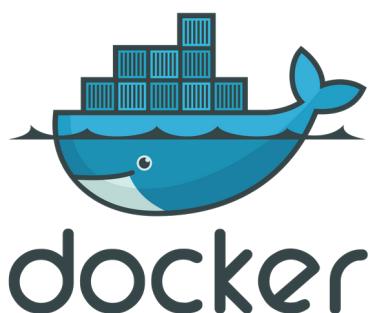


Desplegament d'aplicacions web amb Docker



Índex

Introducció	3
Objectius	3
Desenvolupament de l'aplicació	4
1. Estructura de l'aplicació en NodeJs.....	4
2. Funcionalitat de l'aplicació en NodeJs	4
Docker	7
1. Introducció	7
2. Història	7
3. Diferència entre les Màquines virtuals i Docker.....	8
4. Contenidors de Docker.....	9
5. Integració de Docker	11
6. Sistemes operatius suportats a Docker	11
7. Eines de Docker	12
7.1 Docker Engine.....	12
7.2 Docker Hub	13
7.3 Docker Compose	14
7.4 Kitematic	15
7.5 Dockerfile	16
Kubernetes	17
1. Introducció	17
2. Història	17
3. Components	17
Part Pràctica	19
1. Instal·lació de Docker	19
1.1 MacOS	19
1.2 Windows.....	20
1.3 Linux	22
2. Instal·lació de Kubernetes.....	24
2.1 MiniKube en Windows.	24
2.2 MiniKube en Linux.....	25
2.3 Minikube en MacOS	25
3. Expressions en Docker.....	25
3.1 Creació d'un Dockerfile a partir de l'aplicació.....	29
3.2 Creació d'un docker-compose a partir de l'aplicació	32
4. Kubernetes	35
4.1 Llançament d'una aplicació	36
Conclusions	45
Bibliografia	46
Annex Entorn de Validació	47
Annex Presentació.....	49

Introducció

Actualment en el món de desenvolupament d'aplicacions, siguin web o multiplataforma, es pot requerir més complexitat que no tan sols escriure codi. Com per exemple: múltiples llenguatges, utilització de diferents frameworks, arquitectures, etc. Docker simplifica i accelera el flux de treball mentre permet als desenvolupadors tenir llibertat per innovar amb altres eines.

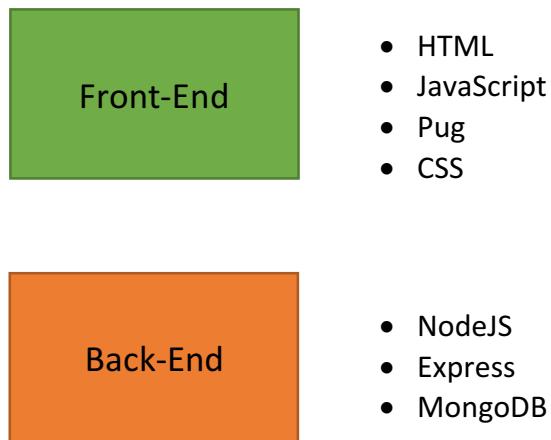
Objectius

Els objectius principals d'aquest projecte són els següents:

- Aprofundir en les diferències entre maquines virtuals i contenidors.
- Utilització bàsica de contenidors amb Docker.
- Desplegament d'una aplicació feta en NodeJs amb Docker i Docker Compose.
- Desplegament d'una aplicació feta en NodeJs amb Kubernetes.

Desenvolupament de l'aplicació

1. Estructura de l'aplicació en NodeJs



2. Funcionalitat de l'aplicació en NodeJs

És un mini-joc online per a 2 participants a la vegada. Aquest té un sistema d'autenticació dels usuaris, aquests es guardaran en una base de dades no relacional feta en MongoDB.

Aquesta base de dades, a part de tenir totes les credencials dels usuaris, també tindrà un recompte de partides guanyades de cada jugador.

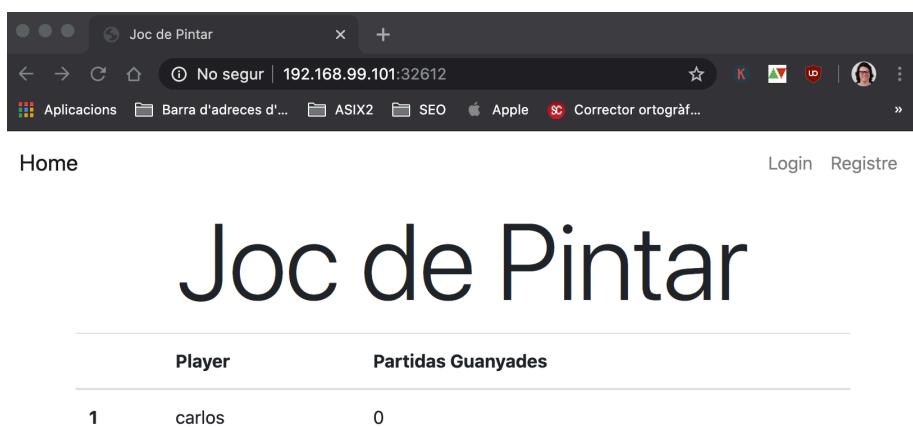
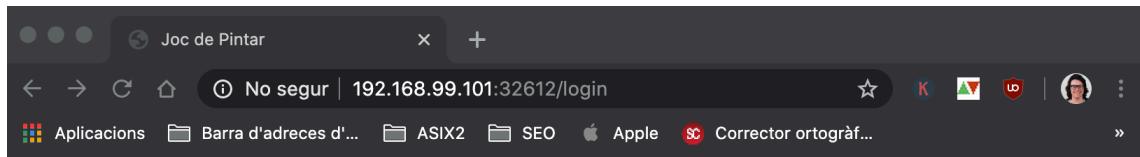


Figura 1. Pàgina principal de la aplicació.



Home

Login Registre

Login

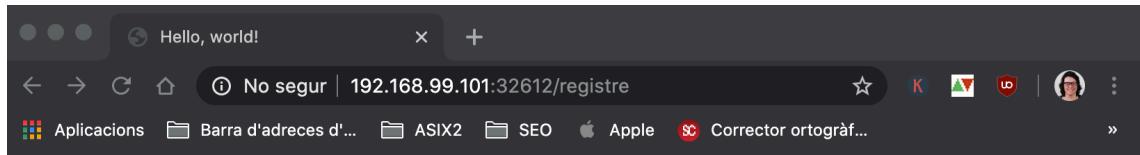
Nom

Contrasenya

Log In

No estas registrat? Registre't [aqui!](#)

Figura 2. Pàgina de Login, on els usuaris ja registrats podran accedir a jugar



Home

Login Registre

Registre

Nom

Contrasenya

Repeixeix Contrasenya

Registra't

Figura 3. Pàgina de registre, on els usuaris nous podran fer el seu registre per poder accedir al portal.

Les pàgines de Login i registre estan connectades amb la base de dades.

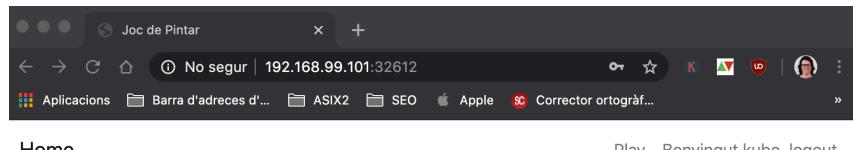


Figura 4. Pàgina de Benvinguda al usuariloguejat

Un cop l'usuari s'hagi registrat, accedirà aquesta pàgina.

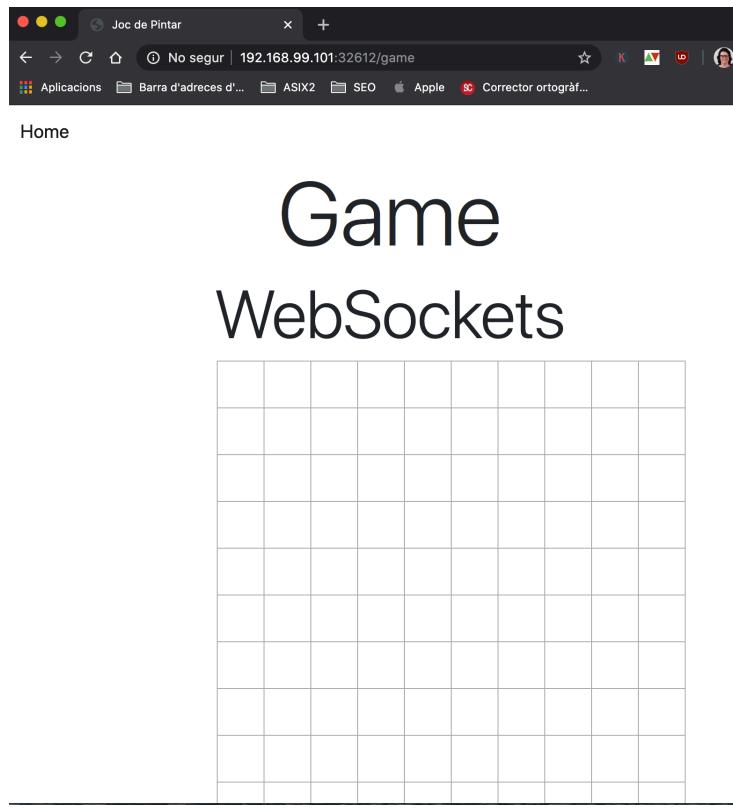


Figura 5. Pàgina on es permet jugar al joc desenvolupat.

El codi de desenvolupament de l'aplicació es pot trobar en directori Aplicació (del lliurement de projecte).

Docker

1. Introducció

Docker és una plataforma per a desenvolupadors y administradors de sistemes per desenvolupar, implementar y executar aplicacions en contenidors.

És un projecte de codi obert que permet automatitzar el desplegament d'aplicacions desenvolupades en diversos llenguatges o API's. El seu desplegament es fa dins de contenidors que proporcionen una capa addicional d'automatització de virtualització en diferents sistemes operatius com ara en Linux, Windows, MacOS...

Al permetre aïllar cada aplicació desplegada en diferents contenidors garanteix un aïllament de recursos del nucli de Linux, permetent l'execució per separat de diversos contenidors en una única instancia Linux. Això garanteix que no hi hagi una sobrecàrrega de creació de varies maquines virtuals per poder desplegar aplicacions.

2. Història

El seu creador va ser Salomon Haykes juntament amb altres contribuents. Va començar com un projecte intern de dotCloud, una empresa dedicada al Paas. Docker va ser alliberada el març del 2013, i un any mes tard va deixar d'utilitzar LXC com entorn d'execució per defecte i el va substituir per la seva pròpia biblioteca “libcontainer” escrit en Go.

3. Diferència entre les Màniques virtuals i Docker

El concepte inicial és bastant similar al que defineix una màquina virtual, però no són completament iguals.

Un contenidor és molt més lleuger que una maquina virtual, ja que en ella s'ha d'instal·lar un sistema operatiu per el seu correcte funcionament. En canvi en el contenidor Docker s'utilitza el sistema operatiu que té la màquina host en la que s'està executant aquest contenidor.

Els contenidors s'executen (nativament) en Linux i comparteixen el kernel de la màquina hoste juntament amb altres contenidors. S'executa un progrés discret, no tenint més memòria que qualsevol altre executable, per això es lleuger.

Per altre banda, una Mànica virtual executa un sistema operatiu complet amb accés virtual a recursos de la maquina host a mitjançant el hipervisor. En general, les màquines virtuals proporcionen un entorn amb més recursos que necessita l'aplicació en concret.

A l'hora de seleccionar Màniques virtuals o Docker per tenir una aplicació en execució, la opció més viable és Docker ja que, com s'ha esmentat anteriorment, no requereix d'un consum de recursos elevat per la seva execució. També té la facilitat de poder ser exportat i distribuït entre màquines.

També comentar que el temps de càrrega dels contenidors Docker en comparació amb les Màniques virtuals és més baix, ja que en les Màniques virtuals s'ha d'arrencar tot un sistema operatiu des de zero.

