ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Scuola di Ingegneria e Architettura
Corso di Laurea in Ingegneria Elettronica, Informatica e delle Telecomunicazioni

METODI E MODELLI PER L'INTEROPERABILITÀ DI INFRASTRUTTURE DI VIRTUALIZZAZIONE ETEROGENEE: OPENSTACK COME CASO DI STUDIO

Elaborata nel corso di: Sistemi Operativi

Tesi di Laurea di:
NIKOLAY IVANOVICH ZULUAGA PAVLOVA

Relatore:
Prof. ALESSANDRO RICCI

Correlatore:
Ing. NAZZARENO POMPEI

ANNO ACCADEMICO 2012-2013
SESSIONE III

Indice

Introduzione

Al giorno d'oggi si può dire di essere pervasi di risorse di computazione. Il mondo dell'*Information Tecnology* ha visto, dalla seconda metà del novecento sino ad oggi, una crescita di proporzioni esorbitanti tale da diventare ormai punto saldo della società odierna. Questo grazie al continuo progresso in campo elettrico ed elettronico, l'incessante lavoro di ricerca e l'interesse dell'uomo nel trovare sempre nuovi metodi e servizi in grado di semplificare la vita quotidiana.

Con lo sviluppo dei computer, sempre più potenti, è presto sorto il bisogno di trovare metodi con i quali sfruttare a pieno le loro risorse, senza trascurare però quello di garantire una certa sicurezza e affidabilità nelle operazioni compiute. Una delle tecnologie sviluppate da tempo, e che al giorno d'oggi mostra al meglio le sue potenzialità è quella della virtualizzazione. Grazie alle tecnologie di virtualizzazione una singola macchina fisica, specie se ricca di risorse di computazione, risulta non più legata ad un solo specifico ruolo, ma è in grado di fornire svariate immagini di macchine, note come "macchine virtuali", opportunamente settabili in grado di compiere diversi task in maniera parallela e concorrente, in modo isolato o cooperativo. La gestione del periodo di vita di tali macchine virtuali, così come il compito di monitoraggio delle stesse, viene affidato a delle applicazioni che non risiedono nel ambiente virtuale (sotto il controllo del loro sistema operativo) ma astraggono dal *O.S.* garantendo così l'isolamento dei problemi riguardanti una singola immagine. Si ottiene dunque un livello d'astrazione in più che permette di pensare a nuovi metodi per garantire servizi in maniera ottimale.

I calcolatori al giorno d'oggi si presentano in svariati modi e misure, dallo smartphone (ormai equiparabile a livello hardware a un vero e proprio desktop computer) a macchine server atte a supportare ambienti di produzione e servizi di massa. Combinando a piacere questi diversi elementi si possono creare sistemi di grandi dimensioni in grado di fornire gli specifici

servizi desiderati, tali sistemi però, hanno pur sempre bisogno di operazioni di management e manutenzione. Al crescere delle dimensioni di tali sistemi cresce allo stesso modo la difficoltà nel gestirli, ragion per cui è necessario affidarli a personale altamente qualificato in grado di operare tempestivamente al presentarsi di un guasto o di una difficoltà. Con l'uso ormai onnipresente della rete però, tali sistemi possono non esser più centralizzati, ma distribuiti geograficamente nell'intero pianeta (dati i relativi possibili benefici che ciò comporta), complicando ancor di più le cose. Urge dunque tenere presente un modello gerarchico che definisca in maniera precisa i diritti e i doveri dei singoli operatori all'interno del sistema.

Assieme alla ricerca per potenziare le caratteristiche delle macchine vi è anche una costante ricerca riguardo al come ottimizzare la loro gestione. Non è più pensabile di lasciare tutto in mano a persone fisiche e sempre più task all'interno della sfera del management vengono lasciati ad entità computazionali indipendenti (ad esempio agenti computazionali) in grado di far fronte a diverse difficoltà che possono presentarsi all'interno di un sistema lasciando gli amministratori di sistema liberi di pensare ad altri compiti. Caratteristica particolare del progresso è che nuove scoperte o invenzioni portano a nuove problematiche alle quali bisogna far fronte. Non si può pensare di gestire un ambiente produttivo allo stesso modo in cui si gestisce il computer di casa. Questo ragionamento può essere scalato fino al livello in cui ci si trova al giorno d'oggi. Gli ambienti produttivi di maggiori dimensioni non sono più geograficamente centralizzati bensì dislocati nel territorio, suddividendosi in diversi sottosistemi centralizzati. Va da sé che la gestione del macro-sistema non può essere la stessa del singolo micro-sistema. Caso particolare di questo scenario è l'ormai noto *cloud*, un'infrastruttura di *resource sharing* che fa affidamento sulle tecnologie di virtualizzazione per offrire ai suoi utenti un insieme di risorse apparentemente illimitato, astraendo dal vincolo della singola macchina fisica ed offrendo un vasto insieme di vantaggi propri del modello su cui si basa.

Il seguente lavoro ha come intento presentare una delle principali problematiche che ci si può trovare ad affrontare quando si ha a che fare i suddetti macro-sistemi ovvero l'interoperabilità tra i singoli sotto-sistemi, specie nel caso essi non siano omogenei tra di loro. Inoltre si vuole presentare il modello di una possibile soluzione a tale problema basato su un'interfaccia in grado di astrarre dalla specifica natura dei singoli sotto-sistemi, supportata da opportuni protocolli di comunicazione per garantire al meglio l'affidabilità

e tempestività delle più ricorrenti operazioni nella gestione quotidiana di un ambiente di produzione di considerevoli dimensioni.

Nel primo capitolo s'introduce il concetto di virtualizzazione, prestando particolare attenzione alla figura dell'*Hypervisor*, di vitale importanza per l'esistenza stessa dei sistemi di virtualizzazione adoperati negli ambienti di produzione, in quanto è quest'ultimo che si occupa della gestione delle macchine virtuali all'interno di una singola macchina fisica. S'illustra infine il modello più usato, basato sulla virtualizzazione, secondo il quale più macchine fisiche disgiunte sono gestite da una stessa entità che comunica direttamente con i singoli *Hypervisor* prendendo come esempio la piattaforma *VSphere* di *VMware*.

Successivamente, nel secondo capitolo, s'introduce il modello di *Cloud Computing*, scenario nel quale si presenta più spesso la situazione che si vuole analizzare date le grandi dimensioni di tale modello. In questo tipo di sistemi non soltanto la possibilità di astrarre dalle tecnologie di virtualizzazione è di vitale importanza, è necessario anche avere gli opportuni meccanismi che permettano una gestione e monitoraggio dell'intera infrastruttura in maniera centralizzata. Si vuole sottolineare come le tecnologie di virtualizzazione siano alla base di questa realtà, la quale, se non impossibile, sarebbe di certo molto onerosa e poco conveniente in assenza di tali tecnologie. Per completezza si concludono i preamboli presentando diversi esempi di *Cloud* presenti al giorno d'oggi, illustrandone brevemente le caratteristiche e i possibili modi d'uso.

Il terzo capitolo vuole essere il "*core*" dell'intero lavoro di tesi, nel quale si analizza il problema dell'interoperabilità tra sistemi basati su una data tecnologia di virtualizzazione. Si introduce brevemente il formato *OVF*, spesso erroneamente interpretato come soluzione immediata a tale problema e si propone un modello secondo il quale è possibile ovviare, entro certi limiti, alle problematiche di interoperabilità e gestione remota, garantendo un insieme di operazioni, sebbene limitato, anche in situazioni avverse.

Il quarto e ultimo capitolo analizza il caso di studio concreto di una piattaforma *OpenStack*, la quale adopera un modello molto simile a quello proposto. Si introduce l'ambiente predisposto facendone presente il suo possibile uso e si conclude con esempi utili a capire il funzionamento dei metodi di gestione delle risorse virtuali in una piattaforma basata sul modello del *Cloud Computing*.

Capitolo 1

Tecnologie di Virtualizzazione

Per comprendere a pieno le problematiche trattate in questo lavoro, che si presentano specie nei sistemi distribuiti in larga scala o comunque di grandi dimensioni, è necessario tenere chiari i concetti alla base delle tecnologie di virtualizzazione, i diversi modelli secondo i quali può essere realizzata e i diversi elementi che implementano tale modello al giorno d'oggi.

1.1 Introduzione

Uno dei supporti maggiormente usati per gestire la complessità di struttura di una macchina è dato dal concetto di astrazione. Tramite questo approccio si tende a scomporre un problema in diverse parti ben definite, dette *"livelli"*. In questo modo di può assumere un certo ordine nelle azioni da compiere al fine di portare a termine un compito. Caratteristica particolare di questo approccio, è che ciascun livello è concepito in modo da nascondere agli altri livelli la sua complessità e tutti i dettagli che lo riguardano. Ad ogni livello va poi associata una "*interfaccia*" che elenca l'insieme di servizi che offre tale livello. In questo modo i livelli comunicano tra di loro, delegando la complessità del risolvere certi problemi al livello di competenza.

Se si considera un computer, macchina per la quale risulta nota la complessità di funzionamento, si può pensare anzitutto di fare una suddivisione in due livelli: *hardware* e *software*. L'interfaccia caratteristica del livello hardware risulterà quindi l'*Instruction Set* (IS). Tale interfaccia, propria della CPU, espone al livello software un insieme di servizi per interrogare le risorse fisiche della macchina. Se poi l'IS viene reso standard, allora un ele-

mento incapsulato nel livello software potrà (potenzialmente) funzionare su una qualsiasi macchina che presenti lo stesso *instruction set* per il quale è stato progettato.

Tramite questo approccio è stato possibile progettare dei sistemi operativi basati su un'interfaccia standard frapposta tra il livello software e quello hardware, compatibili con gran parte dei computer presenti in commercio. In questo modo si presenta dunque un vantaggio di compatibilità, limitato soltanto dal fatto che si deve sempre tenere conto della natura dell'interfaccia con la quale si ha a che fare. D'altro canto si possono individuare anche svantaggi in questo approccio. Si pensi allo scenario (piuttosto comune) di un livello mappato direttamente su un altro (detta mappatura 1 a 1). Nel caso di un computer risulterebbe in una macchina sotto il diretto controllo di un unico sistema operativo. Al giorno d'oggi, considerata la capienza e potenza raggiunta a livello hardware, tale mappatura si traduce in uno spreco di risorse, poiché mediamente inutilizzate per l'intero ciclo di vita della macchina. Inoltre, l'eventuale malfunzionamento (accidentale o doloso) del sistema operativo, può compromettere direttamente la salute macchina. Sono stati proprio i suddetti vantaggi a rendere tanto popolare l'approccio della virtualizzazione negli ultimi decenni, nonostante il termine e modello fossero stati coniati già negli anni '60 presso i laboratori *IBM*.

L'idea di base attorno alla quale ruota il concetto della virtualizzazione è proprio quella di frapporre un ulteriore livello tra quello hardware e software, in modo da poter usufruire di una serie di vantaggi che verranno illustrati più avanti in questo capitolo. Nota infine la natura del modello di virtualizzazione e compreso perché ad oggi è un approccio ampiamente adoperato si potrà passare a discutere delle problematiche che possono sorgere se si vogliono sfruttare a pieno le potenzialità della virtualizzazione.

1.2 Concetti di Base

Il livello frapposto tra quello hardware e software che va a realizzare il concetto di virtualizzazione introdotto precedentemente viene chiamato *Virtual Machine Monitor (VMM)*, esso si concretizza in un modulo software prioritario rispetto ad altri, il quale emula al meglio il comportamento del hardware sottostante, rendendolo disponibile a ulteriori moduli software che si affidano al *VMM* per adempiere ai loro compiti. Ogni richiesta da parte di uno di

questi moduli software, destinata alle risorse fisiche del sistema, passa dunque attraverso il *VMM*. Risulta quindi evidente la nuova disposizione dei livelli d'astrazione. Implementando dunque opportunamente il *VMM* è possibile mettere a disposizione dei livelli soprastanti meccanismi di concorrenza onde ottimizzare l'utilizzo delle risorse.

Il nome *Virtual Machine Monitor* deriva proprio dal fatto che i livelli soprastanti al *VMM* vengono chiamati *Virtual Machines (VM)* e sono strettamente legate al loro livello sottostante che le gestisce, coordina e monitora. Noti dunque questi due elementi, si ha da una parte, un livello che mette a disposizione in maniera intelligente e sicura le risorse fisiche del sistema (*VMM*), e dall'altra, un livello soprastante che ne usufruisce tramite una determinata interfaccia (*VM*). Si completa così una prima implementazione del modello di virtualizzazione. I singoli metodi di implementazione dei livelli relativi al modello possono portare poi a diversi tipi di virtualizzazione. In particolare, per le *VMs*, esse possono incapsulare singole applicazioni (*Process Virtual Machine*) o interi sistemi operativi (*System Virtual Machine*). Al fine di comprendere al meglio la natura del *VMM* ed evidenziare la differenza tra questi due tipi di virtualizzazione (che risultano essere quelli più diffusi al giorno d'oggi), è anzitutto necessario comprendere il funzionamento di un'applicazione o sistema operativo, rispetto al hardware ad esso sottostante facendo a meno del neo-introdotto livello di virtualizzazione. Tornando alla visione in cui si ha un livello software che dialoga 1 a 1 con

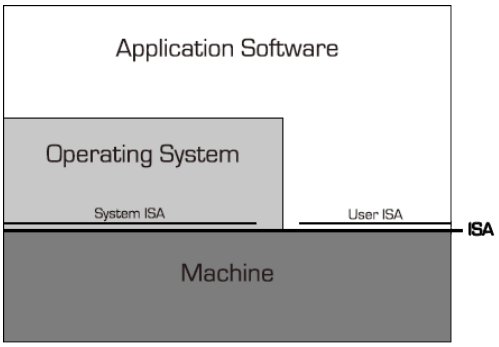

Figura 1.1: ISA - Instruction Set Architecture

il livello hardware sottostante (essendo questo un approccio in precedenza largamente utilizzato), ogni richiesta avviene tramite l'interfaccia messa a disposizione dal livello hardware,già nota come *Instruction Set*. In questo

modello, tali richieste possono avvenire in 2 modi: *User Mode* o *Kernel Mode*. Quest'ultimo risulta prioritario e privilegiato rispetto al primo poiché ad esso non si applicano particolari restrizioni o controlli, lo *User Mode* invece è una modalità che presenta limiti determinati da una *security policy* atta a garantire il giusto funzionamento del sistema, tramite questa modalità è possibile eseguire soltanto determinati tipi di operazioni.

Dal punto di vista di un'applicazione o processo è possibile agire in due modi: tramite il sistema operativo oppure interrogare le risorse hardware direttamente. In quest'ultimo caso si andrà ad usufruire dell'*IS* in *User Mode* ma come detto in precedenza le possibilità saranno piuttosto limitate. Viceversa, il sistema operativo mette a disposizione dei processi soprastanti un interfaccia nota come *System Call Interface* tramite la quale sarà possibile interrogare risorse hardware in *Kernel Mode*. In questo caso, la *security policy* viene messa in atto dal sistema operativo, che dovrà quindi prevederne una piuttosto efficace. Dal punto di vista del singolo processo quindi, le istruzioni del *IS* in *Kernel Mode* equivarranno alle *System Calls*, che assieme alle istruzioni in *User Mode* concretizzano la *Application Binary Interface (ABI)*.

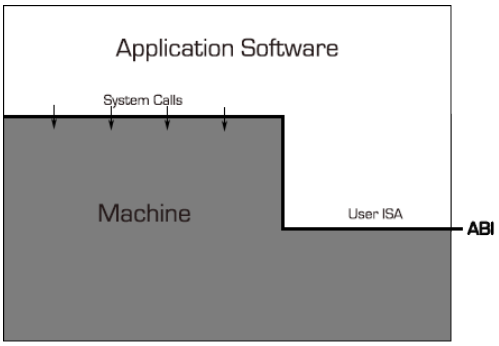

Figura 1.2: ABI - Application Binary Interface

Si evidenziano le particolarità delle interfacce ISA ed ABI poiché alla base del comportamento del *VMM* nei confronti delle macchine virtuali che gestisce.

1.3 Modelli di Virtualizzazione

Con l'intento di illustrare brevemente la differenza sostanziale tra i due tipi diversi di virtual machines, alla luce di quanto si è finora esposto, si può dire che nel caso delle *process virtual machines* il *VMM* si comporta nei confronti della *VM* come un'interfaccia ABI mentre invece, per quanto riguarda le *system virtual machines* esso offre le stesse funzionalità dell'interfaccia ISA.

Rispetto all'astrazione a due livelli inizialmente presentata nell'introduzione, si è andato avanti aggiungendo livelli e specificandone particolari dettagli. Giunti a questo punto, conviene fare un'ulteriore suddivisione lungo la pila di livelli utile a esplicitare la differenza tra i diversi tipi di virtual machines. Tale differenza sta proprio nel ruolo del sistema operativo. Esplicitando sempre il *VMM* come livello "ponte" tra software e hardware, s'introduce il concetto di *host* e *guest* come ruoli rispetto al livello hardware e al livello di virtualizzazione. Se si parla di una *process virtual machine*, il sistema operativo farà parte della parte *host* e si troverà ad affiancare il *VMM* poiché non strettamente dipendente da esso. Nel caso di una *system virtual machine* invece, il sistema operativo è parte integrante di quest'ultima, fa dunque parte della machina virtuale ed è direttamente legato al *VMM*. In questo caso esso fa quindi parte della parte *guest* del modello.

