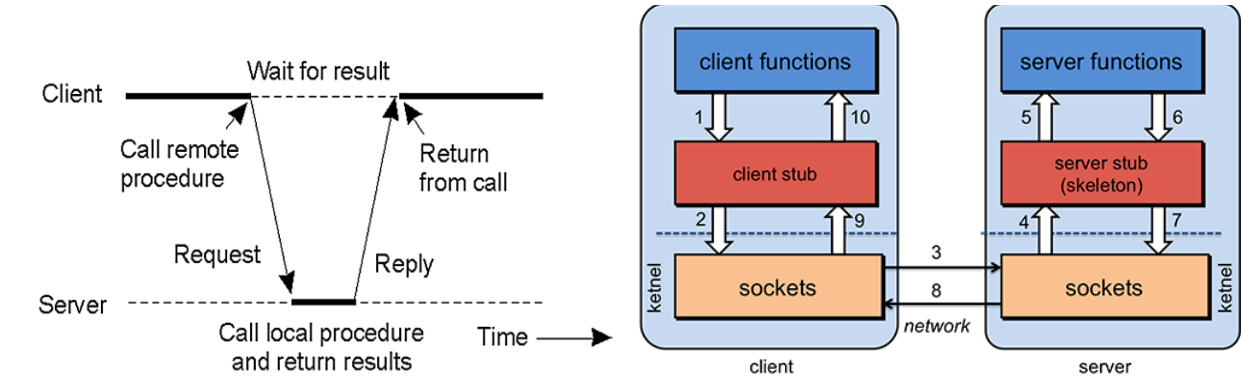
# WEB-API

## Describe RPC and its structure

A **Remote Procedure Call** (**RPC**) is when a program causes a procedure, or subroutine, to execute on a machine different from the one that is executing the program. The caller (client) makes a local call to a software component (**stub**) that communicates with the remote object. On the called (server) side there is a component (**skeleton**) that receives the stub call and makes a local call to the server code. RPC uses the HTTP protocol and data are represented in JSON or XML. Generally, the RPC execution flows works as follows:

1. The client calls the client **stub** like any usual local procedure call.
2. The client **stub** packs the parameters into a message and makes a system call to send the message.
3. The client's local operating system sends the message from the client machine to the server machine.
4. The local operating system on the server machine passes the incoming packets to the server **skeleton**.
5. The server **skeleton** unpacks the parameters from the message.
6. Finally, the server **skeleton** calls the server procedure.

The reply traces the same steps in the reverse direction.




## Describe REST-like

It is the simplest form of RPC, requests are mapped to HTTP methods (**GET**, **POST**). The method name is a parameter of the call and the result can be formatted as XML or JSON data. Its name derives from **RESTful Web Services**: the protocol is stateless, every resource is identified by a URI and the CRUD operations (Create, Read, Update, Delete) are mapped to the HTTP verbs (**POST**, **GET**, **PUT**, **DELETE**).

## Describe static, dynamic and AJAX technologies

In the static case the call is made as soon as the page is loaded. In the dynamic there is a button associated to an event listener that makes the call AJAX makes asynchronous call; allows for web pages to change content dynamically without the need to reload the entire page.

## Describe XML-RPC and JSON-RPC

- **XML-RPC** is a remote procedure call protocol which uses XML to encode data. Data that can be converted into XML are scalars, arrays or structs. There are also three types of messages: request message, response message and error message. The root element of a request message is the ***methodCall***; it contains a ***methodName*** element and a ***params*** element. The former contains the name of the procedure being called and the latter contains a list of values for the parameters of the procedure. The root element of a response message is the ***methodResponse***; it contains a ***params*** element, which is a list of values.
- **JSON-RPC** is a remote procedure call protocol encoded in JSON. The request must contain three properties: ***method*** is a string with the name of the method to be invoked; ***params*** is an array of objects to be passed as parameters to the defined method; ***id*** is a value of any type, used to match the response with the request that it is replying to. The response must contain three properties too: ***result*** is the data returned by the invoked method; if an error occurred while invoking the method, this value must be null; ***error*** is a specified error code if an error occurred, otherwise it is null; ***id*** is the id of the request it is responding to. Notification is possible: in this case the id property must be set to null or not set.

