Systemd (I)

Introducció

Systemd és vàries coses:

-El procés Init (PID 1) del sistema
-El gestor de dimonis
-Un intermediari entre aplicacions d'usuari i certes parts de l'API del kernel de Linux

La configuració general de Systemd es troba a l'arxiu "/etc/systemd/system.conf" ; molts valors per defecte hi són allà establerts.

Per saber la versió actual de Systemd que hi ha funcionant al sistema, fer *systemctl --version*

"Units": tipus i ubicació

Tot el que és gestionat per Systemd s'anomena "unit" i cada unit és descrita per un arxiu de configuració propi, el qual tindrà una extensió diferent segons el tipus de d'unit que es tracti:

.*service* : Descriu la configuració d'un dimoni
.*socket* : Descriu la configuració d'un socket (de tipus UNIX o bé TCP/IP) associat a un .service
.*device* : Descriu un dispositiu hardware reconegut pel kernel (via udev o sysfs) gestionat per Systemd
.*mount* : Descriu un punt de muntatge gestionat per Systemd
.*automount* : Descriu un punt d'automuntatge associat a un .mount
.*swap* : Descriu una partició o fitxer d'intercanvi gestionat per Systemd
.*target* : Defineix un grup d'units (s'utilitza a mode de "metapaquet" d'units)
.*path* : Descriu una carpeta o fitxer monitoritzat per l'API Inotify del kernel
.*timer*: Descriu la temporització/activació d'una tasca programada (usant el programador Systemd)
.*slice* : Defineix un grup d'units associades a processos per tal d'administrar i limitar els recursos comuns (CPU, memòria, discos, xarxa). Usa internament els anomenats "cgroups" del kernel

Els arxius de configuració de les units (siguin del tipus que siguin) poden estar repartits en tres carpetes distintes:

"/usr/lib/systemd/system": Per units proporcionades pels paquets instal.lats al sistema
"/run/systemd/system": Per units generades en temps real durant l'execució del sistema. No persistents
"/etc/systemd/system": Per units proporcionades pels administrador del sistema

Els arxius presents a /etc/... **sobreescriuen** els arxius **homònims** que estiguin a /run/... els quals sobreescriuen els que estiguin a /usr/lib/... (o en algunes distribucions, /lib/...). Si no tenen el mateix nom, tots els fitxers de les tres carpetes es mesclen ordenats pel seu nom de forma numericoalfabètica i es van llegint en aquest ordre fins el final.

D'altra banda, si dins de /usr/lib/..., /run/... o /etc/... hi ha una carpeta anomenada com una unit seguit del sufixe ".d", qualsevol fitxer amb extensió *.conf que hi hagi a dins serà llegit just després dels fitxers de configuració de la unit pertinent. Això serveix per poder **afegir (o sobreescriure)** opcions de configuració concretes (les presents en aquests fitxers) sense haver de tocar les configuracions "genèriques" de la unit. Per exemple: l'arxiu "/usr/lib/systemd/system/beep.service.d/foo.conf" pot ser útil per modificar la configuració definida a "/usr/lib/systemd/systemd/beep.service" (i d'aquesta manera, fer possible que un paquet pugui canviar la configuració establerta per un altre) i l'arxiu "/etc/systemd/system/beep.service.d/foo.conf" pot ser útil per modificar la configuració definida a "/usr/lib/systemd/system/beep.service" (i d'aquesta manera, fer possible que un administrador pugui canviar certes parts de la configuració de la unit preempaquetada al sistema sense haver de reemplaçar-la completament). Aquests fitxers "override" (concretament amb el nom "/etc/systemd/system/nomUnit.d/override.conf") es poden generar d'una manera molt còmoda i ràpida amb la comanda *systemctl edit nomUnit*

Algunes "unit" contenen un símbol @ al seu nom (per exemple, nom@cadena.service); això significa que són instàncies d'una unit-plantilla, el fitxer de configuració de la qual és el que no conté la part "cadena" al seu nom (així: nom@.service) . La part "cadena" és l'identificador de la instància (de fet, dins del fitxer de configuració de la unit-plantilla el valor "cadena" substitueix totes les ocurrències de l'especificador especial %i).

Comandes per gestionar units (principalment de tipus "service")

A continuació mostrem algunes de les comandes més importants per gestionar units principalment (que no exclusivament) de tipus "service":

systemctl [list-units] [-t {service|socket|...}] [--all | --failed | --state=inactive]

Mostra l'estat de les units que estan "actives" (del tipus indicat; si no s'indica, apareixen totes).
Si s'escriu *--state=inactive* es mostra l'estat de les units que estan "inactives"
Com a valor del paràmetre *--state* també es pot posar qualsevol valor vàlid de la columna SUB
Si s'escriu *--failed* es mostra l'estat de totes les units amb errors
Si s'escriu *--all* es mostra l'estat de totes les units ("actives", "inactives", amb errors i altres)

La diferència entre les columnes LOAD, ACTIVE i SUB la diu la sortida de la pròpia comanda:

LOAD = Indica si la unit ha sigut carregada en RAM. Possibles valors: "loaded", "error", "masked"
ACTIVE = Estat genèric de la unit. Possibles valors: "active", "inactive", "failed", "(des)activating"
SUB = Estat més concret de la unit; depén del tipus de unit. Possibles valors: "plugged","mounted","running","exited","waiting","listening",etc

systemctl [-t {service|socket|...}] list-unit-files

La subcomanda *list-units* només mostra les units que Systemd ha intentat llegir i carregar en memòria. Ja que Systemd només llegeix les units que ell pensa que necessita, això no inclou necessàriament totes les units disponibles al sistema. Per veure totes les units, incloent aquelles que Systemd no ha intentant ni tan sols carregar, cal utilitzar *list-unit-files*. Aquesta subcomanda mostra l'"estat de càrrega" de cada unit; possibles valors són:

"enabled" o "enabled-runtime" : La unit s'activarà al següent reinici -i subsegüents- .
NOTA: Això s'aconsegueix gràcies a l'existència d'un enllaç a l'arxiu de configuració de la unit en qüestió dins de la carpeta "/etc/systemd/system/*nomTarget*.target.wants", creat en algun moment previ amb la comanda *systemctl enable* (veure més avall) o bé manualment amb *ln -s*

"static" : La unit no té secció "[Install]" en el seu fitxer de configuració. Això fa que les comandes *systemctl enable* (i sobre tot *systemctl disable*) no funcionin. Per tant, el fet de què la unit estigui activada o no en un determinat "target" dependrà de l'existència "estàtica" del seu enllaç corresponent dins de la carpeta "/etc/systemd/system/*nomTarget*.target.wants" . Aquest tipus d'units solen estar associades a les que realitzen una acció "oneshot" o bé a les que són usades només com a dependència d'alguna altra unit (i per tant no han d'executar-se per sí mateixes)

"generated": La unit s'activarà mitjançant un mecanisme automàtic especial anomenat "generator", executat en arrencar el sistema. Cada unit en aquest estat té el seu propi "generator".

"transient" : La unit és temporal i no sobreviurà al següent reinici

"disabled" : La unit està desactivada i, per tant, no es posarà en marxa als següents reinicis (gràcies a la inexistència de l'enllaç corresponent dins de "/etc/systemd/system/*nomTarget*.target.wants"). Tampoc podrà ser iniciada automàticament mitjançant altres sistemes (com ara via socket, via D-Bus o bé via endollament de hardware. No obstant, podrà ser posada en marxa en qualsevol moment "manualment" executant *systemctl start (*veure més avall)

"masked" o "masked-runtime" : La unit està enmascarada (és a dir, està desactivada i, per tant, no es posarà en marxa als següents reinicis ni automàticament, però a més, tampoc podrà ser posada mai en marxa manualment amb *systemctl start* ni tan sols si és una dependència d'un altre servei)

systemctl {start|stop|restart} nomUnit[.service]

Activa/Desactiva/Reinicia la unit indicada immediatament seguint les indicacions escrites al seu arxiu de configuració corresponent.
NOTA: Si la unit no fos de tipus ".service", llavors caldrà indicar el seu tipus explícitament darrera el seu nom (per exemple, *systemctl start nomUnit.socket*). Aquesta norma és extensiva per la resta de comandes

systemctl {enable|disable} nomUnit[.service]

Activarà/Desactivarà automàticament la unit indicada a partir del següent reinici (i següents)
NOTA: En realitat el que fa *enable/disable* és crear/eliminar un enllaç dins de la carpeta "/etc/systemd/systemd/*nomTarget*.target.wants" al fitxer de configuració de la unit en qüestió (on "nomTarget" ve definit a la directiva WantedBy de la secció "[Install]" de dit fitxer).

systemctl {mask|unmask} nomUnit[.service]

Enmascara/Desenmascara la unit indicada.
NOTA: Això ho aconsegueix vinculant el fitxer de configuració ubicat a /etc/... de la unit en qüestió a /dev/null

systemctl is-enabled nomUnit[.service]

Retorna $?=0 si la unit indicada està configurada per activar-se en els següents reinicis (és a dir, si està en els estats -llistats per *systemctl list-unit-files*-: "enabled", "enabled-runtime", "static", "generated" o "transient") i a més, mostra en pantalla aquest estat.

systemctl is-active nomUnit[.service]

Retorna $?=0 si la unit indicada està activa i, a més, mostra en pantalla aquest estat (valor llistat a la columna ACTIVE de *systemctl list-units*)

systemctl is-failed nomUnit[.service]