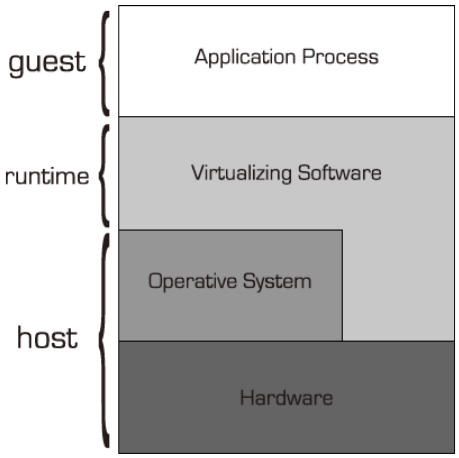

Figura 1.3: Process Virtual Machine

Process Virtual Machine Questo tipo di macchina virtuale incapsula un singolo processo, nasce con l'esecuzione di quest'ultimo e muore quando esso viene terminato. Le *process virtual machines* vengono eseguite come normali processi ad interno del sistema operativo che le ospita ed offrono un ambiente in grado di astrarre dalle specifiche della piattaforma hardware sulla quale si erge l'intero sistema.

Un ottimo esempio di questo tipo di virtual machines può essere ritrovato in alcuni dei più comuni linguaggi di programmazione di alto livello quali Java o il .NET Framework i quali per l'implementazione dei loro programmi fanno uso rispettivamente della *Java Virtual Machine* il primo e del *Common Language Runtime* il secondo.

System Virtual Machine Come già accennato, una *system virtual machine* incapsula al suo interno un intero sistema operativo sopra al quale verranno poi eseguiti diversi processi. Poiché resta però una virtual machine, essa si trova interamente sopra lo strato di virtualizzazione, il quale fa da mediatore tra processi, sistema operativo e hardware sottostante. Rispetto alle architetture precedentemente illustrate, il *VMM* si comporta dunque come un interfaccia ISA. Sistema operativo e relativi processi operano in modalità

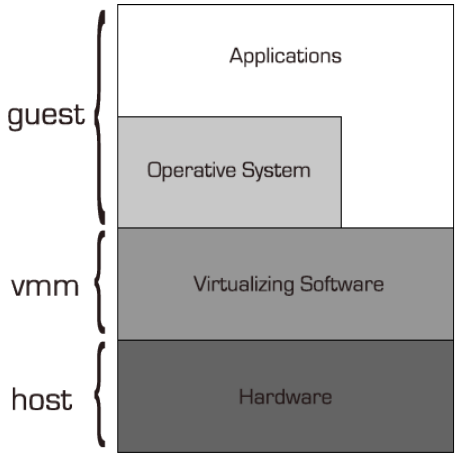

Figura 1.4: System Virtual Machine

user (*guest*). Il *VMM* sarà dunque l'unico componente in grado di effettuare richieste in *kernel mode* e dovrà dunque essere in grado di interpretare e tradurre opportunamente le chiamate che gli arriveranno dall'alto.

Risulta evidente in questo modo che, poiché il sistema operativo appare debolmente accoppiato con il hardware sottostante, lo strato *guest* può essere scalato orizzontalmente dando luogo ad un modello che supporta l'esecuzione in contemporanea di più macchine virtuale su un'unica macchina fisica, ognuna di queste risulterà totalmente disaccoppiata dalle altre ed ogni istanza potrà essere configurata in maniera eterogenea in base alle esigenze. Si avranno così i relativi vantaggi accennati in fatto di ottimizzazione di uso delle risorse a disposizione, sicurezza e *fault tolerance*. Ovviamente tutto ciò grava sulle spalle del *VMM* che dovrà dunque essere robusto ed opportunamente implementato per gestire al meglio lo scenario presente.

1.4 L'Hypervisor (Virtual Machine Monitor)

Lo strato di virtualizzazione (noto anche come *Hypervisor* oltre che come *VMM*) media tra i livelli ad esso soprastanti ed il hardware sottostante. Tuttavia, la flessibilità di questo modello permette di fare una classificazione in base al come si presentano rispetto al layer del sistema operativo. Storicamente sono stati definiti 2 tipi di *hypervisor*:

- Type 1 (o nativo, *bare metal*): che, come risulta evidente nelle figure 1.1 ed 1.4 viene eseguito direttamente sopra lo strato hardware, del quale si avvale per virtualizzare le risorse destinate alle virtual machines soprastanti. Un *hypervisor* di tipo 1 può essere equiparato ad un vero e proprio sistema operativo, di fatto, le più recenti versioni rilasciate da parte dei diversi produttori vengono installate direttamente sullo strato fisico, del quale la procedura d'installazione è in grado di distinguere la natura e quantificarne le risorse per gestirle al meglio.

- Type 2 (o *hosted*): che si presenta come processo ospitato da un sistema operativo sottostante come si può vedere dalla figura 1.3. In questo caso il *hypervisor* risulta equiparabile ad un qualsiasi altro componente software.

Questa classificazione permette immediatamente di determinare delle profonde differenze tra i due tipi di *VMM* a livello di performance. Tipicamente gli *hypervisor* di tipo 1 sono più efficienti rispetto a quelli di tipo 2 e di fatto la loro destinazione è diversa. Essi trovano largo uso negli ambienti di produzione o in ambito professionale, dove si ha a disposizione un parco macchine

con caratteristiche hardware abbastanza elevate. D'altro canto, *hypervisor* di tipo 2 sono di più facile utilizzo e possono essere destinati ad ambienti desktop o casalinghi, permettendo di usufruire dei vantaggi e benefici della virtualizzazione per ovviare a problematiche minori.

L'aggiunta di uno strato di virtualizzazione comporta un carico non indifferente in termini di *overhead* ad ogni richiesta destinata al hardware alla base del sistema poiché esso si dovrà occupare personalmente di tradurre ed instradare le chiamate. L'ottimizzazione di questi fattori tramite diversi metodi come il *caching* o la *paravirtualizzazione*, così come l'implementazione di *policy* robuste in materia di sicurezza e *fault tolerance* determinano in buona parte la qualità del software di virtualizzazione. Va da sé che è possibile aggiungere più livelli di virtualizzazione (basti pensare ad un *hypervisor* di tipo 2 che risiede in una virtual machine, e dunque già sopra uno strato di virtualizzazione) la dove necessario; i carico in termini di perdita di performance però sarà certamente notevole. Inoltre, tale operazione è pressoché inutile vista la natura isolata che caratterizza le singole virtual machines. Sarebbe equivalente e meno oneroso disporre un'ulteriore virtual machine nello stesso livello di virtualizzazione piuttosto che scalare verticalmente il modello.

Visto il ruolo di mediatore che svolge l'*hypervisor* rispetto all'intera infrastruttura virtuale ad esso soprastante, non desta stupore pensare che esso ha bisogno di meccanismi opportuni per poter gestire e monitorare l'attività di ogni componente virtuale sotto il suo controllo. Di fatto, ogni tecnologia di virtualizzazione mette anche a disposizione di utenti e sviluppatori una *API (Application Programming Interface)* tramite la quale è possibile dialogare con l'*hypervisor* per indurlo a svolgere diversi tipi di compiti che possono andare dalla gestione elementare di una macchina virtuale (accensione/spegnimento, allocazione dinamica di risorse, migrazione) al monitoraggio complesso di processi interni ai singoli sistemi operativi virtualizzati. Grazie a questo approccio d'accesso al sistema "dal basso" che si vede contrapposto al solito "dall'alto" relativo ai sistemi non virtualizzati, è possibile ampliare le possibilità di gestione delle infrastrutture di virtualizzazione. Tramite meccanismi di *load balancing*, l'*hypervisor*, monitorando l'uso effettivo delle risorse da parte delle singole macchine virtuali può ridistribuirle al meglio per incrementare l'operatività e l'efficienza del sistema. Altri possibili usi di questi meccanismi d'accesso possono riguardare applicativi lanciati all'interno stesso dell'infrastruttura virtuale. Tramite l'uso delle *API*, applicazioni di monitoraggio o gestione possono intervenire direttamente su altri compo-

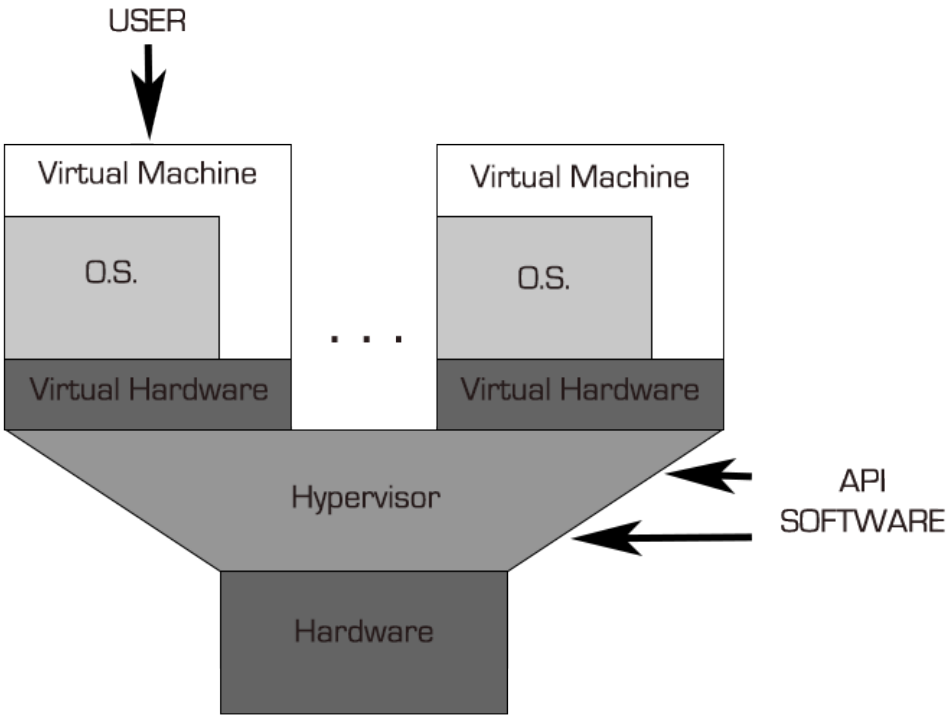

Figura 1.5: Accesso al *VMM* tramite API software

nenti dell'infrastruttura virtuale, scavalcando la barriera d'isolamento della virtual machine grazie al meccanismo d'accesso tramite il *VMM*. In questo modo la gestione di un componente virtuale non è vincolato al ciclo di vita del componente stesso poiché l'applicazione di gestione può risiedere in un altro componente all'interno dell'infrastruttura virtuale o addirittura (sotto opportune configurazioni) all'esterno di essa. La possibilità di gestire l'intera infrastruttura virtuale dall'esterno introduce però una serie non indifferente di problematiche relative alla sicurezza del sistema, sarà compito dell'amministratore di sistema disporre delle opportune *security policy* e configurare a modo il sistema onde garantire l'integrità dello stesso.

1.5 La Virtualizzazione negli Ambienti di Produzione

Come già accennato nella presentazione degli *hypervisor* di tipo 1. Negli ambienti di produzione sono proprio questi ultimi a fare da protagonisti per quanto riguarda l'infrastruttura virtuale. Scenario tipico in un ambiente di media grandezza è quello di avere a disposizione un numero limitato di macchine disposte a RACK presso un locale apposito e configurate in modo da offrire un insieme più o meno vasto di servizi all'ambiente stesso ma anche all'esterno; noto l'insieme con il nome di *Datacenter*, tali macchine saranno poi facilmente raggiungibili tramite svariati protocolli come `http`, `rdp`, `ssh` o altri a seconda delle necessità. Le macchine delle quali si parla non sono però

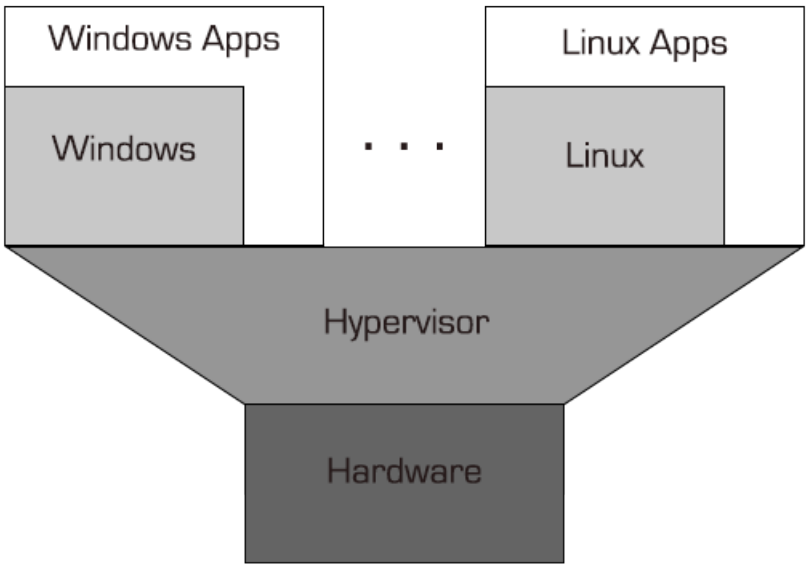

Figura 1.6: Scenario Server

dei semplici personal computer, poiché presentano caratteristiche hardware particolarmente elevate. In esse è presente una quantità di memoria RAM, processori, schede di rete e memoria di massa tale da poter garantire una gestione più o meno fluida di un numero elevato di virtual machines secondo il modello presentato in figura 1.6. È buona pratica quella di riservare

ad una sola macchina virtuale pochi compiti. Questo al fine di sfruttare al meglio la *fault tolerance* introdotta dal *layer* di virtualizzazione, che isola le singole virtual machines dalle altre gestite dallo stesso *VMM*. Ad esempio, si può disporre un insieme di macchine virtuali come web server, e dedicare ognuna ad un solo sito web. In questo modo, se una di esse presenta problemi, tutti gli altri portali sono ancora raggiungibili ed eventuali soluzioni possono essere attuate facile e velocemente tramite l'*hypervisor*. La gestione ed amministrazione di queste strutture risulta infatti semplificata grazie al modello di virtualizzazione poiché si agisce in maniera più o meno centralizzata direttamente sull'*hypervisor* per avere controllo sull'intera infrastruttura virtualizzata. Al giorno d'oggi ogni *vendor* mette a disposizione dei propri clienti diversi *tool* ottimizzati al fine di gestire in maniera immediata un numero dinamico dei loro *hypervisor*. Spesso questi stessi tool si presentano a loro volta come macchine virtuali già configurate e pronte per l'uso, come avviene ad esempio con il *vCenter* di *vmware* in grado non solo di gestire diversi *hypervisor* in contemporanea ma anche diversi *datacenter* dislocati.

Capitolo 2

Virtualizzazione e Cloud Computing

Reso noto il concetto di virtualizzazione, al fine di comprendere a pieno gli obiettivi del presente lavoro risulta conveniente introdurre un altro concetto che appare strettamente collegato al primo e cioè il *Cloud Computing*. Questo modello ha visto la luce soprattutto grazie allo sviluppo infrastrutturale che riguarda la rete dati, permettendo di andare oltre il semplice pattern *client/server* che contro-distingueva i servizi offerti tramite *internet* nei decenni passati. Si è quindi affermato nel mercato grazie ai vantaggi che introduce, spesso legati alle tecnologie di virtualizzazione che sono alla base del modello. Lo studio di questo modello risulta di particolare interesse per il presente lavoro poiché palesa la necessità di meccanismi d'interoperabilità tra tecnologie di virtualizzazione. Questo poiché le dimensioni delle infrastrutture *cloud* spesso possono giustificare i casi in cui le piattaforme di virtualizzazione su cui si basa l'intera infrastruttura non siano omogenee.

2.1 Cos'è il "Cloud Computing"?

È già stato reso ben noto come il progresso tecnologico abbia permesso la nascita di nuove tecnologie con l'intento di sfruttare al meglio le recenti scoperte in ambito elettronico e informatico. Così come la potenza di calcolo e capacità dei singoli computer è cresciuta enormemente, così lo è stato anche per la capacità e velocità delle tecnologie alla base delle telecomunicazioni. Col tempo l'*internet* si è affermato non solo come un modo per poter reperire

documenti ed informazioni in maniera agile ma anche come mezzo per offrire svariati tipi di servizi. Al giorno d'oggi, grazie al supporto di tante altre tecnologie tra le quali quella di virtualizzazione, si è in grado di offrire intere piattaforme o infrastrutture come servizi in rete.

