Guia per Integrar K8s amb Tor

Curs 2021-22 Q1

Ferran Mateu Berga

Aleix Velasco Calvo

Albert Vilardell Barnosell

Tutor: Felix Freitag

Facultat d'Informàtica de Barcelona (FIB)

22 de desembre de 2021

Índex

Introducció

No és estrany llegir notícies de censura a Internet. El control que es du a terme sobre la xarxa no para de créixer. I sembla que el futur continuarà amb aquesta tendència. Una de les eines més populars que s'utilitza per saltar-se aquest control és Tor.

Per una altra banda, el món dels serveis web s'ha revolucionat en els últims anys. Degut a l'augment de tràfic de la xarxa, s'ha creat la necessitat de dissenyar la web de manera escalable. D'aquesta manera, ha aparegut el concepte de containerització i de microservei. L'eina utilitzada per gestionar-ho és Kubernetes (K8s), i ha obtingut moltíssima popularitat.

Aquest treball consisteix a dissenyar una infraestructura Kubernetes escalable i integrar-la dins de la xarxa Tor. L'única manera d'accedir al servei és des de dins de Tor. Així, pretenem assegurar l'anonimitat del client però també la del servidor. L'objectiu del treball és explicar, a mode de guia, els passos i de les decisions de disseny que s'han de prendre per aconseguir-ho i bones pràctiques a seguir, especialment per la part de Tor.

Hem desenvolupat, també, un servei web molt senzill. La importància està en el *back-end*, així que s'hi ha dedicat molt més temps que en el *front-end*. Tant el servei web com la infraestructura estan a les màquines virtuals de la facultat (FIB).

Finalment, hem fet una avaluació de rendiment per veure si la latència afegida de la xarxa Tor fa inviable la idea d'implementar Kubernetes en Tor.

Hi ha una sèrie d'annexos que tracten diferents temes: des de com utilitzar K8s en les VM de la FIB fins a un recopilatori de comandes de Kubernetes que hem utilitzat durant tot el treball.

No tenim constància que s'hagi integrat Kubernetes amb Tor abans en projectes d'aquesta assignatura ni tampoc hem trobat guies semblants a internet. Per tant, procurem fer una guia que pugui ser útil per aquell que vulgui fer-ho.

Tor

El primer que cal fer és muntar la infraestructura de Tor, ja que així ens assegurarem de mantenir un seguit de bones pràctiques al llarg del desenvolupament del projecte que garantiran l'anonimat del servidor i eliminaran qualsevol possible connexió entre nosaltres i el mateix servidor. A més, tampoc hi hauria d'haver cap relació entre el servidor i el *hidden service.*

El servidor

A l'hora de triar el servidor tenim dues opcions, o bé en tenim un de propi i simplement el connectem a la xarxa, o bé, lloguem un servidor a una empresa de hosting.

Servidor propi

Ens interessa ubicar el servidor en un lloc diferent dels que freqüentem habitualment per tal de minimitzar les possibilitats que es pugui relacionar el servidor amb nosaltres. Un cop tenim la localització del servidor hi configurem un servidor SSH per tal de poder-hi accedir remotament a través de TOR. Més endavant hi instal·lem tor a través de:

```
sudo apt install tor -y.
```

Ens assegurem que tant els serveis de sshd i tor estan corrent i que s'engeguen en iniciar la màquina a través de:

```
sudo systemctl enable sshd

sudo systemctl enable tor

sudo systemctl restart sshd

sudo systemctl restart tor
```

Si les anteriors comandes s'han executat sense errors ja ho tenim llest.

Servidor llogat (VPS)

Instal·lació del navegador TOR

Per a la instal·lació del navegador cal anar a https://www.torproject.org/download/ i allà descarregar i seguir els passos segons el sistema operatiu que tinguem.

Contractació del VPS

Un cop tenim descarregat el navegador TOR, a través de la xarxa TOR podem procedir a la contractació d'un servidor remot. D'aquesta manera, no es podrà saber des d'on s'està fent la contractació, és a dir, ni el proveïdor d'internet ni l'empresa de hosting podrà saber des d'on s'està fent la petició.

Abans de triar cap empresa de hosting cal tenir en compte que per poder contractar-ne els serveis haurem de fer pagaments i que ens interessa fer-los anònimament, amb bitcoin per exemple, i no amb targeta de crèdit que és fàcilment rastrejable.

Un cop tinguem el servidor que volem llogar, el lloguem, l'engeguem i ens assegurem que el servei de tor i sshd estan corrent i que s'engeguen en iniciar la màquina a través de:

```
sudo systemctl enable sshd

sudo systemctl enable tor

sudo systemctl restart sshd

sudo systemctl restart tor
```

Si les anteriors comandes s'han executat sense errors ja ho tenim llest.

Configuració de SSH a través de TOR

Per a accedir al servidor tampoc ens interessa que es puguin rastrejar les connexions que hi fem, per tant, encaminarem la connexió SSH a través de la xarxa TOR.

Per fer-ho, utilitzarem un proxy SOCKS5 que encaminarà la sessió ssh a través de TOR.

En instal·lar *tor*, també hem instal·lat tot un seguit d'eines com *torsocks* i *torify* que fan bàsicament el mateix. Amb la comanda següent (al host) podem aconseguir que la sessió SSH vagi a través de TOR:

```
torify ssh alumne@nattech.fib.upc.edu -p 40369
```

Per saber que funciona, podem comprovar amb tcpdump a on s'ha enviat la petició des del host i en la màquina remota podem mirar des de quina IP s'ha accedit [veure Annex 1].

Més endavant, quan configurem el hidden service, crearem un servei per a poder accedir a la màquina a través de la seva adreça onion [explicació a l'apartat Configuració SSH].

Un cop tenim l'accés al servidor de manera anònima (usant *torify*) podem procedir a seguir amb les configuracions que hem de fer per posar en marxa el nostre *hidden service*.

Actualització i posada a punt del sistema

Abans de començar a treballar amb la màquina ens interessa que estigui al dia amb les últimes actualitzacions per assegurar-nos d'estar protegits contra vulnerabilitats conegudes.

Com que actualitzar una màquina és una cosa bastant comuna i sense riscos podem simplement executar: [`apt update && apt upgrade`] i ja tindrem la màquina actualitzada. Tot i això, pels usuaris més preocupats també es poden executar aquestes comandes utilitzant *torify* i així anonimitzar les peticions als repositoris.

Un cop actualitzat el servidor, instal·lem l'editor de text que preferim i les eines bàsiques que creiem que necessitarem i ens agrada tenir.

Creació del *hidden service*

El procés de crear un hidden service, bàsicament, ens dona l'adreça *.onion* mitjançant la qual podrem accedir a un determinat servei.

Per tal d'indicar al servei de Tor que volem un *hidden service* cal que modifiquem el fitxer de configuració de Tor que es troba a: `/etc/tor/torrc` .

Aquest fitxer es divideix en tres seccions separades per línies com la següent:

```
################ Nom de la següent secció ####################
```

La primera secció (aquesta no té títol), és la configuració bàsica del servei. Aquí hem de descomentar la línia:

```
#RunAsDaemon 1
```

Això farà que el servei pugui funcionar com a dimoni. També hem de descomentar la línia:

```
#DataDirectory /var/lib/tor
```

Que especifica el directori on el servei guardarà totes les claus.

A continuació passarem a la secció de configuració de la secció dels *hidden services*, la segona secció.

Aquí hem d'especificar dues coses. Primer, a on volem que es crei el directori amb totes les dades d'aquest hidden service. Hem d'especificar-ho amb la línia:

```
HiddenServiceDir /var/lib/tor/hidden_service/
```

On a hidden_service podem posar el nom que vulguem per identificar un servei en concret.

Un cop tenim això hem d'indicar a Tor que redirigeixi les peticions rebudes a un port en concret de l'adreça *.onion* d'aquest servei a l'adreça IP i port que desitgem. Això ho podem fer amb la línia a continuació de l'anterior:

```
HiddenServicePort 80 127.0.0.1:8080
```

On el 80 és el port de l'adreça *.onion* i 127.0.0.1:8080 l'adreça IP i el port a on es redirigeix la petició.

Amb això tenim que quan es faci una petició al port 80 de l'adreça *.onion* d'aquest servei, Tor redirigirà la petició al servei local que tinguem corrent a 127.0.0.1:8080. Podríem dir que actua com un *proxy*.

Per arrencar el *hidden service* i obtenir l'adreça *.onion* només cal que reiniciem el servei de Tor amb:

```
systemctl restart tor
```

Si naveguem al directori que hem especificat al fitxer de configuració (en aquest exemple: /var/lib/tor/hidden_service/) veurem que s'hi han creat els següents fitxers:

```
root@obelix:/var/lib/tor/hidden_service# ll
total 24
drwx--S--- 3 debian-tor debian-tor 4096 Dec  1 17:56 ./
drwx--S--- 4 debian-tor debian-tor 4096 Dec  6 11:54 ../
drwx--S--- 2 debian-tor debian-tor 4096 Nov 14 17:15 authorized_clients/
-rw------- 1 debian-tor debian-tor   63 Dec  1 17:56 hostname
-rw------- 1 debian-tor debian-tor   64 Nov 14 17:15 hs_ed25519_public_key
-rw------- 1 debian-tor debian-tor   96 Nov 14 17:15 hs_ed25519_secret_key
```

On el fitxer *hostname* conté l'adreça *.onion*.

Per fer proves podem arrencar un servei web a 127.0.0.1:8080 amb la pàgina índex que vulguem i accedir a l'adreça *.onion*.

Per accedir a una adreça *.onion* no ho podem fer amb un navegador normal, ens cal fer servir el navegador de Tor (The Tor Project, n.d.) o si estem en un servidor de testing podem fer servir pàgines com https://tor2web.onionsearchengine.com/ des d'un navegador normal.

COMPTE! Mai hem de fer servir *tor2web* o similars en un servidor de producció si volem mantenir l'anonimat!

Amb això ja tenim el *hidden service* en marxa! Per saber com l'adreça *.onion* és accessible des de l'exterior i com es realitza la connexió amb el client veure Annex 2.

Millorant la seguretat del servidor

Ara per ara ja tenim un servidor amb un *hidden service* en marxa. Ens podem quedar aquí i funcionaria perfectament.

Tot i això, hi ha tot un seguit de comprovacions i de mesures a prendre que milloren la seguretat dels nostres *hidden services* i que ens ajuden a mantenir l'anonimat.

Configuració de firewall

Una opció és que totes les connexions que no passin pel proxy estiguin capades o redirigides a Tor a través de nftables o iptables. Així ens assegurem que el servidor no accedeix a la *clear net* i que totes les peticions les hem configurat per anar a través de Tor.