Containers vs. VMs

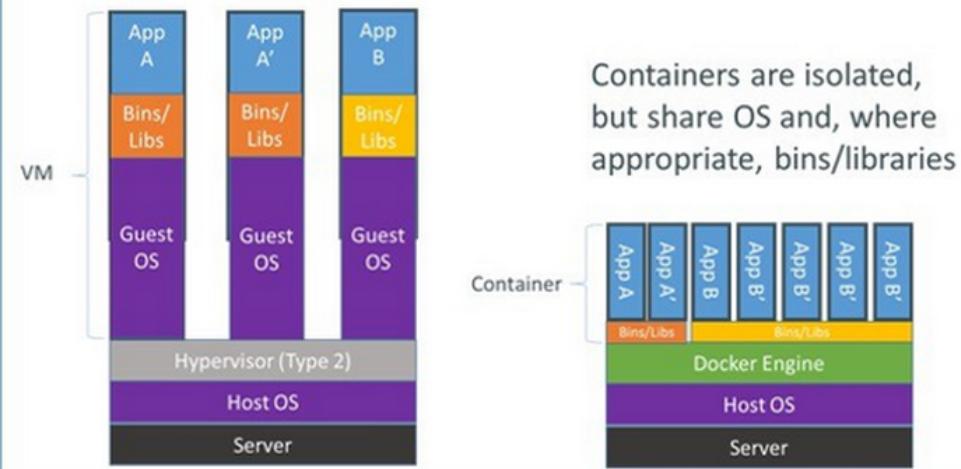


Figura 6. Diferència entre Docker i màquines virtuals

4. Contenidors de Docker

Les característiques que ens proporcionen els contenidors de Docker són les següents:

- Flexible: pot executar qualsevol aplicació, des d'un simple frase "Hola món", fins la aplicació més complexa que tingui una empresa.
- Lleuger: els contenidors aprofiten i comparteixen el kernel del host.
- Actualitzable: es poden implementar actualitzacions sobre la marxa.
- Portable: es pot construir localment, implementar al cloud i executar en qualsevol lloc.
- Escalable: pot augmentar i distribuir automàticament rèpliques del propi contenidor.

Un contenidor s'inicia executant una imatge, creada o descarregada. Una imatge és un paquet executable que inclou tot el necessari per executar el contenidor amb la seva aplicació. Més endavant veurem com crear una imatge des d'una aplicació web que hem desenvolupat nosaltres mateixos. Aquesta imatge conté el codi de l'aplicació, les llibreries que es necessiten per poder fer que tot funcioni correctament, variables d'entorn, arxius de configuració...

Quan executem un contenidor, aquest es converteix en memòria. Podem veure tots els contenidors que tenim executant-se des del propi DockerDashboard (si el tenim instal·lat) o des del terminal del nostre ordinador executant el següent comandament:

```
~ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
0b9792f60c9c	websokets-master_app	"docker-entrypoint.s..."	24 hours ago	Up 6 seconds	0.0.0.0:3000->3000/tcp	node-websokets

Figura 7. Contenidors executant-se vist des de terminal

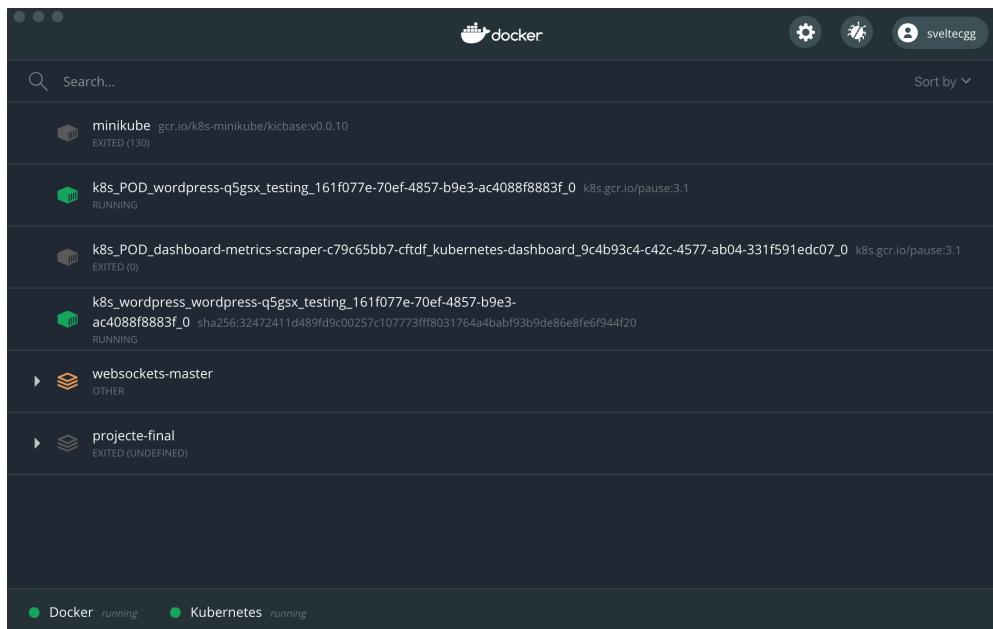


Figura 8. Contenidors executant-se vist des de DockerDashboard

Docker és genial com a entorn de proves, ja que tenim total llibertat de crear tants contenidors i eliminar tants contenidors com vulguem, parar-los i tornar-los a executar. Per exemple podem associar un volum a un contenidor en particular, eliminar el contenidor i tornar-lo a aixecar quan nosaltres requerim d'ell amb el mateix volum que li hem associat. Això ens permet la persistència de dades.

Utilitzant Docker per crear i gestionar contenidors pot simplificar la creació de sistemes altament distribuïts, permetent múltiples aplicacions, les tasques dels treballadors i altres processos per funcionar de forma autònoma en una única màquina física o varies. Això permet que el desplegament de nodes es realitzi a mesura que es disposen recursos o quan en siguin necessaris més. Docker també permet simplificar la creació i el funcionament de les tasques de carregues de treball o cues i altres sistemes distribuïts.

5. Integració de Docker

Es pot integrar docker amb diferents eines d'infraestructura, com per exemple:

- Amazon Web Services
- Ansible
- Chef
- Google Cloud Platform
- Digital Ocean
- Microsoft Azure
- Jenkins
- Vagrant
- Salt

6. Sistemes operatius suportats a Docker

Actualment Docker és suportat per una amplia varietat de sistemes operatius, ja siguin d'escriptori, servidors o cloud.

Sistemes d'escriptori:

- Linux
- MacOS
- Windows 10

Sistemes operatius de servidor:

- Ubuntu Server
- Fedora
- CentOS
- Debian
- Altres

Sistemes Cloud:

- Amazon Web Services
- Azure
- Google Cloud

7. Eines de Docker

7.1 Docker Engine

És un dimoni que executa Docker dins del sistema operatiu, ja que Docker està basat en Linux. Se'l pot denominar Docker Server o Docker Client.

Si el sistema operatiu de la màquina host no està basat en Linux, és necessari tenir una maquina virtual amb Linux que aquest tindrà el dimoni de Docker (que és la part Server). La part client si que es pot executar en qualsevol sistema operatiu.

L'usuari no interactuarà directament amb el dimoni, ja que ho farà mitjançant el client.

7.2 Docker Hub

Docker Hub és un servei en el cloud que facilita la construcció i la distribució d'aplicacions o serveis de contenidors. Ofereix les següents característiques específiques:

- **Repositori de imatges:** permet buscar, administrar, distribuir i obtenir imatges oficials de la comunitat.
- **Construccions automatitzades:** crea automàticament noves imatges quan s'ha realitzat qualsevol canvi en el codi de desenvolupament en una font GitHub o Bitbucket.
- **WebHooks:** permet automatitzar les construccions de les imatges.
- **Organitzacions:** crea grups de treball per gestionar l'accés a usuaris als repositoris de cada imatge.
- **Integració Github/Bitbucket:** permet afegir les imatges penjades al Docker Hub als seus fluxes de treball

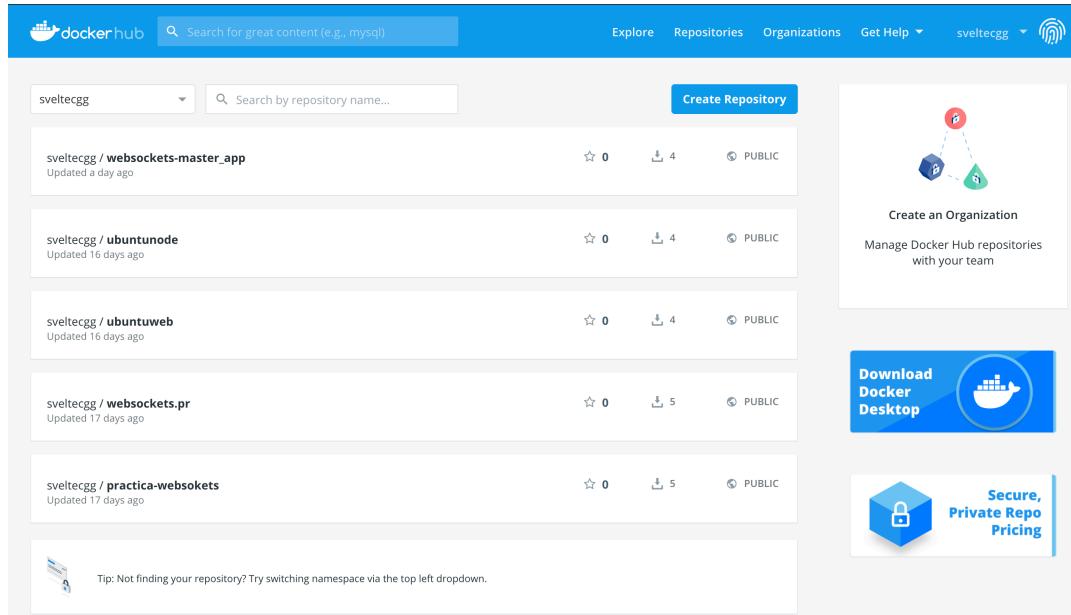


Figura 9. Captura de pantalla dels meus repositoris de Docker Hub

7.3 Docker Compose

Permet definir aplicacions en varis contenidors des d'un sol fitxer amb les mateixes característiques que indicaríem en un comandament "*docker run*" . Amb un sol comandament podem iniciar tots els contenidors necessaris per l'execució d'una aplicació i en l'ordre que nosaltres prèviament li hem especificat.

Aquest fitxer no només ens pot servir per crear contenidors, sinó per tenir una mínima documentació de com s'ha fet un deploy de l'aplicació, en quants contenidors està dividida, volums, enllaços, ports...

El fitxer *docker.compose* és un fitxer de format *yml* en el que especificarem els diferents contenidors i les seves propietats. El fitxer ha de ser anomenat *docker-compose.yml* i en funció del que vulguem fer podem realitzar les següents instruccions:

- *\$ docker-compose up*: amb aquest comandament arranquem en ordre els contenidors definits.
- *\$ docker-compose ps*: permet veure l'estat dels contenidors i en quins estan compostades la aplicació.
- *\$ docker-compose stop*: permet parar els contenidors que estan en marxa.
- *\$ docker-compose restart*: permet fer un restat dels contenidors que formen l'aplicació.
- *\$ docker-compose rm*: permet eliminar completament els contenidors que formen l'aplicació.
- *\$ docker-compose logs*: mostrarà els fitxers logs dels contenidors que formaran l'aplicació.

