Universitat Oberta de Catalunya

uoc.edu

Plataforma d'anàlisi de la xarxa Bitcoin

David Márquez Ruiz
Grau d'Enginyeria Informàtica

Cristina Pérez Solà

5 de juny de 2018

Als meus pares que han fet possible tot això, a la meva tutora, la Cristina Pérez Solà, que m'ha ajudat sempre que l'he necessitat i als meus fills Berta i David que m'han donat la força.

FITXA DEL TREBALL FINAL

Títol del treball:	*Plataforma per analitzar la xarxa Bitcoin*
Nom de l'autor:	*David Márquez Ruiz*
Nom del consultor/a:	*Cristina Pérez Solà*
Data de lliurament (mm/aaaa):	*06/2018*
Titulació o programa:	*Grau d'Enginyeria Informàtica*
Àrea del Treball Final:	*Seguretat Informàtica*
Idioma del treball:	*Català*
Paraules clau	*Bitcoin, docker, grafana*

Resum del Treball (màxim 250 paraules): *Amb la finalitat, context d'aplicació, metodologia, resultats i conclusions del treball*

Avui en dia el *Bitcoin* ja és una realitat, malgrat que està sotmès a grans sotracs produïts per l'especulació que persegueix qualsevol divisa o moneda real. La moneda digital ha vingut per quedar-se tal com va passar amb altres tecnologies com la fotografia digital.

Aquesta moneda es basa en el protocol que porta el mateix nom, Bitcoin, el qual està basat en la tecnologia *peer-to-peer.* Aquest sistema de transferència de dades té una sèrie d'avantatges com la connexió directa entre iguals (nodes) i la no-intervenció d'autoritat central (com serien els bancs) però també presenta inconvenients com seria la poca transparència en el funcionament de cadascun dels components de la xarxa.

A partir d'aquest inconvenient, trobem el nostre objectiu principal que serà definir una plataforma que sigui capaç d'establir el funcionament i el comportament d'aquests nodes que conformen la xarxa Bitcoin.

Per tant, realitzarem una aplicació que ens mostrarà una sèrie de dades que ens permetrà establir unes conclusions de com funciona la xarxa o almenys com ho fa en determinades situacions.

Abstract (in English, 250 words or less):

Today, Bitcoin already is a reality, in spite of is subjected to big bumps produced by the speculation that pursues any divisa or real coin. The digital coin has come to remain as it happened with other technologies like the digital photography.
This coin bases in the protocol that brings the same name, Bitcoin, which is based in the technology peer-to-peer. This system of transfer of data has a series of advantages like the direct connection between equals (nodes) and the no intervention of a central authority (how would be the banks) but also presents inconvenient how would be the little transparency in the operation of every of the components of the network.
From this inconvenient, find our main aim that will be to define a platform that it was able to establish the operation and the behaviour of these nodes that build the network Bitcoin.
Therefore, we will realise an application that will show us a series of data that will allow us to establish some conclusions of how works the network or to the less how does it in determinate situations.

Índex

Llista de figures

Codi

1. Introducció

1.1. Context i justificació del treball

El treball final que cal presentar per a finalitzar el grau d'informàtica està definit i explicat en aquest document o memòria. Vaig escollir el tema de la seguretat informàtica per què em van agradar les assignatures que vaig cursar referents a la criptografia i la seguretat en les xarxes informàtiques i sempre m'he vist atret pels forats de seguretat i vulnerabilitats que presenten els sistemes informàtics.
Vaig poder escollir diferents opcions però la que més em va cridar l'atenció va ser el desenvolupament d'una plataforma per analitzar la xarxa *Bitcoin*.
El *Bitcoin* i altres monedes digitals permeten establir transaccions entre els usuaris sense la necessitat d'una entitat que asseguri la correcta funcionalitat d'una moneda convencional, és a dir, sense la presència d'un tercer que permeti fer l'intercanvi de manera confiable.
Aquestes transaccions assoliran la confiança gràcies a la criptografia, fent servir signatures digitals. No caldrà, doncs, que cap institució central certifiqui que l'operació sigui vàlida i no sigui fruit d'una acció fraudulenta, com seria el cas de la doble despesa (una moneda es fa servir diverses vegades en diferents transaccions).
La descentralització s'aconseguirà mitjançant una xarxa *peer-to-peer* [1] (entre iguals), en la que cadascun dels nodes connectats certifiqui que les transaccions són correctes. Aquestes operacions seran guardades en un llistat o cadena de blocs (*block chain*) que compartiran tots els nodes on cadascuna mantindrà una marca de temps per tal de definir el moment exacte en què s'hi ha produït [2]. Aquesta cadena de blocs anirà creixent progressivament basant-se en una tècnica anomenada *Proof-Of-Work* o prova de treball, en la qual els nodes calculen un *hash* a partir de dos elements: la transacció i un conjunt de zeros necessaris per a elevar la dificultat del càlcul i així, fer computacionalment impossible realitzar-ho amb facilitat. Quan tots els nodes accepten el bloc que s'ha calculat amb el *hash* de més dificultat, aquest bloc s'afegirà a la cadena. A mesura que es creïn els blocs, variarà la dificultat d'obtenir el *hash* i per tant, implicarà més cost computacional.
Aquest deslliurament de l'autoritat central complicarà però, el control de la xarxa i el seguiment del comportament de cada node, que pot actuar com a ens honest o fraudulent. Per tant, seria interessant implementar algun sistema o mètode per saber en quina situació i condicions es troba la xarxa que formen els nodes.

1.2. Objectius del treball

Per tal d'acotar els conceptes, farem una introducció de la xarxa *Bitcoin* i explicarem les característiques principals del protocol i les funcionalitats que poden portar a terme cadascun dels nodes.
Com hem vist, l'estructura de confiança de la xarxa *Bitcoin* està basada en el concepte *peer-to-peer* que té com a conseqüència, junt amb la característica de xarxa distribuïda, que els components que la formen (nodes) no són visibles per a la totalitat de la xarxa. Per tant, es tracta d'una xarxa en la qual no tenim el control dels nodes. Això es complica quan no es pot conèixer en un moment determinat quins nodes estan connectats o quins no ho són. Aquests inconvenients, derivats de l'estructura de la xarxa Bitcoin, ens ha suggerit el primer dels objectius del treball. A partir de la necessitat de monitorar la xarxa definim com a objectiu principal del treball l'elaboració d'una plataforma que ens faciliti crear, configurar i controlar un conjunt determinat de nodes en la xarxa i així, definir la quantitat d'integrants, determinar les característiques, configuracions i les connexions a la xarxa que siguin necessàries. A causa de les complicacions que podem trobar en treballar amb la xarxa global *Bitcoin*, hem determinat estudiar la xarxa en mode *regtest*, que suposa la creació d'una xarxa local de proves on creem i manipulem els nodes *Bitcoin* a la nostra voluntat i així superar les limitacions que implica la creació de blocs i el control de les transaccions.
Un cop posada en marxa la plataforma, ens haurà de proporcionar un quadre on es visualitzin les dades, en temps real, que recollim de cadascun dels nodes (nodes connectats, nodes caiguts, balanç de cada adreça *Bitcoin*, blocs minats per cada node, etc.) i de tal forma, conèixer l'estat en què es troba la xarxa en un moment determinat.
Analitzant les dades que recol·lectem, hem d'arribar a una sèrie de conclusions que siguin significatives per determinar com funcionen els nodes Bitcoin en diferents situacions.
Així doncs podem resumir en els següents quadres, els objectius fixats.

Codi	Descripció
OP1	Introducció al Bitcoin
OP2	Implementació de la plataforma
OP3	Elaboració de conclusions

Per tal d'assolir els objectius principals s'han determinat una sèrie d'objectius intermedis:

Codi		Descripció
OP1		Introducció al *Bitcoin*
	Oi1.1	Estudi *Bitcoin*
OP2		Implementació de la plataforma
	Oi2.1	Selecció de l'arquitectura
	Oi2.2	Implementació de l'aplicació
	Oi2.3	Desplegament a la xarxa local
OP3		Elaboració de conclusions
	Oi3.1	Recopilació de resultats
	Oi3.2	Elaboració de les conclusions

Amb aquests objectius hem obtingut una sèrie de resultats que seran:

Codi	Descripció
R1	Informació i dades sobre la xarxa *Bitcoin*
R2	Aplicació *regtest* de la xarxa *Bitcoin*
R3	Recopilació de dades a partir de les gràfiques
R4	Memòria final del treball

1.3. Metodologia

Per a realitzar el projecte prendrem com a inici un codi subministrat pels tutors que utilitzant la plataforma *Docker* [3] ens permetrà aconseguir les dades necessàries a partir del desplegament de diversos nodes Bitcoin i les connexions que calguin per establir el funcionament normal de la xarxa. Aquest codi està implementat en *python* i marca els passos bàsics de creació i connexió de nodes de la xarxa. A partir d'ell hem anat variant i modificant el codi per tal d'analitzar el comportament dels nodes i les possibles opcions que podem analitzar per tal de comprovar com funciona la xarxa.
Aquestes dades es basaran en la plataforma utilitzada a *Statoshi* [4], on podem veure una sèrie de paràmetres que ens indiquen com es comporten els nodes i com reacciona la xarxa en tot moment. A mesura que ha avançat el projecte hem anat modificant el codi i filtrant les dades que hem pensat serien més adequades per analitzar.

1.4. Planificació del treball

Les tasques que caldrà dur a terme seran:

a. *Instal·lació i configuració de la plataforma Docker en Ubuntu Linux 14 LTS:*
   Configurar i comprovar el correcte funcionament de la plataforma sobre un sistema *Ubuntu Linux 14 LTS*.

b. *Inclusió de l'esquelet subministrat pels professors:*
   Comprovar que el codi proporcionat pels tutors funciona correctament i en cas que no sigui així, trobar una solució.

c. *Aconseguir el funcionament correcte d'un node:*
   Un cop utilitzat el codi a *Docker*, configurar un node *Bitcoin* i provar que tot funcioni bé.

d. *Definir les dades que seran necessàries per a realitzar el projecte:*
   Per tal de saber en quins paràmetres ens basarem per fer l'aplicació, haurem de definir quins punts i quines característiques haurem de trobar i centrar-nos en la realització del treball.

e. *Crear els nodes que siguin necessaris per obtenir un funcionament habitual de la xarxa Bitcoin:*
   Un cop verificat el correcte funcionament d'un node, crear els nodes necessaris en diferents contenidors per tal de crear una xarxa Bitcoin per poder realitzar l'anàlisi de la xarxa.

f. *Variació del codi original per tal d'aconseguir la funcionalitat en l'objectiu principal:*
   Per tal d'assolir els objectius principals, es realitzaran els canvis que calguin en el codi original. Hem d'assegurar una interfície visual que ens proporcioni en tot moment dades en temps real de la situació de la xarxa i les condicions en què es troben cadascun dels nodes.

g. *Comprovació del correcte funcionament de l'aplicació:*
   Comprovar que el nou codi és correcte i funciona tal com esperem.

h. *Execució de la nova plataforma i recopilació de les dades:*
   Una de les fites crítiques del treball que consisteix a provar l'aplicació i anar recollint resultats.

i. *Anàlisi de les dades i elaboració de conclusions:*
   Un cop recollits els resultats de l'aplicació, analitzar i arribar a conclusions per tal d'esbrinar que el treball ha tingut bons resultats.

j. *Recopilació de tots els resultats:*
   Recollir totes les dades i resultats dels objectius per tal de tenir una bona base per reflectir en la memòria del treball.

k. *Finalització de l'aplicació:*
Acabar de depurar la plataforma i deixar-la a punt per presentar, creant una ajuda si cal per explicar el funcionament.

l. *Realització de la memòria:*
Procedir a la creació de la memòria que és una altra fita crítica del treball.

m. *Realització de la presentació virtual:*
Crear una presentació audiovisual del treball final per a presentar al tribunal el treball final.