## Describe the XML-RPC protocol

**XML-RPC** works by sending an HTTP request to a server implementing the protocol. The client in that case is typically software wanting to call a single method of a remote system. Multiple input parameters can be passed to the remote method, one return value is returned. The parameter types allow nesting of parameters into maps and lists, thus larger structures can be transported. Therefore, XML-RPC can be used to transport objects or structures both as input and as output parameters. In comparison to REST, where resource representations (documents) are transferred, XML-RPC is designed to call methods.

## Describe JSON-RPC and give some examples where it can be used

**JSON-RPC** is a remote procedure call protocol encoded in JSON. The request must contain three properties: ***method*** is a string with the name of the method to be invoked; ***params*** is an array of objects to be passed as parameters to the defined method; ***id*** is a value of any type, used to match the response with the request that it is replying to. The response must contain three properties too: ***result*** is the data returned by the invoked method; if an error occurred while invoking the method, this value must be null; ***error*** is a specified error code if an error occurred, otherwise it is null; ***id*** is the id of the request it is responding to. Notification is possible: in this case the id property must be set to null or not set. JSON-RPC can be used as a simple request and response system, or it can be used to send notifications to all clients, or to send a call batch and to receive a response batch.

# PHP

## Describe what a superglobal variable is in PHP

**Superglobal** variables are predefined system variables that provide access to runtime data elements. They are set and managed through web-server runtime environment and are available in all scopes throughout a script. Superglobals are used to form processing, cookies, and other techniques. The most used superglobals are:

- ***$_GET[]***, an array that includes all the ***GET*** variables that PHP received from the client browser.
- ***$_POST[]***, an array that includes all the ***POST*** variables that PHP received from the client browser.
- ***$_COOKIE[]***, an array that includes all the cookies that PHP received from the client browser.
- ***$_SERVER[]***, an array with the values of the web-server variables.
- ***$_SESSION[]***, an array with values concerning a session.

## Describe the properties and use of $_SESSION in PHP

***$_SESSION*** is a **superglobal** variable containing an array with values concerning a session. A session is a way to store user information, in variables, to be used across multiple pages. This is useful because HTTP is a stateless protocol. Unlike cookies, session informations are not stored on the user's computer. A session is started with the ***session_start()*** function.
When a session is started, PHP will either retrieve an existing session using the ID passed (usually from a session cookie) or if no session is passed it will create a new session. PHP will populate the ***$_SESSION*** superglobal with any session data after the session has started. When PHP shuts down, it will automatically take the contents of the ***$_SESSION*** superglobal, serialize it, and send it for storage using the session save handler. Sessions can be started manually using the ***session_start()*** function. If the ***session.auto_start*** directive is set to 1, a session will automatically start on request startup. Sessions normally shutdown automatically when PHP is finished executing a script but can be manually shutdown using the ***session_write_close()*** function.

## Describe the main features and use of cookies in

Since HTTP is a stateless protocol, we must adopt some mechanist to keep track of the state and how it evolves in time. **Cookies** are text files stored on the client computer and they are kept of use tracking purpose. Server script sends a set of cookies to the browser. For example, name, age, or identification number etc. Browser stores this information on local machine for future use. When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user. One relevant cookie is the **PHPSESSID** that contains an identifier for the current session

# CLOUD COMPUTING

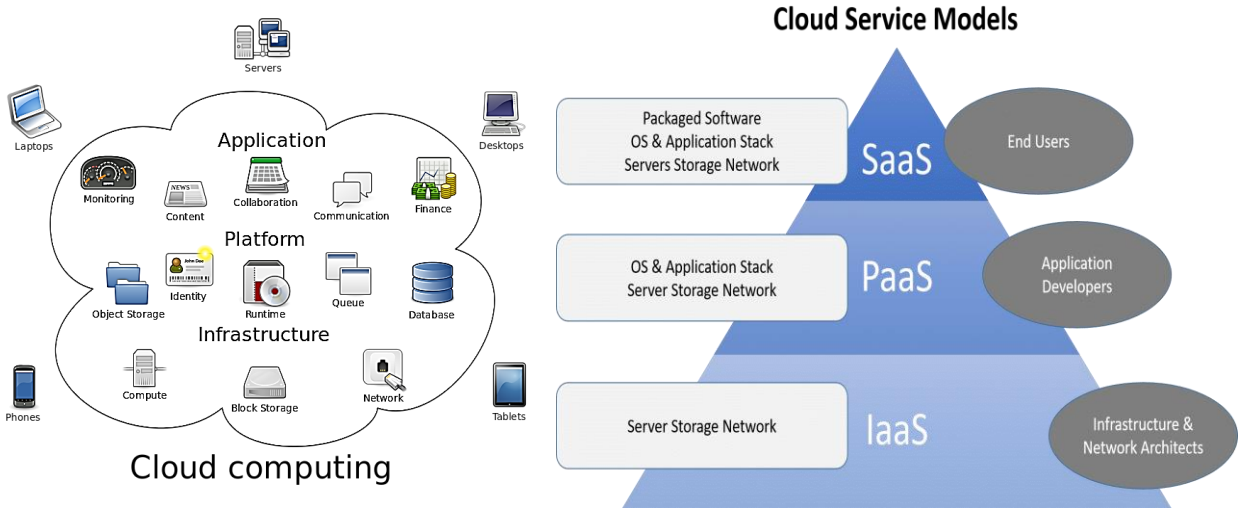
## Describing Virtualization in the Context of Cloud Computing