```

docker-compose.yml
1   version: '3'
2   services:
3     app:
4       container_name: node-websokets
5       restart: always
6       build: .
7       command: npm start
8       ports:
9         - '3000:3000'
10      links:
11        - mongo
12    mongo:
13      container_name: mongo
14      image: mongo
15      ports:
16        - '27017:27017'

```

Figura 10. Exemple de fitxer docker-compose.yml.

Més endavant veurem que significa cada sentència del fitxer.

7.4 Kitematic

És un projecte de codi obert que ens ajuda a simplificar i racionalitzar l'ús de Docker. Automatitza el procés d'instal·lació i configuració dels contenidors proporcionant una interfície intuïtiva gràfica d'usuari.

Un cop instal·lat, Kitematic llança i permet des de la pròpia pantalla d'inici poder gestionar les imatges instal·lades i quins contenidors tenim executant-se.

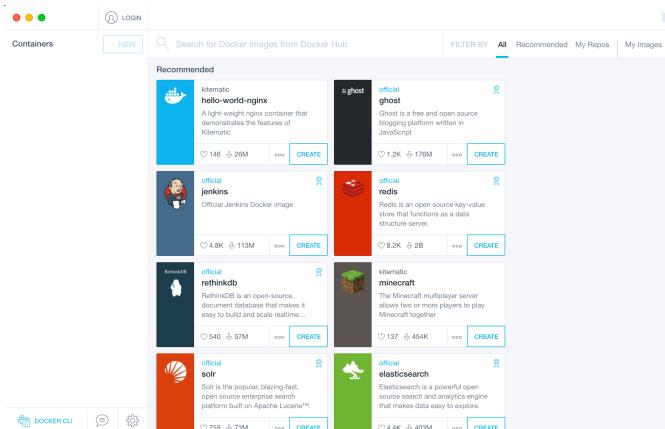
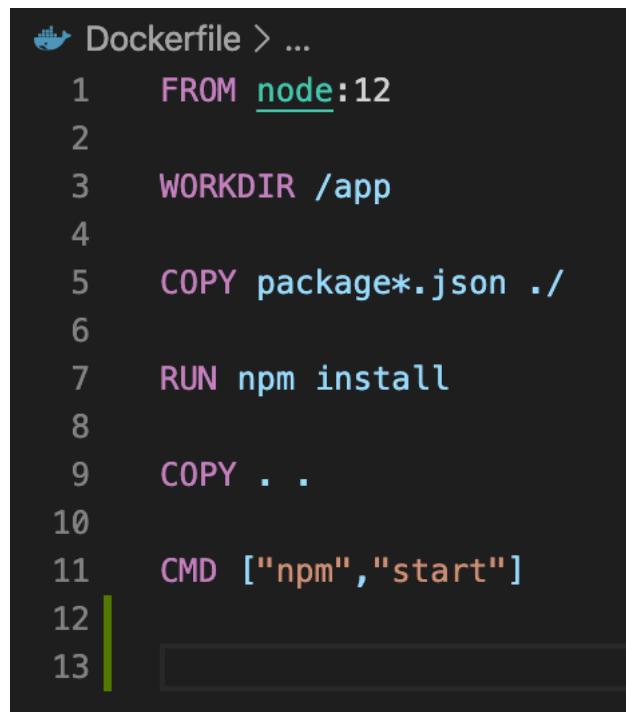


Figura 11. Dashboard de Kitematic

7.5 Dockerfile

Dockerfile és un document de text en yaml que conte totes les comandes que l'usuari pot executar en un terminal per poder crear una imatge. Utilitzant *docker build* executarà les comandes que l'usuari ha definit per crear l'imatge.



```
 1  FROM node:12
 2
 3  WORKDIR /app
 4
 5  COPY package*.json ./
 6
 7  RUN npm install
 8
 9  COPY . .
10
11 CMD ["npm","start"]
12
13
```

Figura 12. Exemple de fitxer Dockerfile

Kubernetes

1. Introducció

Kubernetes és un sistema d'orquestració de contenidors per l'automatització del desplegament, escalat i gestió d'aplicacions, a través de un clúster de hosts. És de codi obert i funciona amb tota sèrie de tecnologies de contenidorització, sovint amb Docker.

2. Història

Kubernetes va ser anunciat per primer cop per Google a mitjans de 2014. El seu desenvolupament i arquitectura van ser fortament influenciats pel sistema Borg de Google. El nom que rebia originalment Kubernetes era Project Seven, en referència al personatge de Star Trek.

La versió 1.0 de Kubernetes va ser publicada el 21 de juliol del 2015. Amb el seu llançament de la primera versió de Kubernetes, Google es va associar amb la Fundació Linux per poder crear la Cloud Native Computing Foundation i va oferir Kubernetes com a tecnologia inicial.

Kubernetes també és utilitzat per RedHat per al seu producte OpenShift, per CoreOS dintre del seu producte Tectonic i per Rancher Labs per la seva plataforma de gestió de contenidors.

3. Components

A Kubernetes es defineixen un conjunt d'eines que ens ajuden a desplegar, mantenir i escalar aplicacions. El conjunt d'eines descrites a continuació estan dissenyades per ser combinades entre elles i per tant poden suportar una gran varietat de càrrega de treball.

- **PODS:** és una visió abstracta dels components continguts. Un POD consisteix en un o més contenidors que corren en el mateix servidor i que poden compartir recursos entre ells. Cada POD té una adreça IP única dins del clúster, que ens permet que les aplicacions utilitzin els ports de la maquines sense risc de conflicte entre ells (poden haver 2 POD's amb els mateixos ports però diferents IP's).

Un POD pot definir un volum, com un directori en un disc local o de la xarxa.

Cada POD es pot gestionar des de l'API de Kubernetes.

- **Namespaces:** permet associar claus anomenades “etiquetes” a qualsevol objecte API, com ara PODS i NODES. Són el primer mecanisme d'agrupació de Kubernetes i s'usen per determinar els components sobre els quals aplicar una operació.
- **Controladors:** és un bucle d'harmonització que duu l'estat actual d'un clúster fins al seu estat desitjat comunicant-se, així, amb la API del servidor per crear, actualitzar i eliminar els recursos que esta gestionant.
Un exemple és el controlador de replicació, que a la part pràctica veurem què és i com funciona.
- **Serveis:** és un grup de PODS que treballen conjuntament. Els PODS que constitueixen un servei estan definits per un selector d'etiquetes. Per defecte, un servei està disponible dins d'un clúster o també poden estar disponibles des de fora d'un clúster. Per exemple tenir varis PODS de front-end que puguin ser visitats per clients.

Part Pràctica

1. Instal·lació de Docker

1.1 MacOS

Per aquest projecte he utilitzat el Docker dashboard ja que permet veure d'una manera més fàcil els recursos consumits, contenidors (actius i aturats), etc.

Per poder fer la instal·lació anirem a la següent pagina:

<https://www.docker.com/products/docker-desktop>

Descarreguem la versió estable del Docker Desktop

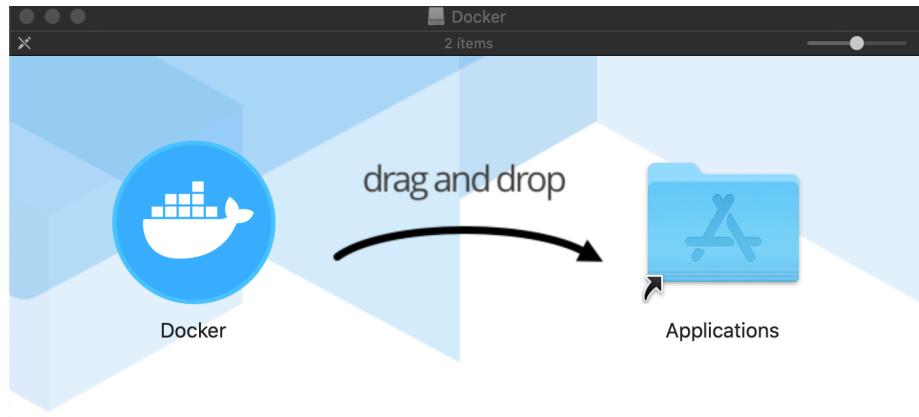


Figura 13. Instal·lació en MacOS.

Copiem el programa en la carpeta d'Aplicacions.

Per mirar si s'ha instal·lat correctament entrarem aquesta comanda al terminal:

```
$ docker --version
```

```
[carlosguimovart@MacBook-MacBook-Pro-de-Carlos ~ % docker --version
Docker version 19.03.8, build afacb8b
carlosguimovart@MacBook-MacBook-Pro-de-Carlos ~ % ]
```

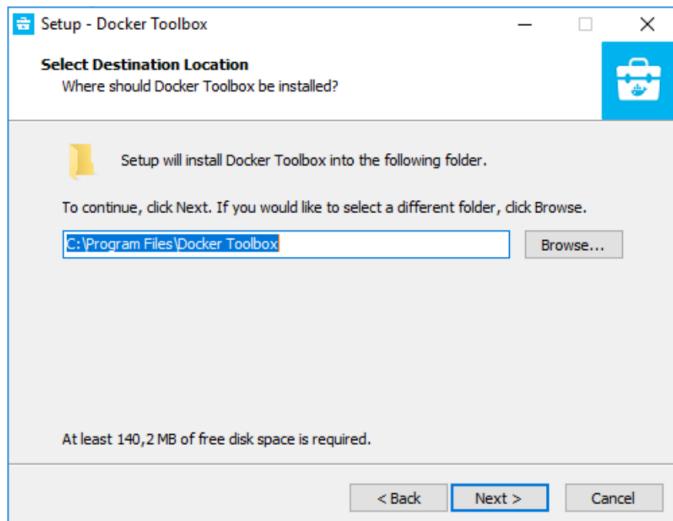
Figura 14. Comprovació de la versió de Docker.

1.2 Windows

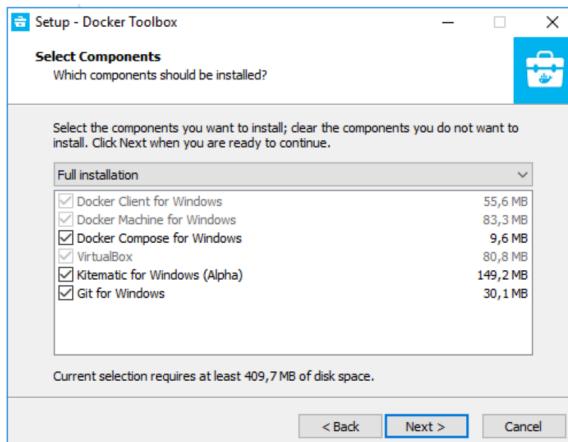
Per poder instal·lar Docker a Windows és necessari tenir una maquina virtual de Linux, ja que Windows no te suport per maquines virtuals. D'aquesta manera no podrem fer una instal·lació nativa de Docker i cada vegada que haguem de crear contenidors i poder gestionar-los haurà de ser a través de la maquina virtual.

Link de descàrrega: <https://github.com/docker/toolbox/releases>

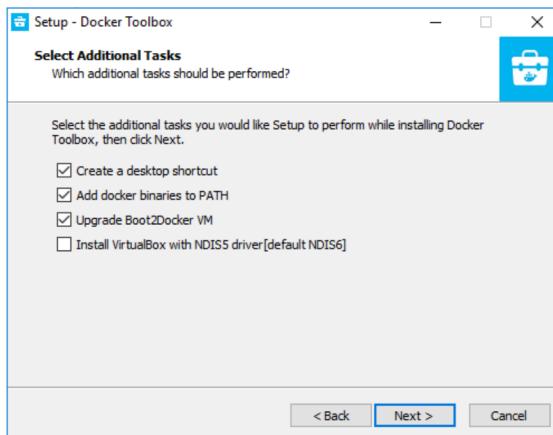
Un cop ens haguem descarregat la darrera versió de Docker Toolbox i executem el .exe i seguim les captures de pantalla següents.



Seleccionem a quin directori volem que s'instal·li.

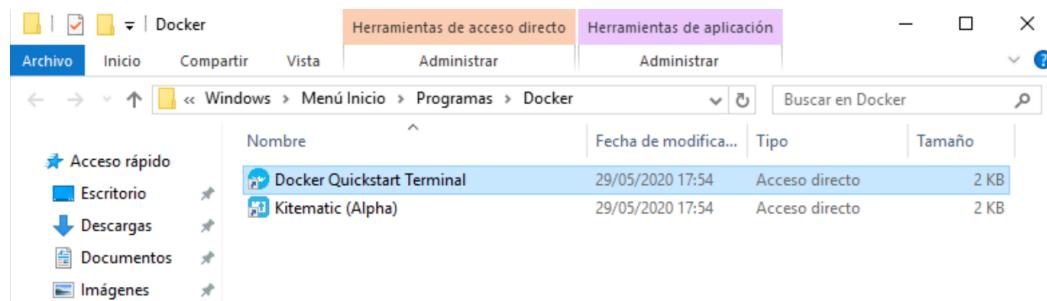


I fem una instal·lació completa. Com podem veure no demana gran quantitat de espai en el disc. És recomanable seleccionar l'última opció per poder tenir un control de versions en el nostre sistema i així poder pujar el codi en un repositori de GitHub.



Marcarem totes les opcions, li donarem a següent i seguidament a instal·lar.

La instal·lació no portarà mes de 8 minuts ja que és una instal·lació rapida i no requereix de moltes dades a descarregar.



S'instal·laran els dos programes per poder crear contenidors. El primer és el terminal, i l'altre, com ja hem vist anteriorment, el Kitematic que ens proporcionarà una part gràfica per la gestió de contenidors.

1.3 Linux

Per fer la demostració de com s'instal·la a Linux he escollit un Ubuntu versió 18.04.

Per poder seguir els comandaments he utilitzat la següent pàgina web:
<https://www.programadornovato.com/02-instalar-docker-en-ubuntu-18-04/>

```
cgm07@cgm08-VirtualBox:~$ sudo apt install apt-transport-https ca-certificates curl software-properties-common
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
ca-certificates ya está en su versión más reciente (20180409).
curl ya está en su versión más reciente (7.58.0-2ubuntu3.8).
software-properties-common ya está en su versión más reciente (0.96.24.32.12).
apt-transport-https ya está en su versión más reciente (1.6.12ubuntu0.1).
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 328 no actualizados
```

Primer de tot instal·larem curl per poder descarregar-nos el docker des d'una URL.