Això ens assegura que no es pugui relacionar cap activitat que fem a internet amb la IP pública del nostre servidor.

Configuració SSH

Com hem explicat abans, podem fer la connexió amb el servidor encaminant la connexió SSH amb *torify*. Tot i això, la petició es fa fora de la xarxa de Tor, estem accedint a una IP de la *clear net* i, per tant, al servidor s'hi pot provar accedir des d'on sigui.

Per tant, qualsevol persona podria provar d'autenticar-se contra el nostre servidor, i si per algun motiu aconseguís les claus o es descobrís una vulnerabilitat podria entrar-hi.

Per mitigar això podem configurar el servidor SSH com a un altre *hidden service* de tal manera que només sigui accessible a través de Tor coneixent-ne l'adreça *.onion*.

Cal crear un nou *hidden service* per tal d'accedir amb una altra adreça *.onion* diferent de la del servei. Així, limitem al mínim la relació entre el *hidden service* i l'administrador.

Per fer això ens cal modificar el fitxer de configuració de SSH per tal que escolti peticions d'una adreça privada i port com la de localhost:8089 i crear un nou *hidden service* com hem fet amb l'altre redirigint les peticions a aquest nou servei a l'adreça que hem especificat a ssh, en l'exemple d'abans: localhost:8089.

Captures de les noves configuracions dels fitxers: `/etc/ssh/sshd_config`

`/etc/tor/torrc`

```
HiddenServiceDir /var/lib/tor/ssh_hidden_service/
HiddenServicePort 22 127.0.0.1:8089
```

```
Port 8089
#AddressFamily any
ListenAddress 127.0.0.1
#ListenAddress 0.0.0.0
#ListenAddress ::
Protocol 2
```

Un cop fem això, només podrem accedir al nostre hidden service de la següent manera:

```
torify ssh alumne@27s6mmuwtf4tgc6gkk2zl7kwqpf4r7tveqpg7q2m46zs7tb3zmpuwyid.onion
```

Si fem el login i comprovem des d'on s'ha accedit a la màquina podrem veure com s'accedeix des de localhost:

```
alumne@obelix:~$ last
alumne   pts/1        127.0.0.1        Sat Dec 18 16:45   still logged in
```

I que accedint-hi des de la IP del servidor no hi podem entrar:

```
alumne@obelix:~$ exit
logout
Connection to 27s6mmuwtf4tgc6gkk2zl7kwqpf4r7tveqpg7q2m46zs7tb3zmpuwyid.onion
sed.
ferran@donete:~$ ssh alumne@nattech.fib.upc.edu -p 40369
ssh: connect to host nattech.fib.upc.edu port 40369: Connection refused
ferran@donete:~$
```

Hi ha altres bones pràctiques a considerar a l'hora de posar en marxa un *hidden service,* però se'n van bastant més enllà de l'objectiu d'aquest treball. En resum, com més cura i més precaucions adoptem, millor.

Per més informació de com muntar un *hidden service* consultar bibliografia (Tasker, 2015).

Kubernetes

En aquest apartat, comentarem a fons la infraestructura Kubernetes que hem implementat i en els problemes que hem tingut.

Què és Kubernetes

Primer de tot, abans d'adentrarnos en la creació de tota la infraestructura de serveis, hem de saber que és Kubernetes. Kubernetes, també anomenat K8s, és un sistema *open source* pel desplegament automàtic, escalat i gestió d'aplicacions en contenidors o així és com ells es defineixen.

Per explicar una mica millor aquesta definició hem d'explicar l'evolució que ha patit el desplegament d'aplicacions al llarg dels anys. Al principi de tot, el que es feia era desplegar diferents aplicacions en una mateixa màquina física, però això es va veure que no era una bona pràctica, ja que el que passava és que algunes aplicacions es quedaven més recursos i feia que les altres tinguessin un baix rendiment. Per aquest motiu es va migrar a les màquines virtuals, diverses màquines virtuals, cadascuna amb una aplicació, en una màquina física. Això va ser un gran avanç, ja que proporciona un nivell més de seguretat, una millor distribució dels recursos i una millor escalabilitat. Però també tenia inconvenients, s'afegia un overhead, ja que cada màquina virtual té el seu propi sistema operatiu. Per aquest motiu es va migrar al sistema de contenidors, que són els que tenim avui dia.

Aquí és on entra en joc Kubernetes, aquest sistema facilita tota la gestió de les aplicacions en contenidors que podem tindre.

Instal·lació

Ara ja sabem que és Kubernetes, així que podem començar a realitzar tota la instal·lació. Per no fer aquest apartat molt enrevessat només comentarem els passos que s'han de realitzar, totes les comandes utilitzades les podreu trobar a l'Annex 3.

Primer de tot instal·lem l'eina docker, per a la creació dels containers, i algunes dependències que necessitarem posteriorment.

A continuació instal·lem les diferents eines que necessitem pels components del cluster. Primer de tot instal·larem Kubernetes, juntament amb Kubeadm, aquesta última l'usarem com a *bootstrap*, és a dir per inicialitzar el cluster de nodes, bàsicament automatitza certs processos per facilitar la creació. Després instal·larem Kubelet, aquest haurà d'anar

instal·lat en tots els nodes, ja que s'encarrega d'inicialitzar els pods i els containers. Finalment Kubectl, per poder controlar el cluster per línia d'ordres. També hem d'instal·lar un *add-on* de xarxa perquè els diferents nodes es puguin connectar amb el Control-plane/Master. Hi ha un munt d'add-on's diferents, però nosaltres utilitzarem flannel.

Un cop que totes les eines s'han instal·lat ja podem començar a crear el nostre cluster amb tres nodes, un Control-plane/Master amb dos Workers. Primer hem d'inicialitzar el Control-plane/Master amb la següent comanda:

```
kubeadm init --apiserver-advertise-address=IP
--apiserver-cert-extra-sans=IP  --pod-network-cidr=RANG_IP
```

Si tot ha anat bé surt un missatge com el següent:

```
Your Kubernetes control-plane has initialized successfully!
```

Ara crearem un Token per a poder afegir els altres dos nodes de manera segura. Utilitzarem la següent comanda:

```
kubeadm token create --print-join-command
```

Aquesta comanda ens retornarà la comanda que utilitzarem per afegir els altres dos nodes com a Workers i té un format com la següent:

```
kubeadm join IP_MASTER:PORT_MASTER --token TOKEN
--discovery-token-ca-cert-hash sha256:HASH
```

La nostra infraestructura

Ja tenim tot en marxa, per tant, ja podem veure com és la nostra infraestructura. Com ja hem mencionat en l'apartat anterior, tenim un total de tres nodes diferents dels quals un és el Control-plane/Master (Obelix) i els altres dos són els Workers (Ciclope i Grendel). Per explicar de manera resumida, hem creat un cluster amb tres nodes, dos Workers on estan les diferents rèpliques del nostre Web Service, i un Master, que realitza tota la gestió del cluster i dels altres dos nodes. Aquest Master pot fer balanceig de càrrega, crear o destruir rèpliques del nostre Web Service, entre moltes altres funcionalitats. Cada node és una VM de la FIB.

Encara que en aquest i en els següents apartats mencionem algunes comandes, hem fet un glossari de les comandes més importants que hem utilitzat durant el treball. Es pot trobar a l'Annex 5.

Utilitzant la següent comanda es pot visualitzar tots els nodes del nostre cluster:

```
kubectl get nodes --all-namespaces
```

Cada node té uns pods per defecte, en el cas del Control-plane/Master (Obelix) són els següents:

- kube-apiserver: És un pod que serveix perquè puguis gestionar el cluster utilitzant crides des de l'exterior.
- etcd: És un pod que serveix per emmagatzemar les dades de tot el cluster.
- kube-scheduler: És un pod que actua com a planificador, que pot crear nou pods i asignar-los a diferents nodes.
- kube-controller-manager: És un pod que executa diferents controladors, com pot ser els dels nodes o els dels endpoints.
- kubelet: És un pod que s'encarrega de què els containers s'executin dins d'un pod.
- kube-proxy: És un pod que s'encarrega d'una part del Service que hi ha en el cluster.

En el cas dels Workers (Ciclope i Grendel) solament té els últims dos.

A més d'aquest pods per defecte, també hi ha el del *Add-on* que està en tots els nodes. Aquest s'encarrega de la comunicació entre el Control-plane i els Workers i d'algunes funcionalitats del cluster.

Podem veure quins pods té cada node, des de el Control-plane, amb la següent comanda (On posa NOM_NODE s'ha d'indicar el node):

```
kubectl get pods --all-namespaces -o wide --field-selector
spec.nodeName=NOM_NODE
```

Podeu trobar més informació dels diferents components d'un cluster de Kubernetes a: https://kubernetes.io/docs/concepts/overview/components/.

Deployment d'un service

Un cop Kubernetes està instal·lat i tots els seus pods funcionen correctament, es pot començar l'etapa del deployment d'un servei. Per definició, un deployment consisteix a escollir el cicle de vida, de manera declarativa, que volem que tingui una aplicació containeritzada. Per exemple, el nombre de rèpliques que tindrà, i la manera en què seran actualitzades. D'aquesta manera, serà el *back-end* de Kubernetes el que s'encarregui que sempre hi hagi el nombre de rèpliques funcionant que hem declarat amb el deployment.

Un exemple de la creació d'un deployment és el següent:

```
kubectl create deployment webapp --image=helloworld:1.0 --port=8080
--replicas=2
```

Amb aquesta comanda, indiquem que el nom del nou deployment serà "webapp", que l'aplicació s'anomena "helloworld" (cal tenir-la en Docker) i volem utilitzar la versió 1.0, el port 8080 i volem dues rèpliques (dos pods). Per comprovar que s'han creat correctament, podem fer servir la comanda `kubectl get deployments` per a veure quants pods estan en estat de READY.

Una de les característiques del deployment de kubernetes és la capacitat d'augmentar dinàmicament el nombre de rèpliques que té. D'aquesta manera, cada deployment és molt escalable. En el cas anterior, hem creat un deployment amb dues rèpliques. Si volguéssim augmentar-ho a quatre, executaríem la següent comanda:

```
kubectl scale --replicas=4 deployment/webapp
```

Una altra opció molt interessant és permetre a Kubernetes escalar dinàmicament el nombre de rèpliques segons el tràfic de la xarxa. Un exemple per definir que volem un mínim de quatre rèpliques i un màxim de deu és el següent:

```
kubectl autoscale deployment webapp --min=4 --max=10
```

Una característica de kubeadm és que no permet crear un deployment amb una imatge Docker local. És a dir, cal pujar-la a un repositori de Docker per així descarregar-la per poder crear el deployment. Això es veu reflectit en el camp `--image` de la comanda. En el nostre cas particular, va quedar així: `--image=16354425/pti02:latest`.

