**Infraestructura I**

**MODULO 1**

* **Arquitectura Cliente Servidor:**

**Caracteristicas Sistema Operativo**: Soporte Red, Compatibilidad de hardwares,

Seguridad, Tolerancia a fallos.

**Componentes de Virtualización**: Maquinas Virtuales – Admin. De MV - S.O. - Hardware

**Beneficios**: Eficiencia, Tiempo de actividad, Despliegue, Ahorro de energía, Snapshots, Backups, Alta diponibilidad, Costo.

* **Maquinas Virtuales:**

**VirtualBox**: manejador de M.V

**Crear Maquina Virtual**: recibimos una imagen .ISO que contiene la M.V. Resto de los pasos en Virtualización playground C2S automatización.

**Redes**: protocolo UDP (protocolo de datagrama de usuario) no establece una conexión

directa, es decir, el protocolo envía los datos a quien los reciba.

TCP (protocolo de control de transmisión) establece una conexión antes de

enviar el tráfico, luego envía los paquetes de datos y confirma la recepción.

**Comunicación a través del Modelo OSI**

**ITSM**: information tecnology service managment. El loco que arma la infrastructuras para las empresas

**ITIL**: guía de buenas practicas para la gestión de servicios de tecnología

**Vagrant**: utilidad que permite controlar el ciclo de vida de las M.V.

1. Se crea el archivo vagrantfile y dentro se le ponen las configuracione( como un script)
2. Se ejecuta con **vagrant up**
3. Ingrasar a la M.V con **vagrant ssh ‘nombre’**
4. Dentro de la M.V servidor se ejecuta **sudo tcpdump -X icmp -i ‘placa de red’** para iniciar el analizador de red.
5. **ping –c 1 ‘ip de donde quiero la respuesta’**

**tcpdump:** utilidad principal es analizar el tráfico que circula por la red

**curl**: forma de verificar la conectividad a las URL. mostrar el contenido de una página

**icmp** maneja los mensajes de error y control para IP.

**MODULO 2**

**C2A automatización** Las 5 tareas más comunes en IT para automatizar y pdf monitoreo y demás herramientas que se utilizan

**C2S** instalación MV con virtualbox pdf

**C3S** ejercicio con vagrant

**C4A y C4S** comandos en linux

**C5S** en discord semana 4 funciones en Shell y playground pdf

**C8A** Python

* **Vagrant**:

Para crear moldes de máquinas virtuales. Modelado de máquinas virtuales iguales. Herramienta de automatización. Crea las MV a través de VirtualBox

1. se instala con un ejecutable y luego solo se utiliza por consola
2. **vagrant versión**
3. se hace el archivo **Vangrantfile. nodepad ++ Vangrantfile.** (se puede hacer en el visual también. Abre el ID y ahí escribi el archivo con las configuraciones.

Vagrant.configure(“2”) **do** |config| (primera 3 líneas para elegir que MV usa)

config.vm.define “server” **do** |server|

config.vm.box = “ubuntu/focal64”

config.vm.provider “virtualbox” **do** |v|

v.memory = 1500

v.cpus = 2

**end**

server.vm.hostname = “server”

server.vm.network “public\_network”

**end**

**end**

1. **vagrant validate** (tengo que validar el arichivo Vagrantfile en la misma ubicación)
2. **vagrant up** (levanta el archivo y arranca a ejecutarlo. Descarga la imagen.ISO de la MV que configure. Me pregunta la interface que quiero usar y elijo ahí la placa de red que estoy usando en mi compu. Para ver la placa ver en opciones de red. Tener el virtualbox cerrado para hacer esto. Genera una carpeta .vagrant donde encuentro la private key).
3. **vagrant ssh** (para conectarme a la MV siempre parado en la carpeta donde esta. Se conecta sin mas configuraciones)
4. **sudo apt update** (para actualizar los paquetes de la MV)
5. **exit** (para salir pero sique corriendo)
6. **vagrant halt** (apago la MV)
7. **vagrant destroy –f** (mata la MV)

* **Configuration Management:**

**Herramientas:**

**Chef**:  plataforma de configuration management con una arquitectura cliente-servidor, principalmente, orientada a Linux. configuraciones son definidas y asignadas en el servidor y es mediante el cliente de Chef que se hacen efectivas.

**Puppet:** igual che chef.  El cliente también funciona en modo “Pull”. La diferencia esta en la capacidad de reporting que tiene puppet.

**PowerSell DSC:** desired state configuration. Originalmente creada para configurar hosts Linux. Puede funcionar tanto en una arquitectura cliente-servidor (modo “pull’) o en una modalidad standalone (modo “push”)

**Ansibile:**  desarrollada en Python. Ansible no requiere de la instalación de un agente en aquellos servidores que se configuraran usando Ansible, sino que la herramienta establece una conexión vía SSH y de esa manera despliega las configuraciones. Las configuraciones en Ansible se escriben en archivos con formato YAML (un superset del formato JSON).

* **Ansibile C11A**:

Software de gestión de las configuración automática y remota. Para configurar numerosos servidores

**Comandos bash:**

**locate.archivo.txt** buscar archive local

**rm –rf archive.txt** borra forma forzada

**find.archivo.txt** buscar archivos

**sudo apt-get install vin**

**vim lista\_nombres.txt** (abre editor tipo nano. Aprieto la i y puedo editar el archivo y guardo con :wq! Para salir sin guardar :q!)

**cat lista\_nombre** veo que tiene el archivo

**head -5 lista\_nombre** muestro los promeros 5 del archivo

**tail lista\_nombre** nuestro los últimos 10

**cat** leer o crear archivos

**cat lista\_nombres | grep –i Esteban** busca y muestra a esteban de forma anidada

**SCRIPT**

**nano script.sh** creo el archivo y lo voa a editar con nano

**#!/bin/bash**

**mkdir Script**

**cd Script**

**makdir Clase1 Clase2 Clase3**

**touch Clase1/proactico.txt Clase2/parctico.txt**

**touch Clase1/teorico.txt**

**bash script.sh** para correr el script y me crea todas las carpetas en este caso

**#!/bin/bash**

**nombre=”pato”**

**edad=35**

**if [[ $nombre = “pato” ]]; then** (para string se usa el = y sino el –eq simpre doble [[)

**echo “Hola $nombre”**

**else**

**echo “Hola no sos pato”**

**fi**

for i in $(cat nombre.txt); do

echo $i

done

**while read line; do**

**echo $linea**

**done < nombres.txt**

cat nombre.txt | sed ‘s/ /./g’ > temporal.txt

for indice in $(cat temporal.txt); do

echo $indice

done

**Módulo 3**

**Docker**

**Containers**

Entornos físicos

Aplicaciones construidas e implementadas en sistemas físicos, relación 1:1

Requieren de nuevos sistemas para el aislamiento de recursos

Entornos virtuales:

Mejor implementación que las otras más rápidas

Muchas en MV

Los 2:

Aceleran implementación de una aplicación

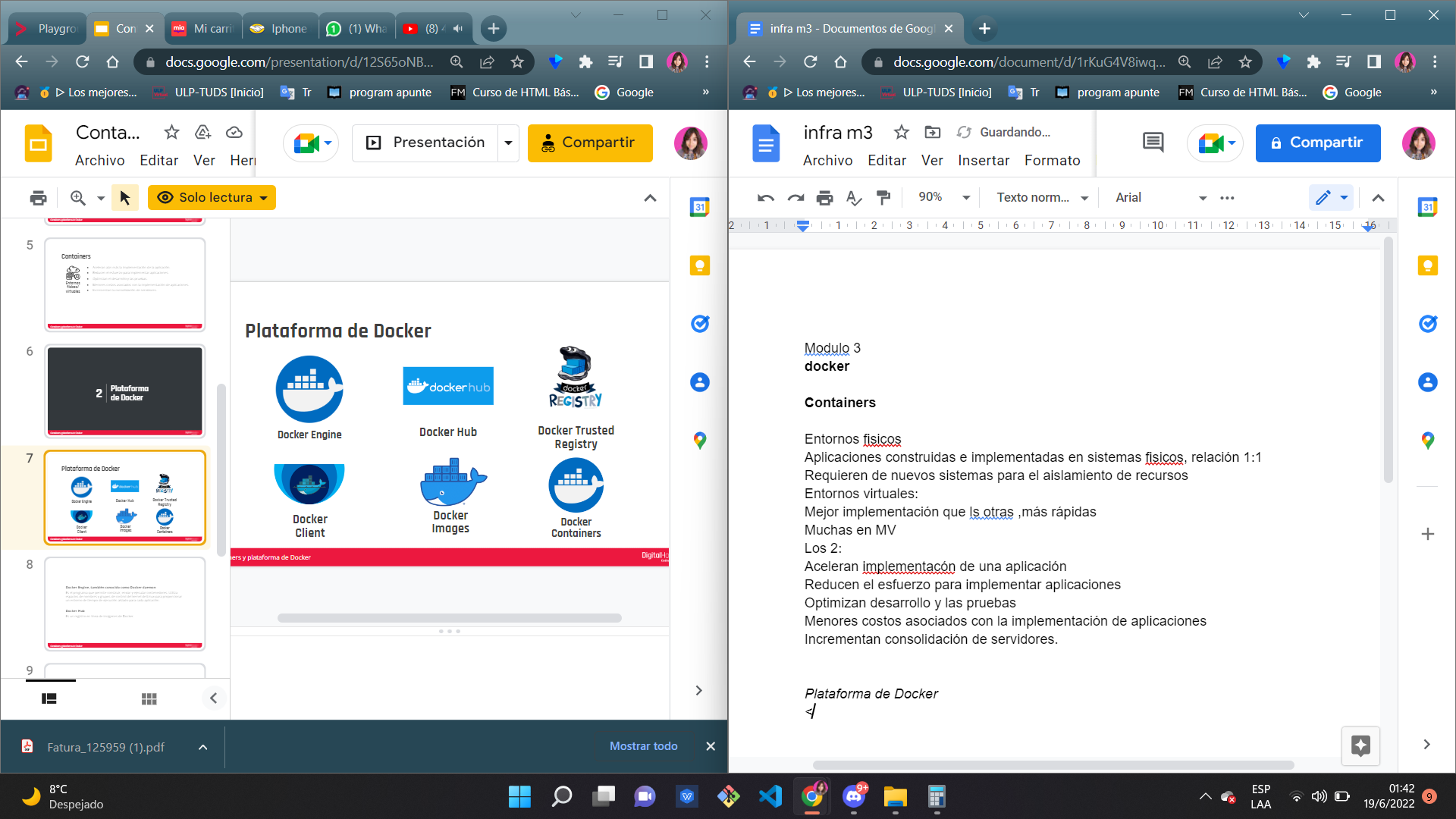
Reducen el esfuerzo para implementar aplicaciones

Optimizan desarrollo y las pruebas

Menores costos asociados con la implementación de aplicaciones

Incrementan consolidación de servidores.