```
cgm07@cgm08-VirtualBox:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
OK
```

Ens descarregarem el fitxer d'instal·lació.

```
cgm07@cgm08-VirtualBox:~$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
Des:1 https://download.docker.com/linux/ubuntu bionic InRelease [64,4 kB]
Obj:2 http://es.archive.ubuntu.com/ubuntu bionic InRelease
Obj:3 http://security.ubuntu.com/ubuntu bionic-security InRelease
Obj:4 http://es.archive.ubuntu.com/ubuntu bionic-updates InRelease
```

Afegim en els repositoris de descarregues , la URL d'on podem descarregar el Docker.

```
cgm07@cgm08-VirtualBox:~$ sudo apt update  
Obj:1 https://download.docker.com/linux/ubuntu bionic InRelease  
Obj:2 http://es.archive.ubuntu.com/ubuntu bionic InRelease  
Obj:3 http://security.ubuntu.com/ubuntu bionic-security InRelease  
Obj:4 http://es.archive.ubuntu.com/ubuntu bionic-updates InRelease  
Obj:5 http://es.archive.ubuntu.com/ubuntu bionic-backports InRelease
```

Farem un update del sistema per guardar el repositori que hem afegit anteriorment.

```
cgm07@cgm08-VirtualBox:~$ apt-cache policy docker-ce  
docker-ce:  
  Instalados: (ninguno)  
  Candidato: 5:19.03.10~3-0~ubuntu-bionic  
  Tabla de versión:  
    5:19.03.10~3-0~ubuntu-bionic 500  
      500 https://download.docker.com/linux/ubuntu bi
```

Figura 15. Validació de compatibilitat

Per poder verificar que el nostre sistema és apte per instal·lar Docker haurim d'entrar el comandament de la captura anterior,i en “Candidato:” ens hauria de sortir una cosa semblant a la que surt en la captura. Això vol dir que esta tot OK per poder ser instal·lat.

```
cgm07@cgm08-VirtualBox:~$ sudo apt install docker-ce  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias  
Leyendo la información de estado... Hecho  
Se instalarán los siguientes paquetes adicionales:  
  aufs-tools cgroupfs-mount containerd.io docker-ce-cli git git-man  
  liberror-perl pigz  
Paquetes sugeridos:
```

L'instal·lem

```
cgm07@cgm08-VirtualBox:~$ sudo systemctl status docker  
● docker.service - Docker Application Container Engine  
  Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: e  
  Active: active (running) since Fri 2020-05-29 18:26:29 CEST; 12s ago  
    Docs: https://docs.docker.com  
  Main PID: 11644 (dockerd)  
     Tasks: 10  
    CGroup: /system.slice/docker.service  
           └─11644 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/contai  
  
may 29 18:26:29 cgmo8-VirtualBox dockerd[11644]: time="2020-05-29T18:26:29.14131  
may 29 18:26:29 cgmo8-VirtualBox dockerd[11644]: time="2020-05-29T18:26:29.14132  
may 29 18:26:29 cgmo8-VirtualBox dockerd[11644]: time="2020-05-29T18:26:29.14133
```

Un cop instal·lat verifiquem que el servei de Docker estigui funcionant. Com veiem a la captura anterior , tot està funcionant perfectament

```
cgm07@cgm08-VirtualBox:~$ docker --version  
Docker version 19.03.10, build 9424aeae9  
cgm07@cgm08-VirtualBox:~$ █
```

I per últim verifiquem la versió de Docker.

2. Instal·lació de Kubernetes.

En el cas pràctica hem instal·lat miniKube, que ens permetrà tenir un clúster de Kubernetes en local. Per això és necessari tenir instal·lat el kubectl, que com ja hem vist anteriorment és l'intèrpret d'ordres del Kubernetes.

Link per a poder fer la instal·lació:

<https://kubernetes.io/docs/tasks/tools/install-kubectl/#install-kubectl-on-windows>

Un cop tinguem instal·lat kubectl haurem d'instal·lar miniKube.

2.1 MiniKube en Windows.

La instal·lació en Windows serà com la de Docker, necessitarem una maquina virtual per que estigui arrencat el nostre clúster de Kubernetes.

Descarregarem un executable en la següent pagina web:

<https://github.com/kubernetes/minikube/releases/latest/download/minikube-installer.exe>

Com en totes les instal·lacions en Windows necessitarem donar-li a “Següent” fins que el tinguem instal·lat.

2.2 MiniKube en Linux.

Per fer la instal·lació hem utilitzat un Ubuntu 18.04.

Cal recordar que s'ha de tenir el VirtualBox instal·lat per poder continuar amb la instal·lació.

Ens dirigirem a la següent web per copiar les ordres que s'han d'executar al terminal.

<https://kubernetes.io/docs/tasks/tools/install-minikube/>

2.3 Minikube en MacOS

Per poder fer la instal·lació necessitem tenir un gestor de maquines virtuals, ja que no hi ha instal·lació nativa en MacOS.

Link per a la instal·lació:

<https://kubernetes.io/docs/tasks/tools/install-minikube/>

3. Expressions en Docker

En aquest apartat veurem totes les comandes que podem executar en el nostre terminal, entre elles: com arrencar un contenidor des d'una imatge, com crear una imatge, com pujar aquesta imatge a Docker Hub...

- \$ docker ps -a → mostrarà quins són els contenidors que tenim creats en local estiguin aturats o oberts.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
8c425858f1c2	32472411d489	"docker-entrypoint.s..."	7 hours ago	Up 7 hours		k8s_wordpres
s_wordpress-q9gsx_testing_161f077e-70ef-4857-b9e3-ac4088f8883f_0	gcr.io/k8s-minikube/kicbase:v0.0.10	"/usr/local/bin/entr..."	45 hours ago	Exited (130) 25 hours ago	0.0.0.0:3000->3000/tcp	minikube
0b9792f60c9c	websocketss-master_app	"docker-entrypoint.s..."	2 days ago	Up 7 hours		node-webssoke
ts_e96b7bc56367	wordpress:php7.2-apache	"docker-entrypoint.s..."	4 days ago	Exited (0) 45 hours ago		projecte-fin
a1_wordpress_1	mysql:8.0.13	"docker-entrypoint.s..."	4 days ago	Exited (0) 45 hours ago		projecte-fin
a1_mysql_1	mongo	"docker-entrypoint.s..."	2 weeks ago	Exited (255) 31 hours ago	0.0.0.0:27017->27017/tcp	mongo

Figura 16. Contenidors en Docker

- `$ docker images` → mostrarà totes les imatges que tenim descarregades.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
minecraft	latest	d48f55766bd0	9 days ago	595MB
itzg/minecraft-server	latest	e3c9380d9976	9 days ago	279MB
wordpress	php7.2-apache	32472411d489	13 days ago	540MB
sveltecg/angularapp	v1.0	0ee445c9088d	2 weeks ago	776MB
ubuntu_node	v1.0	bdd35e871c41	2 weeks ago	267MB
sveltecg/ubuntu-node	v1.0	bdd35e871c41	2 weeks ago	267MB
sveltecg/ubuntuweb	v2.0	7fd1f67e7abd7	2 weeks ago	1.18GB
<none>	<none>	190af552742e	2 weeks ago	969MB
websocketss-master_app	latest	0de0e727e0989	2 weeks ago	969MB
sveltecg/websocketss-master_app	latest	0de0e727e0989	2 weeks ago	969MB
sveltecg/websocketss.pr	latest	0de0e727e0989	2 weeks ago	969MB
<none>	<none>	c9dd2d0beb5e	3 weeks ago	356MB
node	12	9dd56f7e7085f	4 weeks ago	916MB
gcr.io/k8s-minikube/kicbase	v0.0.10	e6bc41c39dc4	4 weeks ago	974MB
mongo	latest	3f3daef863757	4 weeks ago	388MB
ubuntu	latest	1d622ef86b13	4 weeks ago	73.9MB
kubernetesui/dashboard	v2.0.0	8b32422733b3	5 weeks ago	222MB
docker/getting-started	latest	3c156928aec	5 weeks ago	24.8MB

Figura 17. Imatges creades

- `$ docker pull + "nom de la imatge"` → permetrà descarregar qualssevol imatge de les que hi hagin en el Docker Hub.

```
[carlosguimovart@MacBook-MacBook-Pro-de-Carlos ~ % docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
afb6ec6fdc1c: Already exists
b90c53a0b692: Pull complete
11fa52a0fdc0: Pull complete
Digest: sha256:30dfa439718a17baafefadf16c5e7c9d0a1cde97b4fd84f63b69e13513be7097
Status: Downloaded newer image for nginx:latest
carlosguimovart@MacBook-MacBook-Pro-de-Carlos ~ % ]
```

Figura 18. Imatges creades

com veiem en aquest exemple, ens hem descarregat una imatge d'un servidor web nginx

En el següent exemple veurem com podem crear un contenidor a partir d'aquesta imatge de nginx que ens hem descarregat. Exposant els ports, i canviant-li el nom per poder fer-la més personal.

```
[carlosguimovart@MacBook-MacBook-Pro-de-Carlos ~ % docker run --name server-nginx -d -p 8085:80 nginx
47f358136c2a3a5947ae26ba8aac1f0bal36682c9fc1e74624e1fa3aa67b391f
carlosguimovart@MacBook-MacBook-Pro-de-Carlos ~ % ]
```

Figura 19. Arrencar imatge

Comentar que en `--name` hem posat el nom amb el qual s'identificarà el contenidor. `-d` fa que el contenidor s'executi en segon pla. `-p 8085:80` fa que el port 80 del propi contenidor el converteixi al 8085 per poder accedir-hi des de la màquina local.

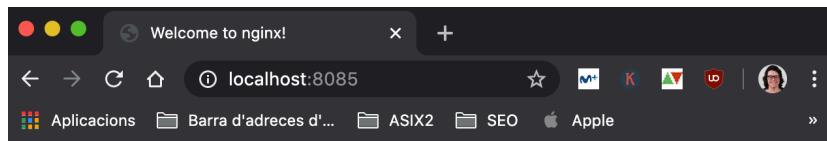
```
carlosguinovart@MacBook-Pro-de-Carlos ~ % docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
47f358136c2a        nginx              "nginx -g 'daemon off;'"
8c425881c2          32472411d489      "docker-entrypoint.s_"
g_161f077e-70ef-4857-b9e3-ac4088f8883f_0   7 hours ago       Up 2 minutes      0.0.0.0:8085->80/tcp   server-nginx
k8s_wordpress_wordpress-q5gsx_testin
0b0792f60c9c        websockets-master_app  "docker-entrypoint.s_"
2 days ago         Up 7 hours        0.0.0.0:3000->3000/tcp   node-websockets
carlosguinovart@MacBook-Pro-de-Carlos ~ %
```

Figura 20. Validació del contenidor

Si mirem els contenidors que estant executant-se en aquests moments, veurem com el contenidor que acabem de crear a partir de la imatge que ens hem descarregat s'està executant amb els paràmetres que li hem declarat anteriorment.

Podem accedir al port 80 del nostre contenidor posant al navegador:

localhost:8085 on 8085 és el port què nosaltres anteriorment li hem declarat.



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

- `docker exec -it + "IDCONTAINER" sh` → connectarà al contenidor mitjançant el seu terminal i podrem esborrar/editar i crear qualsevol fitxer del nostre contenidor

```
carlosguinovart@MacBook-Pro-de-Carlos ~ % docker exec -it 47f358136c2a sh
# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
#
```

Figura 20. Accés al contenidor

- \$ docker commit + “IDCONTAINER” → permetrà fer un commit de la imatge que hem alterat i guardar-la en el nostre repositori local d’imatges de docker. Seguint els exemples anteriors. Hem creat un fitxer en el nostre contenidor per poder crear una imatge diferent a la que ens hem descarregat.

```
[carlosguimovart@MacBook-MacBook-Pro-de-Carlos ~ % docker exec -it 47f358136c2a sh
# touch hola.txt
# ls
bin  boot  dev  etc  hola.txt  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
# ]
```

Figura 21. Validació que el fitxer s’ha creat

Amb aquest contenidor personalitzat per nosaltres li crearem una imatge associada.