Retorna $?=0 si la unit indicada va fallar en intentar activar-se i, a més, mostra en pantalla aquest estat. Si no volem que es mostrin els estats en pantalla (això també va per *is-enabled* i *is-active*), es pot afegir el paràmetre *-q*
NOTA: Una unit pot estar en estat "failed" por múltiples raons: perquè el procés ha acabat amb un codi d'error diferent de 0, perquè ha finalitzat de forma anormal, perquè s'ha superat un timeout determinat, etc.

systemctl status {nomUnit[.service] |PID}

Mostra l'estat i informació variada sobre la unit o procés indicat. Si s'indica una unit es pot veure...:

Loaded: loaded (/usr/lib/systemd/system/cups.service; enabled; vendor preset: disabled)
Active: active (running) since Sat 2017-11-18 20:48:06 CET; 4h 2min ago
Docs: man:cupsd(8)
***Main PID**: 745 (cupsd)*
Status: "Scheduler is running..."
Tasks: 1 (limit: 4915)
CGroup: /system.slice/cups.service
└─745 /usr/sbin/cupsd -l
Darreres línies de journald -u (es pot usar els paràmetres homònims -n nº i -o xxx)

Els valors per la línia "Loaded:" són els mateixos que apareixen a la columna LOAD de *list-units*
Seguidament s'indiquen els valors de l'estat actual i el predefinit pel paquet, que seran un dels detallats anteriorment en parlar de *list-unit-files*
Els valors per la línia "Active:" són els mateixos que apareixen a la columna ACTIVE de *list-units*
El punt ("●") és blanc si la unit està "inactive" ; vermell si "failed" o verd si "active"

Si s'indica un PID en comptes de una unit, es pot veure la mateixa informació, però aquesta manera pot ser útil per conèixer la unit a la què està associat un determinat procés, per exemple (encara que per conèixer aquesta informació també es podrien observar els valors de la columna "unit" mostrada per la comanda *ps* si així s'hi indica amb el paràmetre *-o*):

```
cups.service - CUPS Scheduler
   Loaded: loaded (/usr/lib/systemd/system/cups.service; enabled; vendor preset:
   Active: active (running) since Sat 2017-11-18 20:48:06 CET; 4h 2min ago
     Docs: man:cupsd(8)
 Main PID: 745 (cupsd)
   Status: "Scheduler is running..."
    Tasks: 1 (limit: 4915)
   CGroup: /system.slice/cups.service
           └─745 /usr/sbin/cupsd -l
Darreres línies de journald _PID= (es pot usar els paràmetres homònims -n nº i -o xxx)
```

systemctl show {nomUnit[.service] |PID}

Mostra la configuració actual de la unit (obtinguda a partir del fitxer general system.conf i del fitxer de configuració propi de la unit) indicada en un format adient per ser processat per màquines. Amb el paràmetre *-p "nomClau,unAltreNom,..."* es poden obtenir només les parelles clau<->valor desitjades.

systemctl daemon-reload

Recarrega tots els fitxers de configuració d'units noves o modificades des de la darrera vegada que es va posar en marxa Systemd (incloent els generators).

systemctl help nomUnit[.service]

Obre la pàgina de man associada a la unit indicada (al seu fitxer de configuració deu venir indicada)

systemctl edit nomUnit[.service]

Crea (amb l'editor de text predeterminat del sistema) un fitxer de configuració (inicialment buit) per la unit indicada anomenat"/etc/systemd/system/nomUnit.*tipusUnit*.d/override.conf" per sobreescriure (o ampliar) la configuració ja existent per ella. Un cop guardats els canvis, recarrega la unit automàticament amb aquesta nova configuració. Si es volgués editar directament el fitxer "/etc/systemd/system/nomUnit.*tipusUnit*", cal afegir llavors el paràmetre *--full*

systemctl cat nomUnit[.service]

Mostra la configuració final actual resultant d'haver llegit els diferents fitxers de configuració possibles de la unit indicada

systemd-delta

Mostra quins fitxers de configuració d'units estan sobreescrits o ampliats (de /usr/lib a /etc i/o amb fitxers "overrides"), enmascarats, redireccionats (amb la línia Alias= de la secció [Install]),etc i per quins

systemctl kill [--signal=nº] nomUnit[.service]

Envia una senyal concreta (indicada amb el paràmetre *--signal* ; per defecte és la nº15, SIGTERM) a tots processos associats a la unit indicada.

NOTA: `kill` goes directly and sends a signal to every process in the group, however `stop` goes through the official configured way to shut down a service, i.e. invokes the stop command configured with `ExecStop=` in the service file. Usually `stop` should be sufficient. `kill` is the tougher version, for cases where you either don't want the official shutdown command of a service to run, or when the service is hosed and hung in other ways.

Systemd també permet definir serveis per a què no estiguin associats al sistema global sinó que únicament formin part de la sessió d'un usuari estàndar, generant-se una instància particular del servei per cada usuari actiu a la màquina. D'aquesta manera, cada instància s'iniciarà automàticament després d'iniciar la sessió d'un usuari i es pararà en sortir-ne.

NOTA: Això és possible gràcies a què just després del primer inici de sessió que es realitzi al sistema es posa en marxa (gràcies al mòdul PAM "pam_systemd") la comanda *systemd –user (*qui es qui permetrà aquest funcionament individual per totes les sessions d'usuaris que s'inicïin a partir de llavors), a més del procés amb PID 1 pròpiament dit, que és *systemd –system*. El procés *systemd –user* finalitzarà automàticament just després d'haver-se tancat el darrer inici de sessió existent al sistema.

Els fitxers de configuració de les units "de tipus usuari" es troben en altres carpetes de les de les units "de sistema". Concretament (es mostren en ordre de precedència ascendent):

" /usr/lib/systemd/user": Per units proporcionades pels paquets instal.lats al sistema
"~/.local/share/systemd/user": Per units de paquets que han sigut instal.lades a la carpeta personal
"/etc/systemd/user": Per units proporcionades pels administrador del sistema
"~/.config/systemd/user" : Per units construïdes pel propi usuari

NOTA: La variable especial %h es pot utilitzar dins dels fitxers de configuració de les units "d'usuari" per tal d'indicar la ruta de la carpeta personal de l'usuari en qüestió.

Una altra característica de les units "d'usuari" és que poden ser gestionades per part d'aquest usuari sense que hagi de ser administrador del sistema; això ho pot fer amb les mateixes comandes *systemctl* ... ja conegudes només que afegint el paràmetre *--user* Així, per exemple, per arrencar un servei automàticament cada cop que s'inicïi la nostra sessió caldrà executar *systemctl --user enable nomUnit* ; per veure l'estat de totes les nostres units "d'usuari" caldrà fer *systemctl --user list-units* ; per recarregar les units modificades caldrà fer *systemctl --user daemon-reload*, etc

Seccions i directives comunes en els fitxers de configuració de les units

L'estructura interna dels fitxers de configuració de les units està organitzada en seccions, distingides cadascuna per un encapçalament case-sensitive envoltat de claudàtors (*[Encapçalament]*). Dins de les seccions es defineixen diferents directives (també case-sensitive) en la forma de parelles *NomDirectiva=valor* , on el valor pot ser una paraula, una frase, una ruta, un número, t*rue/yes* o f*alse/no*, una data, etc, tot depenent del seu significat.

NOTA: També poden existir directives on no s'escrigui cap valor (és a dir, així: *NomDirectiva=*). En aquest cas, s'estarà "resetejant" (és a dir, anul.lant) el valor que prèviament s'hagués donat en algun altre lloc

La primera secció (encara que l'ordre no importa) sempre sol ser l'anomenada **[Unit]** i s'utilitza per definir dades sobre la pròpia unit en sí com a unit que és i la relació que té aquesta amb altres units. Algunes de les seves directives més habituals són:

Description=Una breu descripció de la unit
El seu valor és retornat per diferent eines Systemd

Documentation=man:sshd(8) https://ruta/pag.html
Proporciona una lista d'URIS que apunten a documentació de la unit.
La comanda *systemctl status* les mostra

Wants=unservei.service unaltre.service untarget.target ...
Llista les units que seria bo que estiguessin iniciades per a què l'unit en qüestió pugui funcionar correctament. Si no ho estan ja, Systemd les iniciarà en paral.lel juntament amb l'unit en qüestió; si es vol indicar un cert ordre en comptes d'iniciar totes en paral.lel, es pot utilitzar les directives After= o Before=. Si alguna de les units llistades falla en iniciar-se, l'unit en qüestió s'iniciarà igualment

Requires=unservei.service unaltre.service untarget.target ...

Llista les units que imprescindiblement han d'estar iniciades per a què l'unit en qüestió pugui funcionar correctament. Si no ho estan ja, Systemd les iniciarà en paral.lel juntament amb l'unit en qüestió; si es vol indicar un cert ordre en comptes d'iniciar totes en paral.lel, es pot utilitzar les directives After= o Before=. Si alguna de les units llistades falla en iniciar-se, l'unit en qüestió també fallarà automàticament

BindsTo=unservei.service unaltre.service untarget.target ...

Similar a Requires= però, a més, fa que l'unit en qüestió s'aturi automàticament si alguna de les units associades finalitza.

Before=unservei.service unaltre.service untarget.target ...

Indica, de les units llistades a les directives Wants= o Requires=, quines no s'iniciaran en paral.lel sinò després de la unit en qüestió. Si aquí s'indiqués alguna unit que no es trobés llistada a Wants= o Requires=, aquesta directiva no es tindrà en compte.

After=unservei.service unaltre.service untarget.target ...

Indica, de les units llistades a les directives Wants= o Requires=, quines no s'iniciaran en paral.lel sinò abans de la unit en qüestió. Si aquí s'indiqués alguna unit que no es trobés llistada a Wants= o Requires=, aquesta directiva no es tindrà en compte.