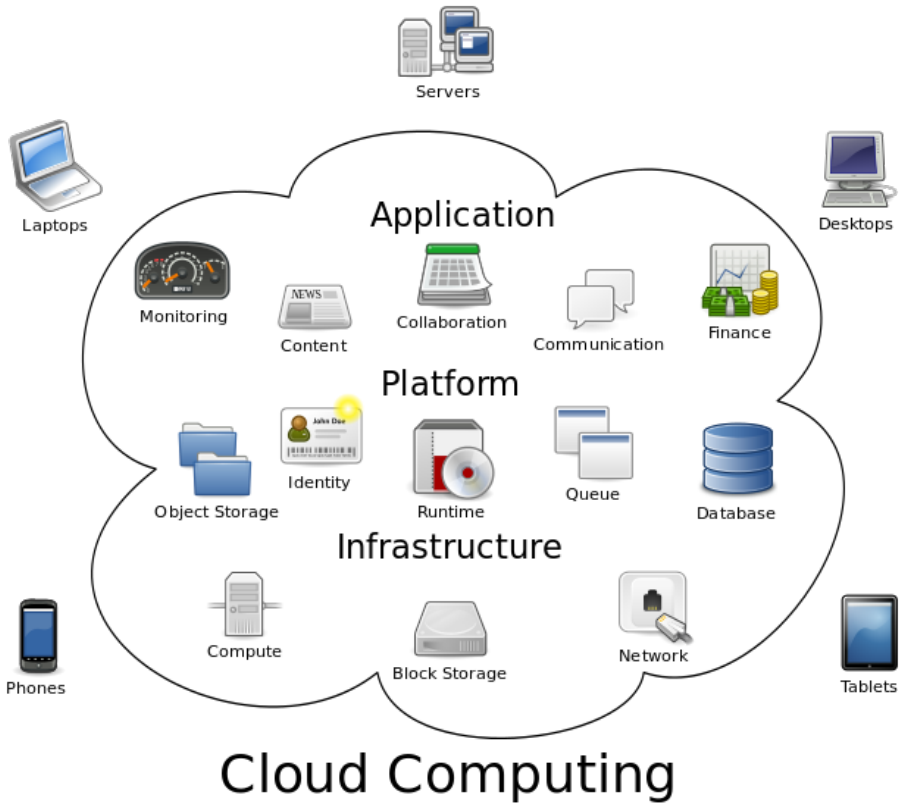

Figura 2.1: Cloud Computing

Si parla dunque di *Cloud* come un luogo virtuale, raggiungibile tramite la rete, in cui si è in grado di reperire e disporre di dati, applicazioni, piattaforme o addirittura intere infrastrutture e in generale risorse *on-demand*. Questo grazie alla sua natura, che risulta essere quella di un sistema distribuito di grandi dimensioni e con risorse apparentemente infinite.

Economicamente parlando, le soluzioni *cloud-based* si sono affermate nel mercato grazie alla semplicità con la quale si presentano, sia a livello di

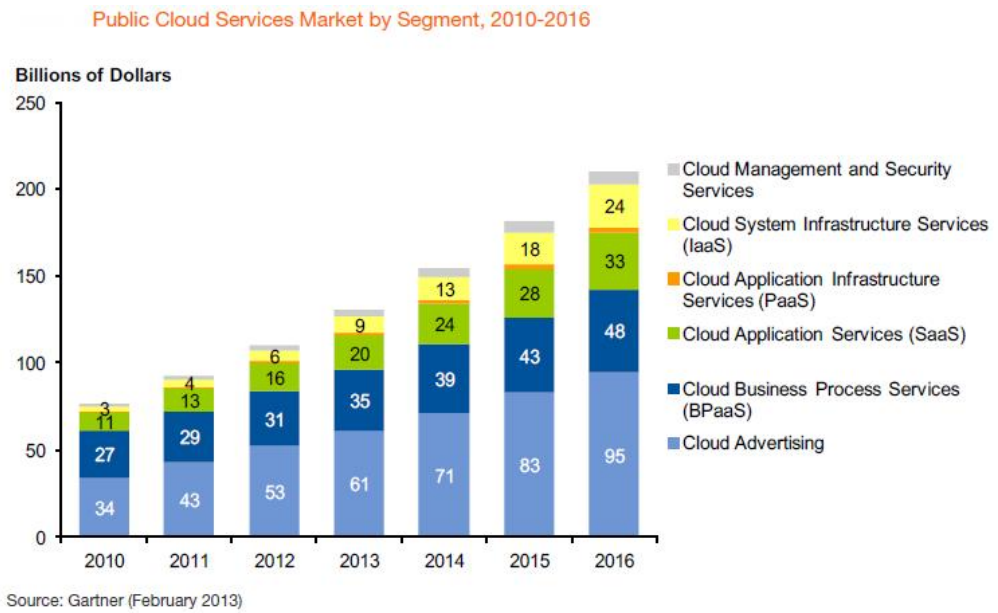

Figura 2.2: Previsioni per il mercato dei servizi *cloud-based* al 2013

messa in piedi che di gestione e manutenzione, rendendole una scelta più che appetibile per aziende che basano gran parte del loro operato su infrastrutture informatiche. D'altro canto proprio la tendenza a virare verso un operato *cloud-based* ha dato una considerevole spinta al mercato relativo tali servizi come evidenziato dal crescente giro di affari che lo riguarda illustrato nella figura 2.2.

Quale sistema distribuito, il *Cloud* è in grado di rendere del tutto trasparenti molti aspetti delle risorse alle quali si sta accedendo, come ad esempio la loro locazione geografica, la loro eventuale migrazione, rialocazione o scalatura. Insomma, il modello del *Cloud Computing* rende molto più agevole ed economico il *deployment* di idee nell'ambito dell'*IT*.

I principali modelli riguardanti un servizio *Cloud-based* sono noti come:

- SaaS - Software as a Service: Tramite questo modello delle applicazioni vengono accedute dall'utente finale tramite un client locale come ad esempio un *web browser*, un client dedicato o una *mobile app*, esse vengono però eseguite intera o parzialmente lato server con i relativi vantaggi che ciò comporta. È possibile ad esempio immagazzinare in un *web-server* grandi quantità di dati che restano tuttavia accessibili e

consultabili all'*end user* tramite il client opportunamente predisposto per tale compito, questa situazione risulta particolarmente vantaggiosa nel caso di dispositivi con ridotta capacità di memoria di massa come ad esempio gli *smartphones* o i *tablet*. Tipicamente la piattaforma e l'infrastruttura sulla quale poggia l'applicazione sono gestite da un *Cloud Provider*, che mette a disposizione del suo cliente (colui che a sua volta eroga il servizio all'*end user*) le risorse necessarie per tenere in vita ed eseguire l'applicativo software. Uno dei principali vantaggi di questo modello riguarda la scalabilità che presenta. Si pensi ad un'applicazione web, le sue risorse a disposizione possono essere facilmente incrementate o diminuite in maniera del tutto dinamica a seconda del carico d'utenza a cui si trova a far fronte senza comportare particolari costi per il gestore proprietario dell'applicativo, il quale si troverà a dover pagare solo per le risorse usate effettivamente.

- PaaS - Platform as a Service: In questo modello il *Cloud Provider* mette a disposizione dell'utente un'intera piattaforma di computazione, tipicamente già provvista da un sistema operativo e configurata opportunamente per far fronte alle necessità del cliente. Tutto questo può comportare un ingente vantaggio economico ad esempio quando si ha necessità di elevate capacità di calcolo per periodi ridotti di tempo; Si abbattono del tutto costi d'acquisto di macchine che andrebbero ad essere inutilizzate ed il servizio può essere ricondotto al caso di "affittare" un computer la quale configurazione opportuna potrà inoltre limitarsi al solo montaggio di un'immagine contenente una macchina virtuale già predisposta per i compiti da eseguire.

- IaaS - Infrastructure as a Service: Questo modello prevede la messa a disposizione di un'intera infrastruttura virtuale e dunque anche di meccanismi di *networking*, *computation* e *storage* così come altri servizi. In questo modo il cliente può disporre come meglio desidera un sistema più o meno complesso che può necessitare di un numero variabile di macchine (virtuali) adatte alle sue necessità. Per questo servizio si può dire senza margine d'errore che si mette a disposizione dell'utente finale l'equivalente di un *Hypervisor* gestendo il quale l'utente è in grado di adempiere alle sue necessità per conto proprio.

Si tende inoltre a suddividere le possibili implementazioni del modello di *Cloud Computing* in diversi tipi: *public*, *private* ed *hybrid cloud*. Le differenze

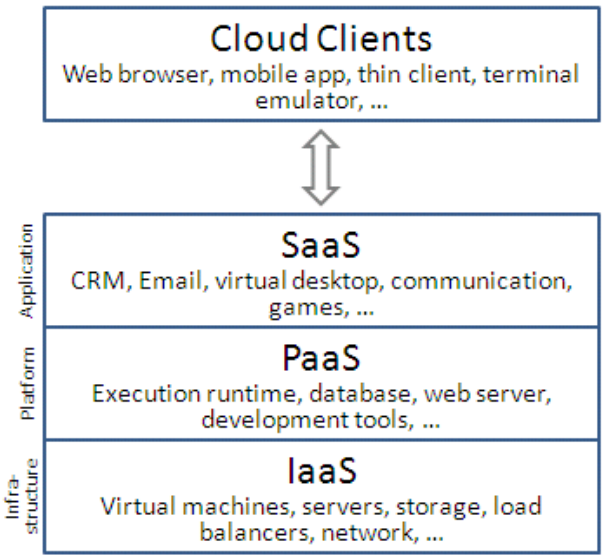

Figura 2.3: Livelli che caratterizzano un servizio *cloud-based*

tra loro a livello di implementazione sono sottili e riguardano principalmente aspetti di sicurezza e limitazioni di accesso, è in particolare la loro destinazione a renderle diverse. La soluzione ibrida, come si può intuire dal suo nome, risulta una combinazione tra le prime due.

2.1.1 Public Cloud

Il modello pubblico risulta quello più noto al mondo, esso rispecchia anche in maniera piuttosto precisa il concetto nudo e crudo di *Cloud Computing* poiché non ne applica alcuna restrizione. Si parla dunque di *public cloud* quando si ha un insieme di risorse fisiche eterogenee accessibili tramite una rete pubblica (come ad esempio via internet), sul quale vi poggia un ambiente virtuale in grado di erogare i propri servizi in maniera condivisa e concorrente all'utenza. L'uso di una singola macchina non risulta monopolizzato bensì si verifica una situazione di *multitenancy* o "multiutenza", in cui una singola macchina (fisica o virtuale) è in grado di servire più client alla volta ottimizzando le performance del sistema. L'erogazione dei servizi *cloud-based* può avvenire su più livelli come quelli sopra elencati. Tipicamente infatti, un *Cloud Provider* mette a disposizione dei suoi clienti diverse tipologie di servizi basati sul metodo di utilizzo di parte della sua infrastruttura.

Essendo il *Cloud* un sistema distribuito che si affida alle tecnologie di virtualizzazione, le basi fisiche di tale sistema si tradurranno in una serie di *Datacenter* di dominio del *Cloud Provider*. Essi con grande probabilità risulteranno ampiamente dislocati nel territorio per garantire una migliore qualità del servizio a un'utenza internazionale. Tutto ciò risulta però del tutto trasparente all'utente finale, il quale unico interesse sarà l'accesso alle risorse. I servizi erogati dal *cloud* saranno poi gratuiti o a pagamento a discrezione del *cloud provider*, tipicamente, specie per servizi IaaS e PaaS il costo viene valutato sulla base delle risorse effettivamente consumate mentre per servizi SaaS i metodi di tariffazione classici prevedono una proporzione rispetto al traffico generato dall'applicazione sulla rete.

Onde garantire una maggiore compatibilità ed *openess* dei loro sistemi con diverse altre soluzioni nell'ambito informatico, i principali *Cloud Provider* mettono a disposizione degli sviluppatori delle API (Application Programming Interface) apposite per regolare l'interazione con le loro piattaforme. In questo modo le *public cloud* risultano ancor più vantaggiose poiché adattabili alle necessita specifiche dei singoli utenti ed espandibili a nuovi orizzonti implementativi.

2.1.2 Private Cloud

Si parla di *private cloud* quando si ha a che fare con una piattaforma *Cloud* destinata all'uso esclusivo di un'organizzazione o di un numero limitato di utenti. Questo approccio può persino andare a sostituire la presenza di un *Datacenter* all'interno di un'azienda, ovvero un'infrastruttura basata su virtualizzazione in grado di offrire i servizi necessari ai suoi dipendenti; oppure può coesistere con esso a seconda dei casi. L'intento di una *private cloud* è ad ogni modo quello di mettere a disposizione una quantità dinamica di risorse da destinare alle necessità dell'azienda. Tipicamente le dimensioni di una *private cloud* saranno minori rispetto a quelle della sua pari pubblica, inoltre essa introduce delle forti restrizioni in termini di sicurezza, è infatti di particolare interesse per l'amministratore della piattaforma il controllo dei dati relativi all'azienda nonché altri fattori che poco interessano una *public cloud* quali la natura dei collegamenti tra le macchine o la loro disposizione. Tipicamente una *private cloud* implementerà opportuni meccanismi di crittografia per le comunicazioni nonché meccanismi di accesso sicuro per i suoi utenti al fine di salvaguardare il suo interno.

Con riferimento ai concetti già inseriti come *IaaS* nonché alla possibilità di estendere la virtualizzazione a più livelli, si può predisporre una *private cloud* anche attraverso una *public cloud*. Si avrà in questo caso una cosiddetta *Virtual Private Cloud* (VPC) che si traduce nell'allocazione di risorse all'interno di una *public cloud* destinate ad un esclusivo utente, nel fare tale operazione, si presta particolare attenzione ai metodi d'accesso. Tipicamente si dispone una VLAN quale canale d'interconnessione tra le macchine virtuali appartenenti alla VPC e connessione VPN per l'accesso ad esse. In questo modo si garantisce l'integrità e sicurezza dei dati che ne trafficano all'interno.

2.1.3 Hybrid Cloud

Come si può facilmente intuire, si parla di *hyrid cloud* quando si ha un sistema composto da più *cloud* che possono essere pubbliche o private. Le singole entità che compongono la *hybrid cloud* restano indipendenti e ben definite ma "cooperano" tra di loro in un qualche modo. Inoltre, tipicamente ogni entità sarà associata ad un *cloud provider* diverso, poiché altrimenti si ricadrebbe nel mero caso pubblico o privato. l'approccio ibrido può risultare molto utile quando si vuole usufruire dei vantaggi in termini di sicurezza che offre una *private cloud* per applicativi che tuttavia, per motivi di disponibilità, risiedono in una *public cloud*. Altri casi possibili possono essere riscontrati nell'attività cooperativa a tempo determinato da parte di due aziende, che si troveranno ad "accoppiare" debolmente le loro infrastrutture private.

Dal punto di vista di questo elaborato, proprio il caso di *hybrid cloud* può presentarsi come un caso di studio. In genere, sin dalle prime fasi della progettazione di un ambiente *cloud-based*, risulta evidente come sia più vantaggioso implementare l'intera infrastruttura virtuale sotto una singola tecnologia. L'evidenza però di casi in cui questo non è possibile, come ad esempio nell'intenzione di unire due *cloud* già esistenti, che eventualmente prevedono tecnologie di virtualizzazione diverse, mette in risalto la necessità di un modello di interoperabilità in grado di astrarre dalle singole tecnologie di virtualizzazione.

2.2 Le Tecnologie di Virtualizzazione nei Sistemi Basati su Cloud

Risulta evidente come tra i grandi vantaggi del modello *cloud* la sua elasticità risulti tra le sue carte vincenti. Tale elasticità è però dovuta a un opportuno uso delle tecnologie di virtualizzazione. I servizi *network-based* presenti in una piattaforma *cloud* apparentemente residenti in un server fisico in realtà vengono offerti da un da un server virtuale che risulta quindi facilmente scalabile e disponibile a seconda delle necessità. La presenza di uno strato di virtualizzazione nei modelli *cloud* permette altresì l'abbattimento dei costi su diversi fronti nonché una semplificazione rilevante a livello di gestione e manutenzione dell'infrastruttura stessa. Basti pensare che realizzare una *public cloud* disponendo di soli server fisici si tradurrebbe in un'operazione fin troppo costosa e poco conveniente, senza considerare la perdita di vantaggi e la tremenda complicazione delle operazioni che riguardano la manutenzione ordinaria ed extra-ordinaria dell'intero sistema. Si può dire che è proprio grazie alle tecnologie di virtualizzazione che i modelli *cloud* trovano un posto nel mondo non solo come modelli teorici ma anche come soluzioni convenienti nonché come opportunità di business come si è già illustrato in precedenza.

