Podman

Imatges i repositoris ("registries")

Una imatge OCI és, molt bàsicament, un arxiu "tar" que combina:

*Un directori que conté el sistema de fitxers arrel ("rootfs") dels contenidors que es posaran en marxa a partir d'ella (on hi apareixen les carpetes típiques d'un sistema Linux: "/usr", "/var", "/home", etc), directori que serveix per allotjar tots els fitxers, llibreries, comandes,... necessàries per poder executar el procés concret (*httpd*, *mysqld*, *bash*, etc) que es desitgi

*Un fitxer JSON que conté la configuració del contenidors que es posaran en marxa a partir d'ella. Per exemple, especifica quina comanda concreta (també anomenada "entrypoint") que es trobi dins del "rootfs" s'executarà per a posar en marxa el contenidor, quines variables d'entorn s'estableixen pel contenidor en iniciar-se aquest, quin és el directori de treball del contenidor, etc

Per defecte, el contingut d'un contenidor és volàtil: s'obté a partir d'una còpia en RAM del "rootfs" de la imatge de la qual prové i quan s'apaga el contenidor aquest contingut es perd. De fet, el "rootfs" d'una imatge és de només-lectura. No obstant, és possible afegir, al "rootfs" original d'una determinada imatge (que anomenarem imatge "base"), contingut addicional en forma de capes diferencials (actualitzant-se convenientment el fitxer de configuració JSON): això és el que se'n diu imatge "layered" (amb capes). Moltes imatges disponibles a Internet són imatges "layered" formades a partir d'una determinada imatge "base" a la qual se li ha afegit diferent software i/configuracions addicionals. Cada capa en una imatge "layered" representa la diferència (els canvis) entre ella i la capa immediatament precedent.

Existeixen varis "repositoris" d'imatges (també anomenats "registres") que són públics i que ofereixen imatges ja generades llestes per descarregar (i així no haver-les de crear "a mà"). El "repositori" més important i complet amb diferència, però, és el "Docker Hub" (https://hub.docker.com/search?q=&type=image). Allà podem trobar (amb l'ajuda del seu cercador i sense necessitat d'autenticar-se: l'autenticació només és necessària si s'hi volen pujar imatges) multitud d'imatges ja preparades que ens poden estalviar molta feina. Les imatges allà disponibles poden ser de tipus "base" (les quals normalment es corresponen a imatges que ofereixen el terminal Bash com a programa principal funcionant en un "rootfs" d'Ubuntu o de Fedora o de Suse o ...) o de tipus "layered" (les quals normalment es corresponen a imatges que ofereixen un determinat servei ja preparat i configurat per arrencar, com ara Apache, MySQL, ElasticSearch, HAProxy, etc).

NOTA: Altres repositoris públics d'imatges que poden ser útils en algun moment més enllà del "Docker Hub" són:

https://registry.fedoraproject.org : Mantingut per Fedora (es pot usar anònimament)
https://quay.io/search : Mantingut per RedHat (es pot usar anònimament)
https://catalog.redhat.com/software/containers/explore : Mantingut per RedHat (cal autenticar-se)
https://cloud.google.com/container-registry : Mantingut per Google (cal autenticar-se)
https://azure.microsoft.com/en-us/services/container-registry : Mantingut per Azure (cal autenticar-se)
https://aws.amazon.com/ecr : Mantingut per Amazon (cal autenticar-se)
https://private-docker-registry.com : Mantingut per Docker (cal autenticar-se)

NOTA: És possible implementar un repositori privat per allotjar imatges pròpies. De fet, un repositori d'imatges no es més que un servidor HTTPS que ofereix una API molt determinada i concreta (documentada aquí: https://github.com/opencontainers/distribution-spec) per tal de fer possible la "pujada" i "baixada" d'imatges, la seva inspecció, etc. Software específic que implementa aquesta API (i que, per tant, permet posar en marxa un repositori privat d'imatges molt fàcilment) són: https://github.com/distribution/distribution (lliure, el qual es pot combinar amb el portal de gestió web i d'autorització http://port.us.org, també lliure), https://quay.io (no lliure) o https://jfrog.com/container-registry (no lliure; també es pot fer servir com a SaaS).

Obtenció d'imatges "base" remotes amb *Podman*

Potser el primer pas per començar a introduir-se en el món dels contenidors és buscar una imatge que ens vagi bé pels nostres propòsits al "Docker Hub" (i/o a algun/s del/s repositori/s indicats a la primera "nota" anterior). L'eina que ens permetrà fer això (a més de descarregar la imatge escollida i executar, a partir d'ella, els contenidors que necessitem) és Podman, i concretament, la comanda següent:

podman search unaparaula : "unaparaula" pot ser qualsevol paraula (en minúscules i sense que s'admetin comodins). Si aquesta paraula coincideix amb alguna part de la URL d'alguna imatge disponible (no necessàriament només amb el seu nom o part d'ell), aquesta imatge apareixerà llistada a pantalla, mostrant-se les columnes "Index" (nom del repositori on s'ha trobat), "Name" (URL sencera de la imatge), "Description", "Stars" (mesura numèrica de la popularitat de la imatge), "Official" i "Automated".

NOTA: La recerca es fa als repositoris indicats dins de la llista assignada com a valor de la directiva *unqualified-search-registries=* de l'arxiu "**/etc/containers/registries.conf**" (o, per l'usuari actual només, si existís, de l'arxiu "**~/.config/containers/registries.conf**"). Concretament, la llista que apareix per defecte a sistemes Ubuntu està formada pels repositoris "Docker Hub" i "Quay" mentre que la que apareix per defecte a sistemes Fedora està formada pels repositoris "Docker Hub", "Registry Fedora" i "Registry Access Red Hat". En qualsevol cas, aquesta llista de repositoris per defecte a cercar es podria editar si fos necessari. En qualsevol cas, per saber la sintaxi i significat de totes les directives possibles de l'arxiu "registries.conf", consulteu *man containers-registries.conf*
NOTA: La comanda *podman search* té diversos paràmetres (els quals es poden consultar a la seva pàgina de manual, *man podman-search* o també executant *podman search --help*) com ara *--limit=nº* (per limitar el número màxim d'imatges mostrades per repositori, per defecte són 25), *-f ...* (per filtrar la sortida segons diverses metadades concretes de les imatges, com ara "is-official", "is-automated", etc), *--format=unFormat* (per canviar les columnes i el format de la taula mostrada...per exemple, per veure el resultat obtingut en JSON caldria indicar el format "*json"*; per veure'l en forma de taula mostrant només les columnes "Índex" i "Name", per exemple, caldria indicar el format "*table {{.Index}} {{.Name}}*"),etc.
NOTA: Per obtenir totes les imatges d'un repositori sense usar cap paraula concreta de recerca, s'ha d'escriure el nom del repositori acabat en "/", així: *podman search registry.fedoraproject.org/*

Un cop ja tenim trobada la imatge que volem, ara ens cal descarregar-la al nostre disc dur local. Això es pot fer amb la comanda següent:

podman pull urlImatge o ***podman image pull urlImatge***: "urlImatge" en realitat no és un nom d'imatge sinó que és una URL que té els següent format general:

nom.DNS.Repositori/propietari/nomImatge:tag

on "nom.DNS.Repositori" por ser *docker.io* (pel "Docker Hub"), *registry.fedora.org*, *quay.io*, etc ; "propietari" és típicament el nom de l'organització, equip o persona que va pujar la imatge en qüestió a aquell repositori (serveix com a "separador de noms" per possibilitar que dues imatges de propietaris diferents es puguin anomenar de la mateixa manera), "nomImatge" és el nom de la imatge en sí i "tag" és el nom de la capa superior concreta dins de la imatge que es vol descarregar (o dit d'una altra manera, la "versió").

NOTA: En realitat, la URL de les imatges pot anar precedida de "**protocol://**" (com per exemple *docker://*) però aquesta part no se sol indicar perquè Podman ja és suficientment intel.ligent com per escollir el protocol adient en cada cas per descarregar cada imatge indicada. D'altra banda, el nom indicat del repositori pot anar seguit de ":nº", on nº representa el número de port on escolta el repositori en qüestió si aquest no n'es el port per defecte.
NOTA: En el cas d'usar "Docker Hub", es pot obviar d'escriure també el propietari si aquest és "library".

Indicar "nom.DNS.Repositori" és opcional. Si no s'indica cap repositori en concret on anar a trobar la imatge a descarregar, per defecte, Podman la buscarà en els repositoris indicats a la directiva *unqualified-search-registries=* de l'arxiu "/etc/containers/registries.conf" (veieu primera nota d'aquesta pàgina) i si troba vàries imatges coincident llavors oferirà a l'usuari triar la que desitji de forma interactiva. D'altra banda, indicar una "tag" també és opcional. Si no s'indica cap "tag", per defecte es descarregarà una imatge que tingui la "tag" anomenada "**latest**" (si n'hi ha).