*Plataforma de Docker*



*Docker Engine*

Programa crea, envía y ejecuta contenedores.

Espacios de nombres y grupos de control del Kernel de linux para proporcionar un entorno de tiempo de ejecución aislado para cada aplicación.

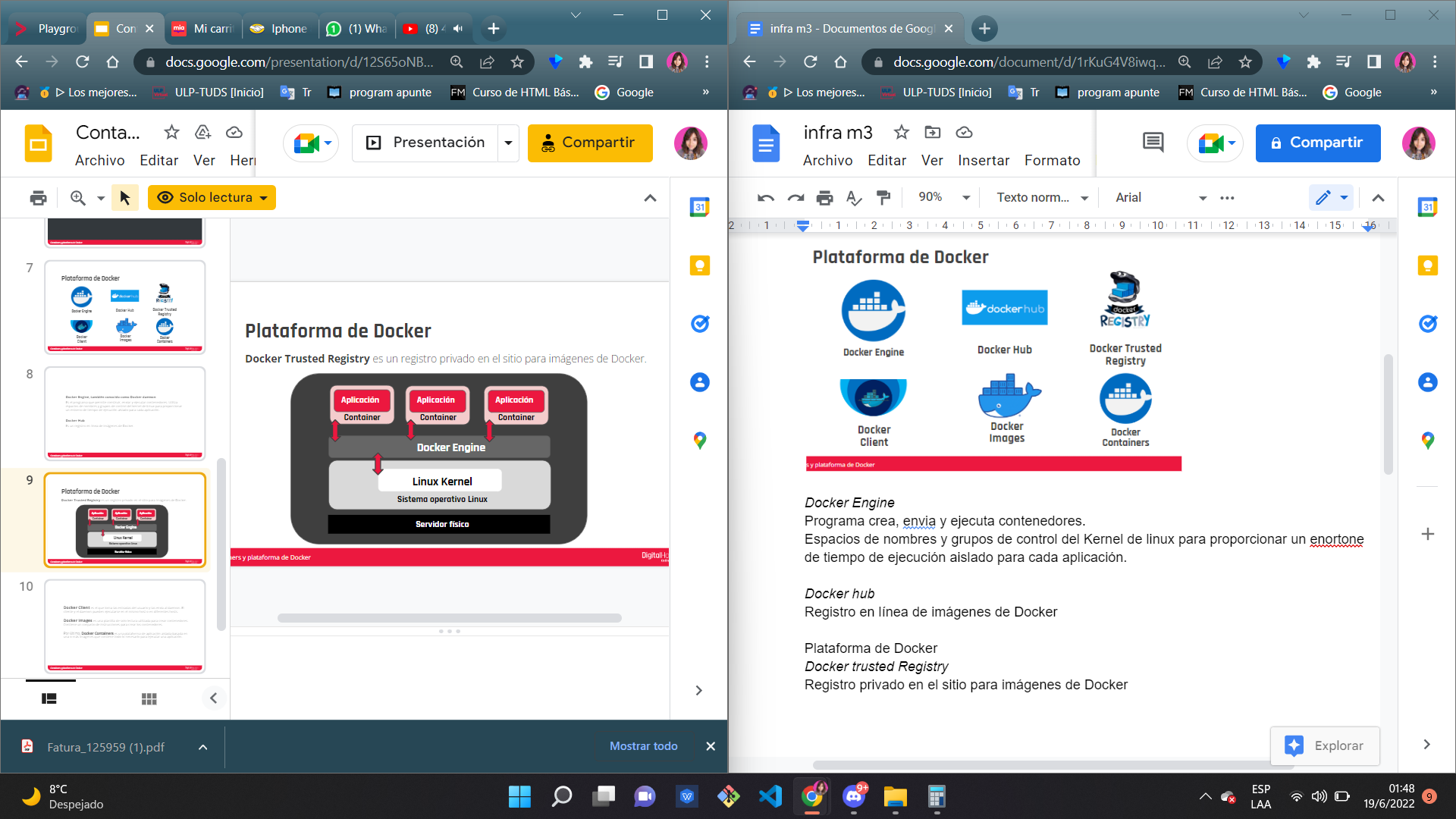
*Docker hub*

Registro en línea de imágenes de Docker

Plataforma de Docker

*Docker trusted Registry*

Registro privado en el sitio para imágenes de Docker



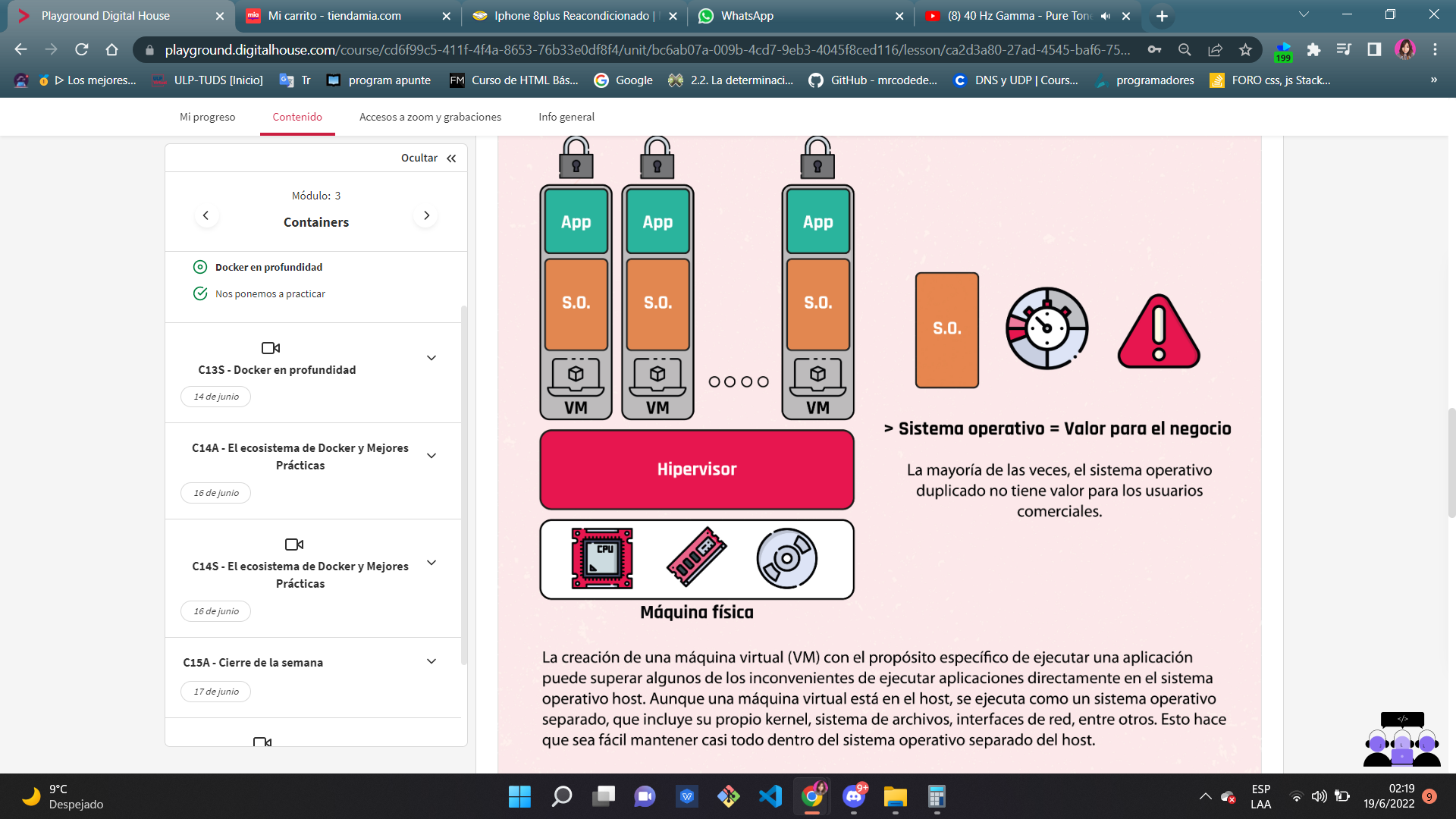
Docker Client: toma entrada de usuario y las envía al daemon. Se pueden ejecutar en el mismo o en diferente host.