Els pods d'un deployment no són fixes, ja que Kubernetes els pot crear i esborrar en qualsevol moment perquè l'objectiu és sempre mantenir l'estat del deployment quan aquest va ser declarat. Per tant, els pods són extremadament volàtils. Això pot suposar un problema si uns pods ofereixen funcionalitat a uns altres, ja que els pods no sabrien comunicar-se amb els altres pods perquè són modificats dinàmicament. Per exemple, l'adreça IP d'un pod es modifica quan aquest és esborrat i se'n crea un de nou. La solució és el concepte de service.

Un service és una abstracció que defineix un conjunt lògic de pods d'un deployment. D'aquesta manera, la volatilitat física dels pods no afecta l'accés a nivell lògic, resolent així el problema del deployment.

La nostra creació d'un deployment és la següent:

```
kubectl expose deployment/webapp --type="ClusterIP" --port 8080
```

Una de les decisions de disseny més importants del service és escollir el tipus `--type`. N'hi ha quatre de diferents:

- ClusterIP: exposa el servei a nivell local. És a dir, fa un ús d'adreces IP privades que provoca que només el node Master i els nodes Workers tinguin accés al servei. És l'opció que ve per defecte.
- NodePort: exposa el servei a l'exterior. L'adreça IP que obté per defecte és la del node Master, i s'assigna un port del rang 30000-32767.
- LoadBalancer: exposa el servei a l'exterior utilitzant un *load balancer* del proveïdor de Cloud contractat. Un dels més usats és Azure.
- ExternalName: exposa el servei a l'exterior i se li assigna un nom DNS.

Les úniques opcions de servei que semblaven viables per la nostra infraestructura eren la de ClusterIP i la de NodePort. Vem intentar utilitzat la de NodePort per així poder obrir el servei a l'exterior, però va resultar impossible perquè els ports oberts de la màquina virtual de la FIB no coincidien amb el rang de ports que s'assigna al service. Per tant, l'única opció va ser emprar el tipus ClusterIP. L'explicació de com vem aconseguir exposar-lo a l'exterior està a l'apartat d'Integració i a l'Annex 4. En qualsevol cas, es pot comprovar l'estat del service mitjançant la comanda `kubectl get services`.

Dificultats i inconvenients

En aquesta secció, comentem els problemes que hem tingut amb Kubernetes i com els hem solucionat. Com que el primer i el tercer inconvenient són una conseqüència d'haver fet servir les VM de la FIB, aquests són comentats amb més detall a l'Annex 4.

En primer lloc, la infraestructura kubernetes té uns requisits de hardware. El que més ens ha afectat és el mínim de dues CPU. Les VM de la FIB en tenen una, però el tutor del treball va aconseguir que ens fiquessin dues CPU lògiques (continuàvem tenint només una de física), i això va permetre desenvolupar el projecte amb èxit.

En segon lloc, kubeadm no inicialitzava bé. Apareixia el següent error:

```
[kubelet-check] It seems like the kubelet isn't running or healthy.
```

```
[kubelet-check] The HTTP call equal to 'curl -sSL
http://localhost:10248/healthz' failed with error: Get
"http://localhost:10248/healthz": dial tcp 127.0.0.1:10248: connect:
connection refused.
```

El problema estava en el fet que hi ha un driver anomenat cgroup, i a Kubernetes valia `systems`, però a Docker valia `systemd`. La solució va ser crear un fitxer que forcés que Kubernetes també utilitzés `systemd`. Vem crear el següent fitxer a `/etc/docker/daemon.json`:

```
{
"exec-opts": ["native.cgroupdriver=systemd"]
}
```

Per acabar, vem reiniciar alguns serveis, i el problema es va arreglar:

```
sudo systemctl daemon-reload
sudo systemctl restart docker
sudo systemctl restart kubelet
```

En tercer lloc, vem tenir un problema amb el *add-on* de xarxa que vàrem escollir: calico. El problema semblava estar en la mateixa màquina de la FIB, que ja portava preinstal·lat un altre *add-on*: flannel. Això, probablement, causava col·lisions que impedien el bon funcionament de calico. La solució va ser formatejar la màquina i utilitzar flannel.

Integració

La part d'integració ha estat la més multidisciplinaria de totes. Tots els membres del grup hem participat en el procés.

Antecedents: Tor i K8s

Abans d'aquest punt, tant Tor com Kubernetes han d'estar instal·lats i en funcionament. Recomanem testejar les dues tecnologies per assegurar que tenen un comportament correcte.

Pel que fa a K8s, el tipus de service del que disposem és ClusterIP. Recordem que els dos tipus de serveis que vem valorar implementar van ser el NodePort i aquest. El primer obre el servei a l'exterior utilitzant ports molts alts que no tenim oberts a la VM de la FIB, mentre que el segon obre el servei a nivell local.

En un principi, vam pensar que NodePort era la millor opció. No obstant això, vem acabar fent servir ClusterIP i ens vàrem adonar que és ideal. Bàsicament, ClusterIP afegeix una capa de seguretat al servei web, ja que té adreçament privat i no és accessible des de l'exterior. Per tant, forcem al fet que l'única manera d'accedir al servei és mitjançant Tor, i això és just el que volem.

Desenvolupament de la integració

La integració ha consistit a utilitzar un proxy. Bàsicament, disposem d'una adreça IP (privada) i d'un port específic que identifica cada servei de tipus ClusterIP. Per tant, cal un pont entre les peticions que venen de la xarxa Tor i aquest servei que és intern a la xarxa Kubernetes.

Vàrem valorar diferents opcions, algunes de les quals mencionem en el següent apartat. El resultat va consistir a aprofitar el propi proxy que té Tor. D'aquesta manera, Tor rep les peticions i s'encarrega de redireccionar-les, internament, cap al servei. La resposta del servei passa també per Tor. Així, l'accés a la pàgina web és exclusiu de Tor.

Els canvis que vàrem dur a terme van ser al fitxer de configuració de tor, localitzat a `/etc/tor/torrc`. Vam afegir la següent línia:

```
HiddenServicePort 80 10.100.0.141:8080
```

Indica que el port 80 de la màquina rebrà les peticions de Tor, i aquestes seran redireccionades al port 8080 de l'adreça IP 10.100.0.141. Aquesta adreça i aquest port pertanyen al nostre servei.

Alternatives

Utilitzar el proxy de Tor és la manera que considerem més neta i senzilla d'integrar les dues tecnologies. No obstant això, hi ha altres opcions que vàrem valorar. En mencionarem dues: nginx i iptables.

El servidor web nginx és bastant utilitzat amb Kubernetes. Permet, entre d'altres, actuar de *load balancer*, de *Ingress Controller*, etc. En aquest cas, la usaríem com a proxy. Caldria instal·lar el pod i configurar-lo per a què redireccionés les peticions de Tor cap al servei de Kubernetes. Si Tor no tingués la funcionalitat de proxy, nginx hagués estat una possibilitat a tenir en compte.

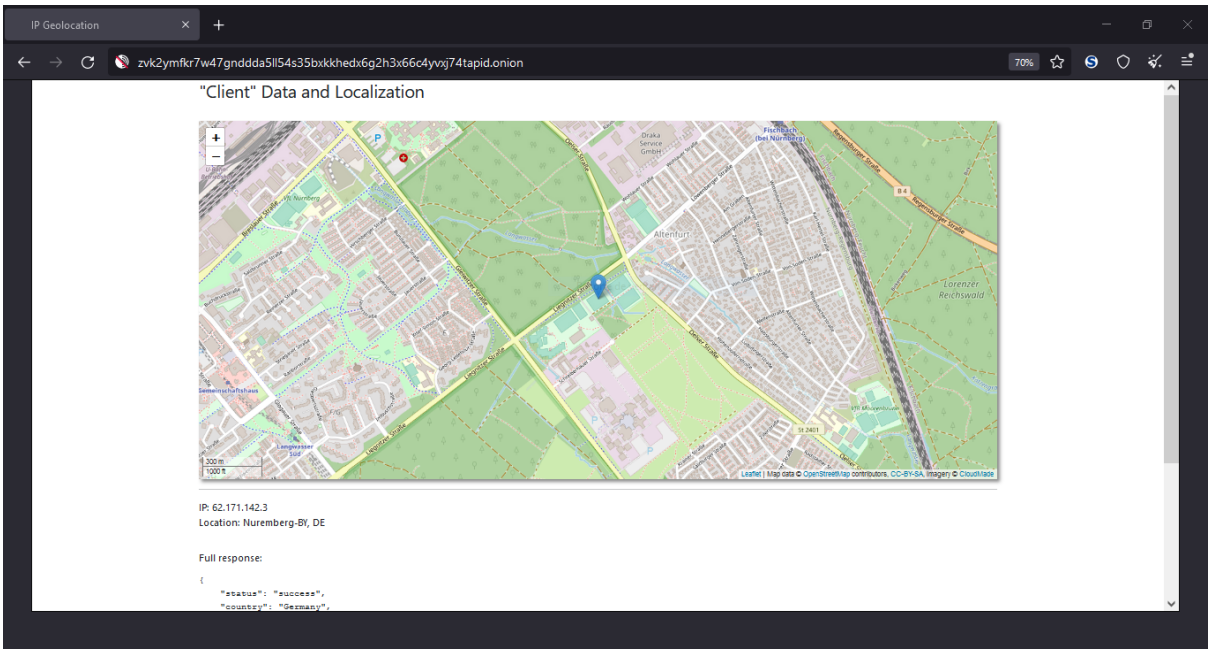
Per una altra banda, vàrem intentar fer servir iptables per fer la redirecció. No va ser possible; probablement perquè Kubernetes les modifica bastant. Considerem que Tor ha estat la millor alternativa, però iptables tampoc hagués estat una mala opció.

Web Service

Pel que fa al Web Service, el *front-end* de tota aquesta infraestructura, volíem fer una web senzilla, ja que no era l'objectiu principal d'aquest treball, però tampoc tan simple com un "hello world". Per això vam decidir fer una web amb un mapa i la informació relacionada del node que es connecta, per així demostrar que la ip del que es connecta al servei no és la del client.

Per fer tot això vam utilitzar dos recursos, el primer ip-api, una api per aconseguir la informació relacionada a la ip i el segon Leaflet, una llibreria *open source* de mapes interactius. No vam fer servir el típic mapa de Google Maps perquè creiem que trenca una mica amb tot aquest entorn d'anonimat que hem creat. A part també vam usar Bootstrap, una biblioteca *open source*, per donar-li un look més vistos a la web.

