Modulo 1

¿Por qué es tan importante la Infraestructura de IT?
La tecnología es el motor de prácticamente todos los aspectos de las empresas actuales. Desde el trabajo de un empleado individual hasta las operaciones de bienes y servicios. Si se conecta correctamente, la tecnología se puede optimizar para mejorar la comunicación, crear eficiencia y aumentar la productividad.

Con una infraestructura de TI, una empresa puede:

-Ofrecer una experiencia del cliente positiva proporcionando acceso ininterrumpido a su sitio web y la tienda en línea
-Desarrollar y lanzar soluciones al mercado con rapidez
-Recopilar datos en tiempo real para tomar decisiones rápidas
-Mejorar la productividad de los empleados
Arquitectura cliente-servidor
Procesa la información de un modo distribuido; de esta forma, pueden estar dispersos en distintos lugares y acceder a recursos compartidos. Además de la transparencia y la independencia del hardware y software una implementación debe tener las siguientes características:

-Utilizar **protocolos asimétricos**, donde el servidor se limita a escuchar en espera de que un cliente inicie una solicitud.
-El acceso es transparente, multiplataforma y multi arquitectura
-Fácil **escalabilidad**, de manera que sea fácil agregar nuevos clientes a la infraestructura o aumentar la potencia del servidor/es (escalabilidad horizontal), aumentando su número o capacidad de cálculo (escalabilidad vertical).
Pasos
Paso 1: **Servidor**
Inicio del servidor (durante el arranque del sistema operativo o por intervención del administrador), en espera de requests de clientes.

Paso 2: **Cliente**
Cliente realiza request al servidor

Paso 3: **Request**
El servidor recibe el request, realiza verificación (en caso que sea necesaria), y la procesa.

Paso 4: **Response**
Cuando el servidor disponga de lo solicitado, lo envía al cliente.

Paso 5: **Middleware**
El cliente recibe el resultado que solicitó y realiza las comprobaciones oportunas (si es necesario), si está todo correcto se lo envía al usuario.

Middleware

Encargada de traducir el request y response (ayuda que se entienda perfectamente) entre el cliente y servidor.

Tipos:

DBMS
Sistema de administración de base de datos que permite crear, recuperar, actualizar datos.

API
Interfaz de programación de aplicaciones que brindan un conjunto de subrutinas funciones y procedimiento para ser usadas en otros softwares.

Data Centers
Almacenamiento de datos que están protegidos contra incendios, accesos indebidos, pérdida de luz

Características fundamentales de un S.O. (para servidor)

Soporte de red
Es indispensable que tengan un soporte completo para poder brindar conectividad

Amplia compatibilidad con el hardware
Capacidad para poder exprimir al máximo las características técnicas del hardware en donde se ejecuta.

Seguridad
Actualizado con todos los parches/updates con estrictas políticas de acceso para prevenir accesos malignos.

Tolerancia a fallos
Mediante una granja de servidores, que interconectadas operan como una gran unidad de proceso (así en el caso que algún servidor se caiga, otro pueda tomar la responsabilidad).

Modulo 2

Scripting

Bash
-interpreta órdenes del usuario, ejecutando en consola de unix (o en un archivo .sh)

Le indicamos al archivo que vamos a escribir en el lenguaje **Bash**

```
#!/bin/bash
```

Indicamos lo que imprimimos por pantalla (**echo**)

```
echo "Ejemplo lo que vemos por pantalla"
```

Como se escribe una variable

```
variable = tipo de dato
```

Para llamar una variable hay que utilizar el $

```
$variable
```

Indicamos que termino la función

```
Fi
```

Comparaciones numéricas

Se puede hacer una comparacion numéricas entre dos valores numéricos utilizando las siguientes sentencias:

number1-eq number2	Comprueba si number1 es igual a number2
numbre1-ge number2	Comprueba si number1 es más grande o igual a number2
number1-gt number2	Comprueba si el number1 es más grande que el number2
number1-le number2	Comprueba si el number1 es más pequeño o igual que el number 2
number1-it number2	Comprueba si el number1 es más pequeño que number2
number1-ne number2	Comprueba si number1 no es igual a number2

Phyton
Su principal objetivo es la automatización de procesos, uno de sus puntos fuertes es que "**comprueba errores sobre la marcha". Python** crea un código con gran legibilidad, que ahorra tiempo y recursos.

