

Módulo 9 – DevOps

Índice

Introducción

¿Qué es un Dev Ops?	3
Ciclo de vida Dev Ops	5
¿Qué es Docker?.....	8
Conceptos básicos de Docker	11

Seguridad Web

Introducción a la seguridad informática	16
Confidencialidad	16
Integridad.....	17
Disponibilidad	17
Ataques comunes a un sistema Web.....	18
Pilares de la ciberseguridad.....	20

Testing

Tipos de pruebas	21
Tipos de metodologías de desarrollo	23
Testing Angular	24
Testing Spring Boot	32

Deployment

Cómo subir tu proyecto Angular a la Nube	40
PREPARANDO EL ENTORNO	40
Subir tu sitio a Firebase Hosting	41
Subir el build a Firebase	43
Integración continua (CI/CD)	44
Subir tu proyecto Spring Boot a la nube	44
Crear una app Spring Boot	45
Desplegar aplicación Springboot en Fly.io usando Dockerfile	46

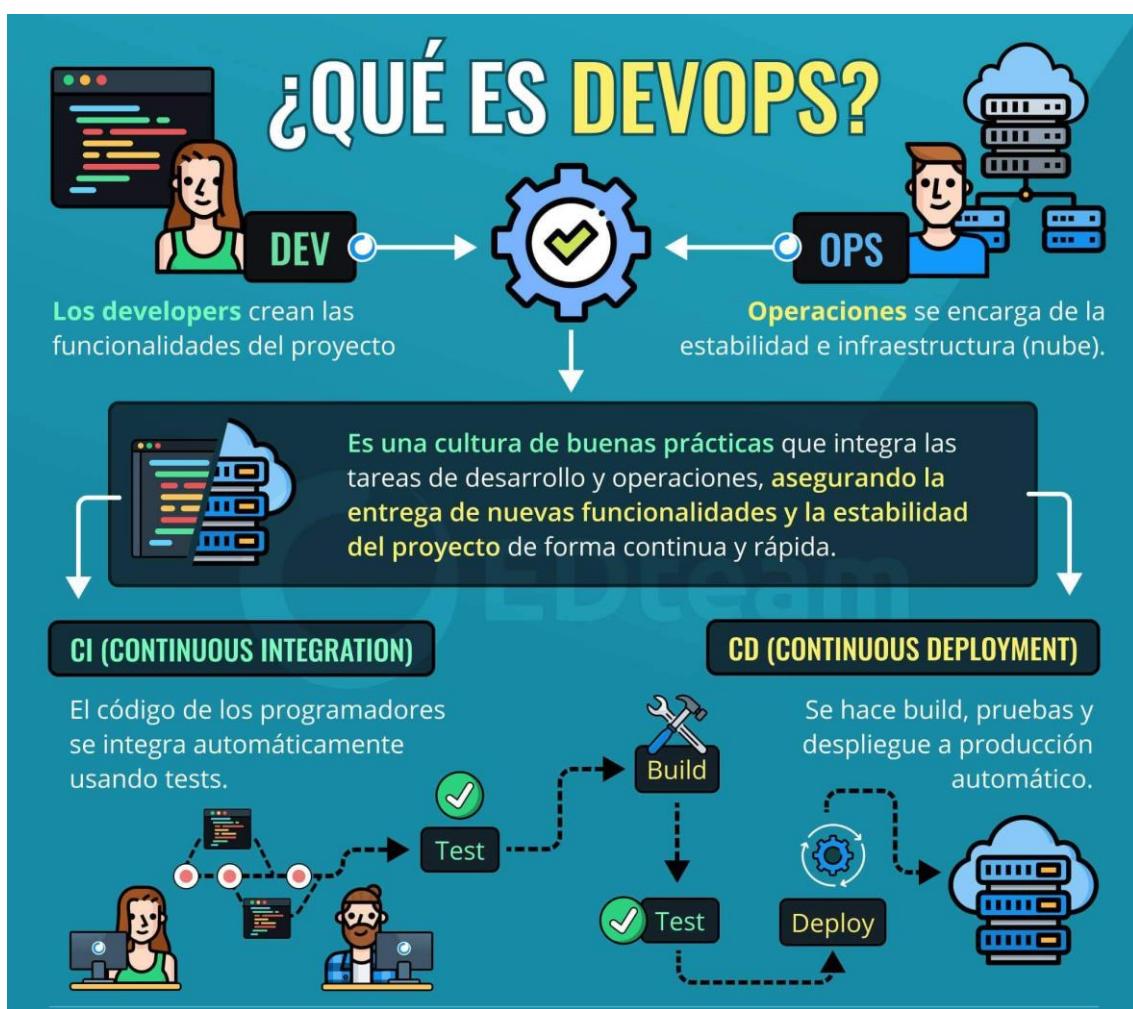
Desplegando: pasos a seguir	48
Cómo subir tu base de datos a clever cloud	54
Conectar base de datos con Spring Boot	59

Comunicación Efectiva

Introducción – Comunicación.....	61
Comunicación (Barreras, Filtros y Ruido)	64
Comunicación efectiva interpersonal - Comunicación Asertiva.....	66
El guión DEEC.....	68
Escucha Empática	70
Técnicas de escucha empática.....	72
Retroalimentación.....	74
Técnicas de retroalimentación	77
Niveles de Comunicación	78
Conclusión Comunicación Efectiva.....	79

¿Qué es un Dev Ops?

El término DevOps, es una combinación de los términos ingleses development (desarrollo) y operations (operaciones), designa la unión de personas, procesos y tecnología para ofrecer valor a los clientes de forma constante. En algunas empresas existe un perfil exclusivamente para realizar estas tareas, así como existe el puesto de desarrollador Front End, desarrollador BackEnd, también existe un DevOps.

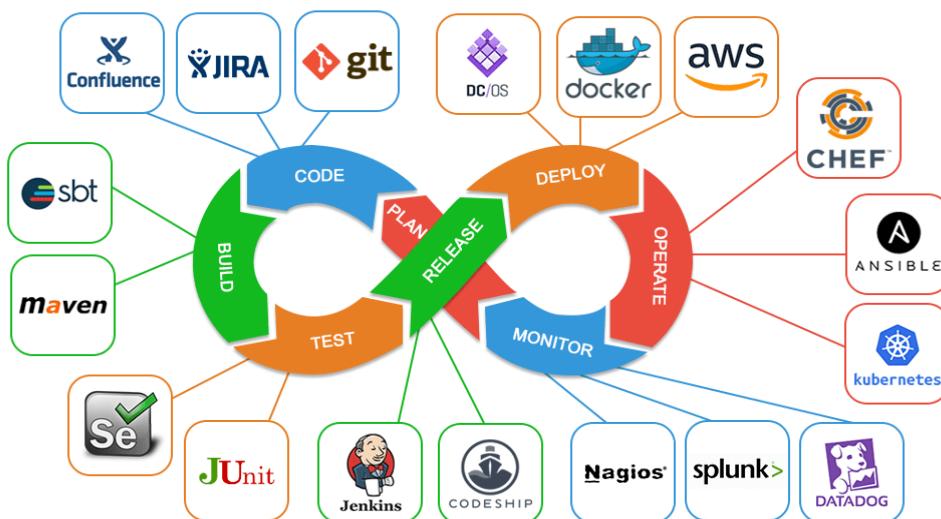


¿Qué significa DevOps para los equipos? DevOps permite que los roles que antes estaban aislados (desarrollo de software, operaciones de TI (backup, virtualización, configuraciones e implementaciones en sistemas operativos, etc), ingeniería de la calidad y seguridad) se coordinen y colaboren para producir productos mejores y más confiables. Al adoptar una cultura de DevOps junto con prácticas y herramientas de DevOps, los equipos adquieren la capacidad de responder mejor a las necesidades de los clientes, aumentar la confianza en las aplicaciones que crean y alcanzar los objetivos empresariales en menos tiempo.

Funciones de un DevOps

Breve descripción de las funciones de un DevOps:

- Deben comprender perfectamente la arquitectura distribuida.
- Saber de lenguajes de programación.
- Comunicación asertiva y manejo de conflictos.
- Saber de Configuraciones: Linux, base de datos, puertos utilizados, variables de entorno, servicios, seguridad, etc.
- Conocer servicios de infraestructuras en la nube: Git, Cloud de Amazon Web Services, Google Cloud Platform, Microsoft Azure, Heroku, Clevercloud o Firebase.
- Conocer de Metodologías Agiles y filosofía de Integración Continua / Entrega Continua (CI/CD)



Ventajas de DevOps

Los equipos que adoptan la cultura, las prácticas y las herramientas de DevOps mejoran el rendimiento y crean productos de más calidad en menos tiempo, lo que aumenta la satisfacción de los clientes. Esta mejora de la colaboración y la productividad es fundamental también para alcanzar objetivos de negocio como estos:



Reducción del tiempo
de comercialización



Adaptación al mercado
y a la competencia



Mantenimiento de la estabilidad
y la confiabilidad del sistema

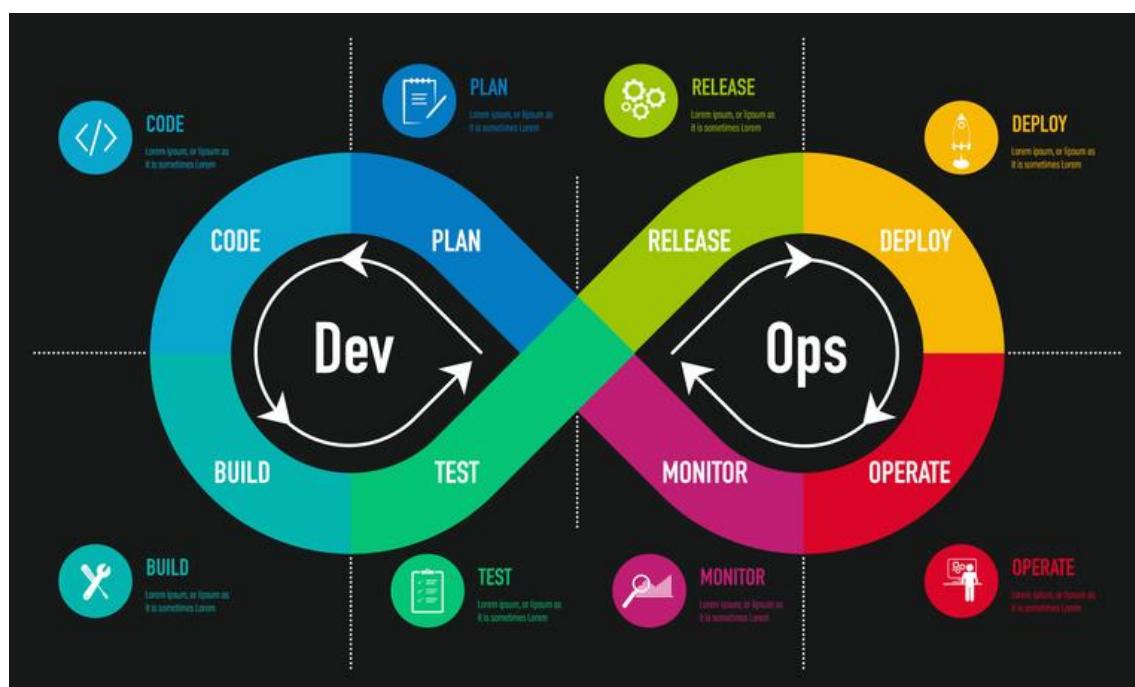


Mejora del tiempo medio
de recuperación

Ciclo de vida Dev Ops

DevOps influye en el ciclo de vida de las aplicaciones a lo largo de las fases de planeamiento, desarrollo, entrega y uso.

Cada fase depende de las demás y las fases no son específicas de un rol. En una auténtica cultura de DevOps, todos los roles están implicados de algún modo en todas las fases.



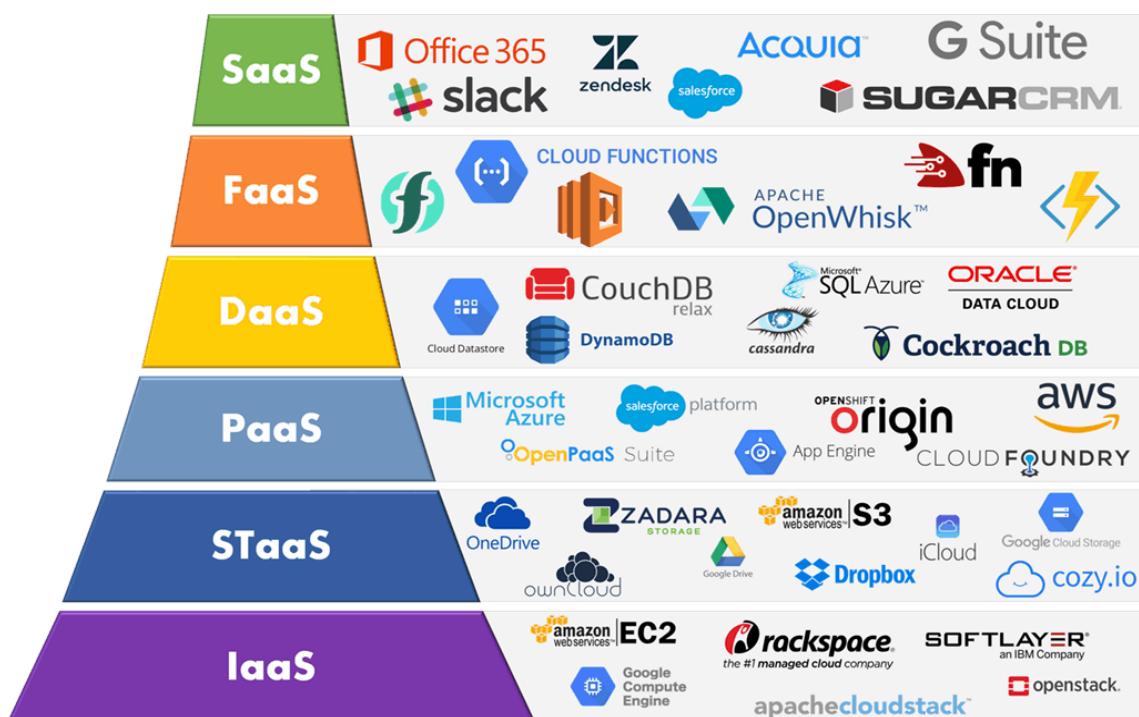
Herramientas que utiliza un DevOps

Los DevOps pueden utilizar distintas herramientas de software, te dejamos una lista para qué explores las qué sean de tu interés o quieras profundizar.

- **Git y Github:** En el sistema de control de versiones qué ya conocemos, pero en el área de DevOps se utiliza para guardar archivos de configuraciones, package, registry, docker files, etc.
- **Docker:** es una plataforma de contenerización, permite a los desarrolladores “empaquetar” aplicaciones en contenedores que son componentes estandarizados ejecutables que se combinan con librerías y dependencias que se requieren para correr código en cualquier ambiente.
- **NPM:** npm es el registro de software más grande del mundo. Los desarrolladores de código abierto de todos los continentes usan npm para compartir y tomar prestados paquetes, y muchas organizaciones también usan npm para administrar el desarrollo privado.
- **Jenkins:** es una herramienta open-source construida qué permite llevar a la práctica la filosofía de Entrega Continua / Integración Continua (CD/CI o Continuous Deployment / Continuous Integration en el idioma inglés) puedes explorar la documentación.
- **Selenium:** es una herramienta open-source que automatiza navegadores web, provee una sola interface que permite a los desarrolladores escribir scripts de prueba en varios lenguajes de programación tales como Java, Ruby, PHP, NodeJS, Python, Perl y C#, entre otros.
- **Kubernetes:** es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Kubernetes facilita la automatización y la configuración declarativa. Tiene un ecosistema grande y en rápido crecimiento.
- **Puppet:** es una herramienta open-source de administración cliente servidor por medio de la cual se hacen configuraciones e implementaciones masivas sobre muchos servers Linux y Windows al mismo tiempo mediante Código Como Infraestructura (Infrastructure as a Code).
- **Chef:** es una plataforma open-source usada para hacer mantenimiento y configuración de servers que también puede integrarse con plataformas de Nube como AWS, Microsoft Azure, Google Cloud Platform y otros.
- **Ansible:** Es una plataforma open-source por medio de la cual se automatizan procesos de aprovisionamiento, administración de configuraciones, despliegue de aplicaciones y otros procesos de TI.
- **Nagios:** es una antigua herramienta DevOps de código abierto lanzada en 2002. Nagios vigila tu infraestructura (servidores y aplicaciones) en segundo plano y te avisa si sospecha que hay un problema.
- **Trello:** es una herramienta de gestión de proyectos basada en tableros. Con Trello, puedes crear tableros con una serie de listas. Luego, puedes mover tarjetas entre estas listas. Trello es una herramienta colaborativa, lo que significa que varias personas pueden editar cada tablero.
- **Atlassian Jira:** Jira es una herramienta de gestión de proyectos diseñada específicamente para los equipos de TI que utilizan un enfoque Agile o

DevOps. Jira te permite hacer un seguimiento de cada tarea necesaria con tarjetas y tableros para el proyecto.

- **Plataformas como servicios (PaaS):** La plataforma como servicio o PaaS es un conjunto de servicios basados en la nube que permite a los desarrolladores y usuarios empresariales crear aplicaciones a una velocidad que las soluciones en las instalaciones propias no pueden alcanzar. Al tratarse de un servicio basado en la nube, no hay necesidad de preocuparse por la configuración y el mantenimiento de servidores, parches, actualizaciones y autenticaciones, entre muchas otras tareas. Cuando las organizaciones utilizan estos servicios, acceden a su infraestructura a través de Internet. Los más populares: Microsoft Azure, Nube de Google, Servicios web de Amazon (AWS), Heroku, Fly, Koyeb, entre otros.

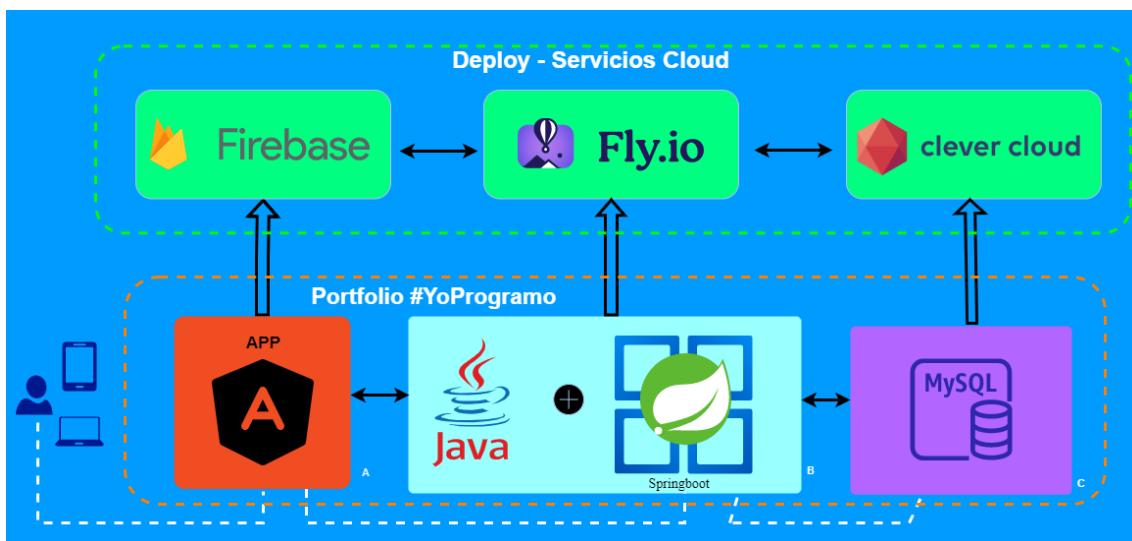


Empresas y servicios de computación en la nube (Fuente de la imagen: imelgrat.me)

¿Qué Herramientas en la nube utilizaremos en #YoProgramo?

Nosotros solamente vamos a focalizarnos en **GitHub**, **Docker** y diferentes **PaaS** para generar el despliegue de nuestro frontend y backend (Apis y Base de datos). Siendo las herramientas para utilizar las descriptas a continuación:

- Frontend Angular: [Firebase](#)
- Backend Spring boot: [Fly.io \(usando Dockerfile\)](#)
- Base de datos Mysql: [Clevercloud](#)



Para generar el despliegue o puesta en marcha de nuestro backend, es importante entender que usaremos los servicios como plataforma de Fly.io, que a su vez aloja nuestra aplicación en contenedores Docker. Si bien solo es necesario configurar un archivo (Dockerfile) en nuestro directorio de proyecto springboot es importante que conozcas que es y cómo trabaja Docker.

¿Qué es Docker?

Un contenedor Docker es un formato que empaqueta todo el código y las dependencias de una aplicación en un formato estándar que permite su ejecución rápida y fiable en entornos informáticos. Un contenedor de Docker es un conocido contenedor ejecutable, independiente, ligero que integra todo lo necesario para ejecutar una aplicación, incluidas bibliotecas, herramientas del sistema, código y tiempo de ejecución. Docker es también una plataforma de software que permite a los desarrolladores crear, probar e implementar aplicaciones en contenedores de forma rápida.

Los servicios de contenedores o Containers as a Service (CaaS) son servicios gestionados en la nube que administran el ciclo de vida de los contenedores. Los servicios de contenedores ayudan a orquestar (iniciar, detener, ampliar) el tiempo de ejecución de los contenedores. Con los servicios de contenedor, puede simplificar, automatizar y acelerar el desarrollo de sus aplicaciones y el ciclo de vida de su implementación.

Los Docker y servicios de contenedores se han adoptado rápidamente y han tenido un gran éxito en los últimos años. Docker ha evolucionado de una tecnología inicial de código abierto casi desconocida y bastante técnica en 2013 a un entorno de ejecución estandarizado.

Terminologías de Docker

Docker:

Plataforma de contenedor de software diseñada para desarrollar, enviar y ejecutar aplicaciones aprovechando la tecnología de los contenedores. Docker se presenta en dos versiones: edición empresarial y edición de comunidad.

Contenedor:

A diferencia de una máquina virtual que proporciona virtualización de hardware, un contenedor proporciona virtualización ligera a nivel de sistema operativo mediante la abstracción del "espacio del usuario". Los contenedores comparten el núcleo del sistema host con otros contenedores. Un contenedor, que se ejecuta en el sistema operativo host, es una unidad de software estándar que empaqueta código y todas sus dependencias, para que las aplicaciones se puedan ejecutar de forma rápida y fiable de un entorno a otro. Los contenedores no son persistentes y se activan desde imágenes.

Motor de Docker:

El software de host de código abierto que crea y ejecuta los contenedores. Los motores de Docker funcionan como la aplicación del servidor del cliente que admite contenedores en varios servidores Windows y sistemas operativos Linux, CentOS, Debian, Fedora y Ubuntu.

Imágenes de Docker:

Colección de software que se ejecutará como un contenedor que incluye un conjunto de instrucciones para crear un contenedor que se pueda ejecutar en la plataforma Docker. Las imágenes no son modificables, de modo que para realizar cambios en una imagen es preciso crear otra nueva.

Docker Registry:

Lugar para almacenar y descargar imágenes. Registry es una aplicación de servidor escalable y sin estado que almacena y distribuye imágenes de Docker.

¿Quién usa Docker?

Docker es un marco abierto de desarrollo de aplicaciones que se ha diseñado para ayudar a DevOps y a los desarrolladores. Con Docker, los desarrolladores pueden crear, empaquetar, enviar y ejecutar aplicaciones fácilmente en forma de contenedores ligeros, portátiles y autosuficientes que pueden utilizarse en prácticamente cualquier lugar. Los contenedores permiten a los desarrolladores empaquetar una aplicación con todas sus dependencias y desplegarla como una sola unidad. Al proporcionarles contenedores de aplicaciones precompilados y autosuficientes, los desarrolladores pueden centrarse en el código de la aplicación y utilizarlo sin preocuparse por el sistema operativo subyacente ni por el sistema de implementación.

Además, los desarrolladores pueden aprovechar miles de aplicaciones de contenedor de código abierto ya diseñadas, que se pueden ejecutar dentro de un contenedor Docker. Para los equipos de DevOps, Docker permite una integración continua y el desarrollo de cadenas de herramientas, al tiempo que reduce las limitaciones y la complejidad necesarias dentro de la arquitectura de su sistema para implementar y administrar las aplicaciones. Con la introducción de los servicios en la nube de orquestación de contenedores, cualquier desarrollador puede desarrollar localmente aplicaciones en contenedores en su entorno de desarrollo y mover y ejecutar posteriormente dichas aplicaciones de contenedores en servicios de nube, como los servicios Kubernetes administrados.

Docker y desarrolladores

Los contenedores pueden ser empaquetados por cualquier tipo de desarrollador. En el sector del software, a menudo los desarrolladores se dividen por especialización, esto es, frontend, el backend, o cualquier punto entre ambos extremos. Aunque en la mayoría de los casos te encontrarás con que son los desarrolladores de backend quienes se encargan de empaquetar los contenedores, cualquiera que esté familiarizado con los conceptos básicos de CaaS puede tener éxito en esta área concreta del ciclo de vida de desarrollo de software.

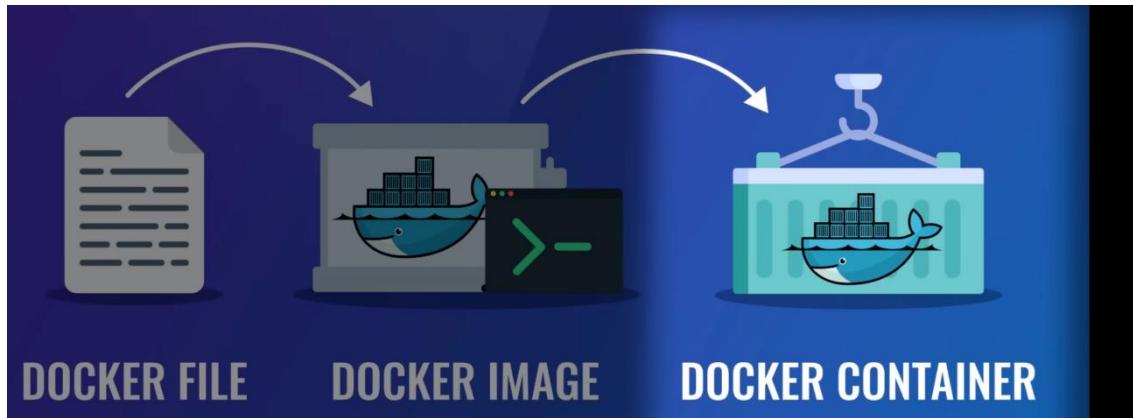
Comparación entre Docker y Kubernetes

Los contenedores Linux existen desde 2008, pero no se conocían bien hasta que en 2013 surgieron los contenedores Docker. Con la aparición de los contenedores Docker, aumentó vertiginosamente el interés por desarrollar e implementar aplicaciones en contenedores. A medida que crecía la cantidad de aplicaciones en contenedores hasta abarcar cientos de contenedores implementados en múltiples servidores, aumentaba la complejidad de su gestión. ¿Cómo se coordinan, amplían, gestionan y programan cientos de contenedores? Aquí es donde Kubernetes le puede ayudar. Kubernetes es un sistema de orquestación de código abierto que le permite ejecutar sus cargas de trabajo y contenedores Docker. Le ayuda a gestionar las complejidades operativas que surgen cuando se decide ampliar varios contenedores desplegados en varios servidores. El motor Kubernetes organiza automáticamente el ciclo de vida del contenedor y distribuye los contenedores de

aplicaciones en la infraestructura de alojamiento. Kubernetes puede ampliar o reducir rápidamente los recursos, en función de la demanda. Suministra, programa, elimina y supervisa constantemente el estado de los contenedores.