La planificació de les tasques serà la següent:

Nom de la tasca	Data inici	Data final	Duració (dies)
PAC 1	**21/02/2018**	**13/03/2018**	**21**
PAC 2	**14/03/2018**	**10/04/2018**	**28**
Instal·lació plataforma	14/03/2018	15/03/2018	2
Inclusió esquelet	16/03/2018	18/03/2018	3
Comprovació funcionament	18/03/2018	19/03/2018	2
Definició dades	20/03/2018	23/03/2018	4
Creació nodes	24/03/2018	26/03/2018	3
Modificacions codi	27/03/2018	07/04/2018	12
Comprovació funcionament	08/04/2018	10/04/2018	3
PAC 3	**11/04/2018**	**08/05/2018**	**28**
Execució i recopilació dades	11/04/2018	25/04/2018	15
Anàlisi de les dades i conclusions	26/04/2018	04/05/2018	9
Recopilació final dades	05/05/2018	08/05/2018	4
PAC 4	**09/05/2018**	**05/06/2018**	**28**
Finalitzar aplicació	09/05/2018	16/05/2018	8
Realització memòria	17/05/2018	05/06/2018	20
Creació presentació virtual	06/06/2018	15/06/2018	10

Figura 1. Planificació final.

El diagrama de *Gant* resultant serà el següent:

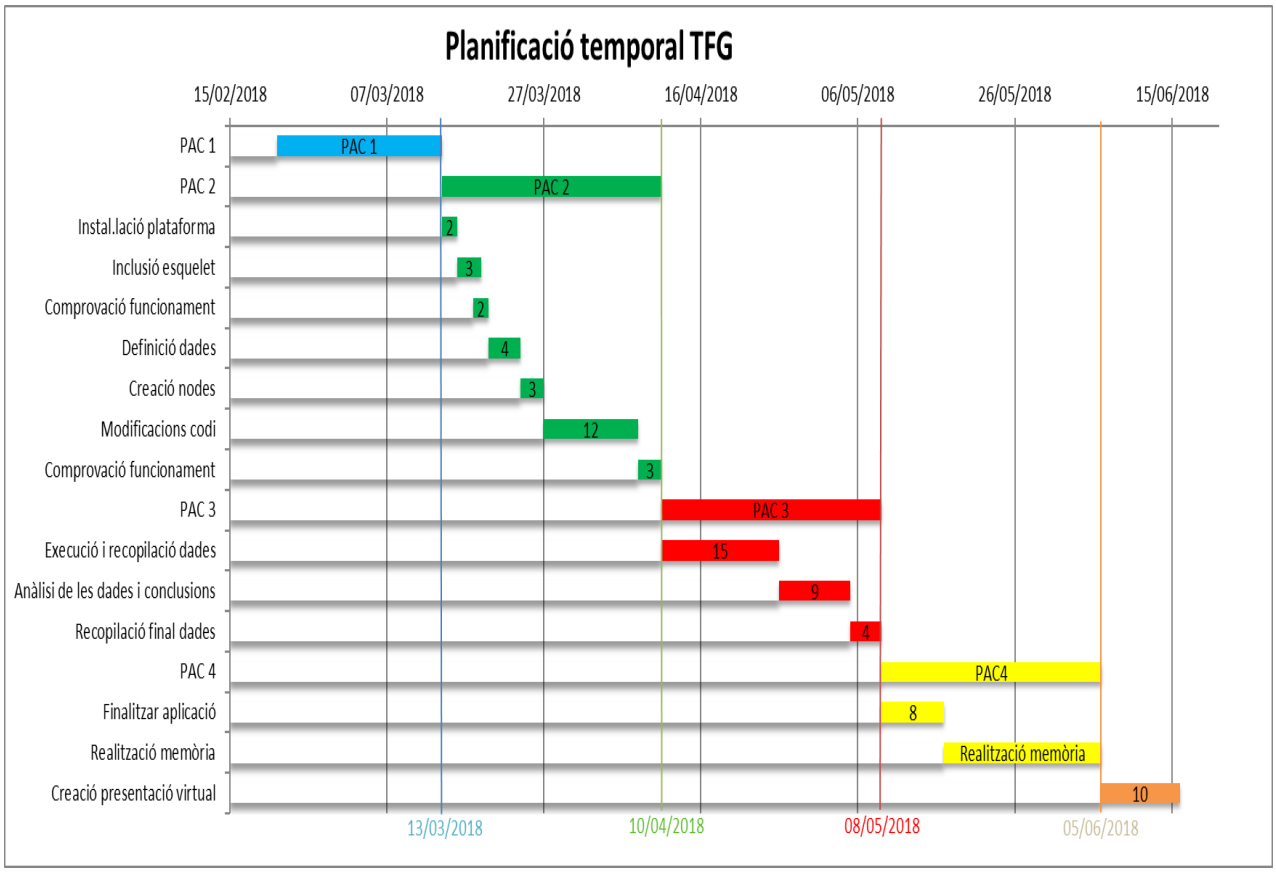

Figura 2. Diagrama de Gant.

2. Introducció al Bitcoin

2.1. Què és el Bitcoin?

El ***Bitcoin*** [5] és el conjunt format per la moneda electrònica, la xarxa que li dóna suport i el protocol pel qual s'interconnecten els nodes que la fan possible. Així, quan parlem de *Bitcoin* ens podem referir a qualsevol dels tres conceptes, però l'objectiu final serà ben senzill: utilitzar els avantatges digitals per tal d'aconseguir la funcionalitat d'una moneda que asseguri les característiques habituals de les monedes reals, que serien determinar el propietari temporal d'aquella moneda i la certesa de què s'utilitzi per fer un sol pagament en una mateixa transacció (evitar el doble pagament amb una mateixa moneda).
Així l'any 2009, un programador o grup de programadors, sota el pseudònim ***Satoshi Nakamoto*** [6] publica un document que detalla el funcionament d'aquest sistema digital financer, és a dir, el *Bitcoin* seria diners. Malgrat que no està reconegut com a moneda de curs legal per cap país en l'actualitat, ja s'utilitza com a mitjà de pagament de béns o serveis. Actualment, ja l'utilitzen més de 25 milions de carteres digitals o *wallets* [7] que ens donaria una idea dels usuaris que poden estar utilitzant el Bitcoin.
En pocs anys, aquest sistema ha produït un daltabaix al sistema financer tradicional, desplaçant la figura centralitzada dels bancs que tots coneixem, ja que es tracta d'un sistema **distribuït** i no requereix, per tant, una figura central que supervisi totes les transaccions de diners que es produeixin.
Però on tenim els bitcoins? Aquesta moneda no la tenim físicament com seria el cas de les monedes FIAT o monedes declarades oficialment de curs legal. Es necessita un moneder digital o ***wallet,*** identificat amb una adreça única. Així, les transaccions de bitcoins es realitzen entre aquests moneders. Aquest moneder no seria res més que un tipus de client de la xarxa *Bitcoin* que porta a terme les funcions de generació d'adreces i intercanvi comercial dels *bitcoins*. Des de la *wallet* es pot decidir transformar la moneda digital en moneda convencional o a la inversa, segons el valor establert per la llei de l'oferta i la demanda.

2.2. La xarxa Bitcoin i el seu funcionament:

Podem diferenciar 3 tipus de nodes que formen part de la xarxa. Aquests serien:

a. Nodes *broadcast*: són els nodes que emeten una transacció com podrien ser les *wallets*, és a dir, aquests nodes fan pública una nova transacció.

b. Nodes *relay*: són els que escampen o transmeten les transaccions entre la resta de nodes.

c. Nodes miners: nodes que realitzen les funcions de verificació de transaccions i creen nous bitcoins.

Figura 3. Nodes que conformen la xarxa.

El que el fa realment innovador, és que s'utilitza una **xarxa entre iguals** (*peer-to-peer*). No hi ha cap entitat que emeti diners, sinó que ho pot fer qualsevol integrant de la xarxa (nodes) seguint unes condicions bàsiques. El Bitcoin seria la unitat de compte dels balanços que consten en un llibre de comptabilitat o ***blockchain***, que és el sistema que permet conservar les transaccions produïdes al llarg del temps. Aquesta cadena de blocs distribuïda, possibilita emmagatzemar les dades de la forma més segura que existeix el dia d'avui. Això ho possibilita un procés anomenat **mineria**, que el que fa és verificar les transaccions que es produeixen mitjançant mètodes criptogràfics i maquinari avançat. A més de verificar les operacions, el procés de mineria crea nous *bitcoins* com a compensació del treball realitzat. La cadena va augmentant de manera que cada bloc que s'afegeix ha hagut que ser acceptat per consens, és a dir, que diversos nodes hagin corroborat que el bloc és correcte.

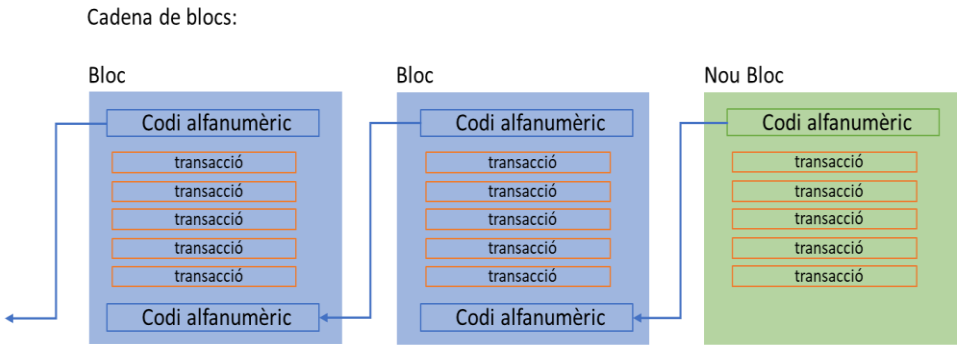

Figura 4. Esquema cadena de blocs.

Un **bloc** és un conjunt de transaccions verificades marcades amb un segell de temps (*timestamp*) junt amb informació addicional que s'inclouen a la cadena de blocs. Cadascun d'aquests blocs (excepte el bloc generatriu, que és el que inicia la cadena) està format per:

a. Un codi alfanumèric que enllaça amb el bloc anterior.
b. Les transaccions que s'inclouen.
c. Altre codi alfanumèric que enllaça amb el bloc següent.

El procés de mineria té com a objectiu trobar el codi que enllaci amb el següent bloc. Aquest codi es genera a partir de la informació del bloc anterior al qual se li aplica una funció resum o ***hash SHA256*** [8]. En aquest punt, ens podem trobar que algú pugui alterar la cadena o afegir transaccions fraudulentes, però el sistema ho té previst: com que el nou *hash* es basa en la informació del bloc anterior, és impossible acceptar un bloc amb un *hash* diferent del que ja ha estat acceptat amb anterioritat.
En el moment que algun dels nodes de la xarxa crea un *hash* vàlid, s'emporta la **recompensa** d'un nombre determinat de *bitcoins*, el *blockchain* s'actualitza i es notifica a tots els nodes restants que s'ha afegit un nou bloc. Aquesta recompensa és l'incentiu de crear nous blocs i permet continuar funcionant el sistema.
Un altre factor important és que el sistema inclou un algoritme que incrementa la **dificultat** de trobar el següent bloc (la dificultat seria la capacitat de computació que es requereix per produir una funció resum), ja que si no, s'arribaria fàcilment a crear nous *hashes* amb l'increment de poder de processament que implica l'avanç tecnològic i d'aquesta manera es crearien tots els *bitcoins* possibles, que són com a màxim **21 milions**.