NOTA: El més típic és tenir una unit A que necessita que la unit B estigui funcionant prèviament per tal de poder-se posar en marxa. En aquest cas, simplement caldria afegir les línies *Requires=B* i *After=B* a la secció [Unit] de l'unit A. Si la dependència és opcional, es pot substituir *Requires=B* per *Wants=B*

Conflicts=unservei.service unaltre.service ...

Llista les units que no poden estar funcionant al mateix temps que l'unit en qüestió. Iniciar una unit amb aquesta directiva causarà que les aquí llistades s'aturin automàticament.

ConditionXXXX=...

Hi ha un conjunt de directives que comencen per "Condition" que permeten a l'administrador comprovar certes condicions abans d'iniciar l'unit. Si la condició no es compleix, la unit és ignorada. Alguns exemples són:

ConditionKernelCommandLine=param[=valor]
ConditionACPower={yes| no}
ConditionPathExists=[!]/ruta/fitxer/o/carpeta
ConditionPathExistsGlob=[!]/ruta/fitxers/o/carpetes
ConditionPathIsDirectory=[!]/ruta/carpeta
ConditionPathIsSymbolicLink=[!]/ruta/enllaç
ConditionPathIsMountPoint=[!]/ruta/carpeta
ConditionPathIsReadWrite=[!]/ruta/fitxer/o/carpeta
ConditionDirectoryNotEmpty=[!]/ruta/carpeta
ConditionFileNotEmpty=[!]/ruta/fitxer
ConditionFileIsExecutable=[!]/ruta/fitxer

AssertXXXX=...

Igual que amb "ConditionXXX", hi ha un conjunt de directives que comencen per "Assert" que permeten a l'administrador comprovar certes condicions abans d'iniciar l'unit. La diferència és que aquí, si la condició no es compleix, s'emet un error.

OnFailure=unaunit.service unaaltra.service ...

Indica les units que s'activaran quan la unit en qüestió entri en estat "failed". Aquesta directiva pot fer-se servir, per exemple, per executar una unit que envïi un correu electrònic quan la unit en qüestió, que podrà ser un servei, falli.

AllowIsolate=yes

Aquesta directiva només té sentit per units de tipus target. Si el seu valor és "yes" (per defecte és "no") indica que el target en qüestió admetrà que se li apliqui la comanda *systemctl isolate* (veure més avall)

D'altra banda, la darrera secció (encara que l'ordre no importa) d'un arxiu de configuració d'una unit sempre sol ser l'anomenada **[Install]** , la qual, atenció, és opcional. S'utilitza per definir com i quan l'unit pot ser activada o desactivada. Algunes de les seves directives més habituals són:

WantedBy=untarget.target unaltre.target ...

Indica els targets on l'unit en qüestió s'activarà en executar la comanda *systemctl enable* Quan s'executa aquesta comanda, el que passa és que per cada target indicat aquí apareixerà, dins de cada carpeta "/etc/systemd/system/nomTarget.wants" respectiva, un enllaç simbòlic apuntant al propi arxiu de configuració de l'unit en qüestió. L'existència d'aquest enllaç és el que realment activa de forma efectiva un servei automàticament. Eliminar els links de totes les carpetes "nomTarget.wants" pertinents implica desactivar la unit (que és el que fa, de fet, la comanda *systemctl disable* a partir de la llista de targets que troba a la línia *WantedBy=*). Per exemple, si l'arxiu de configuració de la unit en qüestió (que anomenarem pepito.service) té una línia com WantedBy=multi-user.target, en executar systemctl enable pepito.service apareixerà dins de la carpeta /etc/systemd/system/multi-user.wants un link apuntant a aquest arxiu de configuració

RequiredBy=untarget.target unaltre.target ...

Similar a WantedBy= però on la fallada de l'unit en qüestió en executar *systemctl enable* farà que els targets indicats aquí no es puguin arribar a assolir. La carpeta on es troba el link de l'unit en aquest cas s'anomena "/etc/systemd/system/nomTarget.requires"

Alias=unaltrenom.tipusUnit

Permet a l'unit en qüestió ser activada amb *systemctl enable* utilitzant un altre nom diferent

Also=unservei.service unaltre.service ...

Permet activar o desactivar diferents units com a conjunt. La llista ha de consistir en totes les units que també es volen tenir habilitades quan la unit en qüestió estigui habilitada

Secció [Service] (per units de tipus .service):

Depenent del tipus d'unit que tinguem ens podrem trobar amb diferents seccions específiques dins del seu fitxer de configuració, normalment escrites entre la secció [Unit] del principi i la secció [Install] del final (si hi és). En el cas de les units de tipus "service", per exemple, ens trobem amb la secció específica anomenada **[Service]**, la qual pot incloure diferents directives com les següents:

NOTA: Les units de tipus device, snapshot i target no tenen seccions específiques

Type=maneraDarrancar

Hi ha diferents mètodes per iniciar un servei, i el mètode escollit, el qual dependrà del tipus d'executable a posar en marxa, s'ha d'indicar en aquesta directiva. Les possibilitats més comunes són:

simple: El servei nativament es queda en primer pla de forma indefinida i és Systemd qui el posa en segon pla (li crea un fitxer PID, el para quan calgui, etc). Systemd interpreta que el servei està llest tan bon punt l'executable associat es posa en marxa (encara que això sigui massa aviat perquè no estigui llest encara per rebre peticions)

forking: El servei nativament ja es posa en segon pla. Systemd interpreta que el servei està llest quan passa efectivament a segon pla. En aquest cas convé indicar també la directiva PIDFile=/ruta/fitxer.pid per a què Systemd tingui un control sobre quin procés és el que està en segon pla i el pugui identificar

oneshot: Útil per scripts, que s'executen (fent *systemctl start* igualment) un cop i finalitzen. Systemd s'esperarà fins que el procés finalitzi i interpreta que està llest quan hagi finalitzat. Es pot considerar l'ús de la directiva **RemainAfterExit=yes** per a "enganyar" a Systemd dient-li que el servei continua actiu encara que el procés hagi finalitzat; en aquest cas, la directiva *ExecStop=* no s'arribarà a fer efectiva mai.

També hi ha les possibilitats "dbus" (similar a "simple" però Systemd interpreta que està llest quan el nom indicat a BusName= ha sigut adquirit), "idle" (similar a "simple" però amb l'execució retardada fins que no s'executi res més; es pot utilitzar aquest mètode, per exemple, per emetre un so just després de la finalització de l'arranc del sistema.) i "notify" (el sistema més complet, on s'estableix un canal de comunicació intern entre el servei i Systemd per tal de notificar-se estats i events via l'API pròpia de Systemd sd_notify() i on Systemd interpreta que està llest quan rep l'estat corresponent a través d'aquest canal; si volem que scripts facin servir aquest mètode cal usar la comanda *systemd-notify*)

ExecStart=/ruta/executable param1 param2 ...

Indica la comanda (i paràmetres) a executar quan es realitza un *systemctl start* Si la ruta de l'executable comença amb un guió ("-"), valors de retorn de la comanda diferents de 0 (que normalment es considerarien senyal d'error) es consideraran com a vàlids.

NOTA: Podem utilitzar fins i tot la directiva **SuccessExitStatus=** per indicar quin valor considerem com a sortida exitosa del programa
NOTA: No caldria escriure la ruta absoluta de l'executable si aquesta es troba a la llista de rutes que mostra la comanda *systemd-path*, però en general es recomana escriure-la per evitar sorpreses
NOTA: No es permeten escriure redireccionadors (">",">>","<"."|") ni el símbol "&" per passar a segon pla

ExecStartPre=/ruta/executable param1 param2 ...

Indica la comanda (i paràmetres) a executar abans de la indicada a ExecStart. Poden haver més d'una línia ExecStartPre al mateix arxiu, executant-se llavors cadascuna per ordre. La ruta de l'executable també pot anar precedida d'un guió ("-"), amb el mateix significat

ExecStartPost=/ruta/executable param1 param2 ...

Indica la comanda (i paràmetres) a executar després de la indicada a ExecStart. Poden haver més d'una línia ExecStartPost al mateix arxiu, executant-se llavors cadascuna per ordre. Un exemple de possible ús: l'enviament d'un correu just després d'haver-se posat en marxa el servei corresponent.La ruta de l'executable també pot anar precedida d'un guió ("-"), amb el mateix significat

ExecStop=/ruta/executable param1 param2 ...

Indica la comanda (i paràmetres) a executar quan es realitza un *systemctl stop* Cal tenir en compte que en el cas d'un servei de tipus "oneshot", si no s'especifica la directiva *RemainAfterExit=yes*, la comanda indicada a ExecStop s'executarà automàticament just després d'ExecStart.

ExecStopPost=/ruta/executable param1 param2 ...

Indica la comanda (i paràmetres) a executar després de la indicada a ExecStop. Poden haver més d'una línia ExecStartPost al mateix arxiu, executant-se llavors cadascuna per ordre.