Conceptos básicos de Docker

Los conceptos centrales de Docker son los dockerfile, las imágenes y los contenedores.



Dockerfile

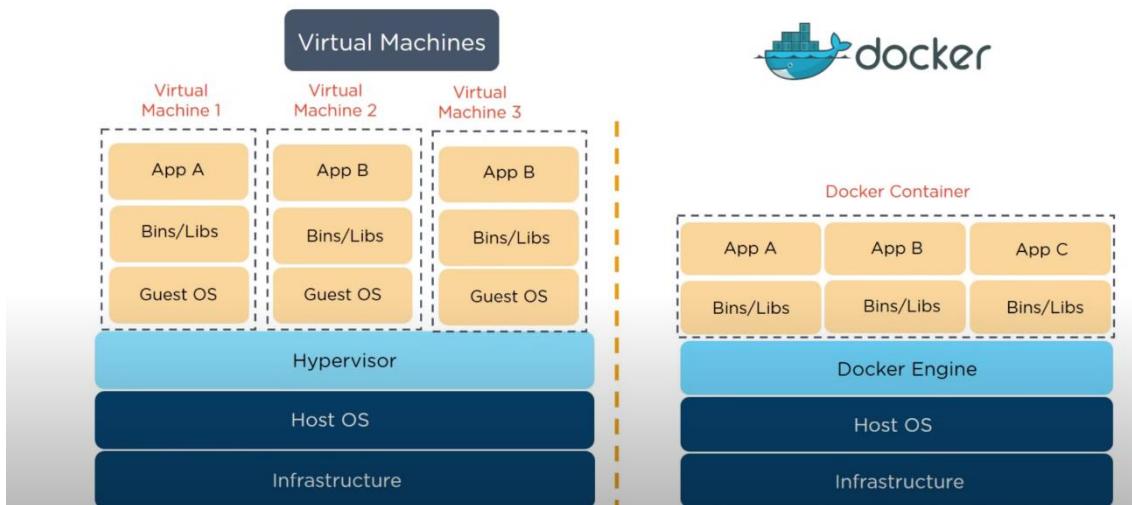
Docker puede crear imágenes automáticamente leyendo las instrucciones de un archivo Dockerfile. Un **Dockerfile** es un documento de texto que contiene todos los comandos que un usuario podría llamar en la línea de comando para ensamblar una imagen.

Imagen

Una **imagen** de Docker contiene todo lo que necesitas para ejecutar tu software: el código, un tiempo de ejecución (por ejemplo, Java Virtual Machine (JVM), controladores, herramientas, scripts, bibliotecas, implementaciones, etc.

Contenedor

Un **contenedor** de Docker es una instancia en ejecución de una imagen de Docker. Sin embargo, a diferencia de la virtualización tradicional con un hipervisor de tipo 1 o tipo 2, un contenedor de Docker se ejecuta en el núcleo del sistema operativo host. Dentro de una imagen de Docker no existe un sistema operativo independiente, como se ilustra en la siguiente Figura:



Comparación entre aislamiento y virtualización

Cada contenedor de Docker tiene su propio sistema de archivos, su propia pila de red (y, por lo tanto, su propia dirección IP), su espacio de proceso propio y limitaciones de recursos definidas para la CPU y la memoria. El contenedor de Docker se inicia de forma instantánea, sin tener que arrancar un sistema operativo. Docker se basa en el aislamiento —es decir, en separar los recursos de un sistema operativo host—, mientras que la virtualización suministra un sistema operativo invitado sobre el sistema operativo host.

Sistema de archivos incrementales

Deployment Image

WLS Domain Image

WebLogic Image

JDK Image

Linux Base Image

El sistema de archivos de una imagen de Docker tiene varias capas, con semántica de copia en escritura. Esto permite heredar y reutilizar, ahorra recursos en el disco y permite la descarga incremental de imágenes.

Como se ilustra en el cuadro anterior, una imagen de Docker con una implementación Weblogic podría basarse en una imagen con un dominio WebLogic Server, que podría basarse en una imagen Weblogic, que, a su vez, se basa en una imagen Java Development Kit (JDK) y que, por su parte, está basada en una imagen básica de Linux.

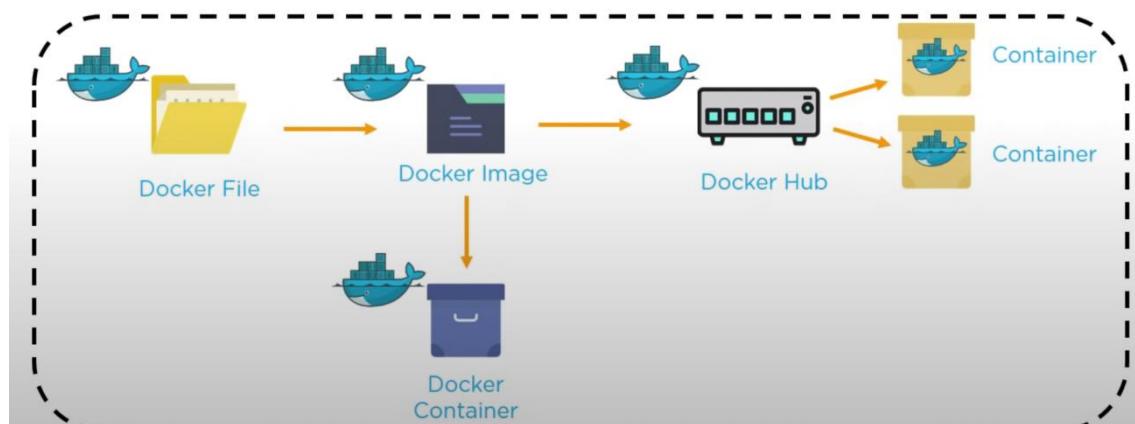
Docker Registry

Si bien las imágenes de Docker son fáciles de crear y los desarrolladores adoran su simplicidad y portabilidad, se percataron con rapidez de que administrar miles de imágenes Docker resulta todo un desafío. Docker Registry se enfrenta a este reto. Docker Registry es una forma habitual de almacenar y distribuir imágenes de Docker. Se trata de un repositorio basado en código abierto con la cesión de licencia de Apache.

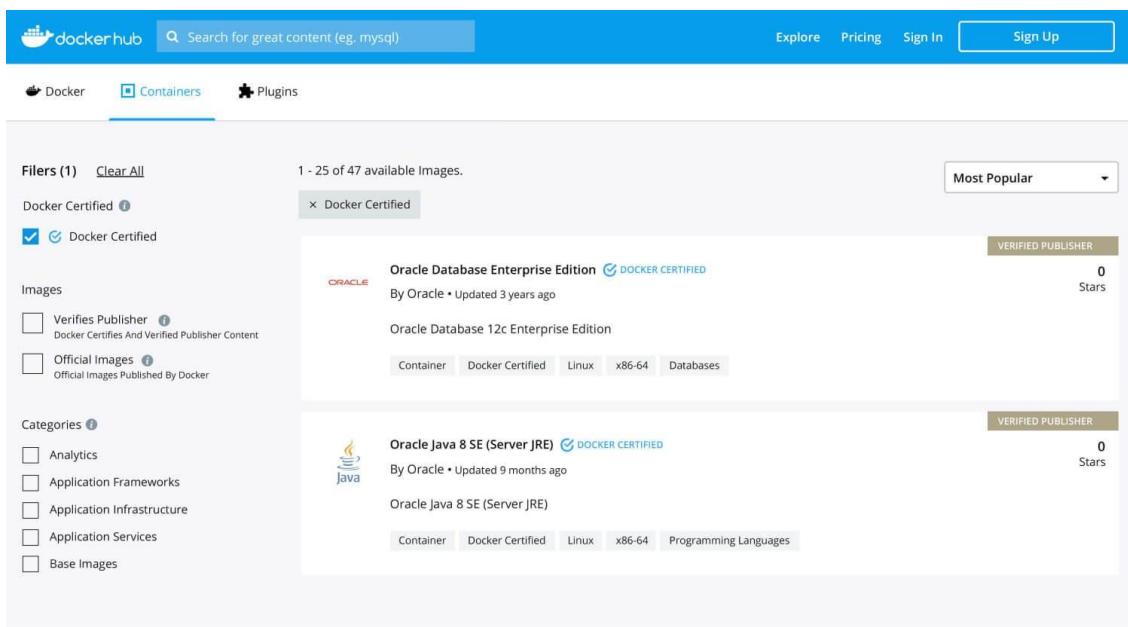
Docker Registry también ayuda a mejorar el control de acceso y la seguridad de las imágenes de Docker almacenadas en su repositorio. Gestiona la distribución de imágenes y también se puede integrar con los flujos de trabajo de desarrollo de aplicaciones. Los desarrolladores pueden configurar su propio Docker Registry o utilizar un servicio de Docker Registry alojado como Docker Hub, Oracle Container Registry, Azure Container Registry, Github Registry, etc.

Docker Hub

Docker Hub es un Docker Registry alojado y administrado por Docker. Docker Hub tiene más de 100 000 imágenes de contenedores de proveedores de software, proyectos de código abierto y de la comunidad. Docker Hub contiene software y aplicaciones de repositorios oficiales como NGINX, Logstash, Apache HTTP, Grafana, MySQL, Ubuntu y Linux.



Al iniciar un contenedor, Docker extraerá automáticamente de forma predeterminada la imagen correspondiente del Docker Hub público, si no está disponible localmente. Además, también puede crear sus propias imágenes y enviarlas a un repositorio público o privado de Docker Hub.



Docker como tiempo de ejecución de microservicios

En la actualidad, la idea de cortar aplicaciones monolíticas en fragmentos más pequeños de microservicios despierta mucho interés entre los desarrolladores de software.

Los microservicios se implementan como un proceso independiente, utilizan protocolos ligeros para comunicarse entre sí y cada servicio es propietario de sus datos. Dado que los microservicios siguen una estrategia de gobernanza descentralizada, requieren una cantidad bastante alta de automatización de la infraestructura, pruebas automatizadas, canalizaciones de CD totalmente automatizadas y equipos de DevOps ágiles y capacitados.

Todavía se debate bastante sobre este estilo arquitectónico; sin embargo, sería ingenuo suponer que una aplicación descompuesta en microservicios se puede operar como un simple conjunto de procesos. Por nombrar solo algunos requisitos, un microservicio debe ser independiente del host y estar aislado a nivel de sistema operativo. Debe ejecutarse dentro de sus límites de recursos, debe ampliarse y reducirse, reiniciarse si falla y detectarse y conectarse a otros microservicios a través de una capa de red definida por software.

Por lo tanto, ejecutar un microservicio en un contenedor de Docker es un excelente punto de partida para lograr la mayoría de estos objetivos.

Docker: dos dimensiones clave

Docker cambia la forma en que creamos, enviamos y ejecutamos software en dos dimensiones diferentes:

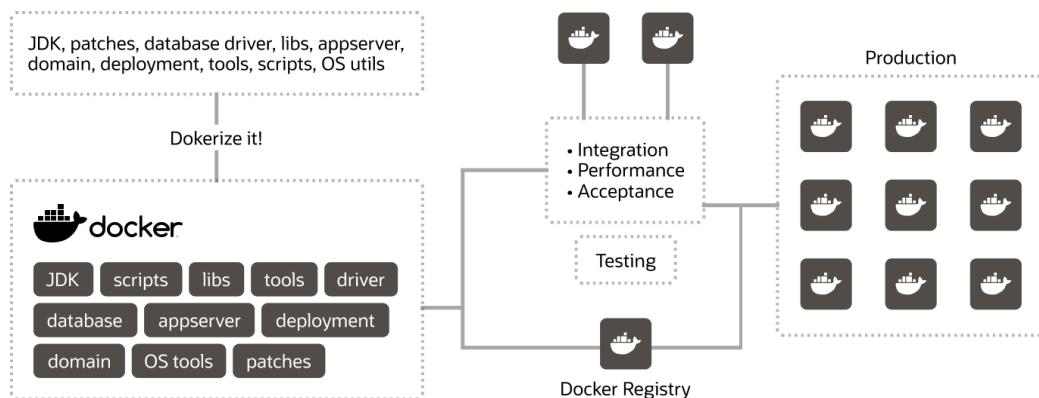
- Mejora el proceso para obtener aplicaciones de manera fiable desde el desarrollo hasta la producción.
- Proporciona un formato de imagen estándar para pasar del entorno local a la nube.

Ambas dimensiones se explican con más detalle en los siguientes párrafos.

Imagen de Docker Image: del desarrollo a la producción

Crear una imagen de Docker con todas sus dependencias resuelve el argumento de "pero es que funcionó en mi máquina de desarrollo". La idea clave es que una imagen de Docker se crea automáticamente mediante una canalización de compilación a partir de un repositorio de código fuente como Git, y se prueba en un primer momento en un entorno de desarrollo. Esta imagen inmutable se almacenará en un Docker Registry.

Como se muestra a continuación, la misma imagen se utilizará para más pruebas de carga, pruebas de integración, pruebas de aceptación, etc. En todos los entornos se utilizará la misma imagen. Las diferencias, pequeñas pero necesarias, específicas de cada entorno, como una URL JDBC para una base de datos de producción, se pueden introducir en el contenedor a modo de variables de entorno o archivos.



Docker Cloud

Docker ha cambiado la adopción de nubes públicas. Por un lado, en el caso de la imagen de Docker, por primera vez en la historia, existe un formato de paquete común que se puede ejecutar tanto en entornos locales como en todos los principales proveedores de nube.

Por otra parte, dado que los contenedores Docker se ejecutan en todas las principales nubes públicas, contribuyen de manera significativa a superar el prejuicio forjado durante años en contra de las nubes públicas: la dependencia del proveedor.

Todos los principales proveedores de la nube ofrecen ahora Docker como PaaS.

Introducción a la seguridad informática

En la actualidad, la seguridad informática ha tomado un papel crucial en el éxito de cualquier software. La misma tiene como objetivo proteger la información a través de medidas preventivas y reactivas. Para ello todo desarrollo de un sistema informático debe ser guiado por un plan de seguridad informática.

Para elaborar ese plan, debemos comprender que la seguridad informática se basa en los siguientes 3 pilares:

1. Confidencialidad de la información.
2. Integridad de la información.
3. Disponibilidad de la información.

Confidencialidad

La confidencialidad hace referencia a que la información sea accesible por aquellas personas autorizadas a la misma. Una de las principales técnicas que se utiliza para lograr esto es la **criptografía**. Aunque cabe aclarar que, en general, es necesaria, pero puede no ser suficiente, es decir, uno puede implementar y aplicar criptografía en un sistema web y el mismo sigue generando información no confidencial. Más adelante en el texto veremos porqué.

Bien, pero ¿qué es la criptografía? Se podría definir de manera muy simple como el arte de ocultar mensajes. Veamos un ejemplo. Supongamos el siguiente mensaje:

el mejor modulo del programa clip es el de ciberseguridad

Entonces, podríamos cifrarlo y lograr el siguiente mensaje cifrado:

hocphmrucprgxorcghocsurjudpdcfolschvchocghcflehuvhjxulgdg

El algoritmo utilizado anteriormente se denomina algoritmo de César, en honor al emperador romano, que fue su inventor. La idea detrás de este algoritmo es bastante simple. Cada letra del mensaje que queremos ocultar (también llamado mensaje en claro) se reemplaza por una letra de N posiciones más adelante en el alfabeto, por ejemplo, si fijamos a N=3 y se necesita cifrar a la letra e, la letra que la reemplazará será la h, ya que es la que le sigue en 3 posiciones más adelante en el alfabeto, es decir, está la f, la g y luego la h.

e	f	g	h
0	1	2	3

Recordemos que apenas hemos introducido al atributo de la confidencialidad, inmediatamente después hemos hablado de la criptografía pero no necesariamente cumplir correctamente con algoritmos criptográficos bien implementados estamos asegurando confidencialidad.

La criptografía no asegura confidencialidad en un sitio web

Es muy probable que a la mayoría que esté leyendo esto, le parezca evidente la afirmación expuesta en el título de este párrafo. Intentemos entenderlo con más profundidad. Supongamos que la plataforma web que estamos desarrollando cuenta con un backend con APIs REST. Éstas probablemente necesiten algún mecanismo de autenticación y autorización. El primero hace referencia a determinar si el usuario que quiere acceder puede hacerlo, el segundo hace referencia a si el usuario que ya está autenticado (es decir, el sistema web ya sabe quién es) puede o no acceder a este recurso. Ahora bien, supongamos que existe una falla en el mecanismo de autenticación o autorización, entonces efectivamente estaríamos ante una probable fuga en la confidencialidad

Integridad

La integridad es la propiedad de la información que asegura que la misma no haya sido alterada por usuarios malintencionados y que el usuario que debe acceder a esa información, lo hace de manera completa y correcta.

Particularmente lo que nos interesa respecto a la integridad de la información es poder determinar si la misma ha sido alterada. Se han diseñado algoritmos que permiten generar una “marca” de un determinado contenido. Por ejemplo, dado un archivo X, se puede obtener una marca generalmente conocida como hash del archivo y, entonces, si el archivo X es modificado en al menos un bit, ese hash cambiará rotundamente, por lo que podemos asegurar que no se trata del mismo archivo.

Existen varios tipos de algoritmos generadores de hashes (a veces llamados funciones hash). Los dos más conocidos son MD5 y SHA. Particularmente para la segunda existen algunas variantes, por lo que se la suele llamar familia de funciones SHA

Disponibilidad

La disponibilidad es un atributo de la información que implica que la misma esté a disposición cuando las personas autorizadas lo requieran. Efectivamente es difícil obtener un nivel de disponibilidad del 100% en un sistema web. Esto depende tanto de la aplicación como así también del entorno en el cual esté montada. En este entorno, existen muchos componentes que pueden fallar tanto por un error sin intención como así también a causa de algún tercero que lo provocó. En cualquier caso, es necesario definir un plan de reacción para poder afrontar a estos eventos inesperados.

Para definir un plan realista, es necesario comprender correctamente el entorno del sistema web. Esto es, comprender:

1. Arquitectura del software.
2. Arquitectura de infraestructura.
3. Servicios externos.

Arquitectura del software

Es necesario identificar servicios web, páginas, controladores, modelos, acceso a base de datos, transferencia de archivos al cliente.

Dependiendo de la arquitectura del sistema web, se deberán plantear acciones para actuar ante un evento no deseado. Los puntos claves de la disponibilidad en un sistema web son: acceso a disco (normalmente a una base de datos), el uso de la red en las peticiones (tanto en recibir como en enviar información).

Arquitectura de infraestructura

En este punto es necesario identificar servidor web, servidores proxy, firewall, herramientas para monitoreo,

Básicamente un sistema web puede ser desplegado en una infraestructura de hardware propietaria, en donde, es muy probable que se requieran configuraciones para fortalecer esta infraestructura. Otra alternativa es usar servicios de hosting, o bien, servicios en la nube. Tener en cuenta que la diferencia entre servicios de hosting y servicios en la nube suelen ser sutiles. Tal vez la más clara sea que un servicio en la nube se paga por lo que se usa.

Servicios externos

Si el sistema web se comunica con otros sistemas, a través de por ejemplo una petición HTTP, es importante actuar ante aquellos casos en que este servicio tercero no funcione como esperamos.

Desarrollo seguro

Para terminar con la introducción, se pretende introducir el concepto de desarrollo seguro. Esto significa empezar el desarrollo teniendo en cuenta la seguridad y analizarla en cada etapa del desarrollo. Esta manera de pensar permite adelantarnos ante futuros problemas y ser más eficientes en el manejo de los recursos del proyecto. Independientemente de la metodología de desarrollo que se utilice se pueden aplicar actividades que contribuyan al desarrollar un producto seguro.

Ataques comunes a un sistema Web

Cuando uno piensa en ataques comunes a un sistema web, es imposible no hacer referencia al proyecto OWASP [<https://owasp.org/>]. OWASP es un proyecto abierto de seguridad en aplicaciones web, el cual, reúne información valiosa tanto para los

desarrolladores como para los analistas de seguridad. Existen muchos diferentes tipos de ataques que se pueden generar hacia un sitio web.

Ataques de inyección SQL

La inyección SQL permite a un atacante modificar los datos total o parcialmente de la base de datos de la aplicación web, evidentemente debe ser un motor de base de datos SQL. Para poder injectar sentencias SQL a una base de datos, la aplicación web debe ser vulnerable, es decir, al menos un parámetro de alguna request HTTP no es validada correctamente. En la jerga de la seguridad informática, se suele llamar a esta validación como *sanitización* que proviene del inglés *sanitize*. Tener en cuenta que este parámetro puede ser que se transmita a través de la URL o del cuerpo de la petición HTTP.

Por ejemplo, una petición HTTP en la cual se envía un parámetro a través de la URL es:

GET /src/contenedor.php?id=9&tipo=algoritmo HTTP/1.1

Host: seclabsis.frc.utn.edu.ar

Como se puede observar, en esta petición de tipo GET se envían 2 parámetros: id y tipo. Ahora, supongamos que en el backend se realiza la consulta a la base de datos siguiente:

```
String user = getUrlParam;  
  
String password = "";  
  
String consulta = "SELECT id, nombre FROM usuarios WHERE nombre='" + user  
+ "' AND password=''" + password + "'";
```

Cuando un parámetro llega a la base de datos.

Sanitizar las entradas antes de ejecutar las consultas a la base de datos. Se puede hacer a través de sentencias preparadas.

```
String consulta = "select * from Persona where nombre = ? ";  
  
Connection conexion =  
DriverManager.getConnection("jdbc:mysql://localhost/prueba", "root", "root");  
  
PreparedStatement ps = conexion.prepareStatement(consulta);  
  
ps.setString(1, nombre);
```

```
ResultSet rs = sentencia.executeQuery();
```

ORM

Con la llegada de ORM, la vulnerabilidad de SQL Injection desaparece. Debido a que este tipo de frameworks realizan el trabajo de saneamiento de entradas.
Ataques de inyección de código JavaScript

Ataques IDOR

Introducción a identificadores

En un sistema informático es crucial identificar objetos, recursos, filas, datos o lo que sea. En general, le llamamos ID a este identificador. En un sistema web estos IDs suelen visualizarse en el front-end, ya sea a través de la URL, en las peticiones HTTP, en cookies, etc. Esto brinda a un atacante información y dependiendo de varios factores esto podría ser información sensible.

Supongamos la tabla MovimientosCuentaBancaria que tenga las siguientes columnas:

Nombre	Tipo
id	INT
cbu_usuario_remitente	VARCHAR(22)
cbu_usuario_destinatario	VARCHAR(22)
monto	DECIMAL(12,3)
fecha	DATE

Si el atacante genera un movimiento en su cuenta enviándole a alguien un dinero y luego consulta ese movimiento:

https://ejemplo.com/get_movimiento_bancario?id=15602

Se puede sospechar que el siguiente es 15602 y el anterior 15601. De manera muy simple se podría recorrer los 15602 movimientos. Este ataque se llamar IDOR (Insecure Direct Object Reference).

Pilares de la ciberseguridad

Confidencialidad					
Cosas imposibles	Herramientas	Ataque	Victima	Defensa	Condiciones
No se pueden cerrar los puertos (cortaría el servicio). No se pueden bloquear las ip (si es dinámico se bloquearía el usado)	Ethercapt Wireshark Metasploit FrameWork	Captura de paquetes de red, por ejemplo, para obtener usuarios y	Cualquier usuario, conversación entre emisor y receptor	Encriptar la comunicación, bloqueo por mac, control de acceso a la red (Radius)	IP Local IP Remota Placa de red encendida

		contraseñas (Sniffing)		
Autentificación				
Herramientas	Ataque	Victima	Defensa	Condiciones
Medusa (Kali) usando un diccionario Aircrack-Ng	Autentificación por fuerza Bruta Wi-Fi WPA/WPA2 Denegación de Servicio (como herram.)	MSQL Server Cualquier usuario en red Servicio SSH	Puertos Reforzar Pass (con múltiples caracteres)	Que la pass sea débil (aumenta la vulnerabilidad). Placa de red encendida. Obtención del ip de la víctima y puerto (NMAP)
Disponibilidad				
Herramientas	Ataque	Victima	Defensa	Condiciones
AirCrack	DOS (denegación de servicio)	Cualquier servidor	Firewall	Placa de red encendida
Integridad				
Herramientas	Ataque	Victima	Defensa	Condiciones
inyección de Código por página web inyección de troyano por USB	Exposure Data SQL Injection Pentesting Cross- Site Scripting	Cualquier servidor Base de datos Aplicación Web	Antivirus Navegador actualizado Conexión de clientes confiables	Placa de red encendida Dirección y acceso web Navegador desactualizado

Testing

Tipos de pruebas:

Test unitarios

Este tipo de testing consiste en probar de forma individual las funciones y/o métodos (de las clases, componentes y/o módulos que son usados por nuestro software).

Debido a lo específicas que son, generalmente son las pruebas automatizadas de menor coste, y pueden ejecutarse rápidamente por un servidor de *continuous integration* (integración continua)

Test de integración

Las pruebas de integración verifican que los diferentes módulos y/o servicios usados por nuestra aplicación funcionen en armonía cuando trabajan en conjunto.

Los tests de integración nos van a permitir testear funcionalidades complejas dentro de una aplicación, como por ejemplo el login. Podremos crear los tests necesarios para evaluar si nuestra app está realizando la función de login correctamente, así como si rechaza también de forma correcta usuarios no autorizados. Este es un buen ejemplo de un test de integración en el que intervienen varios elementos de nuestra app.

Test funcional

Estas pruebas verifican la salida (resultado) de una acción, sin prestar atención a los estados intermedios del sistema mientras se lleva a cabo la ejecución.

Test End-to-End

Las pruebas de punta a punta replican el comportamiento de los usuarios con el software, en un entorno de aplicación completo. Estas pruebas verifican que los flujos que sigue un usuario trabajen como se espera, y pueden ser tan simples como:

- Cargar una página web
- Iniciar sesión

O mucho más complejas como:

- Verificando notificaciones vía email
- pagos en línea

Test de regresión

Las pruebas de regresión verifican un conjunto de escenarios que funcionaron correctamente en el pasado. Una falla en una prueba de regresión significa que una nueva funcionalidad ha afectado otra funcionalidad que era correcta en el pasado, causando una "regresión".