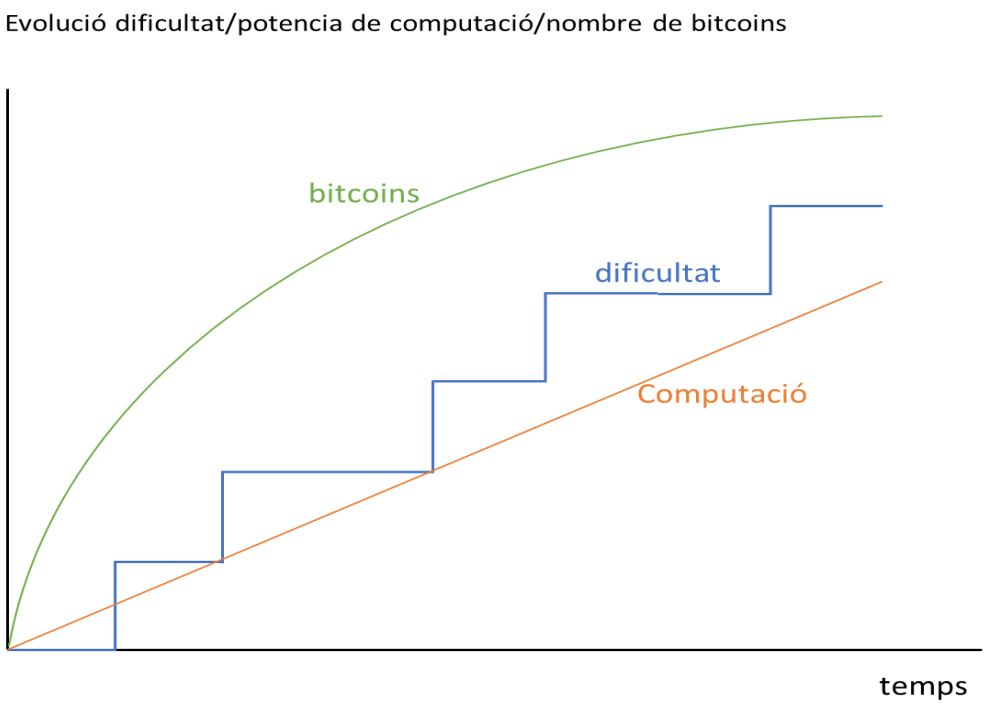

Figura 5. Gràfica de l'evolució amb el temps del nombre de bitcoins, la dificultat i la potència de computació disponible.

Durant el procés de mineria, els nodes miners no actuen sobre les dades que inclouen les transaccions sinó que canvien les dades del hash que finalitza el bloc. Això s'aconsegueix afegint un conjunt de zeros (que anomenem *nonce*). El procés de creació d'aquest *hash* no és trivial i constitueix una veritable prova de treball (***proof of work***). El *hash* que és vàlid s'envia a la resta de nodes i, un cop el confirmen diversos miners, es pot afegir a la cadena i, a continuació, el miner rep la recompensa. Un cop afegit el bloc ja és computacionalment impossible eliminar-lo de la cadena. Aquesta, actualment, ja ocupa 160 Gbytes d'espai de disc.

3. Tecnologies utilitzades:

Per tal de realitzar el projecte ha sigut necessari utilitzar una sèrie de tecnologies que passem a descriure a continuació:

3.1. Bitcoind:

Estem davant del codi font del *Bitcoin* [9]. El codi, que prové originàriament del que va presentar el 2009 *Statoshi Nakamoto, és* un programa executable des de la línia de comandes que possibilita totes les funcions del protocol *Bitcoin* però no inclou cap interfície gràfica. Es troba disponible sota llicència *MIT* en les plataformes *Windows 32/64, GNU/Linux i MacOS X*.
A partir de 2011 es presenta juntament amb *Bitcoin-Qt*, el qual presenta les mateixes possibilitats però que ja inclou capacitat gràfica, fent més amigable l'ús i, el 2014, amb la versió 0.9.0 passa a dir-se *Bitcoin Core*.

```
$ ps -o pid,cmd,etime -p $(pgrep bitcoind)
  PID CMD                                 ELAPSED
15891 ./src/bitcoind -printtocons 113-05:11:35
```

```
bitcoind-ncurses v0.0.20        87 peers    D/U: 43987.83 / 1020512.63 MB
bitcoind v0.9.3.0               0.00000000 BTC

347112: 0000000000000000039fc50baab664cedcdd58cac024a4d5fbdbe50c12ce9498
327416 bytes (319 KB)               Block timestamp: 2015-03-11 05:20:03
Transactions: 613 (534 bytes/tx)    Age: 00:05:33 (0d 00:06:16 by stamp)

Total block reward: 25.08133552 BTC (0.32% fees)
Fees: 0.08133552 BTC (avg 0.13290 mBTC/tx, ~0.25438 mBTC/KB)

Diff:        47,427,554,950         Hashrate (diff):  339,499 TH/s
Est (2016): ~47,973,930,656         Hashrate (2016):  343,410 TH/s
Est ( 144): ~53,880,889,317         Hashrate ( 144):  385,694 TH/s

Chain work: 2**82.405365            Mempool transactions:  3439
```

Està programat en C++ i, malgrat que va ser presentada una primera versió *Windows*, a partir de la versió 0.2.0 del desembre de 2009 va ser desenvolupat per a *Linux* i *MacOS X*.
En el nostre treball hem utilitzat la implementació sense interfície gràfica, la qual permet usar comandes remotes (*RPC*) i establir connexions, locals o remotes, entre els diferents nodes que creem amb el sistema de contenidors *Docker*. *Bitcoin* suporta connexions *JSON-RPC SSL* a partir de la versió 0.3.14. Aquestes comandes ens permeten realitzar processos i aconseguir dades dels nodes que formen la xarxa, de les transaccions i informació sobre els blocs que constitueixen la cadena de blocs.

Entre les comandes més importants descriurem les més utilitzades al treball:

- `addnode`: afegeix o elimina un node a la llista de nodes connectats.

- `getaccount`: torna el compte relacionat a l'adreça donada.

- `getaccountaddress`: retorna l'adreça *bitcoin* per tal de rebre els pagaments d'aquest compte.

- `getaddednodeinfo`*:* ens dóna dades sobre els nodes afegits.

- `getaddressesbyaccount`*:* retorna la llista d'adreces del compte en qüestió.

- `getbalance`*:* retorna el balanç en bitcoins del compte, si es subministra, o del total de comptes, si no s'especifica cap.

- `getblock`*:* ens dóna informació del bloc amb el *hash* donat.

- `getblockcount`: retorna el nombre de blocs de la cadena de blocs més gran.

- `getconnectioncount`: dóna el nombre de connexions d'un node determinat.

- `getdifficulty`: retorna la dificultat de computació com a múltiple de la dificultat mínima.

- `getgenerate`: retorna si el node està minant o generant *hashes*.

- `getmininginfo`: retorna un objecte amb informació relacionada amb la mineria:
  . *blocks*
  . *currentblocksize*
  . *difficulty*
  . *generate*
  . *hashespersec*

- `getpeerinfo`: retorna informació dels nodes connectats al node consultat.

- `gettransaction`: retorna un objecte amb informació de la transacacció amb txid:
  . *amount*
  . *confirmations*
  . *txid*

. time
. details

- `listaccounts`*:* llista que conté els noms dels comptes com a clau i els balanços com a valors.

- `listtransactions`*:* retorna les transaccions del compte proporcionat o de tots els comptes si no.

- `sendfrom`*:* envia una quantitat determinada a l'adreça donada i retorna la txID si tot és correcte.

- `stop`*:* atura el servidor *Bitcoin.*

3.2. Docker:

Així com una màquina virtual, *Docker* [3] aïlla els processos que es volen incloure mitjançant la creació de contenidors en un mateix sistema operatiu. La diferència és, però, que els recursos utilitzats i el temps de resposta del sistema són inferiors ja que es tracta d'un procés molt lleuger i poc exigent pel que fa a necessitats del sistema. La funció de *Docker* seria, per tant, la mateixa que les màquines virtuals però amb un cost en recursos significativament inferior a aquestes.

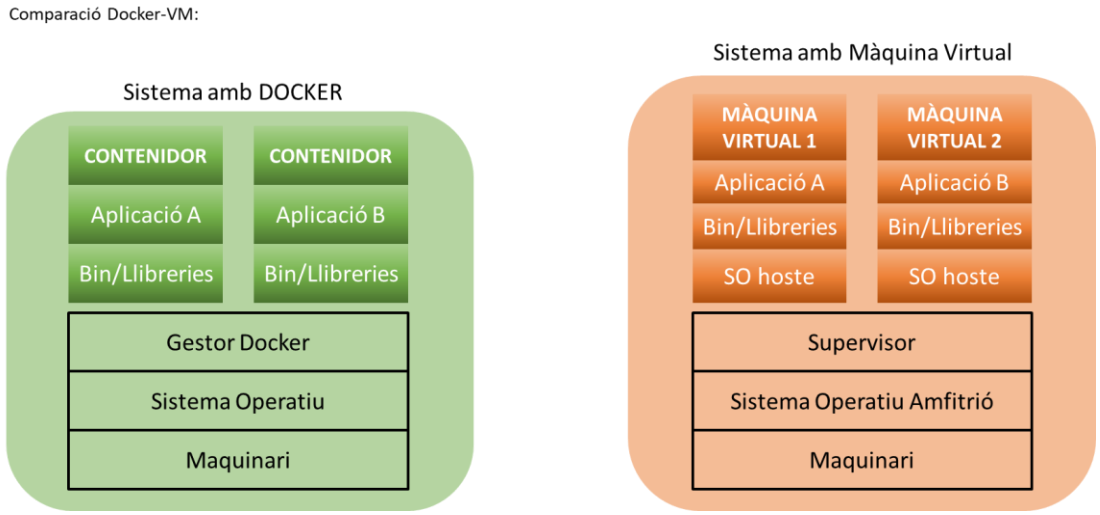

Figura 6. Comparativa contenidors-maquina virtual.

Els avantatges d'utilitzar aquest sistema serien:

a. **Aprofitament del maquinari**:
   Docker es basa en processos i funcions natives del *kernel* del sistema operatiu i, per tant, es poden executar diversos contenidors dintre la mateixa màquina sense penalitzacions en el rendiment, ja que tots utilitzen el mateix maquinari sense la intervenció de supervisors ni recursos extres.

b. **Portabilitat**:
   Com ja hem comentat anteriorment, *Docker* genera contenidors plenament autònoms. Aquest, gràcies al sistema de construir-los, permet compartir-los proporcionant, solament, el codi que hem usat per generar-los.

c. **Seguretat**:
   Els contenidors estan, per definició, aïllats del sistema operatiu i dels components que configuren el maquinari. Per tant, podem arriscar-nos a portar a terme diferents actuacions sense trobar-nos amb problemes derivats.

d. **Desplegament àgil**:
Com que no s'utilitzen agents supervisors ni maquinari addicional, el temps de desplegaments i posada en marxa d'un contenidor serà de pocs segons, a diferència del que passaria en engegar una màquina virtual.

Integració Docker:

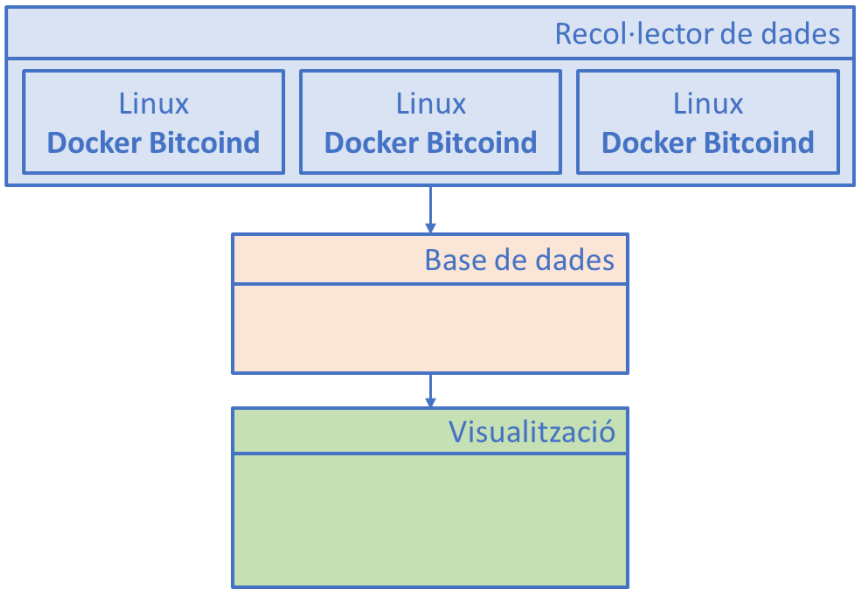

Figura 7. Integració Docker.