Operando direttamente sugli *hypervisor* alla base dello strato di virtualizzazione tutte le operazioni riguardanti gli strati superiori, e dunque la gestione dell'infrastruttura virtuale risultano evidentemente semplificate. Si possono allocare dinamicamente risorse hardware da destinare ad una macchina virtuale, le quali caratteristiche sono facilmente modificabili; oppure tali macchine virtuali possono essere migrate da un *hypervisor* ad un altro (per di più non necessariamente nella stessa locazione geografica) in pochi minuti e con uno sforzo relativamente nullo, specie se si dispone di una rete ad alta velocità come ad esempio la fibra ottica. Nel pensare all'infrastruttura di virtualizzazione che caratterizza un modello *cloud*, pubblico o privato che sia, risulta dunque immediato associarlo ad un *datacenter* di considerevoli dimensioni, con molte macchine fisiche (ognuna sotto il controllo di un *hypervisor*) e molto probabilmente più sedi geografiche. Un sistema di tali dimensioni al giorno d'oggi, grazie al livello attuale delle tecnologie di cui usufruisce, è in grado di gestire un numero di macchine virtuali dell'ordine del centinaio di migliaia. Inutile ribadire la quasi impossibilità di gestire un sistema equivalente composto da sole macchine fisiche; ciononostante un sistema di tali dimensioni presenta ugualmente molte problematiche e punti

fragili. La comunicazione tra il parco macchine del *datacenter* infatti, sorge come problema principale da tenere a bada. Fintanto che si resta in una situazione di omogeneità relativa alle specifiche implementazioni degli *hypervisor* (noto come caso di *vendor lock-in*) ci si trova in una situazione relativamente sicura a patto di restare sotto l'ala di un solo produttore. L'intera amministrazione del sistema avverrà tramite l'API predisposta dal *vendor* che, se sufficientemente agile e robusta, riuscirà a far fronte al problema di gestire infrastrutture di grandi dimensioni con latenze trascurabili e risultati ottimali. Tale situazione però limita anche l'infrastruttura virtuale soprastante, che presenterà esclusivamente macchine virtuali di un certo produttore e dunque, riportando tale situazione al modello *cloud*, esso rimane strettamente legato ad una determinata tecnologia di virtualizzazione.

2.3 Esempi di Sistemi Cloud

Sebbene gli origini del concetto alla base del *cloud* possano essere ricondotte ad affermazioni pubblicate in letteratura già negli anni '60 ,le tecnologie allora disponibili non rendevano certamente possibile la realizzazione dei sistemi ad oggi presenti. Come già ribadito in precedenza, grazie al progresso tecnologico su diversi campi riguardanti l'informatica, l'elettronica e le telecomunicazioni, al giorno d'oggi l'idea del *cloud* si è ben affermata nella realtà mondiale, tanto da portare i principali nomi nel business dell'IT a commercializzare soluzioni *cloud-based* a diversi livelli tra i quali quelli precedentemente elencati.

Pioniere tra tutti *Amazon.com*, con i suoi *Amazon Web Services (AWS)* già nel 2004 inizio a pubblicizzare a livello commerciale servizi per gli sviluppatori, mettendo a loro a disposizione risorse *on-demand* per poter mettere in piedi i loro progetti sul web. È stato reso noto che tali servizi sono principalmente basati su una distribuzione di *Xen* quale tecnologia di virtualizzazione, essa risulta però opportunamente modificata per adempiere alle esigenze del *Cloud Provider*. Dai documenti ufficiali relativi la descrizione dei servizi forniti, risulta che l'intera struttura alla base degli AWS appare suddivisa in 8 regioni geografiche a loro volta composte da diverse *Availability Zones* ben delimitate ed isolate (potendo così associare ad ognuna un *datacenter* in maniera univoca). In questo modo *Amazon* garantisce un trattamento dei dati conforme alle normative legali relative al territorio in questione poiché essi

restano confinati nella zona di appartenenza; in alternativa si può optare per replicare una configurazione del cliente su più zone o regioni in modo da incrementare la disponibilità del servizio a livello internazionale. Al giorno d'oggi l'offerta di *Amazon* spazia dallo *storage* con il suo *Simple Storage Service* (S3) che garantisce un'altissima disponibilità ed integrità dei dati depositati alla messa a disposizione di *virtual machines* per il deployment scalabile di applicazioni web con il servizio *Elastic Compute Cloud* (EC2). L'offerta dunque spazia da servizi *SaaS* ad *IaaS*. Quale *Cloud Provider*, *Amazon* mette a disposizione degli sviluppatori l'*AWS API* per garantire una maggiore compatibilità con altri progetti. Così facendo, risorse acquisite dagli *AWS* possono anche essere inglobati in *private clouds* con un pool di risorse eterogenee ad esempio.

Altre alternative di soluzioni *cloud-based* sono ad esempio i vari servizi erogati dalla piattaforma *Google Apps*. La nota azienda mette a disposizione svariati applicativi tipici di una *office suite* così come altre interessanti soluzioni accessibili direttamente da un qualsiasi *web browser*. *Google* offre dunque un insieme di *SaaS* in maniera gratuita ad i suoi utenti, dando la possibilità di ampliare le risorse a disposizione con piani a pagamento. Rilasciato poi nel 2008, *Google* ha ampliato il suo parco servizi con il suo *Google App Engine*, piattaforma che offre servizi *PaaS* e *IaaS* come proposta in alternativa commerciale all'EC2 di *Amazon*.

Esempi di servizi *IaaS* possono essere identificati facilmente nelle diverse soluzioni per il deployment di *private clouds* quali *Eucalyptus*, *Open Nebula* e *OpenStack* oppure restando nell'ambito del *public cloud* con le soluzioni offerte da *Windows Azure*, proposta *Microsoft* nell'ambito *cloud* che offre svariati servizi spaziando dai *SaaS* ai *PaaS*. Facendo dunque mente locale, risulta evidente come gran parte delle applicazioni che vengono usate odiernamente per adempiere alle più svariate necessità fanno affidamento in parte o interamente ad un modello *cloud-based*.

Nota di particolare interesse, specie ai fini di questo elaborato, va alla struttura dello strato di virtualizzazione alla base dell'offerta dei servizi. Non risulta sorprendente pensare che *Cloud Providers* proprietari di tecnologie di virtualizzazione utilizzino i propri mezzi per realizzare l'intero loro sistema (come avviene ad esempio nel caso di *Windows Azure*, che fa affidamento sull'*Azure Hypervisor*, versione modificata del già affermato *Hyper-V*, *hypervisor* di proprietà *Microsoft* rilasciato nel 2008 nella sua prima versione stabile). Gli stessi *vendor* di tecnologie di virtualizzazione quali *vmware*

propongono ai loro clienti tool per il deployment di intere strutture *cloud* sebbene non si propongano al mercato come *Cloud Providers*, questo a patto di restare entro i confini delle loro tecnologie. Tuttavia questo non è per forza il caso generale. Specie nel caso di *Providers* che non sono strettamente collegati con lo sviluppo di tecnologie di virtualizzazione, essi si trovano ad effettuare delle scelte progettuali al momento di pensare alla loro piattaforma *cloud*. Possono dunque decidere di seguire la strada delle tecnologie *open-source* e sviluppare delle soluzioni personalizzate che incontrano al meglio le loro esigenze oppure stipulare un contratto con aziende che garantiranno al meglio il funzionamento dello strato di virtualizzazione. In linea teorica vi è una terza via che fa fede alla struttura di sistema distribuito caratteristica dei sistemi *cloud*. Si può pensare di far coesistere più tecnologie di virtualizzazione gestite da un *middleware* in grado di governare tale situazione di eterogeneità delle risorse, rendendo del tutto trasparente all'utente finale, ma eventualmente anche all'amministratore di sistema laddove opportuno, la vera natura degli strati inferiori alla base del sistema.

Capitolo 3

Interoperabilità tra Infrastrutture di Virtualizzazione

L'intento di questo lavoro è principalmente quello di proporre un modello di gestione di infrastrutture virtuali che sia in grado di astrarre dalla specifica natura dei singoli *hypervisor*. In questo modo, sistemi e *datacenters* a larga scala (come ad esempio le piattaforme *cloud* trattate nel capitolo precedente) potranno beneficiare di una maggiore elasticità nonché di una migliore *fault tolerance*. Eventuali incompatibilità con determinate tecnologie di virtualizzazione da parte di applicativi ad esse soprastanti (per quanto limitate vista l'ideologia del concetto di virtualizzazione) potranno essere confinate a singole sezioni del sistema e gli applicativi potranno comunque venire correttamente eseguiti sui settori gestiti da altre tecnologie. Inoltre, nella realtà odierna, soggetta a costanti cambiamenti improvvisi, spesso anche sostanziali, per quanto riguarda le strutture aziendali un modello altamente elastico e tollerante in termini di tecnologie adoperate si presenta come una soluzione vincente ed economica.

Tale modello sarà caratterizzato da uno o più *datacenter* che al loro interno presenteranno un insieme di macchine fisiche e virtuali eterogeneo poiché composto da *hypervisor* di diversi produttori. Tuttavia l'intento è quello di permettere una gestione del *datacenter* il più fluida possibile, rendendo trasparente anche allo stesso amministratore del sistema, laddove opportuno, la sua natura eterogenea. Tale campo non è di certo del tutto inesplorato. Visto l'interesse crescente nel settore della virtualizzazione e l'insorgere delle problematiche nel progettare sistemi distribuiti a larga scala, non c'è da sor-

prendersi se tali problemi siano già stati posti ed in parte anche affrontati e superati. Tuttavia resta una tematica aperta a molte diverse possibili scelte progettuali che rendono difficile determinare quale sia la soluzione ottimale o addirittura stabilire uno standard operativo.

Il progetto risulta molto ambizioso poiché s'introduce un numero non indifferente di problematiche tipici della progettazione di sistemi distribuiti quali la compatibilità, la coordinazione, la latenza, la concorrenza e l'atomicità delle operazioni che dovranno senz'altro essere gestiti a dovere onde garantire il corretto funzionamento del modello.

3.1 Problematiche di Interoperabilità

Se si considera come ipotesi di avere a disposizione un insieme di macchine fisiche, ognuna con un *hypervisor* diverso, il problema di mettere in piedi un'infrastruttura virtuale funzionante che si poggi indiscriminatamente sullo strato fisico risulta, come già accennato, particolarmente complesso. Bisognerà introdurre dei meccanismi che andranno a mascherare in qualche modo la natura del sistema, aggiungendo inevitabilmente dell'*overhead* ad ogni singola richiesta e risentendone dunque a livello di performance. Diversamente dalle soluzioni collaudate e presentate dai singoli *vendor* di tecnologie di virtualizzazione, che prevedono un'infrastruttura omogenea, l'idea di una piattaforma eterogenea sarà potenzialmente meno performante ed offrirà un insieme ridotto di servizi all'utilizzatore, garantendo d'altro canto diversi altri vantaggi e la libertà di non essere legati ad una singola tecnologia.

Si può optare di stratificare il problema o quantomeno suddividerlo in singole problematiche che potranno essere affrontate in maniera più agile onde raggiungere una soluzione. Allo stesso modo le soluzioni possono prevedere approcci diversi. Si può pensare di agire "dall'alto" direttamente sullo strato virtuale oppure "dal basso", come ad esempio cercando di coordinare e monitorare gli *hypervisor* che restano sovrani dell'infrastruttura virtuale a loro sovrastante. Si può persino pensare di agire sul modello introducendo direttamente modifiche hardware al sistema, soluzione alquanto costosa sebbene sia comunque stata contemplata in ambito di ricerca poiché d'altra parte introduce ad ogni modo una grande serie di vantaggi [30].

Al fine di avere un modello pienamente operativo, vista la natura eterogenea delle risorse a disposizione sarà inevitabile dover sacrificare qualcosa

in termini di performance o servizi a disposizione. Ad ogni modo, caposaldo dell'obiettivo resta quello di minimizzare il più possibile i costi del modello sotto tutti gli aspetti. In prima battuta e in ordine sparso, le problematiche che ci si troverà ad affrontare saranno le stesse che riguardano gli ambienti propri di una sola tecnologia di virtualizzazione nonché altre problematiche aggiuntive emerse, caratteristiche dal fatto che si ha a che fare con macchine eterogenee:

- Performance/Fairness: Il modello proposto, onde essere considerato come una soluzione valida ai problemi posti deve poter garantire una certa qualità di servizio a diversi livelli. Da una parte agli utilizzatori finali dei servizi erogati, in gergo *end-users* dovrà essere garantito un livello di *performance* il più elevato possibile onde soddisfare loro aspettative, d'altro canto a livello di gestione il sistema dovrà garantire a gruppi eventualmente numerosi di amministratori di poter eseguire operazioni di gestione e monitoraggio senza interferire con l'operato degli altri colleghi. La descrizione di questo aspetto può certamente essere specificata andando ad agire sui singoli aspetti che possono riguardare le *performance* del sistema e la concorrenza delle operazioni al suo interno.

- Latenza: Quale sistema distribuito, il modello proposto fa fortemente affidamento alla rete a sua disposizione. Va da sé che onde garantire una maggiore efficienza le tecnologie alla base delle telecomunicazioni devono essere all'altezza delle aspettative. L'efficienza del modello può però essere incrementata anche agendo sul modello stesso stando ben attenti al volume di dati che comporta la comunicazione tra le macchine, virtuali o fisiche che siano, nella loro gestione ordinaria e straordinaria. Inevitabilmente però s'introduce un certo tempo di latenza tra una richiesta e la sua effettiva evasione, tanto più se si considera che la rete è fortemente soggetta a problemi di congestione ed errori. La minimizzazione di tali tempi di latenza sarà un punto di particolare interesse in fase progettuale del modello. Bisognerà quindi scegliere opportunamente i protocolli che si vogliono adoperare ai fini di implementare tutta la parte di comunicazione e ridurre al massimo l'*overhead* introdotto nelle richieste composto da meta-dati ed istruzioni.

- Coordinazione: Il modello proposto presenta un insieme elevato di componenti che coesistono nello stesso ambiente e spesso si possono trovare

ad utilizzare in maniera esclusiva o meno le risorse fisiche a disposizione. Dall'uso delle CPU alla memoria di massa, così come l'uso della rete; bisognerà implementare un opportuno modello di coordinazione che tenga conto di tutti gli scenari possibili onde garantire un funzionamento ottimale. D'altro canto, nell'erogazione dei servizi, bisogna considerare che sarà possibile e probabile avere sistemi distribuiti che faranno affidamento sul modello proposto. Sebbene in tal caso il problema di coordinazione tra le macchine virtuali sia compito degli sviluppatori del sistema distribuito un questione, un modello opportuno faciliterà la messa in piedi di sistemi complessi.

- Sicurezza: Nel progettare un ambiente virtuale sono tanti gli aspetti delicati riguardanti la sicurezza del sistema. Tra i più elementari vi è una opportuna *policy* che regoli l'interazione tra macchine virtuali. Allo stesso modo però, aspetti come la monopolizzazione di risorse fisiche da parte di una sola macchina virtuale, la sicurezza della rete e dei settori di memoria a disposizione di una istanza virtuale sono tutti aspetti da curare fin nei minimi dettagli poiché possono compromettere il corretto funzionamento del sistema nonché comportare un punto debole dello stesso agli occhi di attacchi esterni. Anche a livello amministrativo e di gestione bisogna implementare una *policy* opportuna che preveda un insieme definito di ruoli e permessi atti a garantire la corretta gestione del sistema. Non è da trascurare che una volta che si ha un sistema a larga scala l'amministrazione del sistema non sarà più, tipicamente, centralizzata ma verrà distribuita in sezioni e in alcuni casi anche stratificata.

- Disponibilità: Essendo un sistema distribuito che garantisce un accesso uniforme alle risorse, la disponibilità dei dati in ogni momento diventa un aspetto di vitale importanza. Non ci si può permettere di interrompere un servizio causa inaccessibilità ad un certo settore del sistema; esso dovrà essere progettato in modo da poter garantire interventi di manutenzione su settori delimitati senza compromettere il fluido e continuo funzionamento dell'infrastruttura virtuale. Ovviamente tale aspetto non potrà essere mai garantito nella sua interezza ma salvo casi catastrofici la corretta implementazione del sistema così come l'adoperare opportune tecnologie di accesso ed I/O possono incrementare

sensibilmente la disponibilità delle risorse così come dei servizi erogati dalle piattaforme facenti parte del sistema in questione.

- Scalabilità: Posta la *vision* del modello di poter essere esteso a larga scala, anche a livello internazionale, occorre progettare il sistema in modo "intelligente". Le soluzioni proposte dovranno essere progettate in modo di garantire gli stessi vantaggi per configurazioni delle più svariate misure, da un *datacenter* di piccole dimensioni ad un insieme di *datacenter* a scala globale. L'idea di base è non dover, laddove possibile, implementare soluzioni *ad-hoc* che tengano conto delle misure del sistema poiché tale approccio si tradurrebbe in un costo non indifferente in termini di lavoro. Risulta più vantaggioso, anche se apparentemente più complesso, progettare un sistema che goda al meglio delle caratteristiche del *"one size fits all"*.