Test de aceptación

Las pruebas de aceptación son pruebas formales, ejecutadas para verificar si un sistema satisface sus requerimientos de negocio. Estas pruebas requieren que el software se encuentre en funcionamiento, y se centran en replicar el comportamiento de los usuarios, a fin de rechazar cambios si no se cumplen los objetivos. Estos objetivos pueden ir más allá de obtener una respuesta específica, y medir el rendimiento del sistema.

Test de rendimiento

Las pruebas de rendimiento verifican cómo responde el sistema cuando éste se encuentra bajo una alta carga. Estos tests son no-funcionales, y pueden tener diversas formas para entender:

- Fiabilidad
- Estabilidad
- Disponibilidad de la plataforma.

Tipos de metodologías de desarrollo

Test driven development:

TDD son las siglas de *Test Driven Development*. Es un proceso de desarrollo que consiste en codificar pruebas, desarrollar y re-factorizar de forma continua el código construido.

La idea principal de esta metodología es realizar de forma inicial las pruebas unitarias para el código que tenemos que implementar. Es decir, primero codificamos la prueba y, posteriormente, se desarrolla la lógica de negocio.

Los pasos para definir las pruebas son:

1. Tener bien definidos los requisitos de la función a realizar.
2. Tener criterios de aceptación, contemplando todos los casos posibles.
3. Pensar en cómo vamos a diseñar la prueba.
4. Pensar qué queremos probar.
5. Hacer todos los test que sean necesarios para definir correctamente si un componente es correcto.

Principios:

- Principio de responsabilidad simple.
- Principio de abierto/cerrado (OCP).
- Principio de sustitución de Liskov (LSP).
- Principio de segregación de Interfaces (ISP).
- Principio de inversión de dependencia (DIP).

Ciclo de Vida:

1. Elegir un requisito.
2. Codificar la prueba.
3. Tener claro qué vamos a probar y qué filosofía vamos a llevar a cabo para que lo que probemos sea significativo y demuestre que nuestro desarrollo es correcto.
4. Verificar que la prueba falla.
5. Codificar la implementación.

6. Ejecutar las pruebas automatizadas.
7. Refactor.
8. Actualización de la lista de requisitos.

Ventajas:

1. Mayor calidad en el código desarrollado.
2. Diseño orientado a las necesidades.
3. Simplicidad, nos enfocamos en el requisito concreto.
4. Menor redundancia.
5. Mayor productividad (menor tiempo de debugging).
6. Se reduce el número de errores.

Desventajas:

La principal desventaja es que no está diseñada para trabajar con test integrados, ya que se necesita tener conocimiento de los datos del repositorio. Osea tener un especial cuidado en la gestión de la BD.

Testing Angular

Los Frameworks modernos se encuentran desarrollando suites de testing integradas a sus propios sistemas. En el caso de Angular, normalmente debería usarse Jasmine y Karma.

Herramientas:

Jasmine

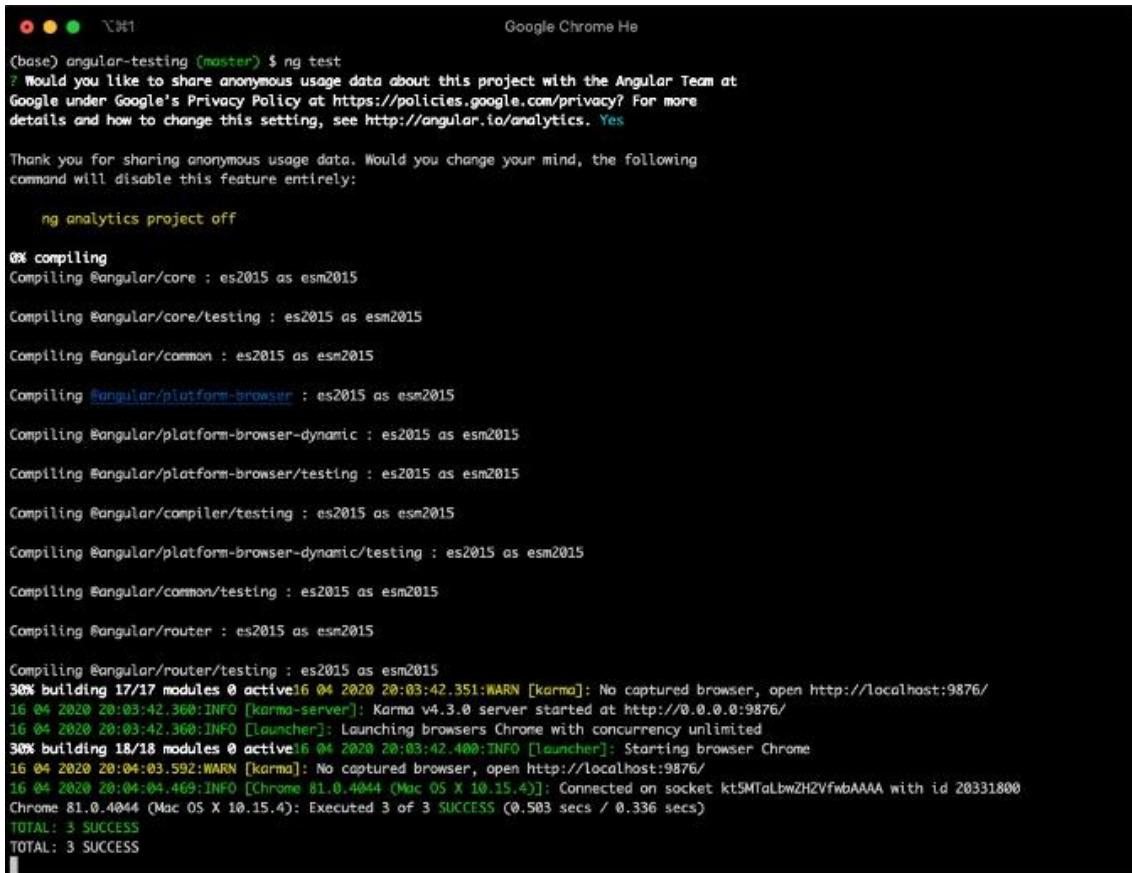
Es una suite de testing que sigue la metodología Behavior Driven Development. Tiene cosas muy buenas como que no requiere un DOM para hacer los tests y la sintaxis es bastante sencilla de entender. Aquí es donde se condifican las pruebas.

Karma

Es el test-runner, es decir, el módulo que permite automatizar algunas de las tareas de las suites de testing, como Jasmine. Karma, además, ha sido desarrollado directamente por el equipo de Angular. Este es el motor que nos permite correr las pruebas de Jasmine.

Formas de escribirlos

Vamos a recrear un ejemplo de código de una app realizada con Angular para ver cómo implementamos los tests que necesitamos. Nos vamos a saltar toda la parte de creación del proyecto y vamos directamente a generar los tests. Cabe destacar que, con Angular, desde hace unas cuantas versiones, no necesitamos configurar nada adicional para que funcione la suite de testing. Una vez creado el proyecto, tan solo será necesario ejecutar `ng test` y se ejecutarán los primeros tests, mostrándonos algo así:



Google Chrome He

```
(base) angular-testing (master) $ ng test
? Would you like to share anonymous usage data about this project with the Angular Team at
Google under Google's Privacy Policy at https://policies.google.com/privacy? For more
details and how to change this setting, see http://angular.io/analytics. Yes

Thank you for sharing anonymous usage data. Would you change your mind, the following
command will disable this feature entirely:

  ng analytics project off

0% compiling
Compiling @angular/core : es2015 as esm2015

Compiling @angular/core/testing : es2015 as esm2015

Compiling @angular/common : es2015 as esm2015

Compiling @angular/platform-browser : es2015 as esm2015

Compiling @angular/platform-browser-dynamic : es2015 as esm2015

Compiling @angular/platform-browser/testing : es2015 as esm2015

Compiling @angular/compiler/testing : es2015 as esm2015

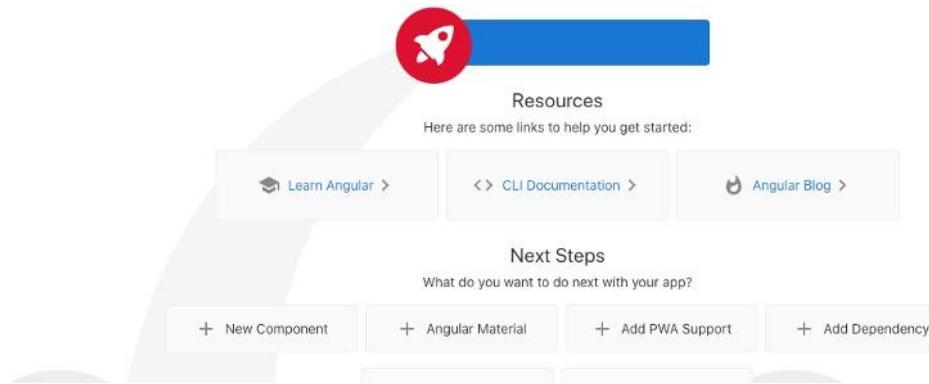
Compiling @angular/platform-browser-dynamic/testing : es2015 as esm2015

Compiling @angular/common/testing : es2015 as esm2015

Compiling @angular/router : es2015 as esm2015

Compiling @angular/router/testing : es2015 as esm2015
30% building 17/17 modules 0 active
16 04 2020 20:03:42.351:WARN [karma]: No captured browser, open http://localhost:9876/
16 04 2020 20:03:42.360:INFO [karma-server]: Karma v4.3.0 server started at http://0.0.0.0:9876/
16 04 2020 20:03:42.360:INFO [launcher]: Launching browsers Chrome with concurrency unlimited
30% building 18/18 modules 0 active
16 04 2020 20:03:42.400:INFO [launcher]: Starting browser Chrome
16 04 2020 20:04:03.592:WARN [karma]: No captured browser, open http://localhost:9876/
16 04 2020 20:04:04.469:INFO [Chrome 81.0.4044 (Mac OS X 10.15.4)]: Connected on socket kt5MTaLbwZH2VfwbAAAA with id 20331800
Chrome 81.0.4044 (Mac OS X 10.15.4): Executed 3 of 3 SUCCESS (0.583 secs / 0.336 secs)
TOTAL: 3 SUCCESS
TOTAL: 3 SUCCESS
```

Y vemos que de forma automática nos abre una ventana del navegador mostrándonos los resultados:



En ambas ventanas, aunque de diferentes maneras, vemos que los tests están corriendo, pero... ¿Cómo? ¿Qué ha pasado? Si no hemos hecho nada. Tranquilos, tan solo estamos comprobando que la suite de tests está corriendo.

Vamos a añadir nuestro componente de login. Tan solo vamos a ver un detalle. Al generar nuestro componente con el comando:

ng g component login

Obtenemos el siguiente resultado en el terminal:

```

16 04 2020 20:04:04.469:INFO [Chrome 81.0.4044 (Mac OS X 10.15.4)]:
16 04 2020 20:04:04.469:INFO [Chrome 81.0.4044 (Mac OS X 10.15.4)]:
16 04 2020 20:04:04.469:INFO [Chrome 81.0.4044 (Mac OS X 10.15.4): Executed 3 of 3 SUCCESS (0ms)
TOTAL: 3 SUCCESS
TOTAL: 3 SUCCESS
^C(base) angular-testing (master) $ ng g component login
CREATE src/app/login/login.component.scss (0 bytes)
CREATE src/app/login/login.component.html (20 bytes)
CREATE src/app/login/login.component.spec.ts (621 bytes)
CREATE src/app/login/login.component.ts (272 bytes)
UPDATE src/app/app.module.ts (471 bytes)
(base) angular-testing (master) $ 

```

Fíjense en algo muy interesante: al crear el componente nos ha generado directamente los archivos del componente y, además, nos ha generado un archivo *login.component.spec.ts*

Cuando hacemos testing, los archivos de test que se nos generan o que generamos se llaman **specs**, así que estos van a ser los archivos en los que los vamos a escribir. En este momento, recién creado, tiene este aspecto:

```
import { async, ComponentFixture, TestBed } from
'@angular/core/testing';

import { LoginComponent } from './login.component';

describe('LoginComponent', () => {
  let component: LoginComponent;
  let fixture: ComponentFixture<LoginComponent>

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ LoginComponent ]
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(LoginComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Vamos, en primer lugar a ver la estructura de un test. Fijaos en tres palabras clave, van a ser las que más vamos a usar:

```
describe('LoginComponent', () => {
  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

En este pequeño bloque tenemos la estructura básica de lo que vamos a hacer, en primer lugar, usamos *describe* para poner una descripción al grupo de tests que vamos a realizar, sobre todo para organizar los tests y que sea posible hacer seguimiento.

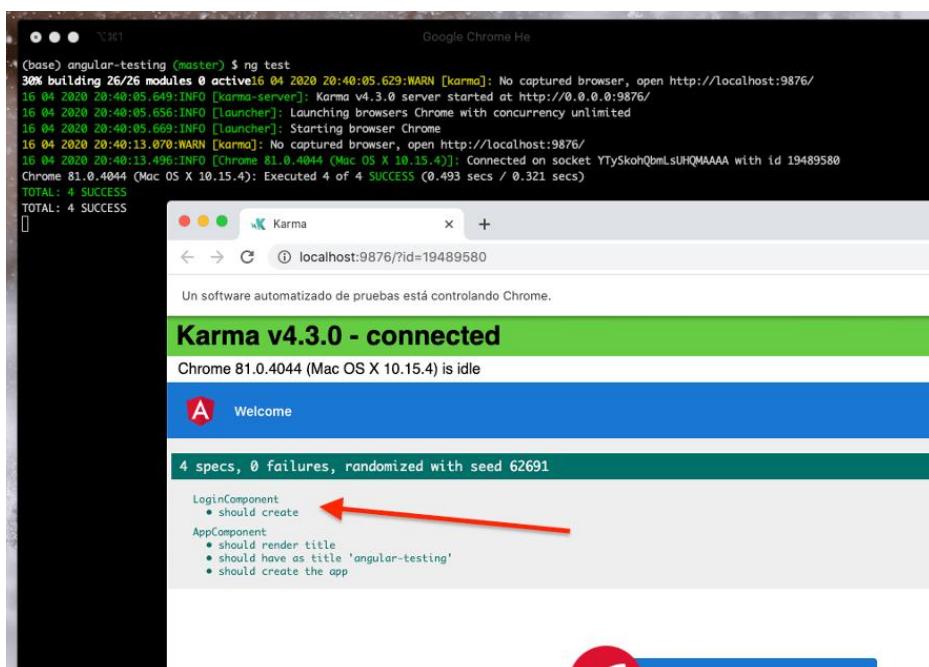
Con el *it* vamos a establecer un test en particular. En este caso, el test que se realiza es que el componente se debe crear. Simplemente es un test que se genera de forma automática cuando generamos el componente.

Por último, usamos *expect* para definir cómo debe funcionar este test en concreto, en este caso, que es uno de los más simples, tan solo se comprueba que el componente se ha generado.

Además, observen que disponemos de varios **Callbacks** que nos permiten realizar operaciones antes y después de ejecutar los tests, sobre todo para realizar la creación y la eliminación de los registros que vamos a usar durante los tests. Estos Callbacks son los siguientes:

- `beforeAll()` => Se ejecuta antes de realizar todos los tests.
- `afterAll()` => Se ejecuta después de finalizar los tests de la suite.
- `beforeEach()` => Se ejecuta antes de cada uno de los tests individuales.
- `afterEach()` => Se ejecuta al finalizar cada test individual.

Si ahora volvemos a ejecutar *ng test* vemos una pequeña diferencia con respecto a la pantalla que teníamos antes:



The terminal window shows the command `ng test` being run, followed by the build process and test results:

```
(base) angular-testing (master) $ ng test
30% building 26/26 modules 0 active
16 04 2020 20:40:05.629:WARN [karma]: No captured browser, open http://localhost:9876/
16 04 2020 20:40:05.649:INFO [karma-server]: Karma v4.3.0 server started at http://0.0.0.0:9876/
16 04 2020 20:40:05.656:INFO [launcher]: Launching browsers Chrome with concurrency unlimited
16 04 2020 20:40:05.669:INFO [launcher]: Starting browser Chrome
16 04 2020 20:40:13.070:WARN [karma]: No captured browser, open http://localhost:9876/
16 04 2020 20:40:13.496:INFO [Chrome 81.0.4044 (Mac OS X 10.15.4)]: Connected on socket YYtSkohQbmLSUHQMAAA with id 19489580
Chrome 81.0.4044 (Mac OS X 10.15.4): Executed 4 of 4 SUCCESS (0.493 secs / 0.321 secs)
TOTAL: 4 SUCCESS
TOTAL: 4 SUCCESS
```

The browser window shows the Karma test runner interface with the following details:

- Header: Karma v4.3.0 - connected
- Middle bar: Chrome 81.0.4044 (Mac OS X 10.15.4) is idle
- Bottom bar: 4 specs, 0 failures, randomized with seed 62691
- Test list:
 - LoginComponent
 - should create
 - AppComponent
 - should render title
 - should have as title 'angular-testing'
 - should create the app

Observen que ahora se han ejecutado 4 tests en lugar de los 3 que teníamos anteriormente y que nuestro test de creación del componente pasa, dado que el componente existe.

En realidad, aún no hemos creado ningún test propio. Hasta ahora solo hemos trabajado con lo que nos genera Angular de forma automática.

Aplicar la metodología TDD o BDD en nuestro desarrollo implica muchos cambios a nivel mental en nuestra forma de desarrollar. Sobre todo, hay un cambio de chip importantísimo que debemos asumir si lo queremos aplicar de forma correcta, y es lo siguiente: Los tests se escriben antes que el código.

Aunque esto parezca algo insignificante, a nivel de desarrollo implica profundos cambios con respecto a cómo estamos acostumbrados a desarrollar. El principal y más importante es que debemos tener totalmente claro cómo va a funcionar y a comportarse cada uno de nuestros componentes antes de iniciar la implementación. Esto conlleva una planificación del proyecto muy exhaustiva y, aunque parezca algo que lo va a alargar mucho, a posteriori nos daremos cuenta de que, aunque efectivamente alarga la fase de estudio previo del proyecto, luego hace que su implementación sea más rápida y, sobre todo, el mantenimiento posterior mucho más sencillo.

Vamos, siguiendo esta filosofía, a testear nuestro login. Para ello vamos a crear varios tests sencillos que definimos de antemano.

1. Debemos obtener una respuesta de ko si alguno de los campos viene vacío
2. Debemos obtener una respuesta de ok si el usuario y la contraseña son correctos y la app debe re-direccionar a la página 'dashboard'
3. Debemos obtener una respuesta de ko si el email del usuario no existe o si el password es incorrecto

Observen que hemos definido tres posibles situaciones, pero digamos que hemos cubierto la mayoría de los casos posibles en nuestro login. Ahora tenemos que implementar estos tests y hacer que pasen.

Vamos a crear el código dentro de nuestro archivo

login.component.spec.ts

```

describe('LoginComponent', () => {
  let component: LoginComponent;
  let fixture: ComponentFixture<LoginComponent>;

  const formBuilder: FormBuilder = new FormBuilder();

  beforeEach(async() => {
    TestBed.configureTestingModule({
      declarations: [ LoginComponent ],
      imports: [ ReactiveFormsModule ],
    })
    .compileComponents();
  });

  beforeEach(() => {
    fixture = TestBed.createComponent(LoginComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });

  it('should detect form is valid', () => {
    fixture.nativeElement.querySelector('button').click();

    expect(component.login()).toEqual('invalid_form');
  });

  it('should validate correct user and password', () => {
    component.loginForm = formBuilder.group({
      email: 'test@test.com',
      password: '123456'
    });
    fixture.nativeElement.querySelector('button').click();

    expect(component.login()).toEqual('login_valid');
  });

  it('should deny access with incorrect password', () => {
    component.loginForm = formBuilder.group({
      email: 'test@test.com',
      password: '123'
    });
    fixture.nativeElement.querySelector('button').click();

    expect(component.login()).toEqual('login_invalid');
  });
});

```

Y ahora, si ejecutamos los tests vemos que, obviamente, fallan. ¿Por qué? Pues porque ni el código del servicio ni el código correspondiente al login están

implementados, por lo que nuestro componente de login no hace nada y, efectivamente, los tests no pasan.

Karma v4.3.0 - connected

Chrome 81.0.4044 (Mac OS X 10.15.4) is idle

DEBUG

A Welcome Twitter

7 specs, 3 failures, randomized with seed 26555

Spec List | Failures 

finished in 0.49s

LoginComponent > should deny access with incorrect password

```
TypeError: Cannot read property 'click' of null
at <jasmine>
at UserContext.<anonymous> (http://localhost:9876/\_karma\_webpack\_/src/app/login/login.component.spec.ts:69:50)
at ZoneDelegate.invoke (http://localhost:9876/\_karma\_webpack\_/node\_modules/zone.js/dist/zone-evergreen.js:364:1)
at ProxyZoneSpec.push..>/node_modules/zone.js/dist/zone-testing.js.ProxyZoneSpec.onInvoke (http://localhost:9876/\_karma\_webpack\_/node\_modules/zone.js/dist/zone-testing.js:292:1)
at ZoneDelegate.invoke (http://localhost:9876/\_karma\_webpack\_/node\_modules/zone.js/dist/zone-evergreen.js:363:1)
at Zone.run (http://localhost:9876/\_karma\_webpack\_/node\_modules/zone.js/dist/zone-evergreen.js:123:1)
at runInTestZone (http://localhost:9876/\_karma\_webpack\_/node\_modules/zone.js/dist/zone-testing.js:545:1)
at UserContext.<anonymous> (http://localhost:9876/\_karma\_webpack\_/node\_modules/zone.js/dist/zone-testing.js:560:1)
at <jasmine>
```

LoginComponent > should validate correct user and password

```
TypeError: Cannot read property 'click' of null
at <jasmine>
at UserContext.<anonymous> (http://localhost:9876/\_karma\_webpack\_/src/app/login/login.component.spec.ts:58:50)
at ZoneDelegate.invoke (http://localhost:9876/\_karma\_webpack\_/node\_modules/zone.js/dist/zone-evergreen.js:364:1)
at ProxyZoneSpec.push..>/node_modules/zone.js/dist/zone-testing.js.ProxyZoneSpec.onInvoke (http://localhost:9876/\_karma\_webpack\_/node\_modules/zone.js/dist/zone-testing.js:292:1)
at ZoneDelegate.invoke (http://localhost:9876/\_karma\_webpack\_/node\_modules/zone.js/dist/zone-evergreen.js:363:1)
at Zone.run (http://localhost:9876/\_karma\_webpack\_/node\_modules/zone.js/dist/zone-evergreen.js:123:1)
at runInTestZone (http://localhost:9876/\_karma\_webpack\_/node\_modules/zone.js/dist/zone-testing.js:545:1)
at UserContext.<anonymous> (http://localhost:9876/\_karma\_webpack\_/node\_modules/zone.js/dist/zone-testing.js:560:1)
at <jasmine>
```

LoginComponent > should detect empty fields

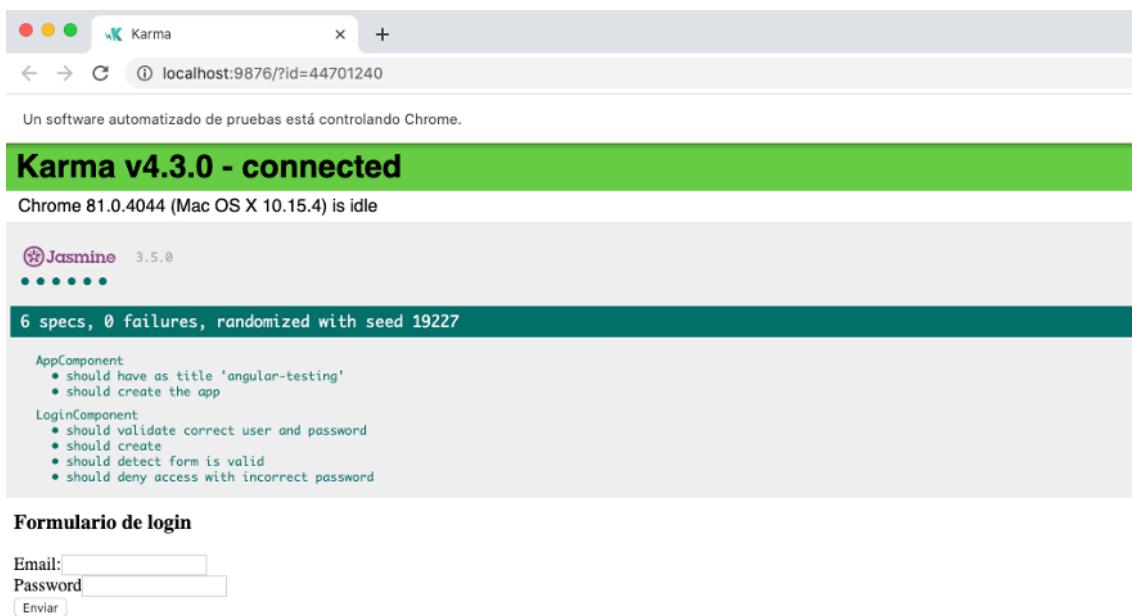
```
TypeError: Cannot read property 'click' of null
at <jasmine>
at UserContext.<anonymous> (http://localhost:9876/\_karma\_webpack\_/src/app/login/login.component.spec.ts:48:50)
at ZoneDelegate.invoke (http://localhost:9876/\_karma\_webpack\_/node\_modules/zone.js/dist/zone-evergreen.js:364:1)
at ProxyZoneSpec.push..>/node_modules/zone.js/dist/zone-testing.js.ProxyZoneSpec.onInvoke (http://localhost:9876/\_karma\_webpack\_/node\_modules/zone.js/dist/zone-testing.js:292:1)
at ZoneDelegate.invoke (http://localhost:9876/\_karma\_webpack\_/node\_modules/zone.js/dist/zone-evergreen.js:363:1)
at Zone.run (http://localhost:9876/\_karma\_webpack\_/node\_modules/zone.js/dist/zone-evergreen.js:123:1)
at runInTestZone (http://localhost:9876/\_karma\_webpack\_/node\_modules/zone.js/dist/zone-testing.js:545:1)
at UserContext.<anonymous> (http://localhost:9876/\_karma\_webpack\_/node\_modules/zone.js/dist/zone-testing.js:560:1)
at <jasmine>
```

En este caso observen que nos indica que tres de nuestros tests no han pasado. El error es, concretamente, que el botón de *submit* no existe. Ni siquiera hemos creado el formulario.

