

Conversación con Gemini

Exported: 15/6/2026 9:13:27

Jordi H.

Has dicho tengo algunos problemas con la asignación de puertos en mis contenedores corriendo podman. usé la interfaz de cockpit y en algún caso no especificué el puerto del servidor y se me han montado los puertos en 0.0.0.0, es decir, en todas las interfaces. ahora tengo algunas coincidencias que quiero evitar y cambiar esas por una sola ip. El problema es que no sé dónde guarda cockpit esas configuraciones para arrancarlos y pararlos. me puedes ayudar

Gemini

Es un problema común cuando se usan herramientas de interfaz gráfica: la abstracción es genial hasta que algo sale mal y no sabes dónde está el "interruptor" real. Cuando usas **Cockpit** para gestionar contenedores (que por debajo utiliza el plugin cockpit-podman), este no guarda la configuración en archivos de texto editables al estilo Apache o Nginx. En su lugar, Cockpit simplemente interactúa directamente con la **API de Podman**. Aquí te explico cómo corregir esa asignación de 0.0.0.0 y dónde reside realmente la configuración

1. ¿Dónde se guarda la configuración?

Podman es "daemonless" (sin demonio), por lo que la configuración de cada contenedor se guarda en una base de datos local de SQLite o archivos JSON (dependiendo de tu versión) dentro de

- Si eres root: `/var/lib/containers/storage/overlay-containers/`
- Si eres usuario (rootless): `~/.local/share/containers/storage/overlay-containers/`

Importante: No te recomiendo editar esos archivos a mano. Es extremadamente fácil corromper la base de datos de Podman. La forma correcta de "cambiar" la configuración es mediante la recreación o actualización del contenedor

2. Cómo cambiar la IP (de 0.0.0.0 a una IP específica)

Lamentablemente, Podman (y Docker) no permiten modificar el mapeo de puertos de un contenedor que ya existe. Tienes que **borrarlo y volverlo a crear** con los parámetros correctos. Si quieres hacerlo por línea de comandos (que es más preciso que Cockpit), el formato para restringir la IP es `IP_HOST:PUERTO_HOST:PUERTO_CONTENEDOR`

Paso a paso:

1. Identifica los parámetros actuales: Si no recuerdas cómo creaste el contenedor, usa:
`podman inspect nombre_del_contenedor`
2. Detén y elimina el contenedor:
`podman stop nombre_del_contenedor`
`podman rm nombre_del_contenedor`
3. Crea el nuevo contenedor especificando la IP: `Bash`

CODE

```
podman run -d \
--name nombre_del_contenedor \
-p 192.168.1.50:8080:80 \
nombre_de_la_imagen
```

En este ejemplo, el servicio solo escuchará en la IP 192.168.1.50 y no en todas las interfaces.

3. El truco de Cockpit (si prefieres la interfaz)

Si quieres seguir usando Cockpit, al crear un contenedor nuevo (o tras borrar el anterior):

1. Ve a la sección de Networking / Puertos.
2. En el campo donde introduces el puerto del host, Cockpit suele permitir el formato completo. En lugar de poner solo 8080, prueba a poner 192.168.1.50:8080
3. Si la interfaz de tu versión de Cockpit es muy rígida y solo tiene un campo numérico, entonces estás obligado a usar la terminal para tener ese control fino

Un consejo para el futuro: Quadlets

Si te vas a centrar en Python y análisis de datos, probablemente automatices mucho tus despliegues. En sistemas modernos (como RHEL, Fedora o Debian con Podman reciente), la forma "profesional" de guardar configuraciones es usar **Quadlets**. Son archivos .container que se guardan en /etc/containers/systemd/ y que permiten gestionar contenedores como si fueran servicios de Systemd. Ahí sí tienes un archivo de texto claro donde puedes poner PublishPort=127.0.0.1:8080:80. ¿Te gustaría que te ayude a generar el comando podman run exacto para uno de tus contenedores o prefieres ver cómo pasar esa configuración a un archivo de Sy

Jordi H.

Has dicho crees que puedo usar un cliente de sqlite y toquetear la base de datos de contenedores?

Gemini

Poder, puedes... pero es como intentar operar a corazón abierto con un tenedor. **No te lo recomiendo** en xplico por qué tu instinto de "tocar la base de datos" es peligroso en este caso concreto.

1. El riesgo de corrupción

Podman utiliza SQLite para el seguimiento de los contenedores (especialmente en versiones recientes), pero esa base de datos no es solo un registro pasivo. Está estrechamente vinculada al **estado del sistema d** iver) y a los procesos de **common**.

- Inconsistencia: Si cambias una IP en la base de datos pero no notificas al subsistema de red (CNI o Netavark), el contenedor creará que está en una IP mientras que las reglas de iptables o nftables del host seguirán apuntando a la antigua. Resultado: un contenedor "zombie" que no responde
- Bloqueos: Si intentas escribir en la base de datos mientras el servicio de Podman (o Cockpit) está consultándola, puedes corromper el índice y perder el acceso a todos tus contenedores

2. Dónde está el "corazón" de Podman

Si tienes curiosidad técnica, la base de datos suele estar en:

- Root: /var/lib/containers/storage/libpod/bolt_state.db (o similar, dependiendo de si usa BoltDB o SQLite según la versión)
- Rootless: ~/.local/share/containers/storage/libpod/bolt_state.db.

3. La alternativa "limpia" y segura

Dado que mencionaste que vas a centrarte en **Python**, la mejor forma de "toquetear" la configuración de forma programática sin romper nada es usar la **API de Podman** o su librería de Python. En lugar de editar un .db

, puedes hacer un script qu

1. Lea la configuración del contenedor actual (podman inspect).
2. Extraiga la imagen, volúmenes y variables de entorno.
3. Lo borre y lo cree de nuevo con la IP correcta.

Ejemplo rápido en Python:Python

CODE

```
import podman

# Conecta con el socket de podman
with podman.PodmanClient(base_url="unix:///run/user/1000/podman/podman.sock") as client:
    container = client.containers.get("mi_contenedor")
# Aquí podrías automatizar la recreación con los nuevos parámetros de red
print(container.attrs['NetworkSettings']['Ports'])
```

Resumen

Si tocas la base de datos SQLite directamente:

- Probabilidad de éxito: 5% (y requiere reiniciar todo el sistema de contenedores).
- Probabilidad de desastre: 95% (tendrás que borrar /var/lib/containers y empezar de cero).

Mi recomendación: Usa la terminal para hacer un podman stop, podman rm y podman run. Es más rápido, más seguro y es la forma en la que Podman espera que trabajes. ¿Quieres que te ayude a redactar el comand o podman run exacto para uno de tus contenedores basándonos en lo que tienes ahora? Solo pásame el resultado de un podman inspect <nombre> y te lo prepa

Jordi H.

Has dicho pues voy a tener que hacer esos podman inspect + podman run para una media docena de contenedores mal especificados. además tengo creados servicios systemd para arrancar en boot. hay faena !!!
! empezamos. primer caso, una bd Mongo: [root@ymir vllm]# podman inspect MongoDB [{ "Id": "6153f39e14d649fa322293b8ce6d5b32bb90b2a65f6c39964a6298129a2dde95", "Create": "2025-11-11T10:53:54.420881432+01:00", "Path": "python3", "Args": ["/usr/local/bin/docker-entrypoint.py", "mongod"], "State": { "OciVersion": "1.1.0+dev", "Status": "running", "Running": true, "Paused": false, "Restarting": false, "OOMKilled": false, "Dead": false, "Pid": 4833, "CommonPid": 4656, "ExitCode": 0, "Error": "container is stopped", "StartedAt": "2026-03-10T05:41:40.942135643+01:00", "FinishedAt": "2026-03-10T05:39:35.314734974+01:00", "Health": { "Status": "", "FailingStreak": 0, "Log": null }, "CgroupPath": "/machine.slice/libpod-6153f39e14d649fa322293b8ce6d5b32bb90b2a65f6c39964a6298129a2dde95.scope", "CheckpointedAt": "0001-01-01T00:00:00Z", "RestoredAt": "0001-01-01T00:00:00Z", "Image": "324bbe4f61c0c2d1264378595686c278d204eb11f771670d9984e9c3bfce615e", "ImageDigest": "sha256:4c6f9fd348026ab15ad21bb7b58bcb48e81de34c7cad8bdbc9161244b296d6a0", "ImageName": "docker.io/mongodb/mongodb-community-server:latest", "Rootfs": "", "Pod": "", "ResolvConfPath": "/dades/run/containers/storage/overlay-containers/6153f39e14d649fa322293b8ce6d5b32bb90b2a65f6c39964a6298129a2dde95/userdata/resolv.conf", "HostnamePath": "/dades/run/containers/storage/overlay-containers/6153f39e14d649fa322293b8ce6d5b32bb90b2a65f6c39964a6298129a2dde95/userdata/hostname", "HostsPath": "/dades/run/containers/storage/overlay-containers/6153f39e14d649fa322293b8ce6d5b32bb90b2a65f6c39

```

964a6298129a2dde95/userdata/hosts",      "StaticDir": "/dades/var/lib/containers/storage/overlay-containe
rs/6153f39e14d649fa322293b8ce6d5b32bb90b2a65f6c39964a6298129a2dde95/userdata",      "OCIConfigPa
th": "/dades/var/lib/containers/storage/overlay-containers/6153f39e14d649fa322293b8ce6d5b32bb90b2a
65f6c39964a6298129a2dde95/userdata/config.json",      "OCIRuntime": "runc",      "ConmonPidFile":
"/dades/run/containers/storage/overlay-containers/6153f39e14d649fa322293b8ce6d5b32bb90b2a65f6c3996
4a6298129a2dde95/userdata/conmon.pid",      "PidFile": "/dades/run/containers/storage/overlay-containe
rs/6153f39e14d649fa322293b8ce6d5b32bb90b2a65f6c39964a6298129a2dde95/userdata/pidfile",
  "Name": "MongoDB",      "RestartCount": 0,      "Driver": "overlay",
  "MountLabel": "",      "ProcessLabel": "",      "AppArmorProfile": "",
"EffectiveCaps": null,      "BoundingCaps": [      "CAP_CHOWN",
  "CAP_DAC_OVERRIDE",      "CAP_FOWNER",      "CAP_FSETID",
  "CAP_KILL",      "CAP_NET_BIND_SERVICE",
  "CAP_NET_RAW",      "CAP_SETFCAP",      "CAP_SETGID",
  "CAP_SETPCAP",      "CAP_SETUID",      "CAP_SYS_CHROOT"
],      "ExecIDs": [],      "GraphDriver": {      "Name": "over
lay",      "Data": {      "LowerDir": "/dades/var/lib/containers/storage/overla
y/6efdb5a114ce69407ebcd49654d276d407f589359e38a03007ffc4ad31b6b036/diff:/dades/var/lib/containers/storag
e/overlay/5c4c9d fb5bd8885e4fe0adeeb82fb0d42e0e07943f2b816b9ac04c812331d777/diff:/dades/var/lib/contai
ners/storage/overlay/6d22b8e15a4793eff68af61f91ab051a9aaed11624692fa0b7879508d85dda18/diff:/dades/var/li
b/co ntainers/storage/overlay/fb4dd80312e63cb5f956edb14c2d48ecf9c9f3ef5e62d7046e4f893b4032c8e6/diff:/d
ades/var/lib/containers/storage/overlay/bb3f2a54b5290839b05c80e2186358c77d74e746d4df1eb40dcaf0c5 35c22
9f5/diff:/dades/var/lib/containers/storage/overlay/ce79292feb177890c7b18a8f76767e3f2f546218f0c2e4d35
c3e345955703912/diff:/dades/var/lib/containers/storage/overlay/bfdd6b9abd9823e8a01f57c4a 2377eeafc5f9
8b515197ea558448ecd08260638/diff:/dades/var/lib/containers/storage/overlay/34740bb8394eec02a02851143
245dc0eb046977221e1f18c5113470160f7497c/diff:/dades/var/lib/containers/storage/ov erlay/45dec1815c9b3c6
330a14cbb7dd3d5e457f7211b50ae1ab7b6e874a184ca1740/diff:/dades/var/lib/containers/storage/overlay/bedf9a
c8250062299f9dd80a98c3f361cde10962bc7e5cde831e1f4b9f3487bb/diff:/dade s/var/lib/containers/storage/overl
ay/9fbd4552f5c170edfd948a615e4db2026075e33dc0f1f6c46c32be016072350b/diff:/dades/var/lib/containers/sto
rage/overlay/767e56ba346ae714b6e6b816baa839051145ed78cfa0 e4524a86cc287b0c4b00/diff",
  "MergedDir": "/dades/var/lib/containers/storage/overlay/0912596659e95be3f7c71c970702df74929e
1e2e854f64b025b10b09505282a7/merged",      "UpperDir": "/dades/var/lib/containers/storage/
overlay/0912596659e95be3f7c71c970702df74929e1e2e854f64b025b10b09505282a7/diff",      "Wo
rkDir": "/dades/var/lib/containers/storage/overlay/0912596659e95be3f7c71c970702df74929e1e2e
854f64b025b10b09505282a7/work"      }      },      "Mounts": [      {
        "Type": "volume",      "Name": "635bbdd2cc0dcf68cb0a1ed52a78a904d6
f74cd03de304f2c4bba480fd3149dc",      "Source": "/dades/var/lib/containers/storage/volu
mes/635bbdd2cc0dcf68cb0a1ed52a78a904d6f74cd03de304f2c4bba480fd3149dc/_data",
        "Destination": "/data/configdb",      "Driver": "local",      "Mode": "
",      "Options": [      "node
v",      "exec",      "nosuid",
        "rbind"      ],      "RW": true,      "Prop
agation": "rprivate"      },      {      "Type": "bind",
        "Source": "/dades/mongo8",      "Destination": "/dat
a/db",      "Driver": "",      "Mode": "",      "Opt
ions": [      "rbind"      ],      "RW": true,
        "Propagation": "rprivate"      }      ],      "Depend
encies": [],      "NetworkSettings": {      "EndpointID": "",      "Gatewa
y": "10.88.0.1",      "IPAddress": "10.88.0.3",      "IPPref
ixLen": 16,      "IPv6Gateway": "",      "GlobalIPv6Address"
: "",      "GlobalIPv6PrefixLen": 0,      "MacAddress": "76:21:ce:75:c2:ef",
        "Bridge": "",      "SandboxID": "",      "HairpinMode": f
alse,      "LinkLocalIPv6Address": "",      "LinkLocalIPv6PrefixLen": 0,
        "Ports": {      "27017/tcp": [

```

```

{
    "HostIp": "",
    "HostPort": "80
00"
    }
    ],
    "SandboxKey": "/run
/netns/netns-b704a5e8-a1d0-00f6-5d78-2fa27daf9503",
    "Networks": {
        "podman": {
            "EndpointID": "",
            "Gateway": "10.88.0.
1",
            "IPAddress": "10.88.0.3",
            "IPPref
ixLen": 16,
            "IPv6Gateway": "",
            "GlobalIPv6Address"
: "",
            "GlobalIPv6PrefixLen": 0,
            "MacAddress": "
76:21:ce:75:c2:ef",
            "NetworkID": "podman",
            "Drive
rOpts": null,
            "IPAMConfig": null,
            "Links": null,
            "Aliases": [
            "6
153f39e14d6"
            ]
            }
        },
        "Name
space": "",
        "IsInfra": false,
        "IsService": false,
        "KubeExitCodePropagatio
n": "invalid",
        "lockNumber": 0,
        "Config": {
            "Hostname": "6153f3
9e14d6",
            "Domainname": "",
            "User": "mongodb",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": fals
e,
            "Tty": false,
            "OpenStdin"
: false,
            "StdinOnce": false,
            "Env":
[
            "MONGODB_INITDB_ROOT_PASSWORD=Mongoplex.00",
            "
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
            "container=podman",
            "HOME
=/data/db",
            "GLIBC_TUNABLES=glibc.pthread.r
seq=0",
            "MONGODB_INITDB_ROOT_USERNAME=myUserAdmin",
            "HOSTNAME=6153f39e14d6"
            ],
            "Cmd
": [
            "mongod"
            ],
            "Image": "docker.io
/mongodb/mongodb-community-server:latest",
            "Volumes": null,
            "WorkingDir": "/",
            "Entrypoint": "python3 /usr/local/bin/docker-entrypoint.py",
            "OnBuild": n
ull,
            "Labels": {
                "description": "Container configu
red with a standalone instance of MongoDB",
                "maintai
ner": "support@mongodb.com",
                "name": "MongoDB
Standalone",
                "org.opencontainers.image.ref.name": "ubuntu",
                "org.opencontainers.image.version": "22.04",
                "s
ummary": "MongoDB Standalone Container",
                "vendor": "MongoDB",
                "version": "8.2.1"
            },
            "Annotations": {
                "io.container.manager": "libpo
d",
                "org.opencontainers.image.stopSignal": "15"
            },
            "StopSignal": 15,
            "HealthcheckOnFailureAction": "none",
            "CreateCommand": [
                "podman",
                "run
",
                "-d",
                "--name",
                "Mongo
DB",
                "-p",
                "8000:27017",
                "-v",
                "/dades/mongo8:/data/db",
                "-e"
,
                "MONGODB_INITDB_ROOT_USERNAME=myUserAdmin"
,
                "-e",
                "MONGODB_INITDB_ROOT_PASSWORD=Mongoplex.0
0",
                "docker.io/mongodb/mongodb-community-server:latest"
            ],
            "Umask": "0022",
            "Timeout": 0,
            "StopTimeout": 10,
            "Passwd": true,
            "sdNotifyMode": "container",
            "ExposedPor
ts": {
                "27017/tcp": {}
            },
            "HostConfig": {
                "Binds": [
                    "635bbdd2cc0dcf68cb0a1e
d52a78a904d6f74cd03de304f2c4bba480fd3149dc:/data/configdb:rprivate,rw,nodev,ex
ec,nosuid,rbind",
                    "/dades/mongo8:/data/db:rw,rprivate,rbin
d"
                ],
                "CgroupManager": "systemd",
                "CgroupMode": "host",
                "ContainerIDFile": "",
                "LogConfig": {
                    "Type": "k8s-file",
                    "Config": null,
                    "Path": "/dades/var/lib/containers/storage/ove
rlay-containers/6153f39e14d649fa322293b8ce6d5b32bb90b2a65f6c39964a6298129a2dde