NOTA: Per saber les "tags" que incorpora una determinada imatge allotjada en un repositori, es pot fer servir l'eina Skopeo (instal.lada amb el paquet homònim), la qual serveix, a més de per realitzar tasques generals relacionades amb la gestió d'imatges dins de repositoris (com ara descarregar i pujar imatges -cosa que Podman, de fet, també ho pot fer-, moure imatges entre repositoris, etc), pel que ens interessa ara: inspeccionar les metadades de la imatge sense haver de descarregar-la. Concretament, la comanda seria similar a aquesta: *skopeo inspect docker://docker.io/library/fedora* (on aquí indicar "protocol://" és obligatori). La llista de "tags" disponibles apareixerà a pantalla sota l'apartat "RepoTags" (i a més podrem veure més dades com la data de creació de la imatge, la seva arquitectura, els checksums de les capes, etc)

Així doncs, els següents valors per "url/Imatge" són equivalents: "*docker.io/library/fedora:latest*", "*docker.io/library/fedora*", "*docker.io/fedora:latest*" (se sobreentén que es busca la imatge dins el propietari per defecte de "Docker Hub", que és "library"), "*docker.io/fedora*", "*library/fedora:latest*" (s'iterarà per tots els repositoris configurats per defecte fins trobar la/es imatge/s coincident/s) i "*library/fedora*". No obstant, si indiquem "*fedora:latest*" (o "*fedora*") en un sistema Fedora, la imatge descarregada serà diferent pel motiu indicat a la següent nota.

NOTA: Cal tenir en compte que un repositori pot estar configurat per oferir les imatges directament "des de l'arrel" (és a dir, sense haver d'especificar cap "propietari"). Això vol dir que, per exemple, si a sistemes Fedora indiquem com a valor de "urlImatge" directament el valor "fedora" (o "fedora:latest") obtindrem una imatge del repositori "registry.fedoraproject.org" perquè aquest té una imatge amb aquest nom directament "penjant" del seu nom DNS. Òbviament, aquesta imatge, tot i que sigui d'un Fedora igual, no serà pas la mateixa que "library/fedora" perquè estan en repositoris diferents (i s'han creat per equips diferents de maneres diferents).

Si Podman s'executa amb un usuari sense privilegis, el magatzem on es guardaran les imatges locals es troba dins de la carpeta "**~/.local/share/containers/storage**". Si Podman s'executa com a "root", en canvi, el magatzem on es guardaran les imatges locals es troba dins de la carpeta "**/var/lib/containers/storage**"

NOTA: Les ubicacions anteriors es poden personalitzar editant l'arxiu de configuració "**~/.config/containers/storage.conf**" (en el cas de voler configurar el magatzem "rootless") o "**/usr/share/containers/storage.conf**" i "**/etc/containers/storage.conf**" (en el cas de voler configurar el magatzem "root"). En tot cas, la pàgina del manual d'aquest arxiu de configuració, *man storage.conf*, és molt recomanable consultar per conèixer el significat i sintaxi de les directives que s'hi poden incloure.

NOTA: Un altre arxiu important de configuració és "**~/.config/containers/containers.conf**" (si afecta només a l'usuari actual) o "**/usr/share/containers/containers.conf**" i "**/etc/containers/containers.conf**" (si afecta a tothom) , on es pot configurar el valor per defecte dels paràmetres utilitzats per la comanda *podman* i relacionades. Per exemple, es pot indicar el "CR" que Podman usarà per defecte (el qual també es pot indicar amb el paràmetre *--runtime*), entre altres coses. Per més informació sobre el significat i sintaxi de les directives que hi pot incloure, consulteu *man containers.conf*

Una comanda que ens serà molt útil al llarg d'aquest document és ***podman info*** *(o* ***podman system info,*** és el mateix), la qual mostra informació tant del sistema amfitrió (com la versió del kernel, espai usat i disponible de swap, etc) com del propi Podman (com la seva versió, les rutes actuals dels arxius de configuració mencionats als paràgrafs anteriors, tota la pròpia configuració allà establerta, entre la qual podem destacar el mètode d'emmagatzematge usat -el que se'n diu "driver"-, etc)

Gestió bàsica d'imatges locals amb *Podman*

podman image list o ***podman image ls*** o ***podman images*** : Llista les imatges disponibles localment. Mostra el repositori d'on es va obtenir i quan, el seu "tag", el seu "id" (un valor aleatori però únic generat en el moment de la descàrrega de la imatge al disc local) i el seu tamany. Paràmetres que són interessants:

--format=unFormat Canvia les columnes i el format de la taula mostrada, igual que el paràmetre homònim vist amb *podman search*. En aquest cas, però, els noms de les columnes possibles a triar seran diferents (veieu *man podman-images*)
--sort=cadena Ordena per la cadena *created*, *id*, *repository*, *size* o tag (per defecte és "created")
-f filtre Mostra les imatges que coincideixen amb el filtre indicat, el qual pot ser *since=nomImatge* (mostra les imatges creades després de la indicada), *before=nomImatge* (mostra les imatges creades abans de la indicada) o *readonly={true|false}*, entre d'altres.
-q Mostra només els "id" de les imatges

podman image rm idImage o ***podman rmi idImatge*** : Esborra la imatge (indicada pel seu "id") del disc local. Es poden indicar més d"un "id" separats per espais per esborrar-los tots de cop. També es pot escriure ***podman rmi -a*** per esborrar totes les imatges existents. Paràmetres que són interessants:

-f Elimina tots els contenidors que estan usant les imatges indicades abans d'esborrar aquestes

NOTA: Es pot indicar una imatge, en comptes del seu "id", per la seva URL completa, així: *podman rmi docker.io/library/fedora:latest*

NOTA: Si tenim compte d'usuari, podrem "pujar" la nostra imatge al "Docker Hub" (o qualsevol altre repositori compatible que volguem) amb la comanda ***podman push idImatge docker://url/Imatge*** (o ***podman image push idImatge docker://urlImatge***, on "urlImatge" és la URL de la imatge desitjada, que ha de tenir el mateix format que l'explicat en veure *podman pull*). L'usuari/contrasenya a usar es pot haver preestablert abans amb la comanda ***podman login nom.DNS.Repositori*** (on aquesta dada es pregunta interactivament -o bé mitjançant els paràmetres ***-p*** i ***-u*** - i es guarda en el fitxer "***/run/user/UID/containers/auth.json***" per posteriors usos) o a la pròpia comanda *podman push* amb el paràmetre ***--creds=nomUsuari:contrasenya*** . Si no es fa cap dels dos mètodes d'autenticació anteriors, l'usuari i contrasenya es preguntaran interactivament. L'usuari introduït ha de tenir "permisos d'escriptura" dins del espai de noms "propietari" per tal de poder ubicar-hi la imatge correctament.

Creació d'imatges pròpies amb *Podman* i "Containerfiles"

Encara que existeix la possibilitat de convertir en una nova imatge el contingut d'un determinat contenidor que s'hagi diferenciat prou de la seva imatge original (és a dir, traspassar de RAM a un nou fitxer en disc els canvis apareguts), la manera més comuna de crear imatges personalitzades noves a partir d'una determinada imatge original és mitjançant l'ús de fitxers "Containerfile". Aquest procés consisteix en:

1.-Crear un fitxer de text (anomenat "Containerfile") que defineixi els següents elements:

*La imatge base a partir de la qual es vol començar la personalització del seu "rootfs". Això es fa mitjançant la directiva **FROM;** la imatge a usar es pot especificar usant la mateixa sintaxi que a la comanda *podman pull* i, si no es troba en local, es descarregarà automàticament

*Les diferents comandes que s'hi volen executar en aquesta imatge base per tal d'obtenir el nou "rootfs" de la imatge personalitzada. Això es fa mitjançant la directiva **RUN** i pot haver-hi més d'una si es vol executar vàries comandes separades una rera l'altra. Aquestes comandes solen tenir a veure amb instal.lacions de paquets, edicions de fitxers, etc i totes elles s'executaran amb permisos d'administrador dins de la imatge (de fet, no hi existeix cap altre usuari a no ser que es creïi precisament amb *RUN)*. Cada directiva *RUN* implica la creació d'una capa nova dins de la imatge...si es vol minimitzar aquest fenòmen es poden concatenar vàries comandes en una sola directiva *RUN* fent ús de l'operador "&&"; també és comú executar la comanda *dnf clean all* (o similar) per encongir el tamany final de la imatge.

*El procés intern que es posarà en marxa automàticament quan es llenci un contenidor a partir de la imatge personalitzada. Això es fa mitjançant la directiva **CMD**, la qual té la forma *CMD["/ruta/executable","param1","param2",...]* . Aquesta directiva, però, és opcional: si no s'indica (o si es vol sobreescriure), caldrà especificar el procés en qüestió en el moment de llençar el contenidor. Si aquest procés intern fos un servei, també caldrà indicar, a més, en quin nombre de port estarà escoltant amb la directiva **EXPOSE**.

*Una altra opció molt habitual és **COPY**, la qual serveix per "ficar" dins de la imatge (dins de la carpeta que indiquem) els fitxers i/o carpetes que desitgem (les quals haurem d'ubicar a la mateixa carpeta on es troba l'arxiu "Containerfile" per tal de què COPY les trobi). Aquesta directiva té la forma *COPY fitxer1 fitxer2 subcarpeta1 /ruta/carpeta/dins/imatge*) on "fitxer1", "fitxer2" i "subcarpeta1" són els noms dels diversos elements (els quals, repetim, han d'estar dins de la mateixa carpeta on es troba el "Containerfile") que es copiaran dins de la ruta (absoluta) indicada com a darrer paràmetre.

