UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Autor: Jesús Navalon i Pastor

Tutor: Javier Esparza Peidro

Novembre 2011

"Sempre que us preguntin si podeu fer un treball, digueu que sí i poseu-vos de seguida a aprendre como es fa."

Franklin Delano Roosevelt (1882-1945)
Polític nord-americà.

Índex de continguts

Capítol 1.
Introducció

Aquest capítol pretén ser l'avançada a la resta del document, oferint a colp d'ull tant les motivacions que ha generat la creació d'aquest projecte com els seus objectius. Nogensmenys també s'inclourà una breu descripció de la resta dels continguts.

1.1. Visió General

Malgrat que els monitors de màquines virtuals (d'ara endavant hipervisors) van aparèixer cap als anys 70. No va ser fins el segle XXI que aquesta tecnologia acabés impulsant-se degut a la necessitat de consolidar els diferents servidors d'avui en dia amb una administració simplificada. El primer hipervisor per a PC que va aparèixer als anys 90 va ser VMWare i a dia d'avui ja n'hi ha més d'una vintena als mercats.

Si en un principi la dificultat era administrar diferents servidors amb diferents sistemes, el problema ha derivat en la gestió dels hipervisors i en la planificació i creació dels sistemes a virtualitzar.

Per a finalitzar amb aquesta visió general, cal esmenar que aquest projecte no solament pretén solucionar alguns dels problemes sorgits a l'anterior paràgraf si no que pretén ser un esborrany que serveixi com a guia per a similars implementacions reals.

1.2. Aplicacions semblants al mercat

Actualment existeixen varies aplicacions disponibles que permeten la gestió a distància dels hipervisors alguns tan senzills com virt-manager[1] que permeten la creació de màquines i la seva gestió de manera bàsica o altres més complicats i complets com openNebula[2] que a més a més són capaços de crear una infraestructura completa al núvol amb una gestió avançada.

Aquest projecte pretén ser un punt intermedi entre ambdós extrems, unificant el simplisme amb el major nombre de característiques, permetent la flexibilitat i compatibilitat necessària per a satisfer les necessitats de qualssevol tipus d'usuari.

1.3 Objectius

Com hem dit abans, aquest projecte està enfocat a la creació d'una infraestructura completa al núvol, però mantenint la senzillesa i la flexibilitat per a què puga complir les necessitats de qualssevol usuari.

Així doncs, el propòsit d'aquest projecte és la d'obtindre un programari que permeta la creació, difusió i execució de màquines virtuals en una xarxa per omissió i adequar-les per a què estes estiguen disponibles per al seu ús. En altres paraules: crear una xarxa de

1 virt-manager.org
2 opennebula.com

sistemes preparats per al seu ús immediat.

No obstant, per tal que aquest projecte compleixi els requisits de compatibilitat caldrà que sigui multiplataforma per tal que pugui ser executat en qualssevol entorn i modular per a què qualssevol usuari pugi afegir o esborrar característiques per a obtindre un sistema d'acord a les seves necessitats.

Degut a que la gestió de les diferents màquines, així com dels seus hipervisors s'ha de realitzar de manera remota, cal protegir les connexions que es realitzen, així com evitar un accés indegut als sistemes de gestió.

Aquests elements seran estudiats i explicats més endavant per a una millor comprensió del lector.

1.4 Contingut de la memòria

Aquest apartat està destinat a oferir una breu explicació del que podrem trobar en cadascú dels capítols i annexes que composen aquest document.

En la "*Introducció*" és tractarà de donar una visió global del projecte, que objectius s'han establert i quin serà el contingut de la resta del document.

Al segon capítol "*l'estat de la qüestió"* estudiarem quin són els requisits necessaris per a dur a terme el nostre projecte. Donarem un colp d'ull a les possibles solucions i abordarem algunes de les eines que utilitzarem per a poder resoldre aquests problemes.

Durant el tercer capítol "*Solució del problema*" intentarem donar a conèixer quines han estat les solucions implementades per a resoldre les dificultats plantejades a l'anterior capítol. Tant funcional com estructural-ment, posant èmfasi en aquells punts més importants al projecte

Rere les anteriors explicacions, passarem al següent capítol, "*un viatge pel projecte*" on es mostrarà una petita demostració de com un usuari pot crear la seva pròpia xarxa mitjançant les aplicacions Blasar i Kuasar. Pretén ser una guia des de zero, que comence des de la configuració del servidor, fins a poder instal·lar Màquines Virtuals als hipervisors remots.

Al capítol "*conclusions i futurs treballs*" es parlarà de les opcions que presenta el projecte realitzat i el possible enfocament que podria tindre els següents treballs basats en l'aplicació desenvolupada. A més a més, s'exposaran les conclusions pròpies a les que s'ha arribat després de tot aquest procés

Finalment, la part final del document conté un parell d'annexes que complementen la resta d'aquesta memòria, afegint informació que millore la comprensió del conjunt. Entre aquests annexes es pot trobar un glossari de termes, acrònims, referències, bibliografia i documentació utilitzada.

Capítol 2.
l'estat de la qüestió

En aquest capítol estudiarem quin són els requisits necessaris per a dur a terme el nostre projecte. Donarem un colp d'ull a les possibles solucions i abordarem algunes de les eines que utilitzarem per a poder resoldre aquests problemes.

L'ambició d'aquest projecte és la d'aconseguir un sistema de disseny i implementació d'infraestructures de màquines virtuals. Que permetran a grans trets:

- Dissenyar una xarxa de màquines virtuals (Tant de manera local com al núvol).

  Inclou:

  - Creació de projectes per a diferents xarxes.
  - Establir màquines virtuals, segons SO i característiques de maquinari.
  - Configurar la xarxa de manera Manual / Automàtica, tenint present la més nova versió d'IP (Ipv6).
- Obtenció de les màquines que composen la nostra xarxa d'hipervisors.

  Inclou:

  - Obtenció dels servidors de manera Manual / Automàtica.
  - Obtenció d'informació d'estat (Recursos lliures, Màquines corrent, etc.).
- Implementació dels projectes que hem creat.

  Inclou:

  - Enviar les dades de les màquines al sistemes on s'allotgen els hipervisors i comprovar-les.
  - Crear les màquines a l'hipervisor.
  - Configurar les màquines virtuals.
  - Gestionar l'arrancada de les màquines.

El projecte està destinat a treballar sobre qualssevol hipervisor, malgrat això, només s'ha desenvolupat per a funcionar amb VirtualBox d'Oracle. No obstant no entrarem dins de les comunicacions entre el servidor i els hipervisors perquè aquests poden treballar segons les característiques dels diferents hipervisors i les seues utilitats per a gestionar màquines.

Per a complir els requisits que hem exigit al nostre projecte hem de tindre en compte un seguit d'aspectes, tant estructurals com programàtics.

2.1 Centralitzar o Globalitzar?

Un dels primers problemes que poden sorgir quan plantegem un nou projecte és:

Centralitzem les dades o les Globalitzem?

Centralitzar el nostre projecte suposaria la creació d'un servidor que es dediqués a controlar i gestionar cadascú dels hipervisors enregistrats en ell, així com controlar l'accés dels diferents usuaris al sistema.

Les bondats de la centralització són ben conegudes per tothom:

- Millor facilitat per gestionar màquines
- Millor gestió dels usuaris i permisos
- Menys dades redundants
- Millor protecció dels hipervisors ja que la connexió és indirecta

Malgrat això, entre els seus defectes es troben:

- Un error pot deixar sense servei a tots els usuaris o a la gran part
- Una vulnerabilitat pot comprometre a tots els usuaris i als seus hipervisors.
- Flexibilitat reduïda.
- Major desenvolupament

Per altra banda la globalització permet que cada usuari no sols tinga un sistema independent a la resta d'usuaris, si no que permet una seguretat i flexibilitat més individualitzada. A canvi és necessari obtindre alguns desavantatges com la redundància de dades o la pitjor protecció dels hipervisors.

Així doncs una de les primeres decisions que hem de prendre és la de decidir com volem que es comuniquen les nostres aplicacions.

A continuació presentem dos diagrames que mostren les diferències entre els dos tipus estudiats.

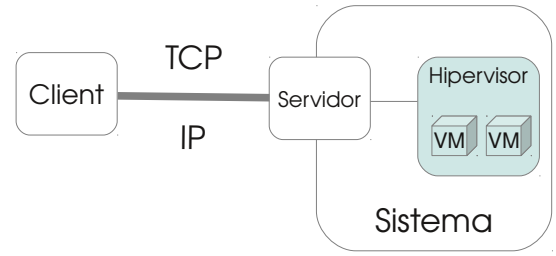


Diagrama de comunicació globalitzat

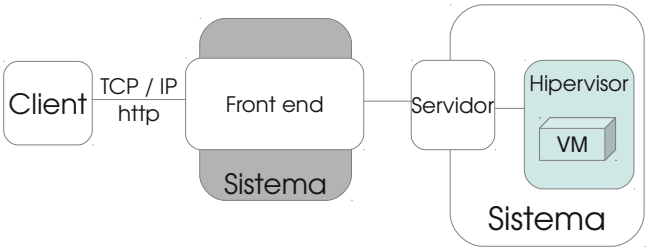


Diagrama de comunicació centralitzat

2.2. Un programa, mil usuaris.

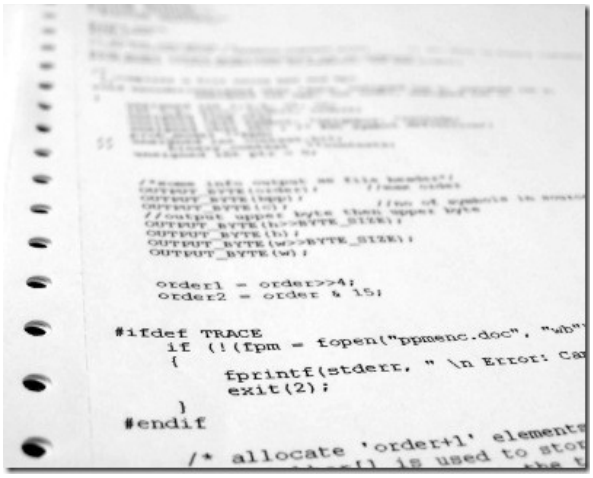
Dos de les característiques que hem volgut dotar al nostre projecte ha estat des d'un principi la compatibilitat i la flexibilitat. Aquestes no solament ens permeten abstraure'ns de l'entorn en el que és executada la nostra aplicació, si no que també la fan única respecte a les altres.

Les millores que oferixen no tan sols beneficien als usuaris, sinó també als desenvolupadors que troben una manera més ràpida i senzilla per a desenvolupar sense tindre en compte gairebé l'entorn on s'han d'iniciar. Al mateix temps la flexibilitat aporta una manera ràpida i senzilla de cercar i reparar els errors causats al programar.

Decidit doncs, quines característiques s'han d'implementar, només cal elegir les eines per dur-les a terme.

2.2.1. La compatibilitat

Per a què el nostre projecte sigui compatible amb el major nombre d'entorns ens cal d'un llenguatge de programació que siga compilat i entès pels diferents entorns. A més cal que siga capaç de cobrir completament les necessitats, i si es possible que ens done més facilitats de desenvolupament.



Si bé la majoria de llenguatges poden treballar en diferents plataformes com és el llenguatge C, no ho són així les llibreries amb que treballen. Així per exemple per a establir una connexió a Windows cal utilitzar la llibreria “winshock.h”, en canvi a Linux caldria fer ús de la llibreria “sys/socket.h”.

Per a evitar aquests problemes, escollirem un llenguatge de programació que treballe sota una maquina virtual, i que sigui aquesta la que s'encarregue de traduir al sistema (siga qui siga) les nostres ordres.

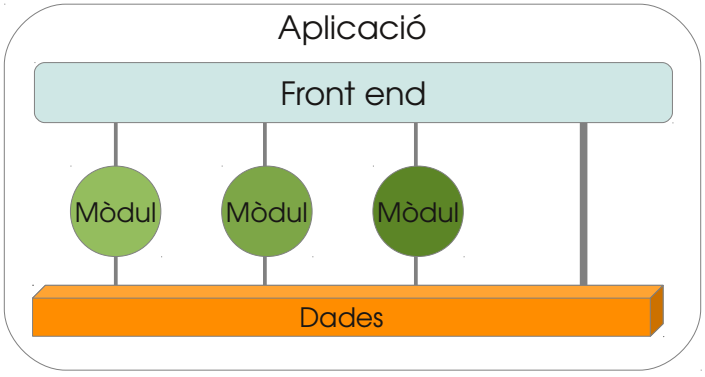
Java: És un llenguatge orientat a objecte desenvolupat per Sun Microsystems al principi dels anys 90. És un llenguatge molt semblant a C en quant a sintaxis, però l'ús d'objectes és més senzill i elimina eines de baix nivell que solen portar a molts errors. Per altra banda la gestió de la memòria està gestionada pel propi llenguatge i no pel del programador, per lo que simplifica molt la tasca de creació i alliberament.

Mono: És un projecte de codi obert, iniciat en un principi per Ximian, i que actualment és impulsat per Novell per a crear un grup d'eines lliures, basades en GNU/Linux i compatibles amb .NET. Actualment pot treballar en sistemes GNU/Linux, BSD, Mac OS, Solaris i Windows.

2.2.2. La flexibilitat

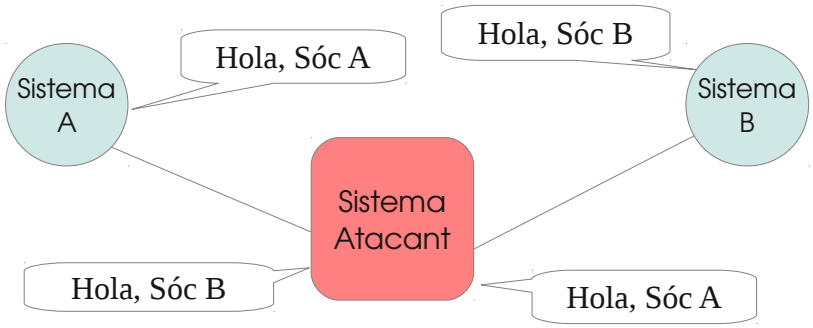
Com havíem dit abans, la flexibilitat és un dels principals punts forts del nostre projecte. El problema que esdevé es trobar alguna manera eficient per a què un usuari pugi afegir o eliminar tantes característiques com necessitats haja de cobrir.

Per aquesta raó s'ha de trobar una solució per tal que un usuari puga afegir característiques noves sense tindre que compilar tota l'aplicació de nou, es a dir fer una aplicació modular. A més a més també caldria implementar un sistema per tal de compartir la informació entre tots els mòduls.



2.2.3. Seguretat de les comunicacions

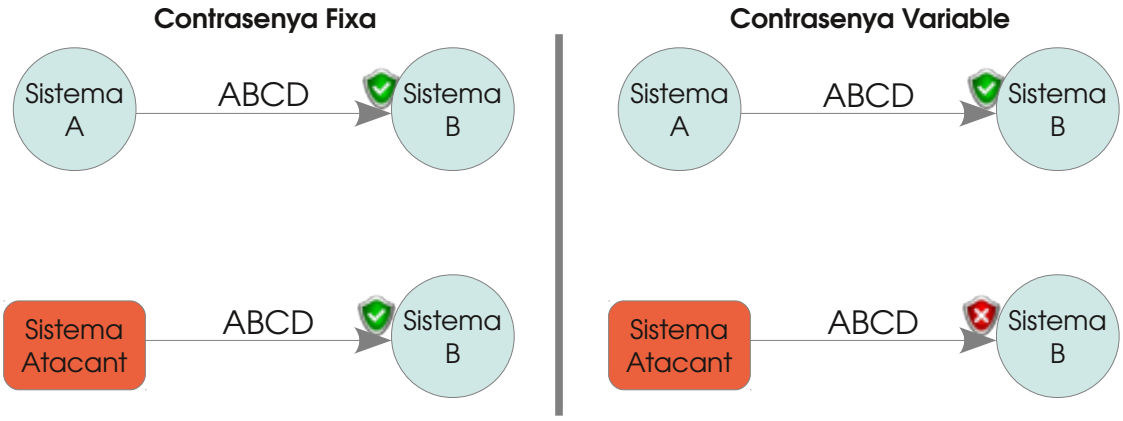
És imprescindible que les comunicacions que es realitzen entre les nostres aplicacions es facin de manera segura, evitant que qualssevol tercer pugui obtindre accés a dades de gran importància que pogueren comprometre la seguretat, no tan sols de la màquina virtual, sinó també la del propi servidor on s'allotja. A més a més, també seria necessari la implementació de cap mètode per tal d'evitar la tècnica d'atac "*man-in-the-middle"*. Aquesta tècnica es basa en la introducció d'un sistema en mig d'una comunicació entre altres dos per tal de llegir, inserir i modificar a voluntat els missatges d'ambdues parts sense que cap d'elles conegui que l'enllaç entre ells ha estat violat.



Exemple d'un atac "Man-in-the-middle"

2.2.4. Seguretat d'accés

Tan o més important que la seguretat de les comunicacions és la seguretat d'accés. Suposant que algú hagués pogut accedir a les comunicacions entre els sistemes, deuria ser necessari algun tipus de contingència per a evitar que inclús obtenint dades d'accés, sigui impossible tindre accés al servei. En altres paraules, implementar un accés amb contrasenyes variables.



2.2.5. Verificació de les dades

En cas d'haver de transferir una gran quantitat de dades de colp seria convenient afegir algun tipus de verificació de “hash” que permetera comprovar que les dades han estat enviades de manera correcta.

2.2.6. Antibloqueig dels fils de comunicació

Moltes vegades una comunicació pot quedar bloquejada a l'espera d'un comandament que mai arriba o simplement per què hi ha hagut un error i alguna de les parts ha tancat sense acabar la comunicació correctament. Si aquestes comunicacions es quedaren al limbe del programa podria ser un problema en el consum dels recursos. Així mateixa s'hauria de trobar una solució que permetera entre d'altres coses:

- Comprovar que les connexions no hagen mort i donat el cas finalitzar-les per a alliberar recursos.
- Obtindre cap eina que permeta la limitació de fils de comunicació per tal d'evitar una saturació al server i que pugi perjudicar a l'hipervisor (i per tant a les màquines virtuals) que conviu al sistema.
- Permetre l'eliminació de fils de connexió de manera local

Capítol 3

Solució del problema

En aquest capítol intentarem donar a conèixer quines han estat les solucions implementades per a resoldre les dificultats plantejades a l'anterior capítol. Tant funcional com estructural-ment, posant èmfasi en aquells punts més importants al projecte.

3.1. Mono vs. Java

Com hem vist al capítol anterior, era necessari que el nostre projecte funcionara en qualssevol tipus d'entorn. Per una banda, perquè abstrau les dissimilituds entre entorns i per l'altra perquè simplifica el desenvolupament i correcció d'errors del nostre projecte.

Malgrat que totes dues plataformes de programació eren vàlides per al desenvolupament de la nostra aplicació, el fet que Mono sigués una plataforma gairebé nova, poc desenvolupada en comparació a Java i amb una menor comunitat que pogués solucionar alguns dels problemes que pogueren sorgir. És va decantar cap al llenguatge Java.

Java, a més de ser multiplataforma, posseeix una gran flexibilitat de programació, permetent entre altres característiques utilitzar en cas que fos necessari, codi natiu utilitzant la *"Java Native Interface"* o bé incloent llibreries de tercers que permeta fer ús de codi precompilat. A més a més, la comunitat que hi ha darrere d'ella permet obtindre solucions de manera fàcil i senzilla.

3.2. Estructura del projecte.

Una volta elegit el nostre entorn, el següent punt és decidir com ha de ser estructurat el nostre projecte. Com hem vist al punt 2.1 "Centralitzar o Globalitzar?" és necessari saber com ha de comunicar-se les nostres aplicacions i com han de ser emmagatzemades les dades del sistema.

Per una banda, les dades podrien ser emmagatzemades en un servidor comú que permetera una gestió més senzilla. A més a més, permetria una millor protecció dels servidors on s'allotgen els hipervisors, però malgrat això aquesta forma de gestionar les dades també comporta una sèrie d'inconvenients.

Així per exemple, una vulnerabilitat en la gestió d'usuaris podria afectar a tots el comptes creats a més de les màquines vinculades. Per altre costat, el mantindre totes les dades centralitzades significa que tots els usuaris han de connectar-se a un mateix sistema. El que podria suposar una saturació del servidor per la realització de tasques que consumiren prou recursos, tan de xarxa, com de procés com per exemple la còpia de fitxers entre dos sistemes remots.

Per aquest motiu es declinem per un sistema globalitzat, malgrat que la gestió de les dades puguen ser més redundants i més complicades de gestionar.

En compensació a aquests inconvenients, obtenim un major rendiment i menor probabilitat d'errors d'accessibilitat a causa de saturacions al servidor, fallides, etc.

Per tant l'estructura que desenvoluparem estarà format per un client que s'encarregarà de connectar amb un servidor. Aquest servidor gestionarà als hipervisors que haja d'administrar segons les ordres que li passe el client.

Per a entendre millor les funcions de les dos aplicacions s'estudiaran de manera separada més endavant.

3.3. Sistema Modulat

Per a complir amb el requisit de la flexibilitat hem de trobar algun mètode que permeta afegir i eliminar característiques segons la necessitat de l'usuari.

Amb aquest sistema obtenim dos beneficis. Per una banda, l'aplicació es adaptable a qualssevol tipus d'usuari i de necessitat, i per altra el consum de recursos és molt menor que si s'hagués de carregar tot de colp.

Com hem dit abans, el nostre sistema es dividirà en dos aplicacions. El client i el servidor. Cadascú contindrà els seus propis mòduls per tal de complir amb els requisits que se'ls demana.

Així doncs, el client haurà de disposar principalment d'eines de gestió de Màquines, de xarxa, d'implementació de les xarxes virtuals creades a les xarxes reals, etc. Al nostre projecte s'han implementat els següents mòduls:

VMCreator: *Permet crear projectes de les màquines virtuals que volem implementar en una xarxa real. Ens permetrà definir el tipus d'hipervisor on va a ser allotjat, el tipus de SO que va a ser instal·lat i les característiques de la màquina (Memòria, Interfícies de xarxa, etc.)*

NetCreator: *s'encarregarà d'assignar les IPs a les diferents màquines creades al mòdul VMCreator. Serà capaç d'auto assignar les adreces IP segons una adreça de xarxa i una màscara*

ServerManager: *Serà l'encarregat de gestionar els servidors on s'allotgen els hipervisors. Permetrà afegir, esborrar servidors, així com obtindre informació dels recursos del sistema. També permetrà una gestió bàsica de les màquines instal·lades al hipervisor (Eliminar,*

arrancar, etc.)

Deployer: *S'encarregarà de distribuir les nostres màquines en els diferents hipervisors disponibles, utilitzant algoritmes segons l'espai en disc disponible o la memòria ram. A més s'encarregarà de comprovar les dades de configuració així com la disponibilitat dels servers on s'han d'allotjar les màquines per a seguidament instal·lar-les on pertoquen.*

Més endavant s'explicaran aquests mòduls de manera més extensa així com el seu funcionament.

Al servidor, en canvi, li caldran mòduls que permetin la comunicació entre els diferents hipervisor que s'hagin de gestionar. En el nostre cas, només crearem un mòdul anomenat ***virtualbox*** que permetrà la comunicació entre els comandaments del servidor i el gestor de l'hipervisor *VboxManage* mitjançant execució de comandaments. També serà explicat més detalladament el seu funcionament, encara que deixarem de banda la comunicació que es realitza entre el mòdul i l'hipervisor ja que treballa sota comandaments.

3.4. Seguretat

Com s'ha vist al capítol anterior, la seguretat és un punt molt important a tindre en compte. Ens cal mantindre la seguretat tant de les nostres comunicacions com dels nostres servidors. I per això cal prendre un seguit d'accions i mesures per tal d'evitar atacs o accessos no permesos.

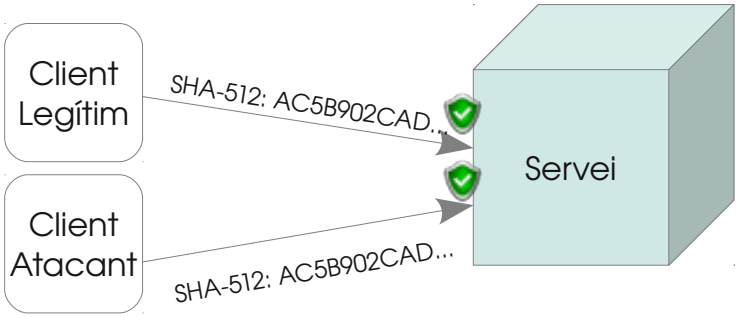
S'esdevenen doncs dos tipus d'accions a prendre en consideració. Primerament la protecció de les dades mitjançant xifrat i segon la seguretat d'accés al servidor utilitzant un sistema d'accés per usuari i contrasenya.

Per a xifrar les nostres comunicacions farem ús del protocol SSL que ens proporciona autenticació i privacitat de la informació entre els extrems. I d'aquesta manera evitarem que cap tercer pugui accedir a aquesta informació.