No obstante, hay una cosa muy importante con respecto al esquema que antes explicaba. Ya tenemos totalmente definido cómo debe funcionar nuestro sistema de login y sus reglas principales. Cuando ahora implementemos el código, tenemos totalmente claro cómo debe responder, lo que no nos deja dudas a la hora de hacerlo, sin lugar a interpretación que, como todos sabemos, es un gran problema en desarrollo.

Vamos a implementar el código del componente de *login* (podés ver el código completo en este repositorio). En este caso estamos realizando un ejemplo muy sencillo, por lo que he adaptado el código tanto del formulario de login, que es extremadamente simple, como el del componente, que no realiza un login real sobre un api, sino que lo simula para que veamos exactamente como responde cada uno de los tests tal y como los hemos fijado en los specs.

Y ahora, cuando pasamos los tests obtenemos la siguiente pantalla, en la que todos nuestros tests pasan sin problema:



Un software automatizado de pruebas está controlando Chrome.

Karma v4.3.0 - connected

Chrome 81.0.4044 (Mac OS X 10.15.4) is idle

Jasmine 3.5.0

• • • • •

6 specs, 0 failures, randomized with seed 19227

```

AppComponent
  • should have as title 'angular-testing'
  • should create the app

LoginComponent
  • should validate correct user and password
  • should create
  • should detect form is valid
  • should deny access with incorrect password

```

Formulario de login

Email:

Password:

Testing Spring Boot

Herramientas:

Junit 4

Es la herramienta de testing automatizado más popular para Java. *JUnit* puede utilizarse para cualquier tipo de testing automatizado y no solo para pruebas unitarias. Proporciona la estructura base sobre la cual implementar los tests, en la cual se apoyan otras herramientas más complejas. Los test de *JUnit* se implementan muy fácilmente. Se utilizan clases POJO, normalmente marcadas con el sufijo “Test”.

Mockito

Mockito es un framework de Mocking. Te permite escribir hermosos test con una API limpia y simple. Sus tests son muy sencillos de leer y producen verificaciones de errores. *Mockito* está basado en *EasyMock*, y el funcionamiento es muy parecido, aunque mejora el api a nivel sintáctico, haciéndolo más entendible para nosotros, y además permite crear mocks de clases concretas.

MockMvc

MockMvc se encuentran en la frontera entre los tests unitarios y los tests de integración. No son tests unitarios porque los endpoints se prueban de forma integrada con un contenedor MVC con dependencias e inputs simulados. Pero tampoco podríamos considerar estos test como de integración, ya que si se definen correctamente, con *MockMvc* sólo probaremos un único componente con una plataforma simulada, pero no una combinación de componentes.

REST Assured

Es un Framework escrito en Java y diseñado para simplificar las pruebas sobre servicios basados en REST. Ofrece un DSL descriptivo (lenguajes específicos de dominio), que muestra una unión a un punto de conexión HTTP y da los resultados esperados. Este framework soporta las operaciones POST, GET, PUT, DELETE, OPTIONS, PATCH y HEAD y contiene herramientas para invocarlas y verificarlas.

Comandos

Anotaciones Junit

Anotación	Descripción
@Test	Indica a JUnit que se trata de un método de test.
@Before	Se ejecuta siempre antes de cada método de test. Se suele utilizar para preparar el entorno de prueba (por ejemplo: leer datos de entrada, inicializar la clase, etc..)
@After	Se ejecuta siempre después de cada método de test. Se suele utilizar para limpiar el entorno de desarrollo (por ejemplo: borrar datos temporales, restaurar valores por defecto, etc..)
@BeforeClass	Se ejecuta solo una vez, antes de la ejecución de todos los tests. Se suele utilizar para ejecutar actividades de inicio muy costosas (por ejemplo: conexión a una base de datos). Estos métodos se tienen que definir como estáticos.
@AfterClass	Se ejecuta solo una vez, después de la ejecución de todos los tests. Se suele utilizar para ejecutar actividades de cierre (por ejemplo: desconexión a una base de datos). Estos métodos se tienen que definir como estáticos.
@Ignore	Indica a JUnit que el método de test está deshabilitado. Útil cuando el código ha cambiado sustancialmente y el test no está adaptado. Es de buena práctica indicar el motivo de por qué se ha deshabilitado.

Aserciones Junit

Método	Descripción
assertTrue([mensaje], condición booleana)	Comprueba que la condición sea verdadera.
assertFalse([mensaje], condición booleana)	Comprueba que la condición sea falsa.
assertEquals([mensaje], valor esperado, valor actual)	Comprueba que dos valores sean iguales. Nota: en arrays comprueba su referencia, no el contenido!)
assertSame([mensaje], valor esperado, valor actual)	Comprueba que ambos parámetros sean el mismo objeto.
assertNotSame([mensaje], valor esperado, valor actual)	Comprueba que ambos parámetros no sean el mismo objeto.
assertNull([mensaje], objeto)	Comprueba que el objeto sea nulo.
assertNotNull([mensaje], objeto)	Comprueba que el objeto no sea nulo.
fail([mensaje])	Hace que el método falle. Debería ser utilizado solo para comprobar que una parte del código de test no se ejecute o para hacer fallar un test no implementado.

Formas de escribirlos

Spring es un framework de Inyección de dependencia y para hacer, aunque se la prueba unitaria más sencilla del mundo necesitaremos pre-configurar algunas cosas antes de poder ejecutarla sobre *Spring Framework*. El primer paso será configurar el proyecto de *Maven* para poder disponer de las dependencias de *Spring*.

```

<dependencies>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>5.2.9.RELEASE</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-context
-->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.2.9.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.13</version>
        <scope>test</scope>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.springframework/spring-test -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
        <version>5.2.9.RELEASE</version>
        <scope>test</scope>
    </dependency>

</dependencies>
```

En este caso hemos añadido tanto las dependencias de *Spring* como las dependencias de *Junit*. El siguiente paso es generar un Servicio que tenga la anotación `@Service`. Este servicio devolverá una lista de personas.

```
package com.arquitecturajava.testing;

public class Persona {

    private String nombre;
    private int edad;

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }

    public Persona(String nombre, int edad) {
        super();
        this.nombre = nombre;
        this.edad = edad;
    }

}
```

Vamos a ver el código del Servicio:

```
package com.arquitecturajava.testing;

import java.util.Arrays;
import java.util.List;

import org.springframework.stereotype.Service;

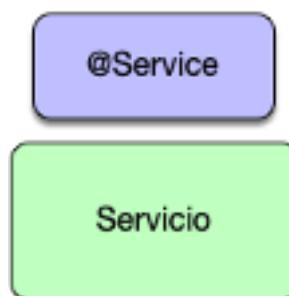
@Service
public class Servicio {

    public List<Persona> buscarTodas() {

        Persona p = new Persona("pepe", 20);
        Persona p2 = new Persona("ana", 30);

        List<Persona> lista = Arrays.asList(p, p2);
        return lista;
    }
}
```

Como vemos hemos añadido la anotación de `@Service` lo que convierte a la clase de Servicio en un Servicio de Spring Framework a disposición del inyector de dependencia.



Es momento de generar una clase de Configuración que sea capaz de localizar el Servicio para posteriormente inyectarlo a nivel de prueba unitaria.

```

package com.arquitecturajava.testing;

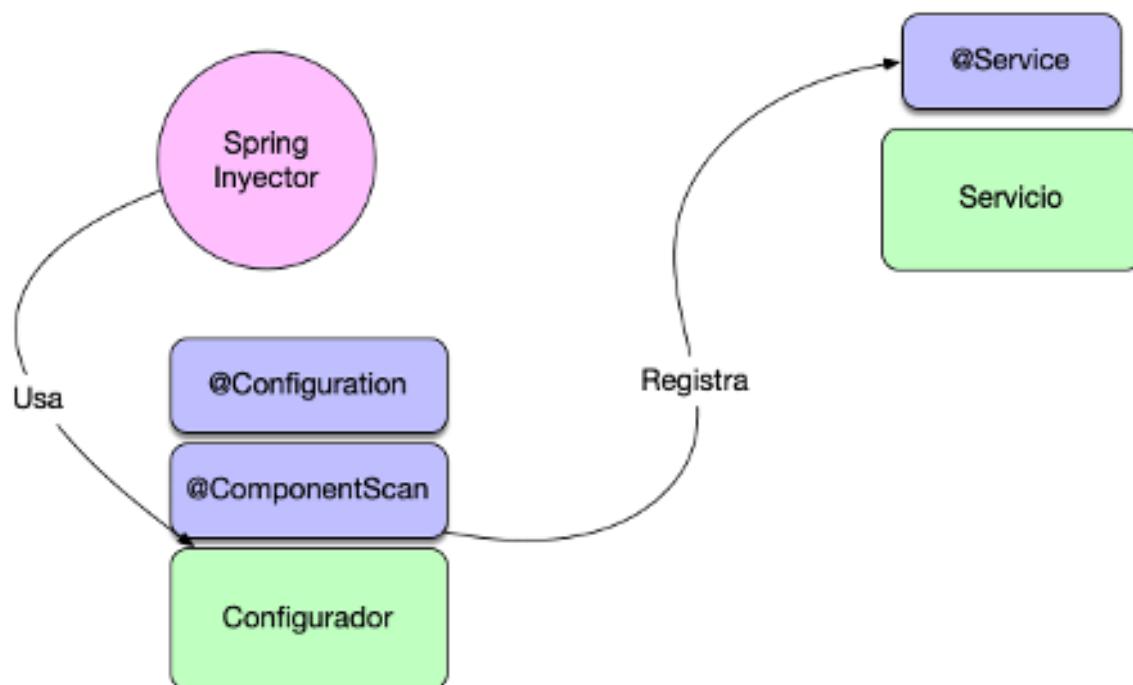
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan("com.arquitecturajava")
public class ConfiguradorSpring {

}

```

En este caso la anotación de `@ComponentScan` se encarga de localizar el Servicio y registrarlo.



El paso que nos queda es crear la prueba unitaria y ejecutarla.

```
package com.arquitecturajava.testing.test;

import static org.junit.Assert.*;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import com.arquitecturajava.testing.ConfiguradorSpring;
import com.arquitecturajava.testing.Servicio;

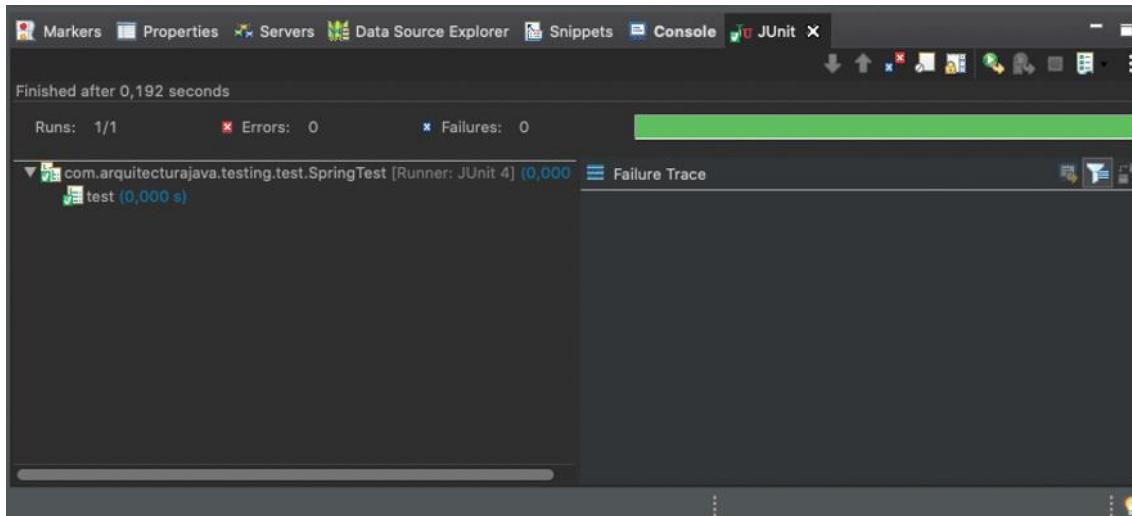
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = { ConfiguradorSpring.class })
public class SpringTest {

    @Autowired
    private Servicio miservicio;
    @Test
    public void test() {

        assertEquals(2,miservicio.buscarTodas().size());
    }

}
```

En este caso hay que usar la anotación `@RunWith` que nos permite ejecutar pruebas unitarias sobre el paraguas de Spring Framework. Evidentemente para poder hacerlo necesitaremos declarar cual es el Contexto de Spring que vamos a utilizar en este caso `ConfigurarSpring.class` que se encarga de cargar en memoria la clase de Servicio. Hecho esto ejecutamos la prueba unitaria y acabamos de pasar el test más básico apoyándonos en Spring.



Referencias:

<https://www.paradigmadigital.com/techbiz/tdd-una-metodologia-gobernarlos-todos/#:~:text=TDD%20son%20las%20siglas%20de,c%C3%B3digo%20que%20tenemos%20que%20implementar.>

<https://programacionymas.com/blog/tipos-de-testing-en-desarrollo-de-software>

<https://www.digital55.com/desarrollo-tecnologia/como-usar-testing-angular-jasmine-karma/>

<https://medium.com/@jorgeucano/introducci%C3%B3n-al-testing-en-angular-da415ef8c47>

<https://danielme.com/2018/11/26/testing-en-spring-boot-con-junit-45-mockito-mockmvc-rest-assured-bases-de-datos-embebidas/>

<https://danielme.com/2016/09/07/tutorial-testing-con-junit-4/>

<https://www.nestoralmeida.com/testing-con-junit-4/>

<https://www.sdos.es/blog/descubre-como-automatizar-service-tests-con-rest-assured#:~:text=Rest%20Assured%20es%20utilizado%20para,el%20cuerpo%20de%20la%20respuesta.>

<https://www.arquitecturajava.com/spring-testing-y-el-manejo-de-junit/>

Cómo subir tu proyecto Angular a la Nube

Firebase es una suite de servicios que automatiza algunos de los elementos del desarrollo de una aplicación. Piensa en Firebase como otorgarle poderes nuevos a tu aplicación móvil o a tu plataforma web.

La base de datos en tiempo real es quizás el servicio más popular y usado de la suite de Firebase, sin embargo, no es el único de los servicios, además de la base de datos en tiempo real, también tenemos:

- Analytics para tu aplicación.
- Almacenamiento en la nube de archivos.
- Autenticación con redes sociales o correo/contraseña.
- Hosting en la nube
- Otros

Vamos a utilizar el Hosting para subir una aplicación de Angular. La guía funciona para una aplicación básica de Angular

PREPARANDO EL ENTORNO

Para poder subir una página web al hosting de la nube de Firebase, primero tenemos que instalar las herramientas de Firebase (firebase tools), una utilidad que instalas para la terminal, a través de la cual puedes ejecutar comandos para administrar tus proyectos de Firebase, entre otras cosas, para subir una carpeta con un sitio web al hosting.

Para instalar las Firebase Tools, necesitas primero tener instalado NodeJS y NPM, una vez que hayas instalado estas dependencias, debes ejecutar el siguiente comando dentro del CMD o la Terminal, dependiendo de si estás en Windows, MacOS o Linux:

```
npm install -g firebase-tools
```

Este comando instalará el paquete, que a su vez expondrá un comando `firebase` en la terminal, que como decía, te permite administrar tus proyectos.

FIREBASE CONSOLE.

Para poder administrar tu cuenta Firebase, primero necesitas haber creado una, ingresa a la dirección <http://console.firebaseio.google.com> para poder iniciar una cuenta Firebase, para hacerlo necesitas tener una cuenta de Google (como un correo de Gmail), al ingresar se creará una cuenta de Firebase para tu cuenta activa de Google, si no habías iniciado sesión en este navegador con una cuenta de Google, se te pedirá que ingreses una.

Welcome to Firebase!

Tools from Google for developing great apps, engaging with your users, and earning more through mobile ads.

[Learn more](#) [Documentation](#) [Support](#)



Una vez en la pantalla que aparece como imagen justo arriba de este texto, vamos a seleccionar la opción **Add project** para crear un nuevo proyecto. Aquí se te pedirá que coloques el nombre del proyecto con el que lo identificarás, el nombre es deliberado y puedes colocar el que mejor describa tu proyecto.

LOGIN EN FIREBASE TOOLS

Ya que tienes una cuenta creada y un proyecto configurado, necesitas enlazar tu cuenta de Firebase con las firebase-tools que previamente instalaste. Para eso debes ejecutar el siguiente comando en la terminal:

```
firebase login
```

Inicialmente se te preguntará si quieres que tu información de uso de la herramienta sea compartida con Firebase o no, si decidás compartir tus datos escribe **Y**, si no, escribe **n**, luego presiona Enter.

Este comando deberá automáticamente abrir en tu navegador web una página para que selecciones la cuenta con la que enlazarás tu herramienta de la terminal, adicionalmente puedes copiar la URL que aparece en negritas en la terminal, en la barra de direcciones de tu navegador para ver la misma página.

Ahí, selecciona la cuenta con la que quieras enlazar la terminal, debe ser la misma que creaste en el paso anterior, para usar el proyecto que creamos. Una vez que autorices los permisos para la app de Firebase, automáticamente la terminal detectará que ya autorizaste y enlazaste tu aplicación mostrando un mensaje como el siguiente:

```
✓ Success! Logged in as argentinaprograma.ar
```

Subir tu sitio a Firebase Hosting

Para subir tu proyecto al hosting cloud de Firebase, debes primero posicionar la terminal en la carpeta que contiene el proyecto. Como este ejemplo está orientado a una aplicación de angular, el primer paso para la subida será generar el *build* de Angular.

ANGULAR BUILD

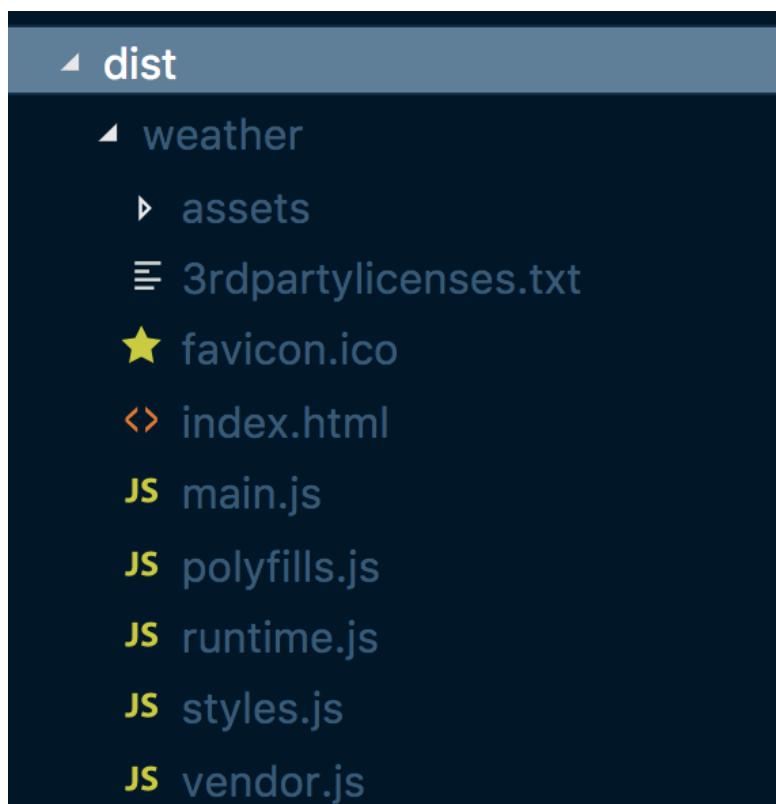
El buid es una versión optimizada de nuestra aplicación, en la que el código fue comprimido, reducido y optimizado para que cargue más rápido, sea de menor tamaño y en general mejore su rendimiento. Para generar el build coloca el siguiente comando en la Terminal o el CMD:

```
ng build
```

El resultado de este comando debe verse como a continuación te muestro:

```
Date: 2018-07-15T06:18:49.945Z
Hash: 8ee114542ed8a659f7e1
Time: 7756ms
chunk {main} main.js, main.js.map (main) 55 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 227 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 5.22 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 16.1 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.11 MB [initial] [rendered]
```

Esto quiere decir que el build de tu aplicación fue generado con éxito, para comprobarlo debes verificar que se añadió una carpeta `dist` en tu proyecto de Angular:



Subir el build a Firebase

Para comenzar con la subida ejecuta el comando `init` de Firebase Tools para que la herramienta configure el proyecto como un proyecto de Angular, y además, enlaza el proyecto que previamente creamos en la consola de Firebase con la carpeta actual, para eso coloca el siguiente comando:

```
firebase init
```

Este comando iniciará un asistente que te guiará por la configuración del proyecto, las opciones que debes seleccionar son:

1. Usando las flechas arriba/abajo selecciona qué característica de Firebase usarás, una vez que hayas seleccionado la del Hosting, presiona la barra espaciadora para activar el uso de esta característica, luego presiona Enter
2. Usando las flechas arriba/abajo selecciona el proyecto con el que integrarás la carpeta actual, en este paso debes seleccionar el proyecto que creamos con anterioridad en este mismo artículo, presiona Enter cuando hayas seleccionado el proyecto adecuado.
3. El siguiente punto es muy importante cuando trabajamos con un proyecto de Angular, en este paso el asistente de Firebase nos pregunta qué directorio contiene nuestra aplicación, es muy importante que indiquemos que es el directorio **dist/nombre-proyecto** sustituye la parte de nombre-proyecto por la carpeta que veas dentro de dist

```
? What do you want to use as your public directory? (public) dist/nombre-proyecto
```

1. El siguiente paso, está relacionado con las Single Page Applications, como Angular suele contener su propio router (si no lo contiene, no importa no es necesario), escribiremos `y` y luego presionaremos Enter. Luego de eso, Firebase notará que ya existe un archivo `index.html` y te preguntará si quieres que lo sobre escriba, o no, escribe `ny` luego Enter para indicarle que no debe sobre escribirlo y se conserve el archivo `index.html` de tu proyecto Angular.

Esto debe completar el asistente de Firebase y habremos configurado con éxito nuestro directorio como una app de Angular que podemos subir al Firebase Hosting. Para terminar, vamos a ejecutar el siguiente comando que se encargará de subir nuestros archivos a Firebase para que podamos visualizar nuestra página:

```
firebase deploy
```

Si todo sale bien, deberás ver un mensaje como el siguiente:

```
== Deploying to 'hosting-angular-56167'...

i deploying hosting
i hosting: preparing dist directory for upload...
✓ hosting: 43 files uploaded successfully

✓ Deploy complete!

Project Console: https://console.firebaseio.google.com/project/hosting-angular-56167/overview
Hosting URL: https://hosting-angular-56167.firebaseioapp.com
```

Como parte del mensaje podrás notar que se imprime la URL en la que podrás ubicar tu sitio web, en mi caso es <https://hosting-angular-56167.firebaseioapp.com> si ingresas ahí, deberías ver tu página web funcionando y en línea

Integración continua (CI/CD)

FUTUROS CAMBIOS.

Lo interesante de Firebase Hosting es que, con una configuración muy simple, ahora tienes un flujo de desarrollo en el que cada modificación a tu aplicación puede ser publicada con un solo comando en la terminal.

Si hablamos de un proyecto de Angular, una vez que quieras subir cambios a tu página web, debes ejecutar los siguientes comandos:

```
ng build
firebase deploy
```

CONCLUSIÓN

Firebase ofrece una solución excelente para almacenar tus páginas web o aplicaciones en el frontend, entre ellas por supuesto tus aplicaciones construidas con Angular. Este servicio ofrece muchos beneficios como un alto rendimiento para cargar tu página, lo que hace que ésta cargue más rápido, un flujo de subida de cambios o actualizaciones que involucra un simple comando, configuración sencilla y muchísimo más.

Deploy Backend – Subir tu proyecto Spring Boot a la nube de Fly.io

Al desarrollar una aplicación, la progresión natural es ponerla en línea y ponerla a disposición de los usuarios finales. Básicamente es habilitar la aplicación para su uso, ya sea un ambiente de desarrollo, para realizar pruebas o ponerla en producción. Para que esta tarea sea posible y más fácil, existen numerosas plataformas en la nube disponibles para alojar su aplicación. Fly.io es una de ellas, la cual nos permite implementar cualquier tipo de aplicación backend o frontend.

¿Qué es Fly.io?

Es un servicio de plataforma como servicio o también llamado PaaS.

Básicamente es un conjunto de servicios basados en la nube que permite a los desarrolladores y usuarios empresariales crear aplicaciones a una velocidad que las soluciones en las instalaciones propias no pueden alcanzar.

Por supuesto que, al tratarse de un servicio basado en la nube, no hay necesidad de preocuparse por la configuración y el mantenimiento de servidores, parches, actualizaciones y autenticaciones, entre muchas otras tareas.

Crear una app Spring Boot

Inicialización de la aplicación Spring Boot

Como siempre, la forma más fácil de comenzar con un proyecto esqueleto de Spring Boot es usar el [Spring Initializer](#):

Elija su herramienta de construcción preferida, usaremos Maven. La única dependencia que necesitaremos es la **Spring Web** dependencia.

Alternativamente, podemos crear la aplicación usando el [Spring Boot CLI](#):

```
$ spring init --dependencies=web heroku-demo
```

Crear un punto final REST

Con nuestro esqueleto hecho, agreguemos un punto final REST simple:

```
@RestController
@RequestMapping("/api/v1.0")
public class TimeController {

    @GetMapping("/time")
    @ResponseStatus(HttpStatus.OK)
    public String getCurrentTime() {

        return Instant.now().toString();
    }
}
```

En caso de que no esté familiarizado con el `@RestController` anotación, es una anotación de conveniencia hecha como una combinación de la `@Controller` y `@ResponseBody` anotaciones.

Este punto final simplemente devolverá la hora actual del servidor a pedido. Ejecutemos la aplicación en nuestra máquina local y probemos si está funcionando:

```
$ mvn spring-boot:run
```

O, usando su IDE, simplemente ejecute la aplicación y navegue hasta el `localhost:8080/api/v1.0/time` Dirección URL.

Alternativamente, puede utilizar una herramienta como `curl` para llegar al punto final:

```
$ curl -X GET 'http://localhost:8080/api/v1.0/time'  
2020-01-04T13:19:30.980Z
```

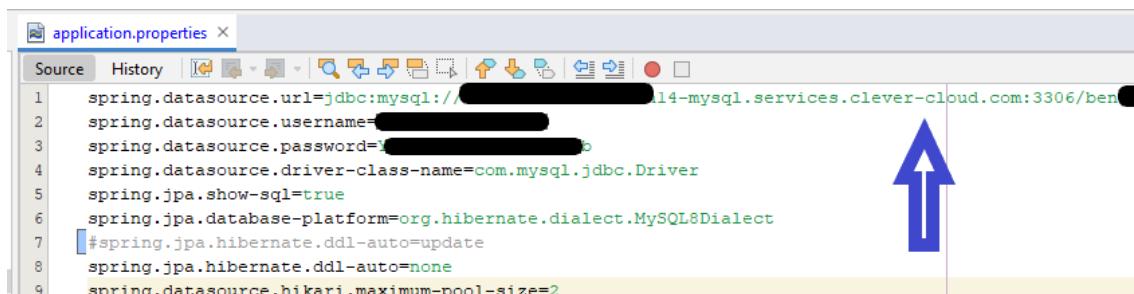
Desplegar aplicación Springboot en Fly.io usando Dockerfile

Fly.io admite implementar aplicaciones de múltiples formas, pero en este caso lo haremos a través del uso de dockerfile. Eso significa que simplemente implementaremos una imagen de Docker de forma totalmente gratuita.