NOTA: Altres directives comunes són **LABEL** (amb la forma *LABEL clau1="valor1" clau2="valor2"* ... la qual serveix per afegir metadades personalitzades a la imatge, tals com la web o el correu del creador de la imatge, etc; se sol indicar just després de *FROM* i pot estar repetida), **ENV** (amb la forma *ENV var1="valor1" var2="valor2"* ... la qual serveix per establir valors de variables d'entorn que seran reconeguts per les comandes executades amb *RUN*) o **USER** (amb la forma *USER nomUsuari*, que serveix per indicar l'usuari amb el què s'executaran les comandes indicades a les següents directives *RUN*; òbviament, aquest usuari ha d'existir prèviament), **ENTRYPOINT** (similar a *CMD* però no permet la seva sobreescriptura mitjançant paràmetres de *podman run*), **WORKDIR** (amb la forma *WORKDIR /ruta/carpeta* ... la qual serveix per indicar la ruta de la carpeta de treball -a mode de "cd"- on s'hi executaran les posteriors ordres *RUN*, *COPY* o *CMD*), etc. La documentació oficial de totes les directives possibles en un "Containerfile" és https://docs.docker.com/engine/reference/builder/

Un exemple de "Containerfile" pot ser el següent, el qual aprofita la imatge "fedora:latest" (la qual pot estar ja descarregada al nostre disc local però no és obligatori: si no s'hi troba, es procedirà automàticament a descarregar del repositori remot adient...la sintaxi per indicar el nom/URL de la imatge és el mateix que l'emprat a *podman pull*) per instal.lar-hi el servidor HTTP Apache. A més, es copia dins del seu "DocumentRoot" un determinat fitxer "index.html" ja preparat (ubicat a la mateixa carpeta on es troba el "Containerfile") i s'indica que en iniciar-se un contenidor a partir d'aquesta imatge sempre es posarà en marxa automàticament el procés Apache en primer pla, i escoltant al port nº80.

```
FROM fedora:latest
RUN dnf -y update && dnf -y install httpd && dnf clean all
COPY index.html /var/www/html
CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]
EXPOSE 80
```

2.- Crear la nova imatge a partir del "Containerfile" configurat al pas anterior. Això es pot fer executant la comanda ***podman image build /ruta/carpeta/on/es/troba/el/fitxer/Containerfile -t nomImatge:tag*** Tant el nom de la imatge com el "tag" es poden escollir lliurement. La nova imatge apareixerà com una més a la llista mostrada per *podman images* (només que el nom del seu repositori serà "localhost").

NOTA: La comanda *build* inspecciona recursivament totes les subcarpetes ubicades sota la carpeta on es troba el "Containerfile"; per tant, és recomanable no tenir-hi fitxers o subcarpetes que no siguin necessaris per la creació de la imatge.

A partir d'aquí podrem executar contenidors basats en la imatge generada (tal com explicarem al proper apartat). En l'exemple mostrat, podrem provar d'accedir des de l'exterior amb un navegador al servidor Apache2 funcionant dins del contenidor per veure si funciona.

NOTA: Existeix la possibilitat de signar les imatges creades per tal de garantir que no puguin ser alterades. Això es pot fer (tal com s'explica aquí: https://github.com/containers/libpod/blob/master/docs/tutorials/image_signing.md) amb Podman, però per aquestes tasques és millor fer servir l'eina Skopeo, la qual també permet -com Podman- "pujar" (és a dir, "push") la imatge recentment creada -i signada- a algun repositori remot on tinguem compte d'usuari.

Creació d'imatges pròpies amb *Mkosi*

L'eina *mkosi* (https://github.com/systemd/mkosi) serveix per generar "rootfs" mínims de diferents distribucions com ara Ubuntu, Debian, Arch, Fedora, Suse i més. Si volem personalitzar al màxim la nostra imatge, podem escollir crear la imatge base amb aquesta eina en comptes de partir d'una imatge base ja prefeta. L'únic requisit és que el "rootfs" generat ha de estar en format "tar" (o "tar.gz" o "tar.xz") per tal de què Podman el pugui importar correctament i generar així la imatge corresponent. Cal tenir en compte, no obstant, que aquesta eina necessita executar-se amb permisos d'administrador. Els seus paràmetres més importants són:

-d nomDistribució : "nomDistribució" pot ser "ubuntu", "debian", "fedora", "arch", "suse"...
-r release : "release" pot ser "focal", "buster", "32"...segons la distribució escollida
-t tar : format que tindrà el "rootfs" generat; aquí només ens serveix el format "tar"
-o /ruta/fitxer.tar : ruta i nom del fitxer "tar" resultant
-p unPaquet,unAltre... : llista de paquets "extra" a afegir al "rootfs" per defecte

Així doncs, si volguéssim generar un "rootfs" mínim de Fedora, podríem executar la següent comanda: *sudo mkosi -d fedora -r 35 -t tar -o fedora.tar -p dnf,iproute,iputils,nano*

Un cop tenim el "rootfs", per generar la imatge corresponent (i ja de pas, afegir-la a la llista d'imatges locals reconegudes per Podman) només cal executar la comanda: ***podman import fitxer.tar localhost/nomImatge:tag*** (o ***podman image import fitxer.tar localhost/nomImatge:tag***, on "nomImatge" pot ser qualsevol i "tag" també -i opcional-). Si volem indicar el valor concret d'alguna opció que hauríem escrit en un hipotètic "Containerfile", podrem fer-ho afegint el paràmetre ***-c OPCIÓ=valor*** (com per exemple *-c CMD ["/bin/bash"]*); aquest paràmetre es pot indicar les vegades que sigui necessari però cal saber que no funciona per les opcions *RUN* i *COPY*.

> NOTA: Existeix també la comanda ***podman export idContenidor -o fitxer.tar*** (o ***podman container export idContenidor -o fitxer.tar***), la qual serveix per exportar a un fitxer "tar" el contingut d'un determinat contenidor. Aquesta comanda és útil si volem moure contenidors entre diferents màquines amfitriones: el podem exportar en la màquina origen i, un cop copiar l'arxiu "tar" a la màquina destí, importar-lo en forma d'imatge amb *podman import*.

Execució de contenidors amb *Podman*

Un cop tenim la imatge descarregada localment podem posar en marxa a partir d'ella els contenidors que volguem amb la comanda *podman run*. Aquesta comanda té moltes opcions que les estudiarem segons volguem iniciar un contenidor interactiu (entrant-hi en un shell) o bé un contenidor no interactiu (que romandrà encés "en segon pla"; molt útil si el procés a executar-hi és un servei/dimoni).

> NOTA: No cal ni tan sols que la imatge estigui prèviament descarregada localment: si no ho està, la comanda *podman run* realitzarà un *pull* automàtic en aquell moment

En el primer cas, normalment només caldrà executar aquesta comanda: ***podman [container] run -it --rm --name nomCont urlImatge [/bin/bash]*** on:

> ***/bin/bash*** és la ruta absoluta del shell que volem executar al contenidor (en realitat, es podria indicar qualsevol altra comanda present dins del contenidor). Si la imatge va ser creada amb una opció com *CMD["/bin/bash"]* llavors aquest paràmetre no caldrà especificar-ho
> ***urlImatge*** és la URL de la imatge local triada (té el mateix format que l'explicat en veure *podman pull*)
> ***--name nomCont*** és el nom que li donarem al contenidor. No és un paràmetre obligatori però sí molt recomanable: si no s'indica Podman li assignarà un nom aleatori. En qualsevol cas, tant el nom com el "id" (que aquest sí que es genera automàticament) ens servirà per referenciar al contenidor en qüestió.
> ***--rm*** elimina el contenidor un cop s'ha sortit de la sessió interactiva. Útil per no tenir contenidors (parats) que no es faran servir més (per defecte els contenidors generats perviuen un cop s'han aturat per reaprofitar-los més endavant). D'aquesta manera es conserva espai i es pot reutilitzar el mateix nom de contenidor múltiples vegades.
> ***-i*** prepara el contenidor per implementar una sessió interactiva
> ***-t*** obre un pseudo-terminal TTY per ubicar-hi la sessió interactiva

El resultat d'executar la comanda anterior hauria de ser l'entrada a un shell Bash on es vegi el prompt canviat indicant que ens trobem a l'arrel del "rootfs" (això és fàcil comprovar-ho executant per exemple *ls*). També podem comprovar (per exemple amb la comanda *whoami*) que dins del contenidor som l'usuari "root". És interessant, d'altra banda, comprovar amb *uname -a* que el kernel que funciona dins el contenidor és, com no podia ser d'una altra manera, el kernel de la màquina amfitriona. Per sortir del shell del contenidor (aturant-lo automàticament llavors) i tornar al terminal de la màquina amfitriona haurem d'executar *exit*.

> NOTA: Tot i que dins del contenidor siguem "root", la comanda executada com a procés dins d'ell (en aquest exemple, el shell Bash) no pot veure més enllà del contenidor. Concretament, aquest procés per defecte només veu com a sistema de fitxers el proporcionat per la imatge, no veu cap procés de fora del contenidor i només pot accedir a la xarxa de l'amfitrió a través de NAT (per defecte es crea dins del contenidor una tarja de xarxa "virtual" amb una IP privada de la xarxa **10.0.2.0/24** -assignada via un sistema DHCP intern- la qual pot accedir a l'exterior a través de la tarja de l'amfitrió funcionant com a porta d'enllaç amb la IP **10.0.2.2**). A més, pot tenir assignat límits d'ús de CPU o de RAM mitjançant "cgroups2". Tot això fa que els contenidors siguin una gran eina per aïllar entorns entre sí d'una manera segura.