Per a evitar l'accés als nostres serveis, s'ha pensat en dos tipus d'accés. Un més senzill, però més dèbil, basat en una simple comprovació usuari / contrasenya i un altra més

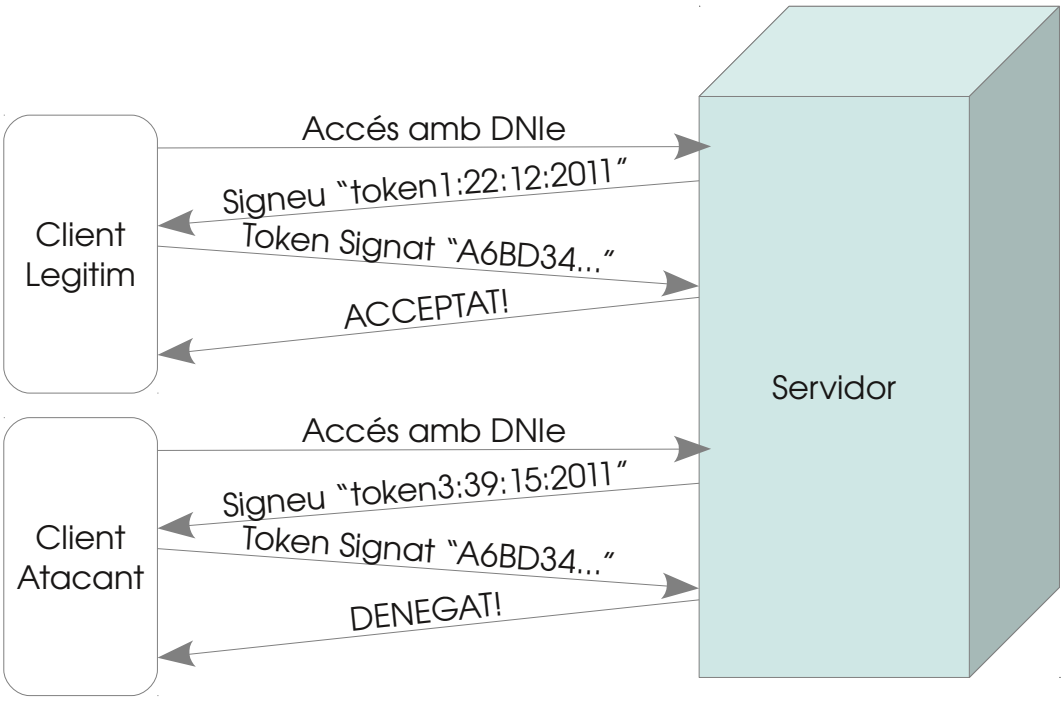
robusta basat en signatures amb DNIe.

El primer tipus permet un accés molt bàsic on la màxima seguretat es troba en què la contrasenya mai s'envia de manera neta, si no que es xifra prèviament amb un algoritme d'una única direcció com SHA-512 i es compara amb la que te el servidor. Aquest mètode protegeix la contrasenya de l'usuari, però no l'accés al servei. Per tant si un tercer interceptés el missatge podria fer ús de la contrasenya xifrada per a accedir al servei degut a que aquesta mai canvia.



L'altre mètode a implementar fa ús de la signatura de documents que ofereix el DNIe. Aquest mètode és més complex però més robust, ja que encara que la contrasenya sigui interceptada, aquesta al canviar constantment degut a que ha de signar cada vegada un tipus de dades diferents, fa impossible que un tercer pugui accedir al servei. No solament no permet això, sinó que gràcies a que el DNIe és una targeta intel·ligent la signatura només pot efectuar-se a dintre d'ella i per tant cap qui no tinga possessió a la targeta i a la seva clau podrà signar les dades. Per aquesta raó l'autenticació amb DNIe deuria ser primordial si s'ha d'accedir als serveis des d'un entorn no segur. Més endavant aquest mètode d'autenticació serà explicat amb més detall.

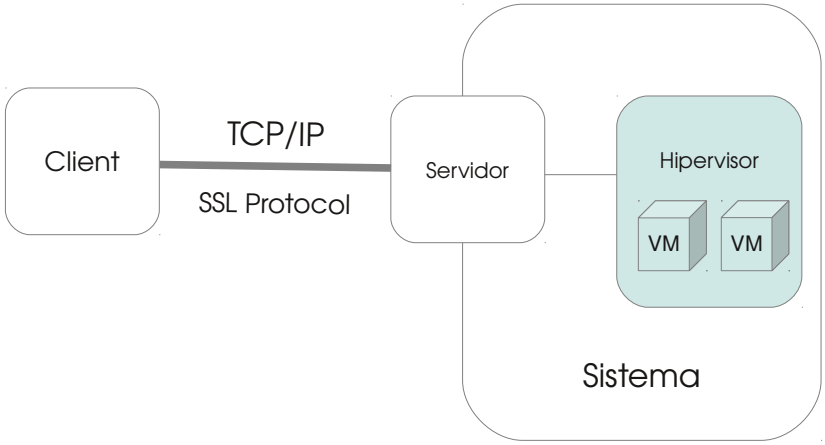
Vet aquí com seria l'autenticació en un sistema que utilitzés aquest mètode:


Client
Legitim
Accés amb DNIe
Signeu "token1:22:12:2011"
Token Signat "A6BD34..."
ACCEPTAT!
Servidor
Client
Atacant
Accés amb DNIe
Signeu "token3:39:15:2011"
Token Signat "A6BD34..."
DENEGAT!

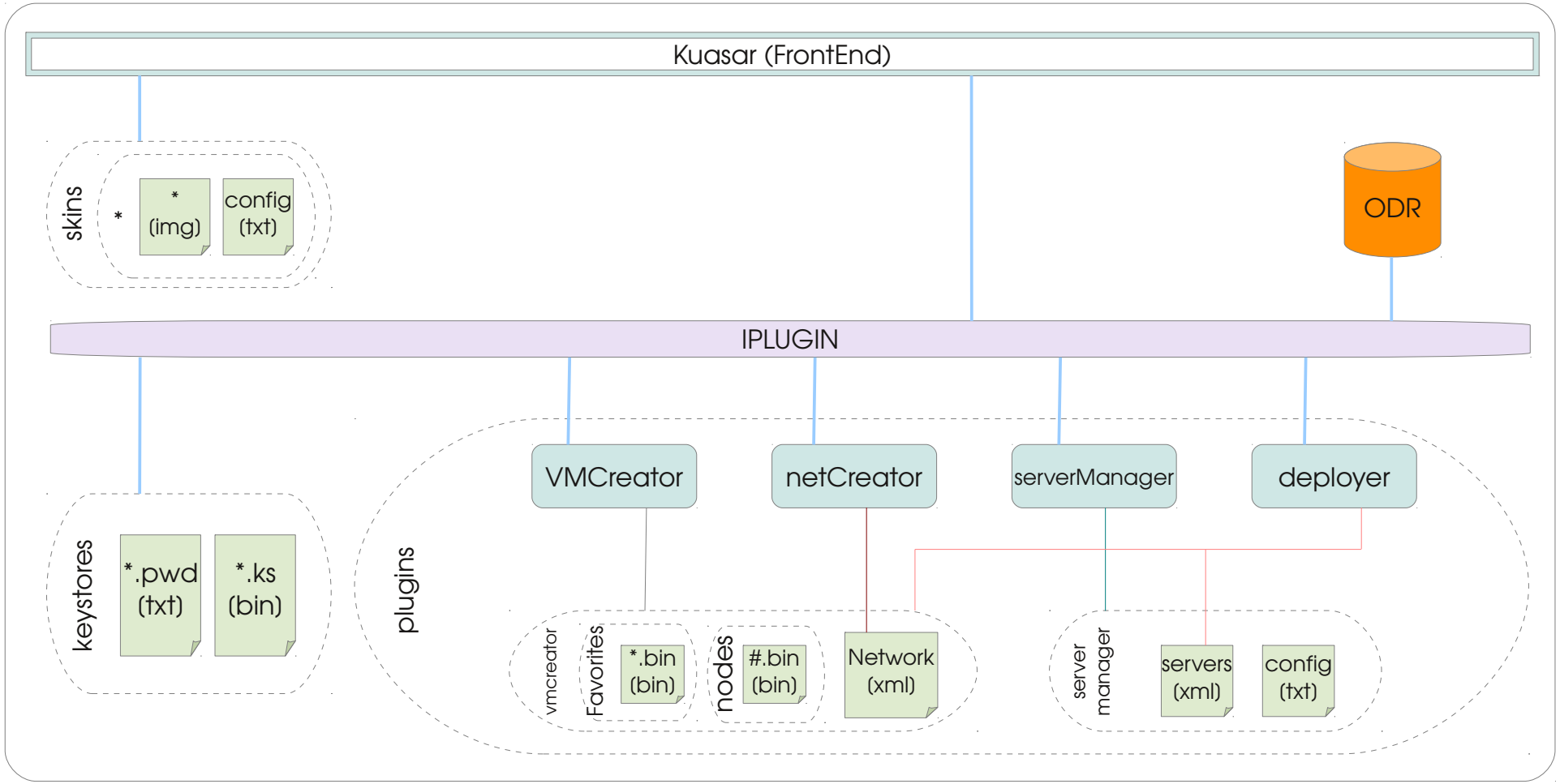
3.5. Estructura d'intercomunicació del sistema:

Com hem dit abans el nostre projecte estarà format per mòduls interconnectats que permetran flexibilitzar les seues característiques i reduir el seu consum. Per altra banda, la modelització permetrà als desenvolupadors trobar i solucionar més ràpidament els problemes descartant i aïllant-los dels demés mòduls. A més les actualitzacions serien més lleugeres i més senzilles que si d'un únic bloc es tractes. És necessari primer de tot visualitzar de manera global com s'interconnecta tot el projecte, tant les dues aplicacions principals, com els diversos mòduls que conformen el projecte. A continuació és mostra un gràfic elemental on es determina aquesta conformació, les diferents connexions entre les aplicacions principals i els seus mòduls, i per altra banda, també les connexions on s'emmagatzemen els diferents tipus de dades que ha d'emmagatzemar per al seu funcionament.

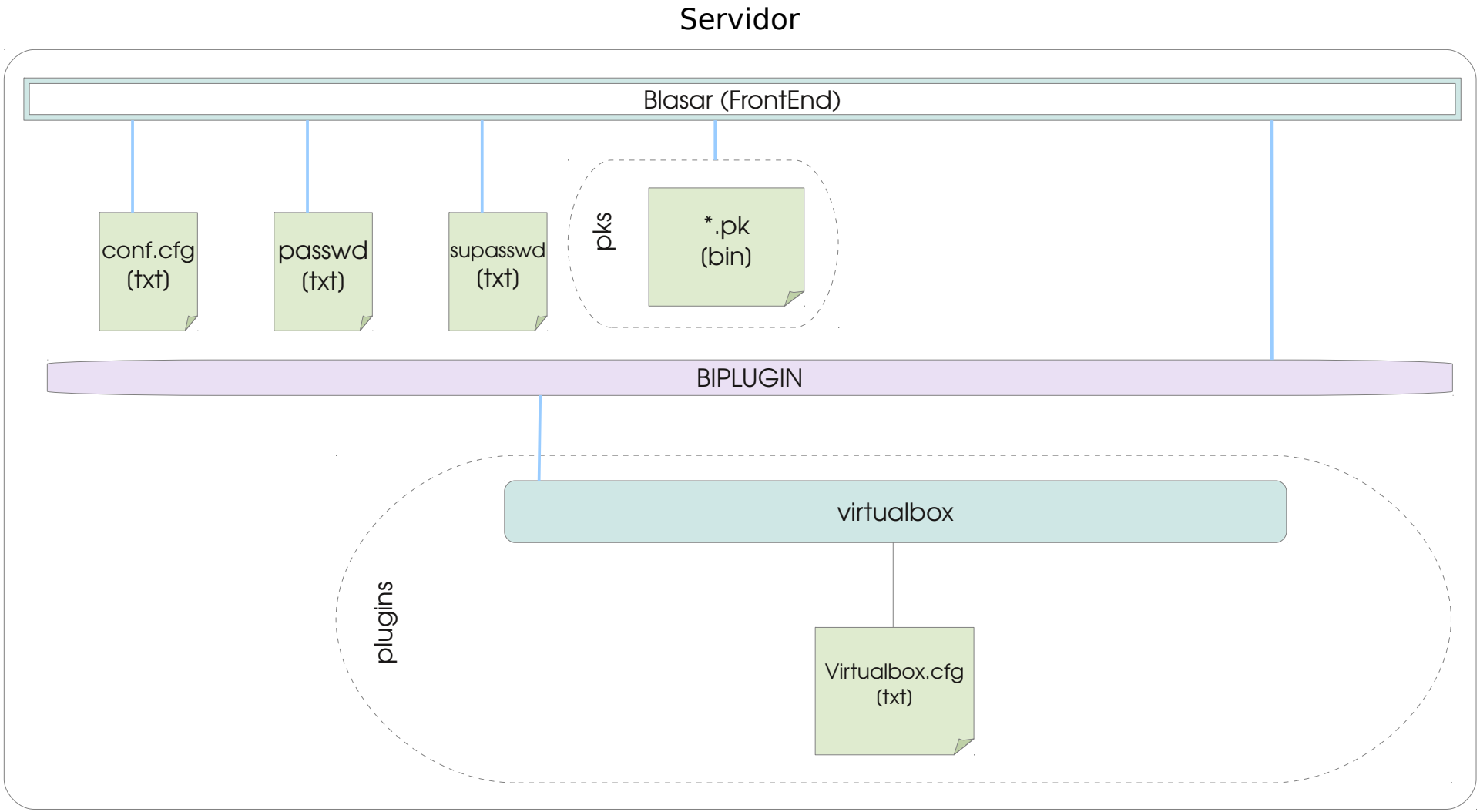
Estructura global:



Client[3]



3 Nota: El símbol * representa a qualssevol nom, mentre # representa un número.


Servidor
Blasar (FrontEnd)
conf.cfg
(txt)
passwd
(txt)
supasswd
(txt)
pks
*.pk
(bin)
BIPLUGIN
virtualbox
plugins
Virtualbox.cfg
(txt)

Com pot observar-se als diagrames totes dues aplicacions es comuniquen mitjançant una connexió TCP/IP sota el protocol SSL. El port per defecte d'aquesta connexió serà la 46600 per tal d'evitar problemes amb altres aplicacions[4]

En quant a la comunicació entre els mòduls, així com entre el “front-end” es realitza a través de les APIs[5] IPLUGIN i BIPLUGINS. Aquestes APIs contenen el codi necessari per a què tant el “front-end” com els mòduls puguen intercomunicar-se entre ells. A més a més l'aplicació client, degut sobretot a que ha de compartir una gran quantia de dades, té implementat un sistema de dades propi (ODR) que permet que qualssevol modul pugui obtindre informació desada per ell o un altre mòdul. Més endavant s'aprofundirà en aquest tema.

4 www.iana.org/assignments/port-numbers

5 Conjunt de declaracions que defineix el contracte d'un component informàtic amb qui farà ús dels seus serveis.

3.6. Aprofundiment per blocs

En aquest subcapítol ens introduirem als indrets del funcionament de cada mètode d'un en un per a què sigui més senzill entendre cada part que forma el projecte. No és vol ensenyar codi sinó més bé mostrar com està lligat cadascú dels mòduls amb els altres i com funciona el seu indret. S'aconsella tindre el codi font a l'abast per a entendre millor aquest punt ja que s'anomenaran noms de classes i paquets.

3.6.1 El Client (Kuasar)

El client del nostre projecte està format per un front-end anomenat com el client (ja que aquest és l'únic que ha d'executar-se) i quatre mòduls (VMCreator, netCreator, servermanager i deployer) a més de la seva API que permet la comunicació entre ells anomenat IPLUGIN (Interface – Plugin). ODR està inclòs dins d'IPLUGIN i per tant serà estudiat amb ell.

3.6.1.1. Front-End (kuasar.jar)

El front-end o kuasar.jar és la base del nostre programa. És qui s'encarrega de carregar tots els mòduls i d'aturar-los, així com de compartir les primeres dades per a l'ús de tothom (directori d'inici, directori de mòduls, configuracions de l'usuari, etc.). A més, és el qui aporta la interfase gràfica on cadascú dels mòduls pot interactuar directament amb l'usuari.

3.6.1.1.1. Inicialització

El front-end així com l'aplicació comença iniciant la classe *Main.java* situat al paquet *kuasar*. El primer que fa només arrancar és llegir els paràmetres que rep. Actualment l'únic paràmetre que permet el client és el '-d' per a posar el client en mode de desenvolupador[6].

A continuació la classe Main carrega les variables de configuració i comparteix algunes que hi són d'interès per als mòduls mitjançant el sistema ODR. Aquesta càrrega de variables es realitza cridant a la classe *kuasar.util.SetVariables.java.*

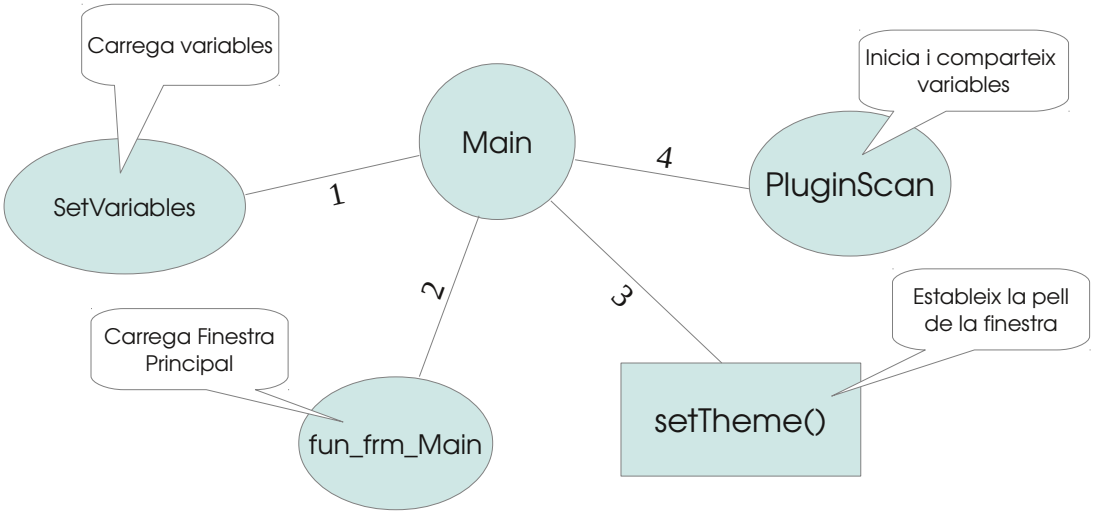
Seguidament d'haver carregat les variables i haver-les compartides, la funció Main carrega la finestra principal (*kuasar.gui.fun_frm_Main.java*) i configura la seva pell.

Després s'encarrega de cercar els mòduls disponibles en el directori *plugins,* els

6 Development mode: Mostra les comunicacions entre el client i el servidor per comprovar errors i trobar anomalies en la conversa.

carrega en memòria i els mostra al *JList* de la finestra principal.

Finalment es mostrat a la finestra principal.



3.6.1.1.2. Càrrega de mòduls

PluginScan és la classe destinada a carregar els mòduls que es troben en el directori *plugins* i d'iniciar-los, així com de mostrar-los en el Jlist que hi ha disponible a la finestra principal. El seu funcionament és el següent:

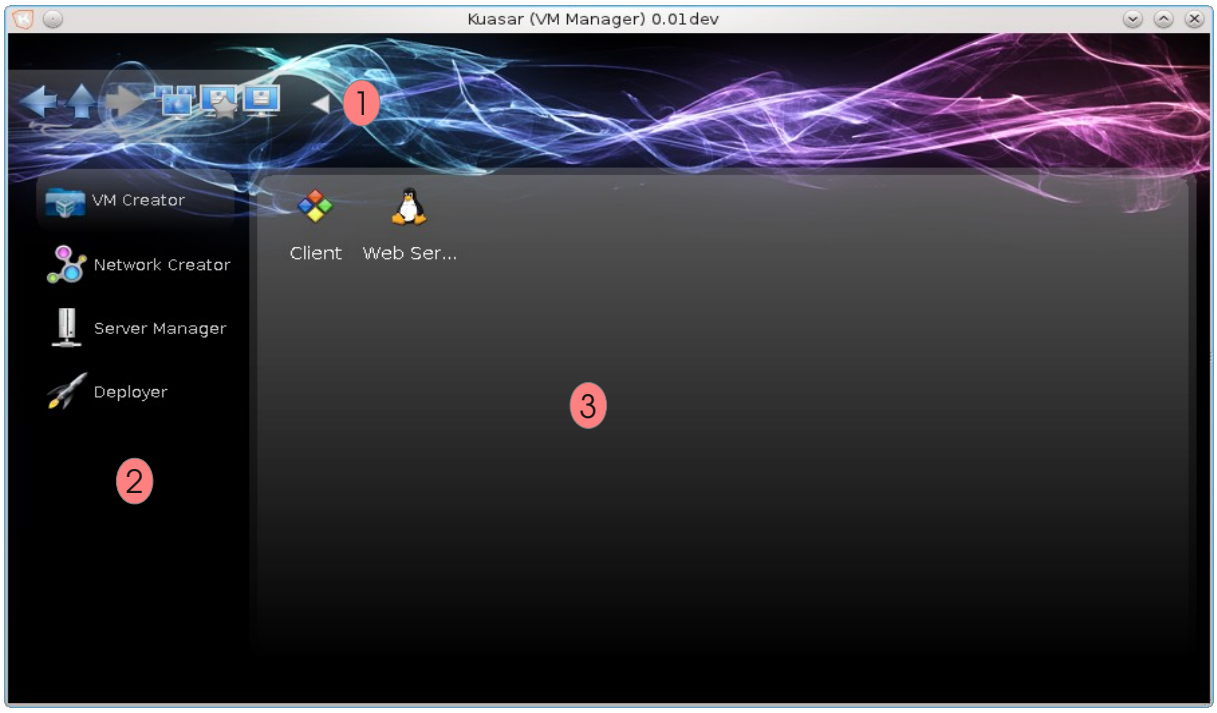
La primera acció que realitza, és cridar a *PluginLoader.java* (*kuasar.util.plugins*) una classe que s'encarrega de cercar en el directori *plugins* tots els fitxers amb extensió .jet (extensió que es va establir com a mòdul, per a diferenciar-los dels .jar, ja que aquests primers no són executables) i inicia la classe principal declarada com *PluginInterface* (Més endavant estudiarem aquesta classe). Finalment, si el mòdul s'ha iniciat correctament, es desa la classe *PluginInterface* en una pila per a posteriorment retornar-lo com un vector.

Una volta PluginInterface li retorna el vector de mòduls a PluginLoader, aquest últim s'encarrega de publicar la llista a l'ODR amb la clau $PLUGINS.

PluginInterface també està destinat a carregar al JList de la pàgina principal tots els mòduls que se li ha passat obtenint el nom i la icona de cada mòdul.

3.6.1.1.3 La finestra principal (front-end)

La finestra principal és l'única eina d'interacció entre l'usuari i l'aplicació incloent els seus mòduls. Vet aquí la finestra i cadascuna de les parts que la composa.



1 – Barra d'eines. Una barra que permet afegir botons d'acció.
Aquesta barra pot amagar-se o reduir-se per a evitar un accés involuntari.

2 – Llista de plugins. És un espai destinat a albergar els diferents mòduls carregats per tal que l'usuari puga seleccionar aquell que necessite.

3 – Panell de treball: És un espai destinat al mòdul per a què puga mostrar una interfase per a interactuar amb l'usuari. Aquesta àrea és carrega una volta l'usuari ha seleccionat el mòdul al llista situat a la banda esquerra.

La classe principal on es troba el front-end s'anomena *fun_frm_Main.* Aquesta classe és una extensió de la classe *frm_Main*, que, al mateix temps, és una extensió d'un *JFrame*. El perquè que *fun_frm_Main* i *frm_Main* no estiguen fusionats en una mateixa classe, és perquè existia una necessitat per a separar mètodes entre una i l'altra per a què els diferents mòduls pogueren accedir a algunes funcions de la finestra principal mitjançant la API IPLUGIN.

fun_frm_Main s'encarrega únicament de la part gràfica de la finestra principal permetent amagar i mostrar la barra d'eines, mostrar la barra d'informació, modificar la grandària dels objectes, etc.

La funció frm_Main conté un parell de mètodes interessants per a ser estudiades. Entre elles hi ha principalment dues. La càrrega i descàrrega de mòduls al JPanel.

La càrrega de mòduls és un mètode molt senzill. Ja que simplement agafa la classe

PluginInterface disponible al vector de mòduls que te desat i executa el mètode *Load* que te implementat. El mètode Load carregarà el seu *JPanel* dins de la finestra principal gràcies a l'API IPLUGIN que disposa d'un mètode de comunicació amb la classe *fun_frm_Main* anomenada GUI.

De la mateixa manera que es carrega el mòdul, ho fa per a descarregar-los. Així doncs, quan la finestra principal rep la senyal de tancament. Obté la llista de mòduls i d'un en un va cridant a la funció *Stop* per a que finalitzen correctament.

3.6.1.1.4 L'Inici de l'ODR

Malgrat que quan nomenem l'ODR o Open Data Resource (Recurs de Dades Obertes) ens referim a la classe que existeix a IPLUGIN, la seva primera declaració està ubicada al paquet kuasar. L'ODR és un sistema per tal que tant els mòduls com el front-end pugin compartir dades. Degut a que els mòduls no poden estar en contacte directe amb el front-end és necessari que l'API, que fa d'intermediari, tingui cap mètode per tal de posar en contacte els uns amb els altres. No obstant es preguntarà per què no s'ha implementat tot dins de l'API. En realitat hauria d'haver estat així de no ser perquè quan és crida a una funció d'IPLUGIN, aquest no sap de quin paquet procedeix. Per tant qualssevol mòdul podria modificar dades o esborrar-les desestabilitzant el sistema. Per aquesta raó era necessari un mètode per a evitar que un mòdul pogués editar variables del front-end, ja que aquest desa variables tan importants com el llistat de plugins o el directori d'inici. Per aquesta raó és va determinar el següent mètode:

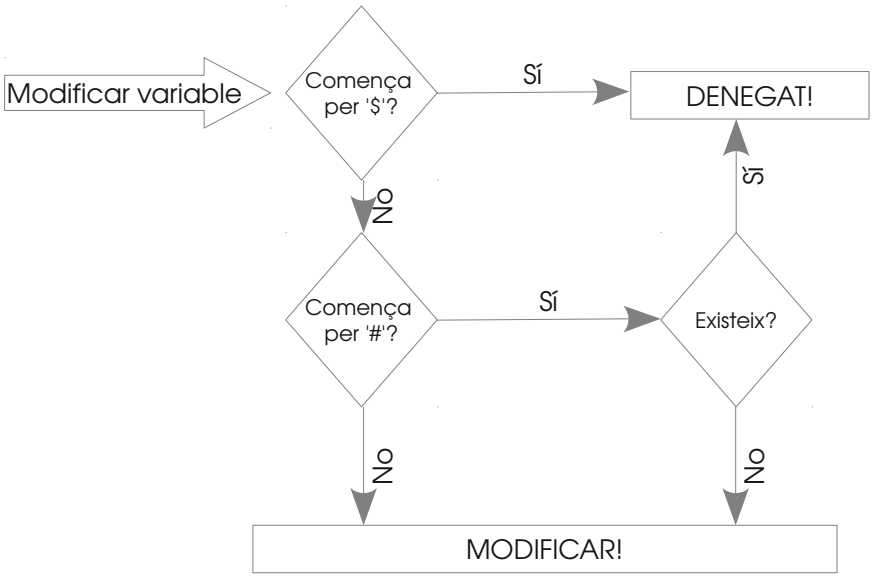
Com l'ODR treballa sota un HashMap per a desar les dades, es crearien 3 tipus de claus.