El funcionament del sistema *Docker* està determinat per dos components: els **contenidors** pròpiament i les **imatges**:

Una imatge seria la captura de l'estat d'un contenidor, és a dir, seria una plantilla en la qual es basa un contenidor. En ella podríem tenir el sistema operatiu, el programari i tot el necessari per utilitzar els contenidors. Les imatges s'utilitzen, per tant, per crear contenidors i, en principi, no canvien en l'execució dels processos. Hi trobem gran varietat d'imatges i les podem classificar a partir de serveis ja instal·lats i configurats. Per altra banda, trobem el *DockerFile*, que seria un arxiu especial que s'encarrega de configurar i crear les imatges amb els components específics que ens interessen en cada cas.
Els contenidors serien les instàncies creades a partir de les imatges i en ells s'executen les aplicacions. A partir d'una mateixa imatge podem executar diferents contenidors: en el nostre cas, a partir de la imatge *btc_testbed*, creem diversos contenidor *btc_nx* que seran els nodes que conformen la xarxa a estudiar.

Imatges i contenidors utilitzats:

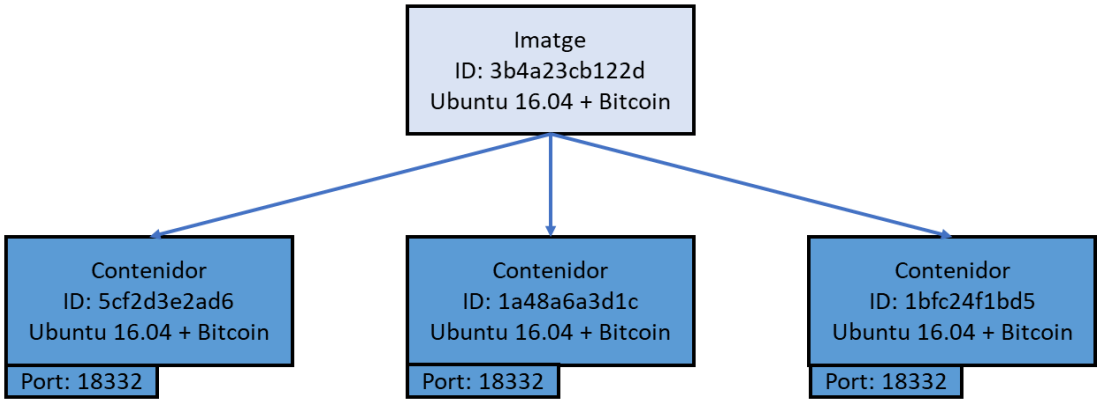

Figura 8. Relació imatge-contenidors.

Docker permet conservar la informació fins i tot quan es destrueixen els contenidors mitjançant els *volums*, que són apuntadors a ubicacions d'emmagatzematge de dades que permeten modificar, eliminar o reconstruir els contenidors tantes vegades com sigui necessari i, a més, conservant les dades que estaven presents anteriorment. Aquests volums són propis de cadascun dels contenidors.

3.3. Grafana:

Grafana [10] és un programari *open-source* o codi obert que permet la presentació gràfica de dades donades en sèries temporals, és a dir, de seqüències de dades, observacions o valors mesurats en determinats moments i ordenades cronològicament. S'utilitza normalment per a visualitzar, de forma elegant, sèries de dades en l'anàlisi d'infraestructures i aplicacions determinades.

Integració Grafana:

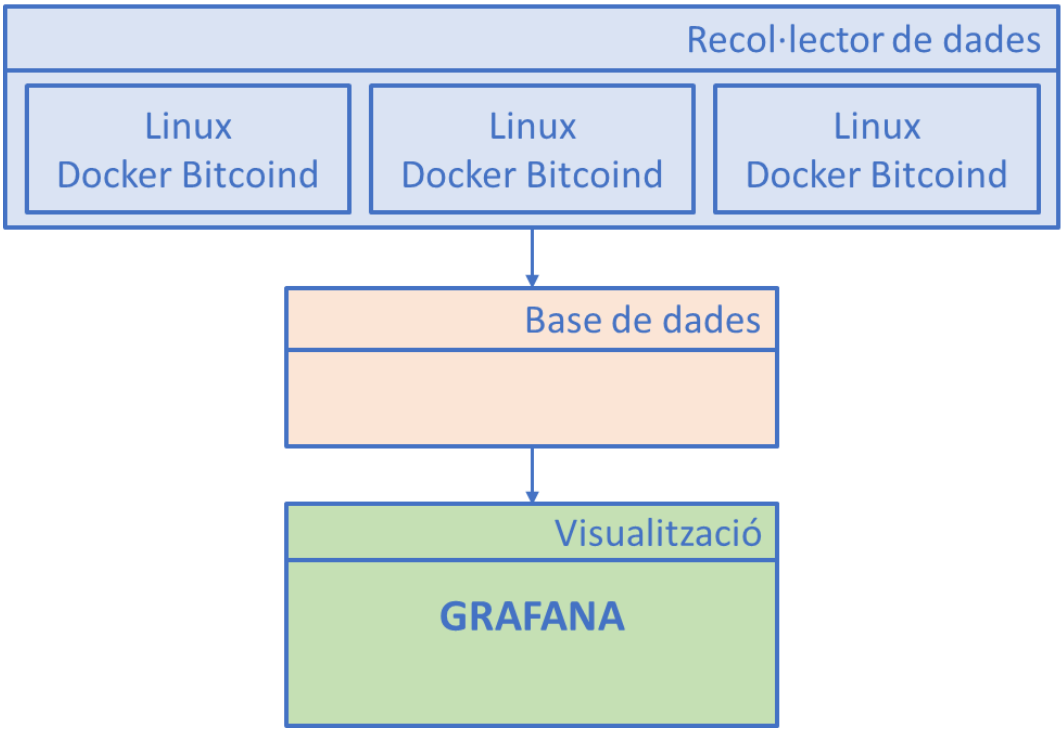

Figura 9. Integració Grafana.

Sol anar acompanyada d'altres aplicacions o *plugins* que la complementen, com *graphite* o, en el nostre cas, *influxDB.* Aquests plugins poden ser del tipus:

a. ***Panel***: seria el *dashboard* o quadre de comandament. Podem crear els que calguin.

b. ***Datasource***: subministra les dades als quadres. Normalment són bases de dades. Cadascun dels *datasources* té el seu editor per realitzar les consultes i és específic per a cada tipus de dades que es vulgui tractar. En principi, es permeten dades de *InfluxDB*, *Graphite*, *OpenTSDB*, *Elasticsearch* i altres.

c. ***App***: serien aplicacions completes dintre de *grafana* que poden incloure els seus quadres, fonts de dades i altres característiques.

El punt fort d'aquest *software* és que podem desentendre'ns de la complexitat de crear escenaris i les seves característiques, ja que proporciona una interfície gràfica (*dashboards*) prou atractiva i senzilla d'utilitzar.
També dóna la possibilitat de rebre i controlar alertes i missatges associats als gràfics creats a partir de les fonts de dades utilitzades.
Grafana, un cop instal·lat, podem visualitzar els seus *dashboards* a través d'un navegador, utilitzant la IP configurada, en el nostre cas serà el *localhost*, i el port 3000 per definició.

3.4. InfluxDB:

Per tal de dur a terme el projecte ha sigut necessari una base de dades que recollís la informació procedent dels contenidors *bitcoin* que hem creat. Després de diverses proves amb altres bases de dades com *Graphite*, ens hem decantat per ***InfluxDB*** [11].

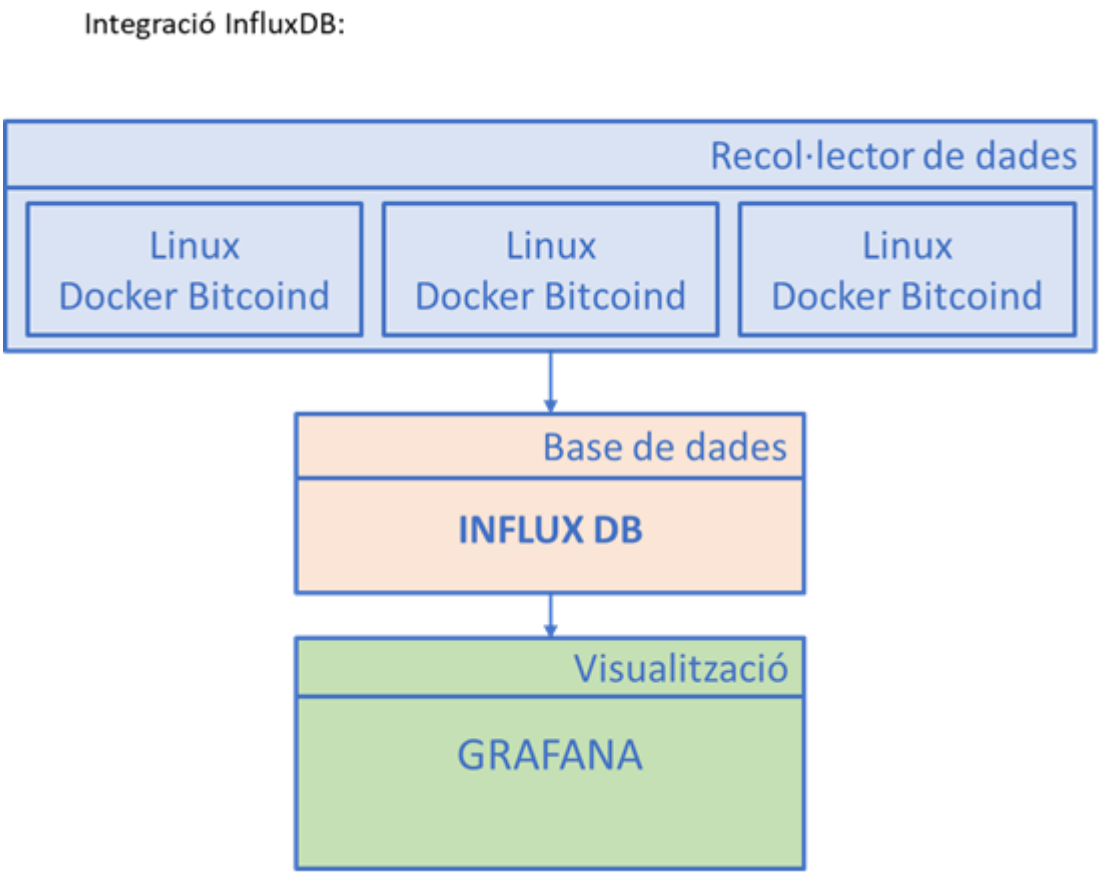

Figura 10. Integració InfluxDB.

Aquesta base de dades és àmpliament utilitzada per emmagatzemar i gestionar dades que depenen, en gran part, de la variable temps. *InfluxDB* s'encarrega de guardar les dades juntament amb el *timestamp* corresponent. Això facilita la ràpida visualització en la interfície gràfica i una bona presentació segons el pas del temps. Ha estat programat en llenguatge *go* [12], llenguatge *open source* desenvolupat per *Google* que permet la interacció via API HTTP (JSON) i interfície web, a més, les dades es gestionen amb un llenguatge basat en SQL.
Una consulta exemple seria:

```
SELECT * FROM "foodships"
```

Podem comptar també amb una llibreria de client, *influxdb-client*, que ens facilita realitzar consultes a la base de dades:

```
clientDB = InfluxDBClient('localhost',
'8086', 'username', 'password', 'nomDatabase')
```

Comptem amb les *queries* tradicionals del llenguatge SQL com seria el *SELECT, WHERE, GROUP BY, INTO, ORDER BY* i altres.
Trobem també una sèrie de consultes clau per controlar *InfluxDB*:

a. *CREATE DATABASE*: crea una base de dades.
b. *SHOW DATABASES*: mostra les bases de dades que tenim creades.
c. *SHOW MEASUREMENTS*: mostra les mesures emmagatzemades de la base de dades.
d. DROP DATABASE: elimina una base de dades.
e. DROP MEASUREMENT: esborra les dades d'una mesura que s'ha afegit anteriorment.