Finalment, el Web Service ens va quedar tal com es mostra a continuació:



Podeu consultar el Web Service utilitzant el navegador Tor amb aquesta adreça onion:

`zvk2ymfkr7w47gnddda5ll54s35bxkkhedx6g2h3x66c4yvxj74tapid.onion`

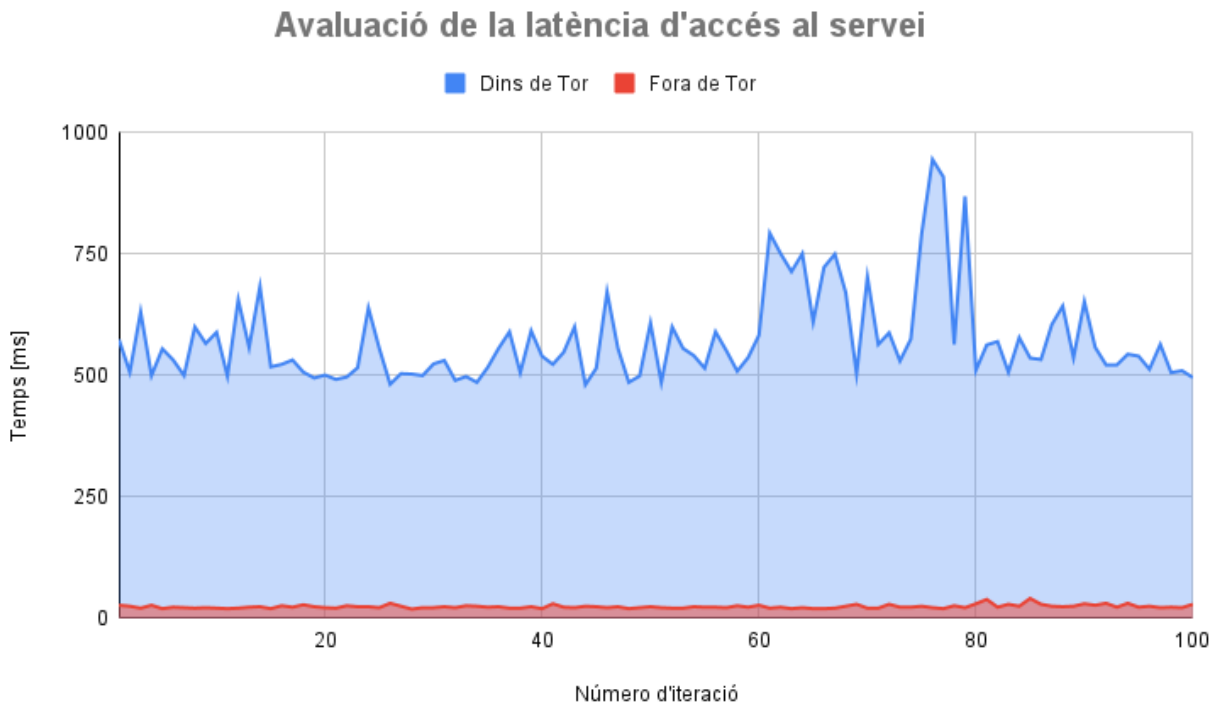
Avaluació de Rendiment

És indispensable conèixer la latència que afegeix la xarxa Tor a la navegació per tal de determinar si és viable oferir un servei dins seu. Per aquest motiu, hem calculat el temps d'accés al servei web des d'Internet (per Internet, ens referim a accés web fora de Tor) i des de dins de la xarxa Tor.

Un punt important a recalcar és que per accedir al servei des d'Internet hem hagut de modificar la infraestructura Kubernetes, ja que la que hem explicat en aquest treball no ho permet. El procediment està explicat a l'Annex 4.

Pel que fa a l'obtenció de les dades, hem utilitzat dos scripts que adjuntem amb el treball. Vàrem recaptar cent dades de cada tipus. Estan adjuntes a l'Annex 6.

El temps mitjà d'accés des d'Internet és de 24,03 ms, i des de Tor és de 573,18 ms. La gràfica que hem extret és la següent:

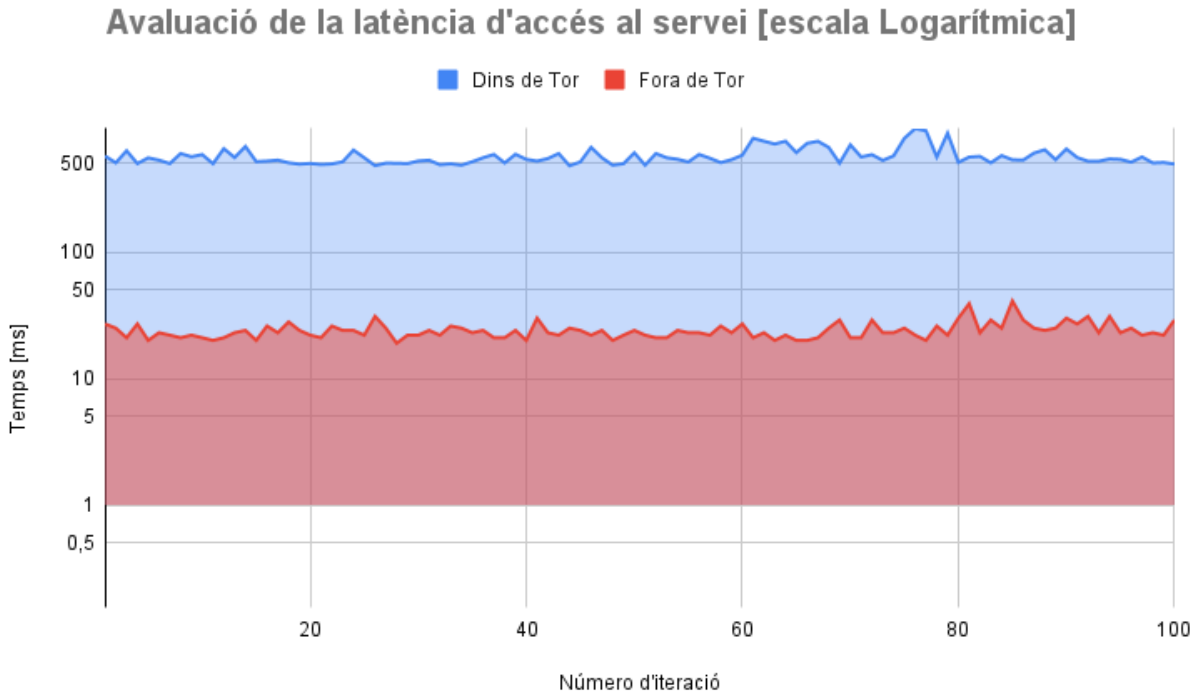


Font: elaboració pròpia

Amb el temps mitjà d'accés, podem obtenir la desviació estàndard, que és de 3,6 ms per Internet i 93,37 ms per Tor. És interessant calcular la proporció que suposa la desviació estàndard respecte al temps d'accés, per veure la constància del mateix. En els dos casos se situa entre el 15% i el 16%, sent Tor lleugerament més irregular.

Altres dades rellevants són la diferència entre els temps màxims i mínims. Pel que fa a Internet, el temps mínim és de 19 ms i el màxim és de 41 ms. Per tant, la diferència és de 22 ms, un 115% respecte al temps mínim. Per tant, l'augment és poc més del doble. Pel que fa a Tor, el temps mínim és de 481 ms i el màxim és de 945 ms. Per tant, la diferència és de 464 ms, un 96% respecte al temps mínim. No arriba a duplicar-lo.

Per veure el comportament amb més claredat, adjuntem una gràfica amb els temps d'accés, idèntica a l'anterior, però a escala logarítmica. Té com a objectiu mostrar la desviació del temps d'accés de manera més visible.



Font: elaboració pròpia

Per tant, el temps d'accés mitjançant Tor és **2.385,27%** vegades més lent que l'accés a partir d'Internet. No obstant això, la desviació del temps d'accés és molt similar proporcionalment en els dos casos.

Conclusions

Aquest treball ens ha servit per profunditzar en Tor i en Kubernetes. Són dues tecnologies molt populars amb les que volíem experimentar. El resultat ha estat que hem après molt i ens ha encantat utilitzar-les.

Hem pogut assolir tots els objectius originals del treball, que eren el d'integrar Tor amb K8s. Addicionalment, hem ampliat el projecte mitjançant una avaluació del rendiment i un atractiu *front-end* mitjançant una pàgina web que també permet a l'usuari veure que realment és anònim.

Desenvolupar una arquitectura tan complexa amb Tor i Kubernetes juntament amb la utilització de les VM de la FIB ha sigut un repte afegit. Creiem que potser hauria sigut més fàcil implementar-ho amb VM allotjades al núvol, ja que les de la FIB ens han aportat moltes restriccions, però creiem que ha valgut la pena.

Amb aquest treball, doncs, hem demostrat que és possible tenir un servei web en un clúster Kubernetes dins de la xarxa Tor. No obstant això, cal tenir en compte la latència que afegeix la xarxa. El temps d'accés és bastant constant, així que un servei que no requereixi baixes latències podria plantejar-se seguir l'arquitectura que hem desenvolupat en aquest projecte. Sens dubte, els principals avantatges de Tor i Kubernetes - anonimat i escalabilitat - poden ser molt interessants i convenients per molts serveis.

Una possible ampliació del treball seria disposar de nodes treballadors (Workers) separats físicament i que es comuniquessin entre ells utilitzant només la xarxa Tor. Això sí, caldria fer un nou anàlisi de rendiment per veure si la latència de la xarxa Tor faria inviable una arquitectura d'aquest estil.

La filosofia d'aquest treball era desenvolupar una infraestructura Kubernetes dins de Tor i documentar-ho perquè hi hagués una guia per la comunitat. Per tant, els tres membres del grup animem, a tothom que vulgui, a seguir els passos descrits en aquest projecte i replicar la infraestructura. Al cap i a la fi, aquest treball és una aportació a la comunitat.