- Compatibilità: Non è certamente trascurabile la natura delle risorse virtuali che saranno erogate dal modello. Se lo strato fisico trova come mediatore un insieme eterogeneo di *VMM* ciò si tradurrà in un insieme eterogeneo di virtual machines. Al giorno d'oggi lo stato attuale delle tecnologie di virtualizzazione prevede una situazione più o meno standard per i formati riguardanti dischi virtuali che vanno ad implementare le risorse di memoria di massa delle quali si avvalgono le singole virtual machines. Tuttavia file di configurazione che permettono il *deployment* e *management* da parte dell'*hypervisor* non sono affatto standard. Poiché ogni produttore presenta al mercato dei formati proprietari le singole tecnologie di virtualizzazione sono pensate per lavorare con un solo formato di virtual machines onde garantire la massima compatibilità ed efficienza. Inoltre anche le *API* fornite dai *vendor* restano confinate ai settori caratterizzati da una tecnologia di virtualizzazione rendendo difficile anche la gestione e il monitoraggio uniforme dell'infrastruttura virtuale.

  Per quanto riguarda il problema della compatibilità, risulta particolarmente utile notare come una soluzione posta su questo campo può automaticamente presentarsi, in parte o completamente, come soluzione anche per altre problematiche progettuali del modello. Se si è in grado di imporre un certo livello di compatibilità senza andare ad agire direttamente sul funzionamento dei singoli *hypervisor* si può delimitare il

sistema in zone che, prese autonomamente, proporranno già le soluzioni offerte dai singoli *vendor* delle diverse tecnologie di virtualizzazione adoperate per far fronte alle problematiche esposte.

Come già accennato gli approcci alla risoluzione delle suddette problematiche possono essere di diversa natura. A seguito, illustrando il formato *OVF* quale standard per le macchine virtuali si propone un modello d'azione "dall'alto", che si applica direttamente sull'infrastruttura virtuale. Diverse proposte in letteratura avanzano la possibilità di gestire le macchine virtuali direttamente sotto il formato *OVF*, si vedrà però le limitazione che ciò comporta nonché gli evidenti svantaggi. Successivamente si illustreranno diversi possibili approcci "dal basso", che si applicano coinvolgendo direttamente gli *hypervisor* e lasciando a questi ultimi la gestione dell'ambiente virtuale a loro soprastante. Questi approcci si rivelano più completi ed applicabili al caso in esame.

3.2 Il Formato OVF

Essendo la virtualizzazione argomento di molto interesse già da più di un decennio, si è ormai giunti ad alcune soluzioni riguardanti la problematica della compatibilità tra tecnologie eterogenee. Dal punto di vista di questo elaborato esse si presentano però come soluzioni parziali. La DMTF (*Distributed Management Task Force*) quale consorzio industriale che include i principali sviluppatori delle attuali tecnologie di virtualizzazione ha elaborato le specifiche per un formato standard riguardante le macchine virtuali al giorno d'oggi ormai ampiamente usato in ambito corporativo. Tuttavia tale formato presenta una serie non indifferente di limitazioni poste a garantire la sua natura di standard che ai fini di questo progetto si rivelano non indifferenti al momento di attuare scelte progettuali. Tale standard resterà però ugualmente utile entro i limiti che esso impone e proporrà soluzioni parziali o comunque alcune semplificazioni per quanto riguarda dei meccanismi che possono rivelarsi molto utili ai fini della compatibilità tra piattaforme.

Stando ai documenti ufficiali relativi al formato, lo scopo dell'*OVF* (*Open Virtualization Format*) è quello di descrivere un formato aperto, sicuro, portatile, efficiente ed estendibile per il *packaging* e la distribuzione di software destinato a venire eseguito su una macchina virtuale. Nello specifico il formato prevede un *XML descriptor* (`.ovf`) che indichi alcune informazioni

indispensabili per capire la natura della macchina virtuale (ovvero la sua configurazione delle risorse virtuali che le devono essere provviste), un file *manifest* (`.mf`) ed un file *certificate* (`.cert`) per garantire l'integrità e sicurezza della macchina. Tali file verranno poi accompagnati da uno o più file immagine contenenti il disco rigido (virtuale) della macchina virtuale. Il tutto può eventualmente venire rilasciato in un unico contenitore che prende il nome di *package OVA (Open Virtual Appliance)*. Starà poi agli *Hypervisor* essere provvisti di meccanismi per interpretare opportunamente le informazioni sul *descriptor*, eseguire i controlli sui file *manifest* e *certificate* e, grazie alle immagini disco, mettere in piedi una (o più) macchine virtuali funzionanti e pronte all'uso.

Il formato si presenta particolarmente utile per gli sviluppatori di applicazioni complesse che prevedono configurazioni hardware specifiche, oppure sistemi distribuiti di più macchine. Grazie al formato *OVF* il processo di sviluppo può essere fatto un volta sola e, grazie allo standard, il *deploy* dell'applicazione può avvenire in qualsivoglia ambiente di virtualizzazione. Se si considera il ciclo di vita di una *virtual appliance* lo scopo della specifica *OVF*, così come intesa dagli sviluppatori copre le fasi di distribuzione e *deploy*. Particolare di non poca importanza è proprio l'esclusione della fase di

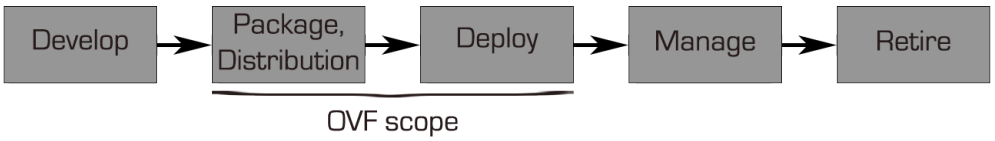

Figura 3.1: Ciclo di vita di una *Virtual Appliance*

management dallo scopo del formato. Di fatto, se si considera quanto detto finora, ogni *hypervisor* provvederà ad interpretare i descrittori ed allocare le risorse per la macchina virtuale. In realtà l'operazione che compie il *VMM* si traduce nel creare una nuova macchina virtuale (con il relativo formato proprietario) basandosi sulle specifiche del *ovf descriptor* ed associarvi copie delle immagini disco allegate per dar luogo ad una macchina già configurata anche a livello software. Bisogna tenere ben presente che secondo la specifica *OVF*, qualsiasi manipolazione dei file componenti la *OVA* si tradurrebbe nell'invalidità del certificato e di conseguenza nella decadenza dallo status di

OVF, in tal caso alla macchina risultante non verrebbe più garantita alcuna portabilità.

Lo standard *OVF* è stato progettato per ovviare al problema della portabilità di macchine virtuali tra diverse tecnologie di virtualizzazione ma le scelte fatte hanno garantito un certo livello di compatibilità senza andare a compromettere i servizi e metodi che i singoli *hypervisor* offrono ai loro formati proprietari poiché in fin dei conti si presenta come formato atto al trasferimento e distribuzione, non all'esecuzione, di macchine virtuali attraverso diverse piattaforme di virtualizzazione.

Nell'ottica della nostra ipotesi di un sistema eterogeneo, vista la relativa semplicità nell'operazione di creare una *OVA* e la possibilità di automatizzare tale procedura, si può ricorrere al formato per implementare procedure di migrazione di macchine virtuali tra diversi *VMM*. Tale migrazione avverrebbe in 4 passaggi:

1. Creazione: l'*hypervisor* host procede a creare un package *OVA* con le specifiche della macchina virtuale che si intende migrare al quale allegherà le sue immagini disco.

2. Trasferimento: il package *OVA* verrà inviato al suo nuovo host con indicazioni per l'immediata installazione.

3. Deploy: una volta ricevuto il package, il nuovo host provvederà ad interpretare i descrittori ed eseguire i controlli di dovere per mettere in piedi una nuova macchina virtuale con le specifiche indicate e successivamente comunicherà al host precedente il corretto adempimento del suo dovere.

4. Eliminazione: il host originario, ricevuta conferma dell'avvenuta installazione, procederà ad eliminare la macchina virtuale migrata. A questo punto anche il package *OVA* può essere dismesso.

Questa procedura presenta però forti limitazioni poiché tutte e 4 le operazioni devono venire compiute in maniera atomica per garantire l'integrità del sistema, eventuali meccanismi di migrazione "a caldo" ovvero meccanismi che considerano il caso in cui la macchina da migrare sia accesa e funzionante (limitandone il *downtime* e garantendo il corretto funzionamento del sistema a migrazione ultimata, rendendo del tutto trasparente l'intera operazione) dovranno essere strutturati in una maniera più complessa.

Sebbene nell'ottica dell'interoperabilità tra piattaforme di virtualizzazione sia stato proposto un modello che preveda l'uso esclusivo di macchine *OVF* e in ambito *cloud* intere infrastrutture che adoperino tale formato come unico formato in uso, risulta evidente che ciò va contro la definizione stessa di *OVF* [18] [34], portando a casi fortemente soggetti ad errori in fase d'esecuzione. Si darebbe luogo ad una piattaforma poco affidabile ed appetibile. Per ovviare, almeno in parte, a tali problemi, bisognerebbe implementare da zero un *hypervisor ad-hoc* che operi opportunamente con il formato *OVF* senza compromettere l'integrità dei contenitori. In questo caso però si ricadrebbe in un modello omogeneo poiché il *VMM* implementato dovrebbe coprire l'intera infrastruttura.

Risulta quindi evidente che approcciare il problema dell'interoperabilità "dall'alto", e dunque operando esclusivamente sullo strato virtuale porta a non poche limitazioni, risultando per giunta un'operazione anche molto complessa e laboriosa.

3.3 Modelli per l'Interoperabilità di Sistemi Basati su Tecnologie di Virtualizzazione

A seguito di un approfondito lavoro di ricerca riguardo le diverse proposte in ambito di interoperabilità di piattaforme di virtualizzazione si è visto come, al fine di gestire infrastrutture di grandi dimensioni, vengano esposti i più svariati approcci per proporre dei modelli di coordinazione e gestione che prevedano la possibilità di un parco macchine con *hypervisor* eterogenei. Studiate le diverse proposte in letteratura sotto diversi aspetti ed integrando lavori e documentazione in vari ambiti correlati si è visto però come alcuni dei modelli proposti siano giustamente preferibili ad altri in termini di costi e performance. In particolare quello presentato, visti i metodi che adopera ed i *pattern* che applica sembra essere la soluzione più appetibile, meno costosa e più facilmente realizzabile [17].

Sebbene sia stata intrapresa dalla DMTF la strada della definizione di uno standard che espone come gli oggetti devono essere rappresentati e i modi in cui devono interagire tra di loro, tale sforzo si è rivelato prevalentemente vano poiché le diverse implementazioni dello standard da parte delle aziende si sono rivelate ad ogni modo incompatibili tra di loro [16]. Al giorno d'oggi non è ancora presente uno standard che soddisfi le necessità che si presentano

negli scenari eterogenei ai fini di garantire l'interoperabilità. Tuttavia, il modello proposto si presenta come un ottimo punto di partenza per studiare e definire uno standard vero e proprio che ponga una soluzione definitiva alle problematiche precedentemente presentate.

In generale gran parte degli approcci proposti in letteratura hanno tra di loro un punto in comune e cioè, la comunicazione tra le macchine che, ovviamente, deve avvenire in modo omogeneo e comprensibile ad ogni host. Inoltre, al fine di gestire l'infrastruttura, bisogna disporre uno strato di controllo più o meno distribuito sulla base delle dimensioni del sistema. Ottimo esempio in questo caso risulta la suddivisione in regioni e zone che propone *Amazon.com* con i suoi servizi *cloud*. Le singole sezioni non sono tuttavia del tutto autonome, devono poter anch'esse comunicare in maniera efficace tra di loro, evidenziando ancora una volta l'importanza di un meccanismo di comunicazione omogeneo. A supporto di questo "linguaggio comune" si può pensare a come sono collegate le macchine e cioè alla rete. In prima battuta risulterebbe conveniente fare ricorso ai protocolli `TCP/IP` per impacchettare istruzioni comprensibili agli *hypervisor* e cioè le loro *API*. Tuttavia, onde progettare un ambiente nel quale sia possibile gestire in maniera del tutto trasparente diverse tipologie di macchine bisogna tenere in conto le differenze tra i modelli d'interazione previsti dai singoli *hypervisor* e le loro relative API. Bisogna considerare che si punta a poter gestire da remoto le singole macchine fisiche dell'infrastruttura e, nel farlo, si necessità di conoscere e supportare tutte le modalità d'accesso richieste dalle singole tecnologie di virtualizzazione. Il problema assume immediatamente una scala maggiore dal semplice *wrapping* delle istruzioni.

Ricordando che oltre ai problemi di compatibilità ci si trova ad affrontare un elenco non indifferente di altre problematiche, può risultare utile prendere spunto dalle soluzioni già esistenti per la gestione di infrastrutture che prevedono più *cluster* di *datacenter* omogenei. Si consideri ad esempio la suite di strumenti e tecnologie di virtualizzazione messi a disposizione da *vmware* [31]. Le soluzioni *hardware-assisted* della gamma *ESXi* e *vSphere* sono al giorno d'oggi le più utilizzate a livello corporativo per implementare infrastrutture virtuali basate su *hypervisor Type 1* grazie alla loro affidabilità e ad una serie di servizi particolarmente utili quali la migrazione "a caldo" o la possibilità di ridimensionare le risorse virtuali dedicate a una *virtual machine* mentre questa è in funzionamento.

A livello di gestione, *vmware* mette a disposizione il *vCenter server*, un

insieme di processi e servizi per monitorare i singoli host fisici e i loro ambienti virtuali, così come un insieme di agenti residenti nelle macchine da monitorare. Il *vCenter* viene rilasciato sotto forma di macchina virtuale a sua volta ed è in grado non solo di monitorare ma anche di compiere operazioni più o meno elementari sui singoli host o macchine virtuali grazie all'uso della *vShpere API*. La gestione dell'infrastruttura virtuale avviene dunque dialogando direttamente con i singoli *VMM* tramite le loro API e nell'insieme viene incapsulato come un *layer* a sé stante. Inoltre, è progettato in modo tale da essere ampiamente scalabile e cioè, è in grado di gestire senza difficoltà configurazioni di poche o molte istanze virtuali. Nel caso di infrastrutture dislocate geograficamente o comunque di grandi dimensioni, risulta conveniente suddividerle in diversi *cluster*, che saranno composti da un insieme di host fisici monitorati da un singolo *vCenter server*. In tal caso però, i *vCenter server* di ogni *cluster* lavorano tra di loro in *linked mode*, garantendo principalmente 2 vantaggi: anzitutto, accedendo alla *User Interface* (UI) di un singolo *vCenter server* si ha comunque una panoramica dell'intera infrastruttura, inoltre, i ruoli degli utenti dell'intero sistema sono sincronizzati tra tutti i *server* via `Lightweight Directory Access Protocol` (LDAP) in modo tale da assicurare che i permessi di un utente siano riconosciuti non soltanto nel suo *cluster* di appartenenza ma nell'intero sistema. In questo modo si garantisce ad ogni istante uno stato consistente della *security policy* imposta dagli amministratori del sistema.

A discapito del fatto che il sistema in questione sia omogeneo o eterogeneo a livello di infrastrutture di virtualizzazione, la sua gestione dovrà essere dedicata ad un *layer* a sé stante, scalabile e decentralizzato, in grado di dialogare con tutti i *VMM* che compongono lo strato di virtualizzazione. La gestione del sistema dovrà garantire ad ogni istante la consistenza del sistema stesso, prevedere meccanismi di accesso sicuri alle macchine ed una struttura gerarchica dei ruoli di utenti ed amministratori. Inoltre, dovrà supportare l'accesso concorrente a funzionalità di monitoraggio delle risorse virtuali introducendo il minimo ritardo possibile e garantendo l'atomicità delle operazioni. Tutto questo dovrà essere fatto con l'ottica di minimizzare i costi in termini di *performance* inevitabilmente introdotti, per fare ciò, sarà di vitale importanza attingere ai principi principali dell'ingegneria dei sistemi software e far ricorso ai *design patterns* più adatti per i singoli componenti del sistema. In figura 3.2 viene dunque esplicitato in prima battuta l'inserimento di un livello di controllo che gestisca le politiche d'interoperabilità.

Tuttavia, poiché tali politiche riguardano diversi aspetti ben identificabili e trattabili separatamente, il livello introdotto si realizza in 3 sezioni ben distinte introdotte di seguito. Il modello vero e proprio, che esplicita i diversi meccanismi che compongono le tecnologie di interoperabilità viene illustrato quindi successivamente in figura 3.3.