**Virtualization** in particularly useful in case of IaaS because these services are usually backed by largescale data centers composed of thousands of computers and to be able to serve many users and host many disparate applications they offer virtualized resources (computation, storage, and communication). Virtualization offer the main advantages of **isolation**, **mobility** and **consolidation**. Workload isolation is achieved since all program instructions are fully confined inside a VM, which leads to improvements in security. Better reliability is also achieved because software failures inside one VM do not affect others. Moreover, better performance control is attained since execution of one VM should not affect the performance of another VM. Encapsulating a guest OS state within a VM and allowing it to be suspended, fully serialized, migrated to a different platform, and resumed immediately or preserved to be restored later. Therefore, allows for facilitating hardware maintenance, load balancing, fault tolerance and disaster recovery. Finally, virtualization makes it possible to consolidate individual workloads onto a single physical platform, reducing the total cost of ownership

## Describe the Cloud Computing and the types of services offered

**Cloud Computing** (**CC**) is a way to use Information Technology infrastructures without the need to install specific HW related to the infrastructures being used.
IT infrastructure can be a virtual machine, a sw platform used to develop and run applications on several machines, or a sw application.
One of the main characteristics of Cloud Computing is the capability to provide resources that are abstracted, or virtualized Resources provided as a Service and with a self-service Interface. The main types of Cloud Computing are:

- **IaaS** (Infrastructure as a Service): Offers virtualized resources (computation, storage, and communication) on demand, running several choices of operating systems and a customized software stack.
- **PaaS** (Platform as a Service): A cloud platform offering an environment on which developers create and deploy applications; different programming languages and different DBMS. One of the most complete is **Google App Engine** (GAE).
- SaaS (Software as a Service): Offers the possibility to design and develop an **application** that exploits software modules accessed via Internet via a simple wire protocol. Most of the time it is **Web API**, so the protocol used is HTTP. Google offers a wide variety of API for desparate purposes, e.g. Google Maps API.




## Describe the main features of SaaS

**Software as a Service** is a cloud computing model in which a provider hosts application and makes them available to customers over the internet. Cloud services like SaaS offer flexible payments and high scalability, which gives customers the option to access more or fewer services or features OnDemand; they are accessible anytime and everywhere. Most SaaS applications can be run directly from a web browser without any downloads or installations required, although some require plugins. Because of the web delivery model, SaaS eliminates the need to install and run applications on individual computers. With SaaS, it's easy for enterprises to streamline their maintenance and support, because everything can be managed by vendors: applications, runtime, data, middleware, OSes, virtualization, servers, storage and networking. An example of SaaS is **Google Apps**.

## Describe the pros and cons of PaaS and give an example

**Platform as a Service** is cloud platform offering an environment on which developers create and deploy applications. It supports different programming languages (PHP, Java, Ruby) and databases. Users can decide the size of the virtual machine, where it is located, etc., and have a web-based console to create applications. They can use an IDE to develop the application and use an **SDK** or **CLI** to deploy it. The main advantages of PaaS are: it allows for higher-level programming with reduced complexity; the overall development of the application is more effective as it has built-in infrastructure; maintenance of the application is easier. Some disadvantages are: developers may not be able to use a full range of conventional tools (e.g. relational databases, with unrestricted joins); developers could be blocked in to a certain platform.

An example of PaaS is **Google App Engine**. It is a cloud service for running web applications on the Google data center. It has simple configuration, transparent scalability and supports different programming languages. Applications run inside a sandbox for security purposes.