```

```

95/userdata/ctr.log",          "Tag": "",          "Size": "0B"          },
    "NetworkMode": "bridge",    "PortBindings": {
        "27017/tcp": [         {                     "Host
Ip": "",                        "HostPort": "8000"
    }                          ],                    "RestartPolicy": {
    "Name": "",                  "MaximumRetryCount": 0          },          "AutoRemove
": false,                       "VolumeDriver": "",            "VolumesFrom": null,
    "CapAdd": [],               "CapDrop": [],                 "Dns": [],          "D
nsOptions": [],                 "DnsSearch": [],               "ExtraHosts": [],   "GroupAd
d": [],                          "IpcMode": "shareable",         "Cgroup": "",        "Cgroups
": "default",                    "Links": null,                  "OomScoreAdj": 0,    "PidMode": "priva
te",                              "Privileged": false,            "PublishAllPorts": false, "Readonl
yRootfs": false,                 "SecurityOpt": [],              "Tmpfs": {},         "UTSMod
e": "private",                   "UsernsMode": "",              "ShmSize": 65536000,
    "Runtime": "oci",           "ConsoleSize": [               0,
    0                           ],          "Isolation": "",     "CpuShares": 0,
    "Memory": 0,                "NanoCpus": 0,                 "CgroupParent": "
",                                "BlkioWeight": 0,              "BlkioWeightDevice": null,
    "BlkioDeviceReadBps": null,  "BlkioDeviceWriteBps": null,    "BlkioDevice
ReadIOps": null,                 "BlkioDeviceWriteIOps": null,   "
CpuPeriod": 0,                   "CpuQuota": 0,                  "CpuRealtimePeriod": 0,
    "CpuRealtimeRuntime": 0,     "CpusetCpus": "",               "CpusetMems": "",
    "Devices": [],               "DiskQuota": 0,                 "
KernelMemory": 0,                "MemoryReservation": 0,         "MemorySwap": 0,
    "MemorySwappiness": 0,        "OomKillDisable": false,
    "PidsLimit": 2048,           "Ulimits": [                    {
        "Name": "RLIMIT_NPROC",    "Soft": 4194304,
        "Hard": 4194304            }                ],
    "CpuCount": 0,              "CpuPercent"

```

Gemini

¡Excelente! Tener el inspect a mano es como tener el plano original. Como bien dices, hay faena, pero al tener servicios de **systemd** de por medio, tenemos que ser quirúrgicos para no romper el arranque automático. Aquí tienes el plan de ataque para **MongoD**

1. Análisis del objetivo

Actualmente tienes:

- Puerto: 8000:27017 (Escuchando en todas las IPs, 0.0.0.0).
- Volúmenes: /dades/mongo8:/data/db (Bind mount crítico para no perder tus datos).
- Servicio: Tienes un archivo .service que probablemente está intentando levantar el contenedor viejo.

2. El nuevo comando `podman run`

Suponiendo que quieres que escuche solo en una IP (usaré 127.0.0.1 como ejemplo, cámbiala por la IP de tu red local si es necesario), el comando corregido basado en tu JSON es: Bas

CODE

```
podman run -d \
--name MongoDB_new \
-p 127.0.0.1:8000:27017 \
-v /dades/mongo8:/data/db:rw,rprivate,rbind \
-e MONGODB_INITDB_ROOT_USERNAME=myUserAdmin \
-e MONGODB_INITDB_ROOT_PASSWORD=Mongoplex.00 \
docker.io/mongodb/mongodb-community-server:latest
```

3. Operación de reemplazo (Paso a paso)

Para no entrar en conflicto con systemd, sigue este orden:

1. Detén el servicio de systemd (esto detendrá el contenedor actual):Bash

CODE

```
systemctl stop container-MongoDB.service
```

2. Elimina el contenedor antiguo:Bash

CODE

```
podman rm MongoDB
```

3. Crea el contenedor nuevo con el nombre original (para que systemd lo encuentre):Bash

CODE

```
podman run -d \
--name MongoDB \
-p 127.0.0.1:8000:27017 \
-v /dades/mongo8:/data/db:rw,rprivate,rbind \
-e MONGODB_INITDB_ROOT_USERNAME=myUserAdmin \
-e MONGODB_INITDB_ROOT_PASSWORD=Mongoplex.00 \
docker.io/mongodb/mongodb-community-server:latest
```

4. Prueba el servicio:Bash

CODE

```
systemctl start container-MongoDB.service
```

4. Ojo con el archivo de Systemd

Si el archivo de servicio de systemd fue generado con podman generate systemd --new, **ten cuidado**. Esos archivos ignoran el contenedor que acabas de crear y prefieren crear uno efímero cada vez que arranca n basándose en la configuración que se guardó en el momento de generar el archivo. **¿Cómo saber!**

Mira el archivo (probablemente en /etc/systemd/system/container-MongoDB.service o ~/.config/systemd/user/):

- Si ves una línea muy larga que empieza por ExecStart=/usr/bin/podman run ..., tienes que editar esa línea a mano para cambiar el -p 8000:27017 por -p 127.0.0.1:8000:27017
- Después de editarlo, recuerda ejecutar: systemctl daemon-reload.

Resumen de la "limpieza"

Para los otros 5 contenedores, el patrón será el mismo:

1. podman inspect (ya lo tenemos para MongoDB).

2. Editar el archivo `.service` de `systemd` (es más limpio que borrar y recrear si el servicio usa `--new`).
3. `systemctl daemon-reload`.
4. `systemctl restart contenedor`.

¿Quieres que revisemos el archivo `.service` de este MongoDB para asegurarnos de que el cambio sea permanente tras un `reboot`

Jordi H.

Has dicho tengo la sospecha que con la tag `:latest` puedo hacer un upgrade y ahora no me interesa. el `inspect` te dió la imagen exacta

Gemini

Tu sospecha es totalmente acertada y demuestras muy buen ojo clínico. En entornos de producción o bases de datos, el tag `:latest` es un peligro porque es un "blanco móvil". Efectivamente, el `inspect` nos ha dado la identidad exacta del contenedor en dos campos clave

1. `Image: 324bbe4f61c0c2d1264378595686c278d204eb11f771670d9984e9c3bfce615e` (Este es el ID local de la imagen)
2. `ImageDigest: sha256:4c6f9fd348026ab15ad21bb7b58bcb48e81de34c7cad8bdbbc9161244b296d6a0` (Este es el "DNI" inmutable en el registro)

Cómo hacer el "Pinning" (fijar la versión)

Para garantizar que **no se actualice nada** y que el contenedor sea idéntico al que tienes ahora, no uses el nombre de la imagen, usa el **ID** que nos dio el `inspect`. Sustituye la última línea de mi comando anterior por el ID largo (o los primeros 12 caracteres): `Ba`

CODE

```
podman run -d \
--name MongoDB \
-p 127.0.0.1:8000:27017 \
-v /dades/mongo8:/data/db:rw,rprivate,rbind \
-e MONGODB_INITDB_ROOT_USERNAME=myUserAdmin \
-e MONGODB_INITDB_ROOT_PASSWORD=Mongoplex.00 \
324bbe4f61c0
```

¿Por qué esto es mejor?

Al usar `324bbe4f61c0` en lugar de `mongodb-community-server:latest`:

- Inmutabilidad: Podman buscará exactamente ese conjunto de capas en tu disco.
- Evitas sorpresas: Aunque mañana MongoDB saque la versión 9.0 y cambie el formato de la base de datos, tu contenedor seguirá arrancando con la misma versión binaria que tienes hoy

El flujo de trabajo con Systemd

Como mencionaste que tienes servicios de `systemd`, aquí es donde la "faena" se vuelve delicada. Si el archivo `.service` tiene escrito `:latest`, la próxima vez que el servidor se reinicie o el servicio se actualice, **intentar** r la nueva versión **si alguien hace un podman pull**. Mi recomendación para este caso de MongoDB: ******

1. Busca el archivo del servicio: `grep -r "MongoDB" /etc/systemd/system/`.
2. Edita la línea `ExecStart`.
3. Cambia `-p 8000:27017` por `-p TU_IP:8000:27017`.

4. Cambia la imagen al final de la línea por el ID 324bbe4f61c0.

De esta forma, Cockpit dejará de "ver" los puertos en 0.0.0.0 y tú dormirás tranquilo sabiendo que no habrá upgrades accidentales. ¿Quieres que verifiquemos el contenido de ese archivo .service antes de que lances el primer systemctl restart

Jordi H.

Has dicho la línea execstart parece inofensiva: ExecStart=/usr/bin/podman start MongoDB

Gemini

Esa es una excelente noticia. Si tu ExecStart es simplemente podman start MongoDB, significa que tu servicio de systemd es **dependiente del estado del contenedor**, no un servicio que lo recrea desde cero (lo cual es mucho más fácil de gestionar en este momento). Sin embargo, tiene una "trampa": como el comando solo dice start, systemd se limita a intentar arrancar el objeto "MongoDB" que ya existe en la base de datos de Podman. Para cambiar la IP, **tienes que recrear ese objeto** por fuerza.

El Plan de Acción (Seguro y sin Upgrades)

Como ya confirmamos que el ID de tu imagen es 324bbe4f61c0, vamos a hacer el "cambiao" manual. Al borrarlo y recrearlo con el mismo nombre, el servicio de systemd ni siquiera se enterará de que el contenedor es "nuevo"; simplemente lo arrancará. Ejecuta estos comandos en orden:

1. Detén el servicio actual: Bash

CODE

```
systemctl stop container-MongoDB.service
```

2. Elimina el contenedor mal configurado: Bash

CODE

```
podman rm MongoDB
```

3. Crea el contenedor con la IP fija y el ID de imagen exacto:

Usa la IP específica que quieras (sustituye 192.168.1.XX por la tuya): Bash

CODE

```
podman run -d \
--name MongoDB \
-p 192.168.1.XX:8000:27017 \
-v /dades/mongo8:/data/db:rw,rprivate,rbind \
-e MONGODB_INITDB_ROOT_USERNAME=myUserAdmin \
-e MONGODB_INITDB_ROOT_PASSWORD=Mongoplex.00 \
324bbe4f61c0
```

(Nota: Al usar el ID 324bbe4f61c0, garantizamos que no descargue nada nuevo).

4. Arranca el servicio de systemd: Bash

CODE

```
systemctl start container-MongoDB.service
```

¿Por qué esto es mejor que editar el `.service`?

Al mantener el ExecStart genérico (podman start MongoDB), tu archivo de systemd es más robusto. Si mañana decides cambiar el puerto otra vez, solo tienes que repetir este proceso de "borrar y crear" en Podman y el servicio de systemd seguirá funcionando sin tocar una sola línea de su configuración

Verificación final

Una vez hecho, comprueba que solo escucha en tu IP con: Bash

CODE

```
ss -tulpn | grep 8000
```

Debería aparecer 192.168.1.XX:8000 en lugar de *:8000 o 0.0.0.0:8000 ¿Quieres que pasemos al siguiente rimero si este MongoDB ha levantado correctamente en Cockpit?**

Jordi H.

```
Has dicho pasamos al siguiente, un servidor elasticsearch [root@ymir ~]# podman inspect es01 [ {
  "Id": "4ab24cbba949bd6b771f91924a2b4e4ecb51ed83f6990a45a88eced31adff94a"
,   "Created": "2025-03-07T12:47:27.606024548+01:00",   "Path": "/bin/tini",   "Args"
": [   "--",   "/usr/local/bin/docker-entrypoint.sh",   "eswrapp
er"   ],   "State": {   "OciVersion": "1.1.0+dev",   "Stat
us": "running",   "Running": true,   "Paused": false,   "Restart
ing": false,   "OOMKilled": false,   "Dead": false,   "Pid":
4488,   "ConmonPid": 4361,   "ExitCode": 0,   "Error": "conta
iner is stopped",   "StartedAt": "2026-03-10T05:41:40.865304042
+01:00",   "FinishedAt": "2026-03-10T05:39:37.381722448+01:00",   "H
ealth": {   "Status": "",   "FailingStreak": 0,   "
Log": null   },   "C
groupPath": "/machine.slice/libpod-4ab24cbba949bd6b771f91924a2b4e4ecb51ed83f6990a45a88eced31adff94a.scope",   "CheckpointedAt": "0001-01-01T00:00:00Z",
"RestoredAt": "0001-01-01T00:00:00Z"   },
  "Image": "869b1daa5a9f7d3087840d9aa8601cc8f7e02678b6d224b107295e2f3504edaf",
  "ImageDigest": "sha256:224c75e346bd745ce908f06a1cbad7bf10988961dcdcdfccb2
2556b3f856b3f0",   "ImageName": "docker.elastic.co/elasticsearch/elasticsearch:8.17.3",
  "Rootfs": "",   "Pod": "",   "ResolvConfPath": "/dades/run/containers/storage/overlay-co
ntainers/4ab24cbba949bd6b771f91924a2b4e4ecb51ed83f6990a45a88eced31adff94a/user
data/resolv.conf",   "HostnamePath": "/dades/run/containers/storage/overlay-containers/4ab24cbb
a949bd6b771f91924a2b4e4ecb51ed83f6990a45a88eced31adff94a/userdata/hostname",   "HostsPath": "/
dades/run/containers/storage/overlay-containers/4ab24cbba949bd6b771f91924a2b4e4ecb51ed83f6990a45a8
8eced31adff94a/userdata/hosts",   "StaticDir": "/dades/var/lib/containers/storage/overlay-containers/
4ab24cbba949bd6b771f91924a2b4e4ecb51ed83f6990a45a88eced31adff94a/userdata",   "OCIConfigPath":
"/dades/var/lib/containers/storage/overlay-containers/4ab24cbba949bd6b771f91924a2b4e4ecb51ed83f699
0a45a88eced31adff94a/userdata/config.json",   "OCIRuntime": "runc",   "ConmonPidFile": "/dade
s/run/containers/storage/overlay-containers/4ab24cbba949bd6b771f91924a2b4e4ecb51ed83f6990a45a88ece
d31adff94a/userdata/conmon.pid",   "PidFile": "/dades/run/containers/storage/overlay-containers/4ab
24cbba949bd6b771f91924a2b4e4ecb51ed83f6990a45a88eced31adff94a/userdata/pidfile",
  "Name": "es01",   "RestartCount": 0,   "Driver": "overlay",
  "MountLabel": "",   "ProcessLabel": "",   "AppArmorProfile": "",   "
EffectiveCaps": null,   "BoundingCaps": [   "CAP_CHOWN",
  "CAP_DAC_OVERRIDE",   "CAP_FOWNER",   "CAP_FSETID",
```

```

"CAP_KILL", "CAP_NET_BIND_SERVICE",
"CAP_NET_RAW", "CAP_SETFCAP", "CAP_SETGID",
"CAP_SETPCAP", "CAP_SETUID", "CAP_SYS_CHROOT"
], "ExecIDs": [], "GraphDriver": { "Name": "overlay
y", "Data": { "LowerDir": "/dades/var/lib/containers/storage/overlay/cb
abe0ff47d258af0ff9859a7de2325a0f1e677b8081a8f383818952786e9e2d/diff:/dades/var/lib/containers/storage/o
verlay/49c9f4 8902793d4020d97473cdc9b6824cd663f1ba25ebcac7f6c3b3368f9596/diff:/dades/var/lib/containers/
storage/overlay/fcce44991cd02af17430214419d54d23566b9b718a6fae742bc05fff1917e6d7/diff:/dades/var/lib/co
ntainers/storage/overlay/0fa29cea04794bc9136e6e506105ce70e6733f18dd29603959227d0c07185c6f/diff:/dades
/var/lib/containers/storage/overlay/716d1227a5e3b9f66719f99a8d1c66e0d9e9a76f54de2e68dfeefd15 613c2e2c/
diff:/dades/var/lib/containers/storage/overlay/8966f2cfe22d5eec7e1793106a686553e93ce9251c84785612e51
79f083daad9/diff:/dades/var/lib/containers/storage/overlay/98a2b9f539355af8217f7d222 648aee9ad7f3b8761a
5c356813425e7b7ef259f/diff:/dades/var/lib/containers/storage/overlay/a0873e8e84e31b387ff250d2ca64d2
4de75bef4da949276e499b435d43a8ea6f/diff:/dades/var/lib/containers/storage/ov erlay/6c5f96c7f9f267e85bf46d
139095756e2d6f2e5bc7f14037bc5a6f9c08d414c9/diff:/dades/var/lib/containers/storage/overlay/fffe76c64ef2de
e2d80a8bb3ad13d65d596d04a45510b1956a976a69215dae92/diff", "MergedDir": "/dades/var/lib
/containers/storage/overlay/f8ad8f578d55758afc53e1e6e2b1193b4fd92ecebe2a4
ad954b269aed616ae00/merged", "UpperDir": "/dades/var/lib/containers/storage/overlay/
f8ad8f578d55758afc53e1e6e2b1193b4fd92ecebe2a4ad954b269aed616ae00/diff", "WorkDir": "
/dades/var/lib/containers/storage/overlay/f8ad8f578d55758afc53e1e6e2b1193b4fd92eceb
e2a4ad954b269aed616ae00/work" } }, "Mounts": []
, "Dependencies": [], "NetworkSettings": { "EndpointID": "
", "Gateway": "", "IPAddress": "", "IPPrefixLen": 0,
"IPv6Gateway": "", "GlobalIPv6Address": "",
"GlobalIPv6PrefixLen": 0, "MacAddress": "", "Bridge":
"", "SandboxID": "", "HairpinMode": false,
"LinkLocalIPv6Address": "", "LinkLocalIPv6PrefixLen": 0, "Ports"
: { "9200/tcp": [ { "HostIp": "", "HostPort": "8008"
} ], "9300/tcp": null
}, "SandboxKey": "/run/netns/netns-815e0f9c-04a2-10cd-e76d-7a
d6b5aca695", "Networks": { "elastic": {
"EndpointID": "", "Gateway": "10.89.1.1",
"IPAddress": "10.89.1.2", "IPPrefixLen": 24,
"IPv6Gateway": "", "GlobalIPv6Address": "",
"GlobalIPv6PrefixLen": 0, "MacAddress": "da:20:3a:44
:e3:6c", "NetworkID": "elastic", "DriverOpts": nul
l, "IPAMConfig": null, "Lin
ks": null, "Aliases": [ "4ab24cbba949
" ] } }, "Namespace": "",
"IsInfra": false, "IsService": false, "KubeExitCodePropagation": "invali
d", "lockNumber": 3, "Config": { "Hostname": "4ab24cbba949",
"Domainname": "", "User": "1000:0", "Attac
hStdin": false, "AttachStdout": false, "AttachStderr": false,
"TTY": true, "OpenStdin": true, "StdinOnce": false,
"Env": [ "TERM=xterm", "PATH=/usr/share/elasticsearch/bin:/usr/loc
al/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
"container=podman", "SHELL=/bin/bash", "ELASTIC_CONTAINER=tr
ue", "HOME=/usr/share/elasticsearch", "HOSTN
AME=4ab24cbba949" ], "Cmd": [
"eswrapper" ], "Image": "docker.elastic.co/elastics
earch/elasticsearch:8.17.3", "Volumes": null, "WorkingDir": "/usr/sha
re/elasticsearch", "Entrypoint": "/bin/tini -- /usr/local/bin/docker-entrypoint.