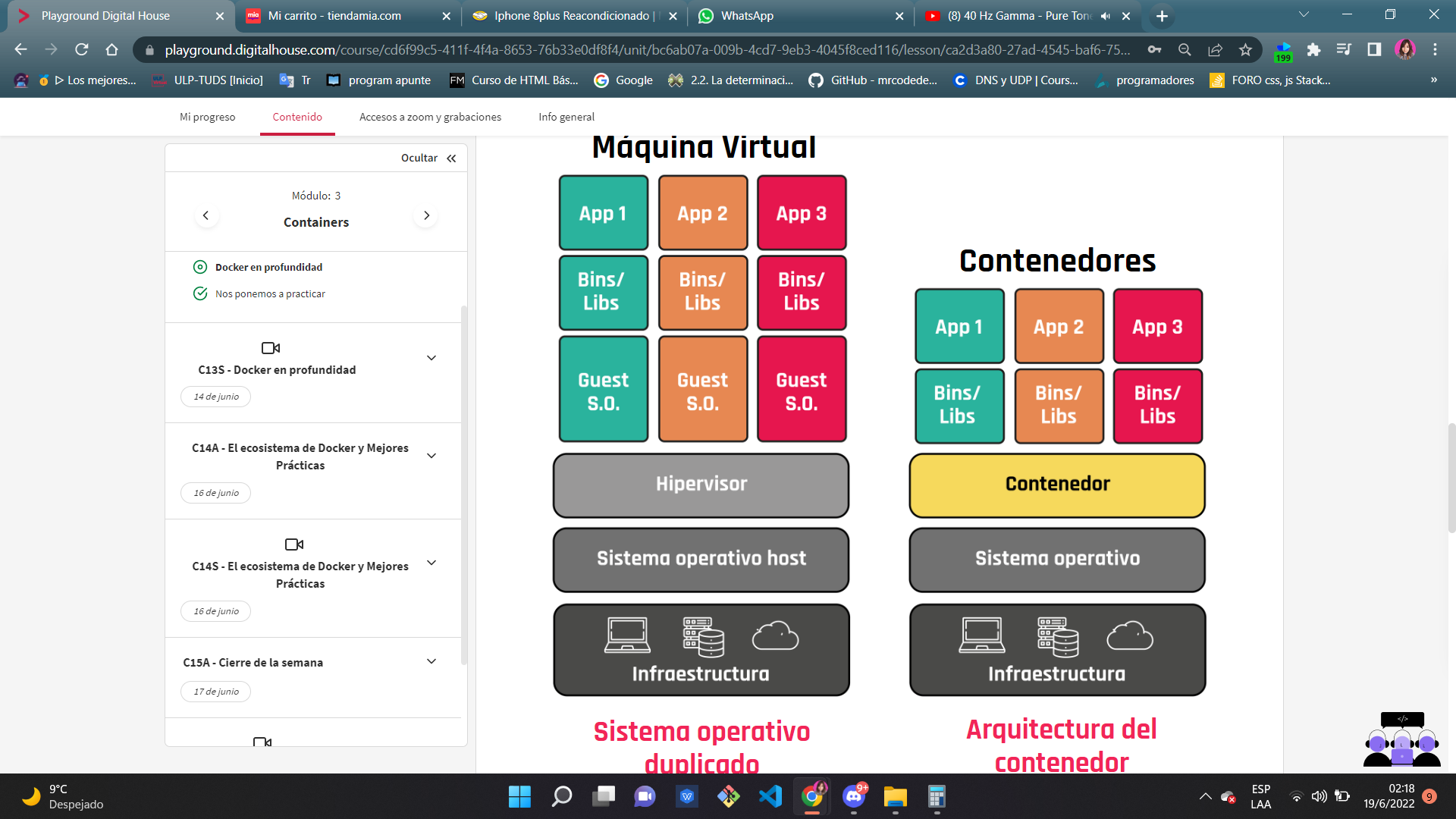
Docker images: plantilla de lectura para crear contenedores, instrucciones para crear contenedores.

Docker contains: entorno de app que tiene una o mas imágenes con lo necesario para ejecutar una app.

Conceptos básicos:

* Host: MV que ejecuta docker daemon, que aloja colección de contenedores
* Cliente: Ejecuta comandos ejecutados (cliente-servidor)
* Imagen: sistemas de archivos capas usadas al crear instancia de un contenedor
* Contenedor: instancia en tiempo de ejecución de una imagen.
* Registro: colección de imágenes de Docker

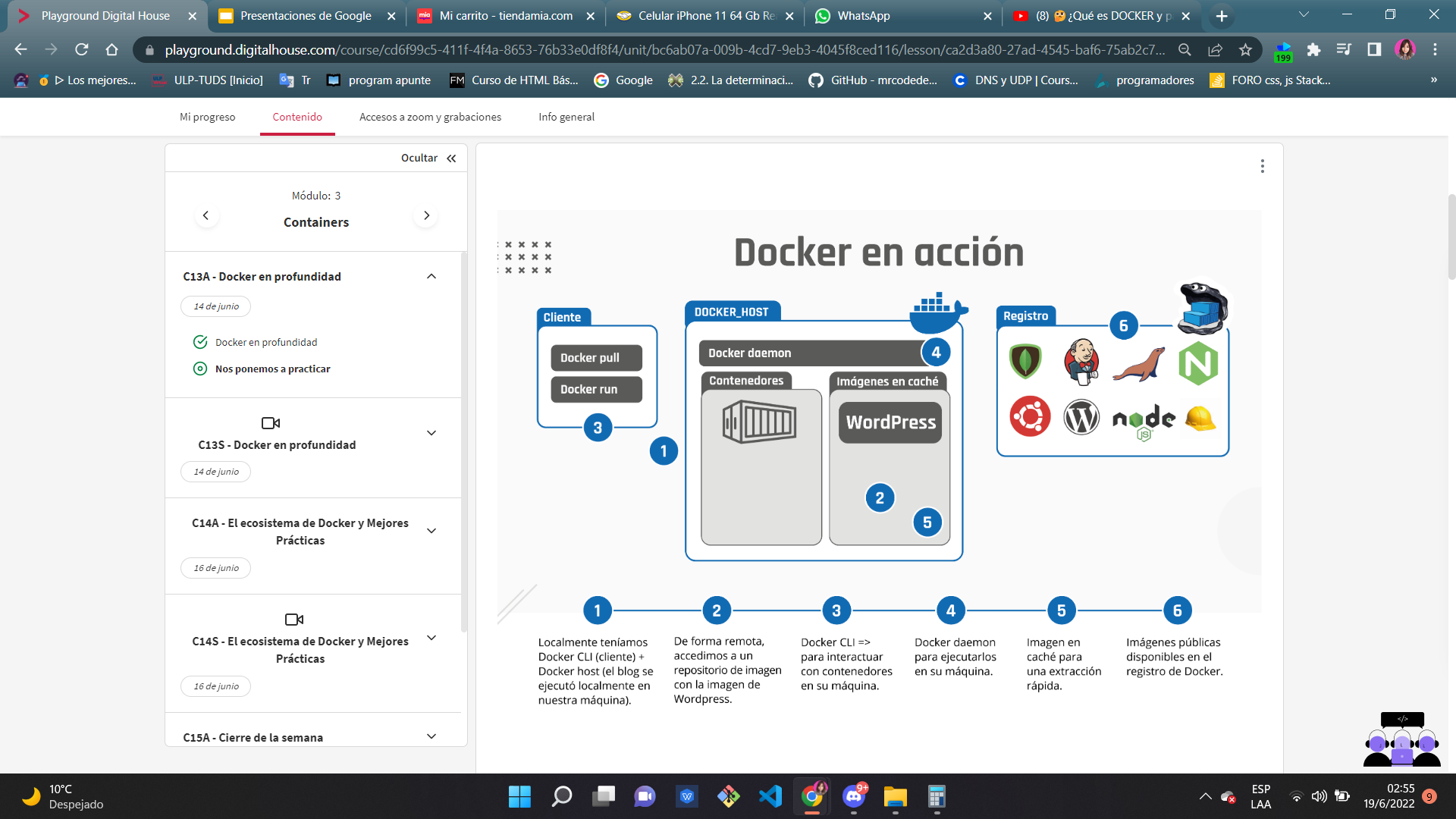




Arquitectura del contenedor:

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. 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 Registry*