Para profundizar sobre la generación de imágenes docker de aplicaciones locales de springboot puede seguir la siguiente guía: <https://spring.io/guides/gs/spring-boot-docker/>

Preparando el Entorno – Pre requisitos:

- Conocimientos básicos de creación de imágenes de docker.
 - Tener compilación funcionando de proyecto springboot en su computadora local.
 - Tener configurada la conexión a la base de datos en la nube de clevercloud.



- Tener habilitado CORS en tus endpoint: En algunos casos puede aparecer el error “No hay un encabezado 'Access-Control-Allow-Origin' presente en el recurso solicitado” al intentar invocar la API

Los errores de uso compartido de recursos entre orígenes (CORS) ocurren cuando un servidor no devuelve los encabezados HTTP que exige el estándar

CORS. Para resolver un error CORS de una API REST de API Gateway o API HTTP, debe configurar de nuevo la API para que cumpla con el estándar CORS.

Existen varias alternativas para solucionarlo, lo que se propone es agregar una clase en tu backend para habilitar todos los orígenes que petitionan:

```

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

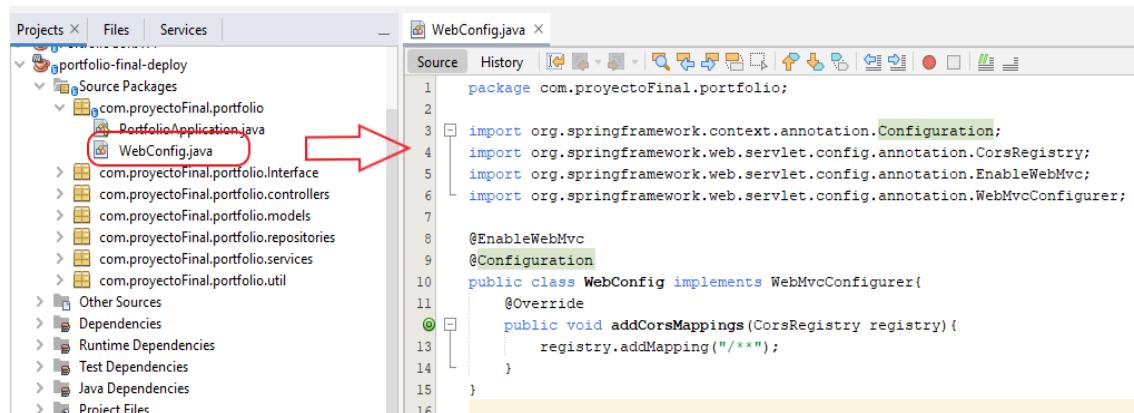
```

```

/**
 * Clase que habilita CORS
 * @author YOProgramo
 */
@EnableWebMvc
@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**");
    }
}

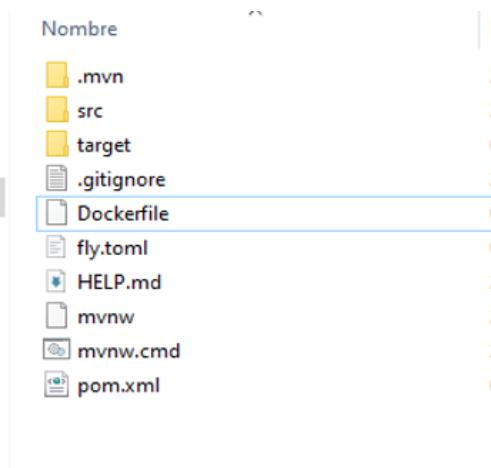
```

Su clase debería quedar tal cual muestra el ejemplo:



Desplegando: pasos a seguir:

- Crear archivo *Dockerfile*, en la carpeta de tu proyecto springboot:

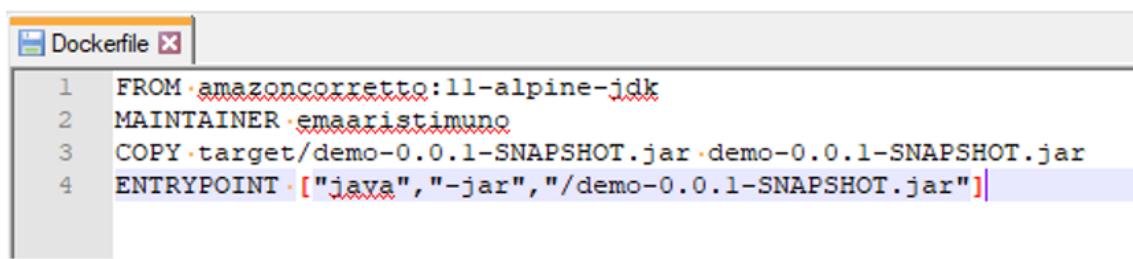


- Al archivo creado agregar la **dependencia** necesaria para levantar una versión de JDK igual a la que usas en tu proyecto local, en el ejemplo suponemos que tenemos la versión 11, por lo tanto elegimos una versión JDK11: **amazoncorreto:11-alpine-jdk** (Existen múltiples paquetes de JDK, puedes elegir el que prefieras en: https://hub.docker.com/_/openjdk) y la referencia del archivo **.jar** que contiene la compilación de nuestro proyecto springboot:

Código base:

```
FROM amazoncorreto:11-alpine-jdk
MAINTAINER emaaristimuno
COPY target/NAME-YOUR-FILE-BUILD-SPRINGBOOT.jar NAME-YOUR-FILE-BUILD-SPRINGBOOT.jar
ENTRYPOINT ["java","-jar","/NAME-YOUR-FILE-BUILD-SPRINGBOOT.jar"]
```

Ejemplo:



```
1 FROM amazoncorreto:11-alpine-jdk
2 MAINTAINER emaaristimuno
3 COPY target/demo-0.0.1-SNAPSHOT.jar demo-0.0.1-SNAPSHOT.jar
4 ENTRYPOINT ["java","-jar","/demo-0.0.1-SNAPSHOT.jar"]
```

Para el ejemplo, la compilación que se tendrá en cuenta para crear la imagen se encuentra dentro de la carpeta target:

target/demo-0.0.1-SNAPSHOT.jar

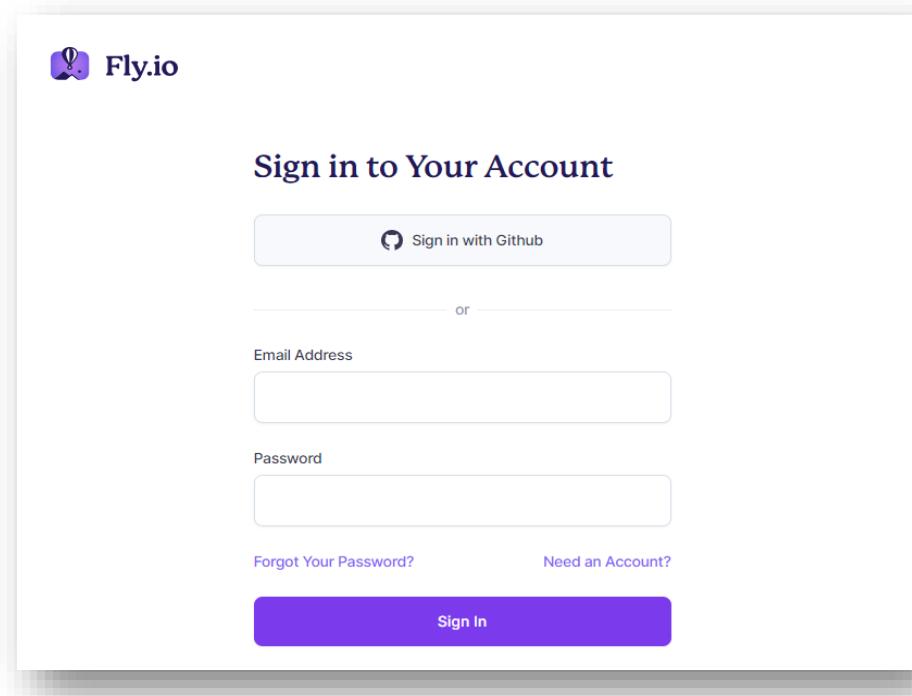
demo > target

Nombre	Fecha de modificación	Tipo
classes	03/10/2022 16:11	Carpeta de archivos
generated-sources	03/10/2022 16:11	Carpeta de archivos
generated-test-sources	03/10/2022 16:11	Carpeta de archivos
maven-archiver	03/10/2022 16:11	Carpeta de archivos
maven-status	03/10/2022 16:11	Carpeta de archivos
surefire-reports	03/10/2022 16:11	Carpeta de archivos
test-classes	03/10/2022 16:11	Carpeta de archivos
demo-0.0.1-SNAPSHOT.jar	03/10/2022 16:11	Archivo JAR
demo-0.0.1-SNAPSHOT.jar.original	03/10/2022 16:11	Archivo ORIGINAL

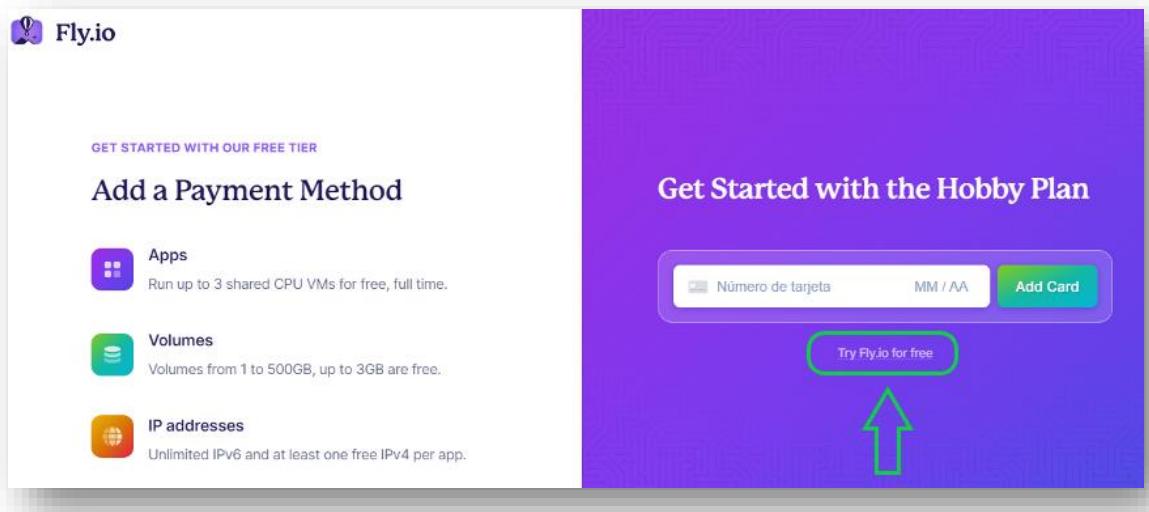
3. Instalar utilidad de comandos para trabajar con Fly: [flyctl](#). Ejecutando en Powershell de Windows: `iwr https://fly.io/install.ps1 -useb | iex`

```
$ iwr https://fly.io/install.ps1 -useb | iex
```

4. Si esta es tu primera vez con Fly.io, tu próximo paso será [Registrarte](#) (usa tu cuenta de github):



5. Te pedirá que ingreses un método de pago, pero no te preocupes puedes elegir **Try Fly.io for Free** como plan para comenzar:



6. Iniciar sesión en Fly: ***flyctl auth signup***

```
$ flyctl auth signup
```

Esto lo llevará a la página de registro donde puede:

Regístrate con correo electrónico: Ingrese su nombre, correo electrónico y contraseña.

Regístrate con GitHub: si tiene una cuenta de GitHub, puede usarla para registrarse. Esté atento al correo electrónico de confirmación que se enviará, que le dará un enlace para establecer una contraseña; necesitará establecer una contraseña para que podamos verificar activamente que es usted para algunas operaciones de Fly.io.

7. Cada aplicación Fly.io necesita un archivo ***fly.toml*** para decirle al sistema cómo nos gustaría implementarlo. Ese archivo se puede generar automáticamente con el comando ***flyctl launch***, que hará algunas preguntas para configurar todo. Repasemoslo ahora:

7.1 Ejecutar los siguientes comandos dentro del directorio donde se encuentra el dockerfile:

```
> flyctl launch
```

7.2 Definir un nombre para nuestra aplicación desplegada en Fly:

```
Creating app in C:\Users\ema_2\Documents\NetBeansProjects\BackendArgProg22-main
Scanning source code
Detected a Dockerfile app
? Choose an app name (leave blank to generate one): deploy-springboot
```

7.3 Seleccionar la región que levantara nuestra app:

```
? Choose an app name (leave blank to generate one): deploy-springboot
automatically selected personal organization: emanueluader@gmail.com
? Choose a region for deployment: [Use arrows to move, type to filter]
  London, United Kingdom (lhr)
  Chennai (Madras), India (maa)
  Madrid, Spain (mad)
  Miami, Florida (US) (mia)
  Tokyo, Japan (nrt)
  Chicago, Illinois (US) (ord)
  Bucharest, Romania (otp)
> Santiago, Chile (scl)
  Seattle, Washington (US) (sea)
  Singapore, Singapore (sin)
  San Jose, California (US) (sjc)
```

Nos creará un archivo **fly.toml** con las configuraciones necesarias:

```
? Choose an app name (leave blank to generate one): deploy-springboot
automatically selected personal organization: emanueluader@gmail.com
? Choose a region for deployment: Santiago, Chile (scl)
Created app deploy-springboot in organization personal
Wrote config file fly.toml
```

7.4 Nos pregunta si queremos instalar una Base de Datos Postgresql, seleccionar que **No**:

```
? Would you like to set up a Postgresql database now? No
```

7.5 Generar el Deploy respondiendo **Yes** a la pregunta:

```
? Would you like to deploy now? Yes
==> Building image
Waiting for remote builder fly-builder-long-cherry-2293... ●
```

Esperar un momento hasta que se genere todos los procesos necesarios para subir la imagen de docker a fly (Según su conexión de internet puede demorar entre 5 y 30 minutos :/):

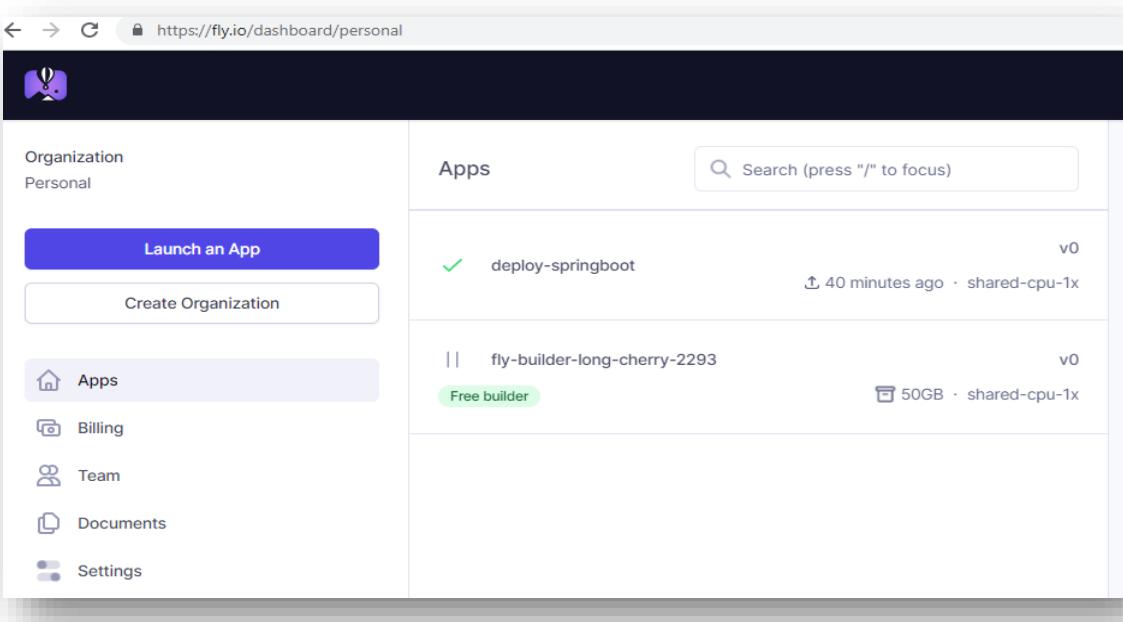
```
? Would you like to deploy now? Yes
==> Building image
Remote builder fly-builder-long-cherry-2293 ready
==> Creating build context
--> Creating build context done
==> Building image with Docker
--> docker host: 20.10.12 linux x86_64
[+] Building 1471.7s (0/1)
[+] Building 2.8s (6/6) FINISHED
```

Si el despliegue se realizó de manera exitosa, la consola arrojará los siguientes mensajes:

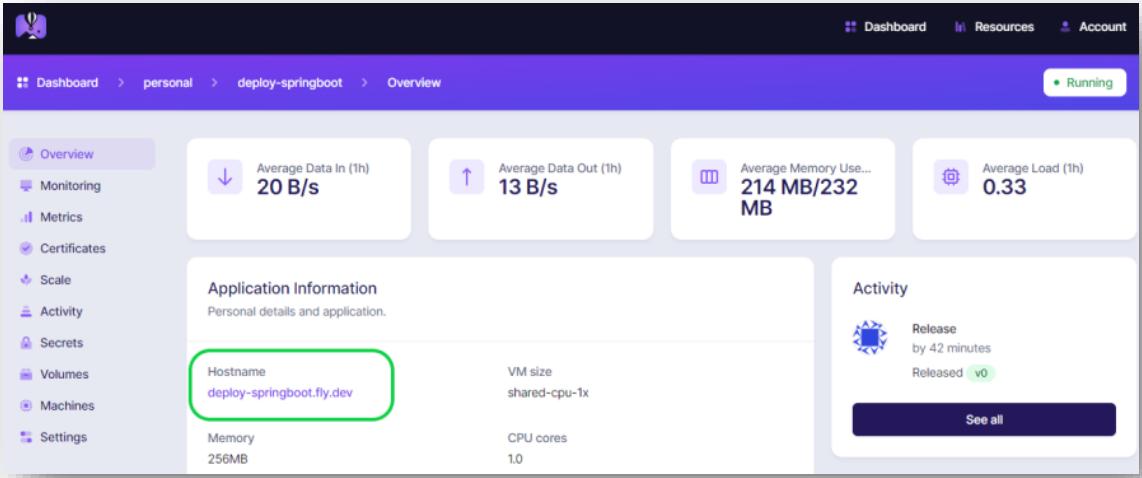
```
[+] Building 2.8s (6/6) FINISHED
=> [internal] load remote build context
=> copy /context /
=> [internal] load metadata for docker.io/library/amazoncorretto:11-alpine-jdk
=> [1/2] FROM docker.io/library/amazoncorretto:11-alpine-jdk@sha256:9eddc0fc6e35c4a09fc402bf67e0aac986ae01e96fb0cb74059f4914016a24d
=> CACHED [2/2] COPY target/portfolio-final-deploy-0.0.1-SNAPSHOT.jar portfolio-final-deploy-0.0.1-SNAPSHOT.jar
=> exporting to image
=> => exporting layers
=> => writing image sha256:caa5d2152a0fcadefef5a9aaaf277c68f5d2231563de47a49cb636bc8b8635e841
=> => naming to registry.fly.io/deploy-springboot:deployment-01GH19ATHCPVQ4FKYAWWRC39Y0
--> Building image done
==> Pushing image to fly
The push refers to repository [registry.fly.io/deploy-springboot]
2af73c1c4a30: Pushed
a7579190b8e3: Pushed
994393dc58e7: Pushed
deployment-01GH19ATHCPVQ4FKYAWWRC39Y0: digest: sha256:795c4e6c6307746bc806be58bd04c163e75119032b10c41394651ca0d95d3d10 size: 953
--> Pushing image done
image: registry.fly.io/deploy-springboot:deployment-01GH19ATHCPVQ4FKYAWWRC39Y0
image size: 378 MB
==> Creating release
--> release v2 created

--> You can detach the terminal anytime without stopping the deployment
```

8. Para revisar la aplicación desplegada deberás dirigirte a <https://fly.io/dashboard>



9. Seleccionar la app, Ejemplo: *deploy-springboot*



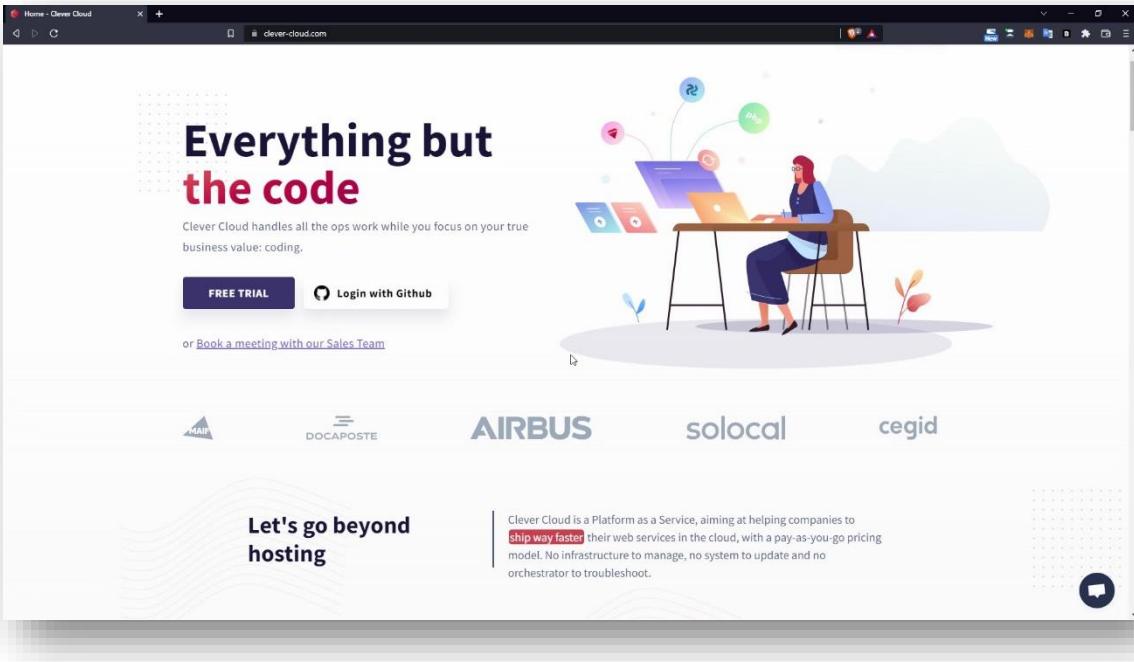
10. Finalmente, verificar el correcto funcionamiento del servidor:



Referencia: <https://fly.io/docs/languages-and-frameworks/dockerfile/>

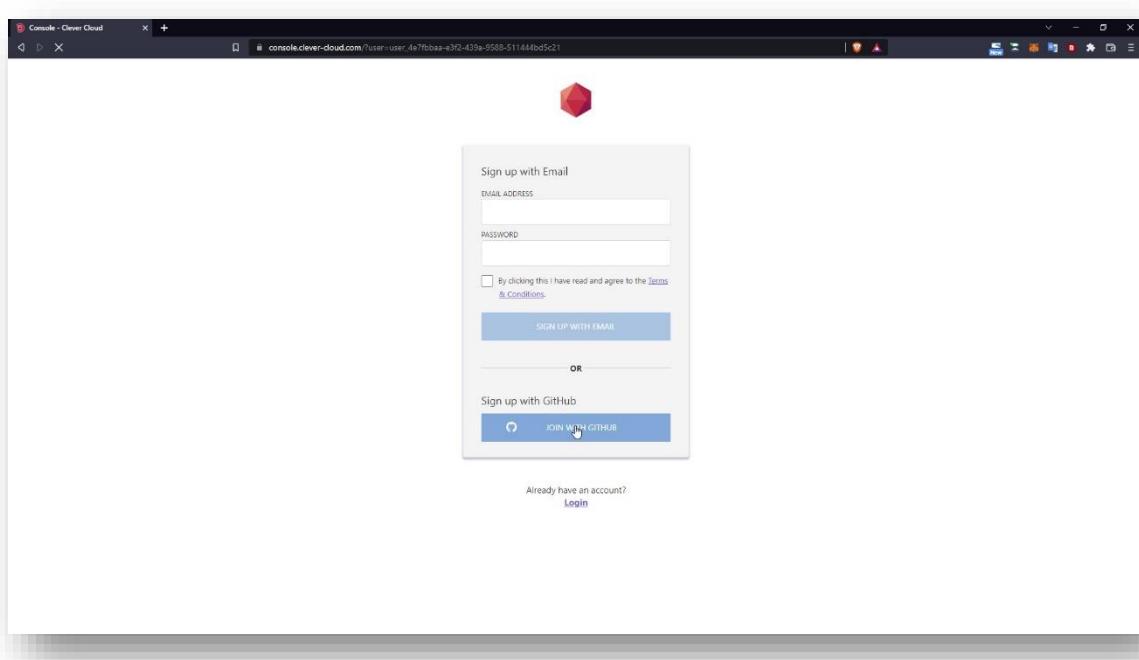
Cómo subir tu base de datos a clever cloud

Ingresa a www.clever-cloud.com

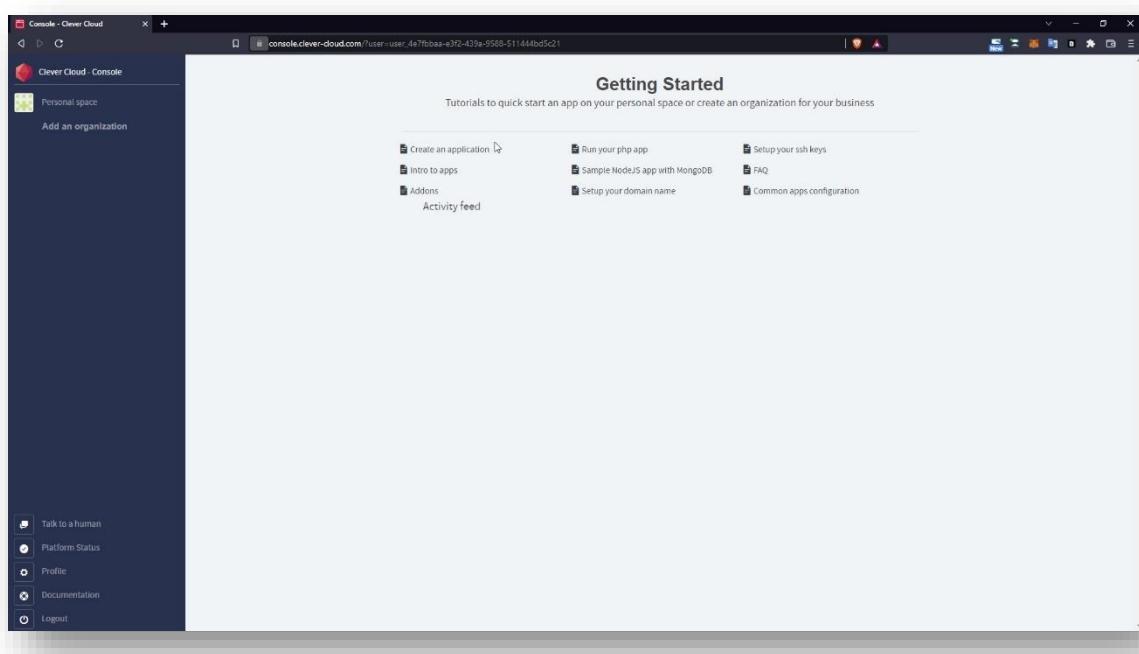


The screenshot shows the homepage of Clever Cloud. The main headline reads "Everything but the code". Below it, a subtext says "Clever Cloud handles all the ops work while you focus on your true business value: coding.". There are two buttons: "FREE TRIAL" and "Login with Github". Below these buttons is the text "or Book a meeting with our Sales Team". On the right side of the page, there's a sidebar with the text "Clever Cloud is a Platform as a Service, aiming at helping companies to ship way faster their web services in the cloud, with a pay-as-you-go pricing model. No infrastructure to manage, no system to update and no orchestrator to troubleshoot." and a small "Ask me a question" button.