```

```

sh",      "OnBuild": null,      "Labels": {      "o
rg.label-schema.build-date": "2025-02-28T10:07:26.089129809Z",
      "org.label-schema.license": "Elastic-License-2.
0",      "org.label-schema.name": "Elasticsearch",      "org.label-schema.s
chema-version": "1.0",      "org.label-schema.url": "https://www.elastic.co/pro
ducts/elasticsearch",      "org.label-schema.usage": "h
ttps://www.elastic.co/guide/en/elasticsearch/reference/index.html",      "or
g.label-schema.vcs-ref": "a091390de485bd4b127884f7e565c0cad59b10d2",      "o
rg.label-schema.vcs-url": "https://github.com/elastic/elasticsearch",      "org.label-sch
ema.vendor": "Elastic",      "org.label-schema.version": "8.17.3",
      "org.opencontainers.image.created": "2025-02-28T10:07:26.0
89129809Z",      "org.opencontainers.image.documentatio
n": "https://www.elastic.co/guide/en/elasticsearch/reference/index.html",
      "org.opencontainers.image.licenses": "Elastic-Lice
nse-2.0",      "org.opencontainers.image.ref.name": "ubuntu",      "org
.opencontainers.image.revision": "a091390de485bd4b127884f7e565c0cad59b10d2",
org.opencontainers.image.source": "https://github.com/elastic/elastic
search",      "org.opencontainers.image.title": "Elasticsearch",
org.opencontainers.image.url": "https://www.elastic.co/products/
elasticsearch",      "org.opencontainers.image.vendor": "Elastic",
      "org.opencontainers.image.version": "8.17.3"    },
      "Annotations": {      "io.container.manager": "libpo
d",      "org.opencontainers.image.stopSignal": "15"    },
      "StopSignal": 15,      "HealthcheckOnFailureAction": "none",
      "CreateCommand": [      "/usr/bin/podman",      "run
",      "--name",      "es01",      "--net",
      "elastic",      "-p",      "8
008:9200",      "-it",      "-m",      "32GB",
      "docker.elastic.co/elasticsearch/elasticsearch:8.17.3"    ],
      "Umask": "0022",      "Timeout": 0,      "StopTimeout": 10,
      "Passwd": true,      "sdNotifyMode": "container",      "ExposedPorts": {
      "9200/tcp": {},      "9300/tcp": {}    }    },
      "HostConfig": {      "Binds": [],      "CgroupManager": "systemd",
      "CgroupMode": "host",      "ContainerIDFile": "",      "LogConfig": {
      "Type": "k8s-file",      "Config": null,      "Path": "/dades/va
r/lib/containers/storage/overlay-containers/4ab24cbba949bd6b771f91924a2b4e4ecb51ed83f6990a45a8
8eced31adff94a/userdata/ctr.log",      "Tag": "",      "Siz
e": "0B"    },      "NetworkMode": "bridge",      "PortBindings": {
      "9200/tcp": [      {      "HostIp":
"",      "HostPort": "8008"    }    ]    },      "RestartPolicy": {      "Name
": "",      "MaximumRetryCount": 0    },      "AutoRemove": fa
lse,      "VolumeDriver": "",      "VolumesFrom": null,
      "CapAdd": [],      "CapDrop": [],      "Dns": [],      "DnsOpt
ions": [],      "DnsSearch": [],      "ExtraHosts": [],      "GroupAdd": [
],      "IpcMode": "shareable",      "Cgroup": "",      "Cgroups": "d
efault",      "Links": null,      "OomScoreAdj": 0,      "PidMode": "private",
      "Privileged": false,      "PublishAllPorts": false,      "ReadOnlyRoot
fs": false,      "SecurityOpt": [],      "Tmpfs": {},      "UTSMode": "
private",      "UsersnsMode": "",      "ShmSize": 65536000,
      "Runtime": "oci",      "ConsoleSize": [      0,      0
],      "Isolation": "",      "CpuShares": 0,      "Memor
y": 34359738368,      "NanoCpus": 0,      "CgroupParent": "",

```

```

        "BlkioWeight": 0,          "BlkioWeightDevice": null,
"BlkioDeviceReadBps": null,      "BlkioDeviceWriteBps": null,    "BlkioDeviceReadI
Ops": null,          "BlkioDeviceWriteIOps": null,          "CpuPe
riod": 0,            "CpuQuota": 0,          "CpuRealtimePeriod": 0,      "
CpuRealtimeRuntime": 0,          "CpusetCpus": "",          "CpusetMems": "",
    "Devices": [],          "DiskQuota": 0,          "KernelMemory": 0
,          "MemoryReservation": 0,          "MemorySwap": 68719476736,          "Me
morySwappiness": -1,          "OomKillDisable": false,
    "PidsLimit": 2048,          "Ulimits": [          {
        "Name": "RLIMIT_NPROC",          "Soft": 4194304,
        "Hard": 4194304          }          ],
    "CpuCount": 0,          "CpuPercent": 0,

```

Gemini

Vamos con **Elasticsearch** (es01). Este caso tiene un par de detalles interesantes: usa una red específica llamada elastic y tiene límites de memoria bastante generosos (32GB)

1. Análisis del objetivo (`es01`)

- Imagen exacta: 869b1daa5a9f (Elasticsearch 8.17.3).
- Red: elastic (es fundamental mantenerla para que otros servicios como Kibana sigan viéndolo).
- Puerto: 8008:9200. Actualmente en 0.0.0.0.
- Memoria: 32GB (-m 32GB).
- Interactividad: Veo que se creó con -it (interactivo y TTY), probablemente porque Elasticsearch a veces lo requiere para el log inicial, aunque en un servicio systemd no es crítico

2. El comando de recreación (Pinning de IP e Imagen)

Sustituye 192.168.1.XX por la IP fija que desees: Bash

CODE

```

podman run -d \
--name es01 \
--net elastic \
-p 192.168.1.XX:8008:9200 \
-m 32GB \
-it \
869b1daa5a9f

```

3. Operación de "Cambiarzo"

Al igual que con MongoDB, asumimos que tu servicio de systemd simplemente hace un podman start es01.

1. Detener el servicio: Bash

CODE

```
systemctl stop container-es01.service
```

2. Eliminar el contenedor viejo: Bash

CODE

```
podman rm es01
```

3. Lanzar el nuevo (con la IP corregida):Bash

CODE

```
podman run -d \  
--name es01 \  
--net elastic \  
-p 192.168.1.XX:8008:9200 \  
-m 32GB \  
-it \  
869b1daa5a9f
```

4. Verificar y arrancar systemd:Bash

CODE

```
systemctl start container-es01.service
```

¡OJO! Un detalle importante sobre los datos

En el inspect de este contenedor, veo que **Mounts está vacío**:

"Mounts": [], Esto significa que **no estás usando volúmenes externos** para los datos de Elasticsearch. Si borras el contenedor (podman rm), **perderás los índices y datos que haya dentro** a menos que estuvieran en un volumen anónimo que no aparece aquí (poco probable en este config) **Antes de borrar nada**

¿Este Elasticsearch tiene datos importantes? Si es así, **no lo borres todavía**. Deberíamos recrearlo mapeando una carpeta de tu host (como hicimos con /dades/mongo8) para que los datos sobrevivan. Si es solo un entorno de pruebas o análisis que puedes recrear con Python, adelante **¿Quieres que busquemos dónde** datos antes de ejecutar el rm o procedemos?**

Jordi H.

Has dicho llegamos tarde. pero ya no hay remedio. antes de recrear los índices vamos a crear el volumen extern

Gemini

¡Vaya! Siento el golpe de los datos perdidos, pero miremos el lado bueno: ahora vas a dejar la infraestructura **bien construida** desde los cimientos. Como analista de datos, sabes que la persistencia es lo primero. Para Elasticsearch, no solo necesitamos guardar los datos, sino asegurar que los permisos sean correctos, ya que ES es muy tiquismiquis con quién es el dueño de la carpeta (normalmente el usuario 1000 dentro del contenedor). Aquí tienes los pasos para crear el volumen externo y el contenedor de forma robu

1. Preparar el host (la carpeta de datos)

Vamos a crear una carpeta en tu ruta de datos (/dades) y darle los permisos necesarios para que el contenedor pueda escribir: Bas

CODE

```
mkdir -p /dades/elasticsearch_data
# Elasticsearch dentro del contenedor usa el UID 1000
chown -R 1000:1000 /dades/elasticsearch_data
```

2. El nuevo comando `podman run` con volumen

Ahora sí, incluimos el volumen -v, la red, los límites de memoria y la IP fija. Bash

CODE

```
podman run -d \
--name es01 \
--net elastic \
-p 192.168.1.XX:8008:9200 \
-m 32GB \
-v /dades/elasticsearch_data:/usr/share/elasticsearch/data:rw,z \
-it \
869b1daa5a9f
```

- -v ...:/usr/share/elasticsearch/data: Mapea la carpeta del host a la ruta interna de datos de ES.
- :z: (Opcional) Si usas SELinux, esto le dice a Podman que reetiquete la carpeta para que el contenedor tenga acceso

3. Ajuste de Systemd

Como el contenedor ahora es "nuevo" y tiene una configuración distinta (el volumen), el servicio de systemd simplemente funcionará si el ExecStart sigue siendo podman start es01

4. Un último detalle técnico: `vm.max\_map\_count`

Elasticsearch suele quejarse (o no arrancar) si el host no tiene suficiente memoria virtual asignada. Al recrear el entorno, asegúrate de que esto esté configurado en tu servidor Linux: Bas

CODE

```
# Para aplicar ahora mismo:
sysctl -w vm.max_map_count=262144

# Para que sea permanente tras reiniciar:
echo "vm.max_map_count=262144" >> /etc/sysctl.conf
```

Resumen de la situación

- MongoDB: Listo y con volumen.
 - Elasticsearch: Listo, con volumen externo y IP fija.
- ¿Qué contenedor es el siguiente en la lista? Si tienes algún **Jupyter Notebook** o herramienta de **Python/Analítica** corriendo en contenedor, esos suelen ser los siguientes en dar problemas de puertos

Jordi H.

Has dicho al ser un contenedor nuevo, creo que modifiqué algo del user y pass de acceso. Ahora mismo, cual es la configuración de fábrica

Para las versiones modernas de **Elasticsearch 8.x** (como la **8.17.3** que estás usando), la "configuración de fábrica" ha cambiado radicalmente respecto a las versiones antiguas. Ahora, la seguridad está **activada** por lo que tienes lo que sucede cuando arrancas el contenedor "limpio" con el volumen que acabamos de crear:

1. Usuario y Contraseña por defecto

No hay una contraseña estática (como "password" o "admin"). Al arrancar por primera vez:

- Usuario: elastic
- Contraseña: Se genera automáticamente y se imprime en los logs del contenedor durante el primer arranque

2. Cómo recuperar la contraseña actual

Como ya has lanzado el contenedor, lo más rápido es mirar el log: Bash

CODE

```
podman logs es01 | grep -i "password"
```

Busca una sección que dice: Password for the elastic user: <contraseña_aleatoria>.

3. Si no aparece o quieres cambiarla (Lo más práctico)

Como eres el administrador del host, no necesitas saber la vieja para poner una nueva. Ejecuta este comando directamente dentro del contenedor: Bash

CODE

```
podman exec -it es01 /usr/share/elasticsearch/bin/elasticsearch-reset-password -u elastic
```

Esto te dará una nueva contraseña aleatoria para el usuario elastic. Si prefieres poner una tú mismo: Bash

CODE

```
podman exec -it es01 /usr/share/elasticsearch/bin/elasticsearch-reset-password -u elastic -i
```

4. El "Obstáculo" del Certificado SSL/HTTPS

De fábrica, ES 8.x no solo pone contraseña, sino que también **obliga a usar HTTPS**. Si intentas conectar tu código de Python o un curl por HTTP normal al puerto 8008, te dará un error de conexión vacía o basur a binaria. Tienes dos opciones

- Opción A (Recomendada): Usar el certificado que ES generó automáticamente. Está en: `/usr/share/elasticsearch/config/certs/http_ca.crt`
- Opción B (Para desarrollo rápido): Desactivar la seguridad SSL (no recomendado para producción, pero útil si solo escuchas en una IP local). Para ello, tendrías que editar el archivo `elasticsearch.yml` dentro del volumen: Ba

CODE

```
# En tu host:
nano /dades/elasticsearch_data/elasticsearch.yml
```

Y cambiar `xpack.security.http.ssl.enabled` a `false`.

Resumen de conexión para tu analítica (Python)

Cuando vayas a conectar con Python, tu configuración de "fábrica" ahora mismo pedirá:

1. Protocolo: https://
2. Usuario: elastic
3. Password: (La que saques con el comando reset-password)
4. Certificado: Deberás ignorar la verificación de SSL o pasarle el archivo .crt.