La informació es passa a la base de dades a partir dels contenidors *bitcoin* mitjançant una cadena JSON com seria aquesta:

```
json_body = [{"measurement": 'altitud',"time":
(time.strftime('%Y-%m-%d %H:%M:%S')),"tags":
{"muntanya": c},"fields": {"value": nova_altitud}}]
```

On tenim els camps claus *measurement, time, tags i fields*. Aquests camps es guardaran a la base de dades mitjançant la comanda *write_points*.

3.5. Tkinter:

Tkinter [13] és la llibreria GUI estàndard del llenguatge de programació utilitzat en el projecte, ***Python*** [14]. S'utilitza, per tant, per donar un aspecte més amigable a una aplicació mitjançant una interfície gràfica. Permet la creació d'objectes com finestres, botons, etiquetes, etc. És, així, una llibreria orientada a objectes.
Els avantatges que té Tkinter són:

a. **Senzillesa**: permet crear la interfície gràfica sense sortir de *Python* o complicar-se utilitzant altres extensions com *Glade* o *GTK3*.
b. **Potència**: sense abandonar el llenguatge *Python* tenim una sèrie d'objectes que ens donen la possibilitat de definir una interfície senzilla però prou potent per assolir les necessitats de la majoria de les aplicacions.
c. **Migració**: un cop realitzat un treball permet fàcilment canviar a altres entorns gràfics, ja que el codi *python* s'ha modificat directament i és fàcil tornar al codi original.

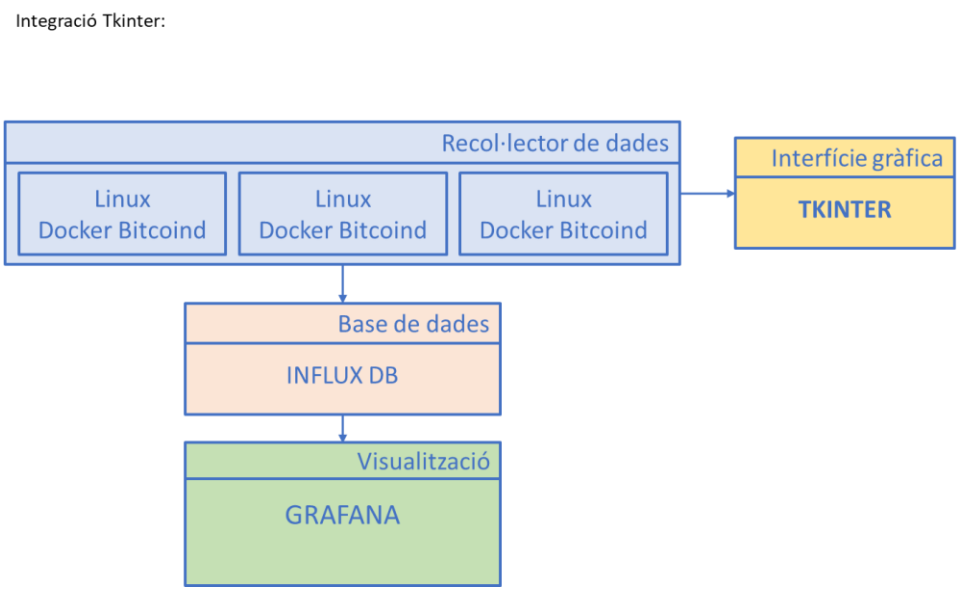

Figura 11. Integració Tkinter.

Cadascun dels objectes que es necessiten es van creant dintre d'un bucle repetitiu, que solament es trenca quan es destrueix l'objecte arrel o responsable d'iniciar aquest bucle.

4. Disseny de la plataforma:

4.1. Arquitectura de l'aplicació:

Com ja hem anat veient en el capítol anterior, els components utilitzats es van integrant en el programa de tal forma que ens donen una arquitectura semblant a aquesta:

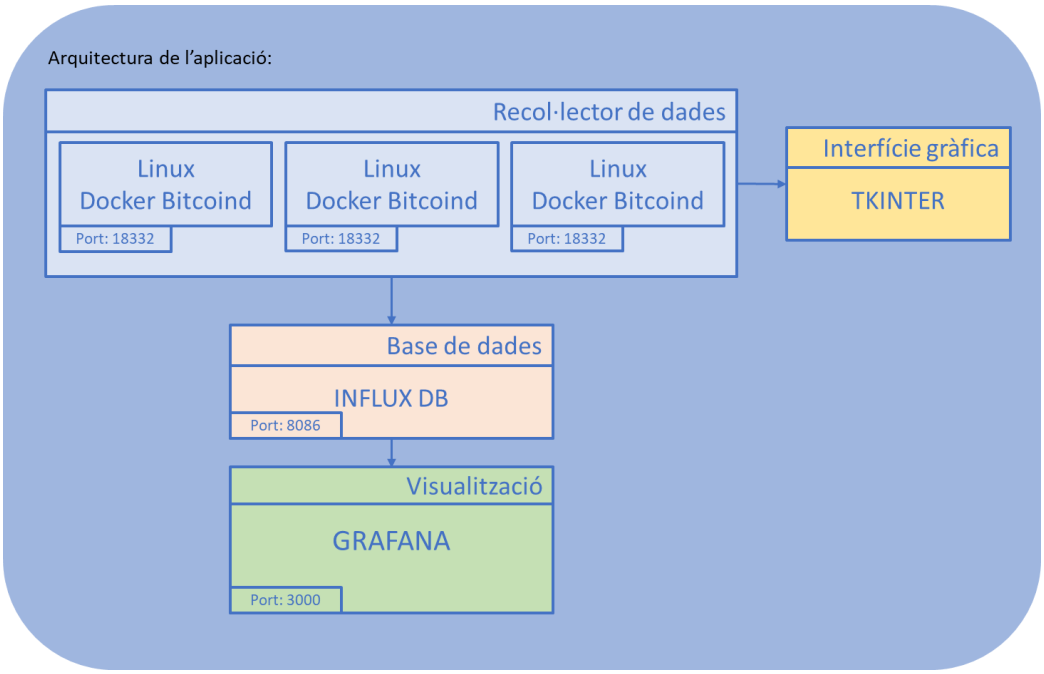

Figura 12. Arquitectura de l'aplicació.

Tots els components interaccionen per tal de recollir la informació de cada node i poder així presentar-ho a través del navegador, mitjançant la plataforma *Grafana*.

4.2. Integració de les tecnologies:

Tenim doncs uns contenidors creats a partir de la imatge *btc_testbed* que serà una imatge que implementa *bitcoind*.

```
david@david-Ubuntu:~$ sudo docker images bt*
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
btc_testbed         latest              77a1f5115383        2 months ago        364MB
```

Figura 13. Imatge btc-testbed.

Aquesta la utilitzem per què ens permet enviar comandes directament als nodes i recollir la informació de manera més eficaç per la feina que volem fer.
A partir d'aquests contenidors, realitzem operacions com crear connexions, afegir o eliminar nodes, afegir blocs o enviar els resultats a la base de dades.

```
[sudo] password for david:
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                    NAMES
4e8d9bed5876        btc_testbed         "bitcoind -debug"   12 minutes ago      Up 12 minutes       0.0.0.0:22006->18332/tcp   btc_n6
762c9d23071c        btc_testbed         "bitcoind -debug"   17 minutes ago      Up 17 minutes       0.0.0.0:22005->18332/tcp   btc_n5
b0906def1ceb        btc_testbed         "bitcoind -debug"   19 minutes ago      Up 19 minutes       0.0.0.0:22004->18332/tcp   btc_n4
ad6cb1205030        btc_testbed         "bitcoind -debug"   20 minutes ago      Up 20 minutes       0.0.0.0:22003->18332/tcp   btc_n3
a90e56a19f38        btc_testbed         "bitcoind -debug"   20 minutes ago      Up 20 minutes       0.0.0.0:22002->18332/tcp   btc_n2
7adaf8cee91f        btc_testbed         "bitcoind -debug"   20 minutes ago      Up 20 minutes       0.0.0.0:22001->18332/tcp   btc_n1
a8057390c74b        btc_testbed         "bitcoind -debug"   20 minutes ago      Up 20 minutes       0.0.0.0:22000->18332/tcp   btc_n0
```

Figura 14. Contenidors btc-nX.

Com hem fet l'estudi sobre una xarxa *regtest* [15], els nodes veiem que utilitzen el port 18332 com a port d'entrada i el 2200x com a ports de sortida. Cada node envia les seves dades a la base de dades *InfluxDB* a través del port 8086 i així es van conservant per què, a continuació, *Grafana* mostri les dades, un cop hem establert la font de l'origen de les dades. També implica que hem de produir accions entre els diferents nodes per tal de simular les actuacions que farien els nodes de la xarxa principal o *mainnet*.

4.3. Desenvolupament de la solució:

Aquí procedirem a explicar el codi implementat per tal d'aclarir algunes parts.

Després d'instal·lar tots els elements necessaris: *bitcoin, docker, influxdb, tkinter* i les seves dependències, hem procedit a analitzar el codi inicial proporcionat per la tutora.
Ha calgut configurar inicialment les constants necessàries com són el nom i les adreces de la xarxa, el prefix dels contenidors i el nom de la imatge *bitcoin*. També ha sigut necessari establir el port (18332) de connexió dels nodes, l'arxiu on es conservaran els logs i un graf predefinit que utilitzarem per establir la topologia de la xarxa. Aquest graf està definit en format *graphml* i es podria modificar de la manera que convingui, però no he pensat que sigui un punt important del treball.

Al bloc principal definim diverses seccions com són:

a. ***Configuració del log i el seu format***: utilitzem dos *handlers* o manipuladors de sortida (pantalla i l'arxiu que trobem a la constant LOG_FILE) :

Codi 1: Python, creació del logging

```
# Configure logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

# create a file handler
handler = logging.StreamHandler()
handler.setLevel(logging.INFO)
logger.addHandler(handler)
handler = logging.FileHandler(LOG_FILE)
handler.setLevel(logging.INFO)
logger.addHandler(handler)

# create a logging format
formatter = logging.Formatter('%(asctime)s -
%(name)s: - %(message)s')
handler.setFormatter(formatter)
```

b. ***Creació de la base de dades (btcBD1):*** mitjançant la comanda adequada creem el client i definim la base de dades on conservarem les dades recollides.

Codi 2: Python, creació base de dades

```
# create data base
clientDB = InfluxDBClient('localhost', '8086',
'admin', 'admin', 'btcBD1')
```

```
clientDB.create_database('btcBD1')
```

c. ***Creació de la imatge i els contenidors***: guardem a la variable client els contenidors bitcoin:

Codi 3: Python, creació d'imatges i contenidors

```
# Create docker client & network
client = docker_setup(build_image=build,
create_docker_network=network,
remove_existing=remove)
```

La funció *docker_setup* crearà la imatge si cal, així com la xarxa i eliminarà contenidors si ja hi són presents.

d. ***Definició del bucle necessari per a la interfície gràfica***: aquí, per mitjà de *Tkinter*, creem un objecte "botó de sortida" i els objectes que formen la interfície inicial:

Codi 4: Python, bucle de l'aplicació

```
scenarioBtc.mainloop()
```

Per iniciar un escenari es clica el botó i inicia una funció general *run_scenario* en la que s'inclouen totes les altres més específiques.