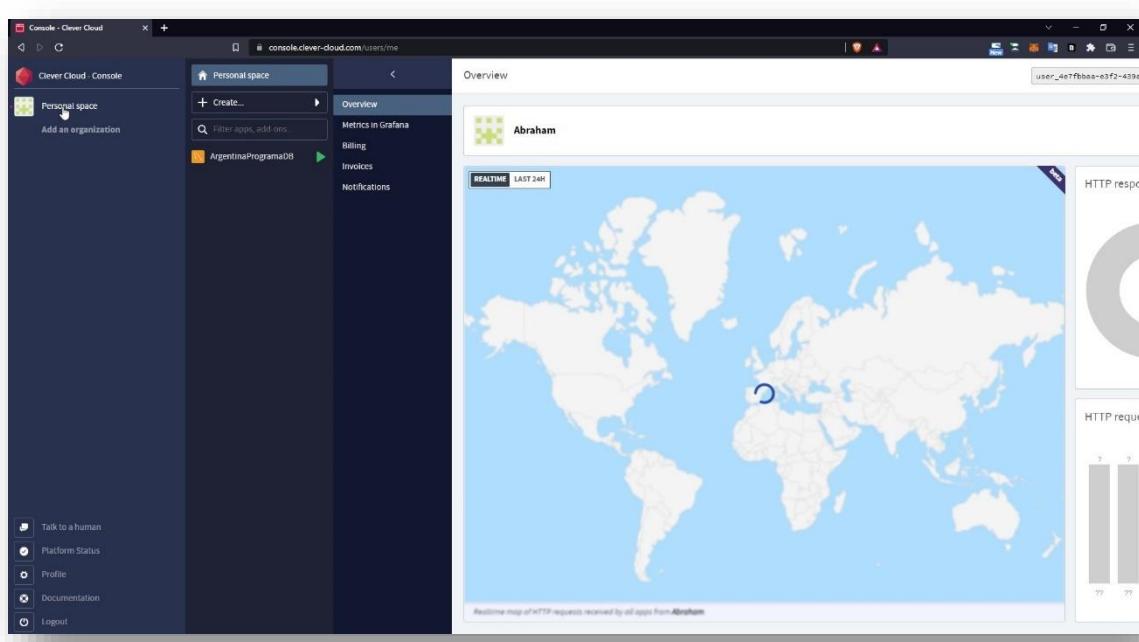
- 1.- Da click en el botón Sign Up para comenzar el registro de tu cuenta (Puedes registrarte con tu email o con tu cuenta de Github)



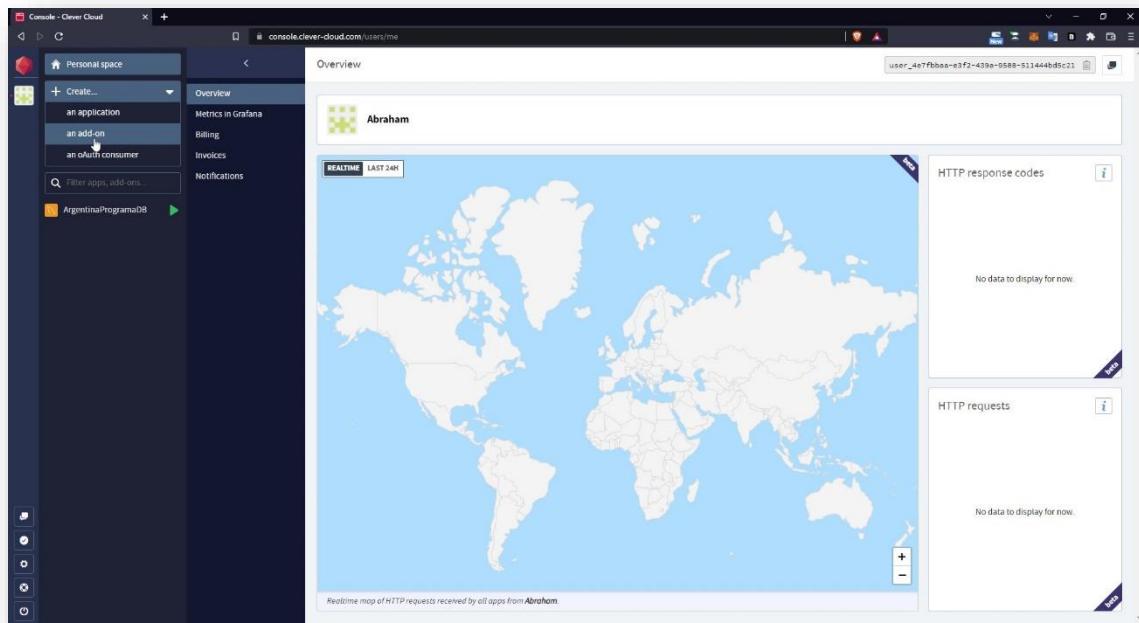
2.- Una vez registrada la cuenta ingresar al dashboard de la plataforma



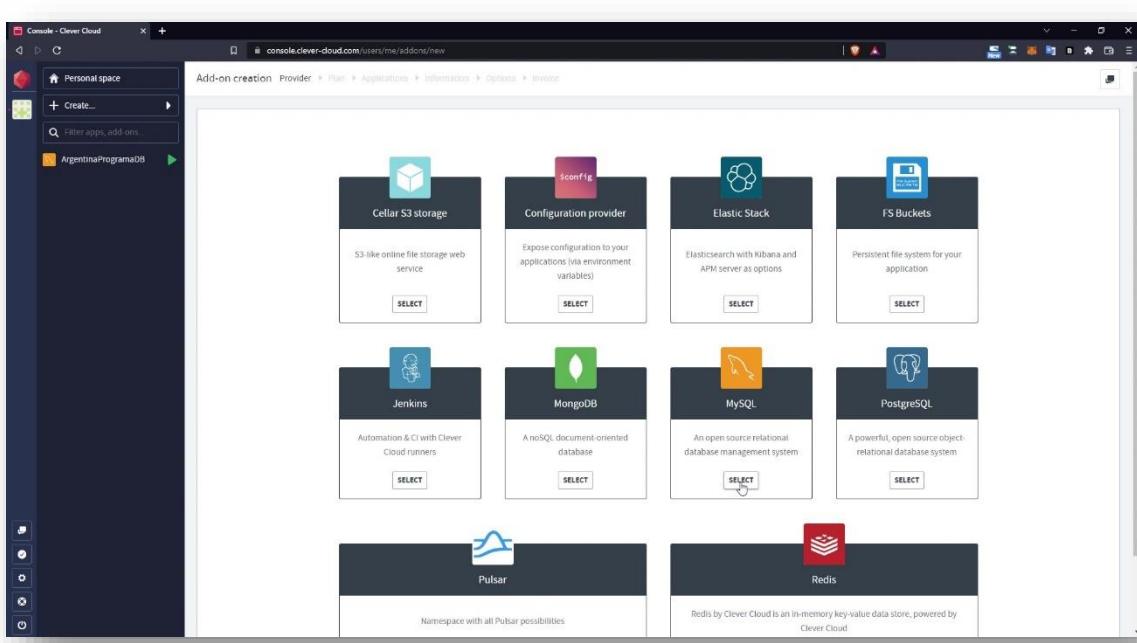
3.- En el panel izquierdo da click sobre el botón personal space para desplegar un segundo menú a la izquierda:



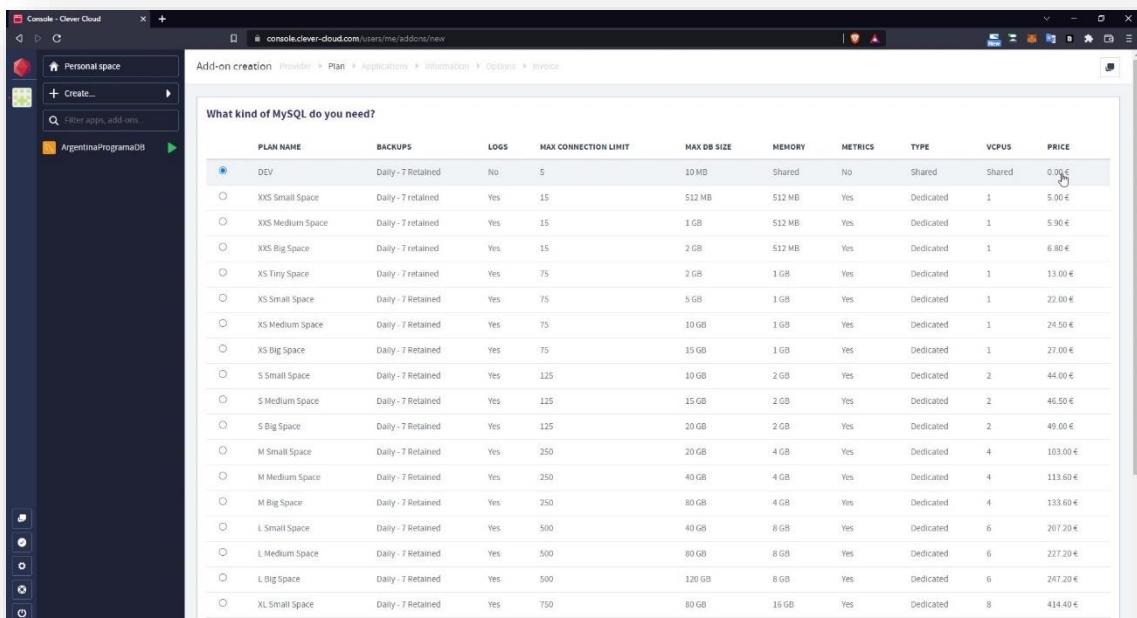
4.- Hacer click sobre el botón Create y luego sobre el botón Add-on



5.- Da click sobre la base de datos que hayas utilizado para tu portfolio web MySQL o PostgreSQL, ten en cuenta que no se permiten bases de datos de otro tipo, y luego hacer click sobre el botón Select:



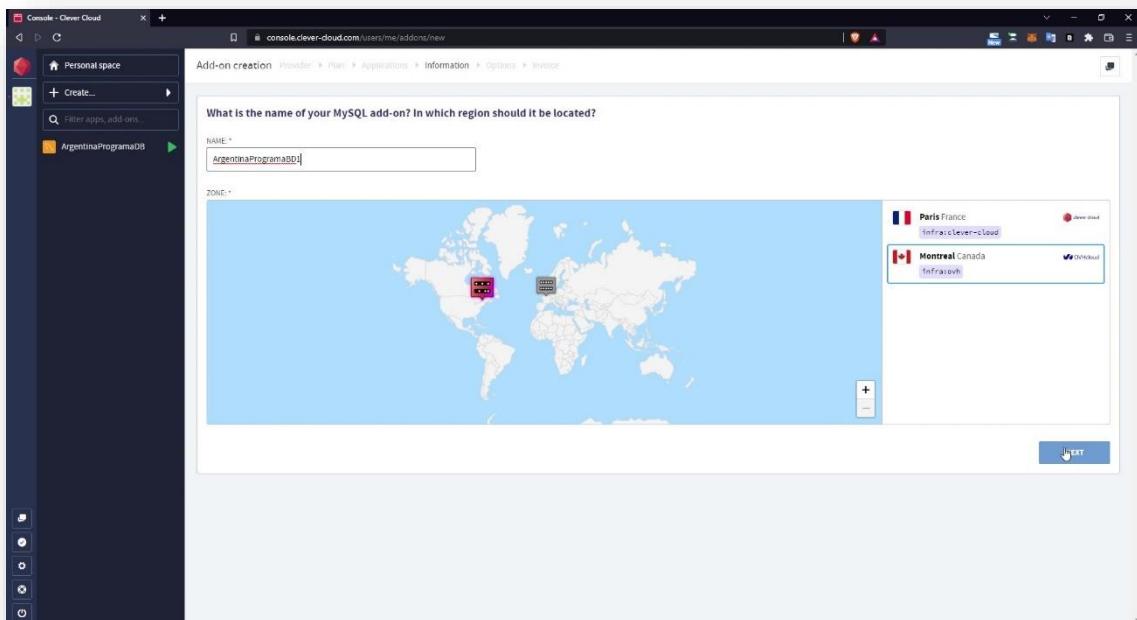
6.- Se desplegará un dashboard para seleccionar el plan de nuestra base de datos, daremos click en la opción del listado DEV y donde el precio sea 0.00 €.



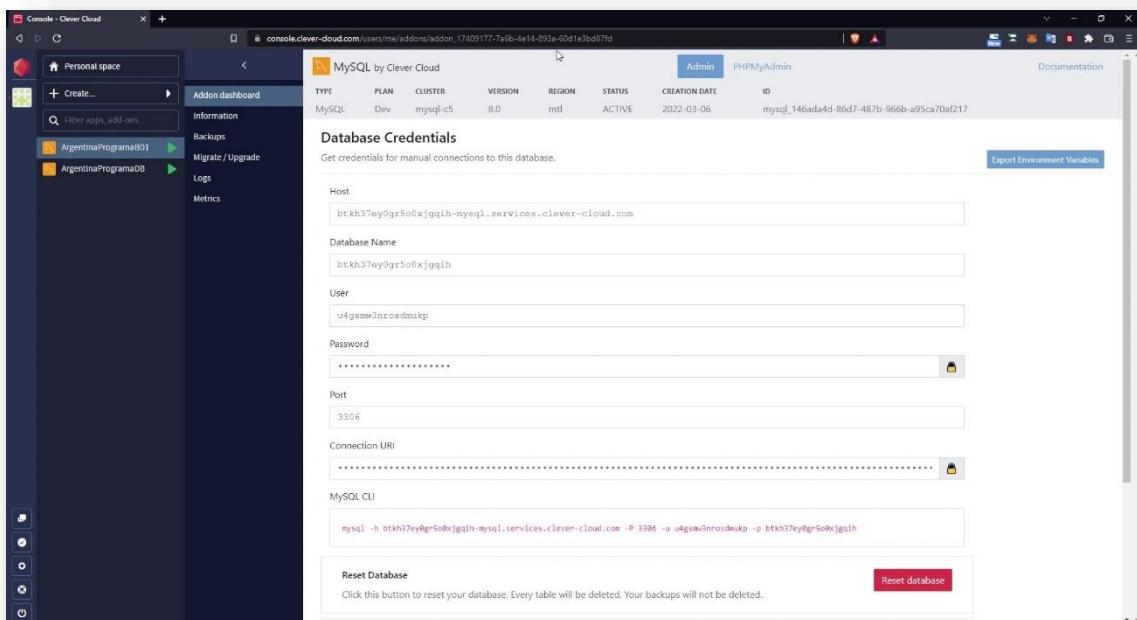
PLAN NAME	BACKUPS	LOGS	MAX CONNECTION LIMIT	MAX DB SIZE	MEMORY	METRICS	TYPE	VCPUS	PRICE
<input checked="" type="radio"/> DEV	Daily - 7 Retained	No	5	10 MB	Shared	No	Shared	Shared	0,00 €
<input type="radio"/> XXS Small Space	Daily - 7 retained	Yes	15	512 MB	512 MB	Yes	Dedicated	1	5,00 €
<input type="radio"/> XXS Medium Space	Daily - 7 retained	Yes	15	1 GB	512 MB	Yes	Dedicated	1	5,90 €
<input type="radio"/> XXS Big Space	Daily - 7 retained	Yes	15	2 GB	512 MB	Yes	Dedicated	1	6,80 €
<input type="radio"/> XS Tiny Space	Daily - 7 retained	Yes	75	2 GB	1 GB	Yes	Dedicated	1	13,00 €
<input type="radio"/> XS Small Space	Daily - 7 Retained	Yes	75	5 GB	1 GB	Yes	Dedicated	1	22,00 €
<input type="radio"/> XS Medium Space	Daily - 7 Retained	Yes	75	10 GB	1 GB	Yes	Dedicated	1	24,50 €
<input type="radio"/> XS Big Space	Daily - 7 Retained	Yes	75	15 GB	1 GB	Yes	Dedicated	1	27,00 €
<input type="radio"/> S Small Space	Daily - 7 Retained	Yes	125	10 GB	2 GB	Yes	Dedicated	2	44,00 €
<input type="radio"/> S Medium Space	Daily - 7 Retained	Yes	125	15 GB	2 GB	Yes	Dedicated	2	46,50 €
<input type="radio"/> S Big Space	Daily - 7 Retained	Yes	125	20 GB	2 GB	Yes	Dedicated	2	49,00 €
<input type="radio"/> M Small Space	Daily - 7 Retained	Yes	250	20 GB	4 GB	Yes	Dedicated	4	103,00 €
<input type="radio"/> M Medium Space	Daily - 7 Retained	Yes	250	40 GB	4 GB	Yes	Dedicated	4	113,60 €
<input type="radio"/> M Big Space	Daily - 7 Retained	Yes	250	80 GB	4 GB	Yes	Dedicated	4	133,60 €
<input type="radio"/> L Small Space	Daily - 7 Retained	Yes	500	40 GB	8 GB	Yes	Dedicated	6	267,20 €
<input type="radio"/> L Medium Space	Daily - 7 Retained	Yes	500	80 GB	8 GB	Yes	Dedicated	6	227,20 €
<input type="radio"/> L Big Space	Daily - 7 Retained	Yes	500	120 GB	8 GB	Yes	Dedicated	6	247,20 €
<input type="radio"/> XL Small Space	Daily - 7 Retained	Yes	750	80 GB	16 GB	Yes	Dedicated	8	414,40 €

7.- El siguiente paso es seleccionar la región del servidor, solo tendremos 2 opciones y siempre debemos seleccionar la región que sea más cercana a nosotros, en este caso seleccionaremos la región de Canadá. Además de eso

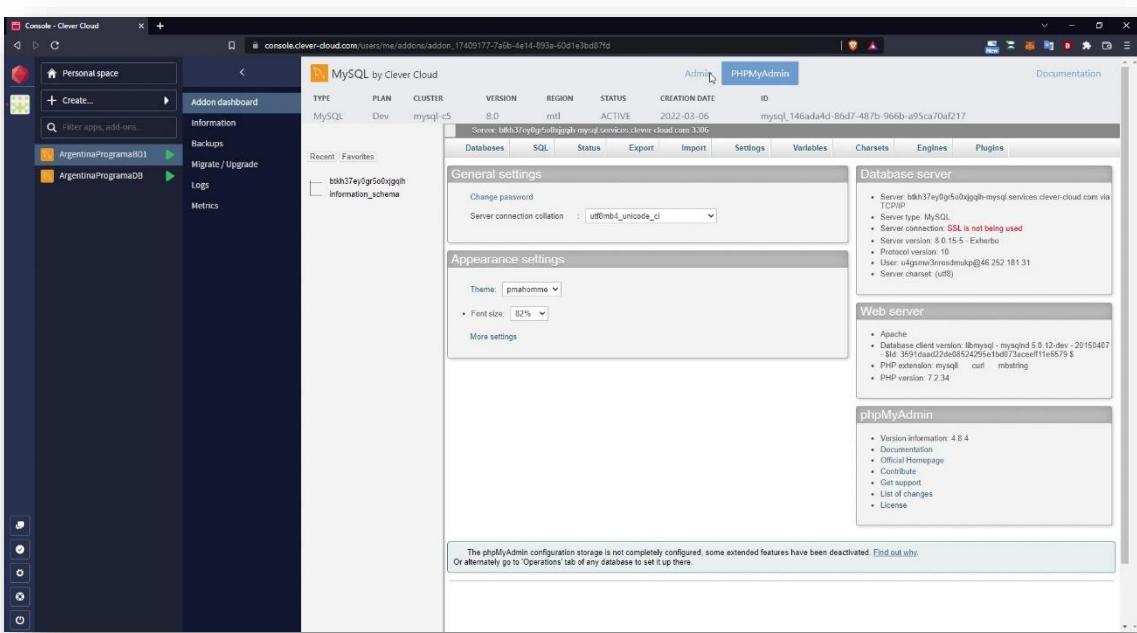
debemos nombrar nuestro Add-on, utiliza un nombre de acuerdo a la normativa y fácil de recordar. Para finalizar hacer click sobre el botón Next.



8.- Se desplegará la información para conectar y usar tu base de datos



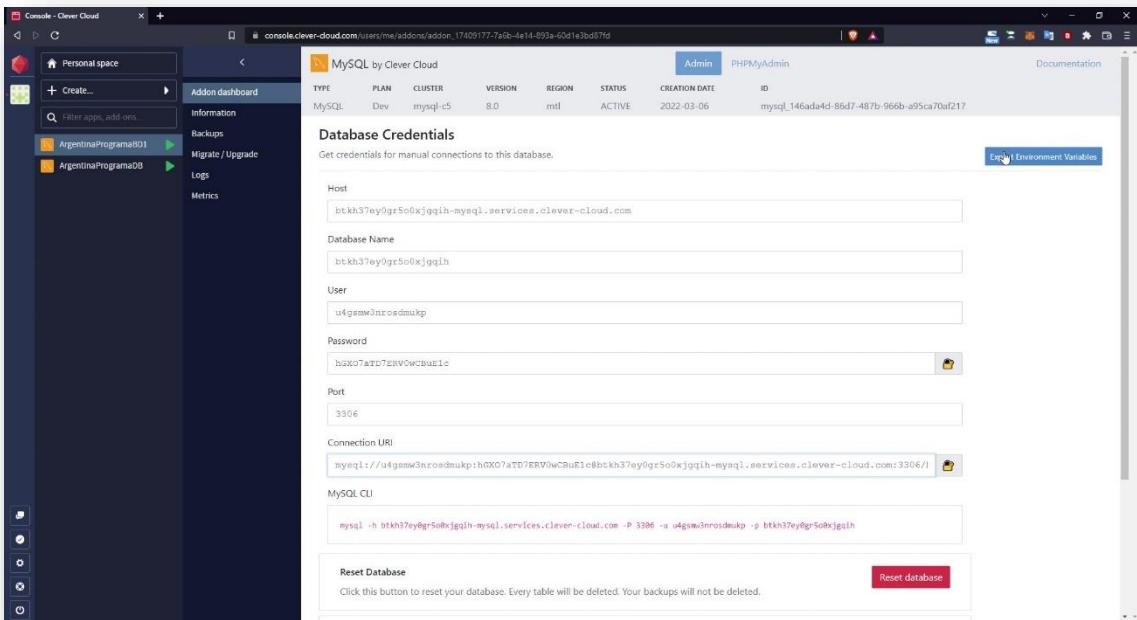
9.- Puedes administrar tu base de datos via CLI o consola, añadiendo el servidor desde Workbench o desde su sección de PHPMyAdmin



The screenshot shows the Clever Cloud MySQL by Clever Cloud dashboard. On the left, there's a sidebar with options like Personal space, Create..., Addon dashboard, Information, Backups, Migrate / Upgrade, Logs, and Metrics. The main area is titled "MySQL by Clever Cloud" and shows a database named "ArgentinaProgramaDB1". The "General settings" section includes fields for "Change password" and "Server connection collation" set to "utf8mb4_unicode_ci". The "Database server" section provides detailed information about the MySQL instance, including its ID, type (MySQL), plan (Dev), cluster (mysql-c5), version (8.0), region (mtl), status (ACTIVE), creation date (2022-03-06), and connection details. The "Web server" section lists Apache, MySQL, and PHP versions. The "phpMyAdmin" section shows version information and links to documentation and reports. A note at the bottom states: "The phpMyAdmin configuration storage is not completely configured, some extended features have been deactivated. Find out why. Or alternately go to 'Operations' tab of any database to set it up there."