- Les que hi començaren pel símbol '$' serien interpretats com a variables del front-end i serien de només lectura per als mòduls, i de total accés per al front-end.
- Les que començaren pel símbol '#' serien aquelles variables de lectura-escriptura, però de només un ús. És a dir si la clau no existeix es permet escriure la variable, però si aquesta ja ha estat establida, no es podrà esborrar ni modificar per cap altre que no sigui el front-end.
- Finalment totes les demés claus serien de lectura escriptura per tothom. Per definició una clau ha d'estar formada per <nom_plugin>.<clau> encarà que aquest mètode no és obligatori.

Degut a que IPLUGIN no podia detectar qui estava demanant l'escriptura de la

variable es va decidir crear el HashMap dins el front-end i implementar les restriccions en una altra classe per a què pogués ser cridat des de l'API. D'aquesta manera els mòduls accedirien sempre amb restriccions mentre que el front-end podria accedir de manera directa i sense restriccions a l'ODR.

Així doncs si pegarem un colp d'ull a la classe *ODRIntercom.java* al paquet *kuasar.util.config* observaríem el conjunt de mètodes que permeten la inserció(s*etValue(), putAll()*) i extracció(*getValue()*) de les dades amb restriccions, mentre que el front-end pot accedir directament a l'ODR situat a la classe *Configuration.java* a *kuasar.util.config.*



3.6.1.2. API (IPLUGIN)

Com ja hem dit IPLUGIN és una API que permet la comunicació entre els mòduls i el front-end. No obstant, aquest no és el seu únic propòsit. Així doncs IPLUGIN també conté algunes utilitats implementades per tal que el codi no es duplique en els diferents mòduls. Al paquet *kuasar.plugin.utils* podem trobar classes com:

- *Connection.java* destinat a connectar-se al servidor i gestionar les dades d'accés.
- *DNIe.java* que permet la comunicació amb el DNI per a signar els *tokens* d'autenticació.
- *SSocketTootls.java* conté una sèrie d'eines i utilitats per a enviar dades mitjançant SSL Sockets. Com pot ser enviar cadenes, binaris, números, etc.
- *XML.java:* Conté tots els mètodes necessaris per a crear documents XML com per afegir nodes, esborrar i modificar-los.

IPLUGIN conté dos classes principals de comunicació amb el front-end per una banda l'ODR que ja hem parlat al punt 3.2.1.3 i que si s'observa només crida als mètodes

implementats a la classe *ODRIntercom.java* del front-end.

I per altra, la classe *GUI.java* que serveix per a cridar als mètodes declarats a la classe *fun_frm_Main.java* per tal d'accedir als mètodes de gestió bàsica de la finestra principal.

Quan intentem programar aquest tipus de mètodes ens topem amb alguns problemes impossibles de resoldre perquè no hi cap manera de dir exactament el tipus de mètode que anem a cridar i ni tan sols el tipus de classe on està. Per tant l'ha d'executar a les palpentes, creient que allò que se li ha programat és cert. És per això que apareix el codi de supressió d'alertes '*@SuppressWarnings("unchecked")'* abans de cridar als diferents mètodes. Si per cap motiu es canvies el nom d'alguns d'aquests mètodes, també s'hauria de canviar, als llocs on se'ls crida. Ja que en cas contrari, no hauria cap tipus d'error, fins que aquest fos cridat alguns d'aquest mètodes mentre està en execució.

3.6.1.3. El creador de projectes, VMCreator

VMCreator és el mòdul que permet crear projectes formats per màquines virtuals que volen ser implementats en sistemes reals. Aquest mòdul no crea res, simplement recopila tota la informació necessària per a què puguen ser instal·lades en els hipervisors (sistema operatiu, memòria RAM, interfícies de xarxa, etc.).

Aquest mòdul intenta facilitar la creació de xarxes, permetent crear màquines preferides per a ser afegides més endavant a un altre projecte. Un exemple del que estem intentant explicar, es per exemple, voler crear varies xarxes formats per servidors del mateix tipus (Servidor DNS, Web, etc.). Utilitzant aquest mòdul només caldria crear un servidor de cada tipus (p.e. DNS) i desar-ho com un preferit. Aleshores a cada xarxa que necessités un servidor DNS se li podria afegir aquest que hem creat, sense tindre'l que tornar a configurar.

Com qualssevol mòdul la primera classe que carrega el front-end està indicat al paquet *META-INF.services* sota el nom de *kuasar.plugin.PluginInterface.* Doneu-vos compte que no és un directori, sinó el nom del fitxer. Aquest és el que indica quin és el mòdul que ha de carregar i que implementa a *PluginInterface*. En el cas del mòdul VMCreator, apareix la classe *VMCreator* situat al paquet *kuasar.plugin.vmcreator.*

Si ens fixem en aquesta classe observarem que implementa a PluginInterface (IPlugin) i per tant implementa tots els seus mètodes:

- **getName()**: Retorna el nom del mòdul que es vol mostrar.
- **getPluginName():** Retorna el nom real del mòdul

- **getIcon()**: Retorna la icona del mòdul.
- **Start()**: S'inicia quan es carrega el mòdul
- **Load():** Es crida quan el client desitja fer ús d'ell (Selecció al JList de la finestra)
- **unLoad():** S'accedeix quan el client canvia de mòdul
- **Stop():** Es iniciat quan el programa vol tancar-se

El primer que fa el mòdul VMCreator quan és carregat, es inserir a l'ODR un conjunt de dades de configuració per a què puguen accedir la resta de mòduls. Així per exemple, publica el directori on situa les seves dades amb la clau *vmcreator.path,* el fitxer xml on desa les xarxes virtuals (*vmcreator.data)* o el directori on es desa la configuració de les diferents màquines (*vmcreator.nodes).* Hi ha d'altres variables menys importants que passarem per alt degut a que no es de gran importància per aquesta memòria.

Una volta el mòdul sigui carregat només caldrà esperar a que el client el seleccione per a mostrar-lo al JPanel. Quan això ocorre es crida directament al mètode Load qui s'encarrega de crear el panell (*pn_Main.java*) i la barra d'eines (pn_ToolBar) que s'ha de visualitzar a la finestra principal. Al mateix temps crida a la funció GUI.loadPlugin per a què es mostre el panell i GUI.loadToolBar per a què es mostre la barra d'eines. Fixeu-vos que les funcions que crida estan a la classe GUI que pertany a l'API i no al front-end. Per la qual cosa el front-end i el mòdul mai estan en comunicació directa. En aquest punt l'usuari ja pot vorer i interactuar amb el mòdul mitjançant el JPanel. Aquest mòdul, malgrat que està dins de la finestra principal, no te cap comunicació directa amb ella i per tant haurà de continuar utilitzant l'API si li en vol fer ús d'ella.

Cal també esmenar que el JPanel que se li passa com argument és una mica especial, ja que aquest en comptes de passar-li un JPanel li passa una classe *AbstractPanel* (*kuasar.plugin.classMod)* compartida per IPLUGIN. Aquesta classe és un Jpanel que se li ha modificat la seva forma i color per adequar-se a l'entorn gràfic de l'aplicació. Així doncs la diferencia amb el JPanel és que aquest te formes arrodonides i conté una capa alfa que el fa semitransparent. És obligatori que quan es cride a GUI.loadPlugin se li passe un AbstractPanel en comptes d'un JPanel, ja que produiria una excepció.

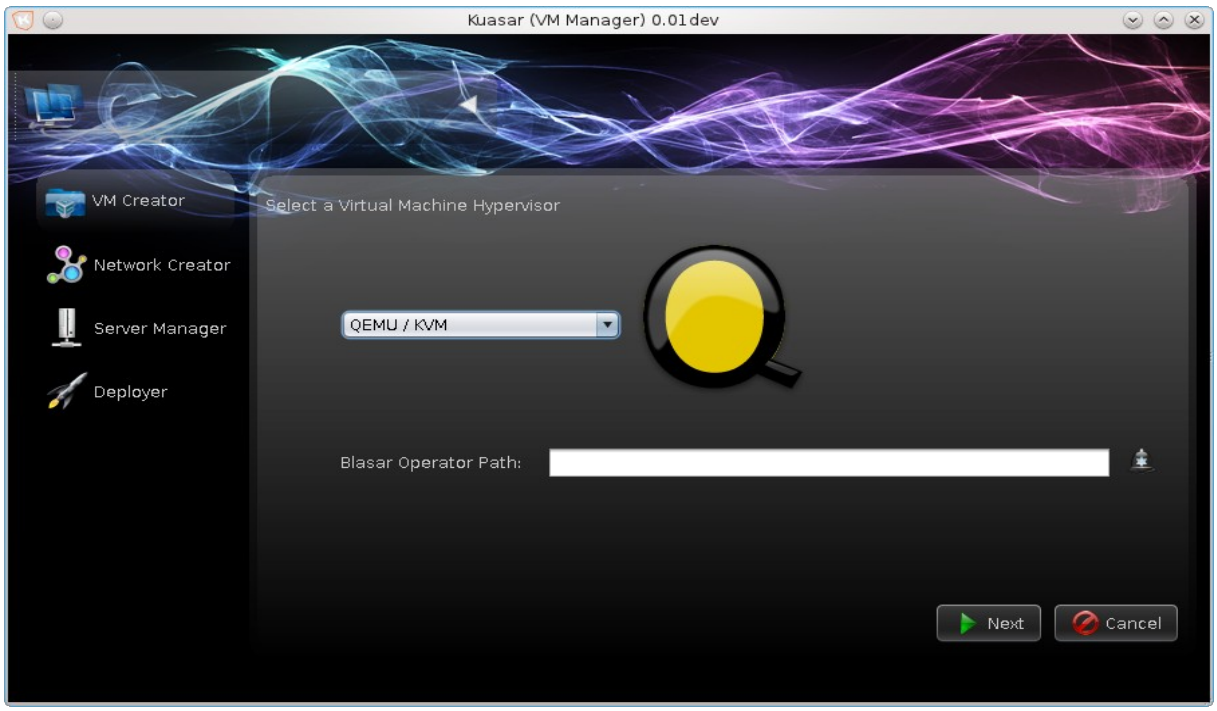
Tornant al Panell principal, el primer que realitza VMCreator es cercar el fitxer de dades xml (per defecte */plugins/vmcreator/network*). D'aquest fitxer diferencia el que son directoris del que son nodes llegint l'atribut type. Si aquest està buit es un directori, mentre que si conté la cadena vm significa que és una node màquina que conté informació sobre la

configuració de les màquines virtuals. A partir d'aquest moment el mòdul comença a carregar dades a la llista diferenciant si són directoris o si son nodes. Per a obtindre el nom d'aquests, llegix l'atribut name i si es un node màquina també llegix la icona del sistema operatiu que conté la màquina utilitzant l'atribut icon. Recordar que totes les gestions es fan mitjançant tant la classe XML com les eines de la llibreria JDOM[7].

A partir d'aquest punt l'usuari pot crear i esborrar directoris (també anomenats projectes) així com crear, modificar i esborrar màquines. De totes aquestes accions només estudiarem com es creen els nodes de màquines virtuals, ja que els demès, només fan ús de les eines XML que hi ha disponibles.

3.6.1.3.1. Creació de nodes màquina

Tots els mòduls de kuasar a excepció de deployer són mòduls de recopilació de dades. També és el cas de VMCreator. L'apartat de creació de nodes màquina és al fi i al cap un seguit de finestres on l'usuari ha d'anar introduint informació sobre la màquina que es vol implementar.



Totes aquestes dades són rebudes i emmagatzemades dins d'un HashMap que ens permet accedir d'una manera més ràpida i natural del mateix mode que si fos una Base de Dades SQL. Però, anem per parts.

El primer paquet que accedeix el mòdul quan un usuari vol crear un node màquina

7 www.jdom.org

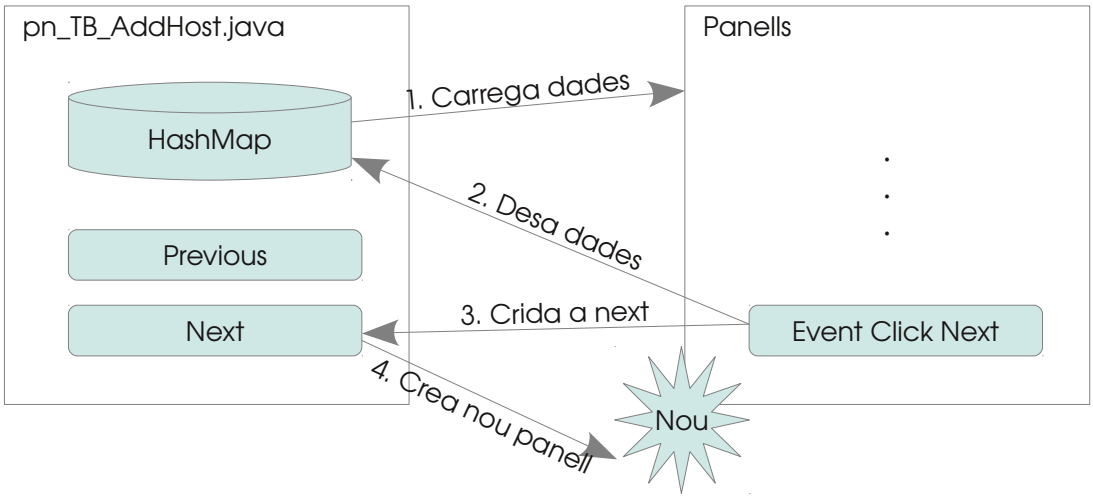
és a *kuasar.plugin.vmcreator.gui.tooltasks.AddHost* si us fixeu, és el paquet més gran que hi ha a tot el mòdul, tan o més gran que la resta de paquets junts. Es per això que la major part d'informació que gestiona aquest mòdul està concentrat en aquest paquet. També cal aclarir que el mateix paquet no solament s'utilitza per a crear un node nou, sinó també per a modificacions d'altres ja creats.

El primer que realitza el mòdul al voler crear un node màquina és crear una barra d'eines (*pn_TB_AddHost.java*) i un panell (*pn_AddHV.java*) al que se li passa com a argument la barra. En aquest cas la barra d'eines no actua com a tal, sinó més bé com una eina de navegació entre les diferents pantalles. A banda d'aquesta utilitat de navegació, la barra també serveix per a gestionar els diferents panells que es mostraran així com a contenidor del HashMap on cada panell ha de desar les dades que ha recopilat. Fixeu-vos que *pn_TB_AddHost.java* te declarat el HashMap (*HashMap<String, Object> data*) com a protected i per tant tots els panells poden accedir a ell directament.

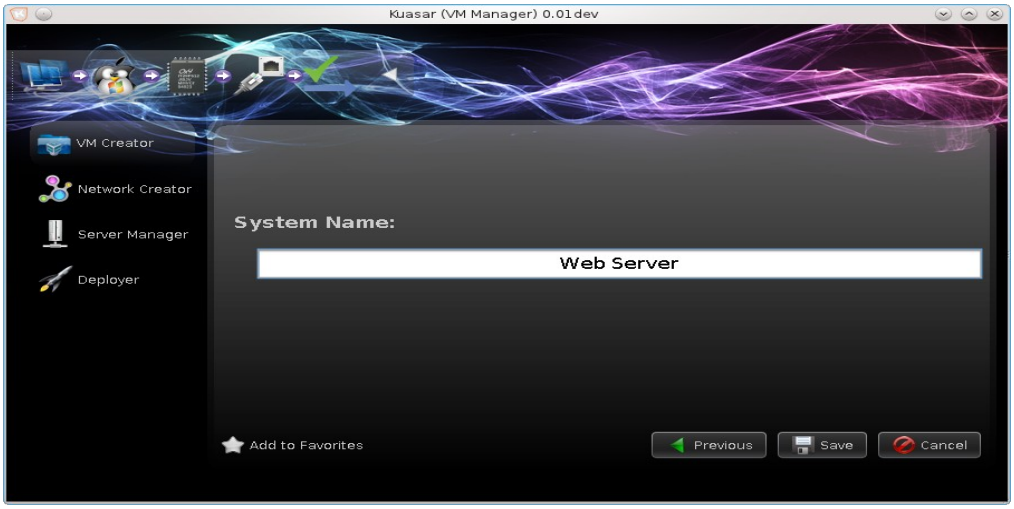
Així doncs, una volta s'ha carregat tot, la primera pantalla cerca si hi ha dades (funció *Load*) disponibles al HashMap. En cas que existiren s'entendria que es una modificació el que es pretén i les carregaria a la pantalla.

Les dades no serien desades de nou fins que l'usuari no premés següent.

Si es prem el botó següent el panell, aquest crida a la funció *next,* implementat a la barra d'eines, per a què carrege el següent panell.



Tots els panells realitzen la mateixa tasca a excepció de l'últim...



L'última pantalla s'encarrega de desar tota la informació (que recordem es realitza en RAM) a un fitxer del disc dur. En aquest cas i per a facilitar l'eina de desat de dades es va realitzar una serialització de les dades o en altres paraules, es va passar l'objecte HashMap a fitxer binari. D'aquesta manera es simplificava el desament de les dades i la posterior recuperació i modificació d'aquestes.

El sistema de desament dels nodes es basa en l'auto-increment de números per tal d'evitar noms duplicats al mateix directori. D'aquesta manera una volta es vol desar el node el programa comença a generar números i comprovar si existeixen fitxers formats pel número generat seguit de l'extensió .bin. Quan troba un número lliure comença la serialització del HashMap de dades i les desa al fitxer. Finalment quan el fitxer a estat desat es crea una entrada al fitxer de nodes xml (*network*) amb el nom que se li ha donat a la màquina, i el nom del fitxer que se li ha donat. Aquest nom de fitxer es guarda a l'atribut path. Exemple:

<Client name="Client" type="vm" icon="windows.png" path="0.bin">

En aquest cas existeix un node màquina (type=”vm”) amb el nom “Client” i on el seu fitxer de dades és el “0.bin”. Aquests fitxers són desats generalment a la carpeta “nodes” dins de la carpeta de configuració de VMCreator (normalment /plugins/vmcreator).

3.6.1.4. El Creador de xarxes, netCreator

El següent mòdul que anem a estudiar és netCreator.

netCreator és el mòdul que permet gestionar les adreces de xarxa de cada màquina que hem creat. Així per exemple, mentre que VMCreator es recopilava informació destinada a la configuració de l'hipervisor, aquest està destinat a la distribució de les adreces IP en els diferents projectes que volem implementar. netCreator disposa de dos formes de distribuir les adreces. Una manual, que permet a l'usuari donar a cada màquina la IP que l'usuari desitge, i una altra automàtica capaç de, donades una IP de xarxa i una màscara, distribuir el rang d'IPs disponibles entre els equips. Això comporta alguns controls per a evitar adjudicar adreces incorrectes o fora de rang. Per altra banda l'adjudicació automàtica és compatible amb la manual, i així una adreça que ha estat adjudicada de manera automàtica pot ser bescanviada per una altra de manera manual.

El propòsit d'aquest mòdul, és doncs, obtenir les dades necessàries per tal que una volta la màquina s'haja instal·lat a l'hipervisor, puga interactuar dins la xarxa.

El mètode d'arrancada d'aquest mòdul és el mateix a quant als altres mòduls. Al paquet *META-INF.services* apareix el fitxer *kuasar.plugin.PluginInterface* que mostra quin és la classe que implementa a *PluginInterface* i per tant la que ha de ser carregada en primer ordre. En aquest cas *netCreator.java* al paquet *kuasar.plugin.netcreator.* Les funcions que conté i el seu ús és el mateix que el que apareix a VMCreator i per tant no cal posar la vista en elles de nou a excepció d'aquella que s'encarrega de mostrar el panell d'inici.

En aquest cas netCreator necessita obtindre tant els directoris de configuració, com els fitxers de dades de VMCreator per a poder treballar, ja que aquest mòdul ha d'assignar adreces a màquines que han estat creades pel mòdul VMCreator. Per aquesta raó, abans d'agafar qualssevol dada ha de comprovar que VMCreator haja estat carregat correctament. Així doncs connecta amb el sistema ODR i pregunta pel llistat de mòduls carregats.

(PluginInterface[]) ODR.getValue("$PLUGINS")

A continuació comprova un per un que el nom del mòdul estigui a la llista.

plugin.getPluginName().equals("vmcreator")

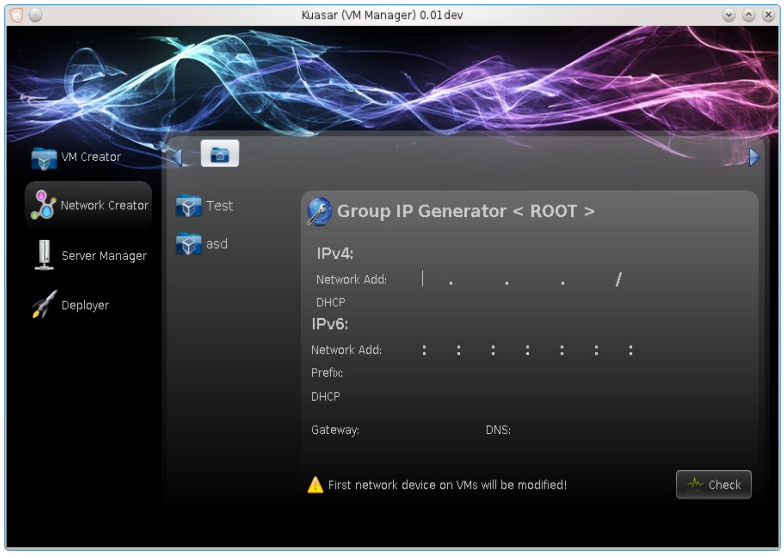
Malauradament, no hi ha cap forma d'identificar el mòdul tret que sigui pel nom per tant, si VMCreator canvies el seu nom en un futur netCreator ho deuria tindre present i

canviar el seu detector.

En cas de no trobar el mòdul, retornaria al front-end un false, el que suposaria que hauria hagut un error. Automàticament el front-end mostraria, al JPanel on deuria obrir-se el mòdul, un missatge d'error aconseguit per la funció getError de la classe netCreator.

En cas contrari, i per tant, suposant que VMCreator hagués estat carregat amb èxit, netCreator continuaria de manera normal, obtenint les variables necessàries de l'ODR. Finalment carregaria el seu panell (pn_Main) a la finestra principal mitjançant la classe GUI de l'API. (*GUI.loadPlugin).*

El panell principal de netCreator es troba a la classe pn_Main.java.



La gestió de les dades és molt semblant a la de VMCreator. Així, per exemple, el primer que fa és una càrrega dels directoris i nodes màquina disponibles al fitxer network. Del mateix mode, netCreator els carrega a una llista per a què l'usuari puga navegar pels diferents nivells. No obstant netCreator no treballa sobre els fitxers de dades de les màquines (directori nodes) si no solament amb el fitxer network, ja que només l'interessa el número de màquines que hi ha a un directori. Recordeu que un directori i un projecte és el mateix.

netCreator té dos maneres d'assignar les adreces depenent si l'usuari selecciona un directori, mode automàtic, o bé si el que selecciona és un node màquina, mode manual.

En el cas automàtic, quan un usuari selecciona un directori per a assignar IPs automàticament, carrega el panell pn_GroupNetwork.java on se li demana a l'usuari unes mínimes dades per a configurar la xarxa com és la IP, la màscara, els servidors DNS i la

porta d'enllaç. Al mateix temps permet seleccionar si vol que a la màquina se li assigne una adreça per DHCP.

En aquest cas, quan l'usuari ingressa totes les dades i prem el botó comprovar s'inicien un seguit de comprovacions:

- Que la IP estigui ben formada
- Identificar el tipus de versió de la IP (4 o 6)
- Comprovar la màscara o el prefixe
- Comprovar que la IP és una adreça de xarxa
- Contar els números d'adreces disponibles
- Contar les màquines per assignar
- Comprovar que hi ha suficients adreces per a totes les màquines
- Etc.

Quan totes les dades han estat comprovades *pn_GroupNetwork* crida a *pn_SaveNetwork* en *kuasar.plugin.netcreator.gui.network* per a què s'encarregue de distribuir les adreces de cada màquina que trobe dins del directori, de manera recursiva en el cas que un directori estiga format per altres directoris més.

Per a desar la configuració de la xarxa, *pn_SaveNetwork* introdueix uns arguments nous a cada node màquina al fitxer xml *network*. Aquest són:

ipv4: La IP en versió 4

ipv6: IP en versió 6

mask: la màscara IPV4 en format prefix

mask.full: la màscara en format per blocs

prefix: el prefix per a ipv6

gw: la porta d'enllaç

dns: els servidors DNS separats per comes.

Exemple de com apareix un node màquina amb IP assignada.

<Client name="Client" type="vm" icon="windows.png" path="0.bin" ipv4="192.168.250.1" mask="26" mask.full="255.255.255.192" ipv6="" prefix=""

gw="192.168.250.25" dns="8.8.8.8,8.8.4.4" />

Una volta totes les dades han estat assignades es desa de nou el XML i s'informa a l'usuari.

En cas que l'usuari vulga afegir una adreça de manera manual, es crida a la classe *pn_VMNetwork.java* situada a *kuasar.plugin.netcreator.gui.network*.