¿Quieres que reseteemos la contraseña ahora mismo o prefieres que busquemos el token de inscripción para

Jordi H.

Has dicho he recreado el contenedor desactivando la seguridad en el comando run. ahora vamos a por mysql. tengo mysql9 y tiene un volumen de datos. debería aprovechar para hacer un downgrade a mysql8. cómo lo ve

Gemini

Hacer un **downgrade de base de datos** (pasar de una versión mayor a una menor, como de MySQL 9 a 8) es uno de los movimientos más peligrosos en administración de sistemas. A diferencia de un upgrade, las bases de datos no están diseñadas para volver atrás. Si intentas arrancar MySQL 8 apuntando a una carpeta de datos que ya ha sido "tocada" por MySQL 9, el contenedor **no arrancará** y probablemente corromperás los archivos de datos.

1. El veredicto: ¿Es buena idea?

Solo si es estrictamente necesario. * **Por qué hacerlo:** MySQL 9 es una versión de "Innovación" (ciclo corto), mientras que MySQL 8.0/8.4 son versiones LTS (Long Term Support). Para analítica de datos, la estabilidad de la 8.0 es preferible.

- El problema: El formato del diccionario de datos y los archivos ibdata cambian. MySQL 8 no sabe leer los cambios que MySQL 9 ha introducido.

2. Cómo hacerlo sin romper nada (Estrategia Dump & Restore)

No puedes simplemente cambiar la imagen en el comando podman run. El proceso correcto es:

1. Exportar (Dump): Mientras MySQL 9 está corriendo, saca un backup completo de los datos.
2. Limpiar: Borrar el contenedor MySQL 9 y vaciar/mover la carpeta de datos del host.
3. Importar (Restore): Levantar el contenedor MySQL 8 (vacío) e importar el backup.

3. Manos a la obra: MySQL 8 con IP fija y Volumen

Primero, saca tu backup (si tienes datos):

```
podman exec MongoDB mysqldump -u root -p --all-databases > backup_total.sql
```

Ahora, vamos a preparar el nuevo contenedor MySQL 8.0 corrigiendo el tema de la IP y los puertos

El comando `podman run` para MySQL 8

Sustituye 192.168.1.XX por la IP que quieras asignar: Bash

CODE

```
podman run -d \
--name mysql_server \
-p 192.168.1.XX:3306:3306 \
-v /dades/mysql_data:/var/lib/mysql:rw,z \
-e MYSQL_ROOT_PASSWORD=TuPasswordSeguro \
docker.io/library/mysql:8.0
```

4. ¿Qué pasa con los servicios de Systemd?

Al igual que en los casos anteriores, si tu archivo .service usa ExecStart=/usr/bin/podman start mysql_server, no tienes que tocar el archivo de sistema. Solo asegúrate de que el nombre del contenedor (--name) coincida con el que espera el servicio

Resumen de pasos para el "Downgrade" seguro:

1. Backup de la base de datos actual (MySQL 9).
2. systemctl stop container-mysql.service
3. podman rm mysql_server
4. Mueve los datos viejos: mv /dades/mysql_data /dades/mysql_data_v9_backup
5. Crea carpeta limpia: mkdir /dades/mysql_data
6. Ejecuta el comando run de arriba (con la imagen :8.0).
7. Importa los datos: podman exec -i mysql_server mysql -u root -p < backup_total.sql

¿Prefieres que verifiquemos el podman inspect de este MySQL antes de que hagas el movimiento para

Jordi H.

```
Has dicho ahora te paso el podman inspect [root@ymir dades]# podman inspect MysqlDB [  {
    "Id": "ef47fb2701bfc305c1fd7030311187435e6d27f4ba5b6bd14342ffdd350996b"
  ,    "Created": "2025-02-24T16:39:41.324384906+01:00",    "Path": "docker-entrypoint.sh
  ,    "Args": [    "mysqld"    ],    "State": {    "OciVersio
n": "1.1.0+dev",    "Status": "running",    "Running": true,    "Paus
ed": false,    "Restarting": false,    "OOMKilled": false,    "Dead": f
alse,    "Pid": 4138,    "ConmonPid": 3959,    "ExitCod
e": 0,    "Error": "container is stopped",    "Star
tedAt": "2026-03-10T05:41:40.864964023+01:00",    "Fini
shedAt": "2026-03-10T05:39:37.692720564+01:00",    "Health": {    "Sta
tus": "",    "FailingStreak": 0,    "Log": null
    },    "C
groupPath": "/machine.slice/libpod-ef47fb2701bfc305c1fd7030311187435e6d27f4ba5b6bd14342ffdd350
996b.scope",    "CheckpointedAt": "0001-01-01T00:00:00Z",
"RestoredAt": "0001-01-01T00:00:00Z"    },
    "Image": "5568fddd4f66008ebae555bfee3e999a8f8e6516fbe974e0af11013e83e241fd",
    "ImageDigest": "sha256:146682692a3aa409eae7b7dc6a30f637c6cb49b6ca901c2cd1
60becc81127d3b",    "ImageName": "docker.io/library/mysql:latest",    "Rootfs": "",
    "Pod": "",    "ResolvConfPath": "/dades/run/containers/storage/overlay-containers/ef47fb270
1bfc305c1fd7030311187435e6d27f4ba5b6bd14342ffdd350996b/userdata/resolv.conf",    "HostnamePath
": "/dades/run/containers/storage/overlay-containers/ef47fb2701bfc305c1fd7030311187435e6d27f4ba5b6bd1
4342ffdd350996b/userdata/hostname",    "HostsPath": "/dades/run/containers/storage/overlay-containe
rs/ef47fb2701bfc305c1fd7030311187435e6d27f4ba5b6bd14342ffdd350996b/userdata/hosts",    "StaticDir
": "/dades/var/lib/containers/storage/overlay-containers/ef47fb2701bfc305c1fd7030311187435e6d27f4ba5b
```

```

6bd14342ffdd350996b/userdata",      "OCIConfigPath": "/dades/var/lib/containers/storage/overlay-contain
ers/ef47fb2701bfc305c1fd7030311187435e6d27f4ba5b6bd14342ffdd350996b/userdata/ config.json",
  "OCIRuntime": "runc",      "ConmonPidFile": "/dades/run/containers/storage/overlay-containers/ef4
7fb2701bfc305c1fd7030311187435e6d27f4ba5b6bd14342ffdd350996b/userdata/conm on.pid",      "PidFile
": "/dades/run/containers/storage/overlay-containers/ef47fb2701bfc305c1fd7030311187435e6d27f4ba5b6bd1
4342ffdd350996b/userdata/pidfile",      "Name": "MySQLDB",      "RestartCount": 0,      "
Driver": "overlay",      "MountLabel": "",      "ProcessLabel": "",      "A
ppArmorProfile": "",      "EffectiveCaps": [      "CAP_CHOWN",
  "CAP_DAC_OVERRIDE",      "CAP_FOWNER",      "CAP_FSETID",
    "CAP_KILL",      "CAP_NET_BIND_SERVICE",
    "CAP_NET_RAW",      "CAP_SETFCAP",      "CAP_SETGID",
    "CAP_SETPCAP",      "CAP_SETUID",      "CA
P_SYS_CHROOT"      ],      "BoundingCaps": [      "CAP_CHOWN",
    "CAP_DAC_OVERRIDE",      "CAP_FOWNER",      "CAP_FSETID"
,      "CAP_KILL",      "CAP_NET_BIND_SERVICE",
    "CAP_NET_RAW",      "CAP_SETFCAP",      "CAP_SETGID",
    "CAP_SETPCAP",      "CAP_SETUID",      "CAP_SYS_CHROOT"
  ],      "ExecIDs": [],      "GraphDriver": {      "Name": "ov
erlay",      "Data": {      "LowerDir": "/dades/var/lib/containers/storage/over
lay/31a9b82977214a3734aae50f548896e01c557c88a2e44c671b6507d1dc9be7eb/diff:/dades/var /lib/containers/s
torage/overlay/14407f3299e69c923759e01843468e44da30250f5cf76560272ec761c5397893/diff:/dades/var/lib/co
ntainers/storage/overlay/25c3b9bc92c 1bc603c72e58297092003571593cd29e7f88f619472c96be0c1db/diff:/dades
/var/lib/containers/storage/overlay/e9625706bda28c234feea49d4fee0d822cf375df112c1943f6fa0f 099a8e681e/di
ff:/dades/var/lib/containers/storage/overlay/10015cfe503d107b0e0c0a64fdbd59af85971ca00eabe79d0b19d85
81e10fa25/diff:/dades/var/lib/containers/s torage/overlay/00e2c2ead5c53982590046a2a32bea91c77c23d4f4
8bf0a218bbc8f7d78cac83/diff:/dades/var/lib/containers/storage/overlay/ddf27df0ca8bbc2b85092475f8c1 e7d
0f0d55bbd1f01fd661c3fe5025ad7a850/diff:/dades/var/lib/containers/storage/overlay/01239645028a2345b3f39
ed820492b67d981313f08c1bb48959ef61d5975e75c/diff:/ dades/var/lib/containers/storage/overlay/edd4dd6b39
c97542e8d93be1f5ace38520260d4ca174d4432f032d1744509a17/diff:/dades/var/l
ib/containers/storage/overlay/d7 b2257a2277cf62be373f5e32a77274c67cdc
db474af84f2f01b95d8f6ebd0d/diff",      "MergedDir": "/dades/var/lib/containers/storage/ov
erlay/52c11f92408ceaf1cc3ba87572a0be425779b04172a04b2177e9121f26055f97/merged",      "
UpperDir": "/dades/var/lib/containers/storage/overlay/52c11f92408ceaf1cc3ba87572a0be425779b
04172a04b2177e9121f26055f97/diff",      "WorkDir": "/dades/var/lib/containers/storage/ove
rlay/52c11f92408ceaf1cc3ba87572a0be425779b04172a04b2177e9121f26055f97/work"
}      },      "Mounts": [      {      "Type": "bind",
    "Source": "/dades/var/lib/mysql",      "Destination": "/var/lib/mys
ql",      "Driver": "",      "Mode": "",      "Optio
ns": [      "rbind"      ],      "RW": true,
    "Propagation": "rprivate"      }      ],      "Dependen
cies": [],      "NetworkSettings": {      "EndpointID": "",      "Gateway"
: "10.88.0.1",      "IPAddress": "10.88.0.2",      "IPPrefix
Len": 16,      "IPv6Gateway": "",      "GlobalIPv6Address":
"",      "GlobalIPv6PrefixLen": 0,      "MacAddress": "ca:24:68:71:f8:d0",
    "Bridge": "",      "SandboxID": "",      "HairpinMode": fal
se,      "LinkLocalIPv6Address": "",      "LinkLocalIPv6PrefixLen": 0,
    "Ports": {      "3306/tcp": [
      {      "HostIp": "",      "HostPort": "8001"
      }      ],      "33060/tcp":
null      },      "SandboxKey": "/run/netns/netns-d30d56bc-4aa6-29
e5-28be-f4d4d81dfac7",      "Networks": {      "p
odman": {      "EndpointID": "",      "Gateway": "10.88.0.1
",      "IPAddress": "10.88.0.2",      "IPPrefi

```

```

xLen": 16,
"IPv6Gateway": "",
"GlobalIPv6Address":
"",
"GlobalIPv6PrefixLen": 0,
"MacAddress": "c
a:24:68:71:f8:d0",
"NetworkID": "podman",
"Driver
Opts": null,
"IPAMConfig": null,
"Links": null,
"Aliases": [
"ef
47fb2701bf"
]
},
"Names
pace": "",
"IsInfra": false,
"IsService": false,
"KubeExitCodePropagation
": "invalid",
"lockNumber": 1,
"Config": {
"Hostname": "ef47fb2701b
f",
"Domainname": "",
"User": "",
"AttachStdin": false,
"AttachStdout": false,
"AttachStderr": false,
"Tty": true,
"OpenStdin": false,
"StdinOnce": fals
e,
"Env": [
"container=podman",
"GOSU_VERSION=1.17",
"MYSQL_
MAJOR=innovation",
"MYSQL_VERSION=9.2.0-1.el9",
"MYSQL_SHELL_VERSION=9.2.0-1.el9",
"TERM=xterm",
"MYSQL_ROOT_PASSWORD=Mysqlplex.00",
"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
"HO
ME=/root",
"HOSTNAME=ef47fb2701bf"
],
"Cmd"
: [
"mysqld"
],
"Image": "docker.io/library/mysql:
latest",
"Volumes": null,
"WorkingDir": "/",
"Entrypoint": "dock
er-entrypoint.sh",
"OnBuild": null,
"Labels": null,
"Annotations": {
"io.container.manager": "libpod",
"org.opencontainers.image.stopSignal": "15"
},
"StopSignal": 15,
"HealthcheckOnFailureAction": "none",
"Umask": "0022",
"Timeout": 0,
"StopTimeout": 10,
"Pas
swd": true,
"ExposedPorts": {
"3306/tcp": {},
"33060/tcp": {}
},
"HostConfig": {
"Bind
s": [
"/dades/var/lib/mysql:/var/lib/mysql:rw,rprivate,rbind"
],
"CgroupManager": "systemd",
"CgroupMode": "host",
"Contain
erIDFile": "",
"LogConfig": {
"Type": "k8s-file",
"Conf
ig": null,
"Path": "/dades/var/lib/containers/storage/overlay-containers/e
f47fb2701bfc305c1fd7030311187435e6d27f4ba5b6bd14342ffdd350996b/userdata /ctr.
log",
"Tag": "",
"Size": "0B"
},
"Net
workMode": "bridge",
"PortBindings": {
"3306/tcp": [
{
"HostIp": "",
"HostPort": "8001"
}
]
},
"RestartPolicy": {
"Name": "",
"MaximumRetryCount": 0
},
"AutoRemove": false,
"VolumeDriver": "",
"VolumesFrom": null,
"CapAdd"
: [],
"CapDrop": [],
"Dns": [],
"DnsOptions": [],
"DnsSearch": [],
"ExtraHosts": [],
"GroupAdd": [],
"IpcMode": "shareable",
"Cgroup": "",
"Cgroups": "default",
"Links": null,
"OomScoreAdj": 0,
"PidMode": "private",
"Privileged": false,
"PublishAllPorts": false,
"ReadOnlyRootfs": false,
"SecurityOpt": [],
"Tmpfs": {},
"UTSMode": "private",
"UsersMode": "",
"ShmSize": 65536000,
"Runtime":
"oci",
"ConsoleSize": [
0,
0
],
"Isolation": "",
"CpuShares": 0,
"Memory
": 0,
"NanoCpus": 0,
"CgroupParent": "",
"BlkioWeight": 0,
"BlkioWeightDevice": null,
"BlkioDevic
eReadBps": null,
"BlkioDeviceWriteBps": null,
"BlkioDeviceReadIOps": null,
"BlkioDeviceWriteIOps": null,
"CpuPeriod": 0,
"CpuQuota": 0,
"CpuRealtimePeriod": 0,
"CpuRealtime

```

```
Runtime": 0, "CpusetCpus": "", "CpusetMems": "", "Devices"
: [], "DiskQuota": 0, "KernelMemory": 0
, "MemoryReservation": 0, "MemorySwap": 0, "MemorySwappi
ness": 0, "OomKillDisable": false, "Pids
Limit": 2048, "Ulimits": [ {
"Name": "RLIMIT_NPROC", "Soft": 4194304, "H
ard": 4194304 } ], "CpuCo
unt": 0, "CpuPercent": 0,
```

Gemini

Confirmado: estás corriendo la versión **9.2.0 (Innovation)**. Si tu objetivo es la analítica de datos, bajar a la **8.0 o S**** es una decisión sabia por estabilidad y compatibilidad con librerías de Python más antiguas. Aquí tienes la hoja de ruta específica para este "downgrade" y la corrección de IP:

1. Análisis del `inspect` (MySQLDB)

- Puerto: 8001:3306 (Actualmente abierto a todo el mundo).
- Volumen: /dades/var/lib/mysql mapeado a /var/lib/mysql.
- Password: Mysqlplex.00 (la usaremos para el dump).
- Imagen actual: docker.io/library/mysql:latest (que resultó ser la 9.2.0).

