|  |  |
| --- | --- |
| **TÍTULO:**  Inicio del curso  **ASIGNATURA:** Fundamentos de docker  **FECHA:** 15/05/2021  **MODULO Nº:** 1 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.)  Los tres grandes problemas de desarrollo de softaware profesional   1. Construir. 2. distribuir. 3. Ejecutar.   Artefactos son la parte del codigo compilado de nuestro codigo.   La ejecucion del codigo no es la misma donde se construño nuestra app | **APUNTES O NOTAS DE CLASE**  Resumen de la clase:  “Docker te permite construir, distribuir y ejecutar cualquier aplicación en cualquier lado.”  **Problemáticas del desarrollo de software**  **1. Construir -** Escribir código en la máquina del desarrollador. (Compile, que no compile, arreglar el bug, compartir código, etc. )  **Problemática:**   * Entorno de desarrollo (paquetes) * Dependencias (Frameworks, bibliotecas) * Versiones de entornos de ejecución (runtime, versión Node) * Equivalencia de entornos de desarrollo (compartir el código) * Equivalencia con entornos productivos (pasar a producción) * Servicios externos (integración con otros servicios ejem: base de datos)   **2. Distribuir** - Llevar la aplicación donde se va a desplegar (Transformarse en un artefacto)  **Problemática:**   * Output de build heterogeo (múltiples compilaciones) * Acceso a servidores productivos (No tenemos acceso al servidor) * Ejecución nativa vs virtualizada * Entornos Serverless   **3. Ejecutar** - Implementar la solución en el ambiente de producción (Subir a producción) El reto Hacer que funcione como debería funcionar  **Problemática:**   * Dependencia de aplicación (paquetes, runtime) * Compatibilidad con el entorno productivo (sistema operativo poco amigable con la solución) * Disponibilidad de servicios externos (Acceso a los servicios externos) * Recursos de hardware (Capacidad de ejecución - Menos memoria, procesador más debil) |
| **RESUMEN**  Docker es un sistema de contenedores que aisla una aplicación de un entorno de desarrollo, esto permite que ejecucion distrubucion y correcto funcionamiento se permita en cualquier tipo de dispotivo. | |

|  |  |
| --- | --- |
| **TÍTULO:** Virtualizacion  **ASIGNATURA:** Fundamentos de docker  **FECHA:** 15/05/2021  **MODULO Nº:** 1 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.)  Virtualizacion   Maquinas virtuales | **APUNTES O NOTAS DE CLASE**  Virtulizacion es : la version virtual de algun recurso, que hace; la virtualizacion ataca los tres principales probelmas de desarrollo de software profesional   Problemas de maquina virtual:   Peso dentro del disco duro  necesidad de mantenimiento igual que una maquina fisca  multipes formatos de archivos  Docker soluciona los problemas narrados con anterioridad con la estandrizacion de contenedores, esto para que haya una manera estandar de desarrollar un app |
| **RESUMEN**  Es importante destacar que docker es una herramienta que no va permitir contruir una aplicación sin ningun en el futuro de compatibilidad con maquinas diferentes  lo otro es que docker gasta mucho menos recuros que cualquier otro sitemas de virtualizacion que exite | |

|  |  |
| --- | --- |
| **TÍTULO:** Que es y como funciona docker  **ASIGNATURA:** Curso docker platzi  **FECHA:** 16 7 05 /2021  **MODULO Nº: 1** | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.)  Arquitectura de docker en orden de importancia.   1. Docker Deamon 2. Rest api. 3. Docker cli.   Entidades de docker:   1. container 2. image 3. data volume 4. network | **APUNTES O NOTAS DE CLASE**   1. Base de docker y hace que docker cumpla sus funciones; ademas es elpunte de comunicacin entre nuestra maquina y docker. 2. Puente de comunicación entre docker y nosotros, ademas noscuminicamos mediante el protocolo http. 3. Cliente docker, por defecto de docker es el que habla directamente con docker deamon, esto es atraves de linea de comando.   Entidades:   1. la parte mas importatne de docker y es donde estara alojada nuestra aplicación ademas es el corazon de docker. 2. Empaqueta nuestro contenedores para poder utilizalo dentro de cualquier otra distribucion de docker. 3. La forma en al que docker nos permite acceder al sistema de archivos de nuestra maquita anfitriona, con la mayor seguridad posible. 4. La forma con la cual los contenedores se comunican entre si y con el mundo esterior |
| **RESUMEN**  Destaco: el container en docker son el corazon del por que exite docker y su principal funcionalidad  podemos cumunicarnos con diferentes contendores a treves del network, y la comunicacion entre nostros y docker deamon va ser a traves de docker CLI esto va ser mediante el protocolo http | |