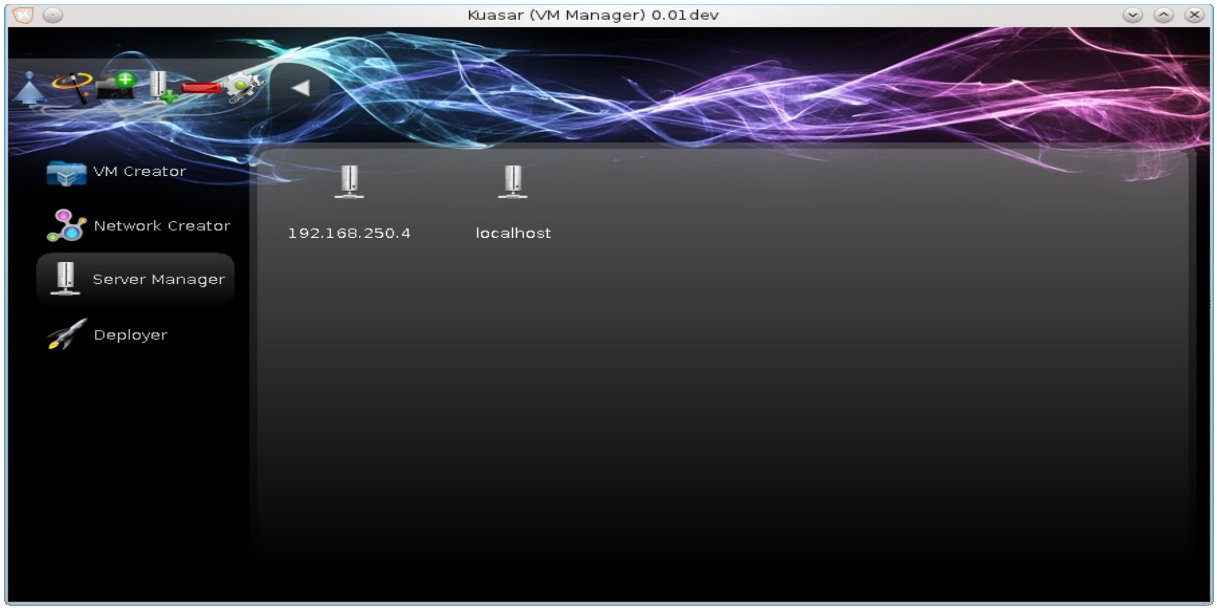
La funció que realitza aquesta classe és molt més lleugera i senzilla d'implementar perquè només ha de comprovar que les adreces siguin correctes i que no siguin especials. Abans de res, la classe carrega les dades del node amb el que vol treballar. Una volta s'accepten les dades, aquestes són desades al fitxer *network* per a un ús posterior.

Indicar com a curiositat que quan es vol assignar per DHCP l'adreça assignada pren el valor 0.0.0.0.

3.6.1.5 El gestor de servidors. serverManager

El mòdul serverManager s'encarrega de gestionar tots i cadascú dels servidors que contenen un servei blasar. Bàsicament s'encarrega de crear una base de dades amb totes les màquines disponibles i que permeta a l'usuari ordenar-les i gestionar-les.

De la mateixa manera que els altres mòduls, serverManager s'inicia des d'una classe que implementa a PluginInterface. En aquest cas s'anomena *ServerManager.java* situat al paquet *kuasar.plugin.servermanager*. A l'apartat Load trobem quin serà el panell que es carregarà a la finestra principal, que de la mateixa manera que els altres mòduls pren el nom de *pn_Main* al paquet *kuasar.plugin.servermanager.gui*



serverManager te un aspecte molt semblant a VMCreator, però a diferencia d'aquest últim no s'organitza de manera multinivell, es a dir no existeix la possibilitat d'organitzar els servidors en subcarpetes.

Quan serveManager es seleccionat el primer que realitza és una càrrega de dades del seu propi fitxer XML anomenat *servers* i situat al directori /plugins/servermanger. Aquest fitxer XML és molt més simple que el networks que utilitzen els anteriors mòduls ja que les dades a recollir no són molt extenses. Així doncs al fitxer podem trobar-nos el següent codi:

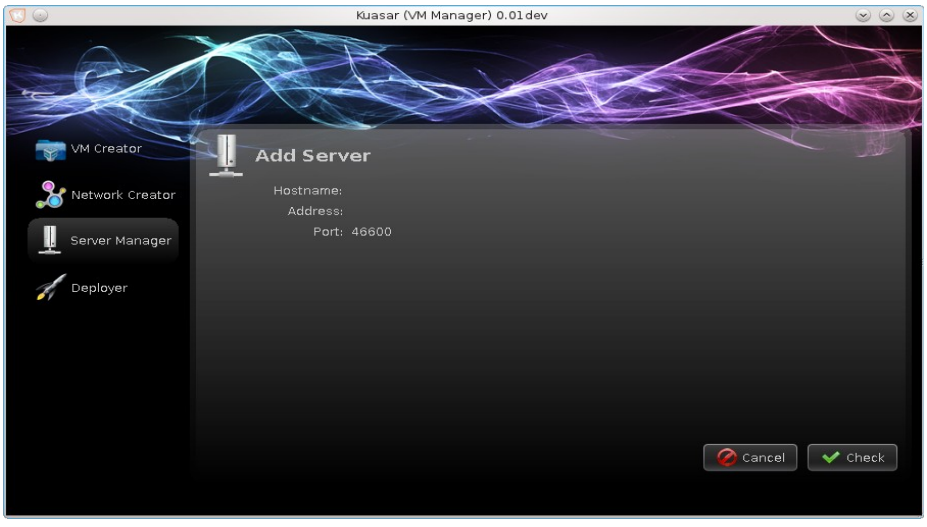
<_192.168.250.4 type="server" name="192.168.250.4" address="192.168.250.4" port="46600" />

Del mateix mode trobem els arguments type i name que indiquen si és un directori o un node (en aquest cas servidor) però hi apareixen altres dos tipus de dades importants. *address*, l'adreça del servidor i *port*, que indica el port per on escolta el servei.

Aquestes són les úniques dades bàsiques que ha de desar serverManager i que permeten connectar-se al servei Blasar.

serverManager bàsicament permet crear i esborrar directoris que serveixen per a ordenar els servidors segons les nostres necessitats i crear i esborrar els nodes servidors. Totes aquestes accions es realitzen de manera similar a com es fa a VMCreator i no anem a entrar al detall, encara que si que mostrarem com es crea un node servidor.

serverManager disposa de dos formes d'afegir servidors, una de manera automàtica, cercant serveis en una xarxa o bé manualment indicant-li la IP i el port on s'ha de connectar. Tot i que ambdós semblen mètodes diferents, només es distingeixen en què l'automàtic fa una comprovació per cada IP de la xarxa.



Quan un usuari vol afegir un servidor automàticament apareix un nou panell (*kuasar.plugin.servermanager.gui.actions.pn_AddServer.java*) on es sol·licita el nom d'usuari, l'adreça i el port on s'ha de connectar. Aquesta pantalla no realitza cap tasca més que la de recopilar la informació per a connectar-se i comprovar que les dades han estat introduïdes correctament.

Una volta l'usuari i el panell han acceptat les dades de connexió *pn_AddServer.java* crea un panell (kuasar.plugin.servermanager.gui.actions.*pn_CheckServer.java*) que s'encarrega d'efectuar la comprovació i l'insereix dins d'un altre JPanel que té com a subpanell. Aquest panell implementa a una classe Thread que permet treballar sense blocar el programa principal i evita que la interfície gràfica quede bloquejada. D'aquesta manera l'usuari no te la sensació que el programa ha deixat de funcionar.

A l'iniciar *pn_CheckServer.java* com a fil comença a realitzar cridades a la classe *Connection.java* disponible a IPLUGIN i que ja hem anomenat a principi de capítol.

El primer que comprova es si el servidor li respon a una petició echo. Aquesta prova simplement es de mera informació per a l'usuari ja que si respon vol dir que la màquina està encesa però que alguna cosa li passa al servidor (Tallafoc, servei no iniciat, etc.).

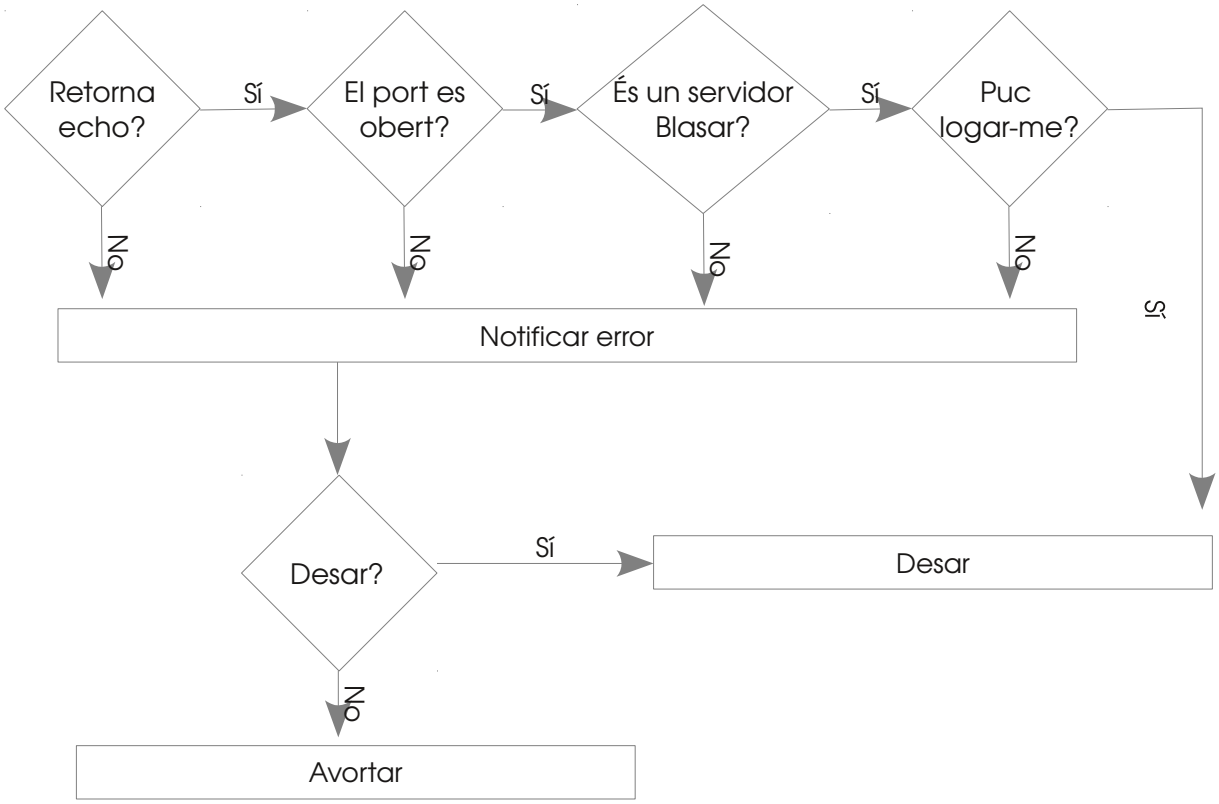
La segon prova que realitza és comunicar-se amb el port. Per a d'això comprova si està obert intentant fer una petició de connexió TCP/IP. Aquesta prova també és informativa ja que podria ser que en cas d'error un altre servei estigués escoltant per eixe port.

La tercera prova realitza una connexió completa obtenint informació del servidor i observant si les contestacions que realitza el servidor pertanyen a un servei blasar.

L'última prova és comprovar si l'usuari té accés o no al servidor. En cas que aquest fos denegat l'usuari deuria comprovar si ha inserit correctament el nom d'usuari i contrasenya o si està donat d'alta al servei.

Malgrat que totes aquestes proves estan disponibles, l'usuari pot indicar quines proves vol realitzar, tot just accedint al panell de configuració del mòdul.

Després de comprovar l'estat de les proves pot ocórrer dos situacions. Si totes les proves han estat satisfactòries es desa les dades del servidor i automàticament torna al panell principal. En canvi si ha hagut qualssevol error s'informa d'aquest a l'usuari i permet elegir entre desar el servidor al fitxer XML o cancel·lar el procés.

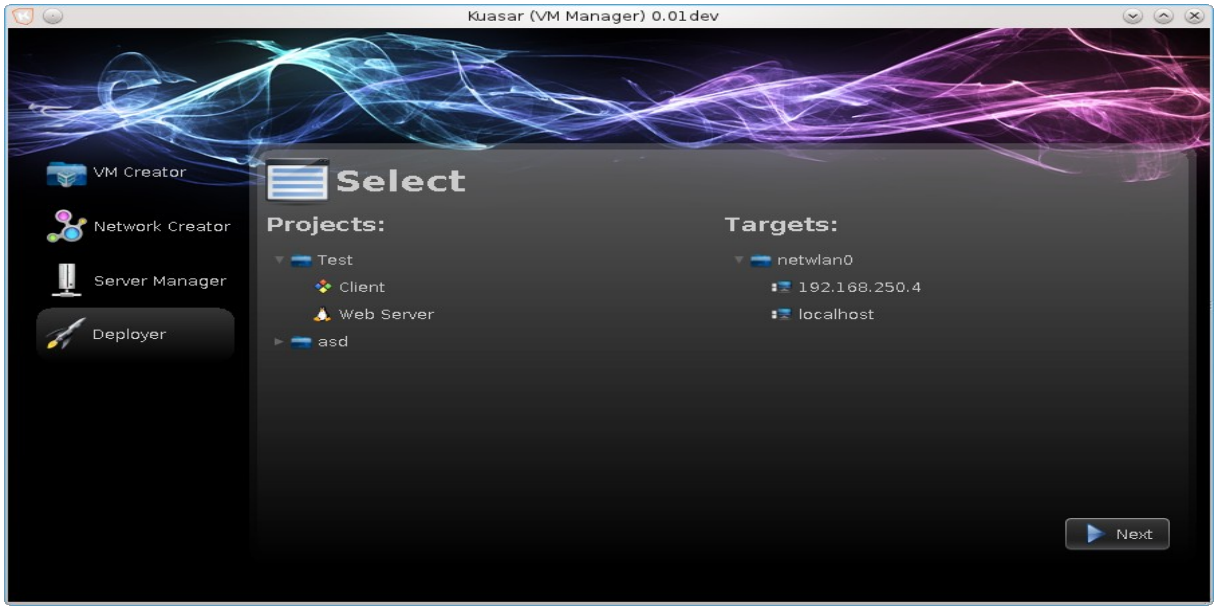


3.6.1.6. L'implementador. Deployer

Deployer és el mòdul que culmina la recopilació de totes les dades i les posa en practica per a poder instal·lar el nostre projecte als diferents hipervisor. La funció de deployer es extraure totes les dades recopilades i transformar-les en comandaments per a què puga ser interpretat pel servidor blasar i puga crear una màquina virtual segons la nostra configuració.

El mòdul deployer és un seguit de panells tal com si d'una instal·lació es tractés que va guiant-nos pel procés d'instal·lació dels projectes, així també ens ajuda a observar el desenvolupament del procés i a conèixer l'estat de cada màquina instal·lada.

A l'igual que els altres, Deployer comença per una classe que implementa PluginInterface. Aquesta vegada *deployer.java* a *kuasar.plugin.deployer.* Com és normal deployer necessita de les dades recopilades per tots els altres mòduls. Per aquesta raó la funció Load denega l'accés si no estan tots els mòduls carregats.



El panell de benvinguda de Deployer, a diferencia de la resta, comença per una classe anomenada *pn_Targets.java.* Aquesta bàsicament s'encarrega d'extraure la informació dels fitxers de dades dels anteriors mòduls (network i servers) i els carregar a la columna que pertoca. L'usuari ha de seleccionar doncs un objecte de cada columna, per una banda el projecte que vol instal·lar i per l'altra a quin servidor, o conjunt de servidors vol instal·lar les màquines. En el moment en què un usuari ja ha seleccionat que s'ha d'implementar i a on, el mòdul crida al panell *pn_Checker.java.*

pn_Checker crea una nova classe(th_checkVMs.java) i l'executa a un nou fil, mentre serveix com a informador de l'evolució de les dades a *th_checkVMs.*

th_checkVMs és qui s'encarrega d'extraure i comprovar que no hi ha cap error en les dades recopilades i que tots els servidors estan disponibles. Totes aquestes tasques no les fa la pròpia classe si no que va cridant a d'altres que s'encarreguen de tasques més específiques:

DataExtractor la missió d'aquesta classe és extraure els fitxers de dades de cada node màquina i transformar la informació que contenen en format XML per a què siga més senzill treballar en ell. A més, elimina totes les dades "brossa" que son només útils per al mòdul VMCreator deixant la informació realment necessària. El problema amb els HashMap on es desava les dades de VMCreator residia en que era molt senzill d'introduir objectes de diferents tipus, però no era molt eficient ni senzill afegir dades estructurats en arbre. Per aquesta raó, i com ja no calia inserir dades en diferents tipus d'objectes(JList, Jtree, etc.) es veia més convenient transformar les dades a un sistema estructurat en arbre com és XML.

ImageChecker la funció d'aquesta classe és la de comprovar totes i cadascuna de les imatges del projecte. La comprovació que és realitza és bàsica detectant simplement si els fitxers imatge existeixen i si es possible la seua lectura.

Finalment ***ServerChecker*** és l'encarregat de comprovar que tots els servidors funcionen correctament i estan disponibles. Per a realitzar totes les comprovacions s'utilitza la classe Connection.java a IPLUGIN i que ja ha estat anomenat anteriorment.

Si alguna verificació resultés fallida s'instaria a l'usuari a corregir els errors, a ignorar-los, o bé l'avortament del procés. En cas que foren ignorats les màquines afectades serien excloses del projecte.

Si tot el procés s'ha dut amb èxit, es carrega *pn_Strategy,java* que s'encarrega de donar a l'usuari dos opcions per a allotjar les màquines. Allotjar per memòria RAM (aquell sistema que tinga més RAM va primer) o bé per espai lliure (el servidor amb més espai te preferència).

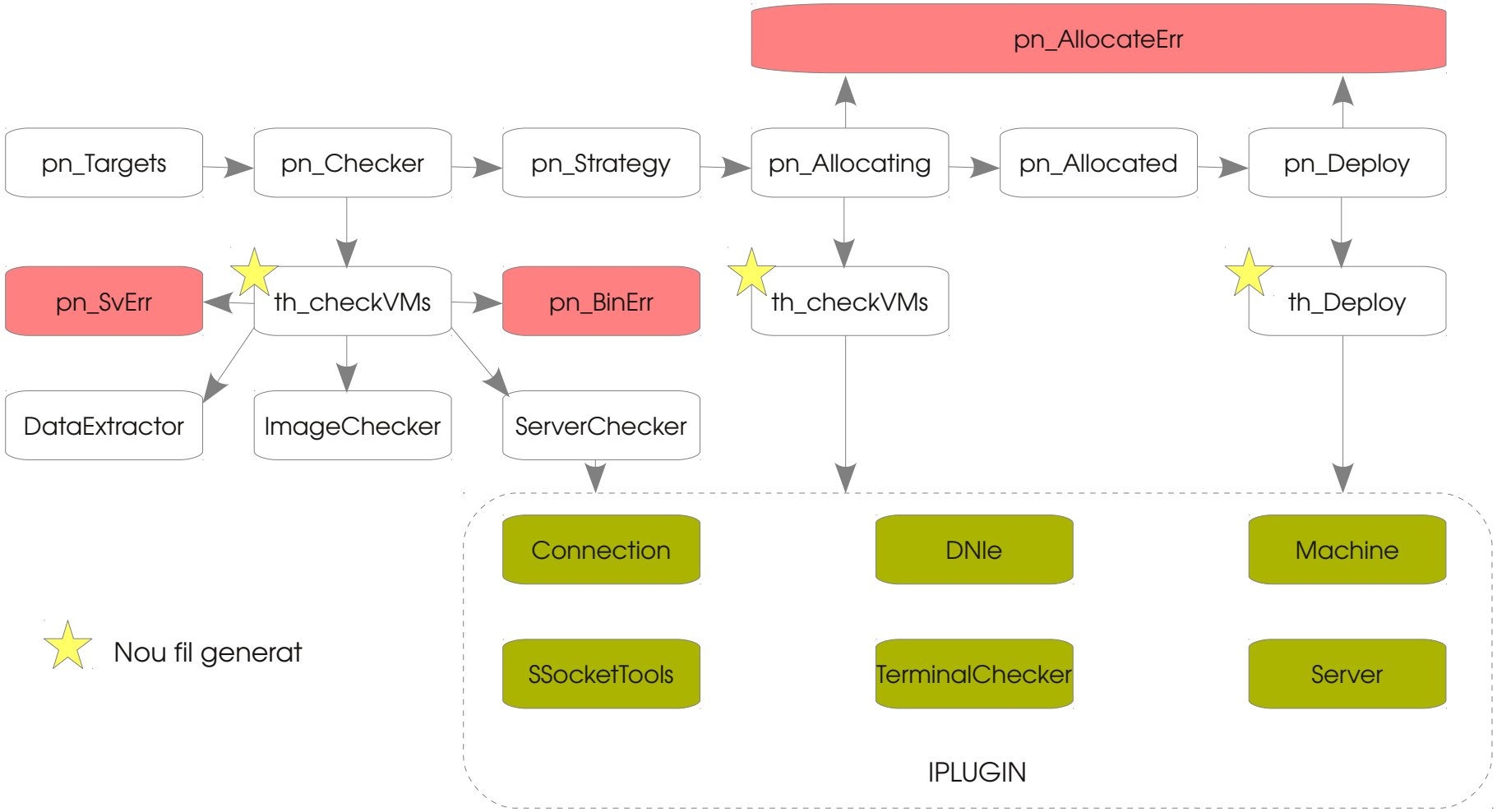
Quan l'usuari ja ha pres la decisió, el mòdul carrega el nou panell *pn_Allocating.java* qui s'encarregarà d'obtindre informació dels recursos disponibles als servidors mitjançant la

classe *th_AllocateVMs* en un nou fil. Després comprovarà i ordenarà les dades per a obtindre els millors servidors per a allotjar les màquines. En cas de no trobar cap servidor per a una màquina apareixerà un panell d'informació *pn_AllocateErr.java* i s'avortarà la instal·lació per no poder-se complir les necessitats del projecte.

En cas contrari, el mòdul carregarà *pn_Allocated.java* qui s'encarregarà de mostrar de manera informativa quins seran els servidors que allotjaran les diferents màquines virtuals, per a què l'usuari puga acceptar o avortar el projecte.

Finalment, si l'usuari accepta tot, Deployer carregarà l'ultim panell(*pn_Deploy.java*) abans de la implementació, que s'encarregarà d'informar de l'estat de la instal·lació de les diferents màquines. Tot i que la tasca de comunicació amb els servidors serà duta per *th_Deploy.java.* Que serà creat i iniciat per pn_Deploy en un nou fil.

Esquema de classes del mòdul Deployer



3.6.2. El Servidor, Blasar

Tot i que blasar és la part més important del projecte és un dels que menys codi té implementat, gràcies sobretot a que no té que controlar gairebé cap tipus de dada i només ha de ser transmissor de comandaments entre el client i l'hipervisor.

De la mateixa manera que el client Kuasar, Blasar està format per tres parts encara que no són exactament les mateixes ni tenen una mateixa tasca, així per exemple, mentre que tots dos tenen una API que realitzen la mateixa tasca i funció, a Blasar el front-end de Kuasar és per a ell el cor del seu funcionament, mentre que els mòduls que conformen Blasar són extensions de comunicació amb altres aplicacions.

3.6.2.1 El cor del servidor, Blasar

L'aplicació blasar.jar és l'eina que utilitza Kuasar per a connectar-se amb els hipervisors i gestionar-los. Està creat per dos parts clarament diferenciats als seus paquets. Per una banda és un sistema que oferix serveis a altres aplicacions també anomenat servidor, i per l'altra també te un petit front-end en forma de consola.

Al codi font d'aquest es pot vorer amb total claredat pels noms dels paquets.

Així per exemple trobem dos tipus de paquets *blasar.Services* dedicat a rebre connexions i establir comunicació amb els clients i *blasar.console* que s'encarrega de la gestió del servidor per mitjà d'una consola de text.

La primera classe en executar-se a l'iniciar Blasar és *Main.java* dins del paquet *blasar.* Aquesta classe de la mateixa manera que ho fa Kuasar inicialitza abans de res les variables generals per a poder treballar. Després d'haver carregat les variables comença a carregar els mòduls que troba de la mateixa manera que ho fa Kuasar amb l'excepció que el fitxer que cerca al paquet META-INF.services és *blasar.Services.Com.vms.PluginInterface.* Però tot i així el sistema és el mateix. Una volta carregats i donat el cas configurats els mòduls, es continua amb la configuració del servidor. Primer carregant les opcions des del fitxer de configuració conf.cfg i després des dels arguments passats per consola. Per tant, si hi ha dos valors diferents de configuració te més rellevància aquell que ha estat passat com a argument.

Una volta tot està enllestit per al seu funcionament la funció d'arrancada Main carrega la classe *Server.java* a *blasar.services* i l'inicia en un altre fil per a posar en marxa el servidor. En cas que el client haja demanat que el servidor funcione de manera interactiva,

es a dir amb consola, s'iniciarà també la classe *InitConsole.java* disponible a *blasar.console.*

3.6.2.1.1 El Servidor

El funcionament del servidor ha estat dissenyat des d'un començament per a un ús amb múltiples connexions. Per a permetre aquesta característica cal que quan un client es connecti al servidor, aquest últim siga capaç d'enviar la gestió de la comunicació a un altre fil d'execució, per tal que el fil principal s'encarregue d'atendre als nous clients. I aquesta, és la tasca que realitza la classe *Server.* Quan la funció Main crida a Server i l'arranca en un fil apart aquest es posa a escoltar en un port del sistema, per defecte el 46600 utilitzant el protocol SSL mitjançant l'objecte *SSLServerSocketFactory.*

A partir d'aquest moment el servidor es quedarà a l'espera d'un client nou. Quan aquest client arribe, immediatament es crea una nova classe *Splitter.java* i se li passa com a paràmetre la connexió (socket) per a després iniciar la classe en un nou fil d'execució. Després de realitzar tota aquesta tasca el fil principal torna de nou a l'escolta d'un nou client. A més a més la classe Server registra en un HashMap el fil Splitter que s'ha creat per a poder intervenir en la connexió en cas que fos necessari. Cada fil es desa al HashMap amb una clau única formada per la adreça de xarxa i el port i esta es esborrada quan una connexió ha estat acabada.