```
[carlosguimovart@MacBook-MacBook-Pro-de-Carlos ~ % docker commit 47f358136c2a
sha256:5408c735df760b4b7ff31f3db9fe8e8c2a9c79e07090b2d54d11a0f3a385fb1
[carlosguimovart@MacBook-MacBook-Pro-de-Carlos ~ % docker images
REPOSITORY          TAG      IMAGE ID            CREATED             SIZE
<none>              <none>   5408c735df76        4 seconds ago       127MB]
```

Figura 22. Fer commit del contenidor editat

```
[carlosguimovart@MacBook-MacBook-Pro-de-Carlos ~ % docker image tag 5408c735df76 nginx-hola]
```

Figura 23. Canvi de nom de la imatge

```
[carlosguimovart@MacBook-MacBook-Pro-de-Carlos ~ % docker images
REPOSITORY          TAG      IMAGE ID            CREATED             SIZE
nginx-hola          latest   5408c735df76        About a minute ago  127MB]
```

Figura 24. Validació del canvi de nom

I ja tindríem la imatge amb els canvis realitzats.

Amb el \$ docker imatge tag + “Image ID” + “nom” li assignarem un nom a la imatge que hem creat a partir del contenidor.

Si ara esborrem el contenidor que hi ha els canvis realitzats i executem un nou contenidor des de la imatge que hem creat hauria de sortir el contenidor modificat. Per això farem un docker rm + “IDCONTAINER”

I executarem de nou el comandament vist anteriorment amb el nom de la nova imatge. Veiem l’exemple.

```
[carlosguimovart@MacBook-MacBook-Pro-de-Carlos ~ % docker rm 47f358136c2a
47f358136c2a
[carlosguimovart@MacBook-MacBook-Pro-de-Carlos ~ % docker run --name server-nginx -d -p 8085:80 nginx-hola
61623873d202947a64a9dde036afc49350911ec8252e22c29a1c966200e2a5
[carlosguimovart@MacBook-MacBook-Pro-de-Carlos ~ % docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                 NAMES
61623873d202         nginx-hola          "nginx -g 'daemon off;'"   14 seconds ago    Up 12 seconds      0.0.0.0:8085->80/tcp   server-nginx
8c425858f1c2         32472411d489        "docker-entrypoint.s..."   8 hours ago       Up 8 hours        0.0.0.0:80->80/tcp   k8s_wordpress_wordpress-q5gsx_testing_161f
077e-70ef-4857-b9e3-ac4088f8883f_0
0b9792f6b9c9          websockets-master_app  "docker-entrypoint.s..."   2 days ago        Up 8 hours        0.0.0.0:3000->3000/tcp   node-websokets
carlosguimovart@MacBook-MacBook-Pro-de-Carlos ~ % ]
```

Figura 25. Arrencar a partir de la imatge creada

```
carlosguimovart@MacBook-MacBook-Pro-de-Carlos ~ % docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
61623873d202        nginx-hola          "nginx -g 'daemon of..." 14 seconds ago   Up 12 seconds      0.0.0.
8c425858f1c2        32472411d489       "docker-entrypoint.s..." 8 hours ago     Up 8 hours        0.0.0.
077e-70ef-4857-b9e3-ac4088f883f_0
09792f60c9c         websockets-master_app  "docker-entrypoint.s..." 2 days ago      Up 8 hours        0.0.0.
carlosguimovart@MacBook-MacBook-Pro-de-Carlos ~ % docker exec -it 61623873d202 sh
# ls
bin  boot  dev  etc  hola.txt  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
#
```

Figura 26. Validació del canvi

- \$ docker stop + “IDCONTAINER” → pararem el contenidor que s'està executant en segon pla.

```
carlosguimovart@MacBook-MacBook-Pro-de-Carlos ~ % docker stop 61623873d202
61623873d202
carlosguimovart@MacBook-MacBook-Pro-de-Carlos ~ % docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
61623873d202        nginx-hola         "nginx -g 'daemon off;'"   4 minutes ago    Exited (0) 5 seconds ago   server-nginx
```

Figura 27. Aturar el contenido

3.1 Creació d'un Dockerfile a partir de l'aplicació

Com ja hem dit anteriorment aquesta aplicació que he creat és un simple joc web que ens permet poder pintar requadres el més ràpid possible contra un altre jugador. Qui obtingui més requadres pintats serà qui guanyi el joc.

Requisits per poder desenvolupar l'aplicació:

- Tenir NodeJs i npm instal·lat correctament en la nostre maquina local per poder fer el desenvolupament.
 - Editor de text pla. Jo en aquest cas he utilitzat el VisualStudio Code, ja que ens ofereix una gran quantitat d'eines per poder desenvolupar qualsevol aplicació en qualsevol llenguatge de programació.
 - Base de dades MongoDB. Ja que més endavant dissenyarem un sistema de registre d'usuaris.

Un cop tenint-ho tot instal·lat i configurat adequadament haurem d'instal·lar les llibreries necessàries.

A continuació deixo els comandaments que s'han d'executar per poder instal·lar-los. Seguidament també hi ha una captura d'altres dependències necessàries.

- #### - *Npm install mongoose*

- *Npm install pug*
- *Npm install socket.io*
- *Npm install express*

```
dependencies: {
  "body-parser": "^1.18.2",
  "cookie-parser": "^0.4.0",
  "cookie-parser": "^1.4.3",
  "express": "^4.16.3",
  "express-session": "^1.15.6",
  "mongoose": "^5.0.16",
  "morgan": "^1.9.0",
  "passport": "^0.4.0",
  "passport-local": "^1.0.0",
  "pug": "^2.0.3",
  "socket.io": "^2.1.0"
}
```

Figura 28. Dependencies del projecte

Fer un deploy de l'aplicació en Docker. Això significa poder executar aquesta aplicació web que acabem de crear en un contenidor de Docker.

Per això dins del directori on es troben tots els fitxers de l'aplicació hem de crear 3 fitxers:

- Dockerfile
- .dockerignore
- docker-compose.yml



Figura 29. Fitxers a crear

En el Dockerfile ens trobarem la següent estructura:

```
1 FROM node:12
2
3 WORKDIR /app
4
5 COPY package*.json ./
6
7 RUN npm install
8
9 COPY . .
10
11 CMD ["npm", "start"]
```

Figura 30. Ordres del dockerfile

FROM → és la imatge que volem descarregar per poder-la personalitzar i introduir-li el codi.

WORKDIR → canviàrà el directori per defecte on executarem les nostres comandes.

COPY → copiarà tots els fitxers que es posin al directori demandat, en aquest cas, copiarà el *package.json* i *package-lock.json* en el nostre directori arrel (on es trobarà la nostra aplicació).

RUN → permet executar comandes directament en el Shell del contenidor. En el exemple el que fa és instal·lar totes les dependències que té el nostre projecte.

COPY → copiarà tota la resta de fitxers del nostre directori del projecte al directori esmentat.

CMD → és una execució de comandaments per defecte un cop després de crear el contenidor. En el exemple li estem dient que executi l'aplicació.

En el fitxer *.dockerignore* posarem tots aquells fitxers i directoris que no volem que es copin en el nostre contenidor.

```
↳ .dockerignore
1   node_module
2   npm-debug.log
```

Figura 31. Fitxers a ignorar

En aquest cas veiem que hem posat la carpeta de node_modules, que és on es troben totes les dependències del projecte instal·lades, i el npm-debug.log que és on es troben tots els logs de possibles falles d'execució de la nostre aplicació.

3.2 Creació d'un docker-compose a partir de l'aplicació

El docker-compose.yml el que ens permetrà serà crear 2 contenidors connectats entre ells, com ja hem vist en el desenvolupament de l'aplicació és que tenim una base de dades feta en MongoDB.

Per poder fer aquesta connexió entre contenidors el fitxer haurà de tenir la següent estructura:

```
↳ docker-compose.yml
1   version: '3'
2   services:
3     app:
4       container_name: node-websokets
5       restart: always
6       build: .
7       command: npm start
8       ports:
9         - '3000:3000'
10      links:
11        - mongo
12    mongo:
13      container_name: mongo
14      image: mongo
15      ports:
16        - '27017:27017'
```

Figura 31. Ordres del docker-compose

Version → s'especifica la versió del docker-compose que volem utilitzar.

Services → s'especifiquen les configuracions de cada contenidor, com veiem en el exemple anterior, hem creat 2 contenidors. Un per l'aplicació, i l'altre és on estarà creat tota la base de dades.

App → li hem posat totes les configuracions del contenidor.

- En **container_name**: li hem declarat el nom que volem que tingui el contenidor que contingui la aplicació.
- En **restart**: li hem demanat que cada vegada que Docker inici de nou, ell aixequi aquest contenidor.
- **Build**: s'utilitza per poder indicar a on es troba el fitxer Dockerfile que hem creat anteriorment. Posant-hi el “.” estem especificant que es troba en el mateix directori que el docker-compose.
- **Command** → una vegada s'hagi creat el contenidor, li determinarem quin es el comandament que necessita per poder arrencar l'aplicació. Igual com hem vist en el fitxer de Dockerfile.
- **Ports** → hi declararem quins ports està utilitzant l'aplicació i en quins ports volem redirigir aquest tràfic de peticions.
- **Links** → hi declararem quin container està requerint per que la aplicació funcioni adequadament.

En el segon container hi declararem el següent:

- **container_name**: com ja hem vist amb l'altre contenidor que hem declarat, hi posarem un nom de contenidor.
- **image**: especificarem quina és la imatge que volem utilitzar i en cas que no la tinguéssim en local, la descarregaria.
- **Ports**: declararem els ports que utilitza el contenidor i quins volem que utilitzi l'aplicació

Un cop ja tinguem aquests 3 fitxers creats al directori del projecte haurem d'executar les comandes per que es creïn quests contenidors.

```
carlosguinovart@MacBook-MacBook-Pro-de-Carlos:~/Websockets-master % docker-compose up
Creating network "websockets-master_default" with the default driver
Pulling mongo (mongo)...
latest: Pulling from library/mongo
23884877105a: Pull complete
bc38caaa0f5b9: Pull complete
2910811b6c42: Pull complete
36505266dcc6: Pull complete
a4d269900d94: Pull complete
5e2526abb801: Pull complete
d3eece1f39ec: Pull complete
358ed78d3204: Pull complete
1a878b8604ae: Pull complete
978c572f0440: Pull complete
35a600fffcf6a: Pull complete
fa9f812cdfe6: Pull complete
7a8109e27110: Pull complete
Digest: sha256:be8d903a68997dd63f64479004a7eeb4f0674dde7ab3cbd1145e5658da3a817b
Status: Downloaded newer image for mongo:latest
Creating mongo ... done
Creating node-websokets ... done
Attaching to mongo, node-websokets
mongo    | 2020-05-29T12:02:54.849+0000 I CONTROL [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'
mongo    | 2020-05-29T12:02:54.852+0000 W ASIO      [main] No TransportLayer configured during NetworkInterface startup
mongo    | 2020-05-29T12:02:54.852+0000 I CONTROL [initandlisten] MongoDB starting : pid=1 port=27017 dbpath=/data/db 6
4-bit host=f21afe66a7f0
```

Figura 32. Execució del docker-compose

Com podem veure executant “`$ docker-compose up`” ha començat a crear els 2 contenidors. Com que la imatge de mongo no el tenia descarregat, s'ha descarregat l'última versió existent. I seguidament ha arrencat els 2 contenidors.

Si ara nosaltres busquem amb el comandament `$ docker ps`, ens mostrerà els 2 contenidors arrencats de manera satisfactòria.

carlosguinovart@MacBook-MacBook-Pro-de-Carlos:~/Websockets-master % docker ps						
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e8e5c3cb107a	websockets-master_app	"docker-entrypoint.s..."	3 minutes ago	Up 3 minutes	0.0.0.0:3000->3000/tcp	node-websokets
f21afe66a7f0	mongo	"docker-entrypoint.s..."	3 minutes ago	Up 3 minutes	0.0.0.0:27017->27017/tcp	mongo

Figura 33. Comprovació de l'execució

Si ara ens dirigim en el nostre navegador web, i busquem localhost:3000, ens hauria de permetre veure l'aplicació desenvolupada arrencada en el nostre contenidor que acabem de crear.



Figura 34. Captura de pantalla del seu correcte funcionament.

4. Kubernetes

Per poder desenvolupar questa part el que he fet és instal·lar kubectl i Minikube.

Minikube és una implementació més lleugera de kubernetes per poder executar-ho localment. També utilitzarem kubectl que és un programa de tipus de línia de comandes que ens permet interactuar amb Minikube.

Un cop ja hem instal·lat el minikube farem un `$ minikube start`, això ens obria la maquina virtual on esta corrent l'instancia de Kubernetes.