|  |  |
| --- | --- |
| **TÍTULO:** conceptos Fundamentales: contenedores  **ASIGNATURA:** Curso docker platzi  **FECHA:** 16 / 05/ 2021  **MODULO Nº:** 2 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.)  Contenedor es una unidad logica | **APUNTES O NOTAS DE CLASE**  Conceptos fundamentales de los contenedores  Un contenedor es una agrupacion de procesos que correren nativamente en la maquita pero estan aislados del resto del sistema.  El contenedor corre de manera nativa dentro de la maquina anfitriona.  Puede contenedor correr nuestra aplicación sin ningun problema pero dentro de los limites que nosotros le demos. |
| **RESUMEN**  A fines practicos un contenedor es una maquina vitual que aisla el software, pero a difrencia de las VM son mas libianos y utilizan el kernel y SO de la maquina anfitriona, ademas la manera de lanzar un contendor es a traves de el siguiente comando **docker run** algunos otros comandos en este [Link](https://collectednotes.com/barckcode/docker-cheat-sheet) | |

|  |  |
| --- | --- |
| **TÍTULO:** El estado de docker  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 2 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.) | **APUNTES O NOTAS DE CLASE**  Comandos que nos ayudan a trabajar con contendores   $ docker run hello-world (corro el contenedor hello-world) $ docker ps (muestra los contenedores activos) $ docker ps -a (muestra todos los contenedores) $ docker inspect <containe ID> (muestra el detalle completo de un contenedor) $ docker inspect <name> (igual que el anterior pero invocado con el nombre) $ docker run –-name hello-platzi hello-world (le asigno un nombre custom “hello-platzi”) $ docker rename hello-platzi hola-platzy (cambio el nombre de hello-platzi a hola-platzi) $ docker rm <ID o nombre> (borro un contenedor) $ docker container prune (borro todos lo contenedores que esten parados)      Docker run siempre crea un nuevo contenedor y lo ejecuta   docker no permite crear dos contenedores con el mismo nombre dentro de la misma instalacion |
| **RESUMEN** | |

|  |  |
| --- | --- |
| **TÍTULO:** Ciclo de vida de un contenedor  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 2 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.) | **APUNTES O NOTAS DE CLASE**  El que determina si un contenedor sigue vivo o no es el main proceso  el contenedor va estar vivo y funcionando siempre y cuando el main proceso con el ide uno este corriendo  matar un contenedor en docker:  docker kill my\_container  para eliminar un contenedor podemos con el nombre o con el container id  con del comando docker run crearmo el contenedor con start lo lanzamos y con stop lo detenemos |
| **RESUMEN** | |

|  |  |
| --- | --- |
| **TÍTULO:** Exponiendo contenedores  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 2 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.)  Recordemos que un servicio vivo no puede estar en mismo puerto al mismo tiempo  Los contenedores tienen su propio sistema de red  docker rm -f MyContainer frena y borra el contenedor | **APUNTES O NOTAS DE CLASE**  Cunado damos docker ps en nuestra terminal sale info de nuestros contenedores vivos, hay una casilla que dice port, ese port nos sera el puerto donde este nuesto contendor en la maquina anfitriona si no  Podemos seleccionar el el puerto por donde se va mostrar nuestro contenedor dentro de la maquina anfitriona de la siguiente forma:   1 2 3 4 5 docker run --name proxy -p 8080:80 nginx   1. Comando de arranque. 2. Nombre del contenedor. 3. Indica que se configure los puertos 4. indica el puerto donde se va lanzar dentro de la maquina anfitriona y el puerto interior del contendor. 5. Imagen.   si le entro -d dentro del comando que ejecutemos, el output no se va conectar automaticamente  para verlos despues necesitamos docker logs --tail 10-f myContainer  con tail le indicaremos que solo queremos ver las ultimas 10 |
| **RESUMEN**  Es importante tener en cuanta que para exponer un contenedor a nuestra paquina anfitriona debemos indicaler en que puerto quiere que estes   ademas de destacar la manera en la que podemos estar ejecutando el proceso principal de un contenedor por debajo con la bandera -d | |