En el segon cas, normalment executarem una comanda similar a ***podman [container] run -d -p 1234:80 --rm --name nomCont urlImatge*** on:

-d serveix per deslligar l'entrada/sortida del contenidor del terminal actual. O dit d'una altra manera, serveix per executar el contenidor en segon pla (molt útil, doncs, per quan el procés que volem executar es tracta d'un dimoni/servei). Si no l'escrivim, el terminal quedarà bloquejat (fins que matem el contenidor pulsant CTRL+C). Si l'escrivim, el control del terminal retornarà a nosaltres (mostrant-se per pantalla l'identificador autogenerat del contenidor acabat de crear) mentre aquest es mantindrà en marxa igualment.
-p x:y indica quin port *x* de la màquina amfitriona estarà lligat al port *y* exposat pel contenidor (és a dir, el que està definit amb l'opció *EXPOSE* o similar en generar la imatge). És a dir, "mapeja" el port *x* de l'amfitrió al port *y* del contenidor. La idea és que per tal de poder connectar al port *y* no es pot fer directament (aquest port està "ocult") sinó que cal fer-ho a través del port *x* de la màquina amfitriona a mode de "túnel". Així doncs, si tenim per exemple un contenidor executant el servidor Apache2 escoltant al port nº80 i hem vinculat aquest port nº80 al port nº1234 de la màquina amfitriona, per accedir-hi a ell des del navegador de la màquina amfitriona (o de qualsevol altra màquina remota) haurem d'escriure http://ip.maq.amfitriona:1234

NOTA: Es pot precedir *x* d'una IP (més dos punts; ":") si la màquina amfitriona tingués vàries IPs i només es vol realitzar el mapeig per la IP indicada. És a dir, si volem que un servidor Apache2 només estigui accessible si hi connectem al port nº1234 des de *localhost*, podríem fer *-p 127.0.0.1:1234:80*
NOTA: El valor de *x* por ser igual a *y* però només si és major de 1024 Podman podrà realitzar el mapeig sense haver d'executar-se com a "root"
NOTA: Si volem que els ports mapejats siguin UDP, cal afegir després de y la cadena "/udp". Així per exemple: *-p 1234:53/udp*
NOTA: Per saber els mapeigs de ports que té un determinat contenidor en marxa es pot executar la comanda ***podman port idContainer*** (o ***podman container port idContainer***). Per saber aquesta informació per tots els contenidors actualment en marxa es pot executar ***podman port -a*** (o ***podman container port -a***).

Un cop executada la comanda anterior, una manera de comprovar si el servei/dimoni contingut al contenidor està funcionant és observar la sortida de la comanda *ss -tln* a la màquina amfitriona: hauríem de veure que el port *x*, efectivament, es troba escoltant. D'altra banda, si el servei contingut és de tipus web (com l'Apache) podríem comprovar si podem accedir a la seva pàgina inicial executant *curl http://ip.maq.amfitriona:1234* (o escrivint aquesta mateixa direcció al navegador).

La comanda *podman run* admet molts més paràmetres, la majoria dels quals es poden fer servir indistintament ja sigui en contenidors interactius o en contenidors de serveis. D'entre ells en podem destacar:

-e VARIABLE=valor estableix un valor a la variable d'entorn indicada, la qual serà tinguda en compte pel procés que s'executarà dins el contenidor (és a dir, l'invocat per *CMD*, el qual pot ser perfectament un shell script). Pot repetir-se els cops que sigui necessari. També es poden escriure dins d'un fitxer múltiples parelles *VARIABLE=valor* (una a cada línia) i llavors fer-ne referència amb el paràmetre ***--env-file /ruta/fitxer***

-v /ruta/carpeta/en/amfitrio:/ruta/carpeta/en/contenidor "mapeja" el contingut de la carpeta de la màquina amfitriona indicada a l'esquerra del ":" dins de la carpeta del contenidor indicada a la dreta. Es permet així que el contenidor pugui accedir a fitxers o subcarpetes de fora d'ell mateix. Aquesta opció és molt útil, per exemple, per escriure en arxius de registre centralitzats a l'amfitrió o per accedir a dades o configuracions comunes a molts contenidors que poden variar al llarg del temps (i que, per tant, no és bona idea que es guardin a la imatge) o, fins i tot, per poder executar binaris que només es troben a la màquina amfitriona si fem un mapeig com *-v /usr/sbin:/usr/sbin* o similar. Si es vol que el "mapeig" sigui de només lectura (és a dir, que el contenidor pugui veure el contingut de la carpeta a l'amfitrió però no en pugui modificar res, cal afegir després de la ruta de la dreta l'apèndix "***:ro***", així: *-v /ruta/amf:/ruta/cont:ro* . Cal tenir en compte, en qualsevol cas, que les rutes hauran de ser sempre absolutes i que hauran d'existir prèviament a l'ús d'aquest paràmetre.

NOTA: El paràmetre *-v* és una drecera d'un tipus de muntatge concret que es pot fer de forma molt més flexible i variada amb el paràmetre general ***--mount***

NOTA: Es poden especificar "mapeigs" que es muntaran per defecte en tots els contenidors en fer *podman run* (o *podman start*) si els indiquem dins de l'arxiu "**/usr/share/containers/mounts.conf**" o "**/etc/containers/mounts.conf**" (per tots els usuaris de la màquina amfitriona) o "**~/.config/containers/mounts.conf**" (per l'usuari actual). El contingut d'aquest fitxer ha de ser el mateix que el valor del paràmetre *-v* (un per línia si s'indiquen varis "mapejos"). Per més informació consulteu *man containers-mounts.conf*

Existeix també un paràmetre anomenat ***--net*** que permet configurar el mode de xarxa del contenidor. No obstant, l´únic mode que funciona executant Podman sense privilegis d'administrador és el que hem fet servir fins ara, el "mode NAT". D'altra banda, cal saber que en el cas d'executar Podman amb privilegis d'administrador, el mode de xarxa sel.leccionat per defecte (és a dir, sense especificar cap mode en concret) és el mode "bridge" ("adaptador pont") el qual genera una interfície de xarxa a la màquina amfitriona (anomenada "cni-podman0") i una altra al contenidor, ambdues de la xarxa **10.88.0.0/16** (la IP del contenidor es pot indicar manualment amb el paràmetre ***--ip=x.x.x.x*** si es vol; en tot cas, la tarja de l'amfitriona serà la porta d'enllaç de la del contenidor), a més d'una altra tarja a l'amfitrió anomenada "veth" que serveix per connectar internament des de l'amfitrió a la "cni-podman0" (i d'allà, a totes les IPs dels diferents contenidors en mode "bridge"). Per saber més opcions de *podman run* (n'hi ha moltes més!), consulteu la seva pàgina de manual, *man podman-run*

NOTA: Desgraciadament, el mode "bridge" de Podman només permet accedir als contenidors des de la màquina amfitriona però no des d'altres màquines remotes. Per aconseguir això caldria connectar-les d'alguna manera a la xarxa 10.88.0.0/16. En aquest sentit, la configuració establerta per defecte amb el mode "bridge" (IP de xarxa, IP de la porta d'enllaç, nom de la tarja en la màquina amfitriona, etc) es troba a l'arxiu "**/etc/cni/net.d/87-podman-bridge.conflist**". Es poden crear altres configuracions (és a dir, altres arxius "conflist") amb la comanda ***sudo podman network create...***) i la que es vulgui fer servir per defecte d'entre les vàries que n'hi puguin haver es pot escollir a la directiva "default_network" de l'arxiu "/usr/share/containers/containers.conf" (o millor, a una còpia que se'n faci a "/etc/containers/containers.conf" per no modificar l'arxiu de configuració per defecte)

Software necessari a la màquina amfitriona per a què Podman pugui funcionar sense haver de ser "root":

*El paquet "**slirp4netns**" (https://github.com/rootless-containers/slirp4netns) permet als contenidors "root-less" implementar el "mode NAT" (també anomenat "mode user") per tal de tenir configuració de xarxa.

*El paquet "**fuse-overlayfs**" (https://github.com/containers/fuse-overlayfs) permet als contenidors "root-less" gestionar l'emmagatzematge de les imatges. Per usar-lo cal que a l'arxiu "storage.conf" estigui sel.leccionat el driver "overlay" (sota la secció [storage]) i que la ruta de l'executable *fuse-overlayfs* sigui el valor de l'opció "mount_program" (sota la secció [storage.options]). Per defecte això ja és així, afortunadament.

*El paquet "**criu**" (https://www.criu.org) permet disposar de les opcions *podman checkpoint/restore* (les estudiarem més endavant).

Gestió bàsica de contenidors amb *Podman*

A continuació es llisten algunes de les comandes més habituals a l'hora de treballar amb contenidors:

podman ps o ***podman ls*** o ***podman list*** o ***podman container ps*** o ***podman container ls*** o ***podman container list*** mostra els contenidors que estan en marxa. Concretament, de cadascun d'ells mostra el seu "id" i -si en té- el seu nom, la URL de la imatge de la qual prové, la comanda que genera el seu procés (opció CMD), quan fa que va ser creat, el seu estat ("restarting", "running" -i mostra des de quan està en marxa-, "exited" o "paused") i el mapeig de ports -si n'hi ha-. Si hi afegim el paràmetre ***-a*** mostra llavors tots els contenidors (en marxa o aturats) existents a la màquina. Els contenidors aturats es mantenen a la màquina amfitriona (a no ser que se'ls hagi invocat amb el paràmetre *–rm* de *podman run*) per tal de mantenir les eventuals dades -incloent logs- que allotgin; es poden llistar aplicant-hi un filtre amb el paràmetre ***-f***, així: *podman ps -a -f status=exited*

podman exec {idContenidor|nomContenidor} comanda ... o ***podman container exec {idContenidor|nomContenidor} comanda ...*** executa la comanda indicada (amb els seus eventuals paràmetres) dins del contenidor indicat (el qual haurà d'estar en marxa, lògicament). Si la comanda fos interactiva (per exemple, la consola del PostgreSQL) es pot afegir els paràmetres ***-it*** per obrir un pseudoterminal interactiu.