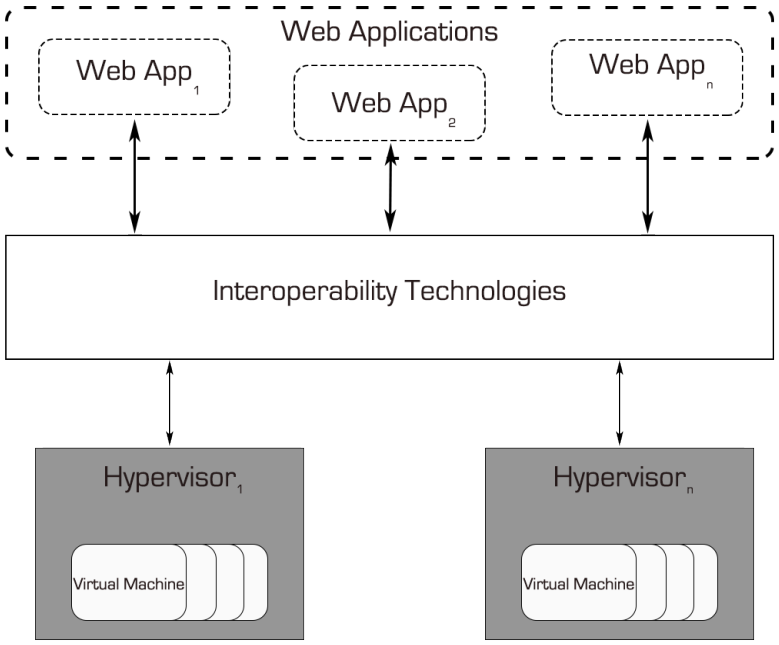

Figura 3.2: Livello per la gestione delle politiche d'interoperabilità

Considerato ora lo scenario d'interesse, ovvero quello di un'infrastruttura di virtualizzazione eterogenea, risulta evidente che per i *client* che accedono alle risorse virtuali del sistema, la natura di quest'ultimo dovrà essere del tutto trasparente. Il sistema dovrà predisporre un'opportuna interfaccia d'accesso alla gestione e monitoraggio delle risorse che renda possibile la suddetta trasparenza. Allo stesso modo, tale interfaccia, poiché a conoscenza della struttura del sistema, dovrà occuparsi di ogni richiesta d'interazione con le risorse proveniente dall'esterno. In sistema conterà un componente di gestione noto come *Virtual Resources Management System* (VRMS) che s'interfaccerà con l'esterno.

Internamente resta da gestire l'aspetto di comunicazione e quello di accesso diretto alle risorse tramite il *VMM*. Per minimizzare i tempi di latenza introdotti dalla richiesta di informazioni e l'effettiva loro evasione, risulta conveniente adoperare un *middleware* atto alla gestione di comunicazioni *pseudo real-time* in ambito distribuito noto come *Data Distribution Service* (DDS). Tale *middleware* si avvale del *pattern Publish/Subscribe* per garantire una quasi immediata evasione delle richieste d'informazione. In alternativa, per ovviare al problema della comunicazione, si può optare per altri tipi di architetture tipiche dei sistemi distribuiti come i sistemi RESTful (*REpresentational State Transfer*) oppure protocolli d'accesso quali SOAP (*Simple Object Access Protocol*), tuttavia, è stato dimostrato in letteratura come le tecnologie che implementano il modello di DDS siano molto più performanti rispetto alle alternative esposte, sia in termini di uso delle risorse di computazione che di scalabilità del sistema e tempi d'attesa [12].

Infine resta il problema d'accesso. Si dovrà disporre di un'interfaccia omogenea d'accesso ai singoli *hypervisor* in grado di dialogare con essi in un linguaggio a loro comprensibile (indubbiamente le loro API) ma altresì con il VRMS, che a tale interfaccia delegherà le istruzioni da eseguire sulle singole macchine in maniera opportuna. Il meccanismo di accesso previsto dovrà "mediare" tra il VRMS e i singoli *hypervisor* per l'acquisizione di informazioni e l'esecuzione di operazioni semplici. Tali compiti restano tuttavia di dovere dell'*hypervisor* in questione. In questo modo si garantisce la *QoS* (*Quality of Service*) già offerta dalle singole tecnologie di virtualizzazione e si snellisce la complessità del sistema in fase d'implementazione. Nel modello proposto si è pensato di agire su due fronti, predisponendo un'interfaccia a due livelli. Dal lato del VRMS, esso comunicherà con un livello di astrazione per la mediazione con i *VMM* mentre dal lato degli *hypervisor*, ad ogni macchina verrà associato un mediatore a conoscenza della tecnologia con la quale ha a che fare, in modo tale che sia configurato per adoperare l'API opportuna. Si definiscono dunque due entità: l'*Hypervisor Resource Broker* (HRB) e l'*Abstract Layer for Hypervisor Resource Broker* (A-HRB).

Il modello proposto, sebbene consigli determinati metodi, specie per l'accesso alle risorse e la gestione della comunicazione, resta comunque del tutto svincolato da specifiche implementazioni di tecnologie. Tuttavia, risulta evidente come alcuni progetti, specie *open-source* si prestino molto bene ad assumere un ruolo all'interno del sistema. Implementare interamente un sistema del genere in ogni suo aspetto si rivelerebbe un compito alquanto

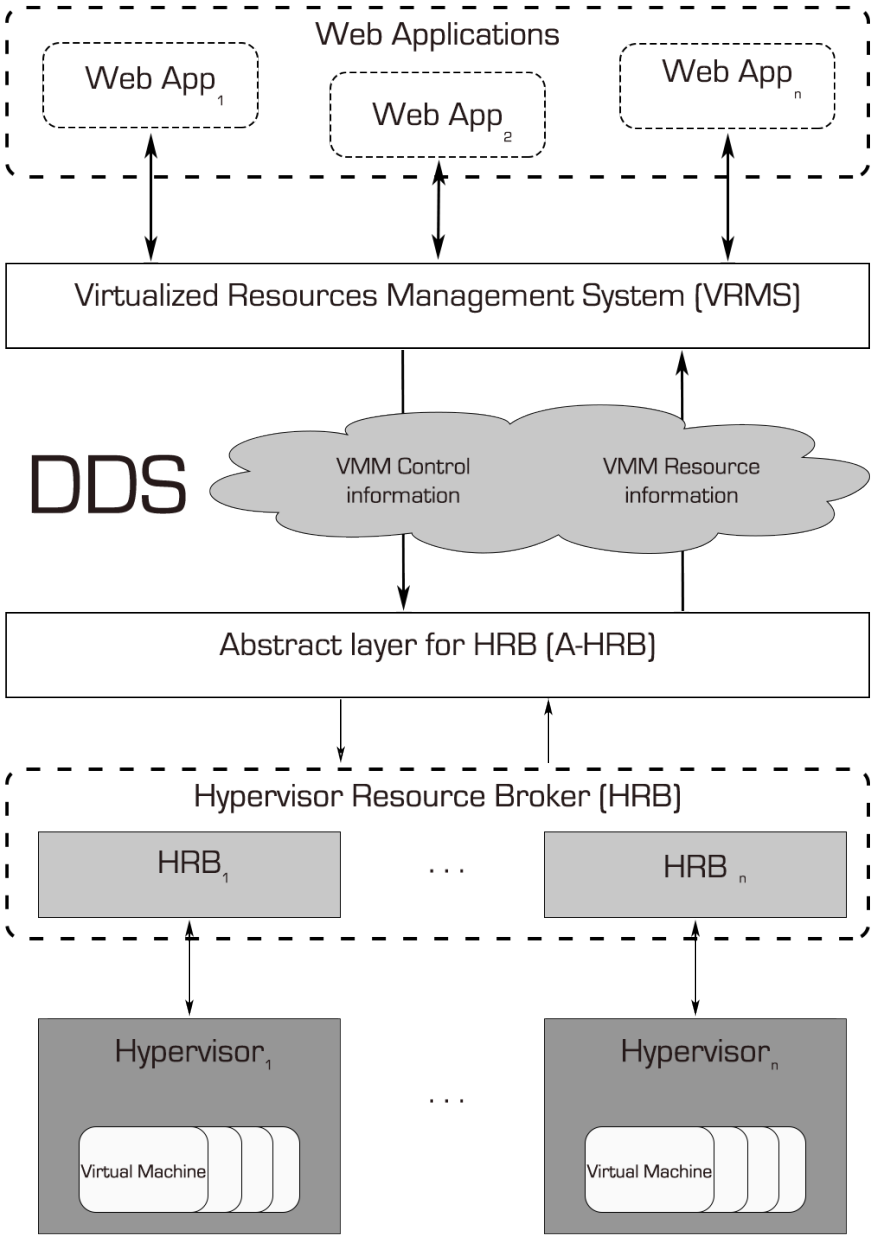

Figura 3.3: Modello per un sistema di risorse virtuali eterogenee

costoso nonché ambizioso. Conviene dunque fare affidamento a tecnologie o progetti già esistenti e in continua evoluzione, onde garantire anche una certa retro-compatibilità e stabilità negli anni a venire. Sebbene VRMS, DDS e (A-)HRB siano componenti a sé stanti del sistema, essi sono legati l'un l'altro dalle particolari scelte in fase di progettazione per quanto riguarda le tecnologie che gli implementano. Il VRMS, essendo un'interfaccia, risulta facilmente implementabile, essa dovrà fornire tutte le funzionalità possibili garantite dai metodi d'accesso ed implementare una *security policy* che tenga conto dei ruoli dell'utente che vi accede.

Per quanto riguarda l'aspetto della comunicazione, VRMS e A-HRB sono registrati nel DDS come *publisher & subscriber* l'uno dell'altro, in questo modo si garantisce una comunicazione bilaterale tra le due entità. L'implementazione dello standard DDS può essere scelta tra le diverse proposte nel mercato così come nelle soluzioni *open-source* quali *OpenDDS*. Tra le soluzioni disponibili vengono proposte svariate API in diversi linguaggi quali C, C++, C#, Java, Ruby, ed altri ancora, rendendo lo standard DDS come un'ottima scelta anche a livello di compatibilità.

Infine, il metodo di accesso alle risorse dovrà garantire l'effettiva evasione delle richieste fatte dal VRMS. Questo risulta un aspetto critico poiché si parla del dialogo con l'*hypervisor* che, a seconda della tecnologia di virtualizzazione, avviene in maniere diverse, con API diverse e tramite protocolli di comunicazione diversi, a volte anche proprietari. Appare dunque come lo scoglio principale da superare per ovviare al problema della compatibilità. Anzitutto, poiché si dovrà garantire sempre e comunque l'evasione delle richieste fatte dal VRMS, il sistema dovrà offrire un insieme di funzionalità comune a tutti gli *hypervisor* e dunque l'insieme d'intersezione tra tutte le API delle tecnologie di virtualizzazione disposizione. In alternativa bisognerebbe tenere conto ad ogni momento dell'identità dell'*hypervisor* con il quale si vuole stabilire una qualche forma d'interazione e dunque limitare la trasparenza del sistema al VRMS. In questo modo si potrebbe anche usufruire di funzionalità esclusive a singoli *hypervisor* ma ovviamente soltanto richiamandole sui componenti del sistema che le supportano. L'intera infrastruttura andrebbe ad essere sezionata in zone caratterizzate da una particolare tecnologia di virtualizzazione e relativo supporto alle loro API. Si è già illustrato in precedenza come anche questa possa essere comunque una soluzione contemplata poiché ne trae evidenti vantaggi di compatibilità con le applicazioni che si affidano alle risorse virtuali del sistema, specie quelle di gestione o che

prevedono ad esempio la migrazione di macchine virtuali, operazione critica se si considera il problema della compatibilità tra macchine virtuali e *VMM*. Ad ogni modo, la scelta tra gli approcci presentati si rifletterà nel modo in cui il VRMS mette a disposizione le funzionalità d'accesso alle risorse e la loro eventuale disponibilità limitata e necessità d'indirizzare esplicitamente le richieste a determinati host fisici.

3.4 API per l'Interoperabilità tra Piattaforme di Virtualizzazione: libvirt

Si è visto come l'accesso alle risorse di virtualizzazione sia un punto critico del modello presentato. Onde implementare il livello di mediazione proposto con tecnologie già esistenti, occorre scegliere l'opzione che preveda una maggiore compatibilità e migliore stabilità nonché *long-term support*. *libvirt* è un progetto *open-source* di *Red Hat, Inc.* interamente sviluppato in C che vanta una *community* di sviluppatori molto attiva, e una vasta collaborazione con i principali nomi in ambito di virtualizzazione. La libreria si pone come obiettivo quello di implementare i meccanismi per la gestione e monitoraggio, possibilmente tramite accesso remoto, di risorse virtuali residenti in una macchina fisica in modo sicuro, consistente e stabile senza preoccuparsi del *VMM* installato nella macchina. *libvirt* si presenta come un'API "universale" per la gestione di macchine virtuali e implementa dunque un *layer* d'astrazione per l'accesso alle risorse che garantisce un certo livello d'interoperabilità in un ambiente caratterizzato da risorse eterogenee. Rispetto al modello presentato, si presta come un'ottima tecnologia per realizzare i livelli di A-HRB e HRB.

Visto l'interesse posto nella questione dell'interoperabilità. *libvirt* è un progetto che nei suoi ormai quasi 10 anni di vita ha visto una veloce evoluzione in termini di funzionalità offerte e tecnologie supportate. Allo stato attuale, *libvirt* risulta compatibile con le principali tecnologie di virtualizzazione ed è supportata dalle principali soluzioni in ambito di gestione delle risorse. Anche se *libvirt* si presenta come una libreria C, essa gode di una vasta serie di *bindings* con altri linguaggi di programmazione quali Python, Perl, Ruby, Java, PHP ed altri, rendendola una soluzione ampiamente compatibile anche con le eventuali scelte di progettazione che possono essere fatte per

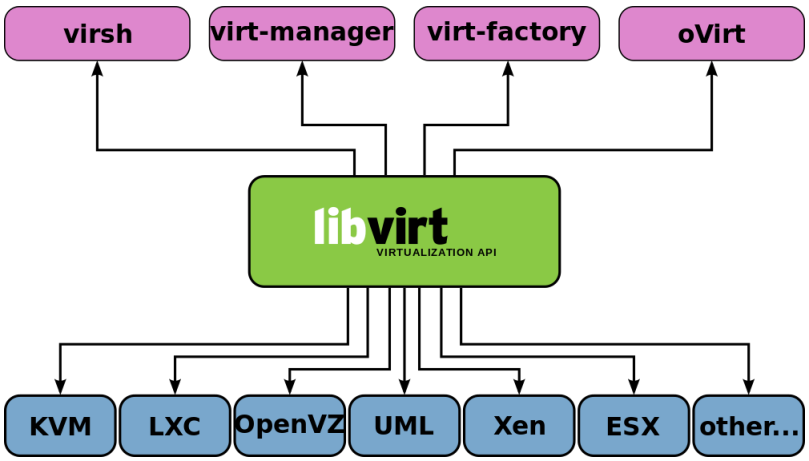

Figura 3.4: Compatibilità della libreria *libvirt* con diverse tecnologie

quanto riguarda il livello di gestione, ovvero, nel caso del presente modello, il VRMS.

Rispetto alle tecnologie di virtualizzazione invece, *libvirt* si presta ad uno sviluppo molto libero poiché, sebbene non tutti gli *hypervisor* in commercio supportino le stesse operazioni, se una funzionalità è utile per la gestione di uno specifico tipo di macchina virtuale o *hypervisor*, l'intento è quello di inserire tale opzione nella libreria. *libvirt* prevede nel suo modello architetturale, una serie di *driver* che si occuperanno di mediare le richieste con i diversi tipi di *hypervisor* rivolgendosi a loro nei metodi previsti dalle loro API e tramite i protocolli dedicati alla comunicazione delle stesse. I singoli *driver* sono dunque a conoscenza della natura dell'*hypervisor* e si occupano di compiere un'operazione di *traduzione* "intelligente" delle richieste fatte loro tramite le API standard previste da *libvirt* in quelle che sono le API previste dalla singola specifica tecnologia di virtualizzazione. Risulta evidente come si possa implementare tramite i *driver* di *libvirt* il concetto di HRB mentre con l'API fornita da *libvirt* il livello A-HRB.

Poiché tra gli obiettivi di *libvirt* risulta quello di poter gestire le risorse virtuali da remoto, la libreria prevede dei meccanismi di accesso tramite meccanismi e protocolli sicuri. Ad ogni modo, un'interazione tra un gestore ed un *hypervisor* avviene anzitutto stabilendo una connessione tramite un puntatore `virConnectPtr`. Stabilita la connessione, tramite l'API di *libvirt* si può accedere a diversi aspetti dell'*hypervisor* come la rete

(tramite `virNetworkPtr`), lo *storage* (con i puntatori `virStoragePoolPtr` e `virStorageVolPtr`) e le macchine virtuali sotto il controllo del *hypervisor* (via `virDomainPtr`). Una volta che si ha una connessione stabile con

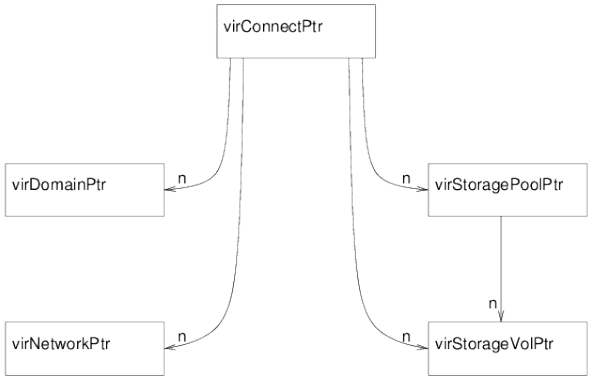

Figura 3.5: Puntatori per l'interrogazione delle risorse remote

l'*hypervisor* in questione, l'API `virInitialize` assegna uno specifico *driver* al *VMM* che è in grado di comunicare con esso nei metodi previsti da quest'ultimo. In prima battuta si può dire che l'assegnazione di un *driver* avviene in maniera dinamica selezionando da un insieme di possibilità la soluzione più opportuna. I componenti che prendono il nome di *driver* e dunque i mediatori, non si limitano però ad essere associati ad *hypervisors* e quindi a sole tecnologie di virtualizzazione. Sono previsti anche *driver* per gestire aspetti di rete o *storage* come l'accesso a NAS o dischi iSCSI, in questo modo *libvirt* ampia le sue possibilità presentandosi come una soluzione elastica per la gestione di risorse in svariati casi e configurazioni.