|  |  |
| --- | --- |
| **TÍTULO:** Binde mounts  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 3 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.)  $ docker exec -it db bash (entro al bash del contenedor) | **APUNTES O NOTAS DE CLASE**  Binde mounst copia lo que esta pasando en el contenedor dentro de un directorio en la maquina anfitriona  primero debemos montar un directorio dentro del contenedor que deseemos  con el siguiente comando creamos el contenedor y indicamos que directorio estara montado al contenedor.  1 2 3 4 5 6  docker run -d --name db -v directorio:/data/MyComponent mongo   1. comando de arranque 2. ejecuenta en backgraund 3. nombre de contenedor 4. Binde mounts con voumen 5. nombre del directorio donde se va guarda la infor del contenedor 6. nombre de la carperta donde estara la info del contenedor dentro del mismo. |
| **RESUMEN**  Este metodo se utliza para compartir data entre el contenedor y la maquina que ejecuta el contenedor. Basicamente enlaza una carpeta que tenemos en nuestro disco con una carpeta que tengamos dentro del contenedor de esta maner acopiamos los datos que genere el contenedor en esa caperta a la carpeta que tengamos en nuestro disco  esto de peligroso por que el contenedor puede tener acceso nuestra informacion | |

|  |  |
| --- | --- |
| **TÍTULO:** Volumenes  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 3 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.)  Volumenes otra forma de manejar datos de contedores en docker  los volumenes pueden ser compartidos entre contendores  estos volumenes son manejados solo por docker | **APUNTES O NOTAS DE CLASE**  Los volumenes fueron una evolucion que docker implemento para darle mas seguiridad a las personas que desarrollan en entornos productivos, es importante destacar que la parte que afectan los volumnes en el disco es manejada por docker y nosotros como usuarios no tenemos acceso a ella   este es el comando que vamos a utilizar para montar un volumen dentro de un contenedor  1 2 3 4 5 6 docker run -d --name db --mount src=dbdata,dst=/data/db mongo.   1. Comando de arranque. 2. Ejecuta en background 3. nombre del contenedor 4. indica que vamos a montar un volumen 5. nombre del volumen 6. nombre la carpeta que vamos a enlazar con el volumen |
| **RESUMEN**  La ventaja de este metodo es que nadie va saber que esta dentro del volumen nisiquiera nosotros lo que significa que todos los archivos dentro de un volumen estaran protejidos y solo seran accedidos por el contenedor | |

|  |  |
| --- | --- |
| **TÍTULO:** Diferencia entre volumenes y binde mounst  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 3 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.) | **APUNTES O NOTAS DE CLASE**  Las siguientes son las principales diferencias entre los volumenes y los binde mount  Volumenes  Binde mounts |
| **RESUMEN** | |

|  |  |
| --- | --- |
| **TÍTULO:** insertar y estraer archivos en un contenedor  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 3 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.)  No hace falta que el contenedor esta corriendo o no seva poder usar el cp de cualquier modo | **APUNTES O NOTAS DE CLASE**  Este el comando que se utiliza para agregar una archivo desde nuestro sistema a nuestro contenedor.  1 2 3 4  docker cp prueba.txt copytext:/testin/test.txt   1. comando de inicio que indica la tarea que vamos a realizar. 2. Nombre de el archivo que vamosa copiar 3. nombre del contenedor 4. dirreccion donde alejaremos el archivo en el contenedor, ademas de eso podemos cambair el nombre del archivo al momento de cambiarlo   con este comando copiamos desde nuestro contenedor a nuestro disco local    1 2 3 docker cp copytest:/testin localtestin   1. accion que va a realizar , 2. contenedor desde donde va realizar la copia y el archivo o directorio que vamos a copiar 3. la dirrecin donde vamos a alojar la copia |
| **RESUMEN**  Las opciones en docker para compartir datos | |