Bibliografia

(n.d.). Stack Overflow - Where Developers Learn, Share, & Build Careers. https://stackoverflow.com/

.A Bhat, U. (2019, March 1). *How does Tor actually work?* Hacker Noon. Retrieved December 10, 2021, from https://hackernoon.com/how-does-tor-really-work-5909b9bd232c

Build & Deploy a Docker Image to Kubernetes Cluster. (2019, May 7). Linode. https://www.linode.com/docs/guides/deploy-container-image-to-kubernetes/

Building a Kubernetes 1.20v Cluster with kubeadm | by Chamod Shehanka | Platformer — A WSO2 Company. (2021, May 30). Medium. https://medium.com/platformer-blog/building-a-kubernetes-1-20-cluster-with-kubeadm-4b745eb5c697

Cómo instalar y usar Docker en Ubuntu 20.04. (2020, June 11). DigitalOcean. https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04-es

Dingledine, R., Mathewson, N., & Syverson, P. (2014). *Tor: The Second-Generation Onion Router*. svn-archive.torproject.org. Retrieved Desembre 10, 2021, from https://svn-archive.torproject.org/svn/projects/design-paper/tor-design.pdf

Docker Hub Quickstart. (n.d.). Docker Documentation. https://docs.docker.com/docker-hub/

examples. (n.d.). GitHub. https://github.com/kubernetes-client/python/tree/master/examples

The Guide to Kubeadm. (n.d.). Densify. https://www.densify.com/kubernetes-tools/kubeadm

Install CNI plugin - KUBERNETES. (n.d.). Calico. https://docs.projectcalico.org/getting-started/kubernetes/hardway/install-cni-plugin

Installing the NGINX Ingress Controller in the Kubernetes cluster. (n.d.). IBM. https://www.ibm.com/docs/en/control-desk/7.6.1.x?topic=kubernetes-installing-nginx-ingress-controller-in-cluster

Kubernetes Documentation. (2021, July 20). Kubernetes. https://kubernetes.io/docs/home/

Muñoz, J. D. (2018, May 9). *Instalación de kubernetes con kubeadm - PLEDIN 3.0*. Plataforma Educativa Informática - PLEDIN 3.0.

https://www.josedomingo.org/pledin/2018/05/instalacion-de-kubernetes-con-kubeadm/

Ruostemaa, J. (2021, June 2). *How to configure iptables on Ubuntu - Tutorial*. UpCloud. https://upcloud.com/community/tutorials/configure-iptables-ubuntu/

(n.d.). NGINX: Advanced Load Balancer, Web Server, & Reverse Proxy. https://www.nginx.com/

Srivathsav, R. (2018, April 14). *TOR Nodes Explained!. Block it, Track it or Use it. But…* | *by Raja Srivathsav* | *Coinmonks*. Medium. Retrieved December 18, 2021, from https://medium.com/coinmonks/tor-nodes-explained-580808c29e2d

Tasker, B. (2015, July 20). *Building a Tor Hidden Service From Scratch - Part 1 - Design and Setup*. Ben Tasker. Retrieved December 6, 2021, from https://www.bentasker.co.uk/documentation/linux/307-building-a-tor-hidden-service-from-scratch-part-1

Tasker, B. (2015, July 20). *Building a Tor Hidden Service From Scratch - Part 1 - Design and Setup*. Ben Tasker. Retrieved December 18, 2021, from https://www.bentasker.co.uk/documentation/linux/307-building-a-tor-hidden-service-from-scratch-part-1

The Tor Project. (n.d.). *The Tor Project*. Tor Project | Anonymity Online. Retrieved December 18, 2021, from https://www.torproject.org

Tous, R., & Freitag, F. (n.d.). microservices · master · pti / pti · GitLab. https://gitlab.fib.upc.edu/pti/pti/tree/master/microservices

Annexos

Annex 1: Comprovació de *torify*

En aquest annex farem una comprovació a fons de la funcionalitat de torify que ofereix tor i com realment la nostra connexió SSH està sent encaminada dins la xarxa i, per tant, està sent una connexió anònima.

El projecte de TOR ofereix una funcionalitat de poder buscar nodes i veure'n les característiques. Farem servir aquesta funcionalitat per veure si efectivament estem fent servir nodes de la xarxa.

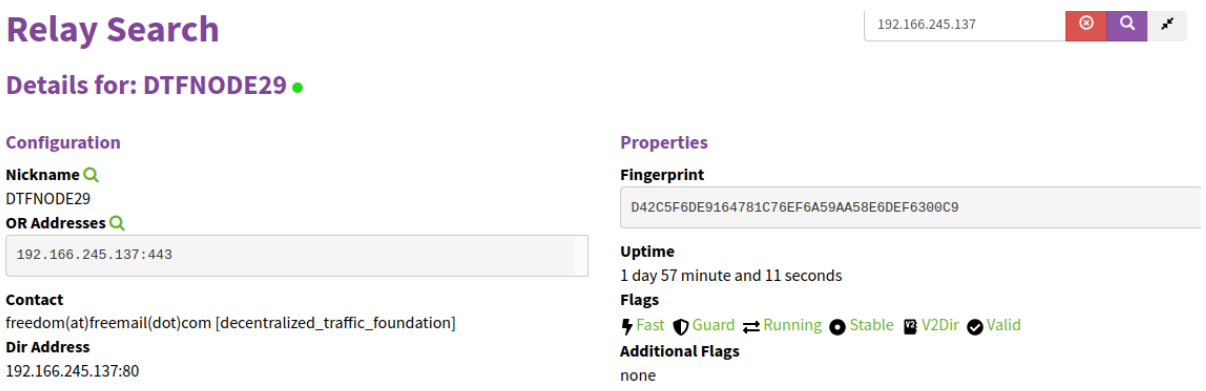
Per començar l'anàlisi accedim al nostre servidor mitjançant la comanda:

```
torify ssh alumne@nattech.fib.upc.edu -p 40369
```

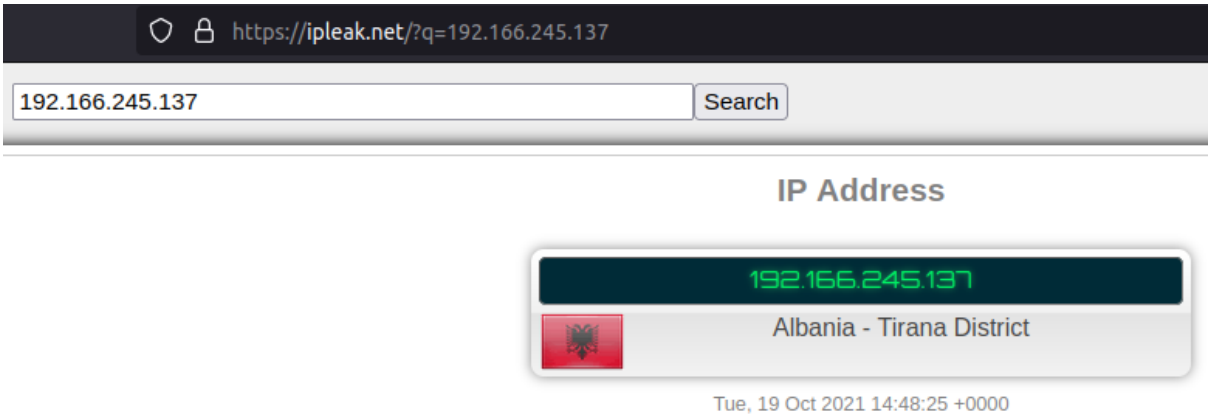
Primer analitzem a on es fa la petició:

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol dec
listening on enp5s0, link-type EN10MB (Ethernet), capture size 262144 b
16:28:36.728477 IP 192.168.1.20.43684 > 192.166.245.137.443: Flags [P.]
16:28:36.787791 IP 192.166.245.137.443 > 192.168.1.20.43684: Flags [.],
16:28:36.936854 IP 192.166.245.137.443 > 192.168.1.20.43684: Flags [P.]
16:28:36.936859 IP 192.168.1.20.43684 > 192.166.245.137.443: Flags [.],
16:28:36.937366 IP 192.168.1.20.43684 > 192.166.245.137.443: Flags [P.]
16:28:36.997372 IP 192.166.245.137.443 > 192.168.1.20.43684: Flags [.],
```

Sortida de tcpdump on s'observa l'intercanvi de paquets després d'executar SSH amb torify.



Resultats al consultar la IP anterior al registre de nodes de TOR: https://metrics.torproject.org/rs.html



Localització del servidor a on s'ha fet la petició.

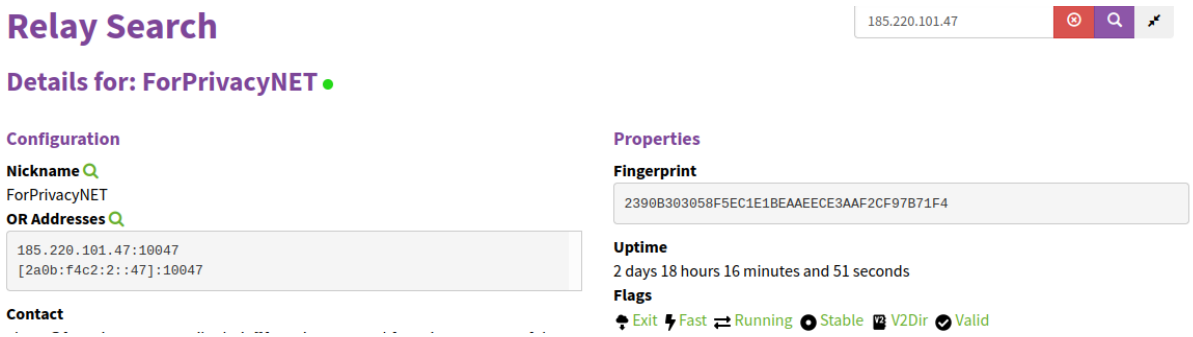
Podem observar com la màquina local (192.168.1.20) es comunica amb una màquina remota (192.166.245.137) que resulta que no és la màquina virtual on tenim el servidor.

En comprovar la IP al registre de nodes de TOR obtenim que es tracta d'un node de la xarxa i a més és el nostre *Guard node* (Srivathsav, 2018). Per tant, la nostra petició no va directament als servidors de la FIB sinó que passa per altres llocs, primerament, per Albània.

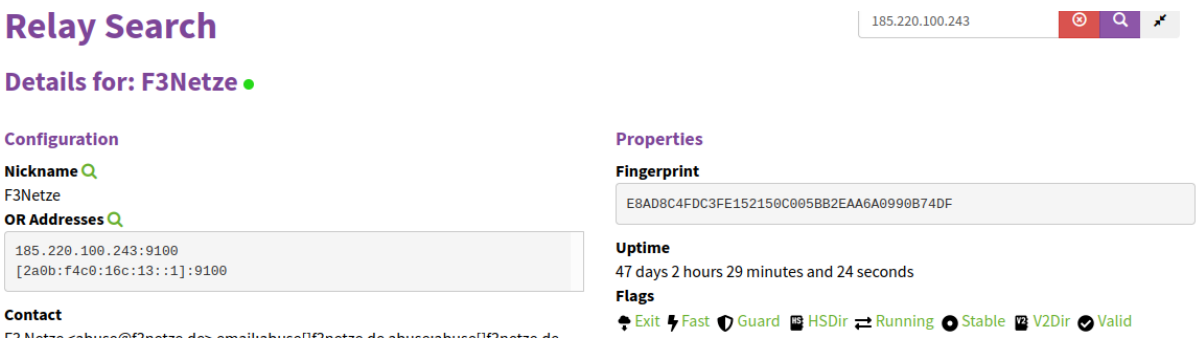
Ara anem a veure, des de la màquina virtual, des d'on hem iniciat sessió. La comanda *last* ens mostra els diferents inicis de sessió a la màquina sent el primer l'últim cop que s'ha iniciat sessió. A continuació analitzem des d'on rep la connexió la màquina virtual:

```
Last login: Tue Oct 19 14:56:20 2021 from 185.220.100.243
alumne@obelix:~$ last
alumne   pts/0        185.220.101.47   Tue Oct 19 16:42   still logged in
alumne   pts/0        185.220.100.243  Tue Oct 19 14:56 - 14:59  (00:03)
alumne   pts/0        23.129.64.144    Tue Oct 19 14:29 - 14:29  (00:00)
alumne   pts/0        23.129.64.144    Tue Oct 19 14:24 - 14:24  (00:00)
```

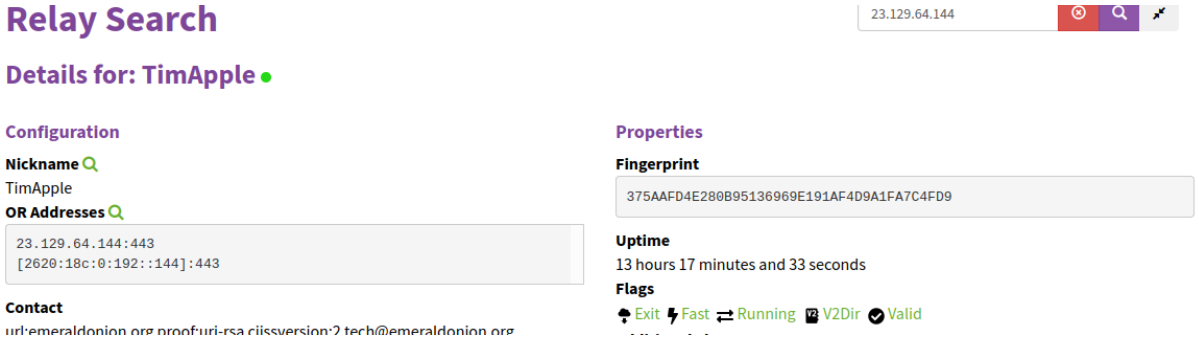
Sortida de la comanda last on s'observa des d'on s'han fet els últims inicis de sessió.



Resultat al buscar la primera IP al registre de nodes de TOR.

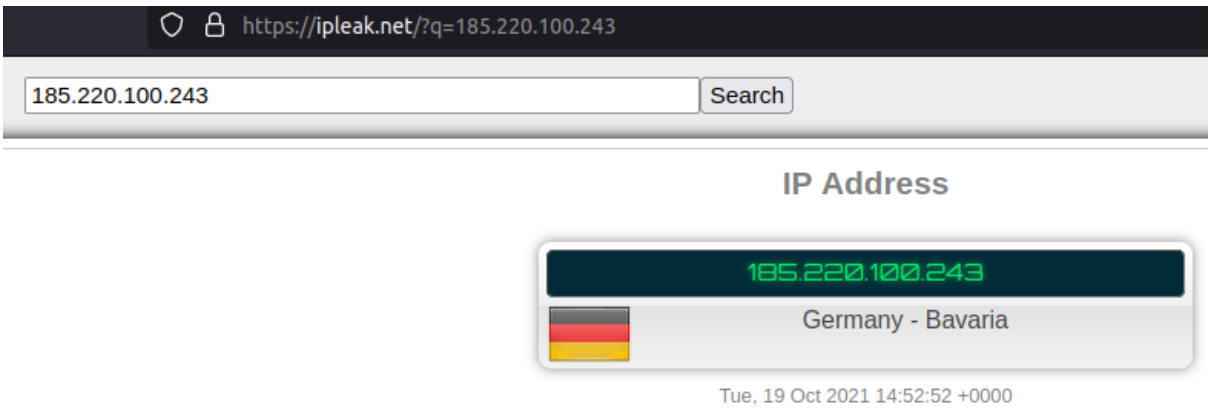


Resultat al buscar la segona IP al registre de nodes de TOR.

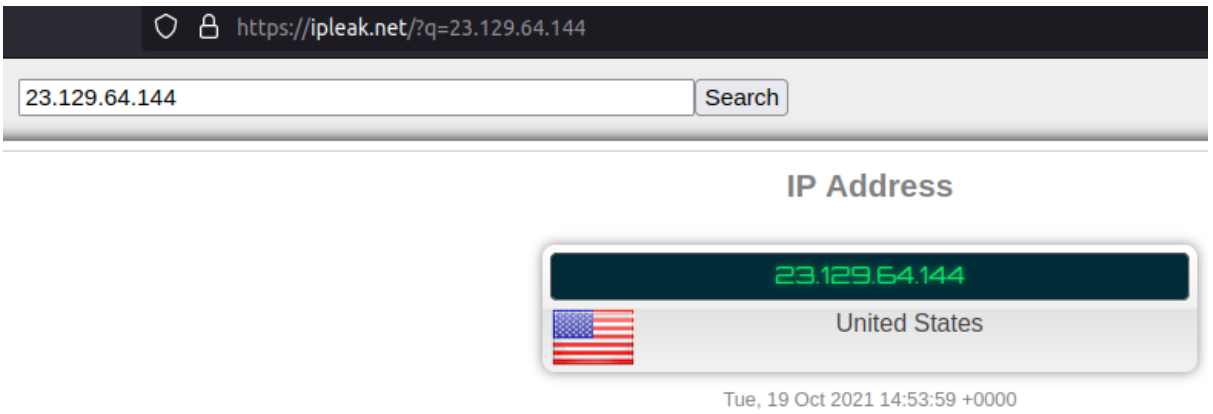


Resultat al buscar la tercera IP al registre de nodes de TOR.

Localització del primer servidor des d'on s'ha rebut la petició.



Localització del segon servidor des d'on s'ha rebut la petició.



Localització del tercer servidor des d'on s'ha rebut fet la petició.

Podem observar que els últims inicis de sessió a la màquina s'han fet des de 3 IPs diferents. Si procedim a comprovar la localització d'aquestes IPs observem que són d'altres països i que van canviant en el temps, ja que el circuit que establim dins la xarxa TOR també canvia.

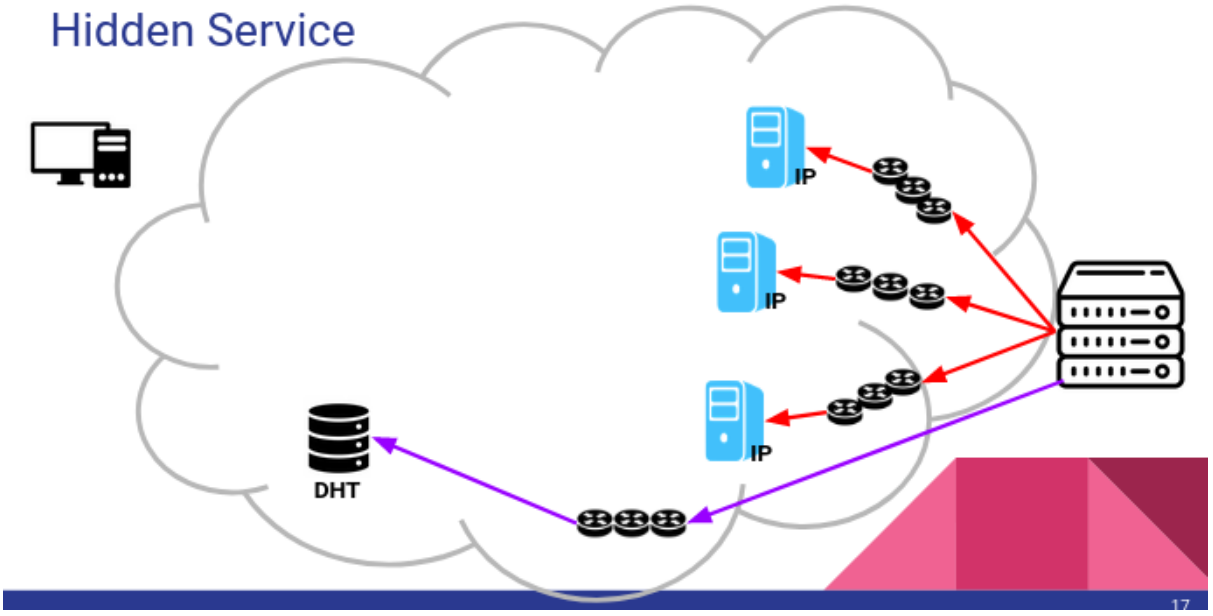
També, observem que els 3 servidors des d'on han arribat els logins són nodes de sortida de la xarxa. Per tant, podem confirmar que hem establert una sessió SSH a través de la xarxa TOR i que, per tant, les dues màquines als extrems (host i màquina virtual) no poden saber on ni qui està a l'altre costat.

Annex 2: Publicació del hidden service a la xarxa i connexió client-servidor

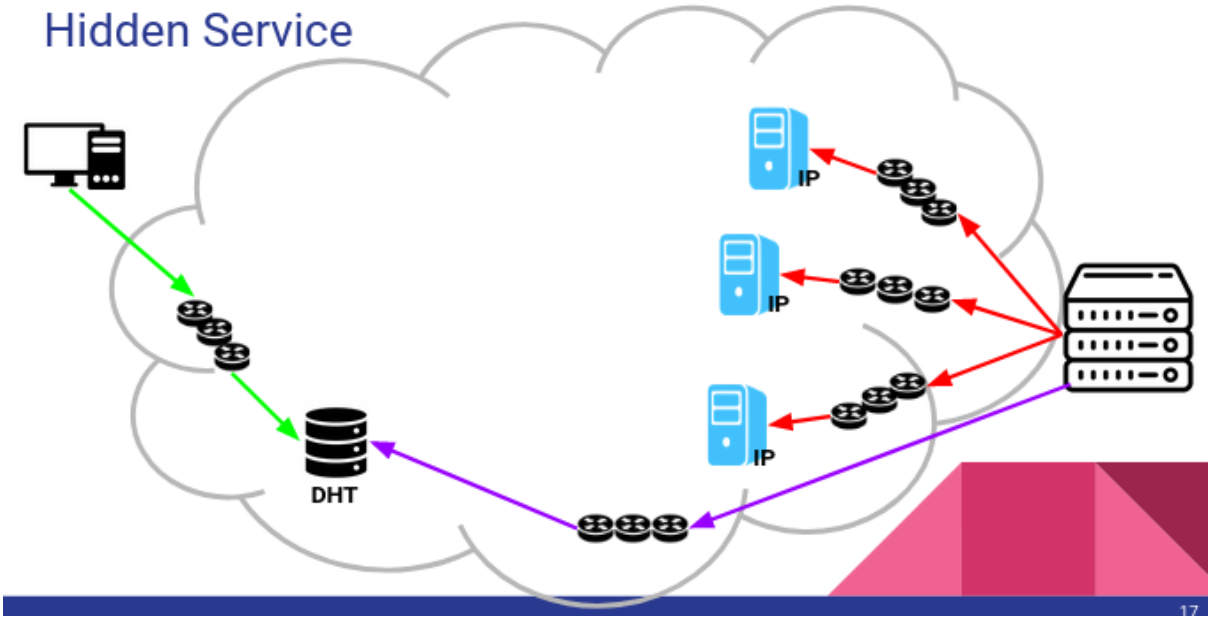
Quan creem un hidden service, el primer que fa el servei de Tor és crear una clau privada amb la seva corresponent clau pública. Després, el servei de Tor selecciona, per defecte, 3 nodes de la xarxa Tor perquè funcionin com a *introductory points* (IP al dibuix) del servei.

Un cop el servei té les adreces dels 3 *introductory points*, crea el que s'anomena el *hidden service descriptor*, que conté l'adreça pública del servei i les 3 adreces IP dels *introductory points*. Un cop té això, ho signa i ho envia a la taula hash distribuida (DHT al dibuix) de Tor perquè la informació sigui accessible des de diferents llocs.

Aquesta taula de hash està indexada amb les adreces *.onion* dels corresponents serveis i, per tant, si algú vol accedir al servei ha de conèixer l'adreça *.onion* i només podrà fer-ho si la donem a conèixer.



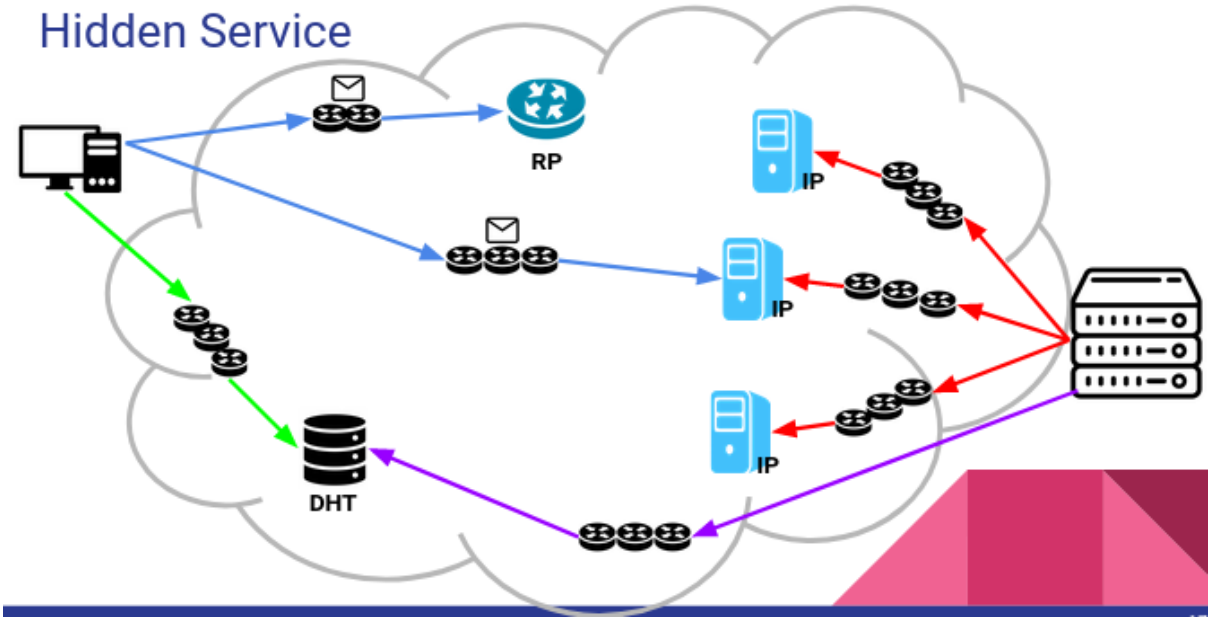
Un cop el client té l'adreça *.onion*, obté de la taula hash la clau pública del servei i les adreces dels *introductory points*.



Seguidament, el client selecciona un node de la xarxa que actuarà com a *rendezvous point* (RP al dibuix) entre el client i el servidor i és on s'executaran totes les peticions i el servei.

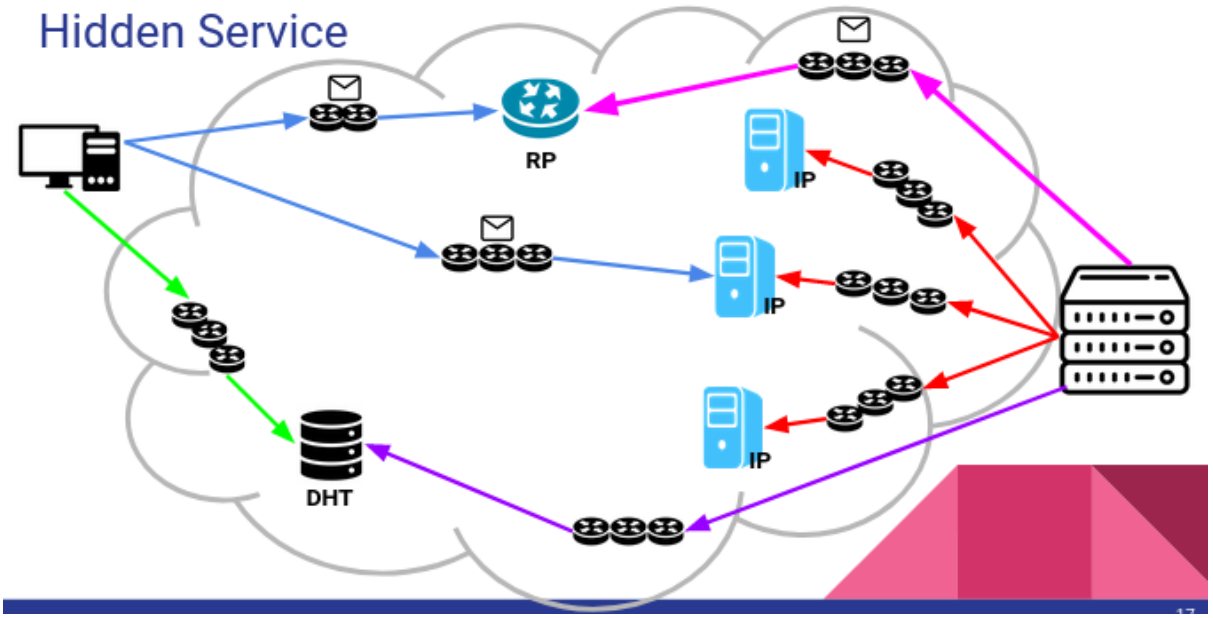
Un cop té seleccionat el *rendezvous point*, li envia un codi secret.

Un cop té això, envia un missatge a un dels 3 *introductory points* dient que es vol connectar al servei i també envia en aquest missatge l'adreça del *rendezvous point* i el codi secret d'abans.



Quan el servidor rep la petició de connexió pot decidir si contestar o no, si ho fa, es connecta amb el *rendezvous point* que ha rebut en el missatge i envia el codi secret.

El *rendezvous point* veu que té dues connexions que es volen connectar amb el mateix codi secret i procedeix a deixar realitzar la connexió.



En aquest punt, el client i servei es connecten i la connexió es realitza com una de normal, tot a través del *rendezvous point*.

Cal remarcar que totes les connexions amb els diferents actors es realitzen creant circuits de tres nodes com és propi a Tor (routers petits negres en el dibuix). D'aquesta manera, la connexió final entre el client i el servidor és de 6 nodes.

Per obtenir informació més detallada consultar bibliografia (.A Bhat, 2019) (Dingledine et al., 2014)

Annex 3: Instal·lació de Kubernetes

En aquest apartat, adjuntem diferents comandes per instalar Kubernetes, amb les diferents eines relacionades i amb totes les seves dependencies. Primer de tot actualitzarem el llistat de paquets del sistema, per asegurarnos de que quan instal·lem totes les eines i les seves dependencies siguin les més noves posibles.