La classe Splitter s'encarrega de redirigir a l'usuari per un seguit de zones per tal d'autenticar-lo i en cas de ser acceptat enviar-lo a la zona de serveis o bé, en cas de denegar-se-li l'accés tancar la connexió.

Així doncs quan un nou client arriba a l'Splitter aquest li dona la Benvinguda i s'identifica com a un servidor Blasar mitjançant la cadena "*Welcome to Blasar (VERSIÓ) - The kuasar project's daemon."*

A partir d'aquest moment l'Splitter l'envia a la classe d'autenticació anomenada *Auth.java* a *blasar.Services.Com.*

Auth, simplement és un conjunt de cadenes de comunicació que s'envien segons la contestació del client. Per tant, si Auth rep una contestació incorrecta automàticament rebutja la comunicació i la dona per acabada.

El sistema de comunicació que existeix entre Kuasar i Blasar conté unes regles necessàries per a entendre's i que són d'ús obligatori. Així doncs, qualssevol comandament que s'envie haurà de començar per un dels següents símbols depenent del que es desitge enviar.

'**<**' Si el que es vol enviar es una resposta a alguna petició.

'**>**' Si es vol fer una petició o pregunta.

'**#**' Quan es vol enviar un estat, error, o informació.

Així per exemple quan un client arriba a la classe *Auth,* aquesta li dona la benvinguda i automàticament rep una pregunta per a què a que conteste quin mètode desitja utilitzar per a autenticar-se:

>Auth?

Immediatament la contestació ha de ser

<0 si es vol cancel·lar

<1 si es vol fer la comprovació del nom d'usuari i contrasenya per text pla.

<2 si es vol autenticar per DNIe

Si l'autenticació es fa de manera correcta Auth finalitza de manera correcta, però si en canvi es rebutjada s'inicia una excepció en cadena que acaba amb la finalització de la connexió.

Si Auth acaba de manera correcta Splitter continua amb la redirecció del client i en aquest cas l'envia directament a la zona de serveis d'usuari implementat a la classe *UserService*.

UserService és l'inici de gestió dels clients, és on poden executar els diversos comandaments disponibles per realitzar tasques al servidor o rebre informació d'estat. A partir d'ací totes les classes són molt semblants i no cal estudiar el seu funcionament, ja que més bé el pas entre les diverses classes es realitza de manera molt pareguda a com és navega pel menú d'una televisió.

A continuació mostrem un llista dels comandaments disponibles a cada classe,

UserService	
Exit	Finalitza la connexió
$*	Si comença per $ són comandaments referits a la màquina on s'allotja el server.
listhvs	Llista els hipervisors instal·lats
switchvm	Canvia a les ordres per a un hipervisor específic.

VMService	
getfreespace	Retorna l'espai que queda lliure per a allotjar imatges
getmachines	Obté les màquines virtuals instal·lades
getdefaultfolder	Obté el directori de les imatges per defecte
createvm	Crea una màquina virtual
setmemory	Establix la memòria RAM
addstorage	Afegix un controlador de disc
addnic	Afegix una targeta de xarxa
setnetwork	Configura l'adreça de xarxa
getenginename	Obté el nom de l'hipervisor
getmachinename	Obté el nom de la màquina virtual
getmachineuuid	Obté l'UUID de la màquina virtual
isrunning	Indica si la màquina està corrent
startvm	Inicia la màquina
deletevm	Elimina la màquina
resetvm	Reinicia la màquina
poweroffvm	Atura la màquina
acpioffvm	Aturada neta
Exit	Surt del Servei VM

SysCommands	
os	Informació quant al Sistema Operatiu
cpu	Informació quant a la CPU
mem	Informació quant a la memòria RAM
fs	Informació quant al sistema de fitxers

3.6.2.1.2 La consola

La consola de Blasar, es un petit front-end en mode text que ofereix a l'administrador un conjunt d'eines per tal d'administrar el servidor. La consola de blasar està implementada al paquet *blasar.console* i està formada per només cinc classes, suficients per a realitzar una gestió bàsica.

Aquesta característica pot ser iniciada de manera opcional si s'afegeix l'argument -i al comandament d'inici de blasar o bé si s'estableix al fitxer de configuració la variable interactive d'aquesta manera: '*interactive=true*'

La primera classe que inicialitza blasar per a activar la consola és *initConsole.java.* Una classe que només prepara la consola imprimint per pantalla el símbol del sistema i llegint els comandaments per a ser redirigits a la classe *RedirectCommands.java*

Per a poder controlar les connexions des de consola, a initConsole se li passa la classe Server.java que disposa d'unes petites eines per a controlar i gestionar les connexions.

Així per tant, accedint-hi a través de la classe Server es pot obtindre les connexions establertes, matar-les, conèixer els usuaris que han iniciat sessió i amb quina adreça, etc. A partir d'aquesta ferramenta es basen les següents classes que veurem i que estan lligades a RedirectCommands. Aquesta última classe funciona de la mateixa manera que ho fa la classe UserService on tot funciona com un tipus de menú de tv.

Existeixen tres classes que pengen de la classe RedirectCommands:

KillComands: Que conté mètodes per a matar connexions.

ShowCommands: Basats en comandaments que donen informació

VMUserCommands: Conté mètodes per gestionar els usuaris i contrasenyes.

De la mateixa manera, RedirectCommands te dos nivells d'accés. Com usuari normal, o com administrador. Açò permet que alguns comandaments no puguen ser utilitzats tret que introduïsca una contrasenya d'accés. Per això, les classes KillComands i VMUserCommands només poden ser accedides per un usuari que tinga nivell d'administrador.

La manera d'identificar-se és molt senzilla i està implementada a la mateixa classe RedirectCommands. Quan un usuari accedeix al menú inicial, pot introduïr a la consola el comandament per a pujar a nivell administrador 'su'. RedirectCommands obtindrà el

comandament i cridarà a la funció setSU qui s'encarregarà de demanar-li la contrasenya, xifrar-la a SHA-512 i comparar-la amb la que te en el fitxer supasswd. Aquest fitxer ha d'estar protegit contra escriptura, perquè en cas contrari aquesta eina seria innecessària.

A continuació es mostra els comandaments disponibles i les classes que els gestiona.

RedirectCommands	
show	Mostra informació del servidor
su	Passa a nivell administrador
?	Mostra l'ajuda
logout	Surt del nivell administrador
vmuser	Gestiona els usuaris
encpwd	Genera Hash SHA-512 per a contrasenyes
exit	Finalitza el servidor
kill	Mata connexions
killall	Mata Totes les connexions

ShowCommands	
connections	Mostra les connexions obertes
interfaces	Mostra les les interfícies de xarxa disponibles.
users	Mostra els usuaris connectats
?	Mostra l'ajuda

KillCommands	
kill	
killall	

VMUserCommands	
add	Afegeix un usuari nou
del	Elimina un usuari
list	Mostra els usuaris
password	Canvia la contrasenya d'un usuari
?	Mostra l'ajuda dels comandaments

3.6.2.2 L'API de Blasar. BIPLUGINS

L'API de Blasar te la mateixa funcionalitat i estructura que l'API de Kuasar i funcionen absolutament igual l'una de l'altra. A l'igual que IPLUGIN, BIPLUGINS conté una classe PluginInterface que serveix per a identificar el tipus de classe que Blasar ha d'iniciar quan carrega un mòdul. No obstant PluginInterface te algunes peculiaritats respecte al d'IPLUGIN perquè no conté els mateixos mètodes. Així per exemple, no existeixen els mètodes Load i unLoad perquè els mòduls no s'han de mostrar a cap lloc com ho feien els mòduls de Kuasar. Per altra banda apareixen de nous que passem a explicar:

getEngine: Retorna una cadena amb el codi de l'hipervisor. (3 caràcters que identifiquen el tipus d'hipervisor amb el que treballa el mòdul. p.e. VBX fa referència a VirtualBox)

getPluginName: Obté el nom del plugin.

GetInterCom: Obté una classe del tipus VMCommands que serveix per a enviar els comandaments des d'el servidor al mòdul.

Com hem vist la funció més interessant està a GetInterCom ja que conté una classe que no hem vist fins ara, VMCommands. Aquesta classe és una porta d'enllaç entre el servidor Blasar i els mòduls per a enviar comandaments per a la creació i gestió dels hipervisors i les màquines. D'aquesta manera permet que qualssevol puga crear un mòdul per a què Blasar puga gestionar hipervisors. Tan sols ha d'implementar els mètodes que conté la interfície VMCommands per a què Blasar puga treballar amb ell.

Els comandaments que inclou són:

long getFreeSpace()	Retorna l'espai lliure disponible per allotjar imatges
String[] getRegisteredMachines()	Retorna les màquines registrades
String[] getRunningMachines()	Retorna les màquines que corren
String getSysProperites(String key)	Retorna una propietat de l'hipervisor
String createvm(String name, String os)	Crea una VM i retorna el seu UUID
String getImagePath()	Retorna el directori on s'ubiquen les imatges
boolean setMemory(String uuid, Long memory)	Establix la memòria de la màquina
boolean storageCtl(String uuid, String type, String controller, boolean cache)	Afegix una controladora de disc
boolean storageattach(String uuid, String storagectl, int port, int device, String type, String filepath, boolean passthrought)	Afegeix un disc virtual

boolean addNIC(String uuid, int nicID, String type, String mac)	Afegeix una targeta de xarxa
boolean setNetwork(String uuid, String operator, String username, String password, String mac, String ip, String mask, String gw, String dns)	Configura la targeta de xarxa
String GetEngineName()	Retorna el nom de l'hipervisor
String getMachineName(String uuid)	Retorna el nom de la VM
String getMachineUUID(String name)	Retorna l'UUID de la VM
boolean isRunning(String uuid)	Indica si esta corrent
boolean RunMachine(String uuid)	Inicia una VM
boolean ShutdownMachine(String uuid)	Atura una VM
boolean ResetMachine(String uuid)	Reinicia una VM
boolean PowerOffMachine(String uuid)	Atura a la força una VM
boolean deleteVM(String uuid)	Elimina una VM

Finalment cal destacar la classe SysInfo. SysInfo.java és una classe final que permet obtindre dades del sistema Operatiu tal com memòria RAM, CPU, espai de disc, etc. Aquesta classe només és un conjunt de mètodes bàsics que fa ús de la biblioteca Sigar[8] per a obtindre les dades necessàries evitant haver d'iniciar contínuament la llibreria Sigar.

3.6.2.3 El mòdul de comunicació per a VirtualBox

El mòdul virtualbox, és qui s'encarrega de transformar les ordres del servidor en ordres a l'hipervisor. En aquest cas VirtualBox. De la mateixa manera que treballen els mòduls a Kuasar és fa a Blasar. Així doncs existeix un fitxer anomenat *blasar.Services.Com.vms.PluginInterface* al paquet *META-INF.services* que conté on està la classe que ha de carregar primerament Blasar. En aquest cas es troba a *blasar.Services.Com.vms.virtualbox.Main.* La classe Main és molt més bàsica que les que s'han vist i en aquesta només cal observar que quan Blasar carrega els mòduls, la funció del mòdul, Start s'encarrega de carregar la configuració inicial (Config.load) per tal d'obtindre les dades necessàries per al seu correcte desenvolupament.

No obstant el punt més important es troba a getIntercom on s'observa com crea una nova classe que implementa a la classe VMCommands. Aquesta classe s'anomena Commands.java i realment el que fa es delegar la implementació dels mètodes a altres classes. Majoritàriament a la classe Machines. I aquesta última és la que s'encarrega d'enviar tots els comandaments per consola. Per exemple, per a crear una màquina

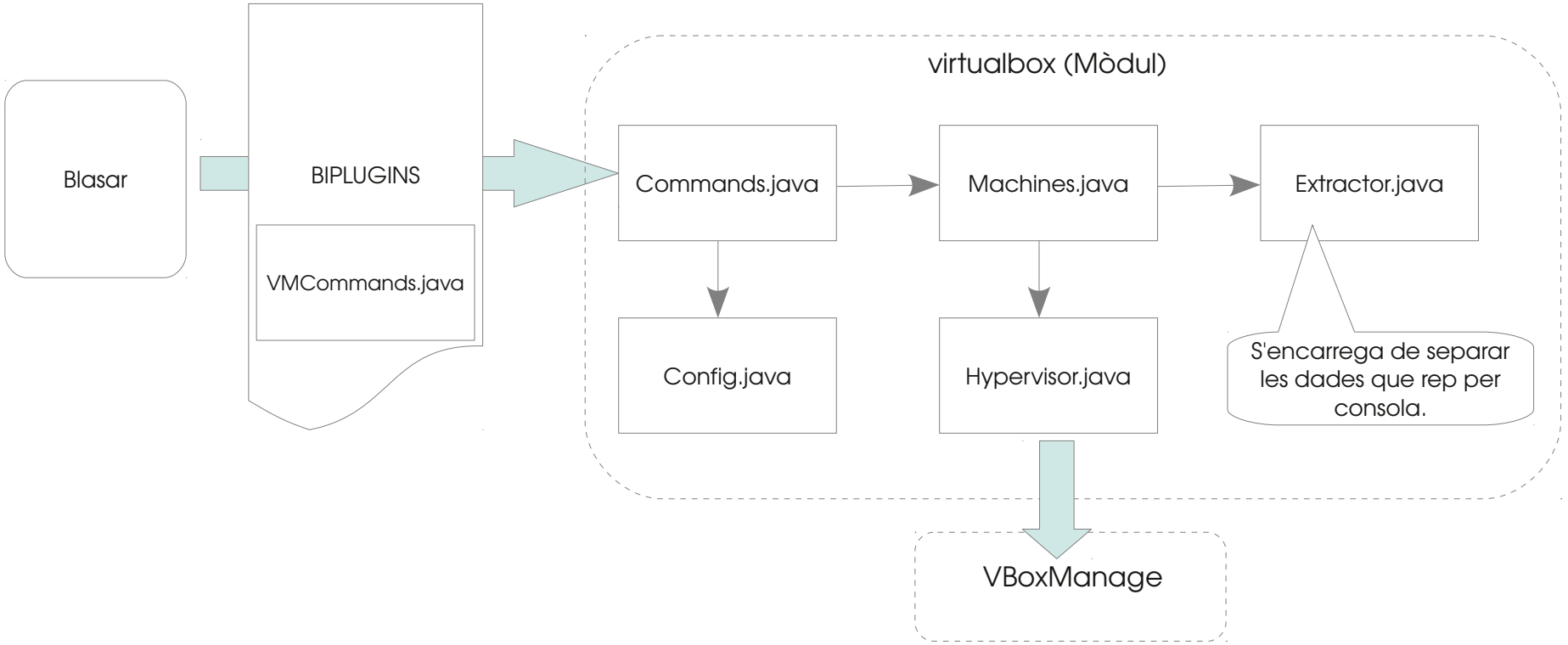
8 www.hyperic.com/products/sigar

Commands.java implementa la funció String createvm(String name, String os), la qual és delegada a la classe Machine fent ús del codi Machines.createMachine(os, name).

Finalment a la classe Machine la funció createMachine és desenvolupada de manera més extensa, comprovant si el nom de màquina ja existeix checkName(preferredName), traduint el nom del sistema operatiu que se li passa per a ser entès per l'hipervisor getOS(os) i finalment executant l'ordre per crear la màquina. Hypervisor.getStream("createvm --name \""+ preferredName + "\" --ostype " + ostype + " --register");[9]

9 Hipervisor és una classe que s'encarrega d'executar el comandament VboxManage i passar-li els arguments de manera correcta.

SIMULACIÓ DE PAS DE DADES PER BLASAR



Capítol 4.
Un viatge pel projecte

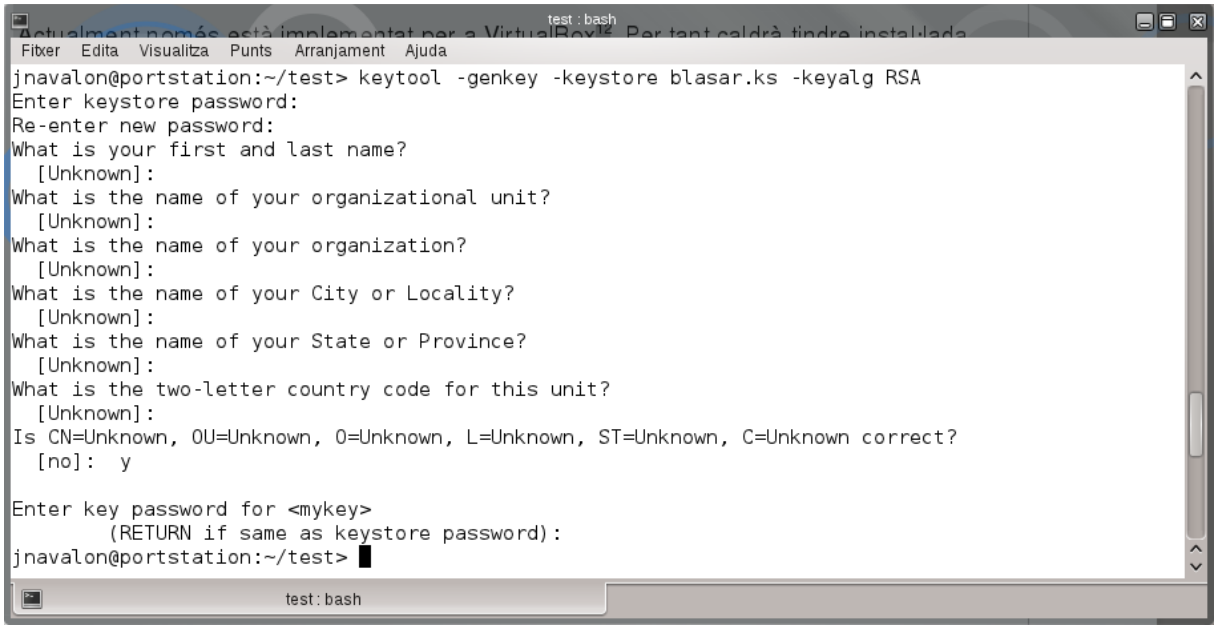
En aquest capítol anem a mostrar una petita demostració de com un usuari pot crear la seva pròpia xarxa mitjançant les aplicacions Blasar i Kuasar. Pretén ser una guia des de zero, que comence des de la configuració del servidor, fins a poder instal·lar Màquines Virtuals als hipervisors remots.

4.1. Requisits

Per a què el projecte funcione, només cal complir dos requisits. El primer tenir instal·lat el JRE[10] [11] de Java tant al sistema on s'executarà el client (Kuasar) com en el servidor(Blasar). I el segon requisit és tindre instal·lat un hipervisor, desenvolupat. Actualment només està implementat per a VirtualBox[12]. Per tant caldrà tindre instal·lada aquesta eina.

Per a xifrar la comunicació entre el client i el servidor ens caldrà generar un keystore. Açò s'aconseguix executant l'eina de Java 'keytool' i seguint les instruccions.

keytool -genkey -keystore nom_del_keystore -keyalg RSA



Aquesta ordre crearà un fitxer al directori on s'haja executat. Deseu aquest fitxer junt al directori blasar.

4.2. Preparació de la Imatge de la Màquina Virtual.

El primer pas després de la instal·lació dels requisits es crear imatges de disc dels sistemes operatius que volem implementar en altres màquines. Aquestes imatges són els discos que genera el VirtualBox quan es crea una màquina virtual. Encara que podeu descarregar també sistemes preinstal·lats des d'Internet[13].

10 java.com/es/download/
11 Es recomana evitar la instal·lació del paquets lliures del JRE de java als sistemes Linux perquè han hagut casos d'alentiment dels efectes gràfics.
12 www.virtualbox.org/wiki/Downloads
13 virtualboxes.org/images/

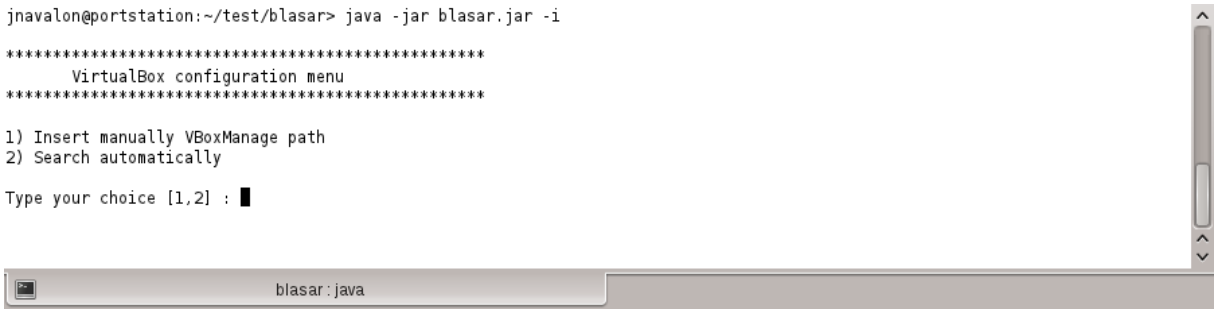
Per tal de configurar la xarxa caldrà guardar dins del sistema de la imatge el programa operator. Actualment només està disponible per a Sistemes Debian / Ubuntu. Encara que es pot modificar per a ser utilitzat en la resta de Sistemes Unix si es coneix quin es el fitxer de configuració de xarxa del sistema.

Així per exemple hem descarregat una imatge d'un Ubuntu Server i l'hem instal·lat un servidor Web i hem desat el fitxer operator.sh al directori /blasar.

4.3. Instal·lació i configuració inicial de Blasar

Si no disposeu del projecte podeu fer-vos amb ell descarregant-vos-ho des de la pàgina del projecte[14].

- Descomprimim el directori blasar i el desem a un directori del nostre sistema.
- Obrim una consola per tal d'iniciar el servidor blasar
- Accedim a la consola i executem l'aplicació com és la primera vegada que l'iniciem haurem de crear el primer usuari. Per tant ens caldrà activar la interactivitat de blasar amb l'argument '-i' per a què ens done una consola de configuració. A més també indicarem el keystore i la seva contrasenya. Aquestes dades poden ser configurades en el fitxer de configuració de Blasar per no tindre que tornar a afegir-les com arguments. -k <fitxer keystore> -w <contrasenya>
- Executarem doncs: java -jar -i -k blasar.keystore -w blasar

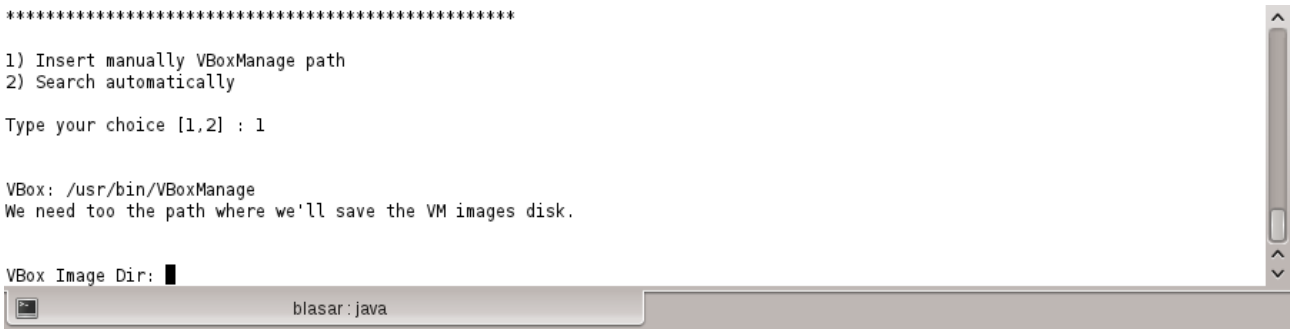


Si observem al carregar el mòdul virtualbox detecta que no te res configurat i us demana algunes dades necessàries per al seu funcionament mitjançant un senzill menú.

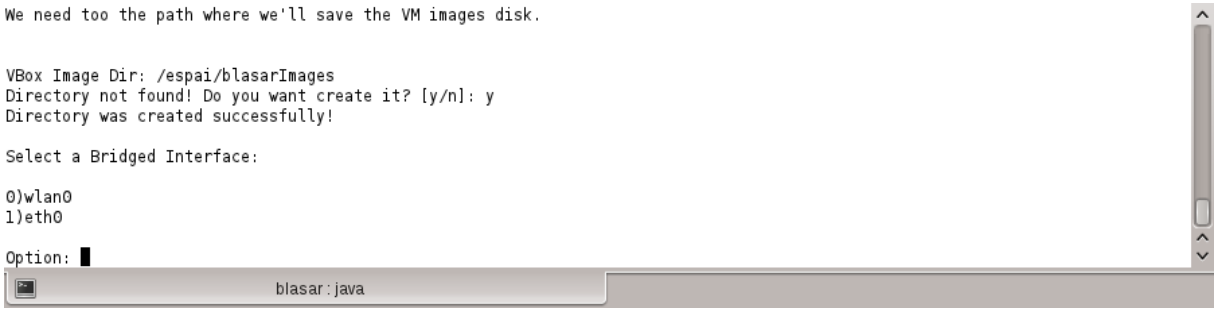
La primera dada que ens demana és on pot trobar l'aplicació VboxManage i ens ofereix dos possibilitats. Podem donar-li-ho de manera manual o el pot cercar al nostre

14 code.google.com/p/the-kuasar-project/

sistema. Com coneguem on està ubicat el nostre VboxManage seleccionarem l'opció 1 per a afegir-ho manualment.