2. El proceso de Downgrade (Paso a paso)

Como comentamos, MySQL 8 no entenderá los archivos de la 9. Hagámoslo bien:

Paso A: Dump de los datos (Vital)

Con el contenedor encendido: Bash

CODE

```
podman exec MysqlDB mysqldump -u root -p'Mysqlplex.00' --all-databases > /dades/backup_mysql_v9.sql
```

Paso B: Limpieza y Backup de carpetas

Bash

CODE

```
systemctl stop container-MysqlDB.service
podman rm MysqlDB

# Movemos los datos de la v9 para que no estorben pero no borrarlos aún
mv /dades/var/lib/mysql /dades/var/lib/mysql_v9_old
mkdir -p /dades/var/lib/mysql
# Asegurar permisos (MySQL suele usar UID 999 o 1000 dependiendo de la imagen)
chown -R 999:999 /dades/var/lib/mysql
```

Paso C: Lanzar MySQL 8 con la IP corregida

Sustituye 192.168.1.XX por tu IP: Bash

CODE

```
podman run -d \
--name MysqlDB \
-p 192.168.1.XX:8001:3306 \
-v /dades/var/lib/mysql:/var/lib/mysql:rw,z \
-e MYSQL_ROOT_PASSWORD=Mysqlplex.00 \
docker.io/library/mysql:8.0
```

Paso D: Restaurar los datos

Espera unos 20 segundos a que MySQL 8 inicialice su estructura vacía y luego: Bash

CODE

```
podman exec -i MysqlDB mysql -u root -p'Mysqlplex.00' < /dades/backup_mysql_v9.sql
```

3. Consideración sobre la imagen

En el comando te he puesto mysql:8.0. Si quieres ser extremadamente preciso y evitar cualquier actualización automática en el futuro, podrías usar el **ID de imagen** o la versión minor específica (ej: mysql:8.0.36)

4. ¿Qué sigue?

Una vez que restaures el SQL, el contenedor MysqlDB tendrá exactamente la misma estructura que el anterior pero bajo el motor de la versión 8 y escuchando solo en tu IP. El servicio de systemd (podman start MysqlDB) seguirá funcionando perfectamente. **¿Te queda algún otro contenedor en la lista o quieres que verifiquemos si**

Jordi H.

Has dicho eso que has hecho de preservar el nombre de volumen, no me hace falta. puedo dejar los viejos donde estan y crear uno nuevo en /dades/var/lib/mysql8/ ? ya puestos, cual es la imagen de mysql8 (me ha s dicho que es LTS) más nueva que me interesa ? el downgrade se hace por cambio en la estructura del journa l. las herramientas de CDC que uso no soportan el

Gemini

Es una decisión muy inteligente. Si tus herramientas de **CDC** (Change Data Capture) como Debezium o similares no soportan el motor de MySQL 9, quedarte en la versión 8 es obligatorio, ya que el binlog y l a estructura de metadatos han cambiado significativamente. Separar las rutas es lo más limpio: mantienes el "cadáver" de la v9 intacto por si algo sale mal en el volcado de dat

1. La mejor imagen de MySQL 8 (LTS)

Actualmente, dentro de la rama 8, tienes dos opciones:

- MySQL 8.0.x: Es la rama clásica. La versión más estable y pulida es la 8.0.40 (o la última que haya en el repositorio)
- MySQL 8.4 LTS: Es la nueva versión Long Term Support oficial de Oracle. Es el sucesor directo de la 8.0 y mantiene la compatibilidad que buscas para CDC, pero con soporte extendido hasta 2032

Mi recomendación: Usa **mysql:8.4**. Es el estándar de oro actual para entornos que necesitan estabilidad a largo plazo

2. Preparación de la nueva ruta

Vamos a crear el directorio específico para la v8:Bash

CODE

```
mkdir -p /dades/var/lib/mysql8
# MySQL usa el UID 999 en sus imágenes oficiales para el usuario 'mysql'
chown -R 999:999 /dades/var/lib/mysql8
```

3. El nuevo comando `podman run` (v8.4 + IP fija + Nueva Ruta)

Sustituye 192.168.1.XX por la IP que decidiste para que Cockpit deje de mostrar 0.0.0.0.Bash

CODE

```
podman run -d \
--name MysqlDB \
-p 192.168.1.XX:8001:3306 \
-v /dades/var/lib/mysql8:/var/lib/mysql:rw,z \
-e MYSQL_ROOT_PASSWORD=Mysqlplex.00 \
docker.io/library/mysql:8.4
```

4. Ejecución del plan de rescate

Puesto que vas a cambiar la ruta, el proceso es:

1. Sacas el dump del contenedor v9 actual (como hicimos antes).
2. Detén y elimina el contenedor v9: `podman stop MysqlDB && podman rm MysqlDB`.
3. Lanza el comando run de arriba (apuntando a la carpeta /mysql8).
4. Restaura el dump:Bash

CODE

```
# Espera a que el log diga "ready for connections"
podman logs -f MysqlDB
# Luego importa
podman exec -i MysqlDB mysql -u root -p'Mysqlplex.00' < /dades/backup_mysql_v9.sql
```

¿Qué pasa con el CDC?

Al usar la versión **8.4**, asegúrate de que tu configuración de MySQL (normalmente en un archivo .cnf que podrías mapear) tenga el server-id y el log-bin activados si tu herramienta de CDC lee directamente el binlog.Ini, TO

CODE

```
# Ejemplo de lo que podrías necesitar en un archivo my.cnf
[mysqld]
server-id = 1
log-bin = mysql-bin
binlog_format = ROW
binlog_row_image = FULL
```

¿Quieres que te ayude a crear un archivo de configuración mínimo para que el CDC funcione a la primera en este nuevo contenedor? En Podman lo mapearías fácilmente con otro -v

Jordi H.

Has dicho voy a priorizar arreglar los otros contenedores. me queda el registry de imagenes de contenedores desarrollados por nosotros

Gemini

¡Perfecto! Priorizar la limpieza de los puertos en el resto de la infraestructura antes de meterse en el berenjenal del downgrade de MySQL es una decisión muy pragmática. El **Registry** es una pieza crítica porque, si falla o cambia de IP sin avisar, tus despliegues de Python o analítica se detendrán al no poder hacer pull de vuestras propias imágenes

1. Análisis del Registry

Normalmente, el Registry oficial de Docker (docker.io/library/registry:2) es muy ligero pero tiene dos puntos donde solemos "pecar" al configurarlo rápido desde Cockpit

1. Puerto: Suele usar el 5000:5000.
2. Almacenamiento: Si no mapeaste un volumen, todas vuestras imágenes desaparecerán si borras el contenedor

2. Identificación (El `inspect` rápido)

Antes de que lances el comando, localiza estos puntos en tu podman inspect <nombre_del_registry>:

- IP actual: Verás que HostIP está vacío (lo que causa el 0.0.0.0).
- Mounts: Busca dónde se guardan las imágenes. La ruta interna suele ser /var/lib/registry.

3. El comando de corrección (IP fija + Persistencia)

Suponiendo que vuestro registro se llama container-registry, el comando para blindarlo a una sola IP (por ejemplo, la IP interna de vuestro servidor de desarrollo 192.168.1.XX) sería: Bas

CODE

```
podman run -d \
--name container-registry \
-p 192.168.1.XX:5000:5000 \
-v /dades/container_registry_data:/var/lib/registry:rw,z \
--restart always \
docker.io/library/registry:2
```

4. ¿Por qué es vital la IP fija aquí?

A diferencia de la base de datos, el Registry suele usarse como parte de una URL. Si en tus scripts de Python o en otros nodos usáis 192.168.1.XX:5000/mi-imagen-analisis:latest, al restringir la IP en Podman te aseguras de que

1. Seguridad: El registro no sea visible desde internet (si el servidor tiene IP pública).
2. Conflictos: Evitas que Cockpit intente asignar ese puerto a otra interfaz de red virtual que pueda crear Podman automáticamente

5. Pasos para el "Cambio"

Si ya tienes imágenes subidas y **no quieres perderlas**:

1. Verifica en el inspect actual si hay un Source en Mounts.
2. Si lo hay, usa esa misma ruta en el nuevo -v.

3. Si no hay mount, ¡Cuidado! Tendrías que copiar los datos del contenedor antes de borrarlo:
podman cp container-registry:/var/lib/registry /dades/container_registry_data

Siguientes pasos

Una vez arreglado el Registry:

- ¿Te queda algún contenedor de herramientas de visualización (tipo Grafana, Superset o Metabase) que suelas usar para la analítica
- ¿O quizás algún proxy inverso (Nginx/Traefik) que esté intentando gestionar todos estos puertos?

¿Quieres que verifiquemos el inspect del Registry para asegurar que no perdéis vuestras imágenes de

Jordi H.

```
Has dicho [root@ymir ~]# podman inspect registry [ {      "Id":
"c59e66fd88ef6cca149b787e75aa251db158de22d261534ed8e97b7bcfd79bac",      "Created"
: "2025-05-21T16:07:11.55712429+02:00",      "Path": "/entrypoint.sh",      "Args":
[      "/etc/distribution/config.yml"      ],      "State": {      "OciVersio
n": "1.1.0+dev",      "Status": "running",      "Running": true,      "Paus
ed": false,      "Restarting": false,      "OOMKilled": false,      "Dead": f
alse,      "Pid": 5183,      "ConmonPid": 5068,      "ExitCod
e": 0,      "Error": "container is stopped",      "Star
tedAt": "2026-03-10T05:41:41.591391811+01:00",      "Finis
hedAt": "2026-03-10T05:39:34.465740119+01:00",      "Health": {      "Sta
tus": "",      "FailingStreak": 0,      "Log": null
      },      "C
groupPath": "/machine.slice/libpod-c59e66fd88ef6cca149b787e75aa251db158de22d261534ed8e97b7bcfd7
9bac.scope",      "CheckpointedAt": "0001-01-01T00:00:00Z",
"RestoredAt": "0001-01-01T00:00:00Z"      },
      "Image": "3dec7d02aaeabc6faad1421be2570b4666be359853098ac7836ad0aa51b52d79",
      "ImageDigest": "sha256:1fc7de654f2ac1247f0b67e8a459e273b0993be7d2beda1f3f56fbf1001ed3e7",
      "ImageName": "docker.io/library/registry:latest",      "Rootfs": "",      "Pod": "",      "Resol
vConfPath": "/dades/run/containers/storage/overlay-containers/c59e66fd88ef6cca149b787e75aa251db158d
e22d261534ed8e97b7bcfd79bac/userdata/resolv.conf",      "HostnamePath": "/dades/run/containers/storage/
overlay-containers/c59e66fd88ef6cca149b787e75aa251db158de22d261534ed8e97b7bcfd79bac/us
erdata/hostname",      "HostsPath": "/dades/run/containers/storage/overlay-containers/c59e66fd88ef6
cca149b787e75aa251db158de22d261534ed8e97b7bcfd79bac/userdata/hosts",      "StaticDir": "/dades/var/li
b/containers/storage/overlay-containers/c59e66fd88ef6cca149b787e75aa251db158de22d261534ed8e97b7bc
fd79bac/userdata",      "OCIConfigPath": "/dades/var/lib/containers/storage/overlay-containers/c59e66f
d88ef6cca149b787e75aa251db158de22d261534ed8e97b7bcfd79bac/userdata/config.json",      "O
CIRuntime": "runc",      "ConmonPidFile": "/dades/run/containers/storage/overlay-containers/c59e66f
d88ef6cca149b787e75aa251db158de22d261534ed8e97b7bcfd79bac/userdata/conmon.pid",      "PidFile": "
/dades/run/containers/storage/overlay-containers/c59e66fd88ef6cca149b787e75aa251db158de22d261534ed
8e97b7bcfd79bac/userdata/pidfile",      "Name": "registry",      "RestartCount": 0,
"Driver": "overlay",      "MountLabel": "",      "ProcessLabel": "",      "
AppArmorProfile": "",      "EffectiveCaps": [      "CAP_CHOWN",
      "CAP_DAC_OVERRIDE",      "CAP_FOWNER",      "CAP_FSETID",
      "CAP_KILL",      "CAP_NET_BIND_SERVICE",
      "CAP_NET_RAW",      "CAP_SETFCAP",      "CAP_SETGID",
      "CAP_SETPCAP",      "CAP_SETUID",      "C
AP_SYS_CHROOT"      ],      "BoundingCaps": [      "CAP_CHOWN",
      "CAP_DAC_OVERRIDE",      "CAP_FOWNER",      "CAP_FSETID
```

```

",      "CAP_KILL",      "CAP_NET_BIND_SERVICE",
      "CAP_NET_RAW",      "CAP_SETFCAP",      "CAP_SETGID"
,      "CAP_SETPCAP",      "CAP_SETUID",      "CAP_SYS_CHROOT"
],      "ExecIDs": [],      "GraphDriver": {      "Name": "o
verlay",      "Data": {      "LowerDir": "/dades/var/lib/containers/storage/over
lay/ae840580fdd2872eba4e999300f674d1138c70e93b783d3401a78c2294ef8d37/diff:/dades/var/lib/containers/sto
rage/overlay/77d5681974c004a9f4 194807c9a291c7e07370340dd766dfa94b19e75e93854d/diff:/dades/var/lib/con
tainers/storage/overlay/d60241f688163c54cc8f412341a59038b62ad81cd2fb96db508f2c0b01232c5a/diff:/dades/va
r/lib/containers/storage/overlay /815e3270705de4d211645c3871c177edf9fcafd24d742de85cceb1a2dc1c9387/di
ff:/dades/var/lib/containers/storage/overlay/08000c18d16dadf9553d747a58cf44023423a9
ab010aab96cf263d2216b8b350/diff",      "MergedDir": "/dades/var/lib/containers/storage/o
verlay/369dfcbcd073e52b2ed4379e308ea1caf7fb0ad3ce07d64675c29b9b0e7c20d0d/merged",      "
UpperDir": "/dades/var/lib/containers/storage/overlay/369dfcbcd073e52b2ed4379e308ea1caf7fb0a
d3ce07d64675c29b9b0e7c20d0d/diff",      "WorkDir": "/dades/var/lib/containers/storage/ov
erlay/369dfcbcd073e52b2ed4379e308ea1caf7fb0ad3ce07d64675c29b9b0e7c20d0d/work"
}      },      "Mounts": [      {      "Type":
"volume",      "Name": "187136a51cd39b0aa0024ad5339ec
14d87837f0e16d15025dc03586a1afe222e",      "Source": "/dades/var/lib/containers/storage
/volumes/187136a51cd39b0aa0024ad5339ec14d87837f0e16d15025dc03586a1afe222e/_data",
      "Destination": "/var/lib/registry",      "Driver": "local",      "Mod
e": "",      "Options": [
"nodev",      "exec",      "nosuid",
      "rbind"      ],      "RW": true,
      "Propagation": "rprivate"      }      ],      "Dependencies": [],      "NetworkS
ettings": {      "EndpointID": "",      "Gateway": "10.88.0
.1",      "IPAddress": "10.88.0.4",      "IPPrefixLen": 16,      "IPv6Gat
eway": "",      "GlobalIPv6Address": "",      "GlobalIPv6PrefixLen": 0,
      "MacAddress": "12:3a:3d:98:64:02",      "Bridge": "",      "Sand
boxID": "",      "HairpinMode": false,      "LinkLocalIPv6Address"
: "",      "LinkLocalIPv6PrefixLen": 0,      "Ports": {
      "5000/tcp": [      {      "HostIp": "
",      "HostPort": "8012"      }
]      },      "SandboxKey": "/run/netns/netns-91dc2fbe-e4b4-
82bc-1448-66e7a43ee28d",      "Networks": {
"podman": {      "EndpointID": "",      "Gateway": "10.88.0
.1",      "IPAddress": "10.88.0.4",      "IPPre
fixLen": 16,      "IPv6Gateway": "",      "GlobalIPv6Address
": "",      "GlobalIPv6PrefixLen": 0,      "MacAddress":
"12:3a:3d:98:64:02",      "NetworkID": "podman",      "Driv
erOpts": null,      "IPAMConfig": null,
      "Links": null,      "Aliases": [      "
c59e66fd88ef"      ]      }      },      "Nam
espace": "",      "IsInfra": false,      "IsService": false,      "KubeExitCodePropagati
on": "invalid",      "lockNumber": 5,      "Config": {      "Hostname": "c59e66fd8
8ef",      "Domainname": "",      "User": "",      "AttachStdin": false,
      "AttachStdout": false,      "AttachStderr": false,      "Tty": true,
      "OpenStdin": false,      "StdinOnce": false,      "Env": [      "P
ATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
      "container=podman",      "TERM=xterm",      "HOME
=/root",      "HOSTNAME=c59e66fd88ef"      ],      "Cmd":
[      "/etc/distribution/config.yml"      ],      "Image
": "docker.io/library/registry:latest",      "Volumes": null,      "WorkingDir": "/",
      "Entrypoint": "/entrypoint.sh",      "OnBu