## Describe the main features of IaaS

**Infrastructure as a Service** is a cloud computing service that enables on-demand provisioning of server running several choices of operating systems and a customized software stack. These services are backed by large-scale data centers composed of thousands of computers. All program instructions are executed inside a virtual machine: this provides isolation, security, reliability (software failures do not affect other VMs) and better performance (the execution of one VM does not affect the performance of another VM). Another interesting feature of this infrastructure is application mobility: this is done by encapsulating a guest OS within a VM and allowing it to be suspended, fully serialized, migrated to a different platform, and resumed immediately or preserved to be restored in a future time. There are two types of VM: **native** and **emulated**. **Native VM** is constituted by a logic machine that runs on the same physical machine. Logical and physical machines have the same ISA and instructions of the physical machine are in large part executed on the real CPU. In **emulated VM** physical machine and logical machine are different and they have different ISA, so there is hardware emulation and language level emulation (Java).

## Describe what a hypervisor is

A **hypervisor** or **virtual machine monitor** (**VMM**) is a component that creates and runs virtual machines. It mediates access to the physical hardware presenting to each guest operating system a virtual machine, which is a set of virtual platform interfaces. There are two types of hypervisor: **type-1** and **type-2** hypervisor. Type-1 hypervisors, also called native or bare metal, run directly on the host's hardware and guest operating systems run as processes on the host. These hypervisors are used to build Hardware Servers for cloud computing. Type-2 hypervisors, also called hosted, run on a conventional operating system just as other programs do. They are often used on clients. VirtualBox and VMWare are example of type-2 hypervisors.

# MISCELLANEOUS

## Ruby on Rails 5

Ruby on Rails, also known as Rails or RoR, is a framework for web applications built using the Ruby programming language and released under an open source license and is based on the MVC architecture (Model-View-Controller): the goal, in fact, is to provide developers with a platform that facilitates the development of web applications and beyond. Ruby on Rails encourages and facilitates the use of web standards such as JSON and XML for data transfer; HTML, CSS and JavaScript for rendering and their graphical display.
RoR was created with the aim of facilitating and speeding up the work of developers of web applications. It is no coincidence, therefore, that no of the key principles on which the development work in Ruby on Rails is based is the methodology DRY (acronym for Do not repeat yourself, do not repeat yourself). Thanks to this device, the programmer will have to define the elements only the first time they will be used: from then on it will be possible to continue to use them without having to recall the definition at any time.
Ruby on Rails is also a full-stack framework: the integration between the various levels and components of the program is so high that the developer will not have to set up links between them. In case an application refers to a data contained in a specific column of the database (also external), the programmer will not have to insert the call in the source code, but it will be the framework itself to retrieve it automatically. Rails, finally, adopts the Convention over configuration development philosophy: the developer of the application must intervene in the configuration of the software elements only when he decides, autonomously and voluntarily, to go beyond the specified conventions. To further facilitate the development work we find the MVC architecture, which provides programmers with a pattern or scheme thanks to which it will be possible to solve, in a rational and rather fast way, any problems related to the organization of a web application. The MVC scheme (model-view-controller) allows to independently manage the data associated with the logical functionality of the software from those related to their translation into graphical elements and from the components that control and adopt the same functionality. The name of the architecture is due to the fact that the paradigm described above provides for the presence of a Model (i.e. the methods that give access to the information that the application can modify), a View (or view, which is responsible for making visible the information necessary for the model and allow interaction between the data) and a Controller (which receives commands from the user and implements them, thus modifying both the model and the view).

## Heroku

Heroku is a cloud programming platform designed to help you build and deploy applications online. Founded in 2007, it is now one of the largest existing PaaS platforms, with hundreds of thousands of customers worldwide. It supports six programming languages (with which to develop apps for distribution on the web) and is based on the Debian operating system or, more recently, on Ubuntu (two Linux distributions). Heroku was one of the first platforms to appear on the web to allow its users to develop, distribute and manage apps directly online. Although it initially provided support only for projects compatible with the Rack web programming interface, over time Heroku has expanded its offering by adding hundreds of services and support for six programming languages (Java, Node.js, Scala, Clojure, Python and PHP). By integrating with Git, for example, users will be able to develop their app on an external platform and then import the project into their Heroku account, letting the PaaS platform do the rest of the "dirty work" to make it up and running.
The purpose of the platform is to provide users with the IT resources (both hardware and software) necessary to distribute and "run" web-based apps on various platforms (such as the Facebook). In this way, developers don't have to worry about having an adequate IT infrastructure to support their creature, but they can rent it from Heroku. The platform also takes care of the "dirty work", compiling and running the app automatically as soon as the source code is loaded into the operating environment. Programmers will then be able to concentrate on developing code that is as "clean" and bug-free as possible, leaving the rest of the tasks to the PaaS platform. At the base of Heroku's services are dyno, a container of functions capable of executing a single command that can be a web process, a worker(batch) process or another type of process whose syntax is allowed in the Procfile file. Depending on the needs and the computing power required, the developer can decide to rent one or more dyno, to ensure its users the best possible experience. To make this possible, Heroku offers a fully scalable and automated platform: the central system will manage the allocation of resources for each dyno and for each service offered through the app, relieving the programmer of tasks of this kind.