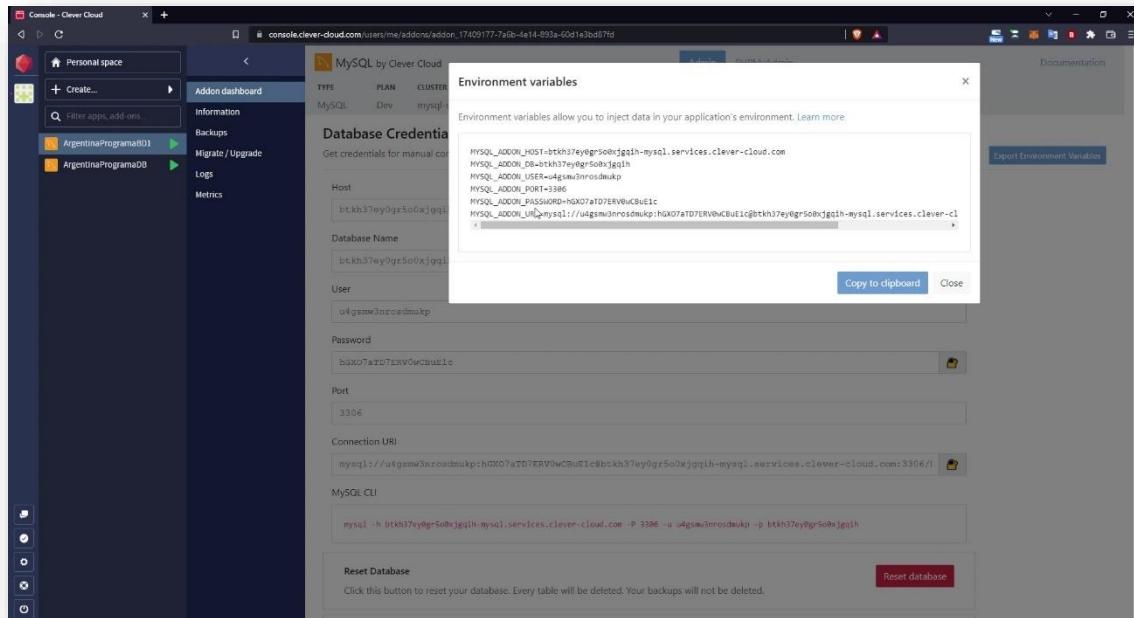
Coneectar base de datos con Spring Boot

Daremos click en el botón “Export Environments Variables”

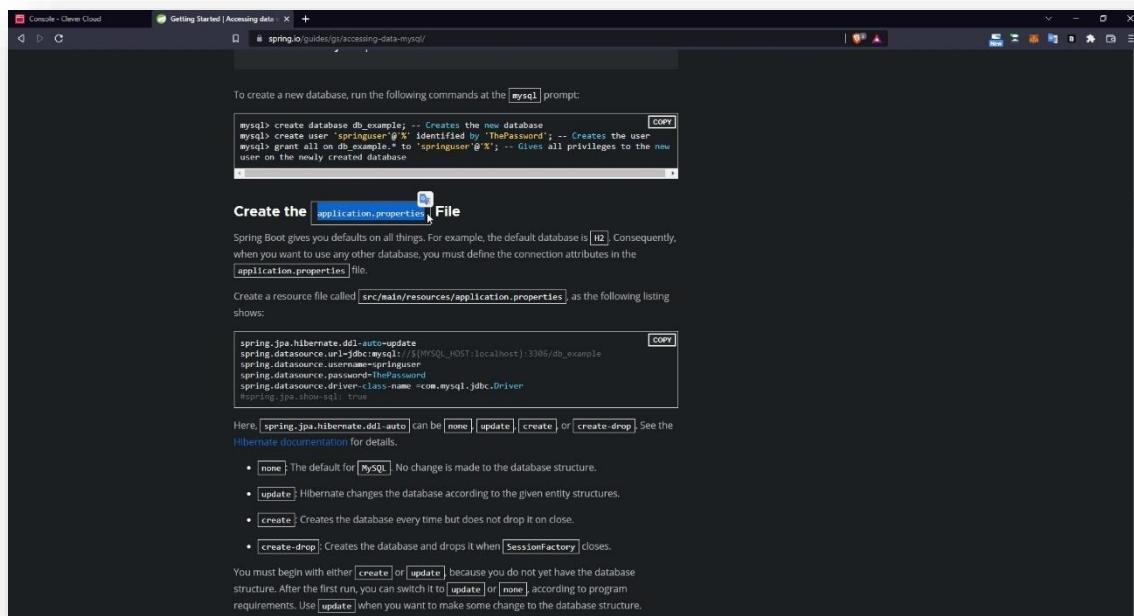


The screenshot shows the Clever Cloud MySQL by Clever Cloud dashboard. The "Database Credentials" section is displayed, containing fields for Host, Database Name, User, Password, Port, Connection URI, and MySQL CLI. A prominent blue button labeled "Export Environment Variables" is visible. Below the connection fields, there's a "Reset Database" button with a warning message: "Click this button to reset your database. Every table will be deleted. Your backups will not be deleted." A "Reset database" button is also located at the bottom right of this section.

Se desplegará un modal con la información de nuestra conexión a la base de datos



Utilizaremos esta información para conectarlo a nuestro Spring Boot de la siguiente manera:



Listo!, ahora tienes tu base de datos lista para usar con tu sistema Back end.

Introducción – Comunicación

En esta sección avanzaremos en el concepto de “comunicación efectiva”, su importancia y su aplicación en la vida diaria y en el ambiente laboral. En lo particular, y dado que uno de los pilares de scrum es justamente la comunicación, profundizaremos en los conceptos y técnicas que nos permitan adquirir habilidades de comunicación interpersonal que nos posibiliten comunicarnos de forma eficiente en equipos de desarrollo de software.

Objetivos:

1. Comprender la importancia de la comunicación interpersonal en un equipo de desarrollo de software.
2. Desarrollar capacidades de comunicación asertiva y empática.
3. Desarrollar capacidades de retroalimentación que habiliten un ambiente de diálogo activo y respetuoso.

Comunicación

Antes de comenzar a abordar los conceptos propios de la Comunicación Efectiva, debemos comprender primero ¿Qué es la comunicación?

Si partimos de lo cotidiano, nos comunicamos todos los días y todo el tiempo, cuando damos un reconocimiento, cuando expresamos nuestras ideas y opiniones, cuando necesitamos resolver un conflicto, cuando necesitamos organizarnos en nuestros hogares o en el ambiente laboral, para negociar, para liderar, etc.

Entonces, podemos concluir que la **comunicación es una acción que realizamos las personas para transmitir y/o recibir mensajes y que tienen un propósito**. Los gestos, los silencios, los sueños, los olvidos y actos fallidos, lo pensado y no dicho, también son constituyen actos comunicacionales de los sujetos.

Nuestra característica central como seres humanos es la posibilidad de comunicarnos, de diferentes formas y por diversos canales. La Comunicación es el sustento de la interacción social a través de la cual integramos al Otro en nuestro mundo y en la satisfacción de nuestras necesidades, tanto primarias y como sociales.

Si vamos a la definición de la palabra comunicación, significa “común ubicación”, lo que nos permite identificar dos puntos de encuentro entre dos o más personas como si de un puente se tratara. Un puente que une dos puntos. Siendo el propósito del puente establecer un medio que permita la unión entre esos dos puntos. Un medio para transmitir una idea, una opinión, un problema, etc. a fin de avanzar hacia una solución. Es decir, necesitamos un medio para transmitir y/o recibir mensajes.

La comunicación es una herramienta esencial necesaria no sólo para la vida cotidiana, sino también para organizar las acciones de las personas en una

organización o equipo de trabajo. Y si pensamos en el ambiente laboral, podemos concluir que la comunicación constituye la espina dorsal de muchos procesos organizativos y en especial, de aquellos trabajos en equipo que se desarrollan en un entorno laboral donde el resultado está ligado al conjunto de acciones que llevan a cabo varias personas. Aquel equipo que no logra una buena comunicación y sinergia dudosamente alcance el éxito.

Y ¿por qué hablamos de equipos? y no de grupos de personas... porque en los equipos las personas se organizan colaborativamente para conseguir un objetivo común. Ej: en un equipo de fútbol cada integrante domina ciertas habilidades (quizás diferentes entre los integrantes del equipo) y, además trabaja cohesivamente con el objeto de perseguir un objetivo común, ganar el partido.

Proceso de Comunicación

Podemos identificar los siguientes elementos del proceso de comunicación:

1. **El emisor.** Es quien transmite el mensaje utilizando su propio lenguaje, sus condiciones socioculturales y sentimientos y emociones.
2. **El receptor.** Es quien recibe el mensaje. Pero lo recibe a través de filtros, que pueden estar dados por el propio lenguaje como así también sus condiciones socioculturales y estados de ánimo (se pone en juego su subjetividad, competencia y cosmovisión del mundo). Emisor y receptor pueden alternarse en dichos roles, y hacen posible un proceso comunicacional.
3. **El mensaje.** Es el conjunto de símbolos que pretenden transmitir información, sentimientos, valores, necesidades, etc. Está construido por un significante (lo que percibimos) y un significado (lo que interpretamos) y para él se pueden utilizar palabras verbales, escritas, gestos, signos y símbolos de diverso tipo.
4. **El canal.** Es el instrumento mediante el cual se transmite el mensaje (el medio por donde transita el mensaje). Son múltiples los medios de comunicación, soportes y formatos y su tipo incide decididamente en el mensaje a transmitir y en sus modos de recepción.
5. **El contexto,** la situación personal y general en las que están inmersos quienes interactúan. Es el aquí y ahora en el cual se da el intercambio comunicacional y que tiene mucha relevancia al momento de gestar un proyecto, reunión o hacer un mero intercambio de información. Lo que sucede en el entorno influye directa o indirectamente en el propósito que nos convoca.
6. **El feedback o retroalimentación.** Es el instrumento mediante el cual se puede indagar para evaluar si el mensaje llegó a su destino y fue decodificado correctamente. Permite establecer el diálogo para llegar a acuerdos.
7. **Interferencias,** también llamados ruidos y barreras, son los factores que intervienen como obstaculizadores de la Comunicación, a los que se debe identificar (si es posible de antemano) e intentar anular, disminuir o disipar con estrategias y herramientas diseñadas para tal fin.

Propósito de la comunicación

"El mayor problema de comunicación es la ilusión de que ha tenido lugar"
(George Bernard Shaw)

Para comprender este concepto, les proponemos analizar el dilema de la naranja:
El dilema habla de dos personas que se disputan una misma naranja. ¿Cómo lo resuelven?

Este es sin duda un muy conocido ejemplo de negociación donde se analiza la importancia de la comunicación para obtener un resultado del tipo ganar-ganar. Por ejemplo, la respuesta más fácil es dividir la naranja en dos y que cada uno se lleve el 50% o, en el peor de los casos, que uno se lleve el 100% por lo que el otro no obtendrá nada (ganar-perder). Entonces la pregunta es: ¿es posible que ambas personas se lleven el 100% de la naranja? Es decir, un ganar-ganar.

La respuesta es "puede ser". Todo dependerá de la comunicación entre esas dos personas y de sus habilidades para detectar los intereses del otro, quizás una de las personas sólo quiere el jugo mientras la otra sólo la ralladura. En ese caso ambos podrían llevarse el 100% (su 100% de la naranja).

En base a lo analizado en el video y, el dilema de la naranja podemos concluir que **el propósito de la comunicación no sólo es conectar, transmitir ideas, opiniones, etc. sino también, llegar a acuerdos que nos permitan resolver problemas o conflictos.**

Finalmente, podemos enunciar otros propósitos de la comunicación no menos importantes:

1. Informar
2. Promover acciones
3. Motivar y persuadir
4. Desarrollar y mantener relaciones
5. Negociar

No olvidemos algo importante: Debemos desmitificar que una Comunicación Efectiva supone la posibilidad de ser 100% claros en lo que se pretende transmitir o, lo que es lo mismo, pretender que quienes sean los destinatarios comprendan la información suministrada en exactamente los mismos términos que quienes construyeron el mensaje.

La Comunicación efectiva pone el foco en mensajes claros, precisos y concisos que sean simples de interpretar, que permitan identificar fácilmente a lo que se quiere apuntar evitando rodeos o derivaciones en otros temas que puedan llevar a la dispersión, la confusión o a comprensiones erróneas o muy alejadas de la finalidad original.

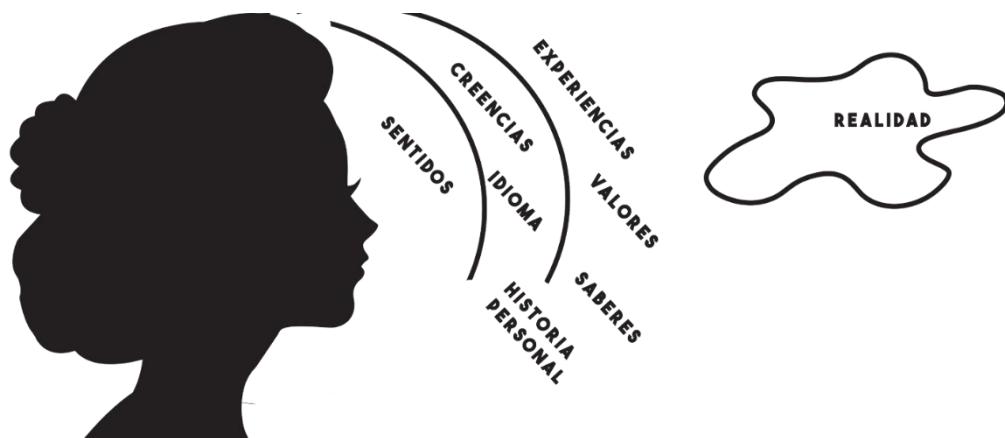
Comunicación (Barreras, Filtros y Ruido)

Barreras de la comunicación

Existen barreras en el proceso de comunicación que pueden dificultar la transmisión del mensaje entre el emisor y el receptor. Entre ellas, podemos nombrar: la escucha selectiva, los filtros, dificultades semánticas, la cantidad de información, la posición jerárquica del emisor, entre otras.

Filtros

De las enunciadas anteriormente, los filtros son sin dudas muy importantes de interpretar. Filtros que todas las personas tenemos y están ligados a la forma en que cada uno de nosotros vemos el mundo. Las percepciones.

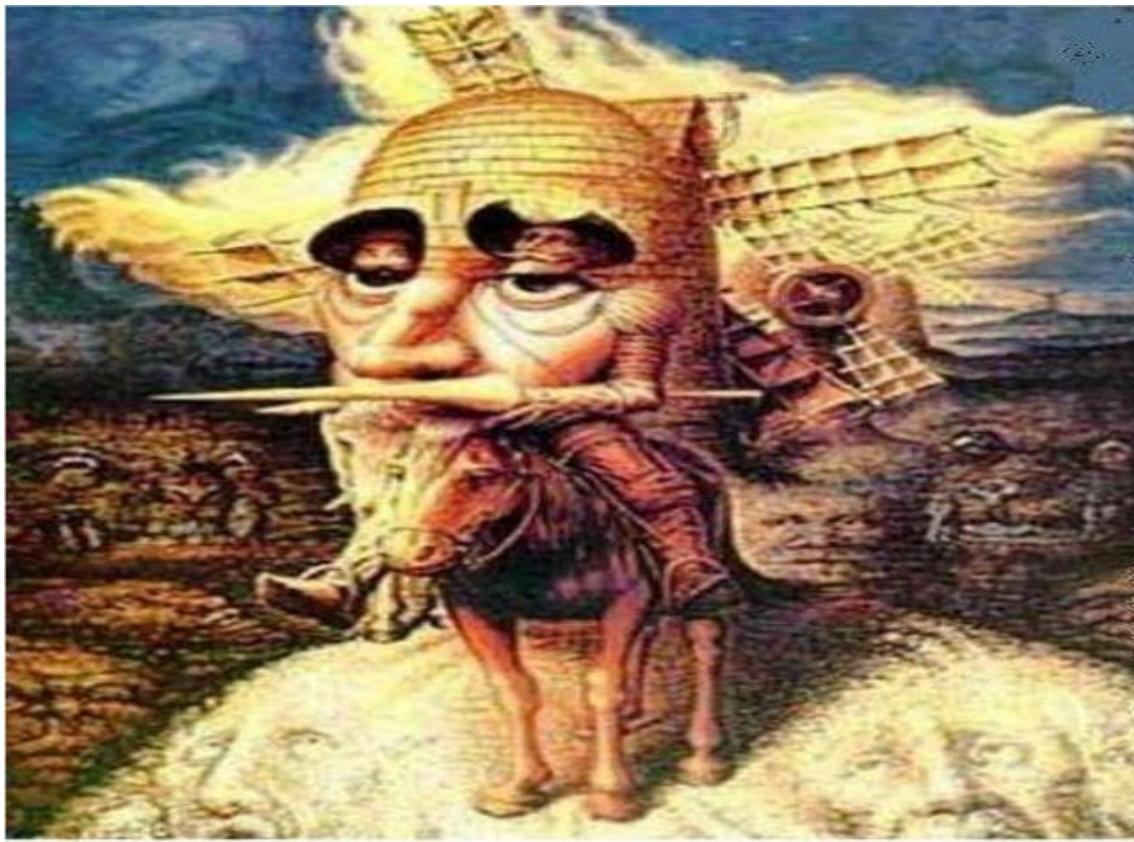


De la imagen demostrada arriba podemos observar que el mundo existe tal cual es (*la realidad*) pero las personas, individualmente, ven esa realidad a través de filtros que pueden estar dados por la historia personal, los valores, las creencias, los saberes o conocimientos, los sentidos, etc.

Dichos filtros se pueden clasificar en:

1. **Neurológicos**, tienen que ver con los sentidos.
2. **Sociales**, tienen que ver con el idioma y las convenciones socialmente aceptadas.
3. **Individuales**, tienen que ver con la historia personal, experiencias, creencias, valores, puntos de vistas, saberes, etc.

Para comprenderlo mejor, observemos la siguiente imagen:



Seguramente, quienes tengan conocimiento sobre la novela de Cervantes “El ingenioso caballero don Quijote de la Mancha” podrán asociar la imagen con la historia narrada en la novela, sus personajes y el contexto en el que se desarrolla la misma pero, quienes no tengan este conocimiento, podrán observar quizás dos caballeros, un molino y “demonios”, otros hasta quizás podrán asociar a Albert Einstein dado que también puede observarse un rostro con los “pelos al viento” con la suma de todas las imágenes, entre muchísimas más.

Podemos entonces concluir, que la realidad existe tal cual es, en este caso la obra de Salvador Dalí pero nosotros, no la vemos tal cual es sino que, todo dependerá de cómo la percibimos. Y esa percepción tiene que ver justamente con nuestros propios filtros.

Y... ¿Qué tiene que ver esto con la comunicación? Justamente, las percepciones también se aplican al proceso de comunicación. **El emisor transmite un mensaje, pero, el receptor lo percibe con sus propios filtros por lo que, puede decodificar el mensaje no del todo tal cual como era la intención del emisor.** ¿No les ha pasado estar hablando con alguien y esta persona, se enojó o nos miró raro? O ¿escribirle un mensaje de whatsapp a alguien y éste lo malinterpretó? Justamente, es por estos filtros que a veces, el mensaje no es decodificado tal cual esperamos. Por ello, debemos prestar atención y entender que no todos vemos la realidad de la misma manera.

Ruidos en la Comunicación

No debemos olvidarnos tampoco de los “ruidos” en el proceso de comunicación. Ruidos que no permiten “la escucha” haciendo que la información que intenta transmitir el emisor no llegue al receptor con claridad.

Podemos enunciar dos ejemplos de ruidos:

1. **Mal uso del canal o medio de comunicación.** Ej. si estoy en una conferencia virtual y, el micrófono no funciona bien y/o hay ruido ambiental.
2. **Emociones fuertes.** Ej. si el receptor está enojado difícilmente escuchará lo que el emisor intenta comunicar.

Comunicación efectiva interpersonal - Comunicación Asertiva

Comunicación Efectiva

Una comunicación efectiva es aquella que tiene por finalidad poder cumplir con determinados objetivos a partir de la puesta en juego de un conjunto de competencias personales y grupales. Implica un proceso que debe ser planificado conscientemente y que debe tener en cuenta no solo la meta a la que se desea llegar sino a quiénes serán los interlocutores, así como los recursos con los que se cuenta y el contexto en el que se desplegará la acción.

Tal como señalamos, en todo proceso comunicacional hay una intencionalidad, un horizonte a alcanzar. La intencionalidad es la dirección que toma nuestro accionar, es el hacia dónde se dirige nuestra conducta. Y es la Comunicación lo que permite la interacción entre los sujetos, la posibilidad de intercambiar mensajes, información, datos, ideas, emociones y pensamientos. En este sentido, cuanto más podamos plasmar en una hoja de ruta la dirección comunicacional de nuestros actos, más cerca estaremos de conseguir una Comunicación Efectiva.

Podemos decir que en primera instancia un comunicador efectivo debe en primer lugar, identificar los elementos de la comunicación, segundo tomarse el tiempo para prepararse, y por último identificar, desde qué nivel de comunicación estamos transmitiendo nuestro mensaje e identificar desde qué nivel de comunicación nos está transmitiendo la otra persona para así estar en “sintonía”.

¿Pero con esto alcanza? Sigamos analizando ...

¿Qué hace un comunicador efectivo?

Un comunicador efectivo debe, además, desarrollar las siguientes habilidades de comunicación interpersonal:

1. **Comunicación Asertiva.** Ser claro, respetuoso y preciso al momento de transmitir el mensaje.
2. **Escucha empática.** Permite la comprensión emocional de la otra persona lo que ayuda a generar un ambiente adecuado para el diálogo.

3. **Retroalimentación (feedback).** Permite cerrar el ciclo emisor-receptor y de esta manera mantener un diálogo activo para llegar a acuerdos de manera respetuosa.

Comunicación Asertiva

Antes de avanzar en el concepto, es interesante que analicemos primero las formas en que nos comunicamos con los demás. Es decir, los grados de asertividad.

1. **No Asertividad:** Evasión. Incapacidad de ser claros y directos al momento de expresar el mensaje o hacerlo sin intensidad y claridad.
2. **Asertividad:** Capacidad de ser claro, preciso y respetuoso al momento de transmitir el mensaje.
3. **Agresión:** Puede ser encubierta o abierta. Consiste en expresar mensajes con un tono de voz elevado, así como el uso de miradas y gestos amenazantes.

La palabra asertividad significa “acertar”, dar en el blanco. Significa ser claro, preciso y respetuoso al momento de transmitir el mensaje. Sin embargo, no depende de nosotros sino de la persona quién la recibe, el receptor. Es decir, podemos pensar que somos claros y precisos, pero dependerá de la otra persona si recibe el mensaje de esta forma. Por ello es importante tomarnos el tiempo acerca de la forma y el medio que utilizaremos para transmitir el mensaje.

CONTENIDO + FORMA + MEDIO = ESTRATEGIA DE COMUNICACIÓN INTERPERSONAL

La planeación consiste entonces en tomarnos un tiempo para reflexionar en:

1. **El contenido del mensaje** debe ser del tipo “yo”, claro, preciso y respetuoso.
2. **La forma en que transmitiremos el mensaje**, para ello debemos evaluar el marco de referencia del receptor. Es decir, su forma de pensar, sus intereses, sus respuestas previas a nuestra comunicación, etc.
3. **El medio de comunicación** consiste en evaluar el medio de comunicación más adecuado. El mismo depende de la estrategia de comunicación, del tipo de mensaje, del tiempo y los recursos disponibles, del marco de referencia del receptor, etc.

El mensaje

El mensaje asertivo además de ser claro, preciso y respetuoso debe ser del tipo “**mensajes yo**” a fin de expresarnos desde lo que pensamos, lo mío.

Esta técnica, permite establecer un ambiente de respeto y escucha (porque tenemos que estar abiertos a escuchar) a fin de dar lugar al diálogo activo a miras

de resolver un conflicto. Para comprenderlo mejor, analicemos los siguientes mensajes:

1. "esta tabla en la base de datos está mal hecha"
2. "la tabla clientes que subiste ayer tiene dos columnas repetidas".
3. "en la tabla clientes hay dos columnas repetidas con distinto nombre lo que provoca redundancia en los datos. Me gustaría revisar esto con vos"

Los dos primeros mensajes son del tipo "**mensaje tú**" lo que puede provocar una reacción de defensa por parte del receptor, además, de no ser del todo claras en el mensaje. Sin embargo, el último ejemplo, no sólo es claro (permite identificar claramente el problema) sino que también libera el estrés y da firmeza.

El guión DEEC

El guión DEEC es una técnica que nos permite crear guiones asertivos permitiéndonos pensar antes de actuar. La misma consiste en:

D - Describir el hecho

E - Expresar lo que pensamos y sentimos

E - Explicar la conducta deseada

C - Consecuencias del hecho

Situación: José llega tarde por segunda vez al Scrum Retrospective Meeting.

Buenos días José, el día de hoy llegaste por segunda vez tarde a la Retrospective.
D

(describo el hecho)

Esta situación me preocupa porque esta reunión es muy importante para evaluar el mi trabajo del equipo.

(expreso lo que pienso y siento)

José, es muy importante tu asistencia puntual, te pido que la próxima Retrospectiva mi llegues a tiempo.

(explico la conducta deseada)

De no hacerlo así, no tendremos en cuenta tus aportes por lo que no podremos tomar decisiones conjuntas.

C
(explico las consecuencias)

La forma

Para identificar la mejor forma de transmitir un mensaje debemos identificar el marco de referencia del receptor. Para ello debemos:

1. **Investigar**, lo que implica un esfuerzo de nuestra parte preguntar, leer material escrito como por ej. certificados de desempeño, cv, etc., hablar con referentes sobre la persona con quién vamos a comunicarnos.
2. **Escuchar**, si hacemos una escucha activa y empática podremos conocer muchos aspectos del marco de referencia de nuestro interlocutor.
3. **Prestar atención al lenguaje no verbal**, sus gestos, ademanes, reacciones, etc.

El medio

Para identificar el medio de comunicación más adecuado en función del mensaje, el tiempo, la forma y los recursos disponibles observemos el siguiente cuadro:

Medio de Comunicación		Ventajas	Desventajas
Oral o verbal	Teléfono		
	Reunión	<ul style="list-style-type: none"> • Compromiso • Sinergia • Calidez • Feedback inmediato 	<ul style="list-style-type: none"> • Consumo tiempo • Exige habilidades interpersonales de comunicación • No siempre se registra lo conversado.
	Entrevista		
	Entre otros		
	Videoconferencias		
Escrito	Personal (nota, memo, fax, mail, etc.)	<ul style="list-style-type: none"> • Registro y archivo • Precisión • Permite "relectura" • Puede "estandarizarse" 	<ul style="list-style-type: none"> • Feedback diferido y escaso • Distancia -frio • Requiere habilidades del emisor • Requiere habilidades de comprensión de textos del receptor. • Momento de recepción del mensaje no claramente definido.
	Grupal (circular)		
	Cartelera		
	Entre otros		
Audiovisual	Diapositivas (pueden contener audios, imágenes, video, texto...)	<ul style="list-style-type: none"> • Canales múltiples • Acceso a la distancia • Permite la repetición 	<ul style="list-style-type: none"> • Tiene un costo de producción • Exige habilidades técnicas • Exige recursos
	Gráficos		
	Entre otros		

Otra noción íntimamente emparentada con la Comunicación Efectiva es la Comunicación Eficaz, si bien muchas veces se usa como sinónimo, la segunda remite al cumplimiento de los objetivos estipulados, para lo cual es necesario confirmar que aquello que se intentó comunicar fue comprendido. Si se evalúa que la interacción ha sido satisfactoria y que los resultados finalmente se lograron, entonces se podrá afirmar que la Comunicación además de Efectiva fue Eficaz.

Escucha Empática

Antes de comenzar a definir el concepto, veamos el fragmento de la película "[Patch Adams](#)" dirigido por Tom Shadyac y a partir de ella, intentemos de responder la pregunta ¿Qué es la escucha ?.

En la escena, podemos observar a *Patch Adams* transmitiendo al psicólogo su historia de vida desde un nivel de comunicación íntimo, pero, el psicólogo ¿lo está escuchando realmente? Podríamos decir que no ¿verdad?