Restart={ always | no | on-success | on-failure | ... }

Indica les circumstàncies sota les que Systemd intentarà reiniciar automàticament un servei que hagi finalitzat. En concret, el valor "always" indica que en qualsevol tipus de finalització es tornarà a intentar reiniciar; el valor "no" indica que en cap finalització s'intentarà reiniciar, el valor "on-success" indica que només s'intentarà reiniciar si la finalització ha sigut correcta i "on-failure" si la finalització no ho ha sigut degut a qualsevol tipus de fallada (ja sigui que s'ha sobrepassat el temps d'espera de l'arranc o l'apagada, que s'ha retornat un valor diferent de 0, etc)

NOTA: Es podria donar el cas de què un servei estigués reiniciant-se tota l'estona. Amb **StartLimitBurts=** es pot configurar el número màxim de vegades que es vol que es reinicïi i amb **StartLimitIntervalSec=** es pot configurar el temps durant el qual es comptarà aquest número màxim de vegades. Si s'arriba a aquest número dins d'aquest temps, el servei no es tornarà a reiniciar automàticament i tampoc no es podrà iniciar manualment fins passat el temps indicat (moment en el qual es torna a comptar). També existeix la directiva **StartLimitAction=** , la qual serveix per indicar l'acció a realitzar quan s'arriba al número màxim de reinicis; el seu valor per defecte és "none" però pot valer també "reboot" (reinici net), "reboot-force" (reinici abrupte) i "reboot-immediate" (reinici molt abrupte)

RestartSec = nºs

Indica el número de segons que Systemd s'esperarà en reiniciar el servei després de què s'hagi aturat (si així ho marca la directiva Restart=).

TimeoutSec=nº

Indica el número de segons que Systemd esperarà a què el servei en qüestió s'inicïi o s'aturi abans de marcar-lo com a "failed" (i reiniciar-lo si fos el cas degut a la configuració de la directiva Restart=). Es pot indicar específicament un temps d'espera només per l'inici amb la directiva **TimeoutStartSec=** i un altre temps d'espera diferent per l'apagada amb la directiva **TimeoutStopSec=**. Si no s'especifica res, s'agafa el valor per defecte (5 min) que està indicat a /etc/systemd/system.conf

RemainAfterExit=yes

Tal com ja ho hem comentat, aquesta directiva s'utilitza en serveis de tipus "oneshot" per a què la directiva ExecStop= no s'executi en acabar l'execució de la comanda sinò en fer *systemctl stop*

PIDFile= /ruta/fitxer.pid

Tal com ja ho hem comentat, aquesta directiva s'utilitza en serveis de tipus "forking" per a senyalar a Systemd quin serà el fitxer PID utilitzat pel servei de manera que el pugui controlar més fàcilment.

User=unusuari
Group=ungrup

Set the user or group that the processes are executed as, respectively. They can take a single user/group name, or a numeric ID as argument. For system services (services run by the system service manager, i.e. managed by PID 1) the default is "root". For user services of any other user, switching user identity is not permitted, hence the only valid setting is the same user the user's service manager is running as. If no group is set, the default group of the user is used.

WorkingDirectory=/ruta/carpeta

Indica el directori de treball del servei en qüestió. Si no s'especifica aquesta directiva, el valor per defecte és "/" (en el cas de serveis de sistema) o $HOME (en el cas de serveis d'usuari -és a dir, iniciats amb *--user* -). Si la ruta es precedeix amb un símbol "-", el fet de què la carpeta corresponent no existeixi no s'interpretarà com un error. Si s'ha indicat la directiva User=, es pot escriure "~" com a valor d'aquesta directiva, equivalent així a la ruta de la carpeta personal de l'usuari indicat a User=.

StandardOutput= { null | tty | journal | socket }

Indica on s'imprimirà la sortida estàndar dels programes indicats a les directives ExecStart=, i ExecStop= . El valor "null" representa el destí /dev/null. El valor "tty" representa un terminal (ja sigui de tipus virtual -/dev/ttyX- o pseudo -/dev/pts/X-), el qual haurà de ser especificat mitjançant la directiva **TTYPath=** . El valor "journal" és el valor per defecte (és a dir, que si el programa en qüestió imprimís alguna cosa a la pantalla del terminal en executar-se en primer pla, aquesta sortida es redireccionarà al Journal en executar-se via un arxiu .service) . El valor "socket" serveix per indicar que la sortida ha de enviar-se al socket associat al servidor per tal de viatjar a l'altre extrem de la comunicació (veure més endavant).

NOTA: El fet de què per defecte la sortida estàndar vagi a parar al Journal es pot canviar de forma general per totes les units a la directiva DefaultStandardOutput del fitxer /etc/systemd/system.conf
NOTA: També existeix la directiva **StandardError= { null | tty | journal | socket }** , similar a StandardOutput= però per la sortida d'error

EXERCICIS:

Tots els exercicis es faran en una màquina virtual on l'usuari pugui tenir permisos d'administrador:

1.-a) Executa *systemctl list-units -t service | grep ufw* (si estàs a Ubuntu) o *systemctl list-units -t service | grep firewalld* (si està a Fedora). ¿Què vol dir la paraula "loaded"? I "active"? Confirma-ho executant *systemctl status ufw / systemctl status firewalld*

b) Ara enmascara la unit ufw/firewalld. ¿Què passa realment quan s'enmascara una unit? Pista: consulteu on apunta el recentment creat arxiu /etc/systemd/system/ufw.service (o /etc/systemd/system/firewalld.service)

c) Executa *systemctl list-units -t service | grep ufw* (o *systemctl list-units -t service | grep firewalld*) de nou. ¿Per què encara apareix la paraula "active"? Confirma-ho executant *systemctl status ufw / systemctl status firewalld*

d) Si ara executes *systemctl stop ufw* (o *systemctl stop firewalld*), ¿què mostra *systemctl list-units -t service | grep ufw* (o *systemctl list-units -t service | grep firewalld*) ? ¿Per què? ¿Què hauries de fer per a què veiessis alguna cosa? Pista: fes servir el paràmetre *--all*

e) Executa la comanda *systemd-delta* . ¿Què significa la relació indicada entre els dos fitxers que apareixen a la línia [MASKED]? ¿I entre els dos fitxers que apareixen a les línies [EXTENDED]?
NOTA: També hi podria haver alguna parella de fitxer en línies [OVERRIDEN] o inclús [EQUIVALENT]

f) Intenta iniciar l'unit ufw (o firewalld) encara enmascarada. Pots? Desenmascara-la i torna-ho a provar. Pots ara?

g) Executa la comanda *systemctl edit ufw* (o *systemctl edit firewalld*) i, a l'editor de text que apareix, escriu la línia *Description=Hola amigo* i guarda. Tot seguit, executa *systemctl cat ufw* (o *systemctl cat firewalld*). ¿Què veus? I si tornes a executar *systemd-delta* ?

2.-a) Crea un fitxer anomenat /etc/systemd/system/pepe.service amb el següent contingut ...:

```
[Unit]
Description=Pepe es colega
[Service]
Type=oneshot
ExecStart=/bin/ls -l
ExecStop=/bin/df -h
[Install]
WantedBy=multi-user.target
```

...i tot seguit (després de *systemctl daemon-reload)* executa *systemctl start pepe* Què veus? I si fas *journalctl -e*, què veus? Per què?

b) Què hauries de modificar de l'arxiu anterior per a què la comanda /bin/df -h no s'executés just després de /bin/ls -l sinò només quan s'escriguís *systemctl stop pepe*? Pista: consulta l'explicació del tipus "oneshot" a la teoria. Prova-ho utilitzant les comandes *systemctl –full edit pepe , systemctl daemon-reload* i, un altre cop, *journalctl -e*

c) Per a què serveix la línia *WantedBy=...* ? O dit d'una altra manera: ¿quina relació té aquesta línia amb la comanda *systemctl enable* ? Pista: observa el contingut de la carpeta /etc/systemd/system/multi-user.target.wants

d) Què hauries de modificar de l'arxiu anterior per a què la sortida de les comandes executades per la unit (ja sigui a ExectStart= o a ExecStop=) no vagi a parar al Journal sinò que es visualitzi al terminal /dev/tty4? Prova-ho.

3.-a) Crea un script anomenat "/opt/yeah" amb el següent contingut (i dóna-li permisos d'execució):

```
#!/bin/bash
while [[ true ]]
do
        curl -s ipinfo.io/ip
        /bin/sleep 3
done
```

b) Crea un fitxer anomenat "/etc/systemd/system/pepa.service" amb el següent contingut ...:

```
[Unit]
Description=Pepa es colega
[Service]
Type=simple
ExecStartPre=/usr/bin/systemd-cat -t PEPA -p crit  echo "Empiezo"
ExecStart=/opt/yeah
ExecStop=/usr/bin/systemd-cat -t PEPA -p crit  echo "Acabo"
#StandardOutput=null
[Install]
WantedBy=multi-user.target
```

...i tot seguit (després de *systemctl daemon-reload)* executa *systemctl start pepa* . *S*i fas *journalctl -f,* què veus? Per què? I si executes *systemctl stop pepa* , què veus llavors al Journal ? Per què? I si descomentes la línia que apareix comentada i tornes a provar?

4.-a) Crea un fitxer anomenat "fiufiu.service" dins de "/etc/systemd/system" amb el següent contingut...:

```
[Unit]
Description=All we are saying is give peace a chance
[Service]
Type=simple
ExecStart=/usr/bin/nc -l -p 5555
Restart=on-success
```

...i seguidament posa'l en marxa amb la comanda *sudo systemctl start fiufiu* . Comprova amb *systemctl status fiufiu* (o també amb *ss -tnl*) que s'hagi iniciat correctament

b) Executa la comanda *journalctl -ef* i seguidament, obre un altre terminal per executar-hi la comanda *nc 127.0.0.1 5555*. Escriu-hi alguna cosa a la connexió oberta per aquest client Netcat i observa a la vegada el que apareix en temps real al Journal. ¿Què passa? Per què?

c) Tanca el client i torna'l a executar. ¿El servei continua funcionant? Ara comenta la línia Restart=... que apareix al fitxer fiufiu.service, reinicia el servei i torna a executar el client un parell de vegades. La primera vegada haurà de connectar-se sense problemes com sempre però la segona ja no. ¿Per què?