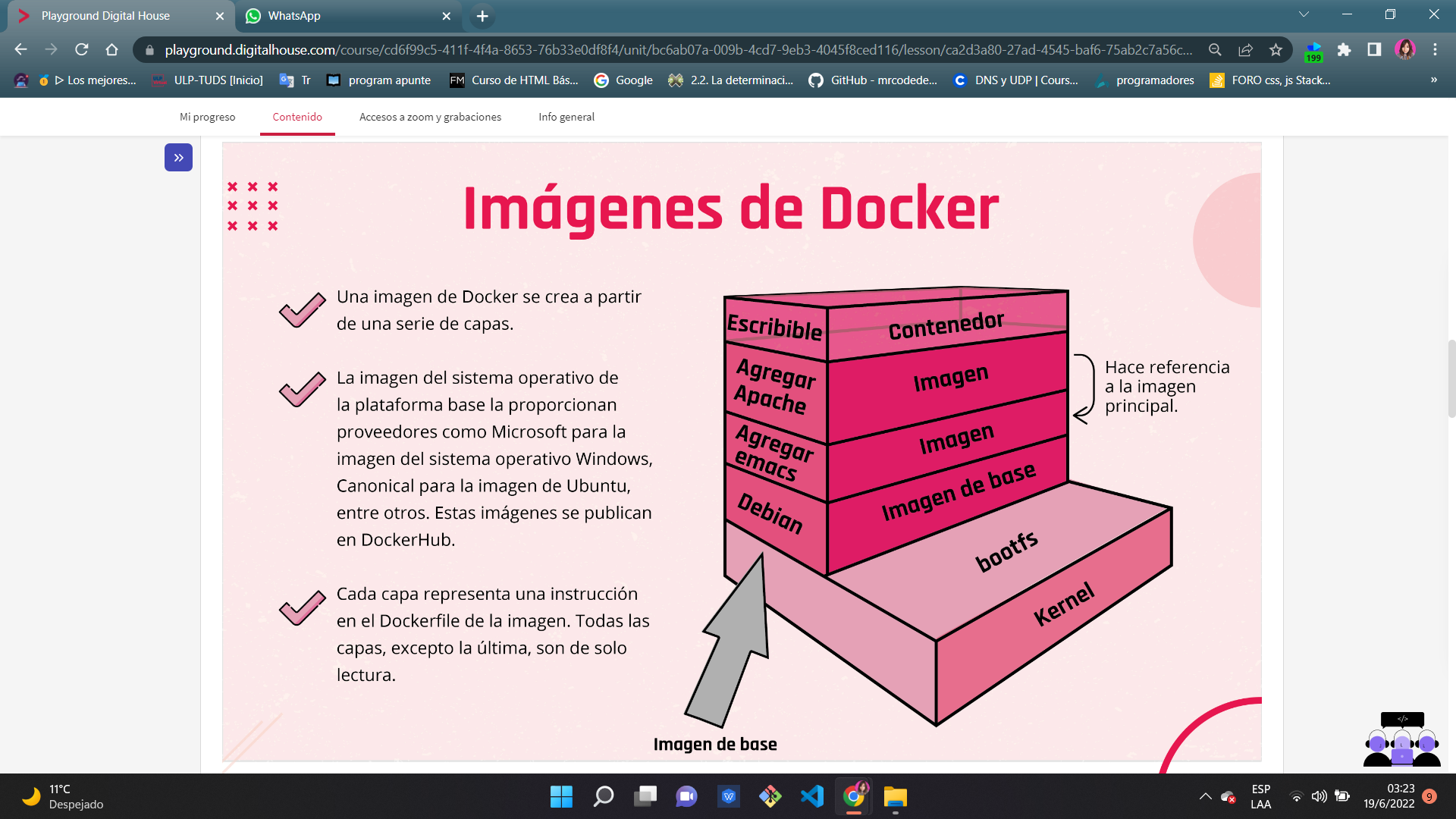
almacena imágenes Docker a usar en el sistema

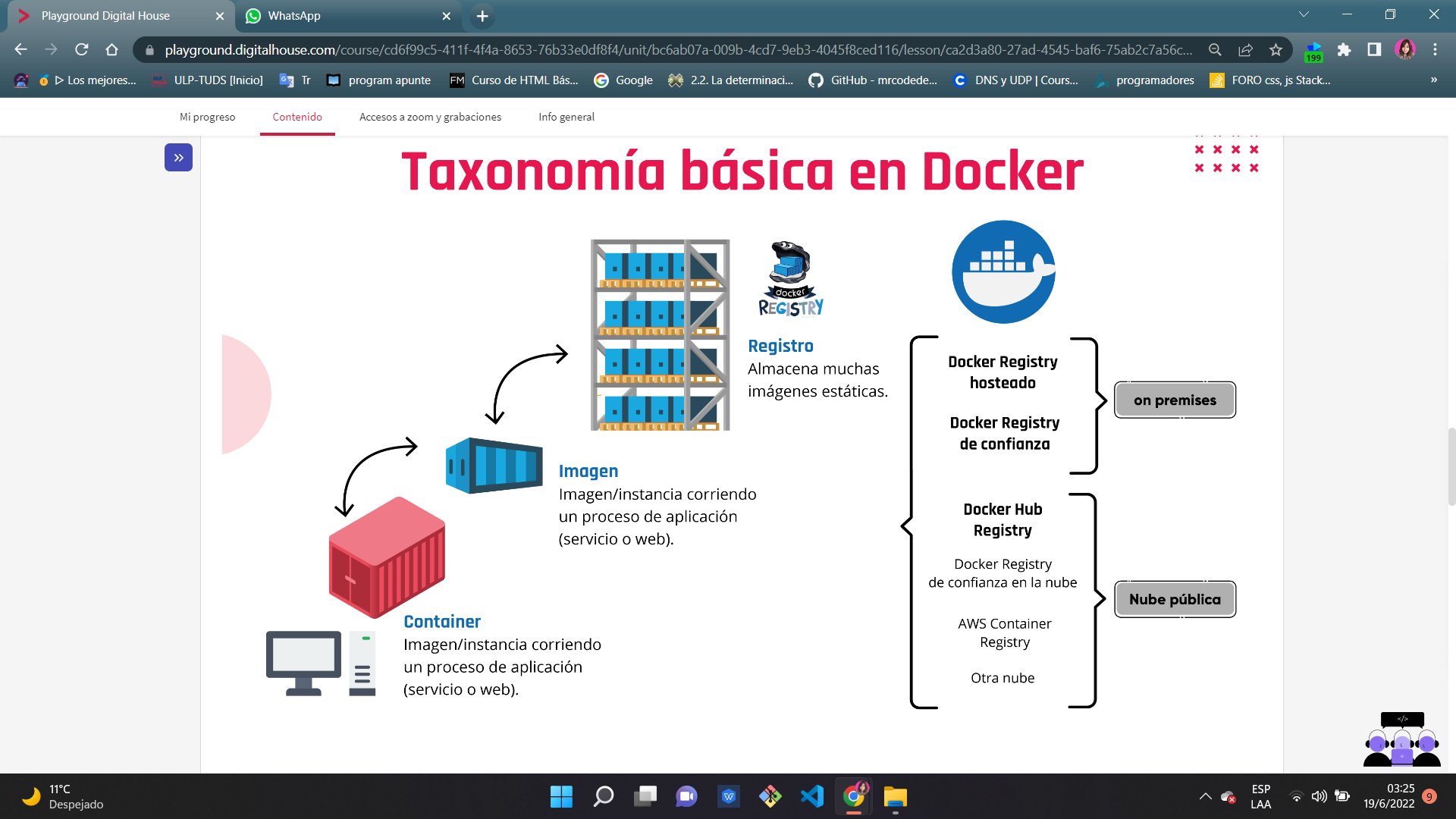
Imágenes almacenadas y disponibles para extraer y recopilar contenedores que ejecuten servicios web

Hay registros privados a nivel local y en la nube pública.

Pone imágenes en un registro y almacena fragmentos de la aplicación estáticos e inmutables.

Se pueden versionar e implementar en varios entornos y proporcionar una unidad de implementación coherente





*Docker File*

Archivo de texto simple con comandos o instrucciones

Ejecutados sucesivamente para hacer cambios en la imagen base para crear una nueva imagen en la ventana acoplada

El dockerfile construye una imagen de docker que corre en un contenedor docker

Docker lee las instrucciones y crea imágenes automáticamente

Capas de imágenes = instrucciones del dockerfile

Instrucciones básicas:

* De: define la imagen base que usa para inicar la construccion
* Correr: toma el comando con sus argumentos y los ejecuta desde la imagen
* CMD: igual que run, pero se ejecuta creado la instancia del contenedor.
* Punto de entrada: se dirige a su aplicacion predeterminada en la imagen cuando se crea el contenedor.
* Añadir: copia archivos de origen a destino
* ENV: crea variables de entorno

Etiquetas de imagen: Identifica versiones de la imagen.

Se listan en su tag o etiqueta asociada.

Pueden agrupar sus imágenes usando nombre y etiquetas, si la tiene, pone una abstracta

**Docker Hub**

Servicio de registro de repositorio.

Extrae y envía imágenes de contenedor

Repositorio en linea basado en la nube, repositorio privado o público.

Crea grupos de trabajo e impulsar repositorios privados

Integra repositorios de código fuente como GitHub

Construcciones automatizadas antes cambios en el código del repositorio, creando imágenes de contenedor.