Ara passarem a definir les diferents funcions que trobem en utilitzar les funcions de la interfície:

a. ***AddNode***:

Afegim un nou node a la xarxa inicial, creant una nova instància de la imatge *btc_testbed*. El node s'afegeix a l'últim node de la topologia escollida. Aquesta funció presenta diverses opcions que són afegir node (funció *addNow*), mostrar la informació dels nodes (funció *showInfoAdd*) o tornar a la finestra inicial:

Codi 5: Python, funció *addNode*

```
def showInfoAdd():
...
counter=1
for c in reversed(get_containers_names(client)):
bufferText= bufferText+("    Node {}:  {} ({}) has
peers: {} \n".format(counter,c,
get_ip_by_container_name(client, c),
rpcp_get_peer_ips(client,c)))
```

```
counter=counter+1
textInfo.insert(END,bufferText)
...
def addNow():
n = count_containers(client)
run_new_node(client, DOCK_NETWORK_NAME, node_num=n)
source = DOCK_CONTAINER_NAME_PREFIX + str(n-1)
dest = DOCK_CONTAINER_NAME_PREFIX + str(n)
rpc_create_connection(client, source, dest)
...
```

b. ***DisconnectNode***:

Funció amb les mateixes opcions que l'anterior però eliminem la connexió d'un dels nodes, per simular la seva caiguda. Se'ns mostra una pestanya que ens permet escollir el node que cau. Em va presentar problemes, ja que eliminava inicialment la connexió però es restablia al cap d'uns segons. La solució va ser canviar l'argument de la comanda *rpc addnode* amb el paràmetre *remove* a *addnode*. Comptem igualment amb un quadre d'informació (funció showInfoDiscon), un botó de desconnexió (funció disconnectNow):

Codi 6: Python, funció *disconnectNode*

```
def showInfoDiscon():
...
def disconnectNow():
numNode= int(nf.get())
if (numNode>0 and numNode <=(count_containers
(client, prefix=DOCK_CONTAINER_NAME_PREFIX))):
     nodePos= str(numNode-1)
     nodeToDesc = DOCK_CONTAINER_NAME_PREFIX +
nodePos
     print ('Node to disconnect:
{}'.format(nodeToDesc))
     print("  {} ({}) has peers:
{}".format(nodeToDesc,get_ip_by_container_name(client
, nodeToDesc), rpcp_get_peer_ips
(client,nodeToDesc)))
     argum=str("'"+str(get_ip_by_container_name(clien
t, nodeToDesc))+"'")
     print('Ip disconnected Node: {}'.format(argum))
     for n in rpcp_get_peer_ips (client,nodeToDesc):
     peer= str(get_container_name_by_ip
(client,n[0],DOCK_NETWORK_NAME))
          print('Peer from disconnect:
{}'.format(peer))
          logger.info("  Disconnecting nodes {} and
{}...".format(nodeToDesc,peer))
          disconnect = rpc_call(client, peer,
"disconnectnode", arguments= format(argum))
          time.sleep(5)
      showInfoDiscon()
```

```
else:
     bufferText="No existing node!! (Please insert a
value > 0)"
      showInfoDiscon()
```

c. ***Mine***:

En aquesta funció fem que el node calculi els *hashes* necessaris per a acceptar un bloc, és a dir, fa "mineria".
Definim, per tant, una sèrie de variables que determinaran les mètriques que volem controlar i mostrar a les gràfiques:

Codi 7: Python, funció *mineNow*

```
def mineNow():
     numNode= int(nm.get())
    if (numNode>0 and numNode<=
(count_containers(client,prefix=DOCK_CONTAINER_NAME_P
REFIX))):
          nodePos= str(numNode-1)
         nodeToMine = DOCK_CONTAINER_NAME_PREFIX +
nodePos
         argum=bm.get()
         generate = rpc_call(client, nodeToMine,
"generate", arguments=format(argum))
         blocks_after = rpc_call(client, nodeToMine,
"getblockcount")
         balance_after = rpc_call(client, nodeToMine,
"getbalance")
         diff_after = rpc_call(client, nodeToMine,
"getdifficulty")
         hash_after = rpc_call(client, nodeToMine,
"getbestblockhash")
         connections_after = rpc_call(client,
nodeToMine, "getconnectioncount")
          bufferText="  Node {} is aware of {}
blocks and has a balance of {} tbtc.\n Difficult is
{} .\n And has {} as the best hash.\n Has {}
connections.".format(numNode,blocks_after,
balance_after,diff_after,hash_after,connections_after
)
          logger.info("  Node {} is aware of {}
blocks and has a balance of {} tbtc.\n Difficult is
{} .\n And has {} as the best hash"
.format(numNode,blocks_after, balance_after,
diff_after,hash_after))
          showInfoMine(bufferText)
         sendData()
```

Així doncs, trobem les variables:

1. *generate*: un node comença a generar hashes.

2. *blocs_after*: mostra els blocs que el node és assabentat.
3. *balance_after*: mostra el balanç del node i les seves adreces.
4. *diff_after*: mostra la dificultat de computació.
5. *hash_after*: mostra el *hash* de més complexitat trobat per a aquell bloc.
6. *connections_after*: mostra les connexions que consten al node.

Aquí també comptem a una finestra d'informació (funció *showInfoMine*). A més tenim diversos objectes "botó" que ens mostren a la finestra una sèrie de dades com:

1. *info*: mostra la informació del node, amb el balanç, la dificultat i els blocs que coneix.
2. *hashes*: mostra el *hash* més complicat que s'ha trobat.
3. *difficult*: imprimeix per la finestra d'informació la dificultat que consta al node.

Cadascuna de les anteriors utilitza les variables abans comentades per mostrar la informació.

d. ***stats***:

Es tracta de la funció que mostra les gràfiques a través del navegador. Això es fa fent ús d'un objecte *webbrowser* que obre l'adreça especificada a l'argument i que està enllaçada a la gràfica predeterminada a *Grafana*.

El codi està configurat pel meu navegador però es pot variar entrant a *Grafana* (accedim a través de localhost:3000) i veient les gràfiques construïdes prèviament.

Codi 8: Python, funció *stats*

```
def stats():
...
     def openBalance():
          webbrowser.open('http://localhost:3000/d/T
     kF1Yinmk/btcgraph?refresh=5s&orgId=1&panelId=2&f
     ullscreen&from=1525730400000&to=1525816799999',n
     ew=1, autoraise=True)

     def openBlocks():
          webbrowser.open('http://localhost:3000/d/T
     kF1Yinmk/btcgraph?refresh=5s&orgId=1&panelId=4&f
     ullscreen&from=1525730400000&to=1525816799999',n
     ew=1, autoraise=True)

     def openDiff():
```

```
            webbrowser.open('http://localhost:3000/d/T
        kF1Yinmk/btcgraph?refresh=5s&orgId=1&panelId=6&f
        ullscreen&from=1525730400000&to=1525816799999',n
        ew=1, autoraise=True)

        def openConnections():
            webbrowser.open('http://localhost:3000/d/T
        kF1Yinmk/btcgraph?refresh=5s&panelId=10&fullscr
        een&orgId=1&from=1525790744393&to=1525816799999
        ',new=1, autoraise=True)
...
```

A part de totes les funcions esmentades, trobem altres que també cal comentar:

a. ***sendData***:

És la funció que s'encarrega d'enviar les dades a la base de dades. Ho fa a través de la comanda *write_points* a la qual li passem un *string JSON* amb el format adequat:

Codi 9: Python, funció *sendData*

```
def sendData():
    for c in reversed
(get_containers_names(client)):
    blocks_after = rpc_call(client, c,
"getblockcount")
        balance_after = rpc_call(client, c,
"getbalance")
        diff_after = rpc_call(client, c,
"getdifficulty")
        hash_after = rpc_call(client, c,
"getbestblockhash")
        connections_after = rpc_call(client, c,
"getconnectioncount")

        json_body = [{"measurement":
'balance","time": (time.strftime('%Y-%m-%d
%H:%M:%S')),
"tags": {"node": c},"fields": {"value":
balance_after}}]
    clientDB.write_points(json_body)
        json_body = [{"measurement":
'blocks","time": (time.strftime('%Y-%m-%d
%H:%M:%S')),
"tags": {"node": c},"fields": {"value":
blocks_after}}]
        clientDB.write_points(json_body)
        json_body = [{"measurement":
'difficult","time": (time.strftime('%Y-%m-%d
%H:%M:%S')),
```

```
"tags": {"node": c},"fields": {"value": diff_after}}]
          clientDB.write_points(json_body)
          json_body = [{"measurement":
'bestHash","time": (time.strftime('%Y-%m-%d
%H:%M:%S')),
"tags": {"node": c},"fields": {"value": hash_after}}]
          clientDB.write_points(json_body)
          json_body = [{"measurement":
'connections","time": (time.strftime('%Y-%m-%d
%H:%M:%S')),
"tags": {"node": c},"fields": {"value":
connections_after}}]
           clientDB.write_points(json_body)
 ...
```

b. ***create_scenario_from_graph_file*** *i* ***create_scenario_from_graph***:

Són les funcions encarregades d'aplicar una topologia determinada a la xarxa. Són funcions ja implementades prèviament i subministrades per la tutora.

c. ***docker_setup***:

Funció que crea la imatge *bitcoin* i els contenidors requerits. És cridada des de funcions inicials ja vistes.

4.4. Disseny interfície gràfica:

La interfície gràfica ja hem comentat que s'ha implementat amb la que es considera la llibreria gràfica estàndard de *Python*, *Tkinter*.

És una llibreria que permet moltes opcions diferents però clares i senzilles, i en això m'he centrat: en fer una interfície prou simple però senzilla d'utilitzar. Lògicament es poden millorar els continguts gràfics però ha semblat suficient fer-ho així.

Tkinter es basa en una estructura que executa un bucle infinit o bucle principal, en el nostre cas serà un bucle creat per l'objecte *scenarioBtc*, que serà un objecte *tk()*. Però l'objecte que inicia la interfície és l'objecte ***root***, que és on es creen els contenidors i tenim les opcions d'inici de l'aplicació.

Codi 10: Python, objecte *root*

```
# Root config
root = Tk()
root.title("Plataforma per analitzar la Xarxa
Bitcoin")
root.resizable(1,1)
root.geometry("800x600+100+100")
app = App(root)

# Loop config
scenarioBtc= Tk()
scenarioBtc.title("Simulator")
scenarioBtc.resizable(1,1)
scenarioBtc.geometry("800x600+100+100")
bar1 = Frame(scenarioBtc,width=420,
height=30,bg="blue")
bar1.place(x=380,y=0)
title= Label(scenarioBtc, text="PLATFORM TO ANALYZE
BITCOIN", font=("Verdana", 14),bg="blue", fg="white")
title.place(x=0,y=0)
scenarioBtc.withdraw()

# Create docker client & network
client = docker_setup(build_image=build,
create_docker_network=network,
remove_existing=remove)

#Define initial interface
exitButton= Button(root, text="Exit", command =
close_window,width=21,height=5).place(x=605,y=30)

# End
scenarioBtc.mainloop()
```

El resultat serà:

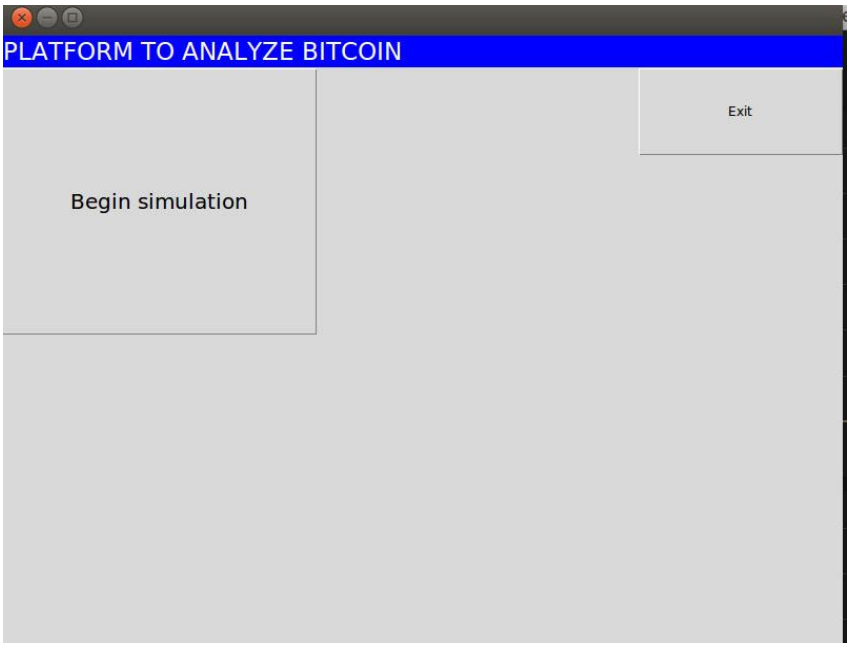

Figura 15. Pantalla inicial.