```

```

  "Labels": null,
  "Annotations": {
    "io.container.manager": "libpod",
    "org.opencontainers.image.stopSignal": "15"
  },
  "StopSignal": 15,
  "HealthcheckOnFailureAction": "none",
  "Umask": "0022",
  "Timeout": 0,
  "StopTimeout": 10,
  "Paswd": true,
  "ExposedPorts": {
    "5000/tcp": {}
  },
  "HostConfig": {
    "Binds": [
      "187136a51cd39b0aa0024ad5339ec14d87837f0e16d15025dc03586a1afe222e:/var/lib/registry:rprivate,rw,nodev,exec,nosuid,rbind"
    ],
    "CgroupManager": "systemd",
    "CgroupMode": "host",
    "ContainerIDFile": "",
    "LogConfig": {
      "Type": "k8s-file",
      "Config": null,
      "Path": "/dades/var/lib/containerstorage/overlay-containers/c59e66fd88ef6cca149b787e75aa251db158de22d261534ed8e97b7bcfd79bac/userdata/ctr.log",
      "Tag": "",
      "Size": "0B"
    },
    "NetworkMode": "bridge",
    "PortBindings": {
      "5000/tcp": [
        {
          "HostIp": "",
          "HostPort": "8012"
        }
      ]
    },
    "RestartPolicy": {
      "Name": "",
      "MaximumRetryCount": 0
    },
    "AutoRemove": false,
    "VolumeDriver": "",
    "VolumesFrom": null,
    "CapAdd": [],
    "CapDrop": [],
    "Dns": [],
    "DnsOptions": [],
    "DnsSearch": [],
    "ExtraHosts": [],
    "GroupAdd": [],
    "IpcMode": "shareable",
    "Cgroup": "",
    "Cgroups": "default",
    "Links": null,
    "OomScoreAdj": 0,
    "PidMode": "private",
    "Privileged": false,
    "PublishAllPorts": false,
    "ReadOnlyRootfs": false,
    "SecurityOpt": [],
    "Tmpfs": {},
    "UTSMode": "private",
    "UsernsMode": "",
    "ShmSize": 65536000,
    "Runtime": "ocicri",
    "ConsoleSize": [
      0, 0
    ],
    "Isolation": "",
    "CpuShares": 0,
    "Memory": 0,
    "NanoCpus": 0,
    "CgroupParent": "",
    "BlkioWeight": 0,
    "BlkioWeightDevice": null,
    "BlkioDeviceReadBps": null,
    "BlkioDeviceWriteBps": null,
    "BlkioDeviceReadIOps": null,
    "BlkioDeviceWriteIOps": null,
    "CpuPeriod": 0,
    "CpuQuota": 0,
    "CpuRealtimePeriod": 0,
    "CpuRealtimeRuntime": 0,
    "CpusetCpus": "",
    "CpusetMems": "",
    "Devices": [],
    "DiskQuota": 0,
    "KernelMemory": 0,
    "MemoryReservation": 0,
    "MemorySwap": 0,
    "MemorySwappiness": 0,
    "OomKillDisable": false,
    "PidsLimit": 2048,
    "Ulimits": [
      {
        "Name": "RLIMIT_NPROC",
        "Soft": 4194304,
        "Hard": 4194304
      }
    ],
    "CpuCount": 0,
    "CpuPercent": 0,

```

Gemini

Analizando el inspect del **Registry**, he detectado un detalle crítico: actualmente tus imágenes se están guardando en un **volumen anónimo** de Podman (187136a51cd3...). Si borramos el contenedor sin "atar" ese volumen a una ruta real en tu disco /dades, corres el riesgo de que, en una limpieza de Podman (podman volume prune), pierdas todo el registro de imágenes. Aquí tienes el plan para estabilizar

1. Análisis del Registry actual

- Puerto: 8012 en el host mapeado al 5000 interno.

- Imagen: registry:latest (ID 3dec7d02aaea).
- Almacenamiento: /var/lib/registry.

2. El comando de recreación (Seguro y con IP fija)

Vamos a sacar los datos de ese volumen críptico y ponerlos en una carpeta clara en tu disco duro para que nunca más dependas de IDs aleatorios.**Paso A: Crear la ruta física**

CODE

```
mkdir -p /dades/container_registry
```

Paso B: Mover los datos (opcional pero recomendado)

Si quieres conservar las imágenes que ya tienes subidas, antes de borrar el contenedor viejo, copia el contenido:**Bas**

CODE

```
podman cp registry:/var/lib/registry/. /dades/container_registry/
chown -R 1000:1000 /dades/container_registry # El registry suele usar el usuario 1000
```

Paso C: Lanzar el nuevo contenedor

Sustituye 192.168.1.XX por la IP fija de tu servidor:**Bash**

CODE

```
podman run -d \
--name registry \
-p 192.168.1.XX:8012:5000 \
-v /dades/container_registry:/var/lib/registry:rw,z \
--restart always \
3dec7d02aaea
```

3. ¿Por qué hemos hecho esto?

1. Cockpit: Ahora la interfaz de Cockpit mostrará la IP específica en lugar de 0.0.0.0, evitando colisiones con otras interfaces
2. Persistencia Real: Tus imágenes ahora viven en /dades/container_registry. Si el servidor explota, tus imágenes están a salvo en tu partición de datos
3. Inmutabilidad: Usamos el ID de imagen 3dec7d02aaea para asegurar que el registro no cambie de versión accidentalmente

4. Actualización del servicio Systemd

Como en los casos anteriores, comprueba si tu servicio simplemente hace un start. Al haber mantenido el nombre --name registry, el servicio de systemd debería levantar el nuevo contenedor sin quejas:**Bas**

CODE

```
systemctl stop container-registry.service
podman rm registry
# Ejecutas el comando 'run' del punto 2
systemctl start container-registry.service
```

Con esto ya tienes **MongoDB, Elasticsearch y el Registry** saneados. Solo te quedaría el "gran movimiento" de MySQL 8 cuando tengas el backup listo. ¿Hay algún otro contenedor "rebelde" que use puertos o ya hemo

¿s cubierto la patrulla principal? Sería un buen momento para revisar si tienes algún **Redis** o **RabbitMQ** que suelen venir configurados "por defecto" con los puertos abiertos

Jordi H.

Has dicho en mi sistema tengo media docena de contenedores, ya sabes. Un elasticsearch, un registry, un mysql y un mongodb. ahora tengo el problema que se están reiniciando solos cada 5 minutos o menos, y además casi a la vez. estoy mirando los logs de cada uno y no encuentro nada llamativo. quisiera mirar los logs del propio podman pero no sé donde están, si existen. al ser un sistema sin daemon no sé dónde mirar

Gemini

Ese patrón de reinicios simultáneos en todos los contenedores cada 5 minutos exactos es una señal clara de que el problema no está dentro de los contenedores, sino en el sistema que los supervisa. Al no haber un demonio (daemon) central como en Docker, el culpable suele ser el "orquestador" de procesos del sistema: **Systemd**. Aquí tienes la guía de investigación para encontrar esos logs "invisibles"

1. ¿Dónde mirar los logs de Podman?

Como bien dices, Podman no tiene un archivo `/var/log/podman.log`. Los eventos se registran en el **Journal** de systemd para ver qué está pasando con el motor de contenedores y los servicios que los levantan, usa:

- Logs generales de servicios de contenedores: Bash

CODE

```
journalctl -u "container-*" --since "15 minutes ago"
```

(Esto filtrará los logs de todos tus servicios que empiecen por "container-", que es el estándar de Cockpit).

- Logs de eventos de Podman: Bash

CODE

```
podman events --since 15m
```

(Este comando es oro: te mostrará en tiempo real si alguien está enviando una señal de STOP o KILL a los contenedores)

2. Los 3 sospechosos habituales de reinicios en bloque

Sospechoso A: El Healthcheck de Systemd / Cockpit

Cuando creas contenedores desde Cockpit, a menudo se activan **Healthchecks**. Si el chequeo falla (porque el servicio tarda en responder o la IP ha cambiado y el check apunta a la antigua), Systemd aplicará la política de reinicio

- Comprueba el estado del servicio:
`systemctl status container-Mysqldb.service`
- Busca líneas que digan Restarting... o Service hold-off time over.

Sospechoso B: OOM Killer (Falta de Memoria)

Tienes un Elasticsearch con **32GB** y otros tres motores de base de datos. Si el sistema se queda sin RAM, el kernel de Linux empezará a matar procesos para sobrevivir

- Mira los logs del Kernel: Bash

CODE

```
dmesg -T | grep -i "oom"
```

O mira el log del sistema: journalctl -ke (la -e te lleva al final). Si ves "Out of memory: Kill process", ya tienes al culpable

Sospechoso C: Conflictos en la configuración de Red

Al haber estado tocando las IPs y los puertos (0.0.0.0 vs IP fija), es posible que los servicios de Systemd estén entrando en conflicto. Si un servicio intenta levantar un contenedor en un puerto que otro aún no ha soltado de l todo, el segundo fallará, se reiniciará, y provocará una reacción en caden

3. Cómo investigar "en vivo"

Abre dos terminales (o dos pestañas en Cockpit):

1. Terminal 1 (Vigilante):

journalctl -f (para ver qué proceso de sistema da la orden de reinicio).

2. Terminal 2 (Podman):

podman ps -a (para ver el código de salida EXIT de los contenedores).

Nota importante: Si ves un Exit Code 137, significa que algo externo (como el sistema o systemd) les ha enviado un SIGKILL. Si es 143, es un SIGTERM (un apagado ordenado) ¿Qué ves en el journalctl - s las últimas 20 líneas de ese log, podremos ver exactamente quién está apretando el botón de reinicio. Would you like me to help you analyze those logs?

Jordi H.

```
Has dicho de març 18 15:10:24 ymir.uab.cat systemd[1]: container-es01.service: start operation timed out. Terminating. de març 18 15:10:27 ymir.uab.cat podman[123617]: es01 de març 18 15:10:27 ymir.uab.ca t systemd[1]: container-es01.service: Failed with result 'timeout'. de març 18 15:10:27 ymir.uab.cat systemd[1 ]: Failed to start Podman container-es01.service. de març 18 15:10:27 ymir.uab.cat systemd[1]: container-es01.service: Service RestartSec=100ms expired, scheduling restart. de març 18 15:1 0:27 ymir.uab.cat systemd[1]: container-es01.service: Scheduled restart job, restart counter is at 48. de ma rç 18 15:10:27 ymir.uab.cat systemd[1]: Stopped Podman container-es01.service. de març 18 15:10:27 ymir.u ab.cat systemd[1]: Starting Podman container-es01.service... de març 18 15:10:27 ymir.uab.cat dnsmasq[1 23911]: listening on cni-podman2(#11): 10.89.1.1 de març 18 15:10:27 ymir.uab.cat dnsmasq[123911]: list ening on cni-podman2(#11): fe80::445d:31ff:fe26:9b01%cni-podman2 de març 18 15:10:27 ymi r.uab.cat dnsmasq[123915]: started, version 2.79 cachesize 150 de març 18 15:10:27 ymir.uab.cat dnsmas q[123915]: compile time options: IPv6 GNU-getopt DBus no-i18n IDN2 DHCP DHCPv6 no-Lua TFTP no-conn track ipset auth DNSSEC loop-detect inotify de març 18 15:10:27 ymir.uab.cat dnsmasq[123915]: using loc al addresses only for domain dns.podman de març 18 15:10:27 ymir.uab.cat dnsmasq[123915]: reading /etc/r esolv.conf de març 18 15:10:27 ymir.uab.cat dnsmasq[123915]: using local addresses only for domain dns. podman de març 18 15:10:27 ymir.uab.cat dnsmasq[123915]: using nameserver 158.109.0.49#53 de m arç 18 15:10:27 ymir.uab.cat dnsmasq[123915]: using nameserver 158.109.0.50#53 de març 18 15:10 :27 ymir.uab.cat dnsmasq[123915]: using nameserver 158.109.254.139#53 de març 18 15:10:27 ymir.uab.cat dnsmasq[123915]: read /run/containers/cni/dnsname/elastic/addnhosts - 1 addresses de març 18 15: 10:27 ymir.uab.cat podman[123636]: es01 de març 18 15:10:27 ymir.uab.cat systemd[1]: container-es01.serv ice: Can't open PID file /dades/run/containers/storage/overlay-containers/4ab24cbba949bd6b771f91924a2b 4e4ecb51ed83f6990a45a88eced31adff94a/userdata/conmon.pid (yet?) after start: No such file or directory de març 18 15:11:06 ymir.uab.cat systemd[1]: container-registry.service: State 'stop-sigterm' timed out. Killing . de març 18 15:11:06 ymir.uab.cat systemd[1]: container-registry.service: Killing process 122714 (conmon) w
```

```

ith signal SIGKILL. de març 18 15:11:06 ymir.uab.cat podman[12
4196]: Error: container cbccdf357f516e4307a30582cb7e1cf03a6910ca38cfaeedade56586fb0d0453 common
exited prematurely, exit code could not be retrieved: internal libpod error de març 18 15:11:06
ymir.uab.cat systemd[1]: container-registry.service: Control process exited, code=exited status=125 de mar
ç 18 15:11:06 ymir.uab.cat systemd[1]: container-registry.service: Failed with result 'timeout'. de març 18 15:11:
06 ymir.uab.cat systemd[1]: Failed to start Podman container-registry.service. de març 18 15:11:
:07 ymir.uab.cat systemd[1]: container-registry.service: Service RestartSec=100ms expired, scheduling
restart. de març 18 15:11:07 ymir.uab.cat systemd[1]: container-registry.service: Scheduled restart job, restar
t counter is at 29. de març 18 15:11:07 ymir.uab.cat systemd[1]: Stopped Podman container-regist
ry.service. de març 18 15:11:07 ymir.uab.cat systemd[1]: Starting Podman container-registry.service...
de març 18 15:11:07 ymir.uab.cat podman[124377]: registry de març 18 15:11:07 ymir.uab.cat systemd[1]: conta
iner-registry.service: Can't open PID file /dades/run/containers/storage/overlay-containers/c59e66fd88
ef6cca149b787e75aa251db158de22d261534ed8e97b7bcbfd79bac/userdata/common.pid (yet?) after start: No such fil
e or directory de març 18 15:11:23 ymir.uab.cat systemd[1]: container-MysqIDB.service: start operation
timed out. Terminating. de març 18 15:11:24 ymir.uab.cat podman[125007]: MySQLDB de març
18 15:11:24 ymir.uab.cat systemd[1]: container-MysqIDB.service: Failed with result 'timeout'. de març 18 15:11:
24 ymir.uab.cat systemd[1]: Failed to start Podman container-MysqIDB.service. de mar
ç 18 15:11:25 ymir.uab.cat systemd[1]: container-MysqIDB.service: Service RestartSec=100ms expired,
scheduling restart. de març 18 15:11:25 ymir.uab.cat systemd[1]: container-MysqIDB.service: Scheduled restart j
ob, restart counter is at 49. de març 18 15:11:25 ymir.uab.cat systemd[1]: Stopped Podman conta
iner-MysqIDB.service. de març 18 15:11:25 ymir.uab.cat systemd[1]: Starting Podman container-MysqIDB.s
ervice... de març 18 15:11:25 ymir.uab.cat podman[125026]: MySQLDB de març 18 15:11:25 ymir.uab.cat s
ystemd[1]: container-MysqIDB.service: Can't open PID file /dades/run/containers/storage/overlay-containe
rs/ef477b2701bfc305c1fd703031187435e6d27f4ba5b6bd14342ffdd350996b/userdata/common.pid (yet?) after start: N
o such file or directory de març 18 15:11:35 ymir.uab.cat systemd[1]: container-MongoDB.service: start
operation timed out. Terminating. de març 18 15:11:57 ymir.uab.cat systemd[1]: container-es01.service: star
t operation timed out. Terminating. de març 18 15:12:00 ymir.uab.cat podman[125765]: es
01 de març 18 15:12:00 ymir.uab.cat systemd[1]: container-es01.service: Failed with result 'timeout'. de març 18 1
5:12:00 ymir.uab.cat systemd[1]: Failed to start Podman container-es01.service. de març 18 15:12:00 ymir.ua
b.cat systemd[1]: container-es01.service: Service RestartSec=100ms expired, scheduling res
tart. de març 18 15:12:00 ymir.uab.cat systemd[1]: container-es01.service: Scheduled restart job, restart
counter is at 49. de març 18 15:12:00 ymir.uab.cat systemd[1]: Stopped Podman container-es01.service. d
e març 18 15:12:00 ymir.uab.cat systemd[1]: Starting Podman container-es01.service... de març 1
8 15:12:00 ymir.uab.cat dnsmasq[126061]: listening on cni-podman2(#11): 10.89.1.1 de març
18 15:12:00 ymir.uab.cat dnsmasq[126061]: listening on cni-podman2(#11): fe80::445d:31ff
:fe26:9b01%cni-podman2 de març 18 15:12:00 ymir.uab.cat dnsmasq[126065]: started, version 2.79 cachesi
ze 150 de març 18 15:12:00 ymir.uab.cat dnsmasq[126065]: compile time options: IPv6 GNU-getopt DBu
s no-i18n IDN2 DHCP DHCPv6 no-Lua TFTP no-contrack ipset auth DNSSEC loop-detect inotify de març 18 15
:12:00 ymir.uab.cat dnsmasq[126065]: using local addresses only for domain dns.podman de març 18 15:12:0
0 ymir.uab.cat dnsmasq[126065]: reading /etc/resolv.conf de març 18 15:12:00 ymir.uab.cat dnsmasq[12606
5]: using local addresses only for domain dns.podman de març 18 15:12:00 ymir.uab.cat dnsmasq[
126065]: using nameserver 158.109.0.49#53 de març 18 15:12:00 ymir.uab.cat dnsmasq[126065]: usi
ng nameserver 158.109.0.50#53 de març 18 15:12:00 ymir.uab.cat dnsmasq[126065]: using nameserver 158.1
09.254.139#53 de març 18 15:12:00 ymir.uab.cat dnsmasq[126065]: read /run/containers/cni/dnsname/
elastic/addnhosts - 1 addresses de març 18 15:12:00 ymir.uab.cat podman[125785]: es01 de març 18 15:12:0
0 ymir.uab.cat systemd[1]: container-es01.service: Can't open PID file /dades/run/containers/storage/o
verlay-containers/4ab24cbba949bd6b771f91924a2b4e4ecb51ed83f6990a45a88eced31adff94a