5.-a) Instal.la el paquet "apache2" i observa, executant la comanda *systemctl cat apache2*, el valor que té la directiva Restart= . Executa llavors *systemctl --signal=9 kill apache2* i comprova amb *systemctl status apache2* si el servei reinicia sol o no. Per què passa el que passa?

b) Canvia ara el valor de la directiva Restart= de l'unit de l'Apache2 mitjançant *systemctl edit --full apache2* per a què valgui "no" (recorda d'escriure com a primera línia el títol de la secció a la què pertany la directiva que vols sobrescriure -és a dir, [Service]-). Després de fer *systemctl daemon-reload* (i de comprovar que la modificació és efectiva amb *systemd-delta* o també *systemctl cat apache2)*, inicia el servei un altre cop i comprova que efectivament estigui iniciat. Torna'l a matar un altre cop amb *systemctl –signal=9 kill apache2* i torna a comprovar un altre cop si el servei ha reiniciat automàticament o no. Què passa ara?

c)Afegeix mitjançant *systemctl edit apache2* la línia necessària per tal cridar a una unit que s'encarregui d'executar en el mode "oneshot" la comanda *play /ruta/un/fitxer.mp3* cada cop que l'Apache finalitzi degut a alguna situació inesperada (com per exemple seria una senyal kill 9)

> NOTA: La comanda play s'encarrega de reproduir el fitxer de so indicat i admet molts formats possibles, no només mp3. Forma part del paquet "sox"

6.a) Executa *systemctl disable ufw* (o *systemctl disable firewalld*) i tot seguit *systemctl show --property "Wants" multi-user.target | grep -E "(ufw|firewalld)"* Què veus? Per què? Pista: observa el missatge que apareix a pantalla en deshabilitar el servei ufw

b) I si ara executes *systemctl enable ufw* (o *systemctl enable firewalld*) i tornes a executar la mateixa comanda? Què veus ara? Per què? Pista: observa el contingut de la carpeta /etc/systemd/system/multi-user.target.wants

c) Dedueix i digues per què la línia After= de la unit ufw.service (o firewalld.service) té el valor que té.

d) Systemd pot no ser el procés INIT del nostre sistema Linux: encara que sigui el més extés amb diferència, et pots trobar distribucions que facin servir sistemes INIT alternatius. Digues, de les següents comandes, quines serveixen per comprovar si el procés INIT del teu sistema és Systemd (o no):

file /sbin/init
man init
pgrep ^systemd$

Systemd (II)

Targets

Podem definir un "target" com un "estat" del sistema definit per un determinat conjunt de serveis posats en marxa (i uns altres que no). La idea és que, en arrencar el sistema, s'arribi a un determinat "target" (i, opcionalment, a partir d'allà, poder passar a un altre si fos necessari). A continuació es llisten els "targets" més importants (tots ells ubicats dins de /usr/lib/systemd/system):

poweroff.target (o "runlevel0.target") Si s'arriba a aquest "target", s'apaga el sistema

reboot.target (o "runlevel6.target"): Si s'arriba a aquest "target", es reinicia el sistema

rescue.target (o "runlevel1.target"): Si s'arriba a aquest "target", s'inicia el sistema en mode text, sense xarxa i només per l'usuari root. Seria similar a un altre target anomenat "**emergency.target**", però l'"emergency" és més "radical" que el "rescue" perquè gràcies a muntar la partició arrel en mode només lectura permet arrencar sistemes que el "rescue" potser no pot.

multi-user.target (o "runlevel3.target") : En aquest cas s'inicia el sistema en mode text però amb xarxa i multiusuari (el target per defecte en servidors)

graphical.target (o "runlevel5.target") : En aquest cas, s'inicia el sistema en mode gràfic amb xarxa i multiusuari (el target per defecte en sistemes d'escriptori) Implica haver passat pel target "multi-user" prèviament.

Altres targets predefinits que s'instal.len amb Systemd i que cal conèixer són:

ctrl-alt-del.target
Target activat quan es pulsa CTRL+ALT+SUPR. Per defecte és un enllaç a "reboot.target"

sysinit.target
Target que executa els primers scripts d'arranc

sockets.target
Target que activa, en arrencar, totes les units de tipus "socket". Es recomana, per tant, que tots els arxius de configuració d'una unit "socket" tinguin a la seva línia Wants= aquest target indicat (o bé WantedBy=)

timers.target
Target que activa, en arrencar, totes les units de tipus "timer". Es recomana, per tant, que tots els arxius de configuració d'una unit "timer" tinguin a la seva línia Wants= aquest target indicat (o bé WantedBy=)

paths.target
Target que activa, en arrencar, totes les units de tipus "path". Es recomana, per tant, que tots els arxius de configuració d'una unit "path" tinguin a la seva línia Wants= aquest target indicat (o bé WantedBy=)

swap.target
Target que habilita la memòria swap

basic.target
Target que posa en marxa tots els target relacionats amb punts de muntatge, memòries swaps, paths, timers, sockets i altres unitats bàsiques necessàries pel funcionament del sistema.

initrd-fs.target

El generador systemd-fstab-generator afegeix automàticament les units indicades a la directiva Before= d'aquesta unit a la unit especial "sysroot-usr.mount" (a més de tots els punts de muntatge existents a /etc/fstab que tinguin establertes les opcions "auto" i "x-initrd.mount"). Veure més endavant una explicació de les units de tipus mount.

initrd-root-fs.target

El generador systemd-fstab-generator afegeix automàticament les units indicades a la directiva Before= d'aquesta unit a la unit especial "sysroot-usr.mount", la qual és generada a partir dels paràmetres del kernel. Ho estudiarem més endavant

local-fs.target

El generador systemd-fstab-generator afegeix automàticament les units indicades a la directiva Before= d'aquesta unit a totes les units de tipus "mount" que es refereixen a punts de muntatge locals. També afegeix a aquest target les dependències de tipus Wants= corresponents als punts de muntatge existents a /etc/fstab que tenen l'opció "auto" establerta. Veure més endavant una explicació de les units de tipus mount.

network-online.target

Target que s'activa automàticament tan bon punt el subsistema de xarxa és funcional. Qualsevol servei que hagi de treballar en xarxa s'haurà d'iniciar com a mínim en aquest target

NOTA: Existeix un altre tarjet relacionat amb la xarxa anomenat "pre-network.target" que està pensat per iniciar serveis abans de què qualsevol tarja de xarxa es configuri. El seu propòsit principal és fer-lo servi amb serveis de tipus tallafocs, per tal d'establir les regles abans de què la configuració de xarxa funcioni. Aquests serveis hauran de tenir una línia *Before=network-pre.target* i també una línia *Wants=network-pre.target* al seu arxiu de configuració
NOTA: Existeix un altre tarjet relacionat amb la xarxa anomenat simplement "network.target" que només indica que l'stack software de xarxa ja s'ha carregat en memòria però això no implica que les interfícies s'hagin configurat encara. Aquest target està més pensat pel procés d'apagada de la màquina per tal de realitzar aquest procés de forma ordenada: ja que l'ordre d'apagada és al revés que el d'arrencada, qualsevol unit que tingui una línia *After=network.target* s'apagarà abans que la xarxa es descarregui i això farà que aquesta unit s'apagui sense interrompre cap connexió que estigui pendent

printer.target

Target que s'activa automàticament tan bon punt una impressora és endollada o apareix disponible durant l'arranc. Aquí on se sol iniciar, per exemple, el servei Cups.

sound.target

Target que s'activa automàticament tan bon punt una tarja d'àudio és endollada o apareix disponible durant l'arranc.

bluetooth.target

Target que s'activa automàticament tan bon punt un controlador Bluetooth és endollat o apareix disponible durant l'arranc.

smartcard.target

Target que s'activa automàticament tan bon punt un controlador Smartcard és endollat o apareix disponible durant l'arranc.

system-update.target

Target especial utilitzada per actualitzacions del sistema. El generador systemd-system-update-generator redireccionarà el procés d'arranc automàticament a aquest target si la carpeta /system-update existeix

umount.target

Target que desmunta tots els punts "mount" i "automount" durant l'apagada del sistema

final.target

Target utilitzat durant l'apagada del sistema que pot fer-se servir per apagar els últims serveis després de què els serveis "normals" ja s'han aturat i els punts de muntatge s'han desmuntat.

Per saber el target on ens trobem en aquest moment podem fer: *systemctl get-default*

Cal tenir en compte que múltiples targets poden estar activats a la vegada. Un target activat indica que Systemd ha intentat iniciar totes les units associades a aquest target. Això vol dir que la comanda anterior només ens diu quin és el target "final" on hem arribat, però al llarg del camí des de l'arranc de la màquina fins arribar a aquest target "final" s'han anat activant diferents targets a mode de "graons" intermitjos. Per veure tots els targets activats, cal fer *systemctl list-units –type=target*

Es pot canviar el target actual a un altre simplement executant: *systemctl isolate nomTargetDesti.target* Per canviar el target per defecte on s'anirà a parar automàticament a cada arranc del sistema es pot fer: *systemctl set-default nomTargetDefecte.target*

NOTA: La comanda anterior, en realitat l'únic que fa és revincular el link "/etc/systemd/system/default.target" a l'arxiu *.target adient.