La classe *App* defineix una sèrie d'opcions necessàries per moure's entre les diferents finestres o objectes *tk*.

Un cop cliquem el botó d'inici entrem, a través de la funció *run_scenario*, en l'objecte principal del programa: ***scenarioBtc***. Aquí trobem els botons principals que donen accés les funcions *addNode, disconnectNode, mine i stats*:

Codi 11: Python, objecte *scenarioBtc*

```
...
scenarioBtc.deiconify()
root.destroy()

# Options buttons
addNodeButton= Button(scenarioBtc, text="Add Node",
command = lambda: addNode(),width=21,
height=10).place(x=0, y=30)
disconnectNodeButton= Button(scenarioBtc,
text="Disconnect node", command = lambda:
disconnectNode(),width=21, height=10).place(x=0,
y=180)
```

```
mineButton= Button(scenarioBtc, text="Mining",
command = lambda: mine(),width=21,
height=10).place(x=0, y=330)
statsButton= Button(scenarioBtc, text="Show Data",
command = lambda: stats(),width=21,
height=10).place(x=0, y=480)
backButton= Button(scenarioBtc, text="Exit", command
= lambda: tornar(),width=21, height=5).place(x=605,
y=30)
...
```

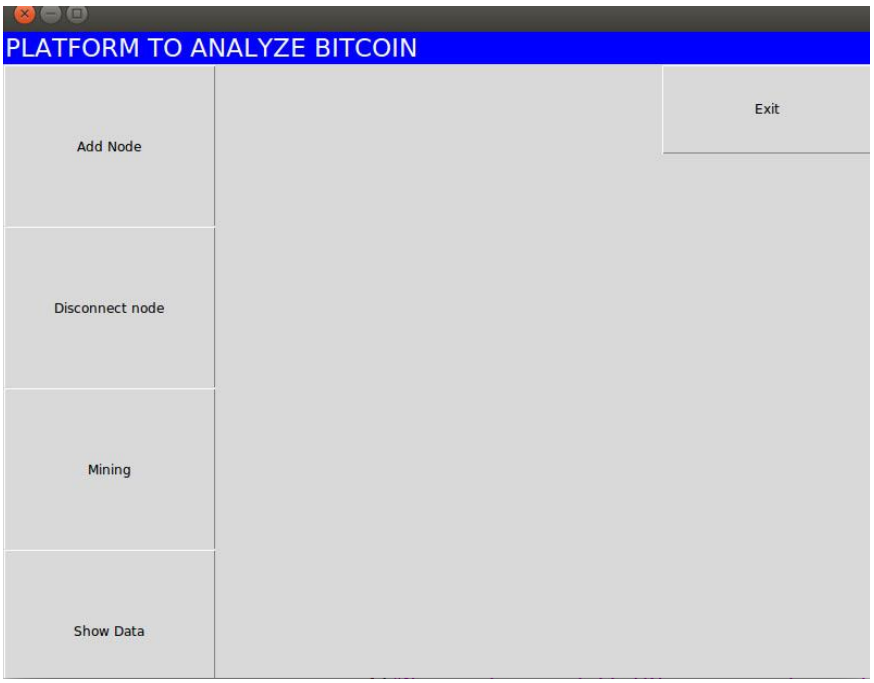

Figura 16. Pantalla opcions.

A partir d'aquest punt podem moure'ns entre les diferents opcions i segons l'escollida accedim a les següents finestres:

addNodeBtc:

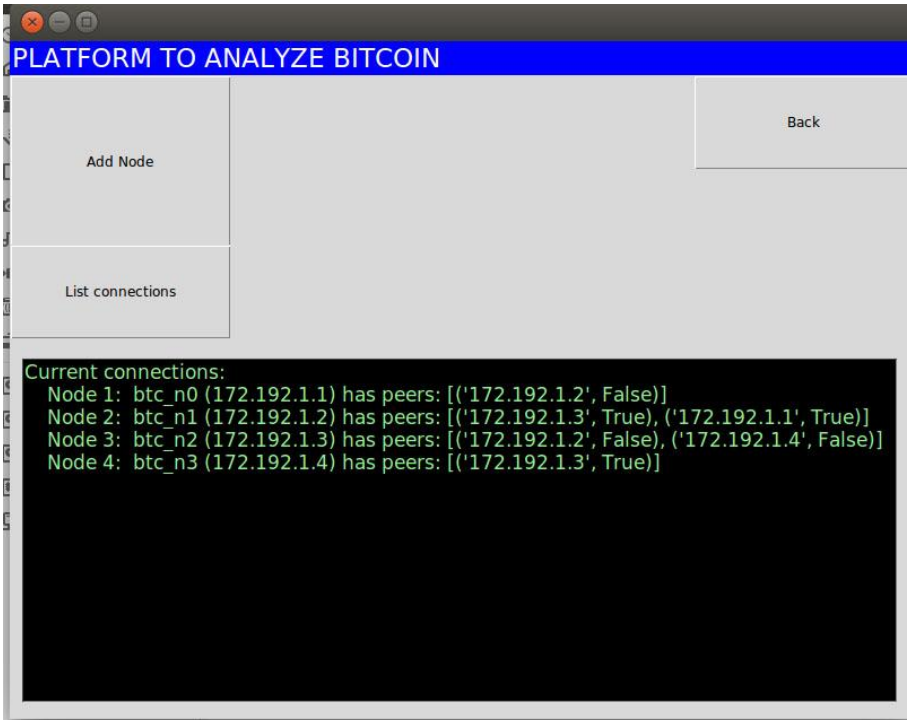

Figura 17. Pantalla Afegir Node.

On destaquem les comandes que configuren la interfície i els objectes "botó" creats:

Codi 12: Python, objecte *addNodeBtc*

```
def addNode():
     addNodeBtc=Tk()
     scenarioBtc.withdraw()
     addNodeBtc.title("Adding Node")
     addNodeBtc.resizable(1,1)
     addNodeBtc.geometry("800x600+100+100")
     bar1 = Frame(addNodeBtc,width=420,
height=30,bg="blue")
     bar1.place(x=380,y=0)
     title= Label(addNodeBtc, text="PLATFORM TO
ANALYZE BITCOIN", font=("Verdana", 14),bg="blue",
fg="white")
     title.place(x=0,y=0)
     textInfo= Text(addNodeBtc, bg="black",
fg="lightgreen",
font=("Verdana",10),width=70,height=15)
     textInfo.place(x=10,y=280)
```

```
def tornar2():
          addNodeBtc.destroy()
          scenarioBtc.deiconify()
...
addButton= Button(addNodeBtc, text="Add Node",
command = lambda: addNow(),width=21,
height=10).place(x=0, y=30)
listconnsButton= Button(addNodeBtc, text="List
connections", command = lambda:
showInfoAdd(),width=21, height=5).place(x=0, y=180)
backButton= Button(addNodeBtc, text="Back", command =
lambda: tornar2(),width=21, height=5).place(x=605,
y=30)
...
```

discNodeBtc:

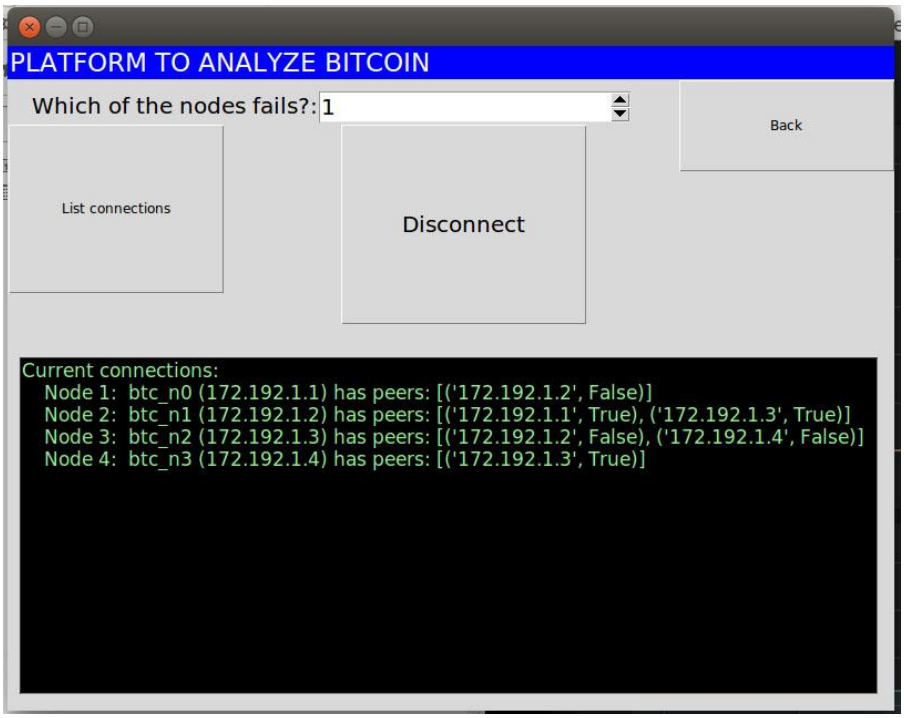

Figura 18. Pantalla Desconnectar Node.

mineBtc:

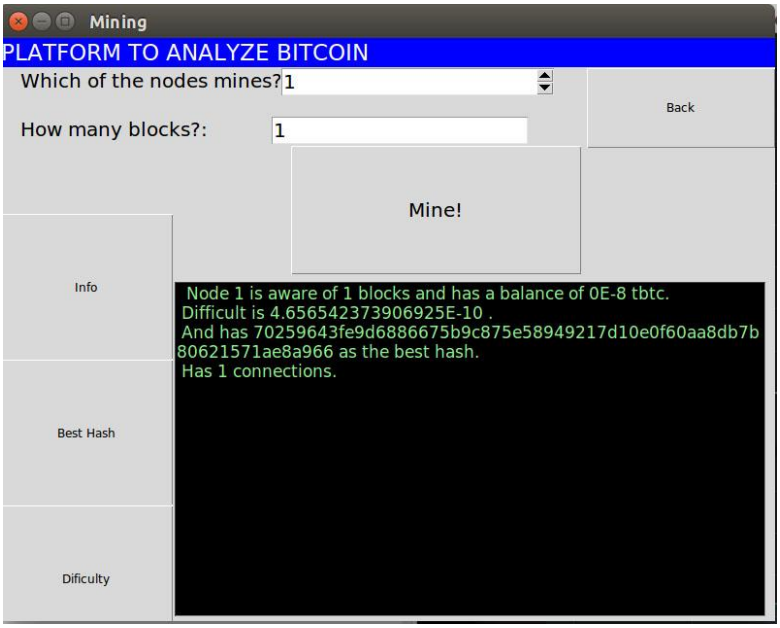

Figura 19. Pantalla Mineria.

statsBtc:

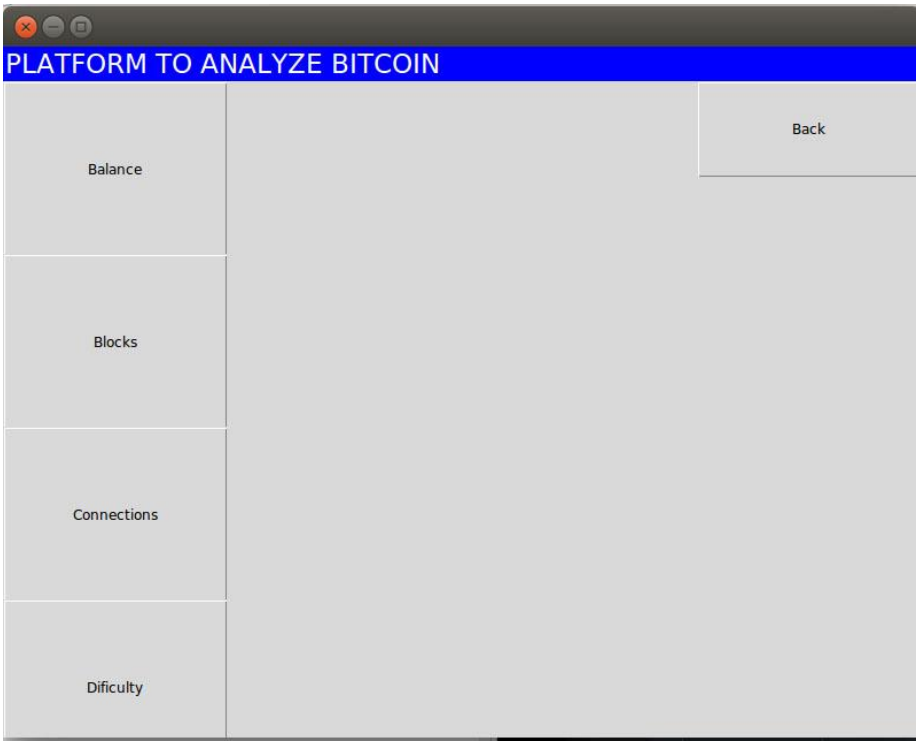

Figura 20. Pantalla Informació.