```
sudo apt-get update
```

Per instalar l'eina docker juntament amb les seves dependencies utilitzarem les següents comandes:

```
sudo apt-get install apt-transport-https ca-certificates curl
software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
apt-key add -

sudo add-apt-repository "deb[arch=amd64]
https://download.docker.com/linux/ubuntu focal stable"

sudo apt-get install docker-ce
```

Per comprovar si s'han instal·lat correctament utilitzem:

```
docker --version
```

Per instalar Kubernetes i les diferentes eines de gestió del cluster i pods utilitzarem les següents comandes:

```
cat << EOF | sudo tee /etc/modules-load.d/containerd.conf
```

Dintre del fitxer afegirem:

```
overlay

br_netfilter

EOF

sudo modprobe overlay

sudo modprobe br_netfilter

cat << EOF | sudo tee /etc/sysctl.d/99-kubernetes-cri.conf
```

Dintre del fitxer afegirem:

```
net.bridge.bridge-nf-call-iptables = 1

net.ipv4.ip_forward = 1

net.bridge.bridge-nf-call-ip6tables = 1

EOF

sudo sysctl --system

sudo swapoff -a

sudo sed -i ‘/swap/s/^\(.*\)$/#\1/g’ /etc/fstab

sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg

echo “deb[signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main” | sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo apt-get install -y kubelet kubeadm kubectl
```

En el cas de que no volgem que s’actualitzin automàticament, per possibles problemes d'incompatibilitat en un futur, hem de afegir la següent comanda:

```
sudo apt-mark hold kubelet kubeadm kubectl
```

Per comprovar si s’han instal·lat correctament utilitzem:

```
kubectl version --client

kubelet --version

kubeadm version
```

Per instalar el *add-on*, flannel, utilitzarem les següents comandes:

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

Annex 4: Deployment d'un service a les VM de la FIB

Aquest annex té com a objectiu explicar les peculiaritats que tenen les màquines virtuals de la FIB que afecten, directament, a la infraestructura Kubernetes.

En primer lloc, vàrem tenir problemes amb els requisits hardware. Kubeadm[1] demana, principalment, un mínim de dues CPU i 2 GB de RAM. Les màquines de la FIB, per defecte, venen amb una CPU i 2 GB de RAM. Kubeadm no iniciava perquè exigia les dues CPU. Vam poder esquivar-ho iniciant kubeadm amb un flag que convertia l'exigència en un simple warning: `--ignore-preflight-errors=NumCPU`. Però després va resultar impossible iniciar el *add-on* (calico, ja que vem començar amb aquest), ja que també requeria dues CPU i no hi ha cap flag per evitar-ho. Vam parlar-ho amb el nostre tutor del treball, i va aconseguir que la FIB ens augmentés a dues CPU lògiques (manteníem només una CPU física) i a 4 GB de RAM. Va ser suficient per desenvolupar tota la infraestructura.

En segon lloc, va resultar impossible utilitzar el *add-on* de xarxa calico. En una màquina virtual en local, vàrem replicar els passos i sí que vam poder fer-ho. Investigant més a fons, vam trobar algun fitxer de flannel preinstal·lat a les VM de la FIB que, molt probablement, provocava col·lisions amb calico. Després de formatejar la VM de la FIB, vam instal·lar flannel i tot va funcionar correctament.

En tercer lloc, vàrem tenir problemes obrint el servei a l'exterior. Cal recordar que un service pot ser de quatre tipus diferents, però que només dos d'ells són aplicables a aquest entorn: ClusterIP i NodePort. Per una banda, ClusterIP permet obrir el servei a nivell local, mentre que NodePort a nivell extern.

Pel que fa a NodePort, no hi ha manera d'implementar-ho, ja que assigna un port dins del rang de 30000-32767, i les VM de la FIB només tenen oberts els ports del rang 8080-8089. Hem investigat molt i no hem aconseguit modificar el rang de ports que assigna; sembla que no és possible. Per tant, l'única solució seria fer forwarding de ports dins de la mateixa màquina.

Pel que fa a ClusterIP, el servei no pot ser exposat a l'exterior, així que caldria un mètode similar a l'anterior, que fes forwarding de l'adreça IP i del port del service. En aquest treball, Tor és l'encarregat de fer aquesta tasca.

[1] Els requeriments complets es poden trobar a: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/#before-you-begin

No és gens fàcil obrir un service a les VM de la FIB. Per aquest motiu, nosaltres proposem un altre mètode diferent, que és el que vam utilitzar per fer l'anàlisi de rendiment des de fora de Tor. Es tracta de jugar amb el propi proxy que té kubernetes (`kubectl proxy`). La idea consisteix a assignar al proxy l'adreça IP del node Master i un port dels que la màquina té oberts. Addicionalment, cal afegir un altre camp, `--disable-filter=true`, que desactiva el control de filtratge que fa el proxy. És insegur fer-ho, però és l'única manera que el servei funcioni. Però l'accés final és a nivell de pod, no de deployment. Les comandes són les següents:

- Obtenir el nom d'algun dels pods del deployment: `kubectl describe pod NOM_DEPLOYMENT | grep Name`

  Com exemple, en el nostre cas, el nom d'un pod era: `webapp-5c9cc48dcd-h75sr`

- Activar el proxy amb la IP del Node Master: `kubectl proxy --address=ADREÇA_IP_MASTER --port=PORT_OBERT --disable-filter=true &`

  En el nostre cas, ADREÇA_IP_MASTER=172.16.4.36 i PORT_OBERT=8080. Aquest port 8080 està obert a l'exterior mitjançant el port 40360.

- La URL per accedir és la següent:

  `http://nattech.fib.upc.edu:MAPEIG_PORT_OBERT/api/v1/namespaces/default/pods/NOM_POD/proxy/`

  En el nostre cas, va quedar:

  `http://nattech.fib.upc.edu:40360/api/v1/namespaces/default/pods/webapp-5c9cc48dcd-h75sr/proxy/`

- Un cop finalitzat, desactivar el proxy: `pkill kubectl`

Annex 5: Comandes de Kubernetes

En aquest apartat, adjuntem diferents comandes de Kubernetes que hem utilitzat durant el treball. N'hi ha moltes més, així que en cas de voler crear una infraestructura Kubernetes, recomanem consultar la llista completa[2].

Els camps a completar aniran subratllats i en majúscula.

Pods:

- Obtenir un llistat de tots els pods: `kubectl get pods --all-namespaces`
- Visualitzar els pods d'un node específic: `kubectl get pods --all-namespaces -o wide --field-selector spec.nodeName=NOM_NODE`
- Esborrar un pod: `kubectl delete pod NOM_POD`
- Veure estat de tots els pods: `kubectl describe pods --all-namespaces`
- Veure els logs d'un pod: `kubectl logs pods/NOM_POD`

Nodes:

- Obtenir tots els nodes: `kubectl get nodes`
- Fer fora un node del clúster: `kubectl delete node NOM_NODE`
- Veure estat d'un node: `kubectl describe node NOM_NODE`
- Obtenir la comanda (des del node Master) per afegir un node; cada vegada en crea una de completament nova: `kubeadm token create --print-join-command`
- Fer que tots els nodes puguin executar pods de treball: `kubectl taint nodes --all node-role.kubernetes.io/master- && kubectl taint nodes --all node.kubernetes.io/not-ready:NoSchedule-`
- Fer que un node puguin executar pods de treball (que sigui *scheduable*): `kubectl uncordon NOM_NODE`
- Obtenir informació de com estan tintats els nodes: `kubectl describe nodes | grep -i Taint`

[2] https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands

- Fer que un node NO rebi nova càrrega de treball (nous pods): `kubectl drain NOM_NODE`

Deployments:

- Creació d'un deployment: `kubectl create deployment NOM_DEPLOYMENT --image=NOM_IMATGE:VERSIÓ --port=PORT --replicas=NOMBRE_RÈPLIQUES`
- Visualitzar l'estat dels deployments: `kubectl get deployments --all-namespaces`
- Esborrar un deployment: `kubectl delete deployments NOM_DEPLOYMENT`
- Veure estat d'un deployment: `kubectl describe deployments NOM_DEPLOYMENT`
- Augmentar dinàmicament el nombre de rèpliques d'un deployment: `kubectl scale --replicas=NOU_NOMBRE_RÈPLIQUES deployment/NOM_DEPLOYMENT`
- Auto-escalar el nombre de rèpliques segons el tràfic: `kubectl autoscale deployment NOM_DEPLOYMENT --min=NOMBRE_MÍNIM_RÈPLIQUES --max=NOMBRE_MÀXIM_RÈPLIQUES`
- Obtenir el fitxer de configuració (yaml) d'un deployment: `kubectl get deployments/NOM_DEPLOYMENT -o=yaml`

Pujar una imatge a Docker Hub (per poder-la referenciar en el deployment):

- Iniciar sessió en Docker Hub: `docker login`
- Crear la imatge donat el Dockerfile: `docker build -t ID_USUARI/BRANCA PATH_DOCKERFILE`
- Pujar la imatge al repositori: `docker push ID_USUARI/BRANCA`

Services:

- Creació d'un service: `kubectl expose deployment/NOM_DEPLOYMENT --type="TIPUS_DE_SERVICE" --port PORT_SERVICE`

  `TIPUS_DE_SERVICE` pot ser: {`ClusterIP`, `NodePort`, `LoadBalancer`, `ExternalName`}
- Visualitzar l'estat dels services: `kubectl get services --all-namespaces`
- Esborrar un service: `kubectl delete service NOM_SERVICE`

- Veure estat d'un service: `kubectl describe service NOM_SERVICE`
- Veure serveis exposats a l'exterior: `kubectl cluster-info`
- Veure informació detallada dels services i dels nodes: `kubectl cluster-info dump`

Proxy[3]:

- Activar el proxy: `kubectl proxy &`
- La URL per accedir el pod és la següent: `http://nattech.fib.upc.edu:MAPEIG_PORT_OBERT/api/v1/namespaces/default/pods/NOM_POD/proxy/`
- Desactivar el proxy: `pkill kubectl`

Configuració:

- Aplicar cert fitxer .yaml de configuració: `kubectl apply -f NOM_FITXER`
- Debugar el clúster: `sudo journalctl -u kubelet | less`
- Netejar (esborrar) tot el clúster: `kubectl delete all --all --all-namespaces`
- Veure la configuració general del clúster: `kubectl config view`
- Obtenir la versió de kubectl: `kubectl version`

[3] Aquí es comenta d'una manera molt genèrica; per fer-ho a les VM de la FIB, recomanem seguir les instruccions explicades a l'Annex 4.

Annex 6: Dades de l'avaluació de rendiment

Número d'iteració	Fora de Tor [ms]	Dins de Tor [ms]
1	27	574
2	25	507
3	21	631
4	27	500
5	20	555
6	23	532
7	22	500
8	21	600
9	22	566
10	21	589
11	20	499
12	21	656
13	23	558
14	24	683
15	20	518
16	26	523
17	23	532
18	28	507
19	24	495
20	22	501
21	21	492
22	26	497
23	24	516
24	24	639
25	22	556
26	31	482
27	25	504
28	19	503

29	22	500
30	22	524
31	24	531
32	22	490
33	26	498
34	25	486
35	23	517
36	24	556
37	21	590
38	21	506
39	24	592
40	20	540
41	30	523
42	23	548
43	22	600
44	25	481
45	24	515
46	22	672
47	24	556
48	20	486
49	22	499
50	24	608
51	22	486
52	21	600
53	21	556
54	24	541
55	23	515
56	23	590
57	22	551
58	26	509

59	23	537
60	27	583
61	21	793
62	23	751
63	20	714
64	22	751
65	20	611
66	20	723
67	21	750
68	25	671
69	29	504
70	21	703
71	21	564
72	29	588
73	23	530
74	23	575
75	25	792
76	22	945
77	20	909
78	26	564
79	22	869
80	30	511
81	39	563
82	23	570
83	29	507
84	25	578
85	41	536
86	29	533
87	25	605
88	24	643

89	25	537
90	30	652
91	27	558
92	31	522
93	23	522
94	31	544
95	23	540
96	25	513
97	22	564
98	23	506
99	22	510
100	29	496
Mitjana:	**24,03**	**573,18**
Desviació estàndard:	3,685776647	93,37081282
Porcentatge de desviació:	15,34%	16,29%
Mínim:	19	481
Màxim:	41	945
Diferència:	22	464
Diferència respecte el mínim:	115,79%	96,47%
Tor més lent en una proporció de:	**2385,27%**	