|  |  |
| --- | --- |
| **TÍTULO:** Imagenes  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 4 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.)  El campo tag en la imagenes indica la version de la misma imagen  con la imagenes e busca solucionar el problema de construcion y de distribucion de software | **APUNTES O NOTAS DE CLASE**  Imagenes = plantillas desde las cuales docker crear contendores, estas imagenes contienen todo lo necesario para que un contenedor pueda ejecutarse  para utilizar imagenes necesitamos el subcomando image. Ej  1 2 docker image ${options}   1. subcomando de docker para trabajar con imagenes. 2. Opciones que puede llevar este subcomando   imágen Una imágen contiene distintas capas de datos (distribución, diferente software, librerías y personalización). . Podemos llegar a la conslusión, que una imágen se conforma de distintas capas de personalización, en base a una capa inicial (base image), la dicha capa, es el más puro estado del SO. . La siguiente ilustración nos mostraría la representación gráfica, del concepto de una imágen en Docker.  Si observamos, partimos desde la base del SO, y vamos agregando capas de personalización hasta obtener la imágen que necesitamos:   1. distribución debian 2. se agrega el editor emacs 3. se agrega el servidor Apache 4. se agregan los permisos de escritura para la carpeta /var/www de Apache   Hay que tener en cuenta, que todo parte del Kernel de Linux, en caso de utilizar alguna distrubución de Linux |
| **RESUMEN**  Recordemos que la imagen es la base que utiliza docker para crear contenedores  ademas que es la solucuion para la contruccion y distrinucion de software | |

|  |  |
| --- | --- |
| **TÍTULO:** Construyendo una imagen propia  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 4 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.)  Docker file archivo que describe que pasara con una imagen cuando se cree  la imagen es la plantilla con la cual se crean contenedores, ademas con esa sola imagen se pueden crear infinitos contenedores  todas la imagenes son un conjunto de capas  proceso al momento de retaggear una imagen en docker | **APUNTES O NOTAS DE CLASE**  Con docker file se basa docker para crear una imagen  las imagenes son un conjunto de capas, estas capas son instrucciones que estan dentro de nuestro archivo Dockerfile  por general el dirrectorio que tiene el dokeFile es el mismo donde se crear las imagenes    1 2 3 4  docker tag ubuntu:platzi 1033812288/ubuntu:platzi   1. comando de docker para trabjar con tags 2. nombre del software base 3. nombre del dueeño del software 4. nombre del software y version del software   este esl proceso para retaggear una nueva imagen en docker |
| **RESUMEN**  Recordemos que:   * la capass de las imagenes de docker son inmutables * la imagen es la pantilla de un contenedor * las capas son intrucciones que tiene nuestro Dockerfile * con el archivo Dockerfile creamos imagenes * link de los comandos https://carbon.now.sh/qQzEbCYvxyJuNTm4hRtQ | |

|  |  |
| --- | --- |
| **TÍTULO:** Sistema de capas y docker para desarrollar apps  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 4 &5 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.)  Docker aprovecha las capas ya construidas para no volver a realizarlas  al momento de realizar cun cambio en docker file afecta el proceso de build y lo vuelve a generar | **APUNTES O NOTAS DE CLASE**  Podemos entender como esta construida una imagen a partir de su docker file  cada capa solo es un pequeño cambio respecto al anterior, pues la capas son inmutables y después que se crearon no se podrá modificar de ninguna manera  por eso no es bueno borrar un archivo que se creo en una capa anterior con una nueva capa, ya que se pierde espacio  Docker commit tiene un funcionamiento muy parecido al commit de git.  Guarda el estado de la capa mutable en un tag de la imagen a partir de la cual se genero el contenedor creando una capa mas para crear uno o mas contenedores a partir de esta nueva imagen. |
| **RESUMEN**  Url de los comando utilizados en clase  https://carbon.now.sh/hsXWISqAgQimneCqcQTq | |