I el resultat d'algunes de les opcions seleccionades en aquesta secció de dades:

Balance:

Figura 21. Gràfic balanç.

Blocks:

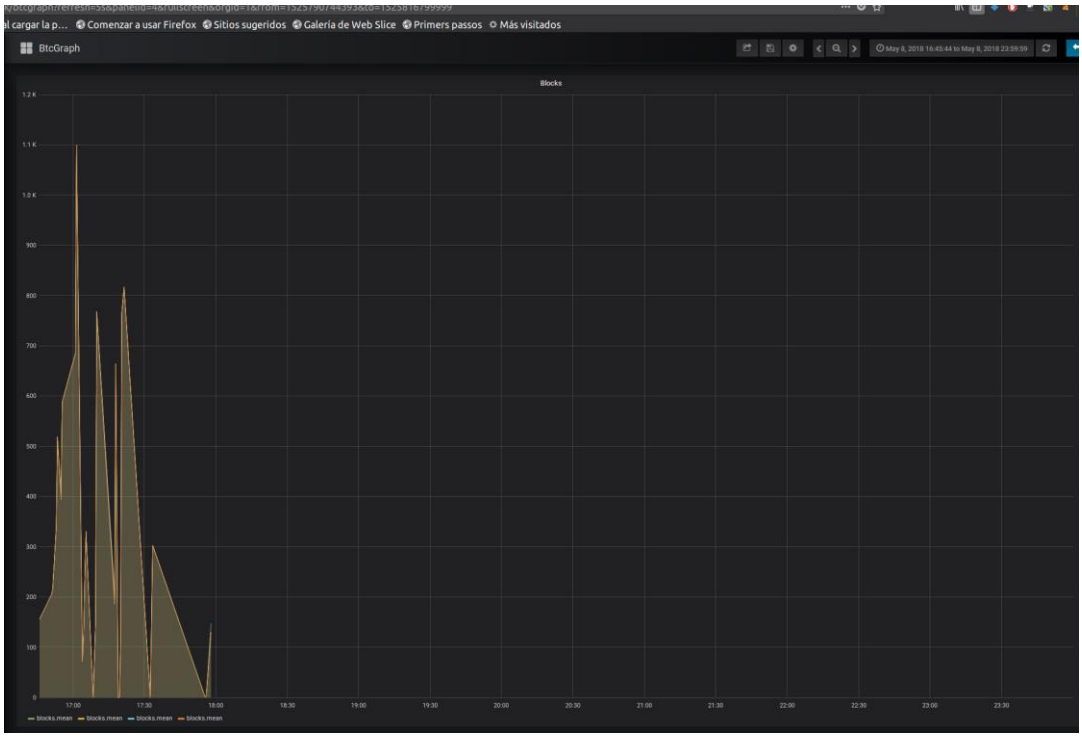

Figura 22. Gràfic blocs.

4.5. Presentació de les gràfiques:

Per mostrar les gràfiques ja hem comentat que farem servir la plataforma *Grafana*.
Anteriorment hem vist alguns dels exemples que podem obtenir en recollir les dades a través de *InfluxDB* i l'únic que hem de fer és configurar *Grafana* mitjançant el port 3000 del nostre navegador per tal d'obtenir les gràfiques. El programa ho fa directament, ja que té predefinides un *dashboard* per a cada opció de visualització, però Grafana és prou potent per mostrar diversitat de gràfics en diferents formats.
El codi necessari per mostrar-les és el que va inclòs en l'objecte *statsBtc*, ja prèviament descrit (Codi 8).

5. Proves i tests:

Les proves de l'aplicació les hem realitzat en un entorn tancat, ja que ho hem fet amb la modalitat ***regtest*** del protocol Bitcoin. El mode *regtest* consisteix en una xarxa local que permet iniciar de zero tota la xarxa Bitcoin quan sigui necessari. És una xarxa no tan coneguda com la *mainnet* o la *testnet*, però permet l'estudi del comportament de la xarxa d'una manera més senzilla, la qual cosa proporciona una forma adequada per crear noves funcionalitats i aprofundir en la investigació de la tecnologia *Blockchain*. Va ser afegida a la versió 0.9.0, presentada el març de 2014, i va esdevenir un tipus de proves addicional. *Regtest* permet, per tant, crear cadenes de blocs privades i mantenir-les, així, sota el nostre control.
Així doncs, seguint aquest patró, la xarxa l'hem estudiat durant algunes hores seguides, o a estones curtes, ja que com es tracta d'una xarxa tancada no esdevenen accions si nosaltres no les provoquem. Seria possible generar accions aleatòries per simular situacions reals, però no ho he acabat de concretar.

Figura 23. Dashboard Grafana.

6. Conclusions i línies futures de treball:

Aquest treball ha estat realitzat com a TFG i la veritat és que al principi no sabia ben bé en què hauria de consistir. A mesura que han anat passant les entregues he anat entenent més bé què era el que havíem d'aconseguir. No es tractava de descobrir secrets sobre el B*itcoin* però sí que calia trobar alguna eina en què basar-nos a l'hora d'estudiar-lo. Així doncs, hem construït una aplicació per tal de recollir informació per poder utilitzar-la a posteriori i conèixer, així, el comportament de la xarxa i els seus nodes.
Ha calgut fer un estudi de les idees principals i els conceptes que engloba el *Bitcoin* i conèixer les funcions i possibilitats que disposem. Hem vist que la criptografia juga un paper molt important en el desenvolupament d'aquest sistema digital: el càlcul i utilització de les funcions resum o *hashes* és el que dóna seguretat a aquesta moneda en l'actualitat. Caldrà però, veure en el futur si cal modificar l'algorisme utilitzat per tal de seguir garantint les característiques necessàries de fiabilitat. També hem vist que el Bitcoin presenta alguns inconvenients com seria el tema de l'escalabilitat, és a dir, en principi el nombre de bitcoins està limitat a 21 milions, què passarà si tothom el vol utilitzar com a mitjà de pagament?
També hi ha un problema greu causat per la generació dels blocs. Actualment aquest procés de mineria està concentrat [16] en països o regions (Xina, Islàndia, etc.) on es donen certes característiques (baix cost energètic i altes inversions en maquinari especialitzat). Això pot provocar un problema de centralització minera, fent impossible un dels objectius originals del *Bitcoin*, que és la universalitat en la creació i producció a la moneda.
També hem estudiat el concepte de *Blockchain* i ha resultat molt atraient, ja que és una tecnologia que està en alça i s'utilitza en molts altres camps (contractes, sanitat [17], logística, impostos).
Per acabar, seria interessant aprofundir més en el tema de la creació i destrucció dels nodes, ja que m'ha donat alguns problemes i, a vegades, la xarxa ha tingut comportaments erràtics.
Per tal de buscar solucions als problemes plantejats hem intentat crear aquesta plataforma i malgrat que falta molt i es presenten molts dubtes, m'ha semblat molt interessant i m'ha ajudat molt a conèixer una mica en què consisteix la "Trinitat *Bitcoin*" (moneda, xarxa i protocol).

7. Glossari:

- **Bitcoin**: ens referim a tres conceptes: la moneda criptogràfica, la xarxa que la suporta i el protocol que necessita per funcionar.

- **Bitcoin Core**: codi font del *Bitcoin* que inclou interaccions per línia de comandes o per interfície gràfica.

- **Peer**: de l'anglès parell o company, representa un parell de nodes connectats en iguals condicions.

- **Xarxa P2P**: xarxa entre iguals o *peers*. Permet la connexió punt a punt. Permet la distribució de la xarxa.

- **TCP**: protocol orientat a connexió. És el protocol fonamental a l'hora d'establir connexions a Internet.

- **RPC**: remote procedure call. Executar codi en una màquina remota.

- **Criptografia de clau publica**: mètode criptogràfic que mitjançant dues claus: una privada i una pública permet establir condicions de seguretat i autenticitat. També coneguda com a criptografia asimètrica.

- **Hash**: o funció resum. Permet crear una seqüència alfanumèrica a partir d'una entrada de manera que serà gairebé impossible desxifrar-la sense la clau secreta.

- **Dificultat**: ens referim a la potència de computació necessària per poder fer el càlcul del *hash* per a un bloc.

- **SHA256**: algorisme de xifrat segur de 256 bits.

- **Python**: llenguatge de programació multiplataforma de codi obert, el nom del qual prové del grup còmic *Monty Phyton*.

- **Blockchain**: o cadena de blocs, representa un diari d'accions o en el nostre cas el llibre de comptabilitat del *Bitcoin*. Cada bloc s'enllaça amb el següent mitjançant el càlcul d'un *hash*.

- **Node bitcoin**: integrant de la xarxa, pot incloure diverses funcions: (moneder, miner, propagador de transaccions o contenidor de la cadena completa de blocs).

- **Docker**: plataforma que permet l'ús de contenidors aïllats dintre d'un sistema operatiu. Aquests contenidors poden albergar aplicacions diverses i executar-se independentment del sistema que les contingui.

- **Satoshi Nakamoto**: creador o grup de creadors que van presentar el 2009 el sistema Bitcoin.

8. Bibliografia:

[1] Wikipedia. https://es.wikipedia.org/wiki/Peer-to-peer [últim accés: 5-6-2018].

[2] Wikipedia. https://en.wikipedia.org/wiki/Blockchain [últim accés: 5-6-2018].

[3] Docker. https://www.docker.com/what-docker [últim accés: 4-6-2018].

[4] Statoshi. http://statoshi.info/ [últim accés: 5-6-2018].

[5] Mastering Bitcoin, Andreas M. Antonopoulos, O'REALLY Atlas. https://en.bitcoin.it/wiki/Mastering_Bitcoin [últim accés: 5-6-2018].

[6] Paper Original Bitcoin. https://blog.bit2me.com/es/paper-original-bitcoin-en-espanol [últim accés: 5-6-2018].

[7] Blockchain.info. https://blockchain.info/es/charts/my-wallet-n-users [últim accés: 5-6-2018].

[8] Wikipedia. https://es.wikipedia.org/wiki/SHA-2 [últim accés: 5-6-2018].

[9]Wikipedia. https://en.bitcoinwiki.org/wiki/Bitcoind [últim accés: 4-6-2018].

[10] Grafana. https://grafana.com/ [últim accés: 5-6-2018].

[11] InfluxDB documentation. https://docs.influxdata.com/influxdb/v1.5/ [últim accés: 5-6-2018].

[12] The Go Programming Language. https://golang.org/ [últim accés: 5-6-2018].

[13] TkInter. https://wiki.python.org/moin/TkInter [últim accés: 5-6-2018].

[14] Python. https://www.python.org/ [últim accés: 5-6-2018].

[15] Regtest Mode in Bitcoin. https://bitcoin.org/en/developer-examples#regtest-mode [últim accés: 5-6-2018].

[16] Menos de 4 millones de bitcoins quedan por ser minados. https://www.criptonoticias.com/mineria/menos-4-millones-bitcoins-quedan-ser-minados/ [últim accés: 5-6-2018].

[17] Así es como el Blockchain transformará el sistema sanitario. http://www.ticbeat.com/salud/asi-es-como-el-blockchain-transformara-el-sistema-sanitario/ [últim accés: 5-6-2018].