## Web-API

An acronym for Application Programming Interface, APIs are programming tools that the major software houses and industries in the IT world - such as Microsoft, Google and Facebook - make available to developers to facilitate their task in the creation of applications of various kinds.
APIs can take different "forms": they can be libraries of functions that allow the programmer to interact with a program or a software platform or simply a series of "calls" to parts of a program that a developer can use to shorten his work. APIs, therefore, are graphical interfaces that third-party developers and programmers can use to expand the functionality of programs, applications and platforms of various kinds (software and beyond). They are, therefore, the open interface through which to interact with otherwise inaccessible programs (or parts of them).
Using an API, a programmer can make two otherwise incompatible programs (or two platforms, or a program and a platform) interact. Therefore, by using programming "tricks", it is possible to extend the functionality of a program far beyond the real intentions of the developer or the software house that created it.

## Java Native Interface (JNI)

The Java Native Interface or JNI is a Java language framework that allows Java code to call up (or be called up by) so-called "native" code, i.e. specific to a specific operating system or, more generally, written in other programming languages, C, C++ and assembly.
The main application of the JNI is to call within Java programs portions of code that have intrinsically unwearable functionality (for example primitive operating system) and therefore cannot be implemented in pure Java. The interface is based on the definition of a set of classes connecting the two contexts, which have a Java interface, but which delegate to the native code the implementation of their methods.
A class can define an arbitrary number of methods implemented in native code. To do this, in the class source, the method must have the native keyword and must have a semicolon instead of the method body. A native method can be static and even final; it makes no sense, however, to define a native method as abstract. In general, native methods are kept private by the class that defines them, while public or protected methods (invoked by clients or subclasses) act as wrappers. In accordance with the principle of information encapsulation, this allows you to define a class interface that is completely independent of the use of native code; in this way, it will be easier in the future to modify the class behaviour while maintaining a native interface that is backwards compatible with the native libraries already implemented. The JNI is mainly used to allow the program access to functions defined in the host operating system libraries through system primitives. In reality, access is indirect, in the sense that the operating system functions are invoked by the functions that implement the native methods of the class that defines them, and which are in turn used by the Java code.
The use of JNI is also necessary when the implementation of a certain functionality in the program depends on the operating system in use at run-time and is not present in the standard Java libraries. The resulting program cannot be called "100%-Java", as it directly uses native code. In Sun's implementation of the standard Java platform libraries, there are many native methods. The implementation of these methods is only present in the virtual machine that will be used at run-time.

## Sandbox

Sandboxing allows you to isolate programs and applications (or parts of them) and test them in a "real operating environment" without running the risk of damaging the computer system. It is therefore ideal for testing the effects of a developing program or a virus of which little is yet known. Although it may appear as a futuristic function or reserved for an "elite" of programmers and geeks, the exact opposite is true: even if we don't notice it, we have to deal with sandboxes every day, in the daily use of the PC: Google Chrome and Internet Explorer, for example, are equipped with "sand fences" in which to run the web pages you visit to avoid that these can cause damage to the entire computer. The sandbox, as mentioned, is a "sterile" environment under strict surveillance where suspicious programs or programs containing viruses or malicious code can be run in the most protected and secure conditions possible. The sandboxes provide the program (or the portion of code) only with the permissions strictly necessary to perform the operations for which they were designed and only within the protected environment in which they are located, preventing access to other portions of the system that could be compromised by the execution of malicious code.

**Virtual Machine**: install a virtual machine inside your computer system and use it to test applications potentially dangerous to the stability of the system. A virtual machine is a software that simulates the operation of a hardware platform on which to install a new operating system without it, however, having full access to all the resources of the computer nor, much less, being able to access the software resources of the main operating system. This solution, therefore, allows you to have a second operating system in the "full" of its features, on which to perform all tests - even on viruses, perhaps - without affecting the stability of the main system.