|  |  |
| --- | --- |
| **TÍTULO:** docker para desarrollar apps  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 5 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.)  ¿docker-compose.yml? Dependecias de docker | **APUNTES O NOTAS DE CLASE**  **Usando docker para desarrollar apps**  en esta clase he de destacar que vimos una primera instancia de como crear una imagen personalidad para nuestro entorno.  En este docker file, pudimos instalar dependencias a nuestro proyecto en que puerto iba salir, o donde ira nuestro proyecto y demás  adjunto imágenes de como puede estar nuestro proyecto dockerfile  }  Ejemplo de Dockerfile |
| **RESUMEN** | |

|  |  |
| --- | --- |
| **TÍTULO:** Aprovechando el caché de capas para estructurar correctamente tus imagenes  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 5 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.) | **APUNTES O NOTAS DE CLASE**  Para estructurar de una manera correcta nuestro proyecto se aconseja primero cpiar los archivos de dependecias primero segudio descargar nuestras dependecias y despues copiar el resto del codigo  esto para que se aproveche el sistema de cache de docker, este sistema es inteligente ya que si ve que ya existe algun archivo similar y sin ningun cambio generar ninguna copia   1. comando de ejecucion 2. subcomandos, el rm borra el contenedor cuando deja de ejecutarse, p puertos, d background 3. nombre contenedor 4. Binde mounts, dirrecion del archivo a copiar, donce va copiar ese archivo dentro de contenedor 5. nombre imagen |
| **RESUMEN**  Imagen de los comandos utlizados  <https://carbon.now.sh/Uc6xPK7XTvKq5fAb1or7>  bien interesante como el sistema de cache de docker funciona de tal manera que no repetie los archivos ya copiados dentro de sus contenedores | |

|  |  |
| --- | --- |
| **TÍTULO:** docker networking colaboracion entre contenedores  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 5 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.) | **APUNTES O NOTAS DE CLASE**  Para conectar contentores entre contendores se utliza las redes de docker  redes docker:   * bridge: red por default de docker y esta presente por retrocompatibilidad * host : repesentacion de docker de la red real de mi maquina * none: red espeacial para contendores que deseemos que no tengan acceso a red   lista de comandos utlizados en clase   si conectamos contenedores a una misma red docker tiene la cualidad de contectandolos automaticamente, eso si se ejecutamos variables de entorno |
| **RESUMEN**  Lista de comandos  https://carbon.now.sh/Y5Cv1SJbS80NBIYYNTvf | |

|  |  |
| --- | --- |
| **TÍTULO:** docker compose: la herramienta todo en uno  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 6 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.)  Dos procesos no pueden escuchar el mismo puerto en la misma maquina  un servicio puedo tener uno mas contenedores de la misma imagen  docker-compose up vanta el servicio y crea los contendores | **APUNTES O NOTAS DE CLASE**  Docker-compose agrupa todos los comandos que hemosestado agupando anteriormente.  Docker-compse describe de forma declarativa la arquitedtura de servicios que nuestra app necesita, come se comunica entre si, cuales son su dependecia etc.  pero todo esto pase en un pequeño archivo llamdo docker-compose.yml |
| **RESUMEN**  La facilidad de docker compose es que no vamos a ser todo manualmente como lo veniamos haciendo  link de la imagen de la descripcion de docker-compose.yml  https://carbon.now.sh/RoVKMXVg8pghI9N5cFBH | |

|  |  |
| --- | --- |
| **TÍTULO:** subcomandos de docker-compose  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 6 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.) | **APUNTES O NOTAS DE CLASE**  Comando de docker-compose |
| **RESUMEN**  Link de los comandos de docker-compose  https://carbon.now.sh/GwjLVFue7hnJo5pqmGfk | |