L'accesso remoto si realizza però tramite un *pattern Client/Server* che, per come è progettato, garantisce una maggiore stabilità di un sistema in continuo cambiamento e una migliore *failure recovery* poiché disaccoppia le entità che interagiscono. Grazie ai *Daemon* previsti da *libvirt* (`libvirtd`),

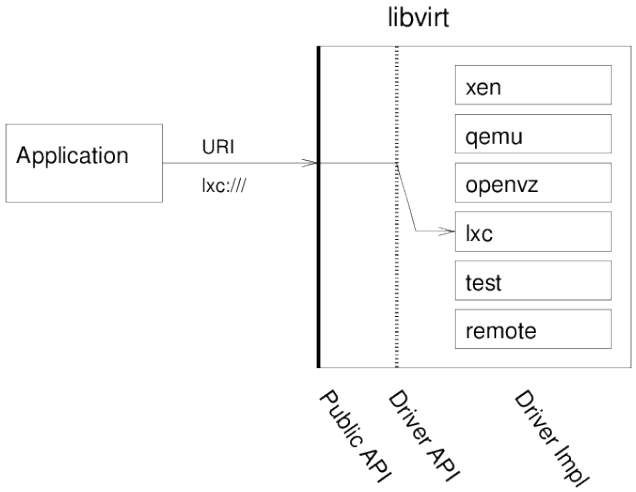

Figura 3.6: *libvirt* driver architecture

che gestiscono l'accesso ai *driver* (e dunque l'interlocuzione con le risorse), l'interrogazione di una risorsa avviene in due passaggi grazie al concetto di `RPC` (*Remote Procedure Call*). Nell'applicazione di gestione (che adopera l'API *libvirt* e che risiede nel *Client*) viene inizializzato un *driver* per gestire connessioni remote, esso si collegherà al *Daemon* `libvirtd` che risiede nella macchina *Server*, il quale gestirà ogni chiamata fatta dal *Client* reindirizzandola al *driver* opportuno. Caratteristica particolare del *libvirtd*, infatti, è che si avvia con la macchina *host* e inizializza tutti i *driver* occorrenti per interrogare le risorse (fisiche e virtuali) della macchina. Essendo un processo, esso può essere configurato opportunamente e riavviato a *runtime*, aspetto che realizza un'agile *failure recovery* del sistema. La struttura che caratterizza la libreria *libvirt* evidenzia come sia necessario che essa vada al passo coi tempi delle tecnologie di virtualizzazione poiché sorge la necessità di supportare al meglio le diverse proposte del mercato. Tuttavia, tale lavoro risulta più complicato di quel che possa sembrare. Data la natura della libreria si incontrano non poche difficoltà nello sviluppo di nuove funzionalità, prima tra tutte ad esempio, il fatto che la libreria sia interamente implementata in C, e

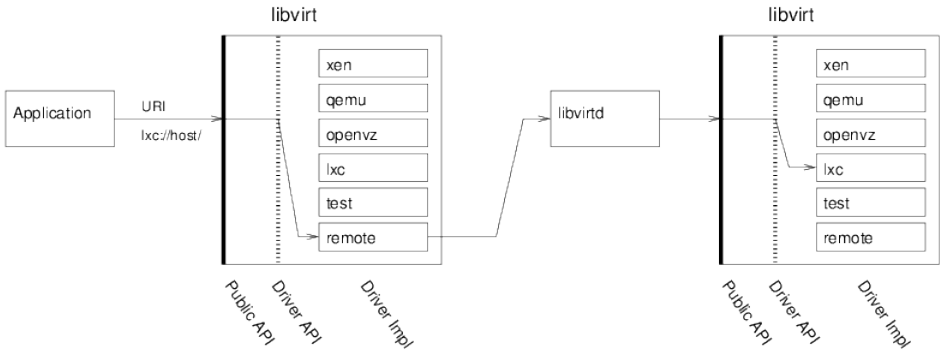

Figura 3.7: *libvirt daemon* architecture

quindi un linguaggio imperativo, limita non indifferentemente la possibilità di scegliere un approccio *Object-Oriented* che in molti casi semplificherebbe molte problematiche. Inoltre, riuscire a garantire la compatibilità con tutti gli *hypervisor* non è poi così scontato. Conflitti nella rappresentazione delle informazioni sono problemi molto ricorrenti e a volte di difficile soluzione, inoltre, le ultime versioni dei *VMM* tendono a presentarsi sempre di più come sistemi chiusi, limitando la possibilità di installare applicativi da terzi (come sarebbe il caso del `libvirtd`) e richiedendo dunque soluzioni che contemplano persino modifiche del *kernel* del sistema operativo sul quale si basa l'*hypervisor*. Infine, l'uso di tecnologie e protocolli proprietari da parte dei *vendor*, come risulta ad esempio il caso di *Hyper-V* della *Microsoft*, accessibile soltanto tramite la *Windows Management Instrumentation* (`WMI`), limita la possibilità di implementazione dei *driver*. Ad ogni modo, il progetto è in costante sviluppo e si concentra principalmente sulla compatibilità con gli stantard DMTF come i modelli CIM (*Common Information Model*). Si

giunge quindi ad un compromesso di compatibilità che si dovrà tenere sempre in conto in quanto limita le funzionalità che il sistema eterogeneo può offrire ai suoi utilizzatori rispetto ad una soluzione che adoperi una sola tecnologia di virtualizzazione.

Capitolo 4

Casi di Studio

4.1 Introduzione

Si è presentato e discusso un possibile modello per l'interoperabilità di risorse virtuali nonché le eventuali possibili tecnologie che potrebbero essere adoperate per implementare le singole parti del modello, analizzandone i vantaggi e gli svantaggi. Al giorno d'oggi però, il problema della gestione di risorse eterogenee risulta (se non certamente superato) affrontato su larga scala. Esistono soluzioni *open-source* e a pagamento che implementando modelli simili a quello presentato fanno fronte al problema di gestire un *pool* di macchine fisiche eterogenee. È possibile altresì trovare in letteratura soluzioni con un approccio radicalmente diverso come ad esempio sistemi *meta-hypervisor* in grado di gestire a livello hardware la coesistenza di più *hypervisor* nella stessa macchina fisica [30]. Punto in comune tra gran parte delle applicazioni sinora sviluppate risulta l'uso dell'API *libvirt* o del modello da essa proposto, come metodo di accesso alle risorse.

Poiché le infrastrutture *cloud-based* sono state presentate come potenziali beneficiarie di un meccanismo di accesso a risorse di virtualizzazione eterogenee, il presente capitolo si propone l'obiettivo di analizzare una delle soluzioni più usate nel *deployment* di *private clouds* a livello corporativo e cioè il sistema operativo *OpenStack*. Risulta noto per la piattaforma studiata che in molti casi essa si avvale della libreria *libvirt* per l'accesso agli *hypervisor* componenti l'infrastruttura mentre in altri sviluppa i suoi propri *driver* sulla base del modello proposto dalla suddetta libreria.

Lo studio della piattaforma *OpenStack* risulta particolarmente interes-

sante ai fini del presente studio poiché in prima battuta si basa sul modello presentato nella sezione 3.3 di questo elaborato. In realtà presenta sottili differenze nelle scelte dei meccanismi interni di comunicazione, infatti, *OpenStack* si avvale principalmente di API RESTful per le comunicazioni tra i suoi moduli [26] [27] [28] malgrado le perdite a livello di performance complessiva. Inoltre, *OpenStack* mira a gestire le interazioni con gli *hypervisor* senza dover installare componenti sulle macchine fisiche, laddove possibile.

4.2 Private Cloud Basato su Piattaforma OpenStack

OpenStack si presenta come un sistema operativo "distribuito" che rende possibile l'implementazione di strutture *cloud* private facendo una serie di assunzioni e semplificazioni rispetto i livelli inferiori dell'architettura. Esso si pone come obiettivo fornire tre tipi principali di risorse agli *end-user* che ne usufruiranno per sviluppare le loro applicazioni o adempiere alle loro necessità senza doversi preoccupare dell'effettiva locazione delle risorse fisiche che vengono accedute, ideale alla base del concetto di *cloud computing*. Tramite una serie di API, *OpenStack* offre al consumatore risorse di computazione (*compute*), rete (*networking*) e memorizzazione (*storage*).

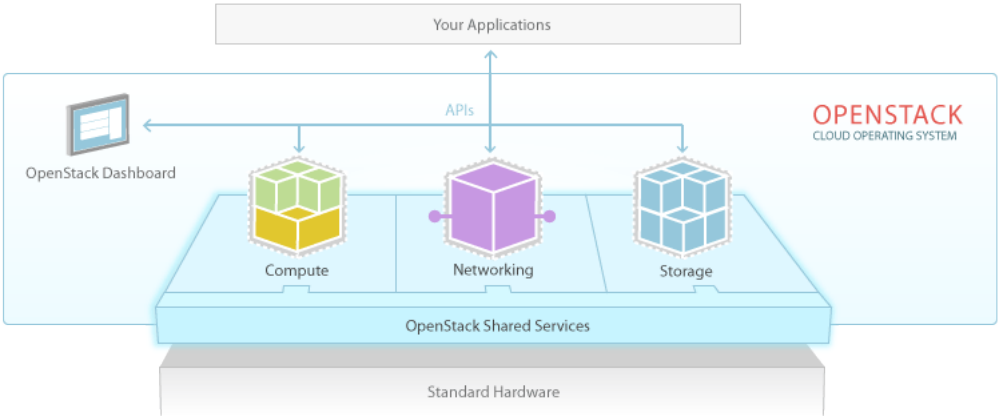

Figura 4.1: *OpenStack* Overview

La figura 4.1 dimostra come agli utilizzatori della piattaforma risulta semplificata la sua architettura, che in realtà presenta diversi moduli in inte-

razione complessa tra di loro. Infatti, una visione completa della piattaforma, che include le diverse interazioni tra i moduli ed i loro singoli componenti può essere vista in figura 4.5.

L'idea alla base di *OpenStack* è quella di fornire tutti i meccanismi necessari alla realizzazione di progetti più o meno complessi tramite i tre elementi sopra elencati, fornendo così una piattaforma *IaaS*. I progetti in essa presente saranno poi composti da una o più macchine virtuali, ma l'aspetto di accesso alle risorse fisiche resterà del tutto trasparente sia all'*end-user* che al *service provider*.

OpenStack offre a supporto del suo modello un insieme di altre funzionalità che coprono i diversi aspetti da gestire in un ambiente *cloud*. Moduli ben delimitati ed indipendenti del sistema operativo si occupano di implementare le politiche di sicurezza, *dashboard* per la gestione delle risorse via *client web* o tramite CLI (*Command Line Interface*) e meccanismi utili al *management* dell'intero ciclo di vita di una macchina virtuale all'interno della piattaforma, specie in fase d'inizializzazione e dismissione.

4.2.1 Panoramica

Il progetto *OpenStack* è un'iniziativa nata nel 2010 dalla collaborazione di *Rackspace Inc.* e la *NASA* per implementare un meccanismo di gestione delle risorse in ambito cloud basato su uno strato fisico di hardware standard. Il progetto è stato ben accolto e si è sviluppato molto rapidamente grazie alla sempre crescente collaborazione di molte altre note aziende del settore informatico che hanno cominciato ad adoperare la piattaforma a scopo di *testing* e più in avanti, con le *release* successive, in maniera stabile nelle loro infrastrutture. Allo stato attuale il progetto conta otto *release* ufficiali stabili ed è organizzato in modo tale da rispettare scadenze semestrali per il rilascio delle nuove versioni.

Visto il crescente interesse nel progetto da parte di tante diverse aziende e comunità di sviluppatori indipendenti, è stata fondata la *OpenStack Foundation* che ora gestisce interamente tutti gli aspetti del progetto e presenta una struttura gerarchica secondo la quale si suddividono i collaboratori li più di duecento aziende multinazionali e la *community* mondiale. La *OpenStack Foundation* si occupa altresì dell'organizzazione di *summit* per gli sviluppatori a livello mondiale nei quali si possono discutere diversi aspetti del

progetto, presentare nuove idee e soluzioni adottate grazie alla piattaforma a livello corporativo.

Il progetto è *open-source* e *freeware*, è interamente implementato in Python e viene distribuito sotto l'*Apache License 2.0*.

4.2.2 Funzionalità Principali

L'organizzazione logica della piattaforma consiste in nove moduli indipendenti che gestiscono diversi aspetti dell'infrastruttura e collaborano al fine di fornire i diversi servizi attesi da una piattaforma *cloud*. *OpenStack* si riferisce ai suoi moduli tramite nomi in codice e le istruzioni a loro impartibili sono ben specificate da opportune API dedicate ai singoli componenti. Conviene, dunque, avere ben presente il ruolo e la natura di ogni modulo a partire dal suo nome:

- **NOVA**: È uno dei moduli più importanti della parte *IaaS* del sistema poiché si occupa interamente dell'aspetto della computazione e dunque della gestione diretta delle risorse virtuali. È progettato in modo da estere ampiamente scalabile orizzontalmente, è dunque in grado di gestire grandi numeri di macchine virtuali in maniera completamente distaccata dalla tecnologia di virtualizzazione in esse installata. A supporto delle sue caratteristiche di scalabilità, NOVA fa ricorso a svariate librerie esterne quali *libvirt*, *SQLAlchemy*, *Eventlet* e *Kombu* che realizzano la gestione di risorse di virtualizzazione, accesso a database, supporto alla programmazione concorrente e comunicazione via AMQP *Advanced Message Queuing Protocol* rispettivamente.

- **SWIFT**: L'aspetto di *storage* è gestito da due moduli diversi in *OpenStack*, in particolare SWIFT gestisce quel che viene definito l'*Object Storage* ovvero un sistema di *storage* ridondante che garantisce la continuità d'accesso alle risorse di memorizzazione. Nell'eventualità ci si siano problemi di accesso ad un singolo server o disco virtuale, è compito di SWIFT mettere a disposizione una sua replica presa dai nodi di ridondanza che compongono il *cluster* e successivamente garantire la consistenza dei dati tra i dischi e le loro repliche.

- **CINDER**: Continuando nell'ambito dello *storage*, questo modulo si occupa di offrire un sistema di *Block Storage* incaricato della creazione,

connessione e disconnessione dei dischi alle macchine virtuali. Di fatto le immagini disco che realizzano l'aspetto di memorizzazione di massa viene trattato in maniera distaccata dalle macchine virtuali. In questo modo si garantisce la preservazione dei dati all'interno della piattaforma, che non sono strettamente legati al ciclo di vita di una macchina virtuale. Inoltre, si semplifica non indifferentemente la migrazione di una macchina virtuale attraverso la piattaforma nonché aspetti di *fault tolerance* e *failure recovery* dovuti alla mancata disponibilità dei dati.

- **NEUTRON**: L'aspetto di *networking* e dunque uno dei servizi apertamente offerti dalla piattaforma, è interamente gestito da questo modulo, che risulta ampiamente compatibile con le principali tecnologie in ambito di reti e la loro virtualizzazione. Precedentemente noto sotto il nome id Quantum, questo modulo è progettato in modo da garantire che la rete non sia il "collo di bottiglia" dell'infrastruttura, implementando meccanismi di separazione delle reti e del traffico tramite VLANs e garantendo così un certo livello di *QoS* agli utilizzatori finali.

- **KEYSTONE**: Le *security policy* all'interno della piattaforma *OpenStack* sono gestite da questo modulo, che offre una *directory* centralizzata contenente tutti gli utenti della piattaforma e i loro permessi, ovvero le risorse, servizi o funzionalità a cui hanno accesso. *OpenStack* prevede un'organizzazione gerarchica in tre livelli che includono tre tipi diversi di utenti: anzitutto c'è il *Cloud Administrator* che si occupa della gestione dell'intera infrastruttura al più basso livello, ha dunque ampio accesso a tutte le risorse ed è suo compito stabilire i diversi gruppi del sistema e chi vi appartiene. Successivamente *OpenStack* prevede che le risorse che offre siano assegnate a diversi progetti (detti *tenant*) che saranno gestiti da amministratori con potere limitato al solo interno del progetto (*Admin User*) e composti da un insieme di utenti semplici che ci lavoreranno o usufruiranno dei servizi offerti (*End User*). Tramite la constante interrogazione di KEYSTONE, ogni operazione all'interno del sistema verrà eseguita con successo o sarà negata, rispettando così la *security policy* stabilita dal *Cloud Administrator* per l'intero sistema e quella relativa ai singoli *tenant*, stabilite dai *User Administrators*. Infine, il presente modulo offre altresì diversi meccanismi d'accesso conformi con i metodi più usati quali credenziali, *token based authentication* e altri ancora.