```
[carlosguinovart@MacBook-Pro-de-Carlos ~ % minikube start
😊 minikube v1.10.1 on Darwin 10.15.5
👍 Using the docker driver based on existing profile
👍 Starting control plane node minikube in cluster minikube
🌟 minikube 1.11.0 is available! Download it: https://github.com/kubernetes/minikube/releases/tag/v1.11.0
💡 To disable this notice, run: 'minikube config set WantUpdateNotification false'

🔄 Restarting existing docker container for "minikube" ...
🌐 Preparing Kubernetes v1.18.2 on Docker 19.03.2 ...
    🚀 kubeadm.pod-network-cidr=10.244.0.0/16
💡 Unable to restart cluster, will reset it: getting k8s client: client config: client config: invalid configuration: no configuration has been provided
🌟 Enabled addons: dashboard, default-storageclass, storage-provisioner
🔥 Done! kubectl is now configured to use "minikube"

❗ /usr/local/bin/kubectl is v1.16.6-beta.0, which may be incompatible with Kubernetes v1.18.2.
💡 You can also use 'minikube kubectl -- get pods' to invoke a matching version
carlosguinovart@MacBook-Pro-de-Carlos ~ % ]
```

Figura 35. Arrencada del minikube.

4.1 Llançament d'una aplicació

Per fer aquesta part, he utilitzat el contenidor creat anteriorment que conté l'aplicació feta en NodeJS.

Abans de tot ens hem d'assegurar que hem penjat el contenidor en un repositori de contenidors, jo en aquest cas l'he penjat en DockerHub.

Per poder penjar una imatge al repositori hem de fer el següent:

- 1) Iniciar secció a Docker Hub fent: `$ docker login`. Nota: s'ha de tenir un usuari registrat a Docker Hub
- 2) Verificar que la imatge que volem penjar és la correcte.

En el meu cas des d'on tenia el `docker-compose.yml` he fet un `$ docker-compose build` per construir la imatge.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
m06websockets_app	latest	cf1cff133bf	7 seconds ago	969MB

Figura 36. Llistat d'imatges.

A continuació la preparem per ser penjada a Docker Hub.

```
carlosguimovart@MacBook-MacBook-Pro-de-Carlos m06Websockets % docker image tag cf1cff133bf sveltecg/node-websockets_clot
```

Figura 37. Canvi de nom de la imatge.

Li canviem el nom de la imatge per penjar-la al repositori.

Un cop li hem canviat el nom, fem un: `$docker push + "nom de la imatge"`

Com detalla en el exemple següent, hem utilitzat la imatge que acabem de generar.

```
carlosguimovart@MacBook-MacBook-Pro-de-Carlos m06Websockets % docker push sveltecg/node-websockets_clot
The push refers to repository [docker.io/sveltecg/node-websockets_clot]
d0fbf7c25aec: Preparing
d0fbf7c25aec: Pushing [=====] 35.92MB
0b5257402c07: Mounted from sveltecg/m06websockets_app
0b5257402c07: Pushed
```

Figura 38. Publicació de la imatge al repositori de DockerHub.

I verificarem que ha estat correctament penjada el Hub.

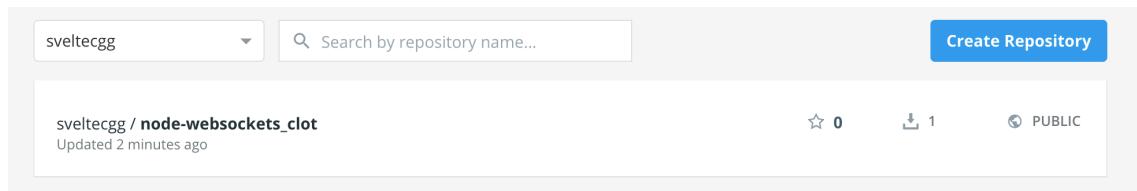


Figura 39. Comprovació de la publicació

3) Obrir el clúster per poder fer el deploy de l'aplicació

Amb `$ minikube start`, ens obre la màquina virtual on s'està executant Kubernetes.

```
carlosguimovart@MacBook-MacBook-Pro-de-Carlos m06Websockets % minikube start
minikube v1.10.1 on Darwin 10.15.5
Using the virtualbox driver based on existing profile
Starting control plane node minikube in cluster minikube
Updating the running virtualbox "minikube" VM ...
Preparing Kubernetes v1.18.2 on Docker 19.03.8 ...
Enabled addons: dashboard, default-storageclass, storage-provisioner
Done! kubectl is now configured to use "minikube"
/usr/local/bin/kubectl is v1.16.6-beta.0, which may be incompatible with Kubernetes
```

Figura 39. Arrencada del Minikube

Mirem si el clúster ha estat arrencat correctament.

```
carlosguimovart@MacBook-MacBook-Pro-de-Carlos Desktop % minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
carlosguimovart@MacBook-MacBook-Pro-de-Carlos Desktop %
```

Figura 40. Comprovació de l'estat de minikube

4) Definició del Deployment.

```
kube > ! 01-Deployment-NodeJsApp.yml
13
14  apiVersion: apps/v1
15  kind: Deployment
16  metadata:
17    |   name: nodeapp
18  spec:
19    |   replicas: 1
20    |   selector:
21    |     matchLabels:
22    |       |   app: nodeapp
23    |   template:
24    |     metadata:
25    |       labels:
26    |         |   app: nodeapp
27    |     spec:
28    |       containers:
29    |         - name: nodeapp
30    |           image: sveltecg/m06websockets_app
31    |           ports:
32    |             - containerPort: 3000
33    |           env:
34    |             - name: MONGO_URL
35    |               value: mongodb://mongo:27017/websockets
36    |           imagePullPolicy: Always
37
```

Figura 41. Estructura del fitxer

Amb aquest fitxer crearem els deployments de la aplicació

Apartats a destacar:

- **Replicas:** numero de POD's (contenidors) corrent en d'una mateixa aplicació
- **Name:** nom personalitzat que tindran els POD's
- **Imatge:** la imatge original d'on serà descarregada.
- **Ports:** ports que utilitza l'aplicació per redirigir-los
- **ENV:** Realització de la connexió al POD de la base de dades. Si el nom del POD és diferent, la URL serà diferent.

5) Definició del servei.

El servei ens permetrà connectar el POD de la aplicació amb el POD de la base de dades.

També ens permetrà accedir als usuaris des de fora del clúster.

Sense un servei un POD no pot ser accedit de cap forma, per això son tan importants.

D'aquesta manera un servei s'assemblaria a un equilibrador de carrega.

En el següent exemple creem un servei per que el POD pugui ser accessible des d'un client

```
kube > ! 01-Service-Loadbalancer.yml
 1  apiVersion: v1
 2  kind: Service
 3  metadata:
 4    name: nodeapp-svc
 5  spec:
 6    selector:
 7      app: nodeapp
 8    ports:
 9      - port: 3000
10        targetPort: 3000
11    type: NodePort
12
```

Figura 42. Estructura del fitxer

Parts a destacar:

- **Spec > selector > app:** tots els POD's que tinguin el Label de app: nodeapp són els que seran exposats al exterior.
- **Ports:** en quest cas el servei, escolta les sol·licituds al port 3000 i els reenvia al port 3000 de cada POD.

6) Definició de la base de dades.

En un principi MongoDB pot ser igual de executada que una aplicació desenvolupada. Però suposa unes configuracions addicionals com per exemple una persistència d'emmagatzematge.

Aquest emmagatzematge persistent no s'ha de veure afectat per que els hi pugui passar els POD's de MongoDB. Si es borra el POD de MongoDB aquestes dades que s'han anat guardant durant la possible utilització de clients, no afectaria a les dades.

En conseqüència, la descripció del component de la vostra base de dades hauria de constar de 3 definicions.

- A. PersistentVolumeClaim
- B. Service
- C. Deployment

PersistentVolumeClaim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongo-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 256Mi
```

Figura 43. Estructura del fitxer

Parts a destacar:

1. **Kind:** tipus de POD
2. **AccessModes:** modes d'accés que tindran les dades.
3. **Storage:** mida màxima que ocuparan els fitxers de la base de dades.

Service

```
apiVersion: v1
kind: Service
metadata:
|   name: mongo
spec:
|   selector:
|       app: mongo
ports:
|   - port: 27017
|       targetPort: 27017
```

Figura 44. Estructura del fitxer

Parts a destacar:

1. **Kind:** tipus de POD
2. **Ports:** ports que utilitza l'aplicació i des de quin port podrà accedir el usuari.

Deployment de MongoDB

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongo
spec:
  selector:
    matchLabels:
      app: mongo
  template:
    metadata:
      labels:
        app: mongo
    spec:
      containers:
        - name: mongo
          image: mongo
          ports:
            - containerPort: 27017
          volumeMounts:
            - name: storage
              mountPath: /data/db
      volumes:
        - name: storage
          persistentVolumeClaim:
            claimName: mongo-pvc
```

Figura 45. Estructura del fitxer

Parts a destacar:

1. **Image**: la imatge que utilitzarà el POD. En aquest cas com que és MongoDB la imatge del repositori es diu mongo.
2. **Ports**: el port que utilitza els POD's que es creïn.
3. **Volumes**: El camp de volums defineix un volum d'emmagatzematge anomenat emmagatzematge, que fa referència a PersistentVolumeClaim.
4. **VolumeMounts**: El camp VolumeMount munta el volum referenciat en la ruta especificada al contenidor, que en aquest cas és / data / db.

Les 3 sentencies anteriors poden estar en un mateix arxiu de Deployment.

7) Aplicar configuracions.

Per poder aplicar totes les configuracions que hem definit a l'apartat anterior haurem de realitzar els següents passos.

Primer de tot executar el fitxer de les configuracions del MongoDB.

```
carlosguimovart@MacBook-MacBook-Pro-de-Carlos kube % kubectl apply -f 01-Mongo-Deployment.yml
persistentvolumeclaim/mongo-pvc unchanged
service/mongo unchanged
deployment.apps/mongo unchanged
carlosguimovart@MacBook-MacBook-Pro-de-Carlos kube %
```

Figura 45. Aplicació de les configuracions de MongoDB

Seguidament aplicarem les configuracions de la nostre aplicació.

```
carlosguimovart@MacBook-MacBook-Pro-de-Carlos kube % kubectl apply -f 01-Deployment-NodeJsApp.yml
service/nodeapp unchanged
deployment.apps/nodeapp configured
carlosguimovart@MacBook-MacBook-Pro-de-Carlos kube %
```

Figura 46. Aplicació de les configuracions de l'aplicació

Després de haver executat els fitxers de configuració. Que en el meu cas han estat 2 mirarem si s'han aplicat els canvis mitjançant `$ kubectl get all`.

```
carlosguimovart@MacBook-MacBook-Pro-de-Carlos m06Websockets % kubectl get all
NAME                 READY   STATUS    RESTARTS   AGE
pod/hello-world-86d6c6f84d-5rrl4  1/1     Running   2          25h
pod/hello-world-86d6c6f84d-88k9v  1/1     Running   2          25h
pod/mongo-6dc456678-xpxh9        1/1     Running   0          140m
pod/mongodb-standalone-0          0/1     Pending    0          145m
pod/nodeapp-676c8cbddc-9nc5b     1/1     Running   0          133m
pod/nodeapp-676c8cbddc-pvjhp     1/1     Running   0          141m

NAME                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)      AGE
service/example-service  NodePort   10.107.245.35  <none>        8080:30953/TCP  25h
service/kubernetes       ClusterIP  10.96.0.1     <none>        443/TCP      26h
service/mongo             ClusterIP  10.109.240.143 <none>        27017/TCP     140m
service/nodeapp            LoadBalancer 10.99.245.94  <pending>    3000:32612/TCP  141m
service/nodeapp-svc       NodePort   10.103.154.19 <none>        80:32161/TCP   141m

NAME                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-world  2/2     2           2           25h
deployment.apps/mongo        1/1     1           1           140m
deployment.apps/nodeapp     2/2     2           2           141m

NAME                DESIRED  CURRENT  READY   AGE
replicaset.apps/hello-world-86d6c6f84d  2        2        2      25h
replicaset.apps/mongo-6dc456678        1        1        1      140m
replicaset.apps/nodeapp-676c8cbddc    2        2        2      141m

NAME                READY   AGE
statefulset.apps/mongodb-standalone  0/1     22h
carlosguimovart@MacBook-MacBook-Pro-de-Carlos m06Websockets %
```

Figura 47. Comprovació del la creació

Un cop ja hagi estat tot ben configurat intentarem connectar-nos en un POD via Google Chrome.

I seguirem els següents passos.

Executem `$ minikube service + "nom del servei"`, que en el nostre cas és el servei nodeapp. Aquesta comanda ens farà de passarel·la per poder-nos connectar al nostre clúster.