NOTA: Una altra manera d'entrar al final de l'arranc del sistema en un determinat target per defecte és afegir la línia *systemd.unit=nomTargetDesti.target* a la llista de paràmetres del kernel indicada a la configuració del gestor d'arranc

Hi ha una sèrie de comandes específiques per passar a determinats estats (poweroff, reboot, etc) que es poden fer servir en comptes de la comanda *systemctl isolate* genèrica. Per exemple:

sudo systemctl rescue : Similar a *systemctl isolate rescue.target*
sudo systemctl poweroff (o *sudo poweroff* a seques): Similar a *systemctl isolate poweroff.target*
sudo systemctl reboot (o *sudo reboot* a seques): Similar a *systemctl isolate reboot.target*

Si es vol aturar (-P) o reiniciar (-r) la màquina en un moment futur determinat (hh:mm), llavors caldrà executar la comanda: *sudo shutdown {-P | -r } hh:mm*
Si es vol aturar (-P) o reiniciar (-r) la màquina d'aquí a una certa quantitat de minuts, llavors caldrà executar la comanda: *sudo shutdown {-P | -r } +m*

NOTA: Dins de la carpeta "/usr/lib/systemd/system-shutdown" poden haver arxius *.shutdown,que són scripts executables que s'executaran just abans de l'apagada/reinici del sistema (és a dir, just en posar en marxa els serveis "poweroff.service" o "reboot.service"). Qui executarà aquests scripts és el binari /usr/lib/systemd/systemd-shutdown, el qual és invocat sempre per aquests serveis, que el col.loquen com a PID 1 i és el responsable de desmuntar dels sistemes de fitxers, deshabilitar la swap, matar els processos que quedin pendents, etc. Als scripts executats per /usr/lib/systemd/systemd-shutdown podem utilitzar un paràmetre ($1) que pot valer "poweroff" o "reboot" depenent de l'acció que realitzarà i que ens podria servir per distingir què volem que faci aquest script segons l'acció indicada. Tots els scripts s'executen en paral.lel. Cal tenir en compte, finalment, que el sistema de fitxers en aquell moment roman muntat però en mode només lectura.

NOTA: A la comanda *systemctl reboot* li podem afegir varis paràmetres interessants, els quals només funcionen, però, en sistemes UEFI que han arrencat mitjançant el gestor d'arranc *systemd-boot*:
--firmware-setup : Indicate to the system's firmware to reboot into the firmware setup interface (a.k.a the "UEFI control panel")
--boot-loader-menu=nºsegons : Indicate to the system's boot loader to show the boot loader menu on the following boot the number of seconds specified as value. Pass 0 value in order to disable the menu timeout.
–boot-loader-entry=entryID : Indicate to the system's boot loader to boot into a specific boot loader entry on the following boot. Takes a boot loader entry identifier as argument, or "help" in order to list available entries.
El que fan els paràmetres anteriors és modificar determinats valors de variables EFI concretes (tal com es podria haver fet també amb la comanda *efibootmgr*) per així modificar el comportament de la UEFI al proper arranc

---

D'altra banda, amb la comanda *sudo systemctl suspend* podem suspendre el sistema (o dit d'una altra manera, ens permeten arribar al target "suspend.target") i amb la comanda *sudo systemctl hibernate* la podem posar a hivernar (o dit d'una altra manera, en permeten arribar al target "hibernate.target"). "Suspendre" significa que es guarda tot l'estat del sistema a la RAM i s'apaga la majoria de dispositius de la màquina; quan s'engega de nou, el sistema restaura el seu estat previ de la RAM sense haver de reiniciar-se de nou: aquest procés és molt ràpid però té l'inconvenient de que obliga a mantenir amb alimentació elèctrica la màquina tota l'estona. "Hibernar" significa que es guarda tot l'estat del sistema al disc dur (si té espai lliure) i s'apaga del tot la màquina: quan s'engega de nou, el sistema restaura el seu estat previ des del disc dur sense haver de reiniciar-se de nou: aquest procés és força lent però té l'avantatge de no haver de mantenir amb alimentació elèctrica la màquina.

NOTA: Dins de la carpeta "/usr/lib/systemd/system-sleep" poden haver arxius *.sleep, que són scripts executables que s'executaran just abans de la hibernació o suspensió del sistema (és a dir, just en posar en marxa internament els serveis "systemd-hibernate.service" o "systemd-suspend.service", els quals, per cert, mai han de ser invocats directament amb *systemctl start* ... sinó fent servir les comandes explicades al paràgraf anterior: *systemctl hibernate* o *systemctl suspend*). Qui executarà aquests scripts és el binari /usr/lib/systemd/systemd-sleep, el qual és invocat sempre per aquests serveis i admet dos paràmetres que podem fer servir en aquests scripts com $1 i $2 respectivament. El primer paràmetre pot valer "pre" o "post" depenent de si la màquina està anant a la suspensió/hibernació o n'està tornant, respectivament. El segon paràmetre pot valer "suspend" o "hibernate" depenent de l'acció que realitzarà i que ens podria server per distingir què volem que faci aquest script segons l'acció indicada. Tots els scripts s'executen en paral.lel.

Si es vol realitzar una tasca llarga i assegurar-se de què la màquina no es suspendrà o apagarà mentrestant, es pot invocar la comanda corresponent a aquesta tasca així: *systemd-inhibit comanda_llarga* La comanda *systemd-inhibit --list* mostra les tasques que tenen aquest truc en marxa. Si es vol especificar una acció concreta a inhibir es pot indicar amb el paràmetre *--what=accio* , on "acció" pot ser per exemple la paraula "shutdown" o "sleep" (equivalent a hibernació o suspensió), entre d'altres. Trobareu més informació als primers paràgrafs de https://www.freedesktop.org/wiki/Software/systemd/inhibit/

Per a què l'inici *amb systemctl start* d'un determinat servei (o target) es produeixi dins d'un target determinat -anomenem-lo "a.target"- des del propi fitxer de configuració del servei en qüestió cal escriure les directives *Wants=a.target*, *Requires=a.target* i/o *After=a.target* (aquestes directives s'asseguren d'arribar primer al target "a.target" per iniciar llavors el servei en qüestió). D'altra banda, també existeix la directiva *Conflicts=a.target* , la qual s'assegura de **no** estar en el target "a.target" per poder iniciar el servei en qüestió.

En el cas de voler iniciar sempre un servei determinat automàticament en el target "a.target", llavors caldrà escriure a més les directives *WantedBy=a.target* o *RequiredBy=a.target* del fitxer de configuració del servei (en aquest darrer cas, en fer *systemctl enable nomServei* es crea un enllaç al seu arxiu de configuració dins de "/lib/systemd/system/a.target.wants").

Els arxius de configuració dels targets només tenen seccions [Unit] (i molt poques la secció [Install]). En aquest sentit, és interessant consultar els arxius corresponents, per exemple, a multi-user.target o graphical.target: només trobem les directives Description, Documentation, Wants, Requires, After, Conflicts i AllowIsolate (i a partir d'elles podem deduir les dependències que hi ha entre targets...encara que per això hi ha comandes específiques que de seguida veurem).

EXERCICIS:

1.-Crea un target nou anomenat "manolo.target" on el sistema haurà d'entrar just després d'activar graphical.target (és a dir,ha de ser l'últim target en activar-se). La idea serà assegurar-te de què entres en aquest target per tal d'executar un determinat servei (l'anomenarem "manolo.service") l'últim de tots. Per fer això, has de fer el següent:

a) Crea un nou fitxer anomenat "/etc/systemd/system/manolo.target" amb el següent contingut:

[Unit]
Description=Manolo is kind
Documentation=http://www.lecturas.com
Requires=graphical.target
After=graphical.target
Conflicts=rescue.service rescue.target
AllowIsolate=yes

b) Crea un nou fitxer anomenat "/etc/systemd/system/manolo.service" amb el següent contingut:

[Unit]
Description=Manolo is superb

Documentation=http://www.hola.com
Requires=manolo.target
After=manolo.target
[Service]
ExecStart=/usr/bin/printf "MANOLO\n"
RemainAfterExit=yes
[Install]
WantedBy=manolo.target

c) Executa *systemctl enable manolo.service* i reinicia la màquina. ¿Creus que el servei "manolo" estarà funcionant automàticament o no? Per què ho creus? Comprova-ho executant la comanda *systemctl status manolo.service* (o també observant si apareix la paraula "MANOLO" al Journal). PISTA: La resposta de perquè el servei "manolo" estarà funcionant (o no) es troba en el que mostra la comanda *systemctl list-units -t target | grep "manolo"* i en entendre el significat de la línia *WantedBy=*

d) Tot seguit executa *systemctl start manolo.service* . Després d'observar que el servei s'hagi posat en marxa correctament (en la sortida de la comanda *systemctl status manolo.service* o també observant si apareix la paraula "MANOLO" al Journal), ¿creus que la comanda *systemctl list-units -t target | grep "manolo"* mostrarà alguna cosa diferent respecte l'apartat anterior? Per què ho creus? PISTA: La resposta es troba en entendre el significat de les línies *Requires=* i *After=* de l'arxiu "manolo.service"

e) Executa *systemctl set-default manolo.target* i torna a reiniciar la màquina. ¿Creus que el servei "manolo" ara estarà funcionant automàticament o no? Per què ho creus? Comprova-ho executant la comanda *systemctl status manolo.service* (o també observant si apareix la paraula "MANOLO" al Journal). PISTA: La resposta es troba en entendre el significat de les línies *Requires=* i *After=* de l'arxiu "manolo.target"

2.-a) Entra en el target de rescat. Què passa?

b) Entra en el target de suspensió. Què passa?

c) Entra en el target multi-user. Què passa?

d) Crea un script executable dins de la carpeta "/usr/lib/systemd/system-sleep" amb el següent contingut...:

```
#!/bin/bash
if [[ "$1" == "pre" ]]
then
        echo "we are suspending or hibernating at $(date)..." > /tmp/systemd_suspend_test
elif [[ "$1" == "post" ]]
then
        echo "...and we are back from $(date)" >> /tmp/systemd_suspend_test
fi
```

...i prova de suspendre el sistema i tornar-lo a "despertar". Què passarà?

e) ¿Per a què serveix aquest programa: https://github.com/ryran/reboot-guard ?