Variables en Python
Python es un lenguaje de programación orientado a objetos, las variables son "etiquetas" que permiten hacer referencia a los datos (que se guardan en unas "cajas" llamadas objetos).

Cada objeto tiene:
-Un identificador único (un número entero, distinto para cada objeto)
Permite referirse al objeto sin ambigüedades

-Un tipo de dato (entero, decimal, cadena de caracteres)
Permite saber qué operaciones pueden hacerse con el dato

-Un valor
El propio dato

Las variables se definen escribiendo su nombre igualando al tipo de dato y puede tener estructuras más complicadas.

Modulo 3

DOCKER

Es una tecnologia q permite ejecutar un proceso de manera aislada del resto de los procesos de la maquina y que defina todas las dependencias que necesita para ejecutarse. Las distribuciones, las dependencias que necesita del Sistema operativo y del lenguaje de programacion con Docker se empaquetan en una imagen de manera que este disponible toda esa info a la hora de ejecutar el programa. Entonces este programa va a ser totalmente portable ya que voy a poder ejecutar esa imagen en una Mac, en una window , en ubuntu y en todas va a funcionar igual gracias al contenedor y su unica condicion es tener aislado en esa maquina docker y un kernel de unix

Vocabulario de Docker
Te dejamos algunos conceptos que serán importantes que reconozcas para que te familiarices con ellos:

Host: una máquina virtual que ejecuta Docker daemon para alojar una colección de contenedores Docker.
Cliente: aquí se ejecutan los comandos que están siendo ejecutados. (cliente-servidor).
Imagen: una colección ordenada de sistemas de archivos (capas) que se utilizarán al crear una instancia de un contenedor.
Contenedor: una instancia en tiempo de ejecución de una imagen.
Registro: una colección de imágenes de Docker.

Arquitectura del contenedor

Los contenedores ofrecen una alternativa a la ejecución de aplicaciones directamente en el host o en una máquina virtual que puede hacer que las aplicaciones sean más rápidas, portátiles y escalables. La flexibilidad proviene de que el contenedor puede llevar todos los archivos que necesita. Al igual que la aplicación que se ejecuta en una máquina virtual, puede tener sus propios archivos de configuración y bibliotecas dependientes, además de tener sus propias interfaces de red distintas de las configuradas en el host. Entonces, nuevamente, al igual que con la VM, una aplicación en contenedores debería poder moverse más fácilmente que sus contrapartes instaladas directamente y no tener que competir por los mismos números de puerto porque cada contenedor en el que se ejecutan tiene interfaces de red separadas.

Docker engine: permite construir, enviar, ejecutar contenedores

Docker Hub: registro en linea de imagenes de Docker

Docker trusted: registry registro privado en el sitio para imagenes de Docker

Docker client: toma entradas del usuario y se las envia al Daemon. el cl y el daemon pueden ejecutar en el mismo o diferente host.

Docker Images: plantilla de solo lectura para crear contenedores que contiene las instrucciones para construirlos

Docker Containers: plataforma de app aislada basada en una o mas imagenes que contiene todo lo necesario para ejecutar la app

¿Qué es un registro Docker?

Los registros Docker sirven para almacenar las diversas imágenes Docker que utilicemos en nuestro sistema. De esta forma podremos subir imágenes nuevas a los registros o descargarlas cuando las necesitemos en alguna máquina Docker.

El registro es como una estantería donde las imágenes se almacenan y están disponibles para extraerlas con el fin de compilar contenedores que ejecuten servicios o aplicaciones web. Hay registros de Docker privados a nivel local y en la nube pública. Docker Hub es que un registro público mantenido por Docker; junto con Docker Trusted Registry, una solución a nivel empresarial

Colocar imágenes en un registro permite almacenar fragmentos de la aplicación que son estáticos e inmutables, incluidas todas sus dependencias a nivel de marco. Después, esas imágenes se pueden versionar e implementar en varios entornos y, por lo tanto, proporcionar una unidad de implementación coherente.