La segona dada que li cal al mòdul és especificar-li el directori on es vol desar les imatges que li envie els clients. Si el directori que hem afegit no existeix, blasar intentarà crear-lo per nosaltres.

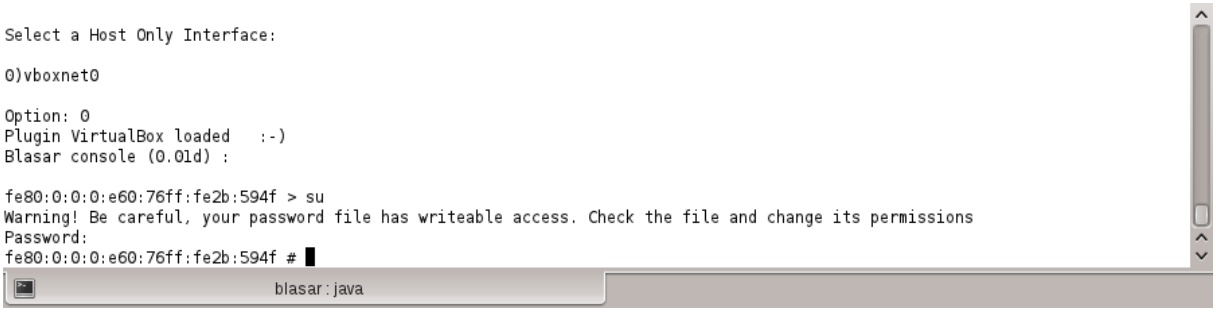


El següent pas és seleccionar la interfície de xarxa del sistema per a què faça de pont amb la targeta de la màquina virtual, en cas de seleccionar una connexió de tipus pont (Bridge). En el nostre cas seleccionarem wlan0.

Finalment ens demanarà seleccionar la interfície de xarxa privada (Host Only) si no existeix cap, us preguntarà si voleu que ho faça per vosaltres. Nosaltres seleccionarem l'única disponible 'vboxnet0'.

Quan tots els requisits han estat coberts Blasar continuarà amb la càrrega de mòduls i iniciarà el servidor i obrirà la consola.

El que ens queda, finalment, és crear un usuari. Per tant escriurem el comandament 'su' per a accedir al nivell administrador. La contrasenya per defecte és *blasar*.



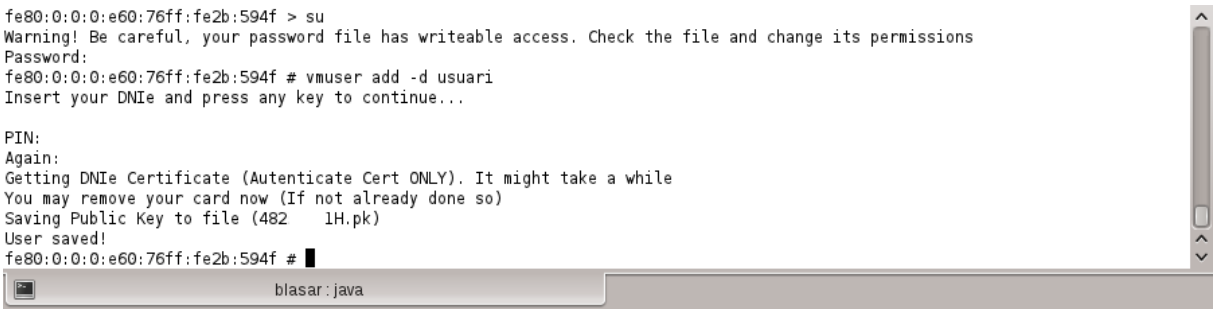
Si el fitxer te permisos d'escriptura se us mostrarà un missatge d'alerta. Però us deixarà accedir d'igual manera.

El següent pas es crear el nostre usuari per a què el client puga accedir al servidor de manera remota. Per a això farem us del comandament 'vmuser add'. En el nostre cas farem ús del DNIe per a autenticar-nos al sistema, per tant afegirem l'opció -d.

Finalment el comandament quedarà de la següent manera:

vmuser add -d usuari

i seguirem les instruccions, en el nostre cas escriure el PIN.



Amb açò Blasar ja estaria preparat per a funcionar i poder instal·lar màquines a l'hipervisor.

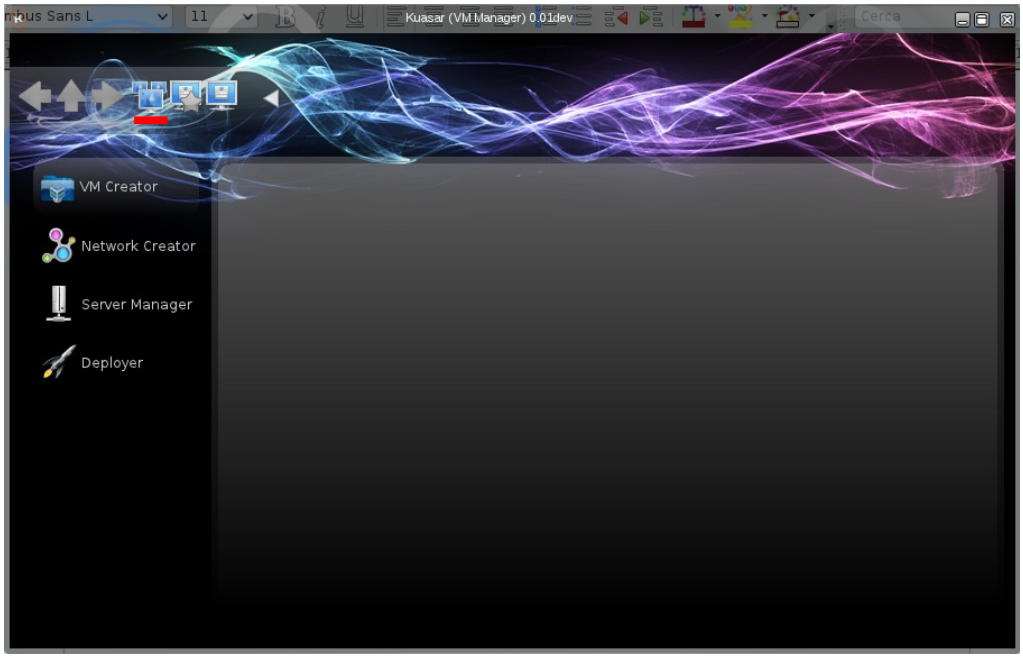
4.4. Instal·lació i Configuració de Kuasar

Kuasar, no requereix de cap configuració per al seu funcionament. Per tal cosa per a posar en marxa l'aplicació només caldrà obrir l'aplicació Java 'kuasar.jar'. També pot ser executada per comandament mitjançant l'ordre 'java -jar kuasar.jar'

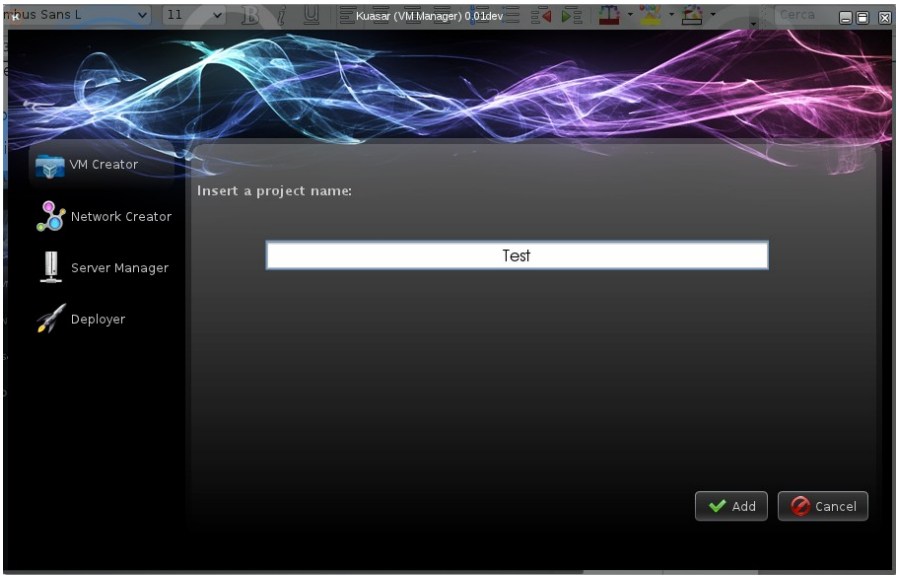
4.5. Crear un projecte

Crear un projecte és una tasca senzilla.

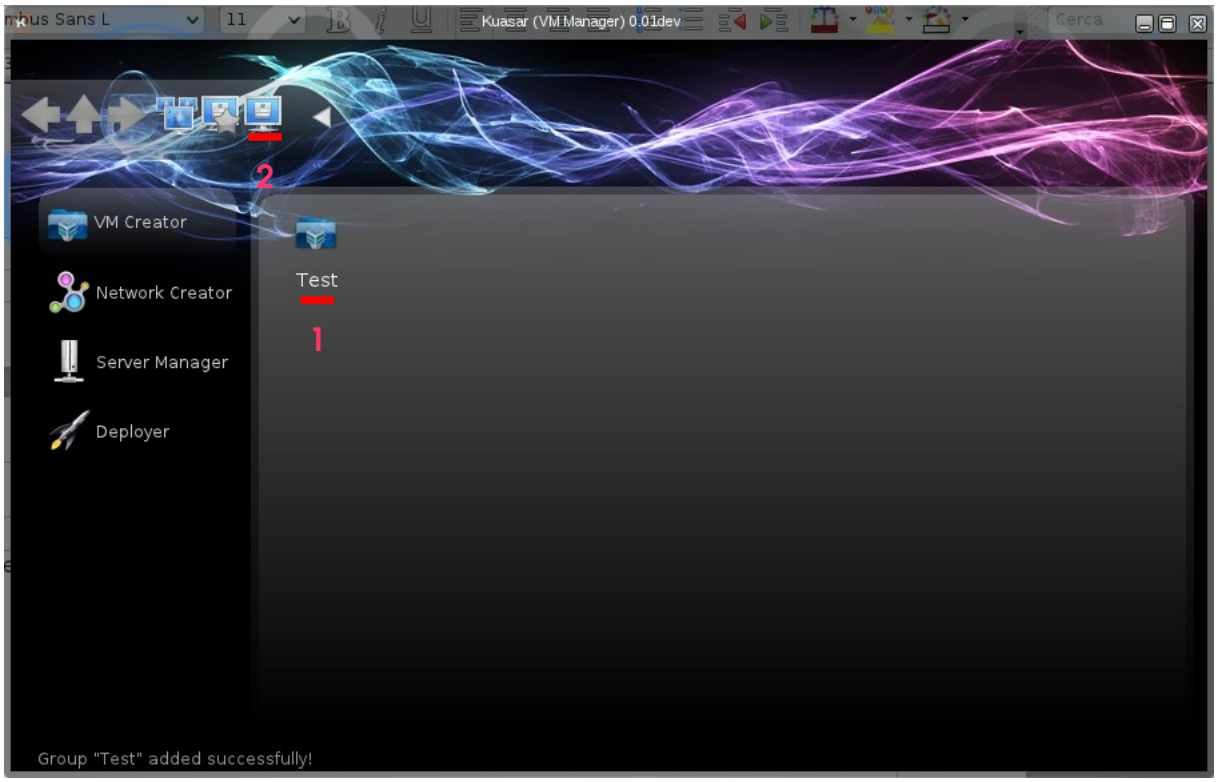
Obrim Kuasar i seleccionem el mòdul VMCreator. Fem clic a la fletxa per a desplegar la barra d'eines i després a la icona de crear projecte.



En la nova finestra introduïm el nom del projecte que volem crear i cliquem a afegir.

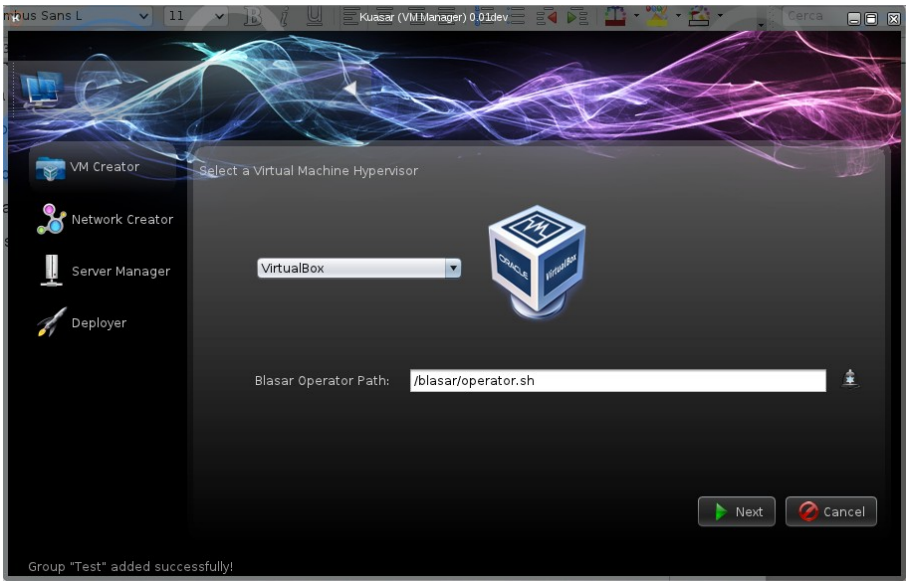


Premem a la nova carpeta que ha aparegut per tal d'accedir dins del projecte. I Premem a la icona per afegir una nova màquina

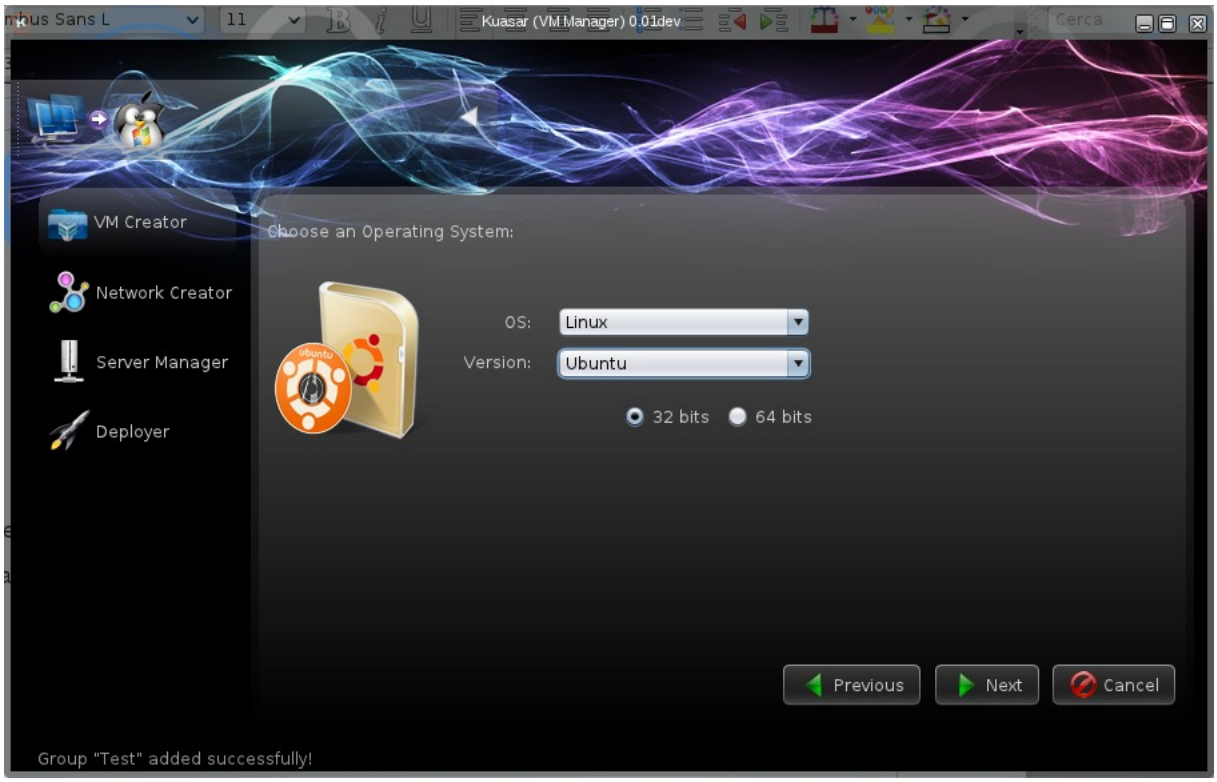


A la nova pantalla seleccionem l'hipervisor. En aquest cas seleccionarem VirtualBox, ja que és l'única que hi ha implementat. I a Blasar Operator Path afegirem l'adreça on hem desat el programa operator. En el nostre cas /blasar/operator.sh

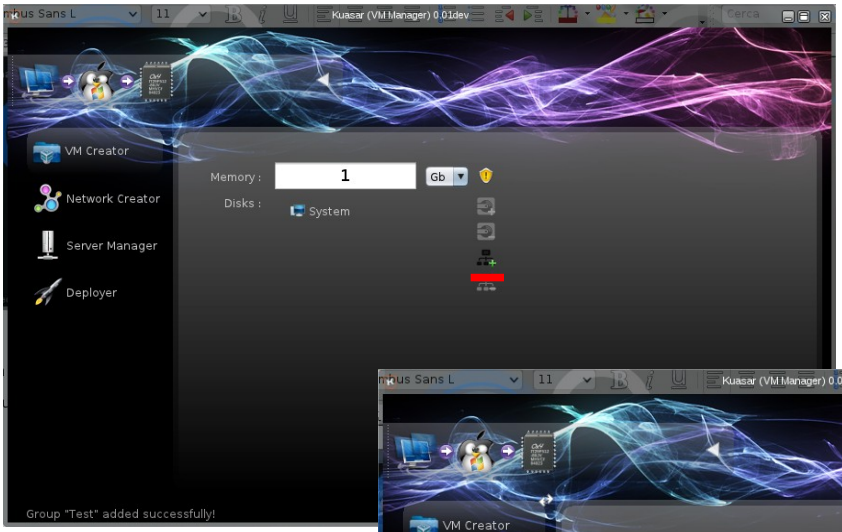
A continuació farem clic a la icona de barret per a afegir el nom d'usuari i contrasenya del sistema on està operator per a què puga arrencar-se amb els privilegis de l'usuari. L'usuari en l'entorn Linux ha de tindre privilegis d'administrador.



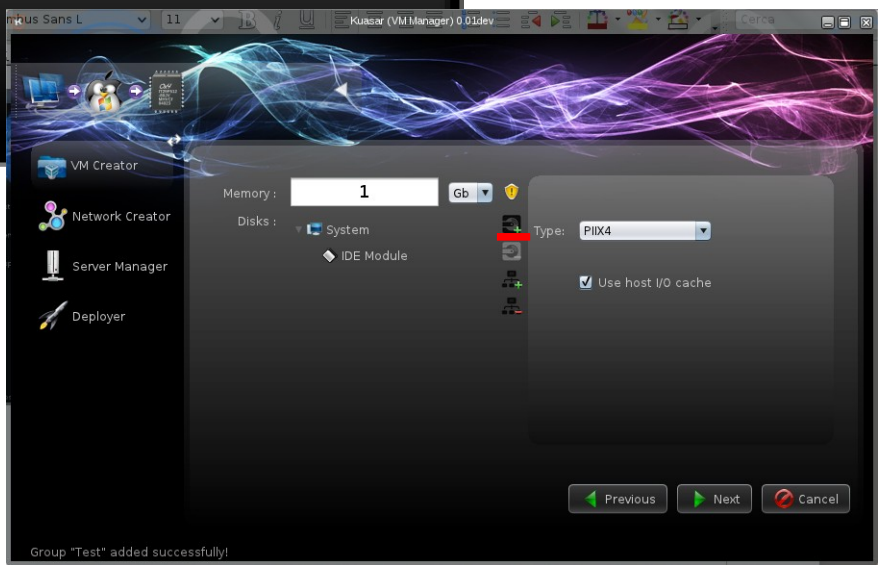
Premem en el botó Next per a seleccionar el tipus de sistema operatiu i la seva arquitectura. Seguim, prement de nou el botó Next.



En la nova pantalla haurem d'indicar la memòria RAM que tindrà la màquina i afegir la imatge que tenim del nostre sistema.

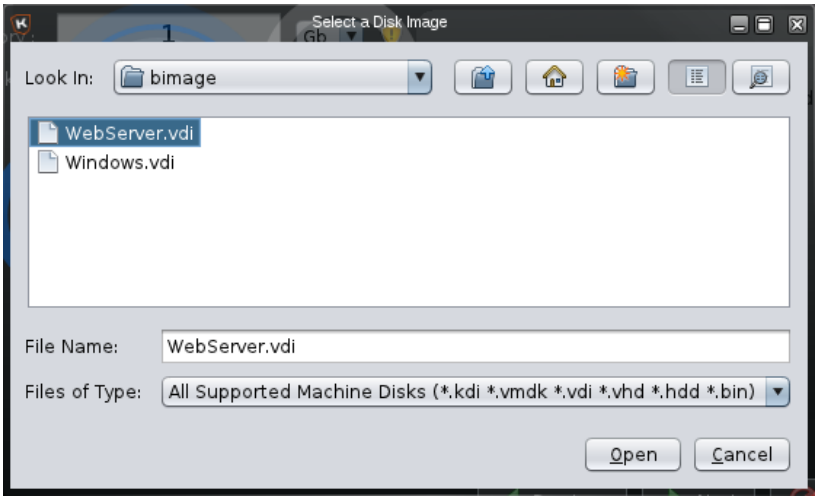


Fem clic per a afegir un controlador, en el nostre cas, seleccionarem un controlador IDE.

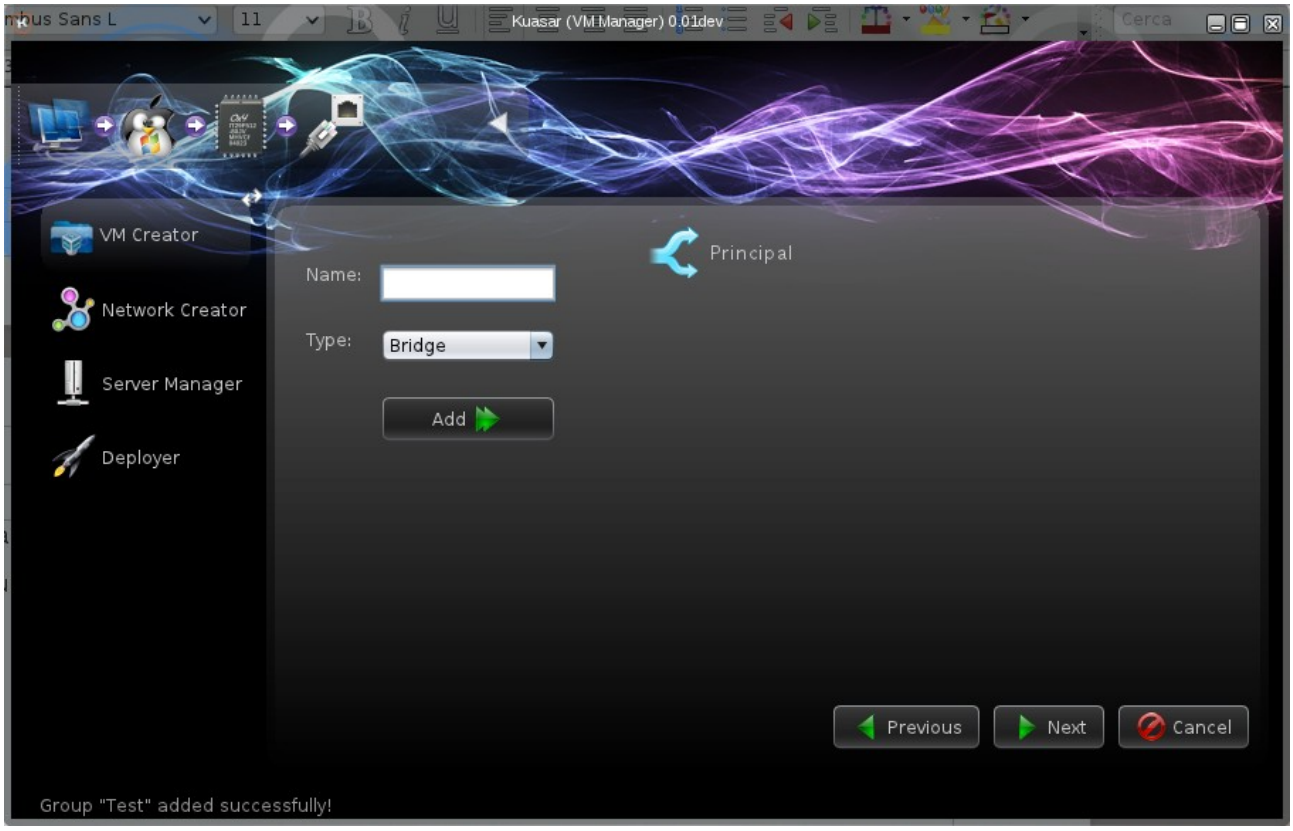


Seleccionem la nova icona que ha aparegut i premem a la icona de disc per a afegir un nou disc dur on indicarem la imatge del nostre sistema.