Boot chain

Per saber la jerarquia de dependències de targets per arribar a iniciar un target (o service!) determinat es pot utilitzar la comanda: *systemctl list-dependencies nomTarget.target* (o *nomUnit.service*)

> NOTA: Una manera alternativa de obtenir una informació similar seria executant *systemctl show -p "Wants" nomTarget.target && systemctl show -p "Requires" nomTarget.target* . També es pot executar *systemctl status*

Les dependències mostrades es corresponen a les units que han sigut "required" o "wanted" per les units superiors. Les dependències recursives només es mostren pels targets intermitjos; si es volen veure també pels service, mounts, paths, socket, etc intermitjos cal incloure el paràmetre *--all* a la comanda anterior.

També es poden mostrar quines units depenen per funcionar del correcte inici d'un target (o service!) determinat amb la comanda: *systemctl list-dependencies --reverse nomTarget.target* (o *nomUnit.service*)

> NOTA: Una manera alternativa de obtenir una informació similar seria executant *systemctl show -p "WantedBy" nomTarget.target && systemctl show -p "RequiredBy" nomTarget.target*

Altres paràmetres interessants d'aquesta comanda són *--before* i *--after*, els quals serveixen per mostrar les units que depenen per funcionar del correcte inici anterior o posterior d'un target, respectivament.

D'altra banda, respecte l'arranc del sistema podem obtenir una informació més detallada sobre els temps que triga cada unit en carregar-se i l'ordre en què ho fa gràcies a la comanda *systemd-analyze*, la qual té vàries possibilitats

systemd-analyze **:** Mostra el temps total emprat en l'arranc del sistema i quina part d'aquest temps ha sigut emprat en tasques del kernel, quina part en ús de l'initrd i quina part en tasques d'usuari

systemd-analyze blame : Mostra els temps disgregats per servei. Cal indicar que aquests temps són "en paral.lel", així que la suma total que surt serà sempre molt superior al temps real emprat en l'arranc.

systemd-analyze dot [nomTarget.target] | dot -T {png | svg} -o foto.{png|svg} : Genera una sortida que si es passa a l'aplicació "dot" (pertanyent al paquet "graphviz") generarà finalment un gràfic (en format png o svg) on es poden visualitzar totes les dependències del target (o servei!) indicat (o, si no s'indica, del "default.target"; també es poden indicar comodins al nom del target/servei).

systemd-analyze plot [nomTarget.target] > something.svg : Genera un gràfic on es mostra els temps d'execució i de bloqueig de cada unit durant l'arranc fins arribar al target (o servei!) indicat (o, si no s'indica, del "default.target")

systemd-analyze critical-chain [nomTarget.target] : Mostra l'arbre de dependències bloquejants pel target (o servei!) indicat. El temps mostrat després de "@" indica el temps que fa que la unit està activa; el temps mostrat després de "+" indica el temps que la unit ha trigat en activar-se.

Altres opcions de la comanda systemd-analyze són *syscall-filter, verify, dump, log-level, security, time ...*

Es pot veure si, un cop iniciat el sistema, encara queden tasques pertanyents a l'arranc per completar executant la comanda *systemctl list-jobs*

EXERCICIS:

1.- a) Quins targets han d'haver-se iniciat per a què el servei gdm es pugui posar en marxa? (això ho pots saber amb la comanda s*ystemctl list-dependencies* ...)

b) Executa la comanda *systemctl list-dependencies --reverse gdm.service* Què veus?

c) Què fa la comanda *tree /etc/systemd/system* i per a què podria servir-te?

2.-a) Executa *systemd-analyze plot* ... i observa quina unit bloqueja més temps l'arranc del teu sistema. Prova de desactivar-la (esperem no trencar res!) i reinicia. Executa ara *systemd-analyze blame* per comprovar si el temps total d'arranc ha disminuït efectivament.

b) Instal.la el paquet "graphviz" i genera un gràfic Png amb les dependències del target multi-user. Fes una llista de les dependències allà mostrades i adjunta la captura del gràfic

Plantilles

Una plantilla és un fitxer de configuració de tipus "service" que té la particularitat de permetre posar en marxa variants d'un mateix servei sense haver d'escriure un arxiu "service" diferent per cada variant. Bàsicament, per utilitzar una plantilla cal fer els següents passos:

1.-L'arxiu "service" que farà de plantilla s'ha d'anomenar "nomServei@.service" . És a dir, cal indicar el símbol arroba abans del punt

2.-El contingut d'aquest arxiu plantilla pot ser exactament igual que el d'un arxiu "service" estàndar

3.-A l'hora d'iniciar, parar, habilitar, deshabilitar, veure l'estat, etc d'una plantilla, caldrà indicar l'identificador concret de la variant amb la què volem treballar. Aquest identificador s'estableix la primera vegada que arrenca la variant i simplement consisteix en una cadena entre l'arroba i el punt, així: *systemctl start nomServei@**identificador.**service* A partir d'aquí, aquest identificador es farà servir de la mateixa manera per la resta de tasques relacionades amb la gestió d'aquesta variant

NOTA: An instance file is usually created as a symbolic link to the template file, with the link name including the instance identifier. In this way, multiple links with unique identifiers can point back to a single template file. When managing an instance unit, systemd will look for a file with the exact instance name you specify on the command line to use but if it cannot find one, it will look for an associated template file.

4.-La gràcia de les plantilles és que el valor de l'identificador indicat al punt anterior es pot utilitzar dinàmicament dins del contingut de l'arxiu plantilla (concretament mitjançant el símbol "**%i**"), de manera que segons el valor que hagi adquirit %i per aquella variant es podria posar en marxa el servei escoltant en un port diferent (si %i representa un número de port), o bé utilitzant un arxiu de configuració diferent (si %i representa un nom de fitxer), o el que ens convingui.

NOTA: Altres símbols especials que es poden indicar en un arxiu de configuració d'una plantilla poden ser
- **%p**: Represents the unit name prefix (this is the portion of the unit name that comes before the @ symbol)
- **%n**: Represents the full resulting unit name (%p plus %i)
- **%u**: The name of the user configured to run the unit.
- **%U**: The same as above, but as a numeric UID instead of name.
- **%H**: The host name of the system that is running the unit.
- **%%**: This is used to insert a literal percentage sign.

Posem un exemple. Imaginem que tenim un determinat servidor web que volem executar amb dues configuracions diferents a la vegada. La solució seria crear un fitxer plantilla anomenat per exemple "servidorweb@.service" amb un contingut similar al següent:

[Unit]
Description=My HTTP server
[Service]
Type=simple
ExecStart=/usr/sbin/webServer --config-file /etc/%i.conf
[Install]
WantedBy=multi-user.target

Amb aquest arxiu, es podria iniciar llavors el servidor dues vegades, cadascuna indicant el nom del fitxer de configuració desitjat, així:

sudo systemctl start servidorweb@config1.service
sudo systemctl start servidorweb@config2.service

Les comandes anteriors el que faran serà executar, respectivament, les comandes: */usr/sbin/webServer –config-file /etc/config1.conf* i */usr/sbin/webServer –config-file /etc/config2.conf*

EXERCICIS:

1.-a) Crea un arxiu plantilla que permeti posar en marxa diferents servidors Ncat de forma permanent (recorda el paràmetre -k) escoltant cadascun d'ells en un port diferent.

NOTA: Hauràs d'instal.lar el paquet "nmap" per disposar de la comanda *ncat*

b) Inicia un servidor Ncat a partir de la plantilla anterior escoltant al port 2222 i un altre escoltant al port 3333. Comprova que, efectivament, aquests dos ports estiguin oberts observant la sortida de la comanda *ss -tnl*

c) Connecta amb el client Ncat a un dels servidors anteriors i envia-li algun missatge. Tanca el client (amb CTRL+C) i ara torna a executar-lo per connectar a l'altre servidor; torna a enviar-li algun altre missatge i tanca'l de nou. Observa les darreres línies del Journal: ¿què hi veus?

2.-Llegeix el següent paràgraf i seguidament contesta :

When the user switches consoles using Ctrl+Alt+F2, Ctrl+Alt+F3,and so on, a new terminal then is spawned. In this case systemd calls a service named getty@.service providing the appropriate argument such as tty2 or tty3 to the unit file. The %i identifier provides this argument value to the agetty binary so the terminal starts on that new console (as it can seen in ExecStart= line from template file).

a) Per a què serveix la comanda agetty ? Busca a la seva pàgina del manual què fa el seu paràmetre -a i afegeix-lo a la invocació de la comanda escrita a la línia ExecStart de dins de l'arxiu getty@.service (recorda d'executar *sudo systemctl daemon-reload* just després). Què passa ara quan pulses Ctrl+Alt+F2, etc ?

b) Enmascara la instància tty5 de la plantilla getty@.service. Què passa ara si fas Ctrl+Alt+F5 ?

NOTA: És possible que també hagis d'emmascarar la plantilla autovt@.service per a què funcioni l'exercici
NOTA: Hi ha altres maneres més sofisticades de desactivar terminals virtuals però les veurem més endavant

c) Què et mostra la comanda *systemctl status getty@** ?

3.-Suposa que tens un arxiu anomenat "/etc/systemd/system/pepe@.service" amb el següent contingut:

[Unit]
Description=lerele
[Service]
Type=oneshot
ExecStart=/usr/local/bin/systemd-email %i admin@elpuig.xeill.net
User=nobody
Group=systemd-journal

on "systemd-email" és un bash shell script escrit per nosaltres dissenyat per enviar correus (suposant que tenim un servidor Postfix o similar configurat a la màquina) que té el següent codi:

#!/bin/bash
systemctl status –full "$1" | mail -s "$1" $2

i suposa que has afegit la línia OnFailure=pepe@%i.service dins la secció [Unit] de l'arxiu ".service" corresponent al/s servei/s que vols monitoritzar.

a) ¿Quina comanda hauries d'executar per posar en marxa una instància del servei-plantilla pepe@ que s'encarregui d'envïar mails en el moment que el servei Cups falli?