podman stop {idContenidor|nomContenidor} o ***podman container stop {idContenidor|nomContenidor}*** atura un contenidor preexistent (que estigui funcionant en segon pla) a la màquina amfitriona. Si indiquem el paràmetre ***-a*** en comptes del "id"/nom del contenidor, s'aturaran tots els contenidors que estiguin iniciats en aquell moment.

podman start {idContenidor|nomContenidor} o ***podman container start {idContenidor|nomContenidor}*** inicia de nou un contenidor preexistent a la màquina amfitriona

podman rm {idContenidor|nomContenidor} o ***podman container rm {idContenidor|nomContenidor}*** elimina un contenidor preexistent a la màquina amfitriona (el contenidor haurà d'estar aturat per a què l'eliminació es pugui realitzar). Si indiquem el paràmetre ***-a*** en comptes del "id"/nom del contenidor, s'eliminaran tots els contenidors existents (i aturats) a la màquina amfitriona. Es pot verificar que, efectivament, el contenidor en qüestió s'hagi eliminat observant la sortida de *podman ps -a*

podman top {idContenidor|nomContenidor} [col1] [col2] ... o ***podman container top {idContenidor| nomContenidor} [col1] [col2] ...*** mostra els processos actualment executant-se dins del contenidor indicat. Per defecte mostra, de cada procés, les columnes USER, PID, PPID, %CPU, ELAPSED, TTY, TIME i COMMAND (és a dir, similar a la sortida de *ps -ef*) però es poden indicar quines columnes en concret es volen veure afegint-les al final de la comanda (la llista de les columnes possibles juntament amb la descripció del seu significat es troba a *man podman-top*)

podman stats {idContenidor|nomContenidor} o ***podman container stats {idContenidor|nomContenidor}*** mostra estadístiques en temps real sobre l'ús de recursos del contenidor indicat (bàsicament, %CPU, %MEM, NET IO i BLOCK IO). Es pot afegir el paràmetre ***--format=unFormat*** per especificar les columnes concretes a mostrar *(*seguint el mateix patró que vam veure amb *podman search*; veieu *man podman-stats* per més detalls) *o* ***–no- stream*** per mostrar només una estadística puntual (com una "foto") sense seguiment en temps real, entre altres.

podman logs {idContenidor|nomContenidor} o ***podman container logs {idContenidor|nomContenidor}*** mostra els registres generats pel contenidor indicat. Es pot afegir el paràmetre ***-t*** (per mostrar el "timestamp" de cada log), el paràmetre ***-f*** (per seguir en temps real la generació de nous logs, tipus *tail -f*), , ***--tail=nº*** (per mostrar només les darreres nº línies) o ***–since=yyyy-mm-ddThh:mm:ss*** (per mostrar només els logs a partir de la data indicada), entre d'altres

podman container inspect {idContenidor|nomContenidor} mostra totes les metadades del contenidor indicat, les quals informen de la seva configuració. Per exemple, per esbrinar la IP d'un contenidor anomenat "lala" es podria fer *podman container inspect lala | grep "IPAddress"*. Es pot usar el paràmetre ***-f*** per filtrar (on les claus de l'objecte JSON retorna serveixen de filtres); per tant, la comanda d'exemple anterior es podria escriure amb filtres d'aquesta manera: *podman container inspect -f {{.NetworkSettings.IPAddress}} lala*

> NOTA: Una comanda semblant però que mostra les metadades d'imatges en comptes de contenidors és ***podman image inspect urlImatge***

podman system df mostra l'espai ocupat per les imatges i els contenidors existents al sistema. Pot combinar-se amb l'ús de la comanda ***podman system prune***, la qual elimina qualsevol recurs (és a dir, imatge, volum, xarxa...) que no estigui associat a cap contenidor actualment existent. Per esborrar també els contenidors actualment aturats, cal afegir el paràmetre ***-a***

podman volume create nomVolum crea a l'amfitrió la carpeta "**~/.local/share/containers/storage/volumes/nomVolum**" (si s'executa sense ser "root") o la carpeta "**/var/lib/containers/storage/volumes/nomVolum**" (si s'executa com a "root"), que serà on es guardaran tots els fitxers que es vulguin compartir amb qualsevol contenidor quan aquesta es munti indicant el paràmetre ***-v nomVolum:/ruta/carpeta/en/contenidor*** (és a dir, sense precedir de cap "/" la part esquerra de ":") de *podman run*

NOTA: En realitat, la comanda *podman volume create* no caldria executar-la explícitament perquè si el volum no existís, en fer *podman run -v nomVolum:...* es crearà automàticament
NOTA: Un cop creats volums, es poden llistar amb ***podman volume ls*** (opcionalment indicant algun filtre mitjançant el paràmetre ***-f ...*** i/o les columnes a mostrar amb ***--format=unFormat***; veieu *man podman-volume-ls* per més detalls), es pot inspeccionar les característiques d'un volum amb ***podman volume inspect nomVolum*** (o les de tots amb ***podman volume inspect -a***), es pot esborrar un volum concret, si no l'està fent servir cap contenidor actualment en marxa, amb ***podman volume rm nomVolum*** (noteu que els volums romanen tot i que el contenidor s'elimini...la idea és que siguin reutilitzables; és per això que cal esborrar-los "a mà") o es poden esborrar tots els volums que no s'estiguin fent servir per cap contenidor actualment en marxa amb ***podman volume prune***

Per saber més comandes de *podman* (n'hi ha moltes més!: ***podman [container] attach, podman [container] commit, podman [container] cp, podman [container] diff, podman [container] pause***, etc), consulteu http://docs.podman.io/en/latest/Commands.html

Pods

Un "pod" és un conjunt de contenidors que comparteixen un espai comú de xarxa. O dit d'una altra manera: els contenidors que pertanyen a un "pod" es veuen entre sí perquè tots pertanyen a la mateixa xarxa (xarxa que en realitat, és "localhost" -o, en altres paraules: tots els contenidors comparteixen la IP 127.0.0.1 dins del "pod", fent-se així indistingibles un de l'altre en aquest nivell-). D'altra banda, els contenidors que pertanyen a un "pod" només són visibles des de l'exterior (sempre com una sola entitat a través de la IP de la màquina amfitriona) a través del "mapeig" de ports adient (si és que n'hi ha). En aquest sentit, un "pod" podrà tenir tots els ports oberts que sigui necessaris: aquests seran automàticament "tunnelejats" al port del contenidor concret de dins del "pod" que el tingui a l'escolta.

Els "pods" són molt útils quan es volen executar varis contenidors que formen part d'una sola "unitat orgànica", com ara un servidor LAMP (format per un contenidor Apache + un contenidor MySQL + un contenidor PHP-FPM, per exemple).

Per a què un contenidor pertanyi a un "pod", primer caldrà crear aquest amb ***podman pod create -n nomPod -p x:y [-p w:z] ...*** Noteu com, tot i estar en principi buit, a l'hora de crear el "pod" ja haurem d'establir prèviament els "mapejos" de ports que volem, ja que aquest aspecte de la configuració ja no es realitza en el moment d'arrencar el contenidor individual sinó que es delega en el procés de creació del "pod". Cal tenir en compte que com que el "pod" es percep com una unitat en sí amb una única adreça IP "vista des de fora", els diferents serveis que puguin haver funcionant en els diferents contenidors que hi pertanyen hauran d'escoltar en diferents ports cadascun per no "trepitjar-se" entre sí.

NOTA: Altres paràmetres interessants de la comanda anterior són, per exemple, ***--ip=x.x.x.x*** (on "x.x.x.x" representa la IP estàtica que s'assignarà al "pod", la qual llavors serà diferent de la de la màquina amfitriona -aquest paràmetre només funciona en "pods" no "root-less", però-), ***--dns=x.x.x.x*** (on "x.x.x.x" representa la IP d'un servidor DNS que faran servir els contenidors del "pod") o ***--add-host=nom:x.x.x.x*** (on "nom" representa el nom d'una màquina i "x.x.x.x" la seva adreça IP, per tal d'associar-ne els dos valors com a informació pels contenidors del "pod").

Un cop creat el "pod", podrem afegir-hi els contenidors que desitgem simplement afegint el paràmetre ***--pod=nomPod*** a la comanda *podman run* en el moment que els volguem executar (sense indicar ara cap paràmetre *-p*). A partir de llavors, tots els contenidors inclosos dins del "pod" es veuran entre sí i, a més, aquell contenidor que implementi una aplicació escoltant en algun port indicat amb el paràmetre *-p* en la creació del "pod" podrà ser també accessible des de l'exterior del "pod".

NOTA: En realitat, no seria necessari haver creat el "pod" abans: si s'indica el paràmetre *--pod* així: ***--pod=new:nomPod***, el "pod" es crearà (si no existeix encara) en el moment d'executar el primer contenidor que hi volguem afegir.

Es poden veure els "pods" existents en un moment donat al sistema amfitrió amb la comanda ***podman pod ls*** (o ***podman pod list***); aquesta comanda mostra les columnes "POD ID", "NAME", "STATUS", "CREATED", "# OF CONTAINERS" i "INFRA ID". La darrera columna indica l'identificador del contenidor "infra"...expliquem això: per defecte, tots els "pods" es creen sempre contenint com a mínim un contenidor predeterminat anomenat "infra"; el qual serveix per allotjar els espais de noms associats amb el "pod" que li permeten a *podman* connectar internament els altres contenidors entre sí per tal de què puguin interrelacionar-se dins del "pod" en qüestió.