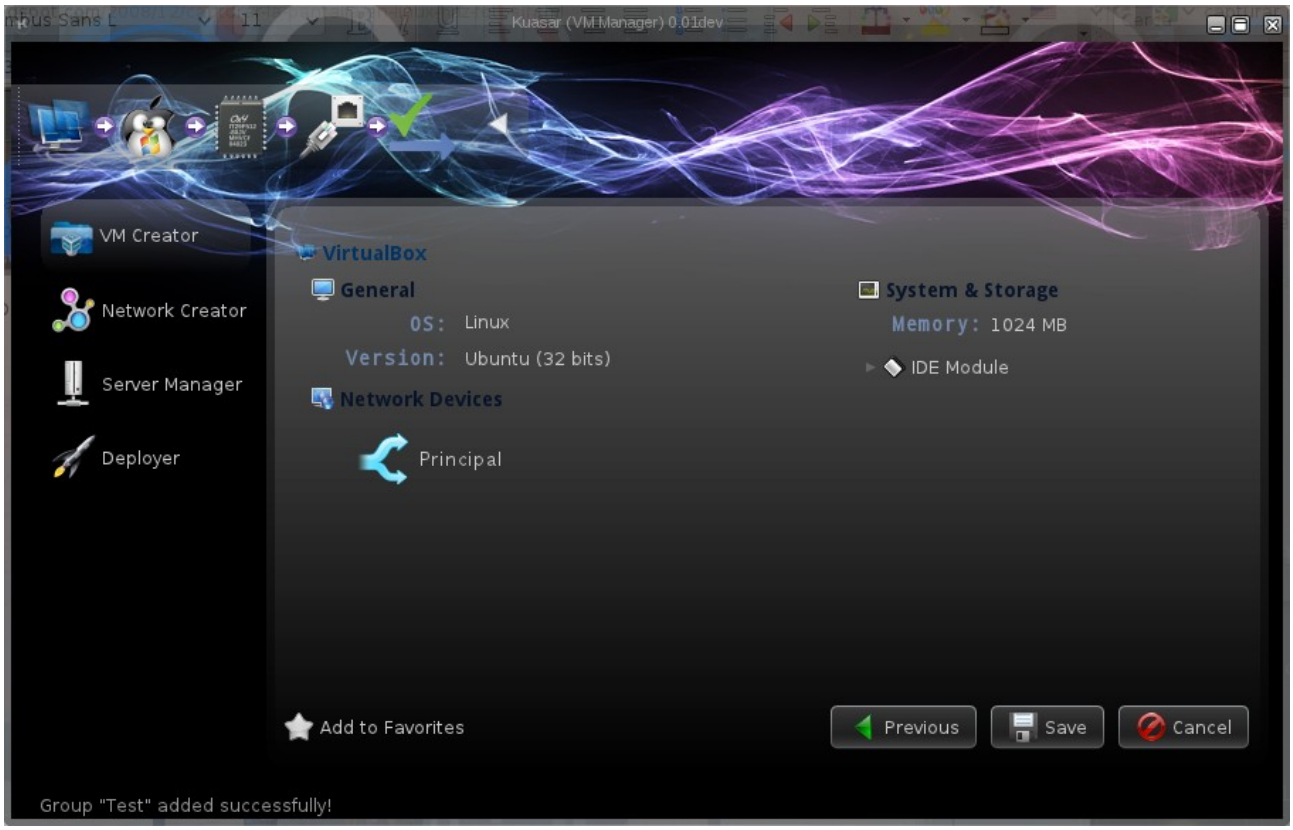
Finalment, premem a Next.



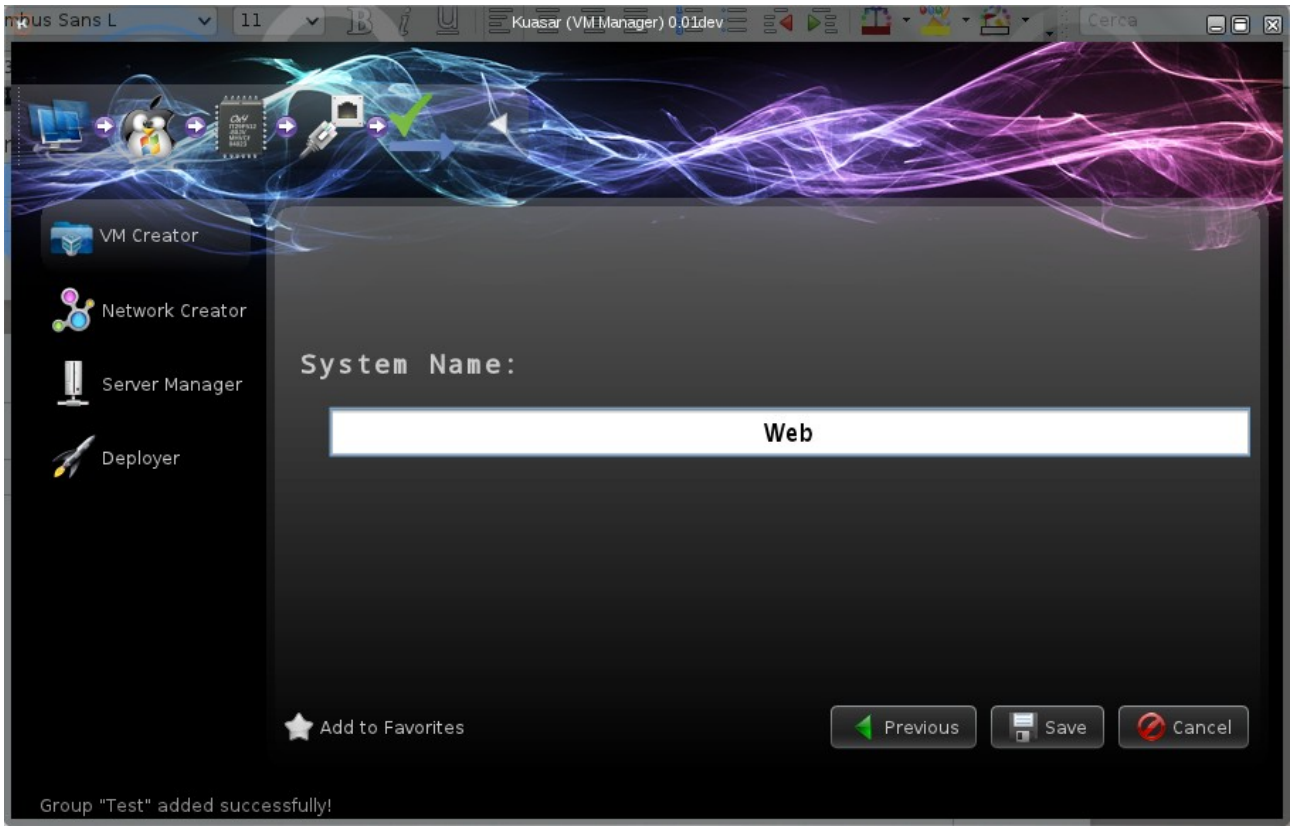
La següent pantalla ens permetrà afegir les targetes que tindrà el nostre sistema. En el nostre cas només crearem una targeta de tipus Bridge.



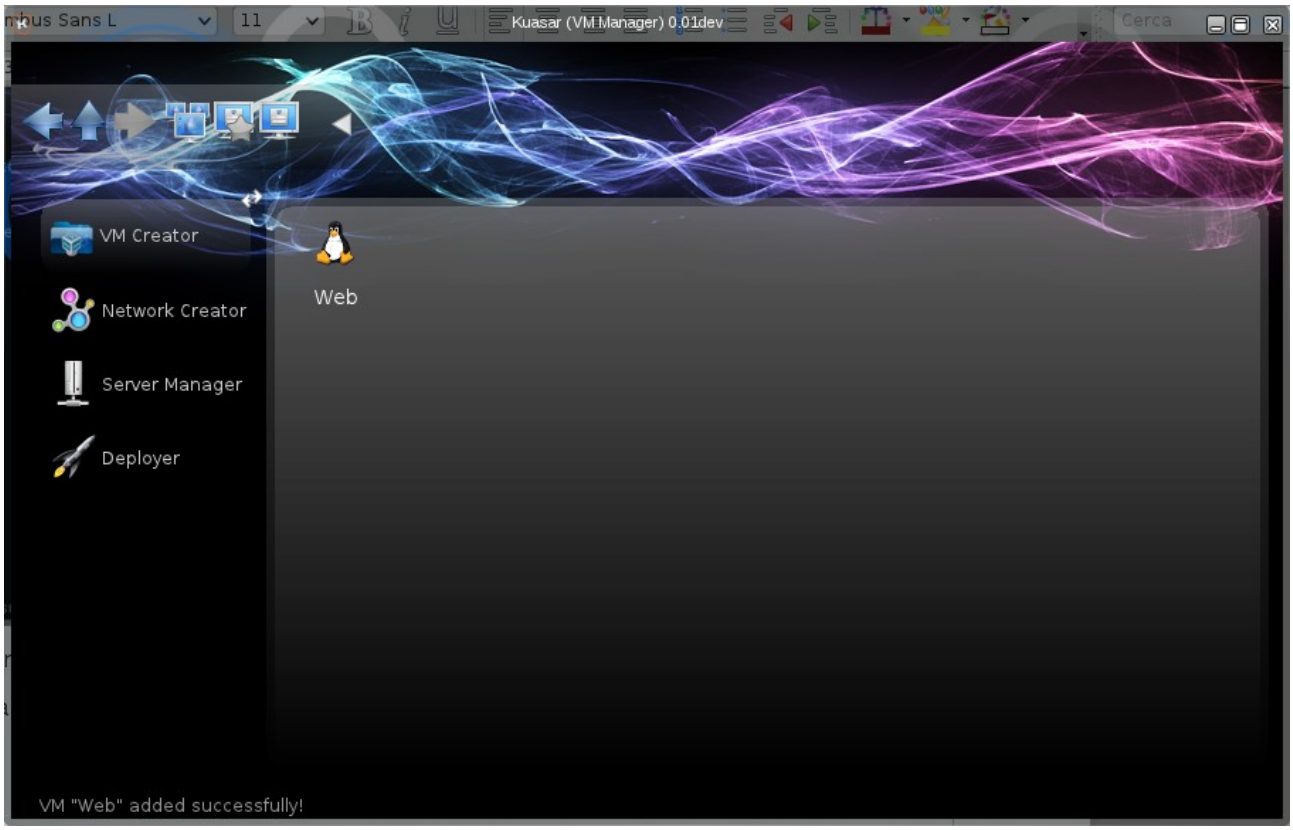
Premem next i comprovem si totes les dades són correctes



Finalment premem el botó save i desem les dades amb el nom que vulguem.



Quan acabeu d'introduir el nom premeu enter per a acceptar el nom i desar la màquina. Immediatament arribareu a la pantalla principal on observareu que la vostra màquina ha estat afegida correctament.



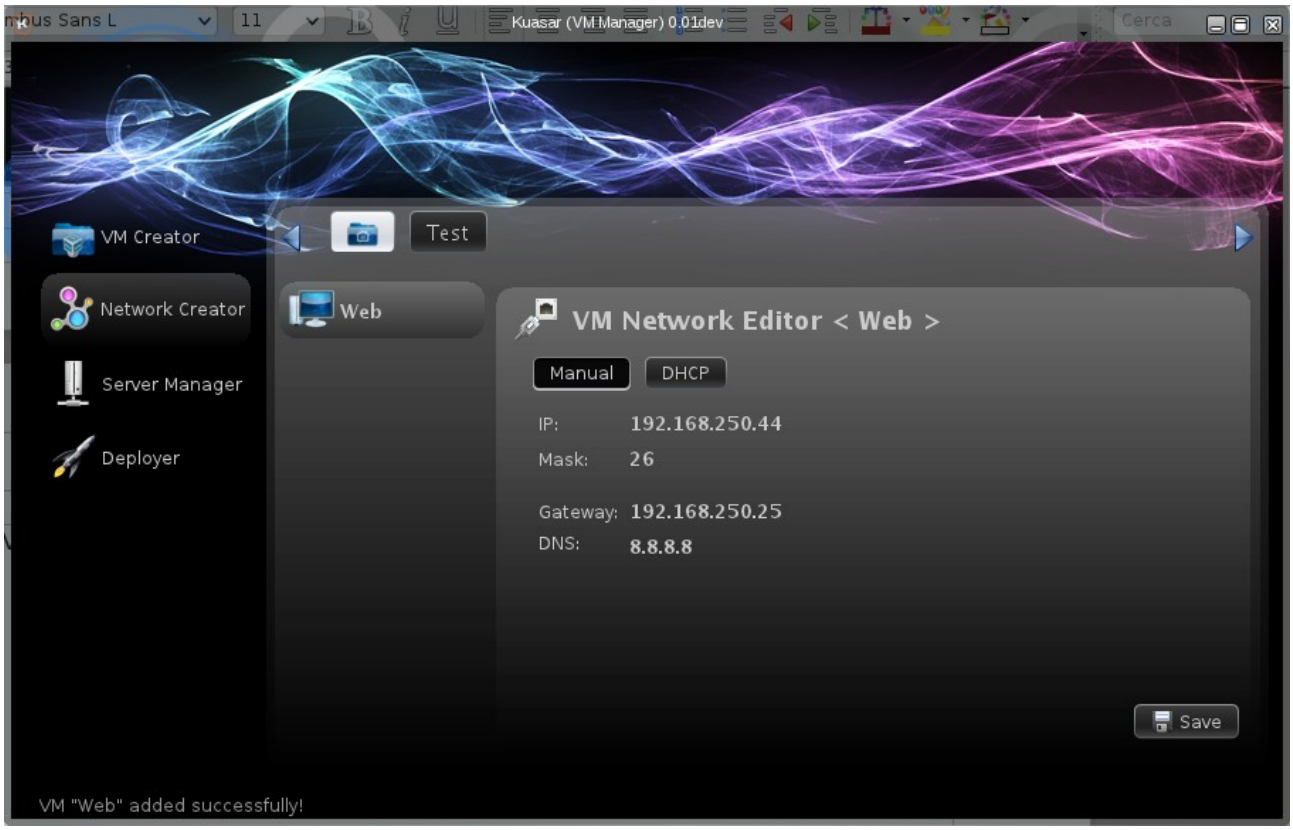
4.6. Establir la configuració de xarxa a la nostra màquina

Per a donar una IP al nostre sistema farem clic al mòdul netCreator per a carregar-lo a la finestra de mòduls.

En aquesta ocasió com només serà una màquina la que hem d'assignar una adreça. No utilitzarem l'eina AutoIP per a què ens l'assigne automàticament.

Navegem per la llista de la dreta fins seleccionar la nostra màquina.

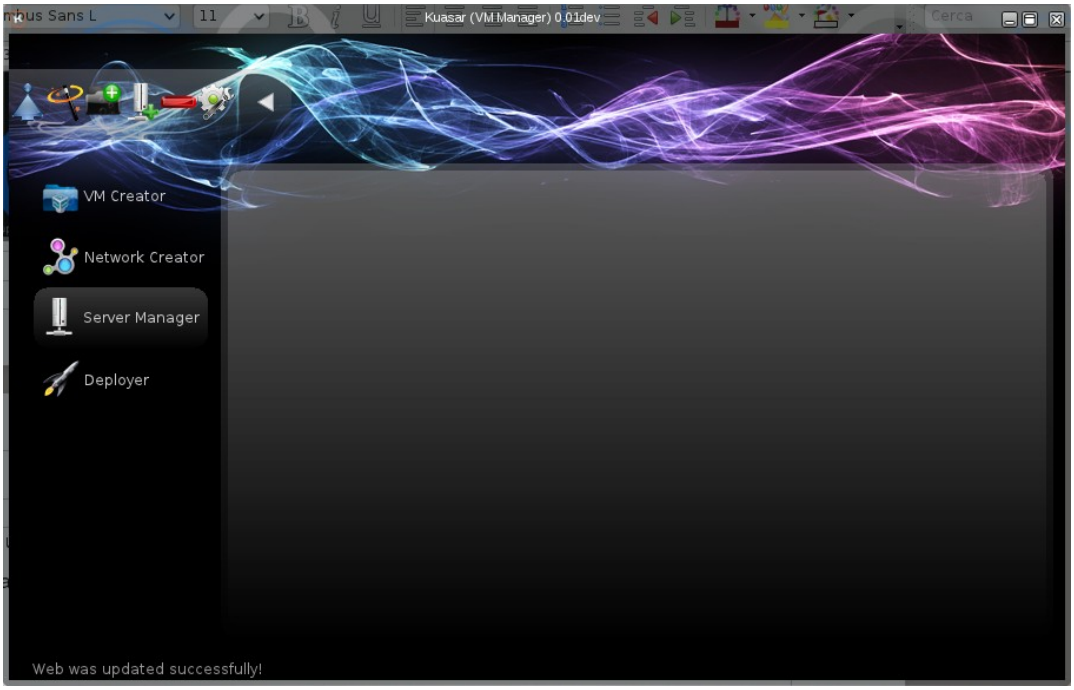
Premem al botó Manual per a desactivar l'autoassignació per dhcp i afegim les dades de connexió.



Fem clic en Save per a desar les dades.

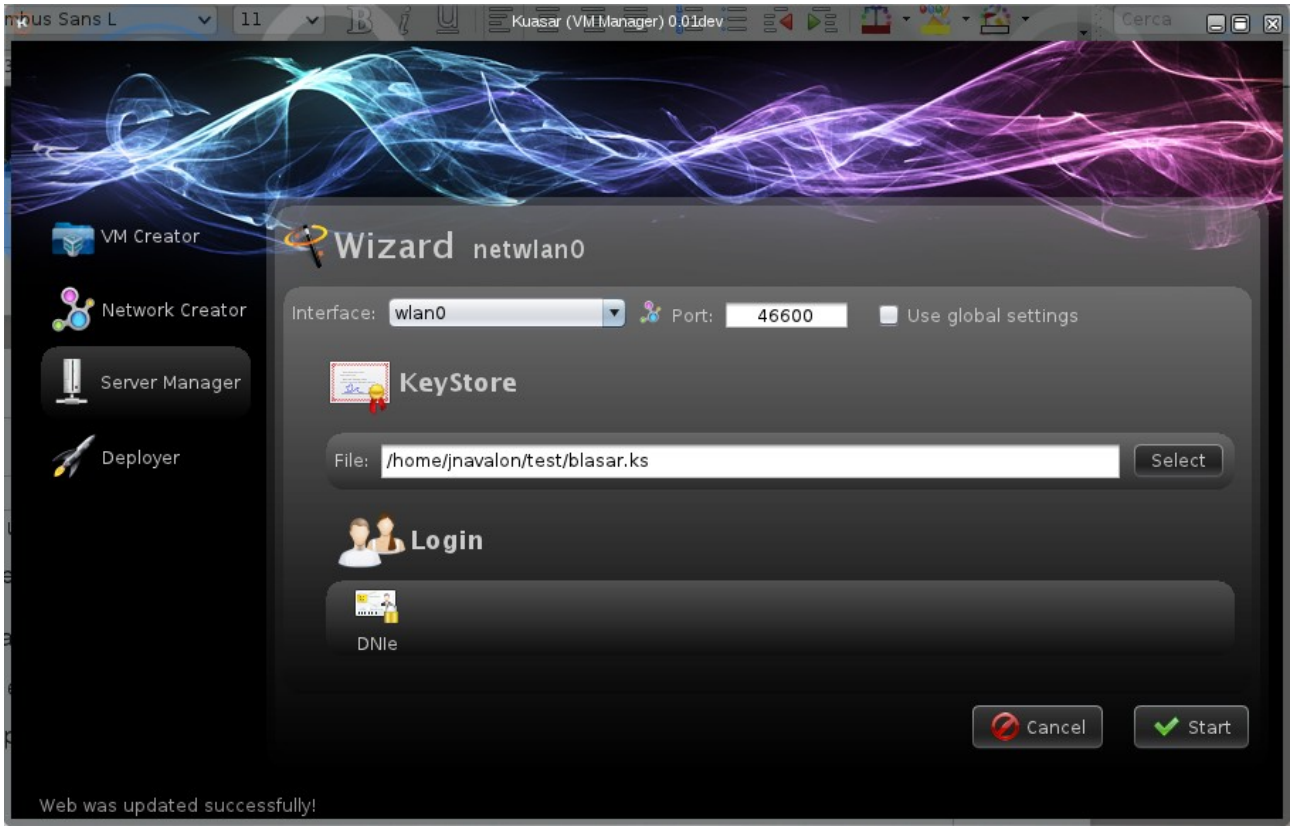
4.7. Afegir servidors Blasar

Per a gestionar els nostres servidors seleccionarem el mòdul Server Manager.

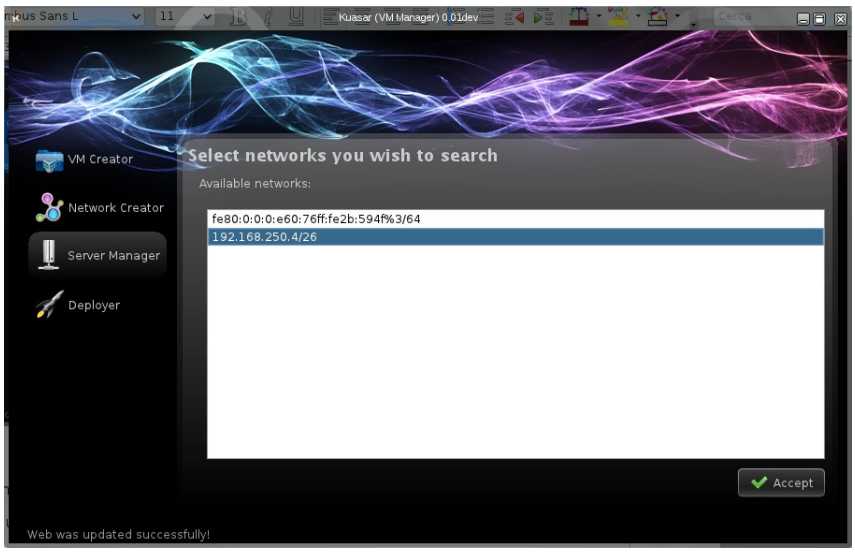


Com no sabem quina IP te el servidor farem un escaneig de la xarxa fent clic a la vareta màgica. Si no teniu desplegat la barra de ferramentes feu clic a la fletxa gris per a desplegar-la.

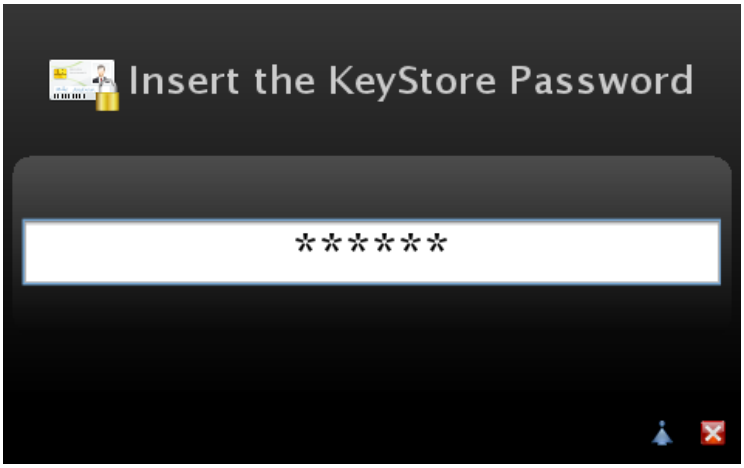
A la nova pantalla seleccionarem la interfície de xarxa on volem cercar el servidor. Seleccionarem el port (per defecte és el 46600) i carregarem el keystore fen clic al botó 'select'. Fent clic en Login activarem la comprovació de l'usuari i seleccionarem l'autenticació per DNIe.



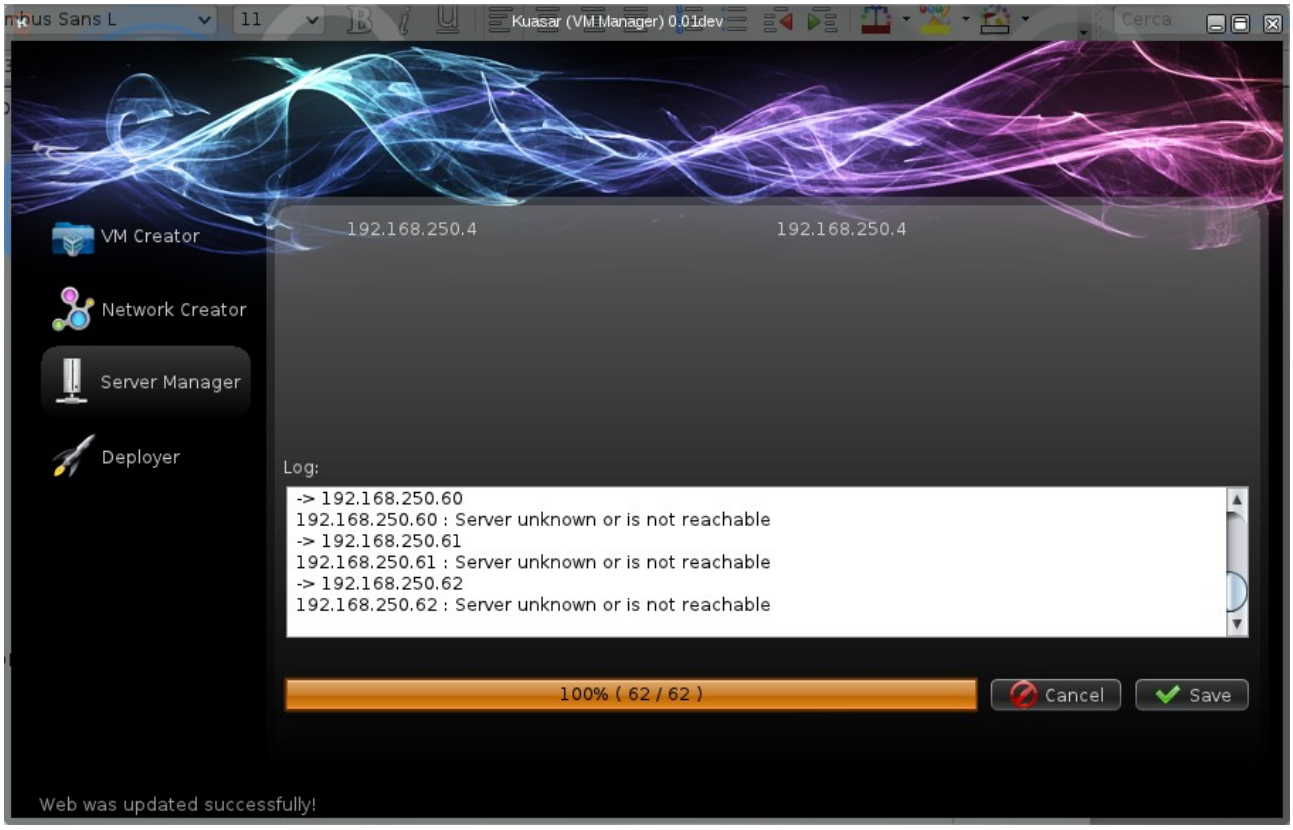
Si tenim més d'una IP assignada farem clic a la icona al costat de la targeta de xarxa per a seleccionar per quin rang volem cercar.