Entonces, escuchar no significa sólo oír, sino que, además significa ser capaz de interpretar lo que se nos está diciendo.

En resumen, escuchar es el proceso de oír, entender, observar y recordar lo que otras personas nos dicen.

Oír + interpretar = escuchar

La escucha empática estimula la comunicación dado que, cuando percibimos que se nos escucha, se nos presta atención, solemos expresar más cosas y con mayor profundidad y detalle que si se nos escucha superficialmente o por compromiso y tal como se puede ver en el fragmento de *Patch Adams*, uno de los errores más frecuentes en la comunicación es no prestar atención a lo que se nos dice y al estado emocional de la otra persona. Esto puede en algunos casos dañar la relación interpersonal.

Antes de continuar, proponemos realizar el [test de capacidad de escucha](#) de Psicología en Positivo a fin de evaluar nuestra propia capacidad de escucha:

1. Escucho sin interrumpir... y menos contradecir.
2. Escucho prestando atención.
3. Escucho más allá de las palabras.
4. Escucho incentivando al otro a profundizar.

Existen distintos tipos de escucha, como se observa en el texto siguiente:

1. **Ignorado:** NO hay escucha
2. **Selectiva:** Sólo Ciertas partes
3. **Atenta:** Con la intención de responder.
4. **Empática:** Comprender lo que el otro siente.

Sin dudas, la mejor será la escucha empática puesto que permite comprender a la otra persona tanto en lo emocional como en lo intelectual respetando sus

respuestas emocionales y sin emitir juicios de valor o censura a dichas emociones. Esto no significa estar de acuerdo con la otra persona, sino escucharla atentamente lo que da lugar a un ambiente de respeto activo donde pueda darse cuenta del diálogo a fin de llegar a acuerdos.

En resumen:

Escucha empática = igualar o empatar

Hay que escuchar los hechos + y prestar atención a los sentimientos. Las personas expresamos con ambos elementos. La respuesta, también debe contemplar estos elementos.

Errores que impiden la empatía

A continuación, compartimos una imagen que resumen puntos que impiden una escucha empática:



“Cada vez que pensamos que somos superiores al otro sobre la base de sexo, raza, carga, edad, experiencia (...),

Cada vez que sostenemos tener un acceso privilegiado a la verdad y justicia.

Cada vez que presumimos que nuestra particular manera de ser, es la mejor.

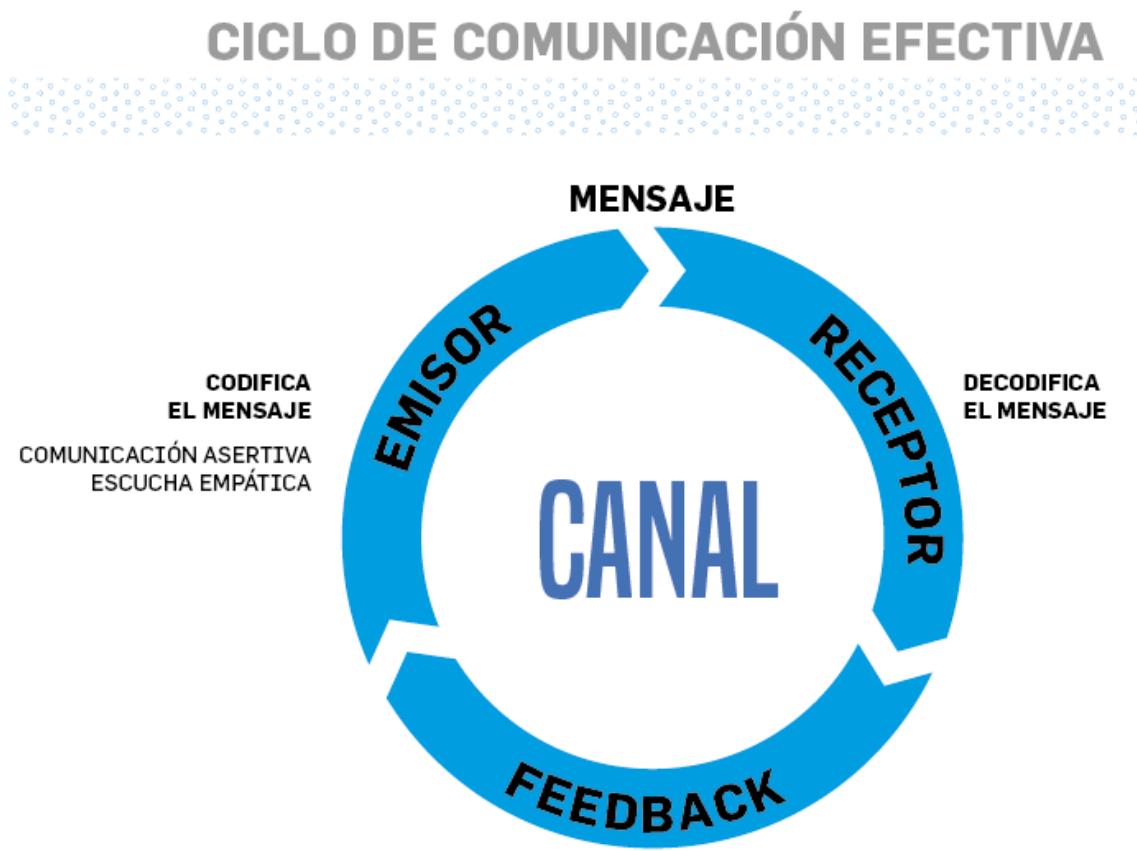
Cada vez que nos olvidamos que somos sólo un observador particular entre infinitas posibilidades...

Cada una de esas veces...

Nuestro escuchar se resiente ” Rafael Echeverría

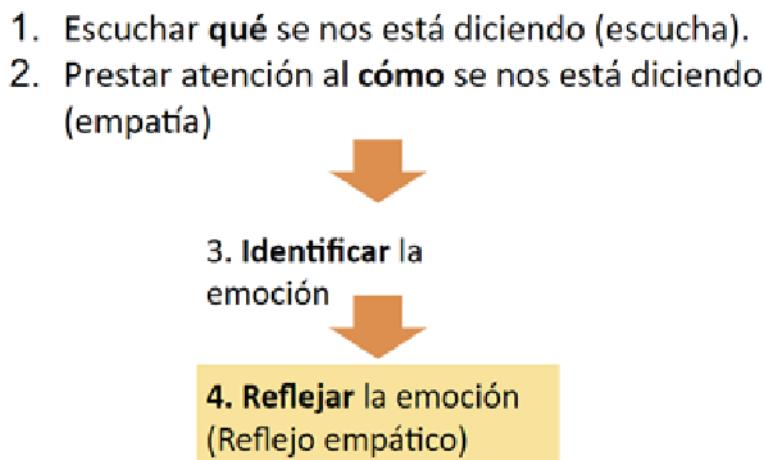
Niveles de Profundidad o Escucha

En base a lo visto anteriormente, podemos definir los niveles de profundidad de la escucha que son: oír, escuchar (interpretar) y empatizar.



Técnicas de escucha empática

En base a los niveles de profundidad podemos enunciar la siguiente técnica de escucha empática:



Como podemos ver, el cuarto punto es “Reflejar la emoción” ¿Qué significa el reflejo empático? y ¿por qué es importante?

El reflejo empático consiste en expresar oportunamente pequeñas frases que le hacen saber al emisor que lo estamos escuchando con atención y comprensión cuidando además la comunicación no verbal. Es decir, nuestra postura y gestos faciales. En este punto es importante agregar que, al momento de reflejar empáticamente, no estamos juzgamos ni calificando la emoción percibida, sólo la afirmamos y la expresamos de manera sencilla. De esta forma, el emisor puede darse cuenta de que hemos captado su emoción.

Ejemplo:

“Me doy cuenta de que esta situación te tiene muy enojada...”

“Te veo muy satisfecho con el resultado...”

Para hacer escucha empática sin dudas debemos “*aprender a mirar*” dado que, muchas veces nos vemos atrapados en nuestros propios discursos y se nos hace difícil sustraernos de nuestro propio discurso para ver qué es lo que está ocurriendo a nuestro alrededor.

El diálogo, la comunicación assertiva y la escucha empática como herramientas para resolver conflictos

Como sabemos, en el ámbito cotidiano y laboral no siempre podemos ponernos de acuerdo con todo. Es decir que no siempre es posible llegar a un consenso total dado que, las personas tenemos diferentes modos de ser, de pensar y de actuar. Por ello, muchas veces debemos llegar a acuerdos que permitan superar los problemas.

En este punto podemos decir que el diálogo, compuesto por las habilidades de comunicación asertiva y la escucha empática es una herramienta fundamental para resolver conflictos.

Entendemos que el diálogo no siempre elimina todo conflicto, pero sí, puede ayudar a que el conflicto se trabaje adecuadamente con el fin de construir y conciliar. Es por ello que necesitamos de ambas habilidades: comunicación asertiva y escucha empática.

Ante un conflicto:

- **¿Qué mirar?** En este punto debemos atender no sólo a lo que se nos dice sino también a cómo se dice a fin de detectar situaciones de peligro.

Son ejemplos:

- cuando la conversación se vuelve una discusión.
- cuando alguien intenta imponer posiciones u opiniones.
- cuando existen actitudes defensivas.
- cuando aparecen acusaciones.
- cuando se vuelve una y más veces sobre el mismo tema.
- cuando aparecen falta de respeto.

- **¿Qué hacer?** En primer lugar, debemos procurar seguridad y reanudar el diálogo aplicando las técnicas de escucha empática y comunicación asertiva.

Retroalimentación

Las habilidades de comunicación asertiva y escucha empática nos permiten desarrollar una tercera habilidad: **la retroalimentación** a fin de impulsar la mejora continua en el ámbito laboral y en el trabajo en equipo.

Retraolimentación = Feedback

Es decir, esta última nos permite dar continuidad al ciclo de la comunicación a fines de impulsar la mejora continua. Sin embargo, es importante mencionar que no todas las técnicas de retroalimentación permiten la mejora.

Para comprenderlo, miremos el siguiente fragmento de la película “[Un sueño posible](#)” dirigida por John Lee Hancock y luego, intentemos responder la pregunta ¿por qué Leigh Anne obtiene resultados y no el entrenador?

La respuesta parece simple, Leigh Anne conoce los intereses, valores, etc. de Michael por lo que, hace uso de ellos para hacer una retroalimentación efectiva.

Características de la retroalimentación

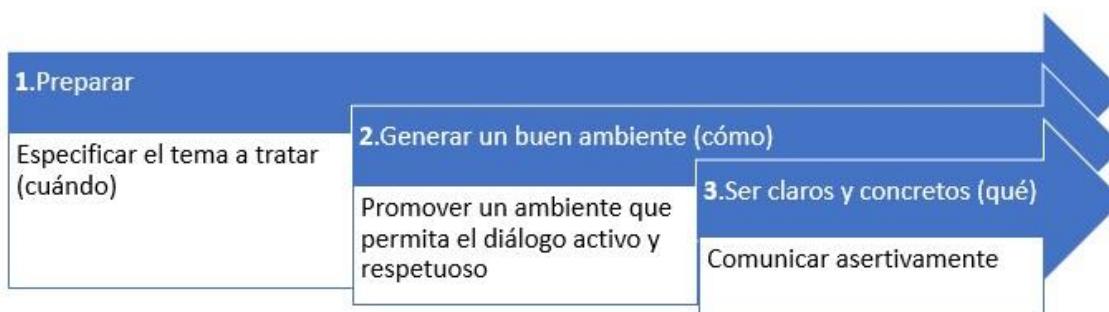
A continuación, enumeramos algunos puntos importantes a tener en cuenta para caracterizar a la retroalimentación:

- Aplicable, dirigido a un comportamiento que se pueda modificar (de lo contrario no tendría sentido)
- Objetivo, focalizado en un problema. No caer en un discurso emocional.
- Oportuno, en el mejor momento.

Claves para un buen feedback

Entonces podemos resumir, la clave para un buen feedback son:

- El uso de las palabras correctas (qué)
- Con el modo correcto (cómo)
- En el momento correcto (cuándo)



1. El primer paso es **prepararse**. Para ello, podemos hacernos las siguientes preguntas:

¿Cuál es el objetivo? ¿Cuáles son los datos y/o la información con la que contamos? ¿Quién o quiénes son los sujetos a quién va dirigido? ¿Cuáles son nuestros sentimientos y emociones? ¿y el de los otros? ¿Qué debo cuidar de mi manera de transmitir el mensaje? ¿Cuáles son los hechos? ¿Qué necesito de la otra persona? ¿Cuál es el momento adecuado? Tomarnos el tiempo para analizar estos puntos, nos permitirá anticiparnos y consecuentemente evitar la improvisación.

2. El segundo paso es **generar un buen ambiente**. Es decir, promover la conexión emocional con la otra persona. ¿cómo? Para ello debemos hacer rapport, es decir establecer una relación de igualdad en la que haya entendimiento y un sentimiento de unidad a fin de conectar con la otra persona y así, generar un buen ambiente.

NO VERBAL + VERBAL = RAPPORT

A continuación, algunas diferencias entre personas que aplican rapport y las que no:

Personas cuya atención está dirigida hacia sí mismas	Personas que establecen Rapport y están "con el otro"
<ul style="list-style-type: none"> • Se centran en sus pensamientos, hacen evaluaciones y juicios. • Su mirada carece de enfoque y mueven mucho los ojos. • Pueden adoptar cualquier postura. • Su lenguaje gira alrededor del "yo". 	<ul style="list-style-type: none"> • Su atención está centrada en la persona que habla. • Miran a la otra persona. • Su postura se amolda al de otra persona. • Su lenguaje gira alrededor de "tú" y utilizan pautas de lenguaje y palabras que encajan con el otro.

Y ¿por qué hacer rapport genera un buen ambiente? Para comprenderlo veamos una maravillosa escena de "[Patch Adams](#)". En la misma podemos observar que Patch Adams no logra conectar con su compañero de cuarto sino hasta que copia sus acciones.

Rapport se basa en las neuronas espejo, descubiertas no hace mucho. Las mismas son las responsables de que podamos empatizar con el otro dado que reflejan el comportamiento de los otros. Por ello, es natural que cuando dos personas que establecen una conversación y se sienten a gusto en ese diálogo imiten las posturas corporales del otro.

3. El tercer paso es **ser claros, precisos y respetuosos del tema a tratar**, cuidando el lenguaje verbal y no verbal, y ser empáticos al entender y comprender lo que el otro nos quiere decir. Utilizando mensajes YO.

¡Recuerda! El propósito de estas interacciones está relacionado a la mejora continua, tanto de la persona a quién nos dirigimos como de nosotros mismos.

Errores en la retroalimentación

A continuación, enunciamos algunos errores propios de la retroalimentación que debemos evitar:

- Dar una retroalimentación negativa en público (delante de los demás).
- Convertir una retroalimentación en un acto de castigo o venganza.
- Descuidar el tono de voz y el lenguaje corporal.
- Retroalimentar sin asertividad o con tono agresivo.
- No tener en cuenta los hechos.
- No dejar hablar al otro.

Técnicas de retroalimentación

El modelo GROW

El modelo GROW es un conocido sistema que permite definir un plan de acción para la mejora continua. También se suele utilizar para la resolución de problemas.

GROW es el acrónimo de Goal, Reality, Options y Way forward.

1. **G- Goal.** Consiste en definir el objetivo. El mismo debe ser alcanzable y medible.
2. **R- Realidad.** Consiste en comunicar asertivamente los hechos.
3. **O- Options.** Consiste en identificar las opciones posibles. Las mismas deben ser medibles y alcanzables.
4. **W- Way forward.** Se basan en el compromiso de las acciones a implementar. Debe hacerse seguimiento.

Feedback 360°

Es un proceso continuo que parte de preguntas, las cuales pueden ser cerradas o abiertas y tiene por objeto no sólo hacer una evaluación a los sujetos sino también impulsar la mejora continua de un sujeto. Permite además que las personas se sientan más valoradas.



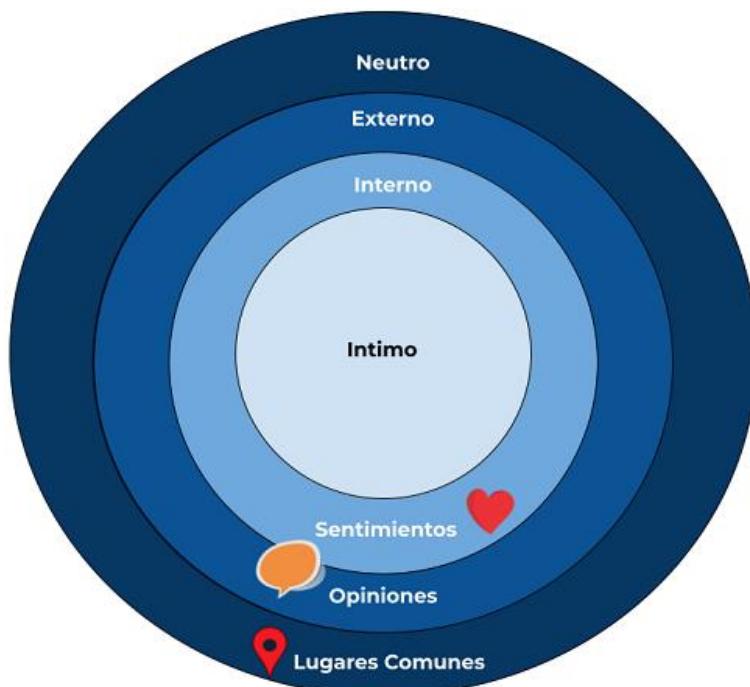
Para comprenderlo, a continuación, enunciamos algunas frases que pueden surgir de las personas en el ambiente laboral y sus respectivas preguntas que inician el ciclo.

Frases	Preguntas
Esta presentación no funcionará	¿Nunca? ¿Nunca has hecho una presentación con esas características? ¿Cómo quieres exactamente que sea?
Esperan que me las arregle con el material y las demos	¿Quién espera que hagas las dos cosas? ¿Cómo sabes que esperan que te las arregles solo con eso?
Estas reuniones me agobian	¿De qué manera te agobian? ¿Cómo te gustaría sentirte? ¿Qué te impide sentirte cómodo?
¡Sé por qué has hecho eso!	¿Qué hice? ¿Cómo debería haberlo hecho?

En este punto es importante, no hacer preguntas del tipo ¿por qué? dado que las mismas llevan más bien a hacer catarsis y no a la solución en sí.

Niveles de Comunicación

Otro punto importante a tener en cuenta al momento de comunicarnos son los niveles de comunicación dado que, nos permiten identificar desde qué nivel estamos transmitiendo el mensaje y desde qué nivel nos está transmitiendo la otra persona. Si ambos estamos en el mismo nivel entonces podemos decir que estamos en la misma “sintonía” obteniendo como resultado una mejor comunicación.



Fuente: Diseñada a partir del [ejemplo](#) en Credenciales Alternativas TEC.

Del gráfico anterior podemos observar los siguientes niveles:

1. **Neutral:** En este nivel se intercambian mensajes superficiales; es decir, temas que no nos afectan y no nos comprometen, pero que tampoco nos aportan mayor valor, por ejemplo: las conversaciones que podríamos tener en una parada de autobús o cuando recién estamos conociendo a alguien.
2. **Externo:** En este nivel intercambiamos opiniones e ideas que se intercambian proporcionándonos información que puede permitirnos tomar decisiones. Ej. las conversaciones que podríamos tener en una reunión de trabajo donde estemos discutiendo sobre un tema puntual expresando ideas u opiniones.
3. **Interno:** En este nivel la información que se comunica es emocional, por lo que requiere de ambos individuos tener la capacidad de comprender las emociones emitidas y recibidas. Ej. las conversaciones que podríamos tener con nuestro equipo de trabajo.
4. **Íntimo:** Este nivel nos permite la expresión y comprensión de sentimientos e ideas más profundas. Ej. las conversaciones que podríamos tener con un amigo íntimo, una hermana, etc.

¿Por qué es importante identificar el nivel de comunicación? Para comprenderlo observemos el maravilloso cortometraje titulado “[¿Qué es eso?](#)” dirigido por Constantin Pilavios.

En el cortometraje podemos observar la conversación entre un padre y un hijo pero, ciertamente ambos se están comunicando desde distintos niveles de comunicación. El padre transmite desde un nivel más bien interno o quizás íntimo mientras que, el hijo se comunica desde un nivel más bien externo o quizás neutro. Esto lleva a que no se puedan “encontrar” por así decirlo y por ende, el hijo no puede llegar a un entendimiento con su padre. No es hasta que el padre se retira, que el hijo toma conciencia de la situación y a continuación, cuando su padre regresa éste le presta atención y, al estar ambos transmitiendo desde el mismo nivel de comunicación en el que pueden finalmente entenderse.

Esto pasa todo el tiempo, en el hogar, en el ambiente laboral, en los equipos de trabajo, etc. Debemos identificar el nivel de comunicación desde el cual transmitimos y desde el cual nos transmiten las otras personas sus pensamientos, ideas, etc. así, llegar a un nivel que permita el diálogo activo y respetuoso.

Conclusión Comunicación Efectiva

Factores que influyen favorablemente en una Comunicación Efectiva

1. **Identificar necesidades y demostrar interés:** uno de los pilares principales de una Comunicación Efectiva es incorporar la perspectiva de los otros. Esto no supone decir lo que los demás quieren escuchar o ser condescendiente, sino partir de una vinculación genuina entre los

interlocutores que permita, además de alcanzar las metas planteadas, la satisfacción de intereses personales.

2. **Clarificar los objetivos y definir los roles:** el mejor punto de partida para una Comunicación Efectiva es plasmar de entrada el para qué y por qué estamos allí, cuáles son los resultados que se esperan, en qué tiempo, cómo se realizarán las actividades y qué funciones debe cumplir cada cual para alcanzar los objetivos enunciados.
3. **Usar un lenguaje claro y evitar los eufemismos:** construir discursos o contenidos descriptivos más que valorativos ayuda a disminuir las distorsiones. Se sugiere evitar utilizar palabras en un idioma distinto a la lengua materna de los interlocutores, así como terminología muy específica (jerga) que pueda ser desconocida para uno o algunos de ellos.
4. **Demostrar y ejemplificar:** mostrar casos y experiencias propias o ajenas colabora en la visualización de aquello que se verbaliza o se redacta. Los ejemplos que incluyen aciertos y errores en torno a una práctica, además de captar la atención -por hacer más dinámico el discurso- se vuelven un recurso de mucha utilidad para complementar conceptos complejos o con un alto nivel de abstracción.
5. **Escuchar atenta y activamente:** la diferencia entre oír y escuchar es que la primera es la recepción de lo que otro dice como consecuencia del acto físico que nos habilita el sistema auditivo. Escuchar implica la internalización de lo oído, que puede ser comprendido o no, pero que requiere poner atención y hacer un ejercicio reflexivo sobre lo percibido. Una escucha activa es aquella que se pone a disposición del otro con una honesta intención de involucrarlo en el devenir comunicacional. Implica, a su vez, ser flexibles, permeables, constructivos y colaborativos sin que eso suponga un sin rumbo o una falsa actitud.
6. **Diagnosticar y agudizar la observación:** obtener información sobre los participantes, su nivel de información acerca del tema a tratar, sus experiencias previas y su situación actual se torna un insumo imprescindible. Una vez desplegada la acción comunicativa es muy importante prestar atención a lo que sucede (en el plano verbal y no verbal) a medida que se desarrolla el proceso y, de ser posible, ir tomando anotaciones de frases, sensaciones o cualquier cosa que nos haya parecido relevante.
7. **Chequear y retroalimentarse:** A medida que nos vamos expresando resulta de mucha utilidad –y es muy valorado por los interlocutores- ir recuperando las ideas volcadas por los participantes, e incluso parafrasear sus dichos. Además de generar empatía, porque supone un acto de reconocimiento, es un muy buen mecanismo para reafirmar ideas e ir monitoreando el nivel de entendimiento de aquello que se está comunicando.

Factores que influyen negativamente en una Comunicación Efectiva

1. **Personalizar y desconfiar:** desde la gestación de aquello que se quiere comunicar y a lo largo de todo el camino es menester hacer el ejercicio de enfocarse en la problemática, tema o asunto y no en las personas, sobre todo si lo que se está abordando son conflictos o dificultades. Para ello es necesario generar un ámbito de confianza donde las intenciones y los objetivos estén clarificados desde el comienzo, evitando caer en situaciones incómodas o donde la gente se sienta engañada, mal predisposta o a la defensiva.
2. **Eludir y mentir:** Las exposiciones o los textos que se elaboren deben procurar ser explícitos y requieren ser confeccionados con cohesión y coherencia interna, dos aspectos centrales de la Comunicación Efectiva. En este sentido es fundamental evitar los rodeos y/o verter ideas que quienes las expresen no estén de acuerdo; no solo por una cuestión de principios sino porque eso generará un escenario de falta de credibilidad que hará inviable o muy difícil el cumplimiento de nuestros objetivos.
3. **Descalificar y subestimar:** Debe estar abierta la posibilidad del intercambio de ideas contrapuestas, las divergencias de opiniones y los desacuerdos siempre que se den en un marco de respeto y evitando las agresiones o el menosprecio por el parecer ajeno. La Comunicación supone un proceso de aprendizaje para quienes son partícipes, y es responsabilidad de quienes comandan las actividades no posicionarse en lugar de “sabelotodo” que solo genera distancia, malestar y obstruye la empatía.

La Comunicación Efectiva implica un trabajo permanente, en el plano individual y colectivo, que sistematice e incluya los aprendizajes de experiencias exitosas pero que al mismo tiempo habilite los espacios para reflexionar sobre las equivocaciones y los errores acontecidos, ya que si se pueden identificar las causas de los mismos se tornará un valiosísimo aporte de cara a cada nuevo desafío. Como todo hecho humano la Comunicación Efectiva es un proceso dinámico que permite ir identificando diversos facilitadores y obstaculizadores a los ya mencionados, los que a su vez servirán de insumo para el diseño de estrategias y herramientas comunicacionales que podremos aplicar en los distintos campos laborales que nos toque transitar, así como en nuestra vida personal.

Referencias

Habilidad Social. Qué es la asertividad y cómo ser más asertivo. <https://habilidadsocial.com/asertividad-10-claves/>

Percepción completa. Comunicación Efectiva • Cómo Mejorar La Comunicación. <https://youtu.be/YBWIMFjzy5o>

Credenciales Alternativas TEC. Comunicación efectiva para el líder actual. <https://www.youtube.com/watch?v=3HbjS-kUy9U&list=PLDef5gRMz9YFc6AhyoSSPkKhdcjVXYkl>

Scarpatti y Asociados. Cuadernillo Taller de Negociación MOTOROLA ARGENTINA SA

Scarpatti y Asociados. Cuadernillo Comunicaciones Efectivas MOTOROLA ARGENTINA SA