```

Gemini

¡Uf! Esos logs son la "pistola humeante". El problema es un **conflicto de tiempos entre systemd y Podman**. Lo que está pasando es un círculo vicioso clásico

1. Systemd arranca el servicio y espera que el contenedor esté listo en X segundos.
 2. Podman/Common tarda demasiado (quizás por el volumen de 32GB de Elasticsearch o la red).
 3. Systemd se impacienta: Como no puede leer el archivo .pid a tiempo, asume que el arranque ha fallado, mata el proceso (Terminating) y, debido a la regla RestartSec=100ms, vuelve a intentarlo o instantáneamente.
 4. Esto ocurre con todos a la vez, saturando el sistema.
- Aquí tienes la solución en tres pasos para romper este bucle:

1. El parche de emergencia: Detén la locura

Primero, para que el servidor deje de intentar arrancar 50 veces por minuto: Bash

CODE

```
systemctl stop "container-"
```

2. Modificar los archivos `.service` (La solución real)

Tienes que editar los archivos de servicio de tus contenedores (probablemente en `/etc/systemd/system/`). El objetivo es darle más tiempo a systemd y relajar los reintentos. Para cada archivo (ej. `container-es01.service`, `container-MysqIDB.service`, etc.), busca y edita estas líneas en la sección `[Service]`

- `TimeoutStartSec=0`: Esto le dice a systemd que no tenga prisa y espere indefinidamente a que el contenedor levante (vital para bases de datos grandes)
- `RestartSec=10s`: Cambia esos 100ms (que son una locura) por 10 segundos. Así, si falla, el sistema respira antes de reintentar
- `StartLimitIntervalSec=0`: (Opcional) Para evitar que el servicio se bloquee por demasiados reintentos.

Ejemplo de cómo debería quedar la sección `[Service]`:Ini, TOML

CODE

```
[Service]
Restart=always
RestartSec=10s
TimeoutStartSec=0
ExecStart=/usr/bin/podman start es01
ExecStop=/usr/bin/podman stop -t 10 es01
Type=forking
PIDFile=/dades/run/containers/.../common.pid
```

3. Aplicar y Resetear

Una vez editados los archivos de los 4 o 5 contenedores:

1. Recarga systemd: Bash

CODE

```
systemctl daemon-reload
```

2. Limpia el estado de fallos: Bash

CODE

```
systemctl reset-failed "container-"
```

3. Arranca uno por uno (no todos a la vez para no saturar el disco): Bash

CODE

```
systemctl start container-es01.service
# Espera un minuto...
systemctl start container-Mysqldb.service
```

¿Por qué fallaba el PID file?

El error Can't open PID file... (yet?) es muy común. Ocurre porque el proceso común de Podman tarda un pelín más en crear el archivo de lo que systemd tarda en comprobarlo. Al poner TimeoutStartSec=0, systemd será mucho más paciente. **Un consejo extra:** Veo en tus logs que dnsmasq se levanta con la red de elastic. Si Elasticsearch tarda mucho en responder debido a su tamaño, es el que suele arrastrar al resto al colapso. ¿Quieres que te ayude con un comando de terminal para editar todos los archivos .service de golpe usando sed y ahorrarte el trabajo manual?

Jordi H.

Has dicho al hacer el systemctl daemon-reload he visto que hay unos cuantos que no se han recargado. debería volver a habilitarlos

Gemini

Es normal que tras editar los archivos a mano y hacer un daemon-reload, algunos servicios se queden en un estado "limbo". No siempre es necesario volver a habilitarlos (enable), pero sí es fundamental **limpiar el historial** y systemd acepte volver a gestionarlos. Aquí tienes los pasos para estabilizarlos sin romper la configuración de arranque:

1. Limpiar el estado de "Failed"

Si un servicio ha fallado muchas veces seguidas (como esos 49 reintentos que veíamos en tu log), systemd lo marca como "fallido" y deja de intentar arrancarlo por protección. Ejecuta esto para resetear el contador: Bas

CODE

```
systemctl reset-failed "container-*.service"
```

2. Verificar si están habilitados (Enabled)

No necesitas hacer un systemctl enable a menos que veas que el enlace simbólico se ha roto. Compruébalo rápido con este comando: Bas

CODE

```
systemctl list-unit-files "container-*.service"
```

- STATE = enabled: Todo bien, arrancarán solos al reiniciar el PC.
- STATE = disabled: Aquí sí, tira un systemctl enable container-nombre.
- STATE = static: Significa que no tienen sección [Install], pero en tu caso deberían ser enabled.

3. El orden de arranque (Estratégico)

Como tienes bases de datos pesadas y un Elasticsearch de **32GB**, lanzarlos todos a la vez tras el daemon-reload puede volver a provocar el timeout. Te recomiendo este orden manual para asegurar que cada uno se cargue correctamente.

a uno tiene su espaci

1. Primero el Registry: Es ligero y no tiene dependencias.

`systemctl start container-registry.service`

2. Segundo MongoDB: Suele levantar rápido.

`systemctl start container-MongoDB.service`

3. Tercero MySQL: (Si ya terminaste el downgrade).

`systemctl start container-MysqIDB.service`

4. Al final Elasticsearch: Es el que más RAM y CPU consume al arrancar. Dale su tiempo.

`systemctl start container-es01.service`

4. ¿Por qué algunos parecen no haberse recargado?

Si después del daemon-reload ves un mensaje tipo "Warning: The unit file, source configuration file or drop-ins of container-xxx.service changed on disk", es simplemente un aviso de que el archivo en memoria es distinto al del disco. Al hacer el `systemctl start` (o `restart`), esa advertencia desaparecerá y usará la nueva configuración con el **Timeout** extendi

Un truco final para tu analítica en Python

Si en el futuro quieres que tus scripts de Python esperen a que las bases de datos estén realmente listas (y no solo que el contenedor haya arrancado), puedes usar este comando en tus scripts de Bash o Python antes de lanzar la analítica:Ba

CODE

```
# Espera a que el puerto 3306 de MySQL esté abierto en la IP fija
until nc -z 192.168.1.XX 3306; do
echo "Esperando a MySQL...";
sleep 2;
done
```

¿Cómo van esos contadores de reinicio? Si haces un `systemctl status container-es01.service`, ¿ya aparece como `active (running)` sin el mensaje de timeout? Sería genial si pudieras confirmarme que Elasticsearch ha aguantado más de 5 minutos encendido. Sería la señal de que hemos ganado la batall

Jordi H.

Has dicho bueno. tenemos el elastic con un time de supervivencia de más de 10 minutos. parece que hemos acertado. una cosita. el comando `systemctl start` no termina. me deja sin prompt en el terminal. he hecho un `ctrl`

Gemini

¡Victoria parcial con esos 10 minutos de Elasticsearch! Es una excelente señal de que el bucle de reinicios se ha roto. Sobre el hecho de que `systemctl start` se quede "colgado" sin devolverte el prompt, es un comportamiento clásico de `systemd` cuando está configurado como **Type=forking** y no recibe la señal adecuada de que el proceso ha terminado de arrancar. Aquí te explico qué está pasando y cómo arreglarlo para que tu terminal vuelva a ser li

1. ¿Por qué se queda colgado?

Al configurar el servicio como `Type=forking`, `systemd` espera tres cosas:

1. Que el comando `ExecStart` se ejecute.

2. Que ese comando cree un "hijo" (el proceso real).

3. Que aparezca el archivo PID especificado en PIDFile.

Si el archivo .pid no aparece o Podman tarda en registrarlo, systemd se queda esperando "eternamente" (o hasta el timeout que pusimos a 0) antes de decirte: "Vale, ya está activo". Al hacer Ctrl+C, cancelas la espera de systemd, pero el proceso de Podman sigue corriendo en segundo plano (por eso Elasticsearch sobrevive

2. La solución: Cambiar a `Type=simple`

Para contenedores modernos de Podman, el tipo forking con PIDFile suele dar más problemas que beneficios. Lo más limpio es usar **Type=simple** (o Type=notify si tu versión de Podman lo soporta bien), ya que delega la gestión del proceso directamente a Podman sin depender de archivos de texto temporales. **Edita tus archivos** así: ****Ini**, **TOML**

CODE

```
[Service]
Type=simple
# Borra o comenta la línea del PIDFile
# PIDFile=/dades/run/containers/.../common.pid
ExecStart=/usr/bin/podman start -a es01
ExecStop=/usr/bin/podman stop -t 10 es01
Restart=always
RestartSec=10s
TimeoutStartSec=0
```

- -a (Attach): En Type=simple, es recomendable añadir -a al podman start. Esto hace que el proceso de systemd esté "atado" a la salida del contenedor, permitiendo que systemd sepa instantáneamente si el contenedor muere

3. Aplicar el cambio final

Una vez editados los archivos de los contenedores:

1. Recarga la configuración: Bash

CODE

```
systemctl daemon-reload
```

2. Reinicia el servicio (esta vez debería devolvarte el prompt rápido): Bash

CODE

```
systemctl restart container-es01.service
```

Resumen de situación

- Elasticsearch: Estable (>10 min).
- Puertos: IP fija configurada (Cockpit debería verse limpio).
- Systemd: Timeout arreglado, solo falta ajustar el Type para recuperar el control de la terminal.

Con esto, tu infraestructura de datos (Mongo, MySQL, ES, Registry) ya no es un "castillo de naipes" que se cae cada 5 minutos. Estás listo para volver a centrarte en la analítica y Python. ¿Te queda algún detalle pendiente con el **downgrade de MySQL** o prefieres dar por finalizada la "operación limpieza" de hoy? Sería un buen momento para hacer un podman ps final y disfrutar de la columna "STATUS" sin el mensaje de "Restarting

Jordi H.

Has dicho al contenedor de mysql no le ha gustado el type simple, lo he revertido a fork y funciona

Gemini

Es totalmente comprensible. Con MySQL (y a veces con bases de datos que tienen scripts de entrada complejos), el Type=forking suele ser más "fiel" a cómo el binario de MySQL gestiona sus propios hilos y el proceso común de Podman. Si funciona y es estable, **no lo toques**. En administración de sistemas, la estabilidad gana a la teoría. El hecho de que no te devuelva el prompt es un "pequeño precio" a pagar por tener un control de PID estricto. Si alguna vez te desespera la espera, recuerda que siempre puedes lanzar el comando en segundo plano o simplemente usar `systemctl start container-Mysqldb &` (aunque lo ideal es dejar que systemd termine su litu

Estado de la Misión: Concluido con Éxito

Haciendo recuento de lo que hemos estabilizado hoy en tu servidor **ymir**:

Contenedor	Cambio Realizado	Estado Final
MongoDB	IP Fija + Volumen Externo	Estable
Elasticsearch	IP Fija + 32GB RAM + Volumen +	Estable (>10 min)
Registry	IP Fija + Volumen Físico (fuera de ID	Estable
Mysqldb	Preparado para Downgrade v8.4	Funcional (Forking)
Systemd	TimeoutStartSec=0 +	Bucle de reinicios roto

Un último consejo de "compañero"

Como vas a centrarte en **Python y Analítica**, ahora que tu infraestructura es sólida, te recomiendo que tu primera tarea sea automatizar un **script de salud (Healthcheck)** sencillo en Python que verifique que estos 4 pilares están vivos. Te ahorrará tener que entrar en Cockpit cada mañana para ver si Elasticsearch sigue ahí. Pyt

CODE

```
import socket

def check_port(ip, port):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        return s.connect_ex((ip, port)) == 0

# Tus nuevas IPs fijas
services = {
    "Mongo": ("192.168.1.XX", 8000),
    "MySQL": ("192.168.1.XX", 8001),
    "Elastic": ("192.168.1.XX", 8008),
    "Registry": ("192.168.1.XX", 8012)
}

for name, (ip, port) in services.items():
    status = "OK" if check_port(ip, port) else "DOWN"
    print(f"{name}: {status}")
```

¿Hay algo más en lo que pueda ayudarte con la configuración de red o prefieres empezar ya con el volcado de datos de MySQL

Jordi H.

Has dicho el sistema está estable y con buen rendimiento. he contado unas 4000 reinicios de contenedores y sus correspondientes logs. me ayudas a limpiar logs

Gemini

¡Menudo festival de reinicios! 4000 entradas en el log es una barbaridad, pero es normal después de ese bucle infinito de 5 minutos que tuvimos. Tener el Journal de systemd inundado no solo es molesto para leer, sino que ocupa un espacio innecesario en disco (especialmente si /var/log/journal está en una partición pequeña). Como Podman vuelca todo al Journal del sistema, vamos a hacer una limpieza quirúrgica

1. Limpiar el Journal de Systemd

Tienes varias formas de "adelgazar" los logs del sistema para deshacerte de esos 4000 registros de error:

- Limpiar por tamaño: Si quieres que el log no ocupe más de, por ejemplo, 100MB: Bash

CODE

```
journalctl --vacuum-size=100M
```

- Limpiar por tiempo: (La más recomendada ahora) Borra todo lo anterior a la última hora, dejando solo los logs de cuando el sistema ya estaba estable: Bash

CODE

```
journalctl --vacuum-time=1h
```

- Limpiar por número de archivos: Conserva solo los últimos 2 archivos de rotación: Bash

CODE

```
journalctl --vacuum-files=2
```

2. Limpiar logs específicos de Podman (Container Logs)

Aunque systemd gestiona el arranque, los contenedores guardan sus propios logs de salida (STDOUT/STDERR) en archivos internos de Podman. Para vaciarlos sin borrar los contenedores: Bash

CODE

```
# Esto vaciará los logs de TODOS tus contenedores actuales
truncate -s 0 $(podman inspect --format '{{.InternalLogPath}}' $(podman ps -aq))
```

3. Limpiar imágenes y contenedores "huérfanos"

Después de tanto podman rm y podman run, es muy probable que tengas capas de imágenes sueltas o volúmenes anónimos (como el que tenía el Registry) ocupando sitio. Para hacer una limpieza general de lo que **no se está usando**: Bash

CODE

```
podman system prune
```

Cuidado: Esto borrará contenedores parados y redes no usadas. Como ahora todos tus servicios están running

y estables, es seguro hacerlo

4. Configurar el límite para el futuro

Para que no vuelva a pasar (aunque ya no tengamos reinicios locos), puedes limitar el tamaño de los logs de los contenedores directamente en la configuración de Podman o en el comando run añadiendo `--log-opt max-size=10mb`

Resumen de comandos para dejarlo impoluto:

1. `journalctl --vacuum-time=10m` (Borras el desastre de los reinicios).
2. `podman system prune -f` (Borras restos de contenedores viejos).
3. `df -h /dades` (Verificas que has recuperado espacio en tu partición de datos).

¿Quieres que echemos un vistazo a cuánta RAM te queda libre después de estabilizar el Elasticsearch de 32GB o ya te pones manos a la obra con tus scripts de analítica

Jordi H.

Has dicho fascinante. no hay nada que purgar en journal [root@ymir ~]# `journalctl --vacuum-time=1h`
Vacuuming done, freed 0B of archived journals from /run/log/journal/fec57d2f8a744dccb77f734a8836c286

Gemini

Vaya, eso tiene una explicación técnica muy sencilla (y un poco traicionera): en sistemas como RHEL, Fedora o CentOS (que suelen ir de la mano con Cockpit), el Journal suele estar configurado como **volátil** por defecto, o simplemente estás mirando el almacenamiento en memoria. Cuando `journalctl` dice que ha liberado 0B, suele ser por una de estas dos razones

1. El almacenamiento es "Runtime" (en RAM)

Si te fijas en la ruta que te ha dado: `/run/log/journal/....`

Todo lo que cuelga de `/run` en Linux es un **tmpfs** (un disco en memoria RAM). Al reiniciar el sistema o los servicios, esos logs desaparecen. Si el sistema ha rotado los archivos hace poco o si el límite de RAM para los logs se ha alcanzado, el propio sistema ya los ha "purgado" para no colapsar la memoria

2. Archivos activos vs. archivados

El comando `--vacuum` solo borra archivos **archivados** (los que ya están cerrados y rotados). Si los 4000 errores han ocurrido hace muy poco, todavía están en el archivo "active", y `journalctl` no los borrará hasta que se sature el espacio o pase más tiempo

¿Cómo ver cuánto espacio ocupan de verdad?