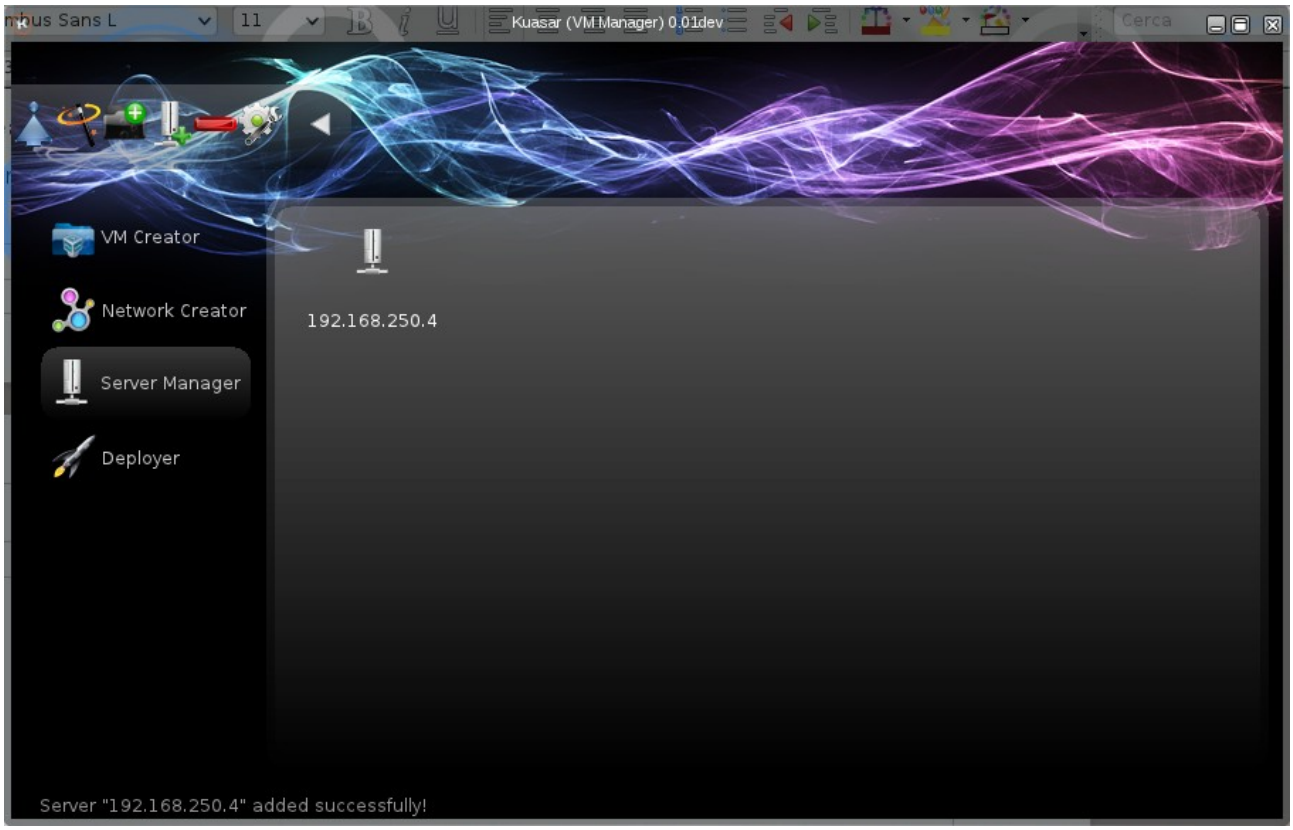
Fem clic a Start per a iniciar la recerca i introduïm la contrasenya del keystore.



El programa automàticament cercarà les màquines i mostrarà a la llista aquelles que continguen un servei Blasar i hagen pogut autenticar-se correctament. Una volta trobat el servidor desarem la llista, fent clic a save.



Al prémer el botó save, desarà els servidors trobats i tornarà a la pantalla principal on veurem que el projecte ha creat un directori amb els servidors dins. El nom del directori pot ser canviat quan s'està seleccionant la targeta de xarxa, a la part superior junt al títol.

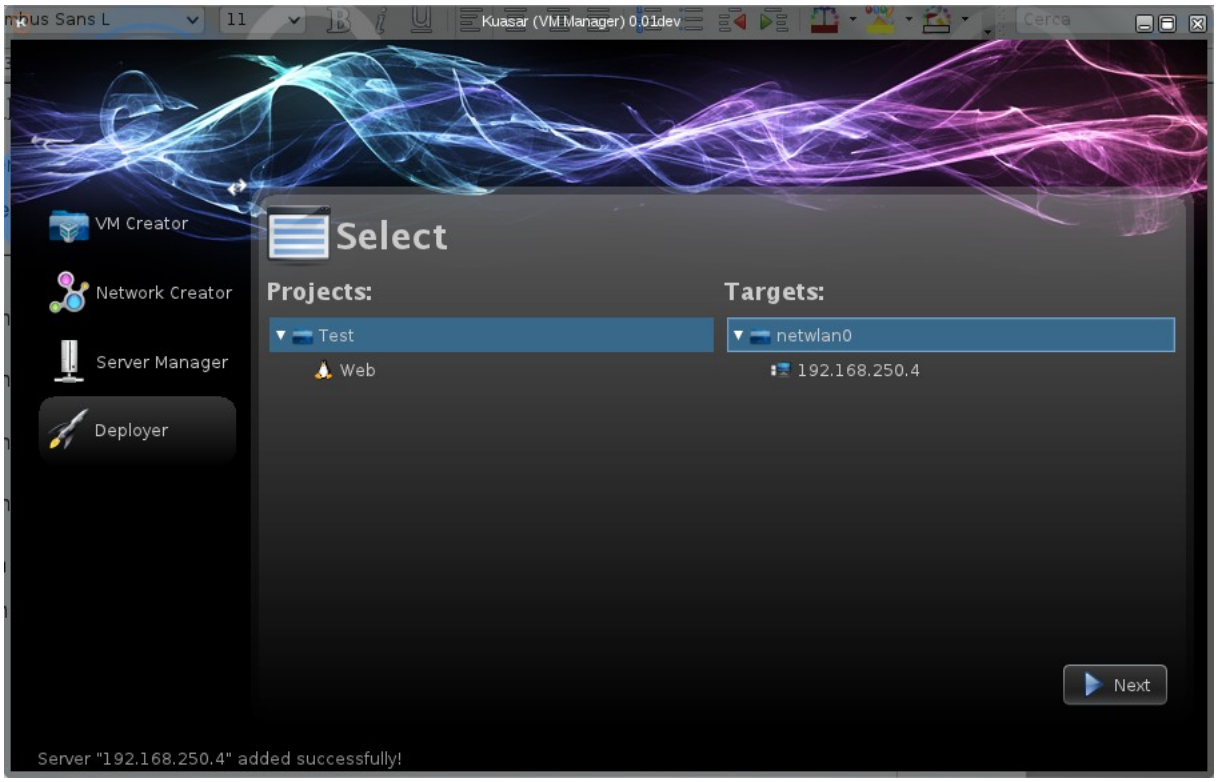


4.8. Instal·lació del projecte a l'hipervisor.

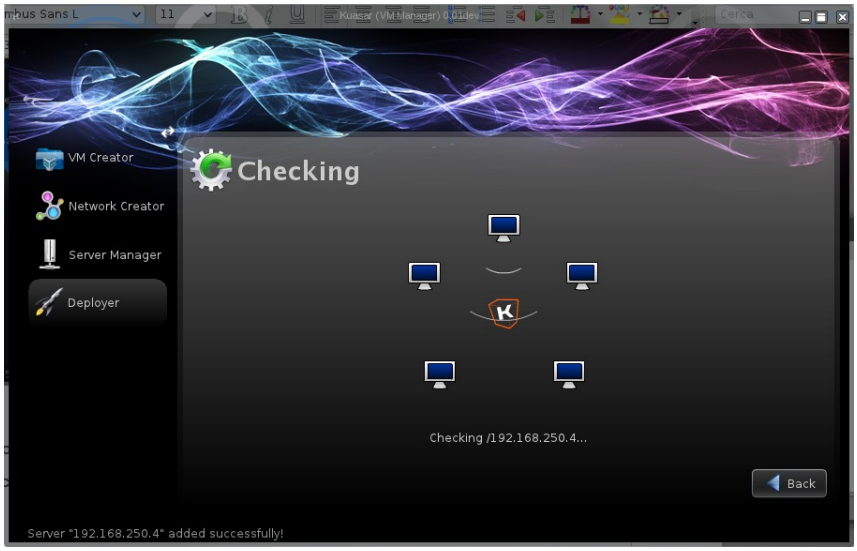
Per a instal·lar definitivament la nostra màquina seleccionarem el mòdul Deployer on apareixeran dos columnes. A l'esquerra els nostres projectes i a la dreta els nostres servidors. Haurem de seleccionar un objecte de cada columna. Les possibilitats són:

- Un projecte en un conjunt de servidors
- Una màquina en un conjunt de servidors
- Un projecte en un Servidor
- Una màquina en un Servidor

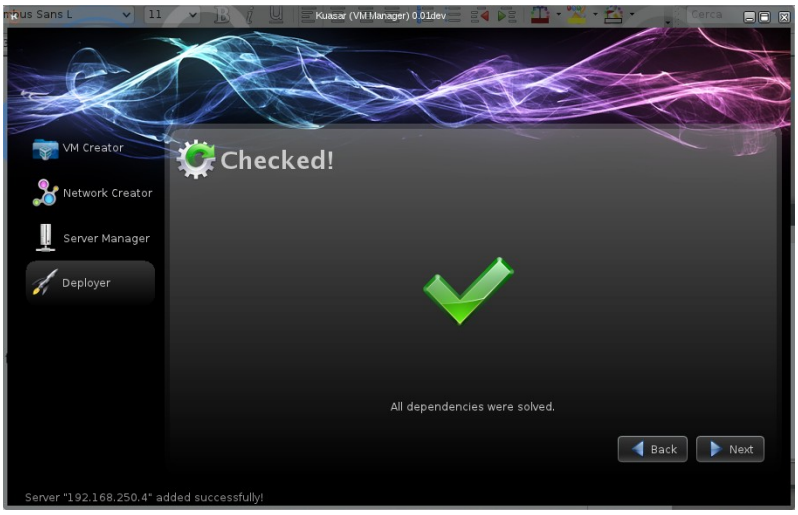
En el nostre cas podem seleccionar qualssevol opció però seleccionarem la primera que és la més habitual. Després premem el botó Next.



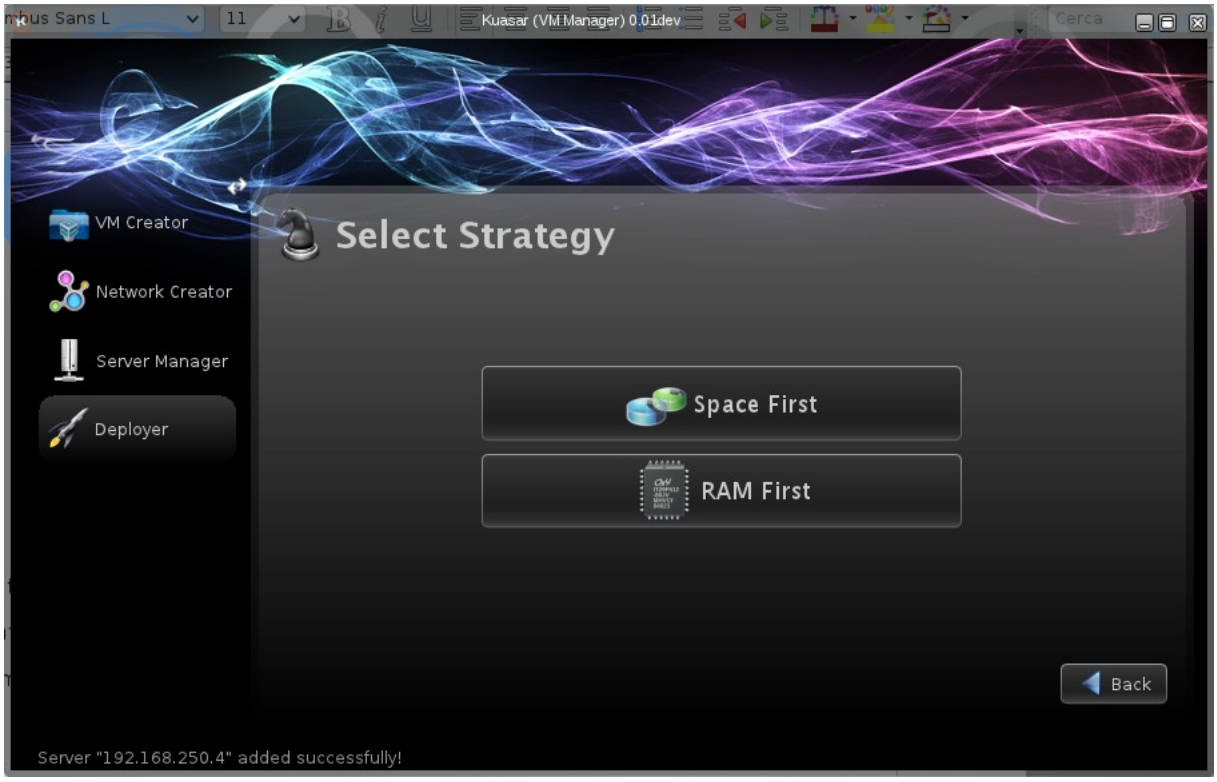
A continuació comprovarà les dades i els servidors i demanarà algunes dades d'accés opcionals.



Si tot ha anat correctament apareixerà una finestra d'informació.



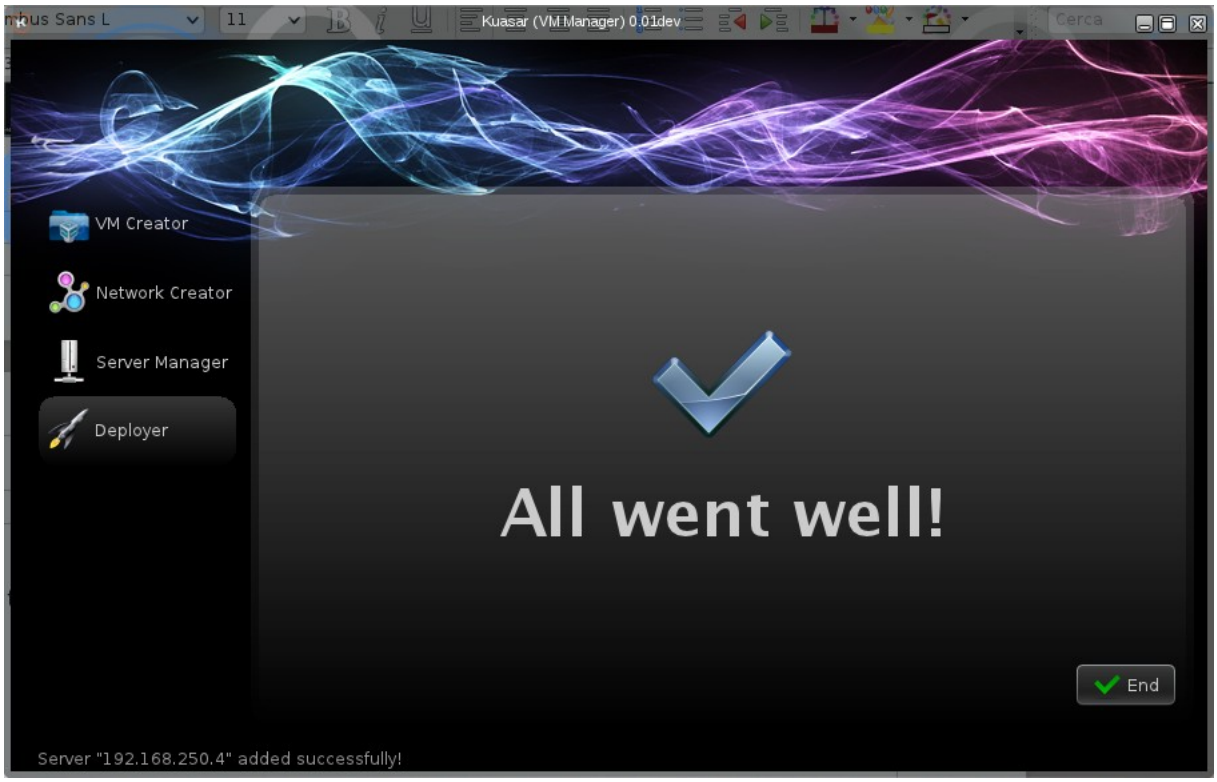
Si tot ha anat bé premem a Next i seleccionem la nostra estratègia. Com s'ha d'implementar només en un servidor ens dona igual quina va a ser l'estratègia. Així doncs seleccionem una qualssevol.



Kuasar intentarà comprovar la memòria del servidor per saber si pot ser allotjada la nostra màquina, en cas afirmatiu, apareixerà una llista amb les màquines i a quin servidor seran allotjades.

Premem el botó Apply per confirmar i començar a crear la màquina. Durant la creació de la màquina aniran apareixent diverses pantalles, informant de l'estat en el que es troba la instal·lació de la màquina.

Si tot ha anat bé apareixerà la finestra de confirmació.



Fem clic a End per a acabar.

Capítol 5.
Conclusions i futurs treballs.

Després de l'explicació donat a terme als capítols anteriors, es parlarà de les opcions que presenta el projecte realitzat i el possible enfocament que podria tindre els següents treballs basats en l'aplicació desenvolupada. A més a més, s'exposaran les conclusions pròpies a les que s'ha arribat després de tot aquest procés.

5.1. Treballs futurs

Com hem dit durant els primers capítols d'aquesta memòria, els hipervisors i les màquines virtuals estan convertint-se en una elecció per a moltes empresses que volen estalviar recursos i administrar de manera més eficaç les seves xarxes i servidors. Gràcies a que les aplicacions són modulars és molt senzill aplicar noves eines que permetin cobrir les necessitats de les empreses. A continuació llistem algunes idees.

- **Compatibilitat per a més Hipervisors.**

Malgrat que les bases del projecte ja estan implementades encara falta molt per a ser una eina suficientment versàtil per a treballar a qualssevol entorn. Per aquesta raó seria necessari implementar nous mòduls per a blasar que permeteren gestionar altres tipus d'hipervisors sobretot els més populars com VMWare, KVM o Xen.

- **Estadístiques de les màquines Virtuals.**

Fins ara només hem aconseguit rebre algunes dades del rendiment dels servidors on s'allotgen els hipervisors. Però seria una gran idea, crear un mòdul o ampliar algun existent per a mostrar en temps real l'estat dels discos, de la memòria, i de l'us de la CPU.

- **Implementar un sistema d'escriptori remot.**

Un altra idea per a implementar, és una eina que permetés accedir des del client Kuasar a les diferents màquines virtuals mitjançant protocols RDP, VNC o tecnologia NX. VirtualBox permet iniciar la màquina en mode RDP per a què es puga accedir a la pantalla des d'un sistema remot. Seria una bona idea explotar aquesta utilitat, ja que no cal instal·lar cap eina addicional al sistema.

- **Millorar la gestió de contrasenyes**

Tot i que per seguretat no s'ha implementat cap mètode per a desar les contrasenyes dels servidors. Seria beneficiós per al projecte un sistema de clau Mestra que permetés xifrar-les. D'aquesta manera no seria necessari introduir-les cada volta que s'inicia el client.

5.2 Conclusions

Després de concloure de manera definitiva el desenvolupament del present projecte final de carrera, és el moment de fer balança amb el resultat final repassant la trajectòria recorreguda per a obtenir conclusions, tant en l'aspecte tècnic i professional com a l'àmbit personal a més de les habilitats adquirides durant aquest temps en diverses tecnologies i eines.

Al voltant de tot el projecte s'ha aconseguit obtindre un millor coneixement de la plataforma Java, el seu funcionament intern, la forma de gestionar els recursos del sistema, a més d'aprendre a desenvolupar aplicacions per a aquesta plataforma. Tot açò ha estat possible gràcies a la immensa documentació que Oracle i la resta de desenvolupador mitjançant dels diversos fòrums posen a disposició dels desenvolupadors malgrat que cal reconèixer la laboriosa tasca d'investigació que s'ha fet pel que fa a això.

Per altra banda, després d'haver cursat la titulació d'Enginyeria Tècnica en Informàtica de Gestió i haver tingut l'oportunitat de fer ús de molts dels coneixements adquirits durant tota la carrera, el repte que suposava realitzar el projecte final de carrera utilitzant Java com a llenguatge de programació, no ha fet més que motivar l'aprenentatge de dit llenguatge, tot i sabent la dificultat inherent d'adquirir dit coneixement. Per tant, després dels resultats obtinguts i malgrat que la major dificultat trobada ha estat a l'aprenentatge d'aquest llenguatge, crec oportú dir que aquest objectiu ha estat assolit amb èxit.

En quant al resultat obtingut a l'aplicació desenvolupada, malgrat tot i que existeixen punts febles que podrien millorar-se, el balanç final és molt positiu, havent-hi creat una eina funcional que afegeix cert valor a les línies d'investigació dels actuals sistemes de virtualització.

Finalment, no desitjava finalitzar la redacció d'aquest document sense abans qualificar d'un gran èxit tot el treball realitzat durant aquests anys d'estudi que han permès un aprenentatge, tant professional com personal. Per tot açò m'és de gran gratitud poder finalitzar aquest cicle de la meva vida per a començar un de nou que de segur m'oferirà nous reptes i alegries.

Capítol 6.
Annexes

Com a última part d'aquesta memòria, en aquest capítol s'inclouran una sèrie d'annexes relacionats amb el projecte que té com a objectiu complementar tota la documentació aportada per a ajudar al lector a la seva millor comprensió.

6.1. Annexe A. Glossari i Acrònims

A.

- API

  Conjunt de funcions i mètodes en programació orientat a objectes que ofereix una llibreria per a ser utilitzat per altre programari com a capa d'abstracció.

C

- Client

  Aplicació informàtica que accedix a un servei remot en un altre ordinador, conegut com servidor, normalment mitjançant una xarxa de telecomunicacions.

E

- Entorn

  Conjunt de programari que oferix a l'usuari una interacció amigable i còmoda.

F

- Fil

  Característica que permet a una aplicació realitzar vàries tasques a la volta (Concurrentment)

- Front-End

  Part del programari que interactua amb l'usuari

H

- Hash

  Funció o mètode per a generar claus que representen de manera unívoca un document, registre, fitxer, etc.

- HashMap

  Vector associatiu que permet l'ordenació d'objectes mitjançant claus

- Hipervisor

Plataforma que permet aplicar diverses tècniques de control de virtualització per a utilitzar al mateix temps diferents sistemes operatius en un mateix ordinador.

I

- IANA

  És l'agència d'assignació de números d'Internet i antic registre central dels protocols d'Internet, com ports, nombres de protocols i empresa, opcions i codis.

J

- .jar

  Tipus de fitxer que permet executar aplicacions escrites en llenguatge Java.

- JRE

  Conjunt d'utilitats que permeten l'execució de programes Java

- JNI

  Permet que un programa escrit en Java executat en la màquina virtual Java puga interactuar amb programes escrits en altres llenguatges com C.

L

- Llibreria

  Conjunt de subprogrames utilitzats per a desenvolupar programari.

M

- Maquina Virtual

  Programari que emula a un ordinador i pot executar programes com si fora un ordinador real.

- Mòdul

  Tros d'un programa que realitza una part de les tasques.

- Multiplataforma

  Programa, sistema operatiu, llenguatge de programació o altre tipus de

programari que pot funcionar en diverses plataformes.

N

- Núvol

  Paradigma que permet oferir serveis de computació a través d'Internet.

P

- Plataforma

  Sistema que funciona com a base per a fer funcionar determinats mòduls de maquinari o programari amb els que és compatible.

R

- Recurs

  Tot tipus de components de maquinari com de programari que son necessaris per al bon funcionament i l'optimització del treball amb ordinadors, tant a nivell individual com col·lectiu o organitzatiu.

- RSA

  Sistema criptogràfic de clau pública desenvolupat al 1977 i és el primer i més utilitzat algoritme d'aquest tipus i és vàlid tant per a xifrar per a signar digitalment.

S

- Servidor

  Ordinador o programa, que formant part de la xarxa, proveeix serveis a altres ordinadors o programes denominats clients.

- SHA

  Sistema de funcions Hash criptogràfiques de l'Agència de Seguretat Nacional dels Estats Units

- SSL

  Protocol criptogràfic que proporciona comunicacions segures per una xarxa. Normalment Internet.

T

- Targeta Intel·ligent

Targeta de la grandària d'una butxaca amb circuits integrats que permet l'execució de certa lògica programada.

6.2. Annexe B. Bibliografia i Referències

[1] La bíblia de Java, ANAYA MULTIMEDIA ISBN: 9788441510371

[2] Lloc web de col·laboració per a programadors de preguntes i respostes

stackoverflow.com

[3] Programació Java + SSL

stilius.net

[4] Programació i desenvolupament de DNIe

forja.cenatic.es

[5] Bloc de programació Java

lefunes.wordpress.com

[6] Fòrum de Programació Java

p2p.wrox.com

[7] Viquipèdia

ca.wikipedia.org

[8] IDE Java, Netbeans

netbeans.org

[9] Bloc de programació Java en sistemes Linux

drwn.wordpress.com

[10] Formació en línia Inteco

formacion-online.inteco.es

[11] Portal Oficial sobre el DNIe

www.dnielectronico.es

[12] Web d'exemples Java i C/Linux

www.chuidiang.com

[13] Documentació Tècnica de Java SE

download.oracle.com/javase/

[14] Google Code

code.google.com

[15] Pàgina de Desenvolupament d'aplicacions

xda-developers.com