> NOTA: Si afegim el paràmetre ***-p*** a la comanda *podman ps -a*, apareixeran dues columnes extres indicant l'identificador i el nom del "pod" on pertany cada contenidor mostrat

Altres comandes interessants són ***podman pod start nomPod*** (per iniciar el "pod" -és a dir, tots els contenidors que hi pertanyen-), ***podman pod stop nomPod*** (per aturar el "pod" -és a dir, tots els contenidors que hi pertanyen-...també es pot indicar ***-a*** en lloc de nomPod per aturar tots els "pods" encesos), ***podman pod rm nomPod*** (per eliminar el "pod" indicat sempre que estigui apagat -incloent tots els contenidors que hi pertanyin-), ***podman pod prune*** (per eliminar tots els "pods" que estiguin apagats -incloent tots els contenidors que hi pertanyin-) o ***podman pod inspect nomPod*** (per inspeccionar totes les característiques del "pod" indicat), entre altres.

> NOTA: Podman permet la creació i l'execució de "pods" definits per escrit en un fitxer YAML de Kubernetes (vegeu *man podman-play-kube*) i viceversa: Podman també pot generar un fitxer YAML de Kubernetes basat en un determinat contenidor o "pod" (vegeu *man podman-generate-kube*). Això fa que Podman sigui una eina molt còmoda per realitzar una fàcil transició d'un entorn de desenvolupament local a un clúster de producció de Kubernetes.

EXERCICIS:

0.-a) Digues si són certes o falses les següents afirmacions: "Un contenidor OCI...

...conté una aplicació compartible, extensible i, a priori, aïllada del sistema amfitrió"
...permet garantir el mateix entorn per aquesta aplicació en qualsevol sistema (proves, producció, etc)"
...pot formar part d'un clúster i, per tant, ser gestionat de forma escalable i dinàmica amb un orquestrador"

b) Arrenca una màquina virtual (Server o Workstation, és igual) amb la seva tarja en mode "adaptador pont". Instal.la-hi el paquet "podman". Tots els exercicis següents els hauràs de fer dins d'ella.

> NOTA: En el cas de què facis servir una màquina Ubuntu 20.04, hauràs de fer les següents passes:
> *echo "deb https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/xUbuntu_20.04/ /" | sudo tee /etc/apt/sources.list.d/podman.list*
> *curl -L https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/xUbuntu_20.04/Release.key | sudo apt-key add -*
> *sudo apt update && sudo apt -y install podman*

1.-a) Executa *podman search ubuntu -f is-official=true* ¿Què veus? Executa tot seguit *podman image pull docker.io/library/ubuntu* ¿Què passa? Finalment executa *podman image ls* ¿Què veus?

b) ¿Què indica l'apartat "Cmd:" que es mostra si executes *podman image inspect ubuntu* ?

> NOTA: Podries mostrar només aquesta dada si executes *podman image inspect --format {{.Config.Cmd}} ubuntu*

c) Executa *podman container run -it --name pepito docker.io/library/ubuntu* ¿Què passa? Si obres un altre terminal (a la màquina amfitriona) i allà executes *podman container ls*, ¿què veus?

d) Dins del contenidor, executa *apt update && apt install iproute2 curl* i tot seguit executa-hi *ip -c a ;*¿quina és la IP del contenidor? Executa *ip -c r*; ¿quina és la seva porta d'enllaç? Executa *cat /etc/resolv.conf*; ¿quins són els servidors DNS que usa el contenidor per defecte? Executa *curl -I https://www.hola.com* ; ¿què veus?

e) Surt del contenidor amb *exit.* ¿Què veus ara si executes *podman container ls*? ¿I *podman container ls -a* ?

f) ¿Què passa si ara tornes a executar *podman container run -it --name pepito docker.io/library/ubuntu*? Per què? Executa ara *podman container start pepito* ¿Què veus ara si executes *podman container ls* ?

g) Executa la comanda *podman container exec pepito cat /etc/os-release*. ¿Què veus?

> NOTA: No hem hagut d'indicar la ruta absoluta de la comanda cat perquè dins de la imatge ja hi ha una variable PATH definida, tal com es pot comprovar amb *podman image inspect --format {{.Config.Env}} ubuntu*

h) Entra al contenidor iniciat a l'apartat anterior executant la comanda *podman container attach pepito*

> NOTA: Pots iniciar el contenidor i "entrar en ell" tot en un sol pas si afegeixes el paràmetre **-a** (d'"attach") a la comanda *podman start pepito*

i) Surt del contenidor anterior i executa *podman rm pepito* Què veus ara si executes *podman container ls -a* ?

j) Executa *podman container run -it **--rm** --name pepito2 docker.io/library/ubuntu cat /etc/os-release* i comprova que la sortida sigui la correcta. ¿Per què ara *podman container ls -a* tampoc mostra res?

k) Crea una carpeta anomenda "pipi" dins de la teva carpeta personal a la màquina amfitriona i executa *podman container run -it --rm **-v $HOME/pipi:/opt** --name pepito3 docker.io/library/ubuntu* . Un cop dins del contenidor, executa *echo "Hola" > /opt/a.txt* i observa quin és el seu propietari (amb *ls -l /opt/a.txt*). En un terminal de la màquina amfitriona observa quin és ara el contingut i el propietari de l'arxiu "~/pipi/a.txt". ¿Què veus? ¿Per què?

kII) Fer ara al revés: des d'un terminal de la màquina amfitriona executa la comanda *echo "Adeu" > ~/pipi/b.txt* i ara observa dins del contenidor quin és el contingut i el propietari de l'arxiu "/opt/b.txt". ¿Què veus? ¿Per què? Finalment, surt del contenidor

l) Executa *podman rmi docker.io/library/ubuntu* i tot seguit *podman image ls* ¿Què veus?

2.-a) Executa *podman image pull docker.io/library/httpd*

NOTA: La imatge anterior es pot haver trobat fent abans *podman search apache -f is-official=true*, per exemple

b) Executa *podman image inspect --format {{.Config.ExposedPorts}} httpd* ¿Què veus? ¿I sota l'apartat "History"?

c) Executa *podman container run -d -p 8888:80 --rm --name manolo docker.io/library/httpd* i tot seguit, a la màquina amfitriona, obre un navegador i escriu a la barra de direccions l'adreça http://127.0.0.1:8888 ¿Què veus?

d) Executa *podman stop manolo* i tot seguit torna a intentar anar a la mateixa adreça de l'apartat anterior amb el navegador de la màquina amfitriona. ¿Què veus ara?

e) Executa *podman start manolo*. ¿Per què no pots? Torna a executar la mateixa comanda que a l'apartat c) però ara sense el paràmetre *--rm* ¿Què veus si executes tot seguit *podman top manolo*?

f) Executa *podman attach manolo* i tot seguit, a la màquina amfitriona, obre un navegador i escriu a la barra de direccions l'adreça http://127.0.0.1:8888 ¿Què veus a la sortida de la comanda "podman attach" anterior? ¿Per què no obtens un terminal com passava a l'exercici anterior (pots consultar la nota següent com a pista)? Pulsa CTRL+C en la pantalla de "podman attach" per aturar el contenidor i torna'l a iniciar executant de nou *podman start manolo*

NOTA: La comanda "podman attach" connecta amb el contenidor indicat però no és igual com fer una sessió SSH (per entendreńs) perquè en realitat "podman attach" el que fa és "endollar-se" a l'entrada i sortida estàndard (stdin/stdout) de la comanda indicada com a CMD/ENTRYPOINT a la definició del contenidor: si aquesta és un terminal, llavors obtindrem una sessió interactiva, sí, però si aquesta és un servidor web (o un altre tipus de programa), obtindrem la seva sortida (en aquest cas, els "logs" de les peticions HTTP).

g) Per obtenir un terminal en un contenidor executat com a segon pla que té un altre procés funcionant com a CMD/ENTRYPOINT (com és el cas del nostre contenidor "manolo"), pots provar d'executar la comanda *podman exec -it manolo bash* (això sí, tot suposant que la imatge del contenidor incorpori el binari "bash"!)

h) Un cop dins del contenidor, observa el contingut de la carpeta "/usr/local/apache2/htdocs". Modifica el contingut del fitxer "index.html" que hi ha allà (de la manera que tu vulguis) i comprova que el canvi s'hagi realitzat tornant a anar a http://127.0.0.1:8888 des del navegador de la màquina amfitriona.

i) Surt del terminal obert a l'apartat anterior amb la comanda *exit* i comprova que, a diferència del que passava amb els contenidors que tenen un terminal com a CMD/ENTRYPOINT, el contenidor "manolo" continua funcionant (ho pots veure fàcilment executant *podman container ls*).

j) Executa *podman stop manolo* i *podman rm manolo* . Tot seguit crea, allà on estiguis en la màquina amfitriona, una carpeta anomenada "docroot" dins de la qual crea un fitxer anomenat "index.html" amb el següent contingut: *<!DOCTYPE html><html><body>Hola guapo</body></html>*

jII) Executa *podman container run -d -p 8888:80 --rm --name manolo2 -v ./docroot:/usr/local/apache2/htdocs docker.io/library/httpd* i tot seguit escriu a la barra de direccions del navegador de la màquina amfitriona (després potser de netejar la catxé amb CTRL+F5) l'adreça http://127.0.0.1:8888 ¿Què veus? ¿Per què?