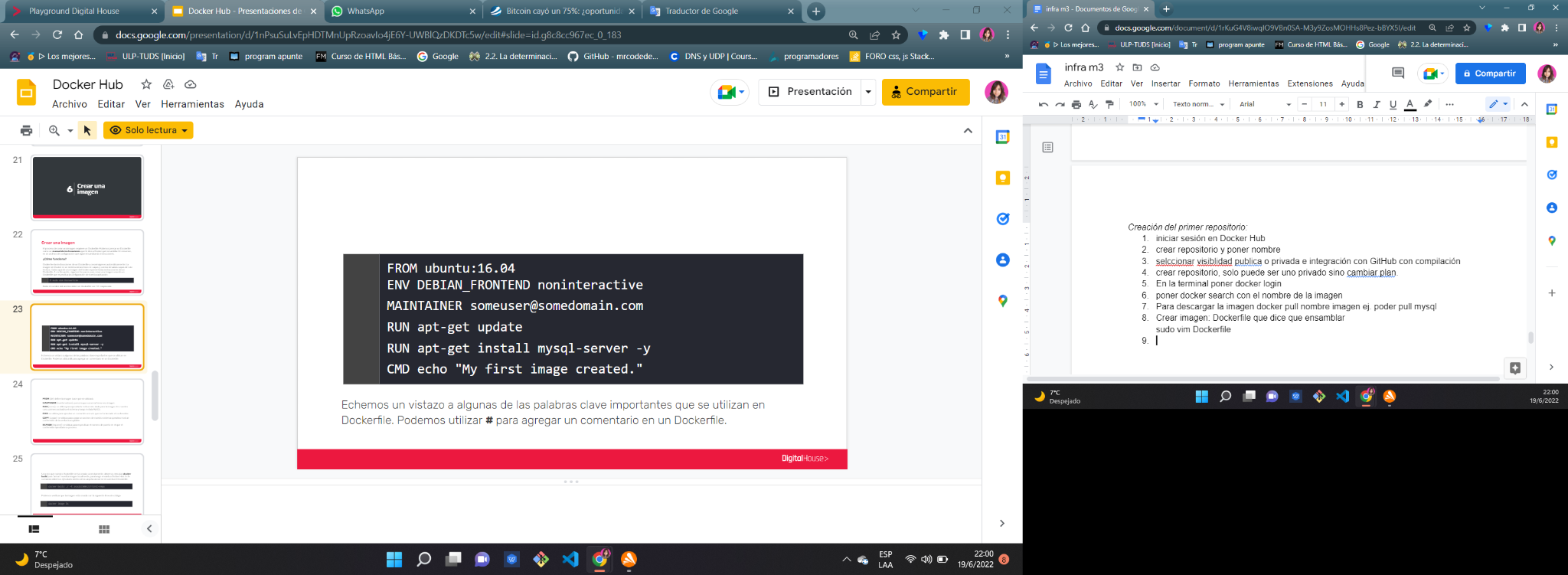
Con webhooks integra docker hub con otros servicios

Imágenes de alta calidad son oficiales y certificadas para extraer y utilizar.

*Creación del primer repositorio:*

1. iniciar sesión en Docker Hub
2. crear repositorio y poner nombre
3. seleccionar visibilidad publica o privada e integración con GitHub con compilación
4. crear repositorio, solo puede ser uno privado sino cambiar plan.
5. En la terminal poner docker login
6. poner docker search con el nombre de la imagen
7. Para descargar la imagen docker pull nombre imagen ej. docker pull bitnami/mysql
8. Crear imagen: Dockerfile dice a Docker que ensamblar

sudo vim Dockerfile

1. 

*From:* imágen base a utilizar

*Maintener:* persona que mantiene esa imagen

*Run:* ejecuta instrucción para imagen

*CMD:* ejecuta un comando lanzado el contenedor.

*Expose:* especifica el numero de puerto donde el contenedor ejecuta el proceso.}

-Comandos van en el dockerfile!-

1. Ejecutar docker build dentro del dockerfile para armar imagen local y después enviarla al docker hub. Ejemplo: docker build ./ -t asadali08527/first-repo
2. Verificar que la imagen que se creó con docker image ls
3. Para enviarla a Docker Hub: docker push. Ejemplo: docker push asadali08527/first-repo

Para evitar poner sudo a cada rato:

sudo usermod -aG docker ${USER}

sudo su - ${USER}

pide password

y empezara usar docker ps sin usar el sudo

*Imágenes certificadas*

Optimizadas y populares por la presencia en el mercado ,calificaciones, puntaje de satisfacción y más.

Depende del SO y su arquitectura

Buscar fecha de actualización y su tipo de editor verificado u oficial

**Docker Networking**

Redes de tipo brigde, locales y excluivas del host donde se crearon

Tipo de red por defecto

Simula creación de switche o hubs, de nivel 2

NO se debe utilizar, crear nuevas redes para usos específicos. Buena práctica

*Resolución de nombres*

Dentro del ámbito de la red misma, utiliza el nombre del container para identificarlo por medio del servicio DNS para redes creadas y containers asignados.

1. *docker network :*
   1. *create:* crea red
      1. *--internal* crea del tipo privada
   2. *connect:*conecta container existente a una red.
2. *docker run:*
   1. *--network <networkName>* nombre red que se conecta el container
   2. *--name:*  nombre del container para que funcione la resolución de nombres.
   3. *-p <hostPort>:<containerPort>* publica en ciertos puertos en los container del host para acceder a ese servicio.
   4. *-P*: public todos los puertos definidos en la especificación container(Dockerfile) en puertos aleatorios del host.

**Docker compose**

Simplifica el uso de docker a través de archivos YAML instruir al docker engine

Crea contenedores y en cada uno diferentes servicios, unirlos en volumen común, iniciarlos y apagarlos

Repite instrucciones en diferentes ambientes.

Construye aplicaciones y microservicios

\*YAML lenguaje de serialización de datos que suele utilizarse en el diseño de archivos de configuración

Beneficios:

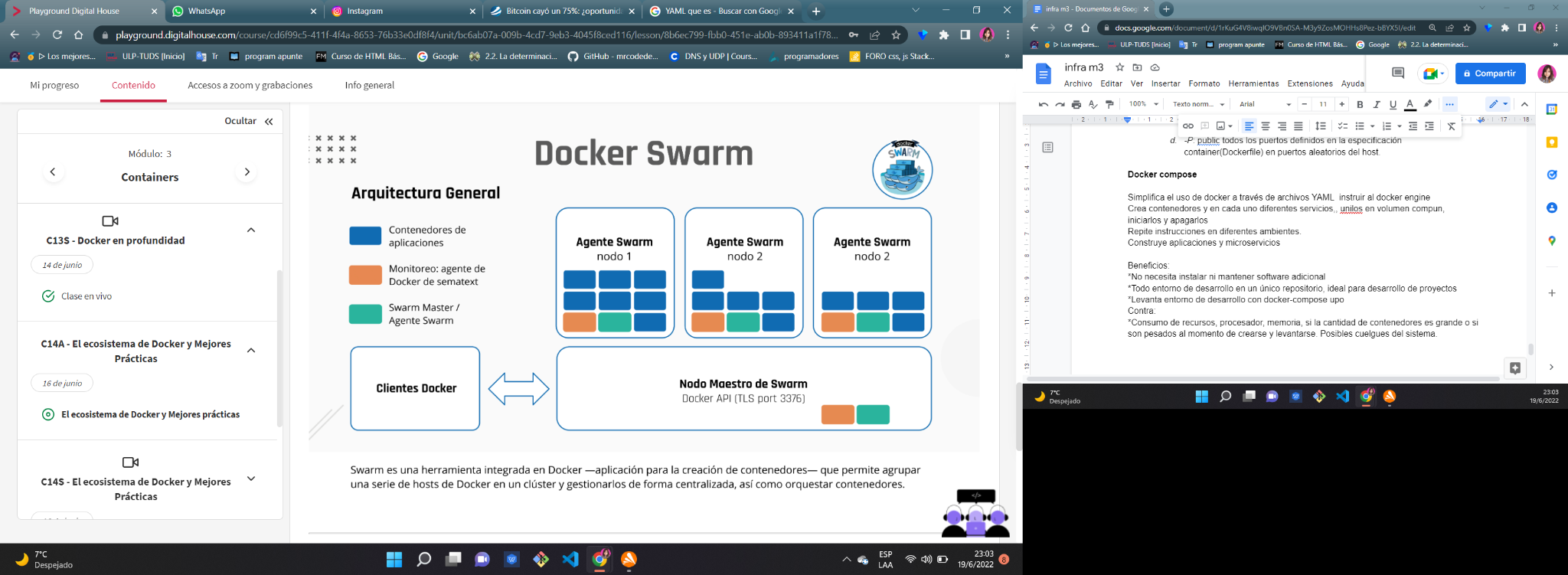
\*No necesita instalar ni mantener software adicional

\*Todo entorno de desarrollo en un único repositorio, ideal para desarrollo de proyectos

\*Levanta entorno de desarrollo con docker-compose upo

Contra:

\*Consumo de recursos, procesador, memoria, si la cantidad de contenedores es grande o si son pesados al momento de crearse y levantarse. Posibles cuelgues del sistema.



\*cluster:sistemas distribuidos de granjas de computadoras unidos entre sí normalmente por una red de alta velocidad y que se comportan como si fuesen un único servidor.

**Mejores prácticas para desarrolladores**

1. Usar imágenes oficiales:

Las imágenes no oficiales incluyen el nombre del usuario que las publicó, mientras que las oficiales, no. Veamos la diferencia a la hora de ejecutar “docker pull”.

Para una oficial sería: *docker pull ubuntu:latest*. Mientras que para una imagen que no lo es, el comando se vería así: docker pull usuario/nginx:latest.

1. Comando copy en vez de add

Add soporta copiar archivos de distintos orígenes.Muchas tareas hace. Genera confusión, inconvenientes técnicos y de performance

Copy, copia un archivo desde un file system local a la imagen. Facilita lectura del dockerfile.

Recomendable:

-Copy: para copiar archivos locales a la imagen en tiempo de compilación

-Run: para usar curl-Client URL- y mas comandos para bajar archivo de una fuente web

-Add para archivos .tar,para desempaquetar dentro de la imagen que se está construyendo.