QUE ES DOCKERFILE?

Es un archivo de texto simple con un conjunto de comandos o instrucciones estas se ejecutan para realizar acciones en la imagen base para crear una nueva imagen de la ventana acoplable.

instrucciones basicas de dockerfile

de: define la imagen base para usar e iniciar el proceso de construccion

correr: toma el comando y sus argumentos para ejecutarlo desde la imagen

CMD: funcion similar a un comando run, ejecuta solo despues de que se crea una instancia del contenedor

punto de entrada: se dirige a su aplicacion predeterminada en la imagen cuando se crea el contenedor

añadir: copia los archivos de origen a destino dentro del contenedor

ENV: establece variables de entorno

TAG O ETIQUETA DE LA IMAGEN

sirven para identificar las versiones de las imagenes. si no tiene etiqueta se asume el valor predeterminado de la última.

IMAGEN DOCKER

una imagen se crea a partir de una serie capas

la imagen del Sistema operativo de la plataforma base la prporcionan proveedores como Microsoft para window, canonical para ubuntu. las imagenes se publican en Docker Hub.

Cada capa erpresenta a una instruccion en el docker file

DOCKER COMPOSE es una herramienta que permite simplificar el uso de Docker. A partir de archivos YAML es más sencillo crear contendores, conectarlos, habilitar puertos, volúmenes, etc. Con Compose podemos crear diferentes contenedores y al mismo tiempo, en cada contenedor, diferentes servicios, unirlos a un volumen común, iniciarlos y apagarlos, etc. Es un componente fundamental para poder construir aplicaciones y microservicios.
En vez de utilizar Docker vía una serie inmemorizable de comandos Bash y scripts, DOCKER COMPOSE te permite mediante archivos YAML para poder instruir al Docker Engine a realizar tareas, programáticamente. Y esta es la clave, la facilidad para dar una serie de instrucciones y luego repetirlas en diferentes ambientes.

DOCKER SWARM herramienta integrada de Docker para agrupar una serie de host de Docker en un cluster y gestionarlo de forma centralizada, asi como orquestar contenedores (ORQUESTAR: Este nuevo enfoque consiste en empaquetar los diferentes servicios que constituyen una aplicación en contendedores separados, y luego desplegar esos contenedores a través de un clúster de máquinas físicas o virtuales.)

Mejores prácticas en DOCKER para desarrolladores

Usar imágenes oficiales
Utilizar el comando COPY en lugar de ADD
Usar multi-stage builds
No generar dependencias externas
Concatenar comandos

NUBE HIBRIDA

Arquitectura que incluye gestion, organizacion y portabilidad de las cargas de trabajo al menos en 2 entornos >> al menos una nube privada y una publica
dos o mas nubes publicas
dos o mas nubes privadas
un entorno virtual o sin S.O. a al menos una nube pub o priv
deben poder realizar lo siguiente:
conectar varias computadoras a traves de la red
consolidar los recursos de IT
escalar rapidamente e implementar los recursos rapidamente
trasladar las cargas de trabajo entre los entornos
Incorporar una sola herramienta de gestion unificada
Automatizar

Este nuevo enfoque consiste en empaquetar los diferentes servicios que constituyen una aplicación en contendedores separados, y luego desplegar esos contenedores a través de un clúster de máquinas físicas o virtuales

COMANDOS - (desde ubuntu) =

docker -v //version
docker pull ubuntu:latest // meto ubuntu en docker -va a pesar menos xq usa solo lo necesario

docker run ubuntu:latest (como es la ultima version puedo poner solo ubuntu y sabe q es la ultima) // para correr mi docker
(si no le pongo que quiero correr se rompe)
docker run nginx:latest // me baja la ultima version delservidor web nginx q es como un apache

docker images // listo mis imagenes
docker ps // listar mis contenedores
docker ps -a // historial de contenedores que corrimos
docker ps -aq // me lista solamente las id