NOTA: És molt important escriure el "./" inicial a la ruta de la carpeta "docroot" per a què podman entengui que estem parlant d'una ruta d'una carpeta i no d'un nom de volum genèric

k) Canvia ara la frase indicada dins del fitxer "docroot/index.html" i guarda aquest canvi. Torna a visitar http://127.0.0.1:8888 i comprova com apareix ara la nova frase sense haver de fet cap reinici ni aturada ni accés via exec, etc al contenidor.

l) Mantenint el contenidor "manolo2" en marxa, ara executa un altre a partir de la mateixa imatge, així: *podman container run -d -p **8889**:80 --rm --name manolo3 docker.io/library/httpd* Comprova que ara tens dos contenidors en marxa amb *podman container ls* ¿Què veus si ara escrius l'adreça http://127.0.0.1:8889 al navegador de la màquina amfitriona?

m) Executa finalment *podman stop manolo2* per aturar (i eliminar) el contenidor i *podman rmi docker.io/library/httpd* per eliminar la imatge

n) Vés a https://hub.docker.com/_/httpd i digues per a què serveix la línia COPY del Containerfile d'exemple que apareix sota l'apartat "Create a Containerfile in your project" i, en general, aquest Containerfile. ¿Per a què serveix, en canvi, el paràmetre -v indicat a la comanda *podman* sota l'apartat "Without a Containerfile"? D'altra banda, ¿què explica l'apartat "Configuration"?

o) La imatge anterior només conté un servidor Apache2 i prou. Si volem utilitzar una imatge amb un servidor Apache2 + l'intèrpret PHP integrat com a mòdul (per tal d'oferir pàgines PHP a més de HTML), caldria usar una imatge de les documentades a https://hub.docker.com/_/php Concretament, vés a l'apartat "Image variants" i digues quina de les imatges disponibles és la que incorpora PHP + Apache2 i on es troba per defecte el seu "DocumentRoot"

Imagina que utilitzes molt una imatge determinada (per exemple la "docker.io/library/ubuntu") però voldries afegir-li algun paquet extra per tal de què estigués disponible per defecte sense haver d'instal·lar-lo manualment a cada contenidor arrencat amb aquesta imatge. Això ho pots aconseguir modificant un contenidor qualsevol (creat a partir d'una imatge base) i, un cop estigui aquest contenidor en l'estat desitjat, fent un "commit" per a generar, a partir d'ell, una nova imatge amb tots aquests canvis incorporats. Al següent exercici es detallen els passos necessaris a fer.

3.-a) Torna a executar *podman image pull docker.io/library/ubuntu* i tot seguit arrenca un contenidor basat en aquesta imatge executant *podman container run -it --rm --name perico docker.io/library/ubuntu*

b) Instal.la-hi, dins del contenidor, el paquet *nano* (recorda de fer primer *apt update*)

c) Obre un terminal a la màquina amfitriona i executa-hi *echo "Hola" > a.txt*. Tot seguit, i mentre el contenidor està en marxa, executa en aquest terminal la comanda *podman container cp a.txt perico:/root*

NOTA: La comanda *podman cp* també permet copiar de "dins" cap a "fora" simplement invertint l'ordre dels paràmetres, així, per exemple: *podman container cp perico:/root/a.txt $HOME*

d) En aquest mateix terminal de la màquina "amfitriona", i mentre el contenidor està en marxa, executa la comanda *podman commit perico ubuntu2* Tot seguit, surt del contenidor (amb *exit*, com sempre)

e) ¿Què veus ara si executes *podman image ls* ? ¿Quina carpeta ha aparegut nova dins de ".local/share/containers/storage"?

f) ¿Què mostra la comanda *podman image diff localhost/ubuntu2*? (pots consultar *man podman-image-diff*)

g) Executa la comanda *podman container run -it --rm --name perico2 localhost/ubuntu2* ¿Què passa? ¿I què passa si, allà dins, executes la comanda *nano /root/a.txt*? Surt del contenidor

h) Executa la comanda *podman image save localhost/ubuntu2 -o imatgeUbuntu2.tar* Tot seguit, executa *podman image rm localhost/ubuntu2* i, després de comprovar que la imatge "ubuntu2" ja no existeix (amb *podman image ls),* executa la comanda *podman image load -i imatgeUbuntu2.tar* ¿Què veus en tornar a fer *podman image ls*?

> NOTA: Comandes similars a *podman image save* i *podman image load* serien *podman container export* i *podman container import*, respectivament. Però tal com diu el seu nom, aquestes darreres treballen a nivell de contenidor i no pas d'imatge

La manera anterior permet personalitzar una imatge base però no canvia per exemple el seu "punt d'entrada" (és a dir, seguirà executant un shell Bash per defecte). Una manera més versàtil de crear imatges personalitzades a partir d'imatges base, on es pot personalitzar qualsevol característica del seu comportament, és utilitzant un "Containerfile", com descriu el següent apartat

4.-a) Crea una carpeta anomenada "lalala" i allà dins crea un fitxer anomenat "Containerfile" amb el següent contingut...:

```
FROM ubuntu:latest
#La variable indicada davant de la comanda apt és necessària per a què la instal·lació no sigui interactiva
RUN apt update && DEBIAN_FRONTEND=noninteractive apt -y install apache2
COPY index.html /var/www/html
CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
EXPOSE 80
```

aII) Crea també, a la mateixa carpeta "lalala", un fitxer anomenat "index.html" amb el següent contingut:

```
<!DOCTYPE html><html><body>Pacooo!!</body></html>
```

b) Executa la comanda *podman image build ./lalala -t miservapache:yeah* ¿Què passa?

c) Executa *podman container run -d -p 8888:80 --rm --name miguelito localhost/miservapache:yeah* i tot seguit, a la màquina amfitriona, obre un navegador i escriu a la barra de direccions l'adreça http://127.0.0.1:8888 ¿Què veus?

d) ¿Quines característiques tindria la imatge obtinguda a partir del següent fitxer "Containerfile": https://hub.docker.com/r/bartekmis/ubuntu-apache/dockerfile?

4BIS.-a) Crea ara una carpeta anomenada "lelele" i allà dins crea ara un fitxer anomenat "Containerfile" amb el següent contingut...:

```
FROM docker.io/library/python:latest
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
COPY holamon.py ./
CMD [ "python", "./holamon.py" ]
EXPOSE 5000
```

aII) Crea també, a la mateixa carpeta "lelele", un fitxer anomenat "holamon.py" amb el següent contingut...:

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_world():
    return 'Hello World'
@app.route('/hello/<name>')
def hello_name(name):
```