|  |  |
| --- | --- |
| **TÍTULO:**  docker-compose como herramienta de desarrollo  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 6 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.) | **APUNTES O NOTAS DE CLASE**  Hasta el momento nuestra imagen ha estado construida podemos contruirla desde nuestro proyecto.  Ingresando a nuestro archivo dockercompose.yml y cambiando la imagen de algun servicio que deseemos por un build y como contexto le mandamos la ubcacion del codigo. Ejemplo :  build .  Hecho el cambio enviamos un docker-compose build para crear las imagenes de nuestro proyecto  para realizar cambios a nuestra imagen debemos hacer un buildm, ademas podemos escoger que servicio le va hacer build.  Para espejar nuestro archivos, debemos crear un volumen en docker para que el pueda ver los cambios que estan sucediendo este proceso en con bind mount  esto copia todo lo que esta en mi carpeta a la carpeta src y exclulle la carpeta node modules de de cualquier cambio  ademas de eso le pasamos el comando de compilacion de nuestro framework o tecnologia. Ej  esto por ejemplo para node |
| **RESUMEN**  La unico problema de este metodo de monitorio es que vamos a modificar nuestro compose file, esto se puede ir dentro de un commit y afectar a nuestro grupo de trabajo, pero sirve mucho para andar monitoriando nuestro condigo y que este se vea reflejado en el contenedor. | |

|  |  |
| --- | --- |
| **TÍTULO:** Compose en equipo: override  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 6 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.)  Docker-compose nacio como proyecto open-source llamado fig , pasado el tiempo fue adquirido por docker y lo renombro como esta hoy en dia   fig =docker-compose | **APUNTES O NOTAS DE CLASE**  Docker para equipos  Compose override: personaliza el composefile pero en un archivo diferente sin dañar ni afectar el original al momento de realizar un commit  este dicho archivo tendra este docker-compose.override.yml este archivo tendra la misma informacion que el archivo docker-compose.yml pero podremos modificarlo sin ningun miedo   de hecho seria un consejo agregar el nuevo archivo a gitignore para no generar problemas  contra indicacion, hay una cosas como pro ejemplo ports que solo debe estar definido en un archivo  como en docker podemos hace scalas, podemos simular que estas realizando peticiones desde mas de un servicio, esto se hace escalando o generando mas de una aplicación dando y un rago de puertos de ejecucion para no generar erorres |
| **RESUMEN**  Comandos de la clase   https://carbon.now.sh/8gGpxHtgSM6j6IXcKs5J | |

|  |  |
| --- | --- |
| **TÍTULO:** Administrando tu ambiente de Docker  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 7 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.) | **APUNTES O NOTAS DE CLASE**  Matenimiento de espcaio personal en docker  puedes limitar la memoria ram de cada contenedor para simular equipos con poca memoria, si dicho contenedor no tiene la memoria suficientes muere automaticamente |
| **RESUMEN**  Link de los comandos vistos en clase  https://carbon.now.sh/X7FqrMFevsAXXW9kZezQ | |

|  |  |
| --- | --- |
| **TÍTULO:** Deteniendo contenedores correctamente: SHELL vs. EXEC  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 6 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.) | **APUNTES O NOTAS DE CLASE**  Si un proceso se demora mas de un tiempo puede ser indicadores de error  si el codigo que lanza al final de la app es mayor a 128 indica error  la momento de escribir nuestro docker file, hay que escribir nuestro archivp en formato  exec que va entre llave los valores    este dockerignore ignorara todos los documentos que deseeo que no esten al momento de hacer build y crear la imagen |
| **RESUMEN** | |

|  |  |
| --- | --- |
| **TÍTULO:** docker para desarrollar apps  **ASIGNATURA:** Curso de docker de platzi  **FECHA:** 16 / 05 /2021  **MODULO Nº:** 6 | |
| **IDEAS CLAVE**  (Ideas principales, palabras clave, preguntas, etc.) | **APUNTES O NOTAS DE CLASE** |
| **RESUMEN** | |