docker stop nombreDelContenedor // si quiero frenar 1 contenedor lo hago con el

containter id o NAME q tiene
docker stop dc0 // puedo frenarlo tmb poniendo los primeros caracteres del id, siempre y cuando no sean iguales a otro contenedor
docker start elId_deMiDocker // prender
docker start funciona con id o nombre, docker run funciona con imagen y lo q hace es correr x 1ra vez un contenedor

docker rmi ubuntu:latest // borro la imagen (puede q si estoy usando esa imagen en otro contenedor, no pueda borrarla. para poder hacerlo hago docker ps -a para ver cual es el contenedor y borrar ese contenedor, y después si borrar la imagen)
docker rm idDelContenedor // borro el contenedor
docker rmi ubuntu:latest --force // borra todo y no le importa nada ---NO USARLO
docker system prune --all (borra TODO - si no esta corriendo)

docker rm $(docker ps -aq) --force // borra todos los id q liste en ese comando

°°°Para correr en 2do plano y no necesitar abrir otra terminal: (-p es puerto) (-d modo detached o background..es decir en 2do plano)

docker run -d -p 90:80 nginx (90 = pto local 80= pto salida) *para ver si lo hice bien, abro la pagina y escribo localhost:90 y me muestra

docker run -d -p 2022:80 httpd // hace correr mi APACHE

_____

MONTAR PÁGINA EN CONTAINER - ejecución desde cero:

docker run -d -p 9090:80 -v $HOME/Desktop/ProyectoWeb:/usr/share/nginx/html nginx
(-v son volúmenes)
($HOME/Desktop/ProyectoWeb: - variable de la ruta dde esta el archivo)
(:/usr/share/nginx/html nginx - a donde tengo q pegarlo)

los archivos de la pagina q quiero copiar los paso a la carpeta $HOME CON WINSCP.NET

_____

echo $HOME // DONDE ESTA Mi VARIABLE
pwd // donde estoy parado yo
_____

con Python:
docker run -d python:3.7 // pide que corra la versión de python y ahi puedo escribir

docker run -it python:3.7 // para entrar en python y ejecutarlo dentro de mi contenedor (con el comando -it puedo interactuar con ese contenedor

_____

docker exec -it idContenedor bash // (ingresar para ver adentro de ese contenedor.)

// no se hace xq es mala practica, lo q hacemos es crear el archivo con un dockerfile:
docker exec -it idContenedor mkdir montoto(creo carpeta)

Modulo 4

SERVERLESS

Serverless significa sin servidor, es una solución que permite crear y ejecutar aplicaciones con rapidez y menor costo total de propiedad, ya que no es necesario aprovisionar y administrar infraestructura. Evidentemente, por detrás hay servidores para ejecutar las aplicaciones, pero el proveedor de nube se encarga de la administración, por lo tanto, de nuestro lado dejamos de preocuparnos por administrar servidores, sistemas operativos, software y demás recursos, y únicamente nos centramos en el código de la aplicación.

Con una arquitectura Serverless podemos procesar miles de peticiones, hasta millones, sin pensar en la capacidad de recursos necesarios para que la aplicación funcione correctamente, la escalabilidad está garantizada.

POSITIVO Reduce costos operativos (en la nube se paga por tiempo de uso) y aumenta prducción
NEGATIVO los prpveedores de nube tienen restricciones en cuanto a la interaccion de los elementos lo cual determina la flexibilidad y personalizacion de sus propios sistemas
NEGATIVO a la hora de cambiar de proveedor supone un costo de actualizacion de los sistemas para que funcione con el nuevo proveedor