**Serverless**

Modelo nativo de desarrollo en la nube

Modelo de función como servicio

No tiene ambiente de infra virtual para administrar

Factura cuando el código se ejecuta, reduce costos operativos

Proveedor de servicios: Preparación, mantenimiento y adaptación de servidores

Apps amplían según la respuesta a la demanda

Algunos servidores ofrecen según su ejecución o uso, si están inactivas no generan cargos

IaaS, infraestructura como servicio pago a nube publica por elementos del servidor que ejecute sus funciones

Aumenta eficiencia, productividad del desarrollador

Facilita adopción de enfoque DevOps, no describen infra necesaria para las operaciones

Como:

Gestionar el SO y de archivos y Ejecutar parches de seguridad,

Equilibrar la carga, administrar la capacidad

Llevar registros y supervisar el sistema

Ofertas serverless:

* Backend como servicio
  + Al desarrollador: acceso a apps y servicios externos
  + Ej.: servicio de autentificación, cifrado adicional BBDD
  + Funciones sin servidor, solicitadas mediante APIS
* Función como servicio: FaaS

Arquitectura para apps asíncronas y sin estado

Apps con gran contenido de imágenes y demanda

Casos de:

Flujos de datos entrantes, chatbots, tareas programadas y lógica empresarial

Apps webs, API de cack, autmatización de procesos empresariales, sitios web sin servidor

**Cloud computing**

Ventajas:

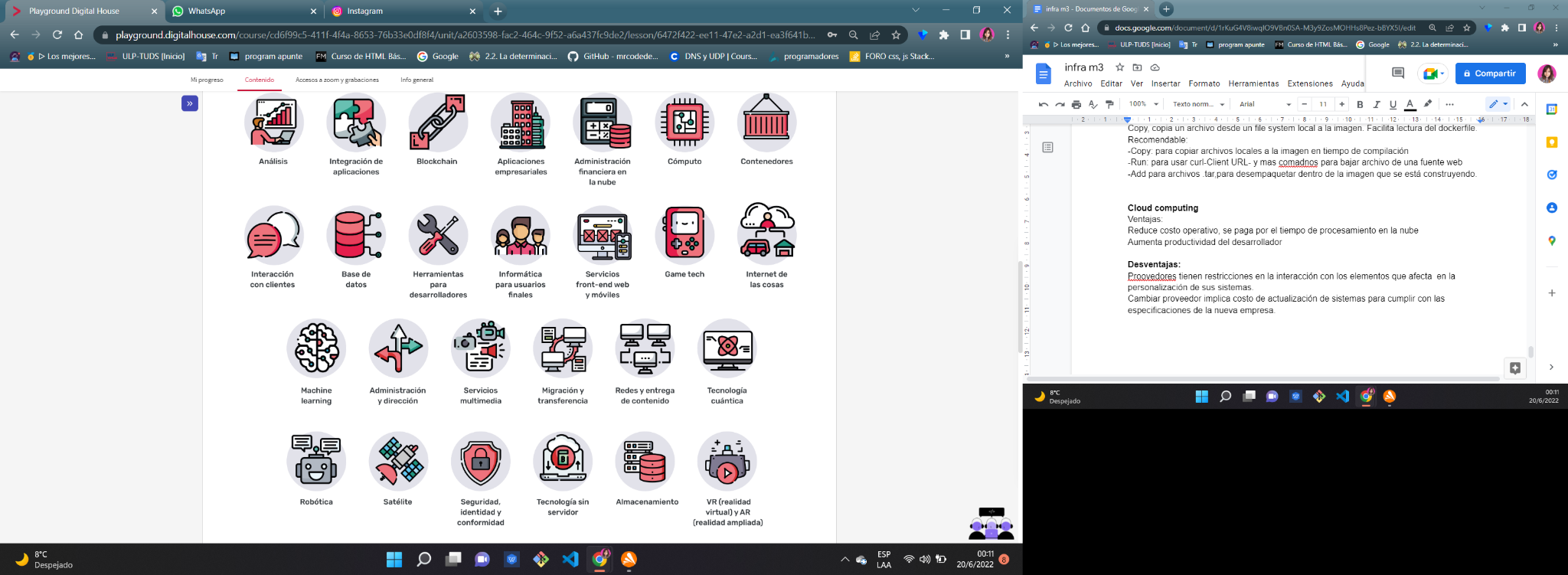
Reduce costo operativo, se paga por el tiempo de procesamiento en la nube

Aumenta productividad del desarrollador

**Desventajas:**

Proveedores tienen restricciones en la interacción con los elementos que afecta en la personalización de sus sistemas.

Cambiar proveedor implica costo de actualización de sistemas para cumplir con las especificaciones de la nueva empresa.



**Nube híbrida**

Arquitectura IT que incorpora gestión, organización y portabilidad de cargas de trabajo de dos o mas entornos.

Suelen incluir: al menos una nube privada y una pública, dos o mas nubes privadas igual que públicas

Entorno virtual o sin SO conectado a una nube.

Funciones:

Conectar varias pc mediante una red

Consolidar recursos IT

Escalar horizontalmente e implementar recursos con rapidez

Trasladar cargas de trabajo entre los entornos.

Incorporar una sola herramienta de gestión unificada

Organizar procesos

*Funcionamiento*

-Red LAN, red WAN(red amplia), VPN(red privada virtual) e API (interfaz de programación de aplicaciones) conectan varias pc entre sí.

-Virtualización, contenedores o almacenamiento extraen recursos agrupables en lagos de datos.

-Sistema de software de gestión asigna recursos a entornos donde apps pueden aplicarse

-Convierte en híbrida cuando los entornos se conectan, que permite que funcionen.

Bases del edge computing.

Determina forma en que se trasladan las cargas de trabajo, unifica la gestión y organiza los procesos.

*Arquitectura moderna*

Ejecutan el mismo SO en todos los entornos de IT, desarrollan e implementan aplicaciones como grupos de servicios pequeños, independientes y sin conexión directa y gestionan todo con una PaaS unificada

Extrae requisitos del sistema de hardware y la plataforma de organización todos los de las apps.

Entorno informático uniforme e interconectado, apps trasladan de un entorno a otro sin tener que mantener un mapa complejo del as API posible de fallar.

Trabajo conjunto entre equipos de desarrollo y operaciones en un modelo DevOps; entorno integrado en una arquitectura de microservicios compatible con los contenedores.

*Seguridad*

Igual que una infraestructura IT local,

Desafios: migración de datos, aumento de la complejidad y mayor superficie de ataque.

Entornos interconectados, defensa contra los riesos de seguridad

Posibilidad de elección de datos confidenciales segun los requisitos y equipos de seguridad

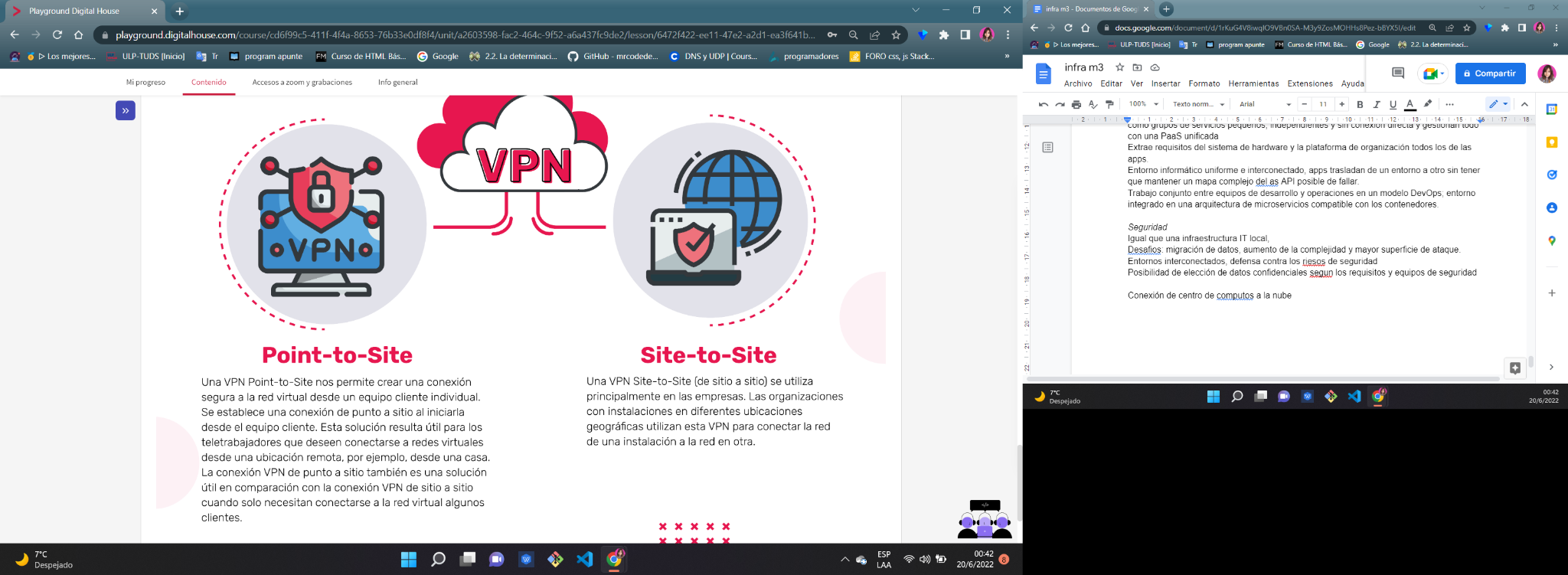
Conexión de centro de computos a la nube

Serveless

Tecnología sin servidor, contratar algo que funcione sin necesitar

es mas barato, se paga por uso, puede tener aproximandamente 1000 request por dia.