Para salir de dudas y ver si realmente tienes basura acumulada, lanza: `Bash`

CODE

```
journalctl --disk-usage
```

¿Cómo forzar la limpieza si te molestan al leer?

Si quieres "limpiar la pantalla" de errores pasados aunque no liberes espacio en disco, lo que necesitas es un **Rotate** manual antes del Vacuum

1. Fuerza la rotación (cierra el archivo actual y abre uno nuevo): `Bash`

CODE

```
systemctl kill --signal=SIGUSR1 systemd-journald
```

2. Ahora sí, lanza el vacuum: Bash

CODE

```
journalctl --vacuum-time=1s
```

Un último detalle para tu faceta de Data Analyst

Ya que vas a trabajar con **Python**, si alguna vez necesitas analizar esos 4000 errores de forma estructurada (por ejemplo, para ver exactamente a qué segundo fallaba cada uno), puedes exportarlos a JSON desde la terminal: Ba

CODE

```
journalctl -u "container-*" --since "today" -o json | python3 -m json.tool
```

Con el sistema ya estable y los logs bajo control, parece que el servidor **ymir** está listo para entrar en producción. ¿Te gustaría que verifiquemos el uso de RAM ahora que Elasticsearch tiene sus 32GB asignados o prefieres que te deje vía libre para empezar con el código

Jordi H.

Has dicho [root@ymir ~]# truncate -s 0 \$(podman inspect --format='{{.InternalLogPath}}' \$(podman ps -aq))
Error: template: inspect:1:13: executing "inspect" at <.InternalLogPath>: can't evaluate field InternalLogPath in type interface {} truncate: falta un fichero como operando Pruebe 'truncate --help' para más información

Gemini

Vaya, parece que hemos topado con una de esas pequeñas diferencias entre versiones de Podman. El campo `.InternalLogPath` no siempre está disponible directamente en el nivel superior del JSON dependiendo de la versión que maneje tu sistema (especialmente en versiones algo más antiguas o configuraciones específicas de almacenamiento). No te preocupes, vamos a usar una ruta más universal que funciona en el 99% de las instalaciones de Podman modernas para limpiar esos archivos `.log` que están llenando el disco

1. La alternativa infalible (JSON directo)

Prueba con esta variante del comando, que busca en la sección de configuración donde Podman guarda la ruta del log: Bas

CODE

```
truncate -s 0 $(podman inspect --format='{{.LogPath}}' $(podman ps -aq))
```

(He cambiado `.InternalLogPath` por `.LogPath`, que es el estándar más común).

2. Si el comando anterior falla (El "Plan B" manual)

Si por lo que sea el comando sigue dándote guerra con el "template", podemos hacerlo a la vieja usanza combinando `find` con la ruta donde sabemos que Cockpit y Podman guardan los datos en tu servidor: Bas

CODE

```
find /dades/var/lib/containers/storage/overlay-containers/ -name "*.log" -exec truncate -s 0 {} +
```

¿Qué hace este comando?

- Busca en tu partición de datos (/dades) dentro de la carpeta de contenedores.
- Localiza todos los archivos que terminan en .log.
- Ejecuta el truncate para dejarlos a 0 bytes sin borrar el archivo (para que el contenedor no se asuste al intentar escribir en él)

3. Verificación de limpieza

Una vez ejecutado cualquiera de los dos, puedes comprobar si ha funcionado con: Bash

CODE

```
du -sh /dades/var/lib/containers/storage/overlay-containers/*/userdata/*.log
```

Deberías ver que todos ocupan **0** o unos pocos **KB** (si el contenedor ya ha vuelto a escribir algo).

¿Cómo va la salud de ymir?

Con los logs vacíos y el daemon-reload aplicado, el sistema debería estar "fresco". ¿Te gustaría que verifiquemos si el uso de CPU ha bajado ahora que los contenedores no están constantemente escribiendo errores de inicio en el disc

Jordi H.

Has dicho eso ya me doy una idea. 252 procesos que ocupan 0.22 de las 64 cpu

Gemini

Eso es la definición de un sistema "en reposo" saludable. Con un **Load Average de 0.22** para **64 CPUs**, el servidor ni siquiera se está despeinando. Esos 252 procesos son básicamente el ruido de fondo del kernel y los servicios de sistema. Lo más importante es que ese número ya no está bailando locamente debido a los constantes picos de CPU que generaba el bucle de reinicios (que entre el conmon, el chequeo de red y el arranque de Java en Elasticsearch, te habrían tenido el Load mucho más al

Estado Final del Servidor `ymir`

- CPU: Prácticamente ociosa (0.22/64).
- Logs: Vaciados y rotados.
- Contenedores: Estables con IP fija y timeouts corregidos.
- Persistencia: MongoDB, ES y Registry con sus rutas en /dades bien atadas.

Una última comprobación (OOM Killer)

Como tienes **Elasticsearch con 32GB** fijos, solo asegúrate de vez en cuando de que el resto de bases de datos no sufran. Con 64 CPUs, sospecho que tienes bastante RAM (mínimo 64GB o 128GB), pero siempre es bueno vigilar: Ba

CODE

```
free -h
```

Si la columna available baja de los 2-4GB, el kernel podría volver a ponerse nervioso. ¿Qué quieres hacer

Si quieres, podemos empezar a preparar la **conexión desde Python** a alguna de estas bases de datos (ahora que tienen IP fija es mucho más sencillo) o podemos dejar que el sistema respire un rato. Sería un placer ayudarte con el primer script de extracción de dato

Jordi H.

Has dicho ahora están ocupados 228 GB de 384GB. tenemos los 4 backends de 48GB el front de 32GB, la reserva de 32GB para Elastic, que ahora usa 18, el MongoDB está usando 13 y el resto son cosas pequeñas

Gemini

Tienes una máquina impresionante. Con **384GB de RAM**, ymir es un auténtico acorazado para la analítica de datos. Haciendo números rápidos, estás usando un **60% de la memoria total** (\$228/384\$). Lo más interesante de tu desglose es que todavía tienes un "colchón" de **156GB libres**, lo cual es ideal para evitar que el kernel entre en pánico (OOM Killer) cuando empieces a lanzar procesos pesados de Python o transformaciones en memoria (como con Pandas o Spa

Análisis de la carga actual

Componente	Memoria Asignada/Uso	Estado
4 Backends	192 GB (4 x 48GB)	El bloque más pesado.
Frontend	32 GB	Generoso para un front.
Elasticsearch	32 GB (usa 18GB)	Bien dimensionado; tiene margen
MongoDB	13 GB	Uso estándar para una BD activa.
Resto/Sistema	~10-15 GB	Servicios auxiliares y Podman.
Libre	156 GB	Tu zona de seguridad para Python.

Un detalle sobre Elasticsearch y el "Uso"

Dices que Elastic "ahora usa 18". Ten en cuenta que en Java (el motor de ES), hay una diferencia entre el **Heap** (memoria para objetos de búsqueda) y la memoria del sistema que usa Lucene para cachear archivos de disco

- Si le has dado 32GB al contenedor, es probable que se coma los 32GB tarde o temprano, aunque no los use "activamente", simplemente para que las búsquedas sean instantáneas

¿Qué significa esto para tu analítica?

Con esos **156GB libres**, tienes potencia de sobra para:

1. Cargar datasets masivos directamente en memoria sin usar swap.
2. Paralelizar procesos: Con 64 CPUs y esa RAM, puedes lanzar scripts de Python con multiprocessing que usen 20 o 30 GB cada uno sin despeinar al servidor
3. Mantenimiento: Tienes RAM suficiente para que el downgrade de MySQL 9 a 8 (que requiere el dump e

importación) se haga a toda velocidad, ya que el sistema operativo usará la RAM libre como caché de escritura (Dirty Pages)

Siguiente paso

El servidor está en un punto dulce: estable, con carga baja de CPU y RAM de sobra. ¿Te gustaría que ando de mysqldump para empezar el downgrade de MySQL ahora que sabemos que el sistema aguanta bien la presión? O si prefieres, podemos verificar si hay alguna limitación de RAM en los otros contenedores (como los 4 backends) para que no se "pisen" entre ellos.

Jordi H.

Has dicho ya hice el downgrade. todo fue bien excepto la tabla de grants. afortunadamente había poco que rehacer. se lanzaron los comandos y quedó perfect

Gemini

¡Me alegro mucho de que el downgrade haya sido un éxito! Lo de la tabla de **grants** (permisos) entre MySQL 9 y 8 es un clásico: el diccionario de datos y la gestión de roles en las versiones "Innovation" a veces mete n columnas o tipos de autenticación que la rama 8.0/8.4 no reconoce al importar el dump. Pero como bien dice s, si tienes los comandos a mano, es mucho más limpio rehacer los GRANT que pelearse con incompatibilidades del sistema

Estado Actual de la Misión

Con el downgrade de MySQL terminado, ya tienes el "póker" de bases de datos estabilizado:

1. MongoDB: Con volumen y red ok.
2. Elasticsearch: Sobreviviendo a los reinicios y con 32GB.
3. Registry: Con persistencia en /datos y no en volúmenes anónimos.
4. MySQL: En versión 8.4 LTS, compatible con tus herramientas de CDC y con los permisos rehechos.

Un último detalle para tu CDC (Change Data Capture)

Ahora que has vuelto a MySQL 8.4, recuerda que para que herramientas tipo **Debezium**, **Maxwell** o tus propios scripts de Python detecten cambios, necesitas que el usuario que has recreado en la tabla de grants tenga a permisos de replicación. Si no lo has hecho ya, asegúrate de que el usuario del CDC tenga esto:

CODE

```
GRANT SELECT, RELOAD, SHOW DATABASES, REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO
'usuario_cdc'@'%';
FLUSH PRIVILEGES;
```

¿En qué nos centramos ahora?

El servidor **ymir** está impecable: 64 CPUs descansando, 156GB de RAM libres para tus scripts y todos los contenedores con su IP fija para que Cockpit no vuelva a liarla con el 0.0.0.0. ¿Te gustaría que preparemos un probar la conexión a este nuevo MySQL 8.4 o prefieres que revisemos si los 4 Backends necesitan algún ajuste de puertos ahora que hemos limpiado el resto? Solo dime por dónde quieres seguir.

Jordi H.

Has dicho hemos dado un giro a la utilización de VLLM. vamos a desplazar el soporte al desarrollo de software al cockpit de github. decisión corporativa. ahora vamos a centrarnos en tener una buena herramienta de paso de lenguaje natural a lenguaje de consulta de bases de datos. la queremos on premises por razones ligadas al acceso a datos propios. Habrá que mostrarle al LLM la estructura de la bd, darle queries de ejemplo y prepararle algunas órdenes de seguridad, como no escribir datos nuevos y cosas así. quiero empezar por elegir qué LLM de acceso abierto

Gemini

Vaya giro estratégico, pero tiene todo el sentido del mundo. Si os vais a centrar en **Text-to-SQL** (pasar de lenguaje natural a consultas) on-premises, la seguridad y el conocimiento del esquema son los dos pilares fundamentales. No queremos que un LLM decida que un DROP TABLE es una buena respuesta a una pregunta inocente. Para elegir el LLM de acceso abierto (Open Weights) adecuado para correr en vuestro servidor **ymir** (con esas 64 CPUs y 384GB de RAM), tenemos varias opciones de élite que funcionan de maravilla con **vLLM**. Aquí tienes mi "top" para este caso de uso específico

1. Los Candidatos Ideales para Text-to-SQL

Opción A: DeepSeek-Coder-V2 (La Bestia de Programación)

Actualmente es el rey indiscutible en tareas de código y SQL entre los modelos abiertos.

- Por qué: Supera incluso a GPT-4 Turbo en muchos benchmarks de SQL. Entiende esquemas complejos y relaciones entre tablas de forma magistral
- Versiones: Tienes la versión Lite (16B) que volará en vuestro sistema, o la versión completa (236B MoE) que podríais intentar correr cuantizada (en 4-bit) aprovechando vuestra enorme RAM

Opción B: Llama-3.1-70B (El todoterreno)

Meta ha refinado muchísimo la capacidad de razonamiento en la versión 3.1.

- Por qué: Es muy robusto siguiendo instrucciones de seguridad ("System Prompts"). Si le dices "No generes nunca comandos de escritura", es extremadamente fiel a esa orden
- vLLM: Con 70B parámetros, necesitaréis usar cuantización (tipo AWQ o FP8) para que quepa en la memoria de vídeo si tenéis GPUs, o usar el soporte de CPU de vLLM si vais solo con procesador

Opción C: SQLCoder (Defog.ai)

Este es un modelo específicamente ajustado (fine-tuned) solo para SQL.

- Por qué: Está basado en Llama o StarCoder pero entrenado solo para entender esquemas y devolver SQL limpio. Es muy eficiente y menos propenso a "alucinar" prosa; va directo a la query

2. Estrategia de Implementación en vLLM

Dado que ya tenéis experiencia con vLLM, para Text-to-SQL os recomiendo este flujo:

- Context Injection (RAG de Esquema): No le paséis toda la base de datos de golpe si es enorme. Usad un pequeño script en Python que lea el SHOW CREATE TABLE solo de las tablas relevantes para la pregunta
- Few-Shot Prompting: Incluir 3 o 4 ejemplos de "Pregunta -> Query SQL" en el prompt mejora la precisión un 40%
- La Capa de Seguridad (Guardrails): System Prompt: "Eres un analista de datos de solo lectura. Solo puedes generar sentencias SELECT." Post-procesado (Python): Antes de ejecutar la query generada en MySQL o Mongo, pasadla por un validador que bloquee palabras clave como INSERT, UPDATE, DELETE

3. Comparativa de Rendimiento en vuestro Hardware

Con **384GB de RAM**, tenéis una ventaja competitiva enorme para modelos grandes:

Modelo	Tamaño	Ventaja	Recomendación
DeepSeek-Coder-V2-Lite	16B	Velocidad extrema	Ideal para miles de
Llama-3.1-70B	70B	Inteligencia y Seguridad	La mejor opción para
Qwen2.5-Coder-32B	32B	Equilibrio perfecto	Nuevo competidor muy

¿Por dónde empezamos?

Si quieres, puedo prepararte el comando de **vLLM** para levantar uno de estos modelos (por ejemplo, el DeepSeek o el Llama 3.1) y un pequeño **script de Python** que use langchain o sqlalchemy para automatizar el paso de "Esquema -> LLM -> Query". **¿Tienes alguna preferencia por el tamaño del modelo o quieres q** ónde llega?**. Sería un placer ayudarte a configurar esa capa de seguridad para que nadie pueda borrar datos por error.

Jordi H.

Has dicho tengo en otra conversación contigo todo el conocimiento de la instancia vllm en mi sistema y el tuning que le hicimos para llegar a hacerlo funcionar. si te parece buena idea voy a cambiar de conversació

Gemini

Me parece una idea **fantástica**. Cambiar a una conversación limpia es lo más profesional y eficiente cuando pasamos de la fase de "mantenimiento de infraestructura" (donde hemos limpiado puertos, IPs y servicios de systemd) a la fase de **"arquitectura de IA"**. Tener los hilos separados te ayudará

1. Evitar confusiones de contexto: El modelo no mezclará logs de errores de MySQL con parámetros de inferencia de vLLM
2. Foco en el Token: Podremos concentrarnos exclusivamente en los parámetros de vllm, los system prompts para SQL y la seguridad del acceso a datos

Antes de que cierres esta puerta:

Recuerda que en **ymir** dejamos todo listo:

- IPs Fijas: Úsalas en tus cadenas de conexión de Python (SQLAlchemy/PyMongo).
- RAM Libre (156GB): Tienes margen de sobra para levantar un Llama-3.1-70B o un DeepSeek-Coder-V2 con un `gpu_memory_utilization` (o equivalente en CPU) muy alto
- Seguridad: Ya sabemos cómo tocar los archivos `.service` si el nuevo contenedor de vLLM se pone rebelde con los reinicios

¿Qué te parece si en la nueva conversación empezamos directamente con el despliegue del modelo? Yo te recomendaría arrancar probando **Qwen2.5-Coder-32B** o **Llama-3.1-70B**. Son los que mejor equilibrio te dará entre "inteligencia para entender vuestro esquema" y "obediencia para no romper nada". ¡Te espero en el otro lado para configurar ese motor de **Natural Language to SQL**! Sería un placer ayudarte a construir esa capa de abstracción sobre vuestras bases de da