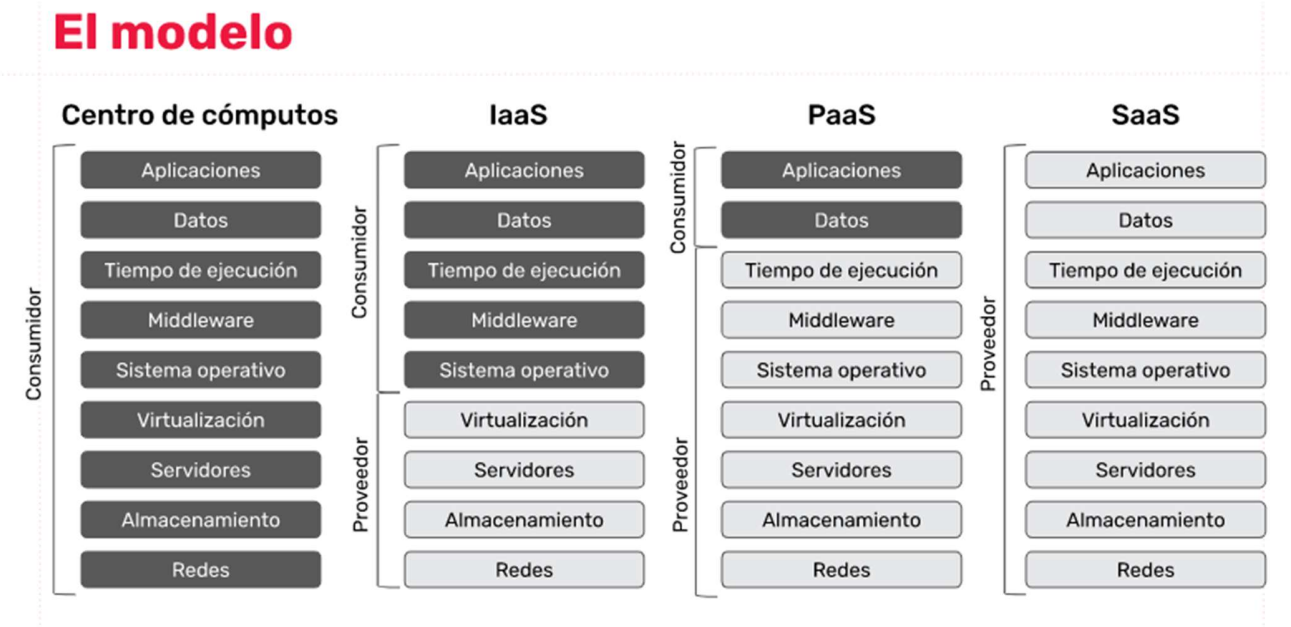
OTROS SERVICIOS QUE SE OFRECE LA NUBE

Storage
Compute
Analisis
interaccion con clientes
machine learning
robotica
satelite
seguridad, identidad y conformidad
tecnologia sin servidor
base de datos
herramientas para desarrolladores
informatica para usuarios finales
servicios front-end web y moviles
game tech
internet de las cosas
adm y direccion
serv multimedia
migracion y transferencia

Almacenamiento redes y entrega de contenido
realidad Virtual y realidad aumentada tecnologia cuantica
Integracion de aplicaciones
blockchain
app empresariales
adm financiera en la nube
Cómputo
contenedores

Modelo de responsabilidad

Es un estandar que nos permite definir diferentes tipos de contratacion de servicios. Define que tareas quedan delegadas a la nube y cuales seran responsabilidad del cliente



Cuanto más control tengamos sobre los activos, mayor será nuestra responsabilidad sobre estos.
Ejemplos de Paas: Amazon DynamoDB
Ejemplos de SaS : Trello, Salesforce y Gmail

Ventajas de las máquinas virtuales en la nube

- menos gasto en hardware
- Software actualizado constantemente
- Solo pagamos por el uso
-Accesibilidad desde cualquier lugar
- Mayor rapidez de implementación

- Menores costos operativos
- Fiabilidad
- Potencia escalable

Desventajas de usar máquinas virtuales ubicadas en la nube
- Menor rendimiento
- Su tiempo de respuesta o latencia es mayor

Tipos de MV

- de uso general
- optimizadas para informática
- optimizadas para memoria
- optimizadas para almacenamiento
- informática acelerada

Escalabilidad de la nube - es la capacidad de crecer en capacidad cuando haya demanda.
Tolerancia a fallos de la nube - es la capacidad de continuar siendo funcional cuando existen errores
Disponibilidad de la nube - cuando tenemos multiples recursos y uno de estos falla, pero tenemos otros que pueden hacer su trabajo.

Proveedores
Amazon EC2
Azure Virtual Machines
Google VM Instances