Sockets

Un aspecte molt interessant de Systemd és que permet que un servidor no estigui permanentment encés sinó que només arrenqui "sota demanda" (és a dir, quan detecti una connexió, normalment externa). D'aquesta manera, aquest servidor no consumeix més recursos que els mínims imprescindibles, en el moment just. Per aconseguir això, el que passa és que sí que hi ha un component "escoltant" tota l'estona possibles intents de connexions, però aquest component no és pas la unit "service" en sí sinò un "gos guardià" que només despertarà la unit "service" quan calgui. Aquest "gos guardià" és la unit de tipus "socket".

Cada fitxer de configuració d'una unit "socket" ha de tenir exactament el mateix nom que el fitxer de configuració de la unit "service" que vol despertar (és a dir, si tenim el servei "a.service", el socket corresponent haurà d'anomenar-se "a.socket"). La idea és tenir la unit "socket" sempre encesa (*systemctl enable a.socket*) però la unit "service" no (*systemctl disable a.service*); quan es detecti una connexió, el "socket" automàticament encendrà la unit "service" (això es pot veure fent *systemctl status a.service* mentre existeix la connexió) i l'apagarà de nou passat un determinat temps sense activitat (per defecte 5 minuts). Òbviament, si paréssim el "socket" (*systemctl stop a.socket*) o el deshabilitéssim pel proper reinici (*systemctl disable a.socket*) ja no hi hauria "gos guardià" amatent i, per tant, el servei ja no es posaria mai en marxa automàticament.

Per canviar el port on escolta un "socket" (entre altres coses) cal modificar la configuració del "socket" pròpiament dit i això no depén de la configuració del servidor en qüestió. Els arxius de configuració de cada "socket" es poden trobar, com qualsevol altra unit, o bé dins de la carpeta "/usr/lib/systemd/system" o bé dins de "/etc/systemd/system" i es pot utilitzar igualment la comanda *systemctl edit a.socket* per tal de generar arxius "override". La secció que ens interessa en aquests tipus de fitxers és la secció **[Socket]**, la qual pot contenir alguna de les següents directives més importants:

ListenStream=[IP:]nºport
Indica el número de port TCP per on escoltarà el socket. Opcionalment, es pot indicar una IP concreta per especificar que només escoltarà en el port oferit per aquella IP i cap més.

NOTA: Es poden indicar vàries línies *ListemStream* per fer que el socket escolti en varis ports a la vegada. D'altra banda, com que aquesta línia pot estar escrita en diferents fitxers, si es vol assegurar que només s'escolti en un port concret sense tenir en compte altres línies que pugui haver llegit Systemd prèviament, es pot afegir primer una línia *ListemStream* buida (així: *ListemStream=*) i després la línia *ListenStream* associada al port desitjat; el que fa la línia *ListemStream* buida és "resetejar" totes les línies *ListemStream* anteriors

ListenDatagram=[IP:]nºport

Indica el número de port UDP per on escoltarà el socket. Opcionalment, es pot indicar una IP concreta per especificar que només escoltarà en el port oferit per aquella IP i cap més.

ListenSequentialPacket=/ruta/arxiu.socket

Indica el socket de tipus UNIX per on s'escoltarà. Només serveix per comunicacions entre processos de la mateixa màquina

Service=unNomAlternatiu

Si el nom de l'arxiu "service" no és igual que el nom de l'arxiu "socket", aquí es pot indicar llavors el nom que té l'arxiu "service" per a què el socket el sàpiga trobar.

Accept=yes

Si s'indica, fa que es generi una instància del servei diferent per cada connexió. Útil quan es fan servir plantilles. Si el seu valor és no (per defecte) només una instància del servei gestionarà totes les connexions.

La comanda *systemctl status *.socket* ens permet saber quants i quins sockets estan escoltant ara mateix; el valor "Accepted" mostra quantes connexions s'han realitzat en total des de què el socket ha sigut iniciat i el valor "Connected" mostra quantes connexions estan actualment actives

Com qualsevol altra unit, es poden veure la llista de sockets amb la comanda *systemctl list-units -t socket* però a més disposem de la comanda específica *systemctl list-sockets*, la qual informa de quin servei corresponent activen i en quin port/socket UNIX escolten.

EXERCICIS:

1.-a) Crea el fitxer "/etc/systemd/system/dateserver.socket" amb el següent contingut:

```
[Unit]
Description=Servei de data al port 55555
[Socket]
ListenStream=55555
Accept=true
[Install]
WantedBy=sockets.target
```

b) Crea el fitxer "/etc/systemd/system/dateserver@.service" amb el següent contingut:

```
[Unit]
Description=Servei de data
[Service]
Type=simple
ExecStart=/opt/dateserver.sh
StandardOutput=socket
StandardError=journal
```

c) Crea el fitxer "/opt/dateserver.sh" amb el següent contingut (i dona-li permisos d'execució!):

```
#!/bin/bash
while [[ true ]]
do
```

#Atenció: comprova que date es trobi dins de /usr/bin ; depenent de la distribució això canvia
/usr/bin/date
sleep 1
done

d) Obre un terminal i executa la comanda *nc ipServidor 55555* . Què veus ? Obre un altre terminal diferent i executa la mateixa comanda. Què veus ? Què et mostra la comanda *systemctl status dateserver.socket*? ¿I la comanda *systemctl status dateserver@** ? ¿I la comanda *systemctl list-units dateserver@** ?

2.- Fes que el servidor SSH que tinguis instal.lat a la màquina (si no el tens, instal.la'l) s'inicïi només a través d'un socket. Concretament:

a) Crea el fitxer "/etc/systemd/system/sshMitjo.socket" amb el següent contingut:

[Unit]
Description=El meu SSH Socket
[Socket]
ListenStream=22
Accept=yes
[Install]
WantedBy=sockets.target

b) Crea el fitxer "/etc/systemd/system/sshMitjo@.service" amb el següent contingut:

[Unit]
Description=El meu SSH Server
[Service]
Type=simple
ExecStart=-/usr/sbin/sshd -i
StandardInput=socket
StandardOutput=socket

NOTA: Aquí la clau està en la combinació del paràmetre -i del binari sshd (el qual fa que habilitar la possibilitat de què pugui rebre peticions a través de sockets), i la directiva StandardInput (la qual realitza de forma efectiva aquest tipus de comunicació entre el socket i el servidor SSH)

NOTA: Important is the "-" in front of the binary name. This ensures that the exit status of the per-connection sshd process is forgotten by systemd. Normally, systemd will store the exit status of a all service instances that die abnormally. SSH will sometimes die abnormally with an exit code of 1 or similar, and we want to make sure that this doesn't cause systemd to keep around information for numerous previous connections that died this way (until this information is forgotten with systemctl reset-failed).

c) Executa la comanda *systemctl enable sshMitjo.socket* i *systemctl disable ssh.service* (si calgués) i reinicia la màquina. Un cop fet, comprova que el socket sshMitjo estigui funcionant pero no el servei sshMitjo. Executa *ssh usuari@ipServidor* per entrar al servidor SSH (ho hauries d'aconseguir sense problemes) i comprova seguidament que ara sí que està funcionant una instància del servei sshMitjo

d) ¿Què faria una comanda com *systemctl kill sshd@172.31.0.52:22-172.31.0.4:47779.service*?

3.-a) En l'exercici anterior hem tingut la sort de què el servidor SSH ofereix un paràmetre (-i) que li permet delegar l'apertura dels sockets (ports) a un "agent extern" com és Systemd. Però no sempre tindrem un servidor que ofereixi aquesta possibilitat. En aquest sentit, llegeix els següents paràgrafs i resum amb les teves pròpies paraules què explica:

One of the limitations of socket activation is that it requires the activated application to be aware that it may be socket-activated; the process of accepting an existing socket is different from creating a listening socket from scratch. Consequently a lot of widely used applications don't support it. The systemd developers have known that it may take some time to get activation support everywhere, so they introduced "systemd-socket-proxyd", a small TCP and Unix domain socket proxy server. This does understand activation, and will sit between the network and our server, transparently forwarding packets between the two. The steps to use this tool are:

Step 1:We create a socket that listens on the port that will eventually be served by the proxy/server combination.
Step 2: On the first connection to the socket systemd activates the proxy service and hands it the socket.
Step 3: When the proxy is started the corresponding server is first brought up (thanks to the Requires/After dependency)

The proxy then shuttles all traffic between the server and the network. The only trick here is that we need to bind the server to a port other than the real-target port (8080 instead of 80 if we are running a webserver, for instance). This is because that port will be owned by the socket/proxy, and you can't bind two processes to the same socket and interface.

Step 1: "myserver-proxy.socket" file

```
[Socket]
ListenStream=0.0.0.0:80
[Install]
WantedBy=sockets.target
```

Step 2: "myserver-proxy.service" file

```
[Unit]
Requires=myserver.service
After=myserver.service
[Service]
ExecStart=/usr/lib/systemd/systemd-socket-proxyd 127.0.0.1:8080
```

Step 3: "myserver.service" file

```
[Unit]
Description=Server example (replace ExecStart value with something more realistic)
[Service]
#We listening server's listening port only to loopback interface because it's there where input packages comes from proxy
ExecStart=/usr/bin/ncat -k -l 127.0.0.1 8080
```