Evita consumo masivo de horas, infra y monta para que se use el servicio

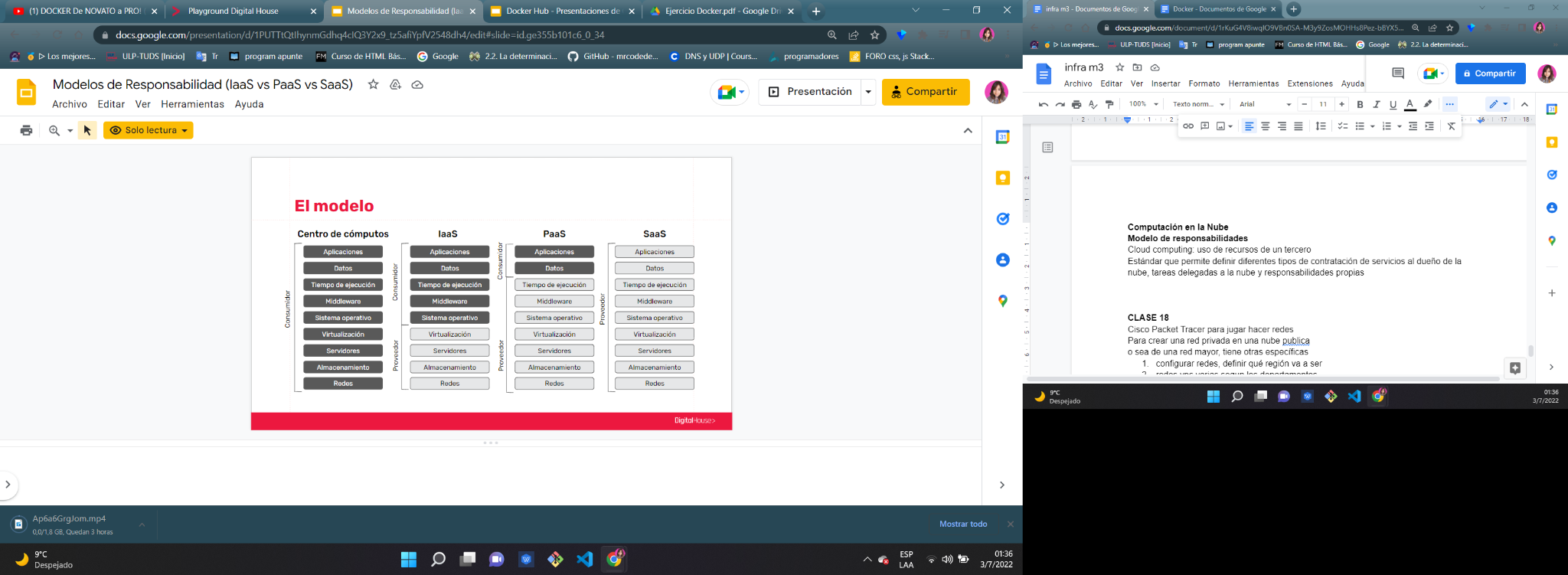


**Computación en la Nube**

**Modelo de responsabilidades**

Cloud computing: uso de recursos de un tercero flexible con interfaces definidas

Estándar que permite definir diferentes tipos de contratación de servicios al dueño de la nube, tareas delegadas a la nube y responsabilidades propias

****

*Proveedor:*

Dueño de la nube o centro de cómputos

*Centro de cómputos:*

Tareas que nos hacemos cargo si alojamos totalidad de servicios de tecnología sin contratar a nadie que administre el sistema

Pro: control total de activos, como usarlos, cuando y para qué

Contra: responsabilidad en salud, mantenimiento y utilización de todo, sin escala flexible.

*IaaS - Infra como servicio*

Instancia MV en su centro sin necesidad de administrar nosotros

Pro: control sobre SO, escalamiento de infra en forma dinámica

Contra: Responsabilidad de administración, salud de SO, apps y servicios sobre este.

*PaaS - Plataforma como servicio*

Ofrecimiento tecnología específica interfaz gráficas o APIS, gestión recursos subyacentes

Pro: Delegación de gestión, monitoreo y escalabilidad de la tecnología

Enfoque a la entrega de valor sin administración ni monitoreo de la infra

Contra: Caro,configuraciones limitadas por el proveedor.

Ejemplo:

Amazon DynamoDB

BBDD no estructurada como MongoDB

Consumo como un servicio, sin instanciar infra ni instalarla

Subscripción al servicio mediante AWS que provee:

-Interfaz web para configuraciones del servicio Dynamo

-API configura parámetros programáticamente

-Cdena de conexión para conectar la BBDD

*SaaS - Software como servicio*

Servicio de una app

Ejemplos: trello, salesforce o gmail

Pro: acceso a una app mediante tarjeta de crédito, sin costo de infra ni operación

Contra: nulo o poco control de sobre la configuración del servicio contratado

Adecuación a lo que el software permite hacer.

Sin control sobre las nuevas funcionalidades del sistema ni cuando se liberan

**MV en la nube**

Mayor recurrencia a MV y apps móviles

Proceso de creación mediante SO o programa de una versión virtual de un recurso tecnológico SO, periférico. disco de almacenamiento u otro que exista real.

Beneficios:

Un servidor conectado 24hs a internet para mostrar página web o servicios online de venta

Opciones:

Comprar un equipo y montar un servidor propio con los gastos extras que requiere conectarlo, configurarlo, instalar sistemas de seguridad, contra incedios, back ups.

Contratar un servicio para crear una MV como servidor para subir BBDD y programas para el negocio.

Acceso ilimitado, contratación y mantenimiento por AWS o Microsoft Azure.

**Como crear una MV en la nube**

*Amazon*

Amazon Web Services y el servicio Amazon Elastic Compute Cloud EC2

MV en la nube son instancias

Plantillas configurables con apps y configuraciones predefinidas de ram, número de procesadores y otros detales físicos.

MV en cualquier en SO como Linux SuSE, Linux Ubuntu y Red Hat Linux o Windows Server 2012.

*Azure*

Plataforma incluye instalación de un servidor web hasta BBDD y MV

*Ventajas*

Económica, se paga servicio a un proveedor que ofrece paquetes según las MV y las características a tener.

Menos gasto en hardware,software actualizado

Pago por el uso, accesibilidad desde cualquier lugar

Mayor rapidez de implementación, menor costo operativo

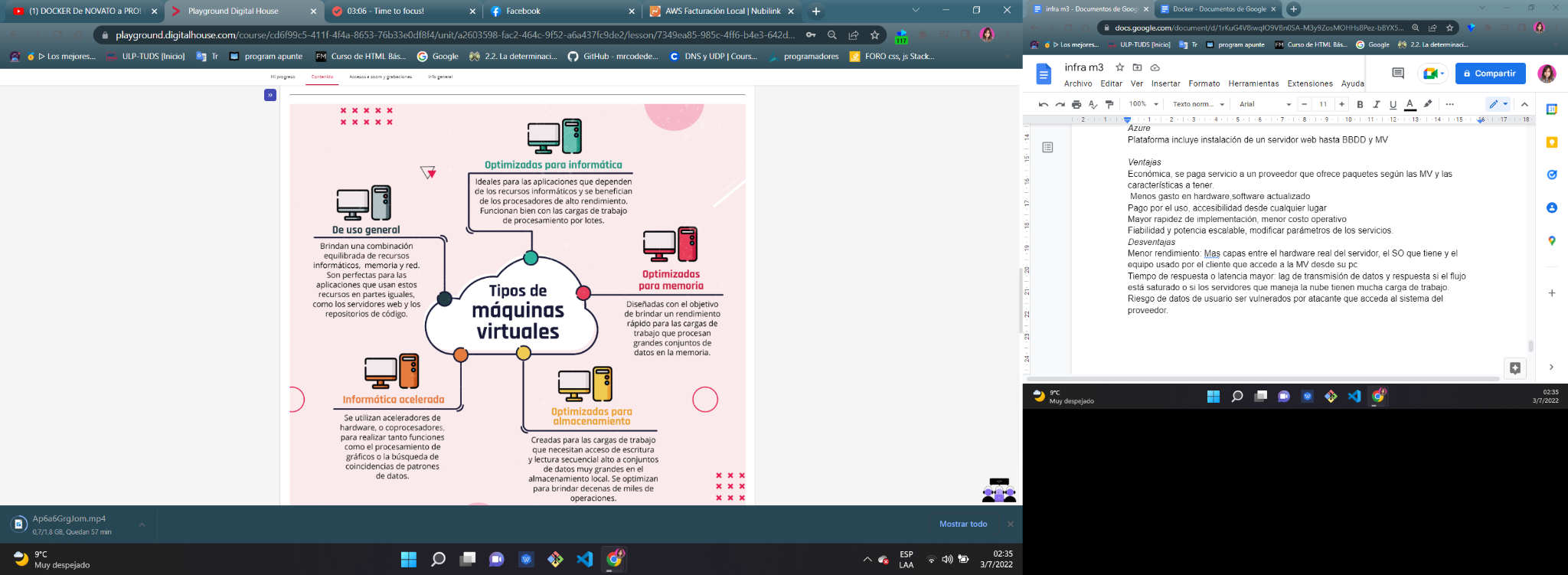
Fiabilidad y potencia escalable, modificar parámetros de los servicios.

*Desventajas*

Menor rendimiento: Mas capas entre el hardware real del servidor, el SO que tiene y el equipo usado por el cliente que accede a la MV desde su pc

Tiempo de respuesta o latencia mayor: lag de transmisión de datos y respuesta si el flujo está saturado o si los servidores que maneja la nube tienen mucha carga de trabajo.

Riesgo de datos de usuario ser vulnerados por atacante que acceda al sistema del proveedor.



**Escalabilidad, tolerancia al fallo y disponibilidad en la nube**

*Escalabilidad*

Capacidad de crecer en demanda, habilidad para incrementar su capacidad y reducir recursos se conoce como elasticidad, elástica es la infra que permite eso.