```
carlosguimovart@MacBook-MacBook-Pro-de-Carlos kube % minikube service nodeapp
|-----|
| NAMESPACE | NAME | TARGET PORT | URL |
|-----|
| default   | nodeapp | 3000 | http://192.168.99.101:32612 |
|-----|
💡 Opening service default/nodeapp in default browser...
carlosguimovart@MacBook-MacBook-Pro-de-Carlos kube %
```

Figura 48. Pont cap al nostre servei

Si copiem la URL que ens mostrarà, hauria de sortir això.



Figura 49. Validació del perfecte funcionament de l'aplicació en kubernetes

Conclusions

Aquestes son les conclusions que he pogut treure de realitzar el projecte final:

Principalment destacar que escollir un projecte de Docker era tot un repte per mi ja que mai havia sentit a parlar d'ell i partia de cap coneixement.

Però al final m'ha acabat agradant poder aprendre una altre nova tecnologia de les que estan utilitzant avui en dia les empreses del sector, doncs no vaig tenir l'oportunitat d'aprendre durant el curs acadèmic. També cal dir que aquest projecte m'ha ajudat a decidir sobre el meu futur professional.

Durant la creació d'imatges de Docker que he anat utilitzant al llarg del projecte han aparegut dificultats, doncs pensava que seria una cosa fàcil i ràpida de fer. Però al cap i a la fi poder solucionar aquests problemes, als quals m'he afrontat durant el projecte, m'han servit d'aprenentatge per possibles solucions a problemes que em pugui trobar en un futur, ja sigui personal o professional.

També m'ha servit per veure com de fàcil és escalar qualsevol aplicació en entorns de producció utilitzant Docker i Kubernetes, de manera immediata pots incorporar nous nodes que permeten destinar més recursos en funció de les necessitats.

Un altra avantatge de Docker i Kubernetes és la facilitat de desplegament un cop coneixes els mètodes a aplicar.

Com a part de validació, he pogut desplegar l'aplicació que he creat mitjançant Docker en un servidor cloud de AWS (veure annex Entorn de Validació). Poder fer el desplegament d'aquesta aplicació, m'ha servit també per conèixer les eines que t'ofereix AWS per poder fer el desplegament de qualsevol aplicació. La majoria de problemes que m'he trobat a l'hora de fer el desplegament han estat relacionats amb errors de codi de l'aplicació.

Bibliografia

- <https://www.youtube.com/watch?v=rmf04ylI2K0>
- <https://www.youtube.com/watch?v=C8e6NJJuiYw>
- <https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/>
- <https://blog.alexellis.io/ingress-for-your-local-kubernetes-cluster/>
- <https://www.youtube.com/watch?v=-NzB4sPZXwU&t=24s>
- <https://www.youtube.com/watch?v=je5WRKxOkWQ&t=646s>
- https://www.youtube.com/results?search_query=deploy+nodejs+app+to+kubernetes
- <https://docs.docker.com/engine/install/ubuntu/>
- https://docs.docker.com/toolbox/toolbox_install_windows/
- https://www.youtube.com/channel/UCrBzMOMcUVV8ryyAU_c6P5g
- <https://hub.docker.com/search?q=&type=image>
- <https://www.docker.com/>
- <https://kitematic.com/>
- <https://www.bluematador.com/blog/safely-removing-pods-from-a-kubernetes-node>
- <https://kubernetes.io/docs/reference/kubectl/cheatsheet/>
- <https://kubernetes.io/docs/concepts/workloads/pods/pod/>
- <https://www.youtube.com/watch?v=eHvKc6hNmhl>
- <https://kubernetes.io/docs/tutorials/stateless-application/expose-external-ip-address/>
- https://www.youtube.com/watch?v=0iMEcrcfG5A&list=PLqRCtm0kbeHA5M_E_Anwu-vh4NWlgrOY_&index=2
- https://www.youtube.com/watch?v=fhYSKEyOs8w&list=PLqRCtm0kbeHA5M_E_Anwu-vh4NWlgrOY
- <https://coderjourney.com/run-kubernetes-locally-using-minikube/>

Annex Entorn de Validació

Per comprovar el correcte funcionament de l'aplicació i de la creació dels contenidors vaig decidir penjar l'aplicació mitjançant Docker. Per ser exactes em vaig crear una conta gratuïta de AWS i vaig crear una instància de EC2, on vaig donar d'alta un servidor on desplegar l'aplicació.

Aquest desplegament en cloud em va servir d'aprenentatge de com poder crear les instances de desplegament i fins hi tot poder canviar les regles de les IPTABLES d'aquesta instància.

```
[carlosguimovart@MacBook-MacBook-Pro-de-Carlos Desktop % ssh -i carlos.pem ec2-user@35.180.35.120
Last login: Mon Jun  1 08:30:04 2020 from static-73-179-27-46.ipcom.comunitel.net
 _ _|_ _|_
 _| | /   Amazon Linux 2 AMI
 ___| \__|_|

https://aws.amazon.com/amazon-linux-2/
10 package(s) needed for security, out of 19 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-29-140 ~]$ ]
```

Les configuracions de seguretat que vaig haver de canviar van ser les següents:

Custom UDP	UDP	3000	0.0.0.0/0	-
Custom UDP	UDP	3000	::/0	-
Custom TCP	TCP	3000	0.0.0.0/0	-
Custom TCP	TCP	3000	::/0	-

```
[ec2-user@ip-172-31-29-140 websockets-docker]$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS              NAMES
4335b8d66313        websockets-docker_app   "docker-entrypoint.s..."  25 hours ago       Up 9 minutes      0.0.0.0:3000->3000/tcp   node-websokets2
9b0054f2497a        mongo                "docker-entrypoint.s..."  3 weeks ago        Up 9 minutes      0.0.0.0:27017->27017/tcp   mongo
[ec2-user@ip-172-31-29-140 websockets-docker]$ ]
```

Aquestes configuracions de les IPTABLES permeten accés des de qualssevol màquina a la nostre aplicació.

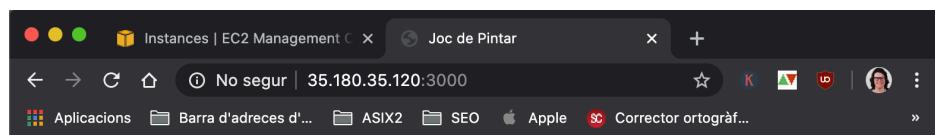
Per poder entrar a l'aplicació és necessari la direcció IP de la instància de AWS i el port corresponent al qual es pot trobar la aplicació.

Per poder crear els contenidors d'una forma òptima vaig penjar el directori on es trobava la meva aplicació a GitHub

 cguinovart Remove the now ignored directory node_modules	×	Latest commit 9c636e1 22 days ago
 views	My first commit of a Docker Project	22 days ago
 Dockerfile	My first commit of a Docker Project	22 days ago
 database.js	My first commit of a Docker Project	22 days ago
 docker-compose.yml	My first commit of a Docker Project	22 days ago
 icon_not_found.png	My first commit of a Docker Project	22 days ago
 index.js	My first commit of a Docker Project	22 days ago
 package-lock.json	My first commit of a Docker Project	22 days ago
 package.json	My first commit of a Docker Project	22 days ago
 routes.js	My first commit of a Docker Project	22 days ago
 user.js	My first commit of a Docker Project	22 days ago

Help people interested in this repository understand your project by adding a README. [Add a README](#)

Fent això vaig poder desplegar els contenidors necessaris mitjançant docker-compose, de la manera ja descrita anteriorment en l'apartat *3.2 Creació d'un docker-compose a partir de l'aplicació*.



The screenshot shows a Mac OS X desktop environment. A browser window titled "Joc de Pintar" is open, displaying the application's interface. The address bar shows the URL "No segur | 35.180.35.120:3000". Below the browser are several application icons in the Dock, including "Aplicaciones", "Barra d'adreses d...", "ASIX2", "SEO", "Apple", and "Corrector ortogràfic...". At the bottom of the screen, there is a navigation bar with links for "Home", "Login", and "Registre". The main content area displays the title "Joc de Pintar" in large, bold, black font. Below the title is a table showing player statistics:

Player	Partidas Guanyades
1 carlos	0
2 aws	0

D'aquesta manera vaig poder accedir a l'aplicació a través d'internet.

Annex Presentació

Desplegament d'aplicacions en Docker i Kubernetes



Carlos Guinovart Galofre
ASIX-DAW
Tutor: Sergi Grau
Curs: 2019-2020



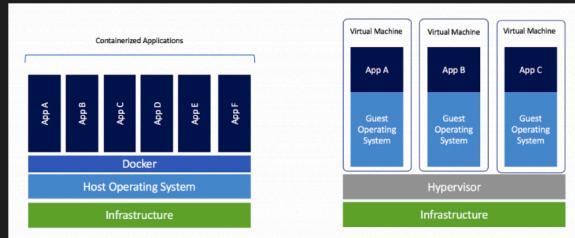
Índex

- Introducció a Docker i Kubernetes.
- Definició de l'aplicació web.
- Funcionalitat de l'aplicació.
- Instal·lació de Docker i Kubernetes.
- Creació d'un contenidor amb Docker.
- Desplegar el contenidor a Kubernetes.
- Conclusions.

Introducció

○ Què és Docker i què ens ofereix

La idea de Docker és poder crear contenidors lleugers i portables per aplicacions de software que puguin executar-se des de qualsevol màquina, independentment del sistema que tingui instal·lat



Introducció

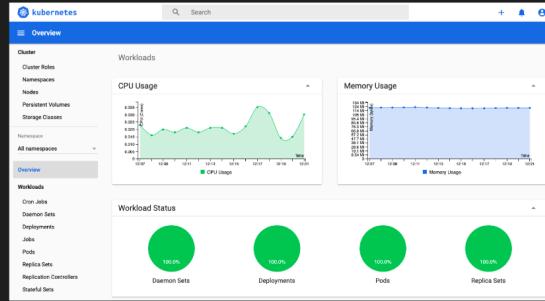
- Imatges en Docker
- Contenidors en Docker



Introducció

- Què es Kubernetes i què ens ofereix

Kubernetes és una plataforma portable i extensible de codi obert per a administrar càrregues de treball i serveis. Kubernetes facilita l'automatització i la configuració declarativa.

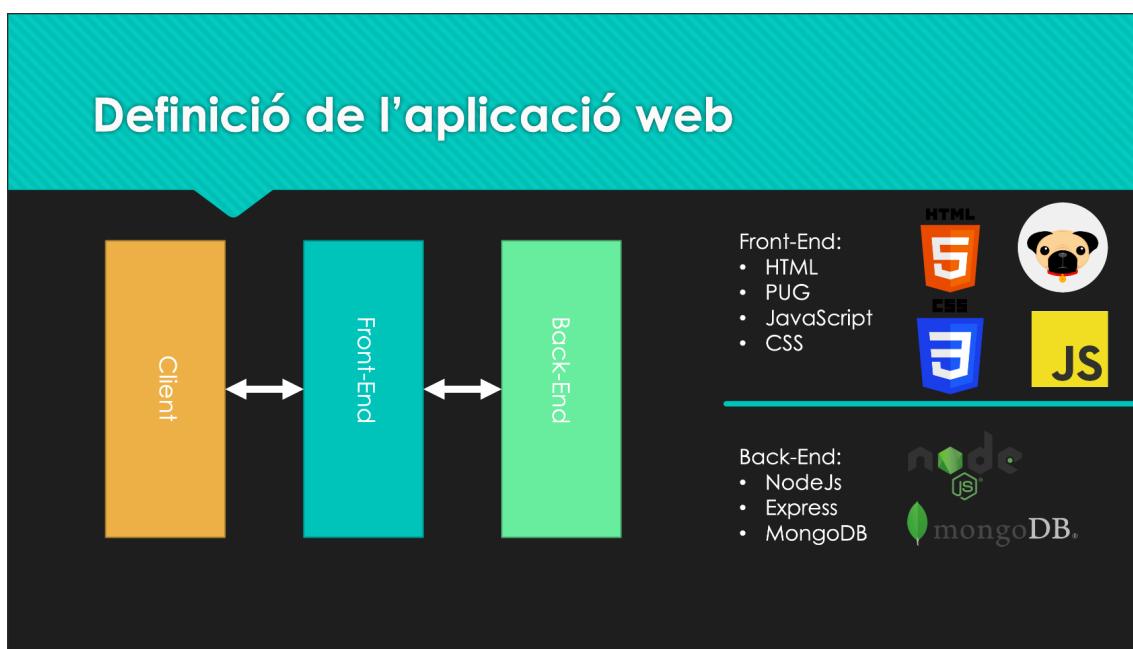


Introducció

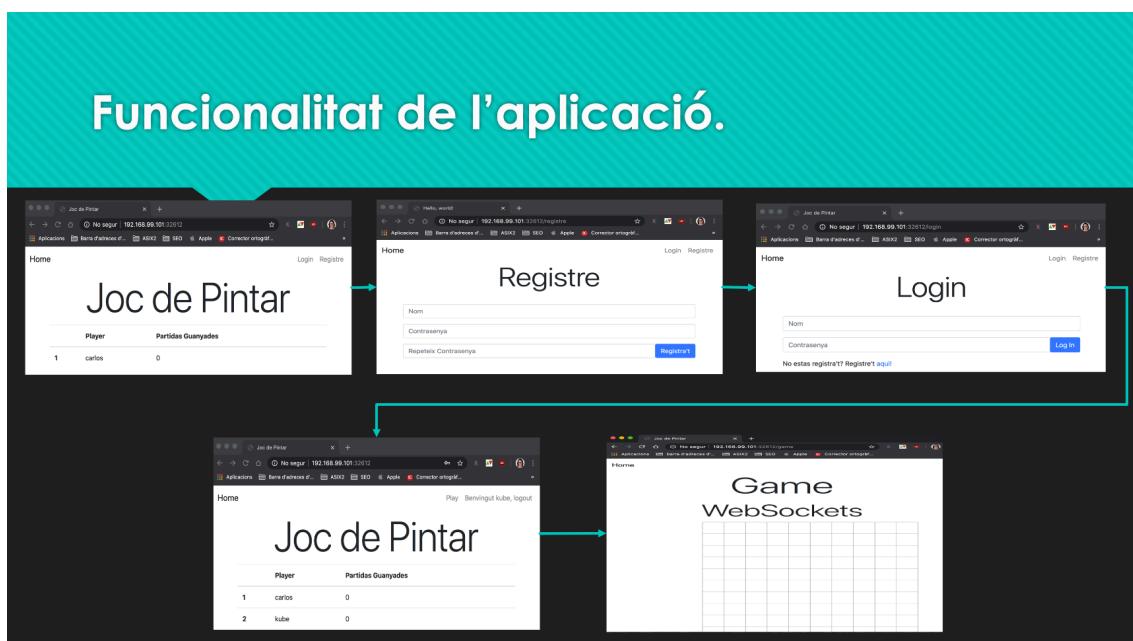
Components de Kubernetes

- Workers
- Deployments
- POD
- Services
- Namespaces

Definició de l'aplicació web



Funcionalitat de l'aplicació.



Instal·lació de Docker i Kubernetes.

Docker

- Linux: Nativa
- Windows: Maquina Virtual de tercers
- MacOS: “Nativa” utilitzant les maquines virtuals del sistema operatiu.



Instal·lació de Docker i Kubernetes.

Kubernetes:

Per utilitzar Kubernetes en local necessitarem MiniKube.
És on corre un clúster o millor dit una instància de Kubernetes en un sol node

Tant com per les 3 versions(Linux, Windows i MacOS) s'haurà d'instal·lar una maquina virtual per simular un servidor al cloud.



Creació d'un contenidor amb Docker.

- **Dockerfile**: és un arxiu de text pla que conté una sèrie d'instruccions necessàries per crear una imatge que, posteriorment, es convertirà en una sola aplicació