```
    return 'Hello %s!' % name
if __name__ == '__main__':
    app.run(host="0.0.0.0",port=5000)
```

NOTA: És necessari indicar que el servidor Flask escoltarà a la IP 0.0.0.0 perquè per defecte ho fa a la 127.0.0.1 i això vol dir que només estaria accessible des de dins del propi contenidor. El port per defecte ja era el 5000 però s'ha explicitat igualment

aIII) ...i un altre anomenat "requirements.txt" contenint simplement una línia que sigui la paraula *Flask*

b) Executa la comanda *podman build ./lelele -t miservpython* ¿Què passa?

c) Executa *podman container run -d -p 5000:5000 --rm --name guillermito localhost/miservpython* i tot seguit, a la màquina amfitriona, obre un navegador i escriu a la barra de direccions l'adreça http://127.0.0.1:5000/hello/federica ¿Què veus?

d) Executa finalment *podman stop guillermito* per aturar (i eliminar) el contenidor i *podman rmi localhost/miservpython docker.io/library/python* per eliminar les imatges

En el següent exercici s'implementarà un "pod" contenint dos contenidors: un executant un servidor de base de dades MariaDB, el qual servirà de suport per l'altre contenidor, que estarà executant un servidor Apache+PHP i que oferirà un lloc Wordpress.

5.-a) Crea les següents carpetes a la màquina amfitriona: "data-mariadb" i "www-html". La primera és on es guardaran les bases de dades del servidor MariaDB (concretament, la base de dades usada per Wordpress) i la segona és on es guardaran les pàgines web que formen el Wordpress. Precisament, descarrega't dins d'aquesta darrera carpeta la darrera versió de Wordpress en format tar.gz (i descomprimeix-lo) executant allà les següents comandes:

curl -o wordpress.tar.gz https://wordpress.org/wordpress-latest.tar.gz
tar -xzf wordpress.tar.gz && rm wordpress.tar.gz

b) Descarrega't les següents imatges, ja preparades, que corresponen respectivament a un servidor de base de dades MariaBD i a un servidor Apache2+PHP (funcionant aquest com a mòdul)

podman image pull docker.io/library/mariadb
podman image pull docker.io/library/php:7.4-apache

NOTA: La documentació oficial de com iniciar i configurar els contenidors generats a partir de les imatges anteriors es troba respectivament, a https://hub.docker.com/_/mariadb i https://hub.docker.com/_/php
NOTA: També existeix una imatge oficial anomenada "wordpress" (https://hub.docker.com/_/wordpress), però en realitat no és gaire més (tal com es pot veure a la seva definició escrita dins el seu Containerfile corresponent, disponible a https://github.com/docker-library/wordpress/tree/master/latest/php7.4/apache) que una adaptació de la imatge "php-apache" incloent les llibreries MySQL necessàries (que nosaltres instal·larem a mà) i una instal·lació de Wordpress inclosa a dins.

c) Crea un "pod" nou amb el nom "felipito" que exposi de forma pública únicament el port 80 (mapejat al port 8000 de la màquina amfitriona; no pot ser en ports menors de 1024, recordem, si no som "root") executant: *podman pod create -n felipito -p 8000:80* Comprova que estigui creat executant *podman pod list*

NOTA: El port d'escolta del servidor MariaDB (el 3306) no el mapejarem perquè no volem que sigui públic: només volem que sigui accessible pel contenidor Apache+PHP que existeixi dins del mateix "pod" que ell

d) Executa dins del "pod" els contenidors corresponents a les imatges descarregades anteriorment, així:

podman container run -d --pod=felipito -v $PWD/data-mariadb:/var/lib/mysql
-e MYSQL_ROOT_PASSWORD="1234" -e MYSQL_DATABASE="wp" -e MYSQL_USER="wordpress"
-e MYSQL_PASSWORD="w0rdpr3ss" --name mimariadb docker.io/library/mariadb

NOTA: En el moment d'iniciar un contenidor a partir de la imatge oficial de MariaDB, es pot ajustar la instància del servidor en qüestió passant una o més variables d'entorn a través de la comanda *podman run*, com ara MYSQL_ROOT_PASSWORD (contrasenya que s'establirà per l'usuari "root" del servidor MariaDB instanciat; obligatori), MYSQL_DATABASE (base de dades que es crearà automàticament; és la que indicarem a la instal·lació del Wordpress per a què la faci servir; opcional), MYSQL_USER (usuari que es crearà automàticament amb permisos d'administració sobre la base de dades anterior; també li haurem de dir al Wordpress per a què el faci servir per tal de gestionar la base de dades anterior; opcional) i MYSQL_PASSWORD (contrasenya d'aquest usuari)

podman container run -d --pod=felipito -v $PWD/www-html/wordpress:/var/www/html --name miapaphp docker.io/library/php:7.4-apache

NOTA: La imatge "php:7.4-apache" per defecte té configurat com a "DocumentRoot" la carpeta "/var/www/html", tal com es pot veure a la comanda anterior

e) Executa la comanda *podman container ls -p* . ¿Què veus? ¿Què és el contenidor "k8s.gcr.io/pause" que allà hi apareix?

f) Arrenca el pod executant *podman pod start felipito* . Tot seguit executa la comanda *podman pod inspect felipito* i digues quins valors mostren les claus "CreateCommand", "State", "Hostname", "InfraConfig.PortBindings", "InfraConfig.HostPort", "NumContainers" i "Containers.Name". Executa també la comanda *podman container inspect miapaphp* i digues quins valors mostren les claus "Mounts.Source" i "Mounts.Destination", "Ports", "Config.Env", "Config.WorkingDir" i "Config.CreateCommand"

g) Entra en el contenidor de l'Apache+PHP executant la comanda *podman exec -it miapaphp bash* i, un cop a dins, executa la comanda *ps -ef* ¿Què veus? I si executes *cat /etc/hosts* , ¿què veus? Finalment, executa ara les següents comandes per tal d'instal·lar les llibreries PHP necessàries per connectar amb bases de dades MySQL/MariaDB (ja que no venen per defecte a la imatge que hem triat)

docker-php-ext-install mysqli pdo pdo_mysql && docker-php-ext-enable mysqli pdo pdo_mysql

NOTA: Les comandes anteriors no són "estàndard" sinó que estan especialment dissenyades per aquesta imatge que hem triat, ja que aquesta està construïda de tal manera que per incloure les llibreries PHP-MySQL/MariaDB cal fer-ho així.

h) Una altra cosa que has de fer és canviar el propietari de tot allò inclòs dins de la carpeta "/var/www/html" del contenidor miapaphp per a què sigui l'usuari i grup "www-data". Això ho pots fer simplement executant, dins d'aquesta carpeta la comanda *chown -R www-data:www-data* .

NOTA: Això és degut a què l'Apache haurà de poder modificar contingut d'aquella carpeta (com per exemple l'arxiu "wp-config.php", entre altres. Com pots intuir degut a l'existència de l'usuari "www-data", la imatge de l'Apache+PHP utilitzada es basa en un sistema Debian

i) Surt del terminal interactiu del contenidor "miapaphp" mitjançant *exit* , atura el pod (executant *podman pod stop felipito*) i torna'l a iniciar (executant *podman pod start felipito*) per tal de què l'Apache+PHP reconegui les llibreries instal·lades al pas g).

j) Ara entra en el contenidor de MariaDB executant la comanda *podman exec -it mimariadb bash* i, un cop a dins, executa la comanda *ps -ef* ¿Què veus ara? ¿Què veus si executes ara *cat /etc/hosts*? ¿I si executes la comanda *ss -tln*? Finalment, executa la comanda *mysql -u root -p1234* Un cop dins de la consola interna del MariaDB, execut la subcomanda *show databases;* ¿Què veus? ¿Què veus si executes *use wp;* i tot seguit *show tables;* i per què?

NOTA: Si volguessis instal·lar algun programa extra dins d'aquest contenidor, pots executar la comanda apt update (podràs comprovar així que la imatge sobre la que es basa aquest contenidor ha sigut construïda sobre un sistema Debian!) i, a partir d'aquí, executar *apt install nano*, *apt install ncat*, etc

k) Obre el navegador de la màquina amfitriona i vés a l'adreça 127.0.0.1:8000. ¿Què veus? Omple el formulari de la primera pantalla de l'assistent d'instal·lació del Wordpress amb els valors demanats adientment (nom de base de dades -"wp"- ; usuari -"wordpress"- ; contrasenya -"w0rdpr3ss"- ; servidor -"127.0.0.1", és molt important que no sigui "localhost"!, llegiu la nota inferior-) i, a partir d'aquí, acaba la instal·lació del Wordpress i comprova que funciona, tant el panell de control (http://127.0.0.1:8000/wp-admin) com el propi blog (http://127.0.0.1:8000)

NOTA: Si s'indica "localhost", el Wordpress (a l'igual que qualsevol altre client MySQL/MariaDB) per defecte emprarà un connector de tipus "socket" per contactar amb el servidor MariaDB. No obstant, la imatge que estem fent servir no està configurada per utilitzar aquest mecanisme de comunicació sinó un que empri TCP/IP. Afortunadament, si s'indica "127.0.0.1", el Wordpress (a l'igual que qualsevol altre client MySQL/MariaDB), per defecte emprarà justament el mecanisme TCP/IP per contactar amb el servidor. Hi ha més a https://mariadb.com/kb/en/troubleshooting-connection-issues i a https://dev.mysql.com/doc/refman/8.0/en/problems-connecting.html
NOTA: El resultat d'haver executat l'assistent anterior és equivalent a editar manualment l'arxiu "wp-config-sample.php" i indicar els valors demanats al formulari dins de les línies define () adients ("DB_NAME", "DB_USER", "DB_PASSWORD" i "DB_HOST"), gravant aquest arxiu amb el nou nom de "wp-config.php" (només si té aquest nom serà tingut en compte pel Wordpress)

6.-a) Executa la comanda *podman generate kube -f mipod.yml felipito* (tant se val si el pod està apagat o no). ¿Quin és el contingut del fitxer "mipod.yml" generat?

b) Apaga "pod" (executant *podman pod stop felipito)* i elimina'l sencer (executant *podman pod rm felipito)*

c) Executa la comanda *podman play kube mipod.yml* i tot seguit, executa *podman pod list.* ¿Què veus? Torna a anar amb el navegador de la màquina amfitriona a l'adreça 127.0.0.1:8000. ¿Què veus? ¿Per què?

NOTA: Recorda que les instal·lacions de paquets dins d'un contenidor (i, en general, totes les comandes que no modifiquin contingut de volums muntats en disc) es realitzen sobre RAM, no sobre la imatge

d) Solucionarem el problema vist a l'apartat anterior generant una imatge nova a partir de l'oficial "php:7.4-apache" que inclogui les llibreries PHP-MySQL/MariaDB ja d'entrada i llavors incloent al pod un contenidor generat a partir d'aquesta nova imatge en lloc del que hem fet servir fins ara. Els passos concrets, per tant, són els següents:

*Torna a aturar i eliminar el pod felipito (*podman pod stop felipito && podman pod rm felipito*)

*Crea una carpeta anomenada "lululu" i dins crea-hi un fitxer "Containerfile" amb el següent contingut (només indiquem els canvis respecte la imatge base, així que no cal especificar ni la línia CMD, ni la línia EXPOSE, etc), i tot seguit executa la comanda *podman image build ./lululu -t php-mysql-apache*

```
FROM docker.io/library/php:7.4-apache
RUN docker-php-ext-install mysqli pdo pdo_mysql && docker-php-ext-enable mysqli pdo pdo_mysql
```

*Crea de nou el pod felipito (*podman pod create -n felipito -p 8000:80*)

*Executa les següents comandes per executar de nou els dos contenidors corresponents a les imatges adients (la mateixa imatge "mariadb" oficial feta servir a l'exercici anterior i la imatge personalitzada Apache+PHP+llibreriesMySQL creada en els passos anteriors):

```
podman container run -d  --pod felipito -v $PWD/data-mariadb:/var/lib/mysql
        -e MYSQL_ROOT_PASSWORD="1234"  -e MYSQL_DATABASE="wp"
        -e MYSQL_USER="wordpress" -e MYSQL_PASSWORD="w0rdpr3ss"
        --name mimariadb docker.io/library/mariadb
podman container run -d --pod felipito -v $PWD/www-html/wordpress:/var/www/html
        --name miapapdo localhost/php-mysql-apache
```

*Confirma que les pàgines de Wordpress tornen a ser visibles anant amb el navegador de la màquina amfitriona a http://127.0.0.1:8000

e) Executa de nou la comanda *podman generate kube -f mipod2.yml felipito* . Torna a apagar el pod (executant *podman pod stop felipito)* i elimina'l sencer (executant *podman pod rm felipito)*

f) Executa la comanda *podman play kube mipod2.yml* i torna a anar, amb el navegador de la màquina amfitriona, a l'adreça http://127.0.0.1:8000. ¿Què veus ara? Un cop comprovat que les pàgines Wordpress es veuen bé torna a aturar el pod creat i elimina'l de nou.