*Tolerancia de fallos*

Responde errores sin dejar de ser funcional, solucionar problemas o reemplazar elementos operados incorrectamente.

*Disponibilidad*

Múltiples recursos y uno falla

**Proveedores**

*Amazon EC2*

Servicio web capacidad informática segura y de tamaño modificable

Control completo sobre recursos informáticos, ejecutado en entorno acreditado de Amazon

Plataforma amplia con elección del procesador, almacenamiento, red, SO y modelo de compra

Instancias de GPU poderosas de machine learning y cargas de trabajo gráficas e instancias de costo por inferencia mas bajas de la nube.

*Microsoft Azure*

SO Windows y Linux, Windows Server 2003, configuración de ram y cpu asociada

Componentes:

Disco virtual con el SO instalado, placa de red virtual, direcciones IP privadas y públicas, grupos de seguridad de red -gestión de permisos de red para una o mas máquinas- definidos por el software

Parametizables no modificables:

Nombre de la MV y SO salvo operaciones avanzadas.

Cambio vertical en el hardware virtual, cpu, memoria, cantidad discos soportados, calidad de los componentes.

*Google Cloud*

Instancia creada en Google Cloud Console o api de Compute Engine

Imágenes de Linux o Windows Server como personalizadas privadas creadas o importadas de mi sistema existente

Implementación de contenedores de Docker en la imagen pública de Container-Optimized OS

Posibilidad de elegir o crear MV personalizadas

Cada instancia pertenece a un proyecto de Google Cloud Console, puede tener más instancias.

Posibilidad de agregar almacenamiento adicional si las apps requieren más espacio.

Secure Shell SSH o Remote Desktop Protocol RDP para configurar aplicaciones de las instancias.

**Creación de MV con Windows server 2016 y conexión remota a escritorio**

<https://drive.google.com/file/d/1dYNOpnR3a2-NuykejP1bIwx0oMfHC-H5/view>

**Ejecución instancia linux más servidor web**

https://drive.google.com/file/d/1Er5kFx\_zNhfczZdZuWpe1idSsUgOBmBA/view

**Cloud Computing: Redes (VPC+ ELB)**

Nube virtual privada dentro de una pública, utiliza sus recursos

Mantiene datos aislados de los de otras empresas, ambiente mas seguro

Se conecta con redes remotas por conexión de red privada virtual VPN

Tiene seguridad, privacidad y control

Ejecuta aplicaciones

Empresas las administran por panel de control del proveedor del servicio

*Beneficios*

Seguridad: Protege ambiente de redes virtual como dire IP, subredes y puertas de enlace de red

Ejemplo, DB en una subred privada sin acceso a internet

Control de datos: está aislada, control de datos que se mezclen con otros de otras empresas-peligro de las nubes públicas-

Rendimiento: Posible priorizar el tráfico de red de ciertas aplicaciones para optimizar el rendimiento

Flexibilidad bajo demanda: Diseño de arquitectura a las necesidades de la empresa.

**Concepto de Elasticidad versus Escalabilidad**

*Escalabilidad*

Aumentar o disminuir los recursos en tamaño o cantidad

Muchos servicios de AWS son escalables de forma predeterminada

*Elasticidad*

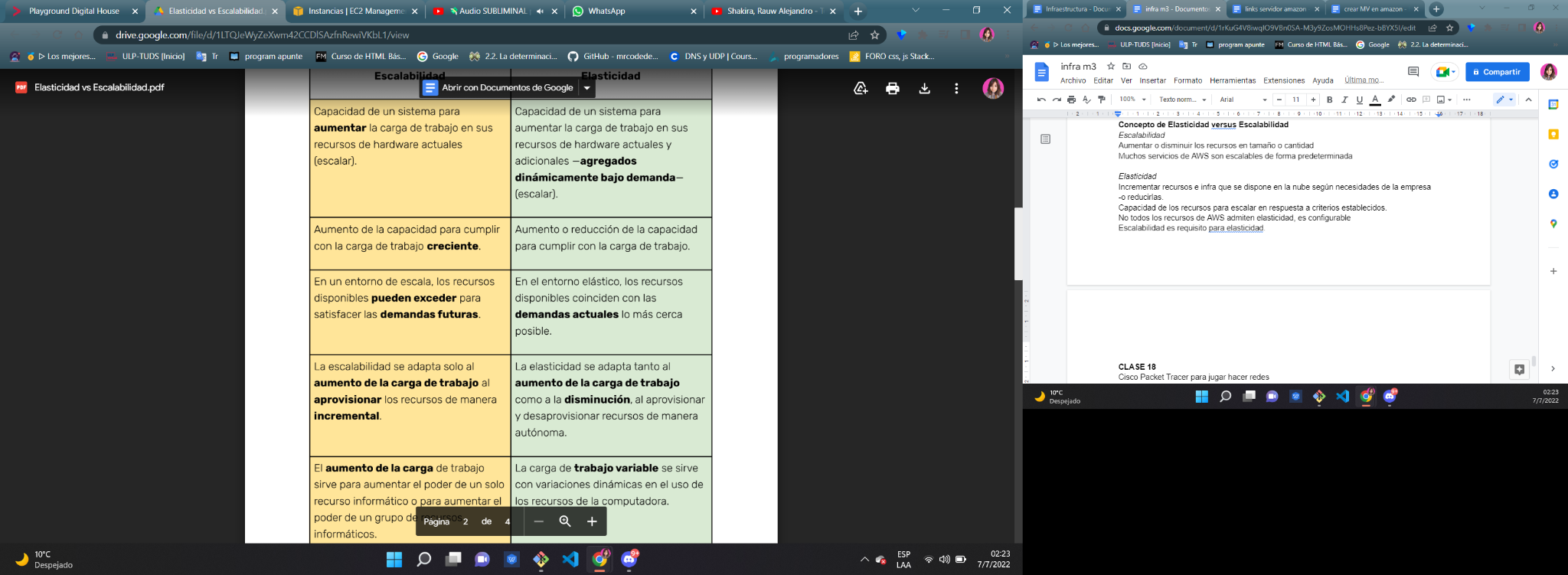
Incrementar recursos e infra que se dispone en la nube según necesidades de la empresa -o reducirlas.

Capacidad de un sistema para aumentar (o disminuir) su capacidad

de cómputo, almacenamiento, trabajo neto, etc.

No todos los recursos de AWS admiten elasticidad, es configurable

Escalabilidad es requisito para elasticidad.



**VPC en AWS**

Amazon Virtual Private Cloud da control completo sobre las redes virtuales como la ubicación de los recursos, conectividad y la seguridad.

Crear VPC, agregar recursos como instancias de Amazon Elastic Compute Cloud EC2

y Amazon Relational DB Service RDS

Definir cómo se comunican entre sí, zonas de disponibilidad o regiones

*Características*

* Analizador de accesibilidad:

Analiza y depura la accesibilidad de red entre dos recursos en la VPC

Especificados los recursos de destino y origen, este da detalles salto por salto de la ruta virtual

* Registros de flujos:

Da visibilidad operativa de las dependencias de red y patrones de tráfico, detecta anomalías, filtraciones de datos y solución de problemas de configuración y conectividad de la red

Archiva registros de flujo para cumplir con requisitos de conformidad

* Replicación de tráfico de la VPC:

Copia tráfico de red de una interfaz de red elástica para la inspección de paquetes

Detecta anomalías de seguridad y de red, información operativa, controles de seguridad y conformidad y solucionar problemas.

* Grupos de seguridad:

firewall para instancias EC2 asociadas, controla tráfico de entrada y salida

Una instancia puede tener más de un conjunto de grupo de seguridad.

* Lista de control de acceso de red o ACL:

Capa de seguridad opcional, actúa como firewall, controla tráfico de una o mas subredes.

*Otros recursos:*

* Transit Gateway: conecta vpc, cuentas y redes a una sola gateway
* Private Link: conectividad privada entre vpc y servicios alojados sin exponer datos
* Network Firewall: seguridad en las redes
* VPN: extiende redes en las instalaciones a la nube y acceder de cualquier lugar
* Client VPN: aumenta o disminuye según requisitos.
* Site to Site VPN: conexión segura entre centro de datos y recursos de la nube
* Gateway de traducción de direcciones de red NAT, cargas de trabajo de subred privada acceda a internet, evita que internet inicialice conexión con ellas.

**C20A Pequeño ambiente en AWS**

**Load balancer**

Balanceador de carga útil a páginas o apps con mucho tráfico

Crea grupo de servidores que atiende las solicitudes que recibe un servicio, página web, mail, web server

* Distribuye solicitudes de clientes o la carga de red eficiente en varios servidores
* Garantiza disponibilidad y confiabilidad al mandar solicitud a servidores en línea
* Flexibilidad de agregar o quitar servidores según requiera la demanda

Beneficios: Escalabilidad, redundancia. eficiencia

Optimizar prestaciones globales de la infra, su rendimiento y su capacidad.

*Elastic Load Balancing -*Load balancer de AWS

Distribuye tráfico de apps por varios destinos como: Instancias EC2, contenedores, dire IP, funciones Lambda y dispositivos virtuales.

*Balanceo de carga*

Distribución eficiente del tráfico de red entrante por grupo de servidores o granja

Enruta solicitudes de clientes a través de todos servidores

Maximiza velocidad y capacidad, que no haya sobrecarga por parte alguno que baje rendimiento

**Creación de buckets en AWS**

<https://drive.google.com/file/d/1VAzseOeZyuO0ZhK0gCYjcFwerrtf9nC4/view>