```
👉 Dockerfile > ...
1  FROM node:12
2
3  WORKDIR /app
4
5  COPY package*.json ./
6
7  RUN npm install
8
9  COPY . .
10
11 CMD ["npm", "start"]
12
13
```

Creació d'un contenidor amb Docker.

- **Docker-compose**: és una eina per definir i executar aplicacions Docker de diversos contenidors

```
👉 docker-compose.yml
1  version: '3'
2  services:
3    app:
4      container_name: node-websokets
5      restart: always
6      build: .
7      command: npm start
8      ports:
9        - '3000:3000'
10     links:
11       - mongo
12   mongo:
13     container_name: mongo
14     image: mongo
15     ports:
16       - '27017:27017'
```

\$ docker-compose up -d

carlosguinovart@MacBook-Pro-de-Carlos ~ % docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
88e5c3b107a7 websockets-master_app "docker-entrypoint.s..." 3 minutes ago Up 3 minutes 0.0.0.0:3000->3000/tcp node-websokets
f21afe66a7f0 mongo "docker-entrypoint.s..." 3 minutes ago Up 3 minutes 0.0.0.0:27017->27017/tcp mongo

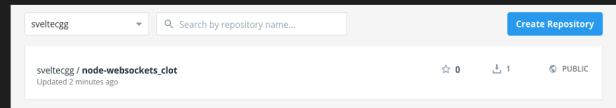
Creació d'un contenidor amb Docker.

- Com penjar la imatge per després ser usada per desplegarla a Kubernetes
- Important! Tenir usuari a Docker Hub i estar logejat.

```
carlosguinovart@MacBook-MacBook-Pro-de-Carlos ~% docker images
REPOSITORY          TAG      IMAGE ID   CREATED        SIZE
m06websockets_app  latest   cf1cff13bf   7 seconds ago  969MB

carlosguinovart@MacBook-MacBook-Pro-de-Carlos m06websockets % docker image tag cf1cff13bf sveltecg/node-websockets_clot

carlosguinovart@MacBook-MacBook-Pro-de-Carlos m06websockets % docker push sveltecg/node-websockets_clot
The push refers to repository [docker.io/sveltecg/node-websockets_clot]
d0fbf7c25e: Preparing [=====] 35.92MB
005257402c07: Mounted from sveltecg/m06websockets_app
```



Desplegar el contenidor a Kubernetes.

```
carlosguinovart@MacBook-MacBook-Pro-de-Carlos m06websockets % minikube start
• Starting control plane node minikube in cluster minikube
• Using the virtualbox driver based on existing profile
• Starting control plane node minikube in cluster minikube
Up to date: 0 nodes
Preparing Kubernetes v1.18.2 on Docker 19.03.8...
Enabled addons: dashboard, default-storageclass, storage-provisioner
Done! Kubectl is now configured to use "minikube"
! /usr/local/bin/kubectl is v1.16.6-beta.0, which may be incompatible with Kubernetes
```

```
kube > / 01-Deployment-NodeJsApp.yaml
13
14  apiVersion: apps/v1
15  kind: Deployment
16  metadata:
17    | name: nodeapp
18  spec:
19    | replicas: 1
20    | selector:
21    |   matchLabels:
22    |     | app: nodeapp
23    | template:
24    |   metadata:
25    |     | labels:
26    |       | app: nodeapp
27    | spec:
28    | containers:
29    |   - name: nodeapp
30    |     image: sveltecg/m06websockets_app
31    |     ports:
32    |       - containerPort: 3000
33    |     env:
34    |       - name: MONGO_URL
35    |         | value: mongodb://mongo:27017/websockets
36    |         | imagePullPolicy: Always
37
```

Desplegar el contenedor a Kubernetes.

```
kube > ! 01-Service-Loadbalancer.yaml
 1 apiVersion: v1
 2 kind: Service
 3 metadata:
 4   name: nodeapp-svc
 5 spec:
 6   selector:
 7     app: nodeapp
 8   ports:
 9     - port: 3000
10       targetPort: 3000
11     type: NodePort
12
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongo-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 256Mi
---
apiVersion: v1
kind: Service
metadata:
  name: mongo
spec:
  selector:
    app: mongo
  ports:
    - port: 27017
      targetPort: 27017
---
```

Desplegar el contenedor a Kubernetes.

```
carlosguinovart@MacBook-MacBook-Pro-de-Carlos kube % kubectl apply -f 01-Mongo-Deployment.yaml
persistentvolumeclaim/mongo-pvc unchanged
service/mongo unchanged
deployment.apps/mongo unchanged
carlosguinovart@MacBook-MacBook-Pro-de-Carlos kube % █
carlosguinovart@MacBook-MacBook-Pro-de-Carlos kube % kubectl apply -f 01-Deployment-NodeJsApp.yaml
service/nodeapp unchanged
deployment.apps/nodeapp configured
carlosguinovart@MacBook-MacBook-Pro-de-Carlos kube % █
carlosguinovart@MacBook-MacBook-Pro-de-Carlos kube % kubectl get all
NAME                                     TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
pod/hello-world-866dcf84d-5rl4           READY     2/2         2h          80:32161/TCP   26h
pod/mongo-64cd45678-8pxnd               READY     2/2         14m         27017/TCP     14m
pod/mongodt-standalone-0-nmc5b          Pending   0/1         0            27017/TCP     13m
pod/nodeapp-676cdcb6dc-pv1hp            Pending   0/1         0            3000:32612/TCP   141m
pod/nodeapp-676cdcb6dc-pv1hp            Running   0/1         0            3000:32612/TCP   141m
service/nodeapp-svc                      NodePort   <none>      10.183.154.19  80:32161/TCP   141m
carlosguinovart@MacBook-MacBook-Pro-de-Carlos kube % █
carlosguinovart@MacBook-MacBook-Pro-de-Carlos kube % kubectl get all
NAME                                     TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
service/kubernetes*                     ClusterIP  <none>      443/TCP       26h
service/kube-dns*                       ClusterIP  <none>      53/UDP,53/TCP  26h
service/nodeapp                          ClusterIP  <none>      27017/TCP     141m
service/nodeapp                         LoadBalancer 10.99.245.94  <pending>     3000:32612/TCP   141m
service/nodeapp-svc                      NodePort   <none>      10.183.154.19  80:32161/TCP   141m
carlosguinovart@MacBook-MacBook-Pro-de-Carlos kube % █
carlosguinovart@MacBook-MacBook-Pro-de-Carlos kube % kubectl get pods
NAME                                         READY   STATUS    RESTARTS   AGE
deployment.apps/hello-world-866dcf84d-5rl4  2/2     Running   0          25h
deployment.apps/mongo-64cd45678-8pxnd       1/1     Running   0          14m
deployment.apps/nodeapp                     2/2     Running   0          141m
carlosguinovart@MacBook-MacBook-Pro-de-Carlos kube % █
carlosguinovart@MacBook-MacBook-Pro-de-Carlos kube % kubectl get replicaset
replicaset.apps/hello-world-866dcf84d      2/2     2          2           25h
replicaset.apps/mongo-64cd45678-8pxnd     1/1     1          1           14m
replicaset.apps/nodeapp-676cdcb6dc        2/2     2          2           141m
carlosguinovart@MacBook-MacBook-Pro-de-Carlos kube % █
carlosguinovart@MacBook-MacBook-Pro-de-Carlos kube % kubectl get statefulset
NAME                                         READY   AGE
statefulset.apps/mongodt-standalone        0/1     22h
carlosguinovart@MacBook-MacBook-Pro-de-Carlos kube % █
```

Desplegar el contenidor a Kubernetes.

The screenshot displays two panels. On the left, a terminal window shows the command 'minikube service nodeapp' being run, which lists a service named 'nodeapp' with port 3000 mapped to URL 'http://192.168.99.101:32612'. A note indicates it's opening in the default browser. On the right, a browser window titled 'Joc de Pintar' shows the application's home page with a single player entry: 'carlos' has 0 wins.

Conclusions

Realitzar aquest projecte de Docker i Kubernetes suposa:

- Assolir coneixement de la tecnologia.
- Experimentar la facilitat de desplegar.
- Ressaltar la facilitat d'escalar recursos de manera àgil.
- Detecció i solució de dificultats tècniques.
- Millora d'habilitats associades a la gestió del projecte: planificació, execució i control d'activitats



Gràcies per l'atenció