- **GLANCE**: A supporto della semplificazione di operazioni relative il *deployment* di una macchina virtuale, *OpenStack* mette a disposizione un *Image Service* garantito da questo modulo. Tale servizio si occuperà di gestire immagini di macchine virtuali e dischi che possono essere comodamente usati come meccanismo di back-up oppure come *template* per semplificare la creazione di macchine virtuali in un sistema nel quale il ciclo di vita di una risorsa virtuale può essere molto breve e continuamente rinnovato.

- **CEILOMETER**: Questo modulo offre una serie di servizi per monitorare l'intera infrastruttura che spaziano dall'uso delle risorse fisiche di computazione o memoria all'analisi del traffico all'interno della rete o delle reti virtuali. Risulta particolarmente utile per operazioni di *trouble shooting*.

- **HEAT**: Altro meccanismo a supporto delle funzionalità di *OpenStack* è quello dell'orchestrazione. Grazie ai servizi offerti da questo modulo e possibile predisporre *template* di sistemi più o meno complessi consistenti in diverse risorse virtuali collegate tra di loro, configurate in un certo modo ed eventualmente avendo anche certi servizi a disposizione, una volta soltanto per poter semplificare in futuro il *deployment* di sistemi che hanno punti in comune con i *template* predisposti.

- **HORIZON**: Esposti i diversi moduli che offrono i meccanismi per operare sull'infrastruttura, resta il presente modulo, che si occupa di offrire un'interfaccia *user friendly* per la gestione dell'infrastruttura. HORIZON offre ai diversi tipi di utenti della piattaforma una *dashboard* tramite un *server web* che tiene conto della *security policy* imposta al sistema già dalla fase d'autenticazione e sfrutta la conoscenza e combinazione opportuna delle API dei singoli moduli per compiere tutte le operazioni di gestioni necessarie che hanno a che fare con la manipolazione delle risorse e la loro suddivisione.

4.3 Metodi per la Gestione delle Risorse Virtuali

Poiché l'aspetto d'interesse di questo lavoro è quello dell'interazione con le diverse possibili risorse di virtualizzazione, basta considerare le funzionalità di NOVA, tralasciando gli altri moduli che, comunque, hanno un ruolo in

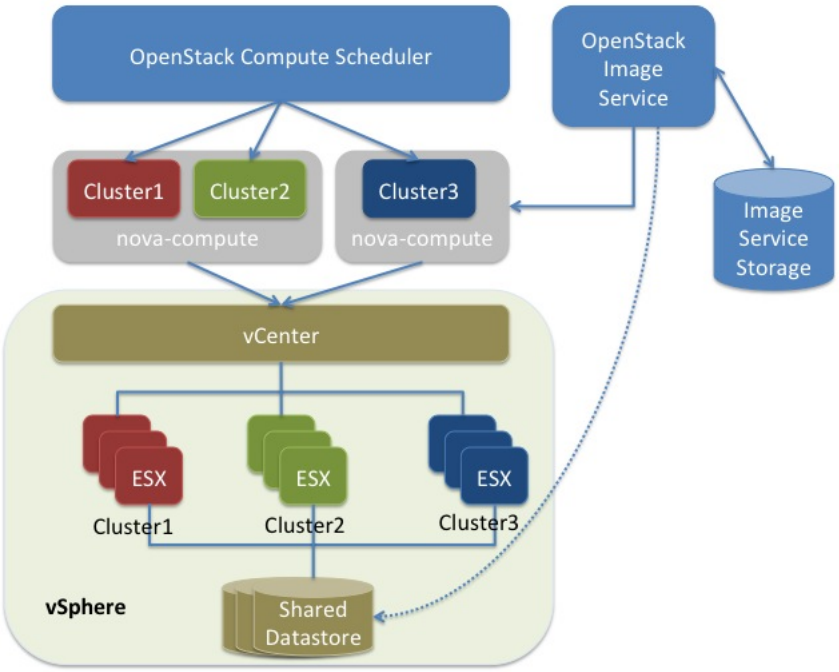

Figura 4.2: Modello di gestione di *VMM* sulla piattaforma *OpenStack*

qualsiasi interlocuzione avvenga tra NOVA e l'*hypervisor* d'interesse, basti pensare che ci sarà un controllo delegato a KEYSTONE per ogni chiamata invocata, onde verificare se l'utente che la evoca ha i permessi per farlo o meno e garantire così la consistenza del sistema.

Come già accennato, *OpenStack* si avvale dei meccanismi forniti da *libvirt* (con i suoi *driver* e *daemon*) per la gestione remota di macchine fisiche, lo fa però soltanto laddove non è già stato sviluppato un *driver* proprio del progetto (vantaggio della sua attiva *community*, che s'impegna a sviluppare aspetti tanto laboriosi del sistema). NOVA in quanto modulo *OpenStack*, viene installato direttamente su uno o più nodi (che possono benissimo essere a loro volta virtualizzati sugli *hypervisor*, caso analogo al *vmware vCenter*) e va opportunamente configurato in modo tale che sia in grado di comunicare con i *VMM* associati al nodo di computazione. La comunicazione diretta tra le due entità avviene secondo il modello proposto da *libvirt* già illustrato nella figura 3.7.

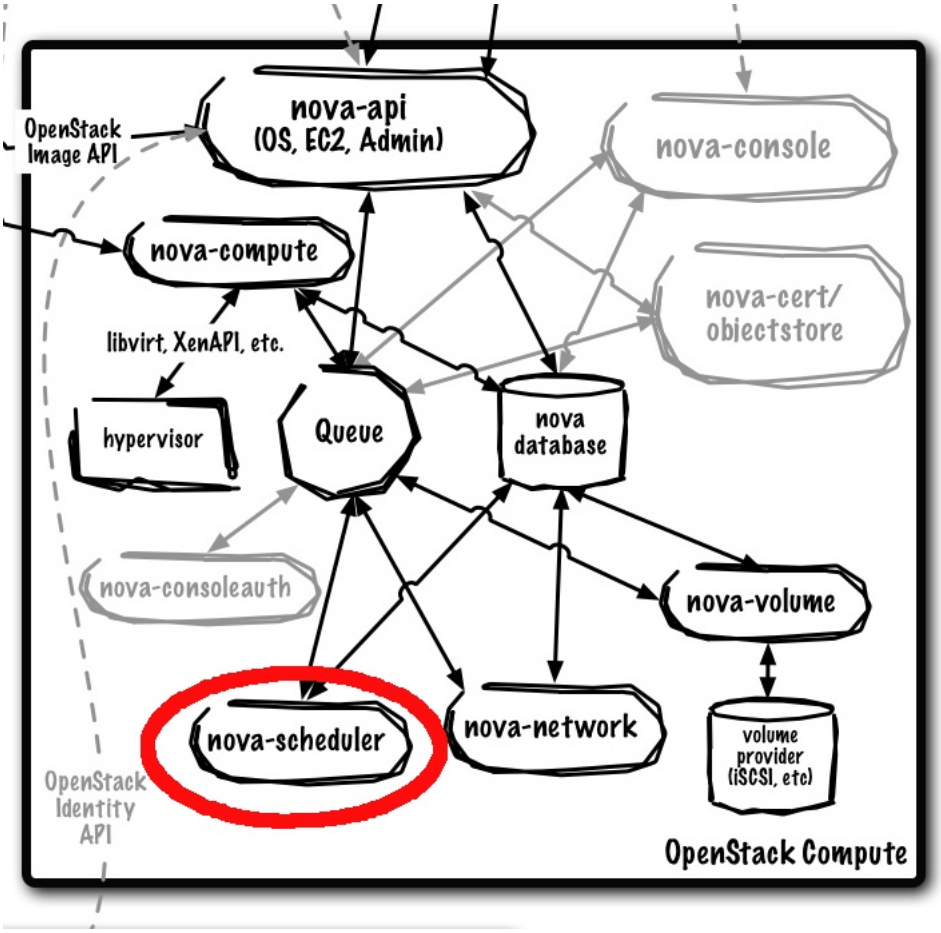

Figura 4.3: Interazione tra i componenti all'interno di un nodo NOVA

Nella figura 4.2 s'illustra un esempio d'interazione tra un nodo NOVA ed un nodo *vmware vCenter*. In questo modo, collegando due nodi si ottiene indirettamente un collegamento con tutti gli *hypervisor* monitorati dal nodo *vCenter*. Le interazioni in questo caso avverranno secondo le API predisposte da *vmware* per i suoi *VMM*. Nel dettaglio però, un nodo NOVA include svariati componenti per permettere una gestione intelligente delle risorse, in particolare diversi canali per gestire le richieste in entrata, inserirle in una coda, un meccanismo di *scheduling*, supporto alla memorizzazione ed indirizzamento così come controlli di sicurezza. L'anatomia di un nodo NOVA viene esplicitata in figura 4.3.

4.4 Esempi

Proviamo ora a considerare l'intera complessità del sistema, ovvero l'interazione tra tutti i moduli che compongono la piattaforma in una operazione tanto basilare quanto utile e ricorrente, l'istanziazione di una macchina virtuale all'interno del sistema. In prima battuta si potrebbe pensare che il modulo NOVA riceve una richiesta via CLI o *dashboard*, verifica i permessi dell'utente che effettua la richiesta e delega all'*hypervisor* la creazione della macchina virtuale secondo una certa specifica per la configurazione tramite le API della tecnologia di virtualizzazione relativa al *VMM*. Ebbene invece l'operazione non risulta tanto semplice. Onde garantire la consistenza dell'intero sistema tanti altri moduli entrano in gioco, vengono dunque informati dell'operazione ed eventualmente interrogati se è previsto un loro ruolo all'interno della richiesta. Di seguito un elenco delle interazioni tra i moduli, successivamente illustrati nella figura 4.4:

- Anzitutto consideriamo la sorgente che origina la richiesta, essa avverrà in due possibili modi, tramite l'interazione con HORIZON e dunque via *dashboard* oppure direttamente tramite CLI interagendo con la piattaforma *OpenStack*.

    1. La sorgente interroga l'utente per le sue credenziali d'accesso che successivamente comunica a KEYSTONE via RESTful APIs effettuando una classica procedura d'autenticazione.
    2. Se l'autenticazione va a buon fine, *KEYSTONE* genera e restituisce alla sorgente un *auth-token* utile alla sorgente per inviare richieste ad altri moduli *OpenStack* sempre tramite chiamate che rispettano i principi REST.

- Successivamente la sorgente interagisce con il nodo NOVA.

    3. La sorgente, ottenuto l'*auth-token*, converte il suo contenuto, una specifica *"launch instance"* o *"nova-boot"* in una nuova richiesta RESTful da inviare al componente nova-api che interfaccia il nodo NOVA con l'esterno.
    4. Per garantire la sicurezza del sistema nel caso la specifica sia stata generata in maniera anomala, nova-api interroga direttamente KEYSTONE per verificare l'autenticità della richiesta.

5. Se autentica, la richiesta verrà trovata nel database di KEYSTONE. Il modulo restituirà al componente nova-api l'autorizzazione a procedere specificando i permessi particolari per compiere successive operazioni che riguardino altri moduli.
6. nova-api cede il controllo al componente novaDB per creare una nuova *entry* nel database relativa alla macchina virtuale da creare. I suoi campi verranno successivamente compilati una volt ottenute le informazioni necessarie.
7. novaDB restituisce il controllo a nova-api una volta creata con successo l'entry.
8. nova-api richiede via `rpc` al componente nova-scheduler di individuare la locazione più opportuna per la virtual machine. Tale scelta verrà effettuata in base ad una politica di *load-balance* e *weight-balance*. Intanto però la richiesta viene inserita nella queue in attesa di venire evasa da nova-scheduler.
9. La richiesta viene letta dal componente nova-scheduler che procede ad individuare la locazione più opportuna per la macchina virtuale.
10. Una volta deciso dove risiederà la macchina virtuale, nova-scheduler interagisce con novaDB per compilare i campi relativi alla posizione della macchina virtuale all'interno dell'*entry* presente nel database.
11. Se l'operazione avviene con successo, novaDB restituisce il controllo a nova-scheduler.
12. Si procede dunque con la creazione dell'istanza. nova-scheduler effettua una chiamata via `rpc` al componente nova-compute per dare le indicazioni di dove creare la macchina virtuale. Tale richiesta, così come quella effettuata da nova-api, verrà inserita nella coda in attesa di essere evasa.
13. La richiesta viene letta dal componente nova-compute che procede con le successive operazioni per l'istanziazione.
14. Come prima cosa, il componente nova-compute, che è il diretto incaricato della comunicazione con il *VMM*, invia una richiesta al nova-conductor che s'incarica della spartizione delle risorse fisiche.

Così come il resto della comunicazione tra i componenti dello stesso modulo, essa avviene in maniera asincrona, via `rpc` ricorrendo alla *queue*.

15. nova-conductor preleva la richiesta dalla coda.
16. Una volta letta la richiesta, nova-conductor cede il controllo a novaDB per inserire nella *entry* del database i dati relativi alle risorse fisiche istanziate per la macchina virtuale.
17. Inserite con successo le informazioni nel database, novaDB restituisce il controllo a nova-conductor.
18. Una volta decise e confermate le disponibilità di risorse per la macchina virtuale, nova-conductor da a nova-compute l'autorizzazione a procedere.

- Successivamente vi è l'interazione con i componenti degli altri moduli *OpenStack* laddove richiesta la loro partecipazione, di norma GLANCE, NEUTRON (già QUANTUM) e CINDER.

19. nova-compute elabora una richiesta RESTful destinata a GLANCE, che s'interfaccia all'esterno tramite il suo componente glance-api, tramite la quale trasferisce l'*auth-token* generato all'inizio da KEYSTONE che contiene le operazioni da compiere ancora. Tale richiesta avviene poiché nova-compute necessita di conoscere l'URI e l'*Image ID* dei *template* eventualmente presenti in GLANCE da caricare sulla macchina virtuale.
20. Così come avviene con ogni richiesta proveniente dall'esterno, glance-api verifica la validità dell'*auth-token* interrogando KEYSTONE.
21. Se la verifica dell'*auth-token* va a buon fine, KEYSTONE da a glance-api il via libera per il trasferimento dell'immagine. glance-api trasferisce dunque a nova-compute le informazioni da esso richieste.
22. nova-compute procede con la configurazione dell'interfaccia di rete. Lo farà tramite una richiesta RESTful destinata al neutron-server, componente di NEUTRON che gestisce tutti gli aspetti riguardanti la rete.
23. In quanto richiesta proveniente dall'esterno, neutron-server verifica con KEYSTONE la validità dell'*auth-token* ricevuto.

24. Ottenuto il via libera da parte dei controlli di sicurezza, neutron-server comunica a nova-compute l'IP assegnato alla macchina virtuale.
25. Infine nova-compute elabora l'ultima richiesta RESTful destinata a CINDER, contenente l'*auth-token* che specifica la richiesta di una immagine disco da assegnare alla macchina virtuale.
26. Ancora una volta, la validità dell'*auth-token* viene verificata da cinder-api interrogando KEYSTONE.
27. Ottenuto il consenso da KEYSTONE, cinder-api procede a creare un disco rigido virtuale e comunica a nova-compute le informazioni relative al tale volume.

- Ultimo passaggio, dunque, l'interazione vera e propria con l'*hypervisor* che ospiterà "fisicamente" la macchina virtuale, sebbene potenzialmente i suoi dischi rigidi possano benissimo risiedere all'interno di un'altra macchina, magari adibita a *file server*.

  28. nova-compute comunica tutte le informazioni necessarie alla creazione della macchina virtuale ottenute dagli altri componenti al *hypervisor* scelto da nova-scheduler. Ciò avviene con l'utilizzo diretto delle API relative all'*hypervisor* oppure tramite *libvirt* a seconda della tecnologia di virtualizzazione.

Per quanto riguarda l'attività di monitoraggio di CEILOMETER, essa avviene in maniera piuttosto indipendente dal resto del sistema, poiché si pone come un *inspector* tra i moduli della piattaforma *OpenStack* e le risorse fisiche di computazione e di rete e solo se richiesto opportunamente dal *Cloud Administrator*.

Risulta evidente come un sistema del genere, in cui componenti della stessa macchina virtuale possano risiedere in diverse macchine fisiche della piattaforma, si fa fortemente affidamento sulla rete, che tuttavia non è affidabile, presenta problemi di latenza ed è soggetta a errori. Per ovviare a queste note problematica si fa certamente ricorso a reti ad alta capacità (*i.e. Gigabit Ethernet*) e le interazioni tra moduli sono progettate in modo da garantire una certa *QoS* all'utente.

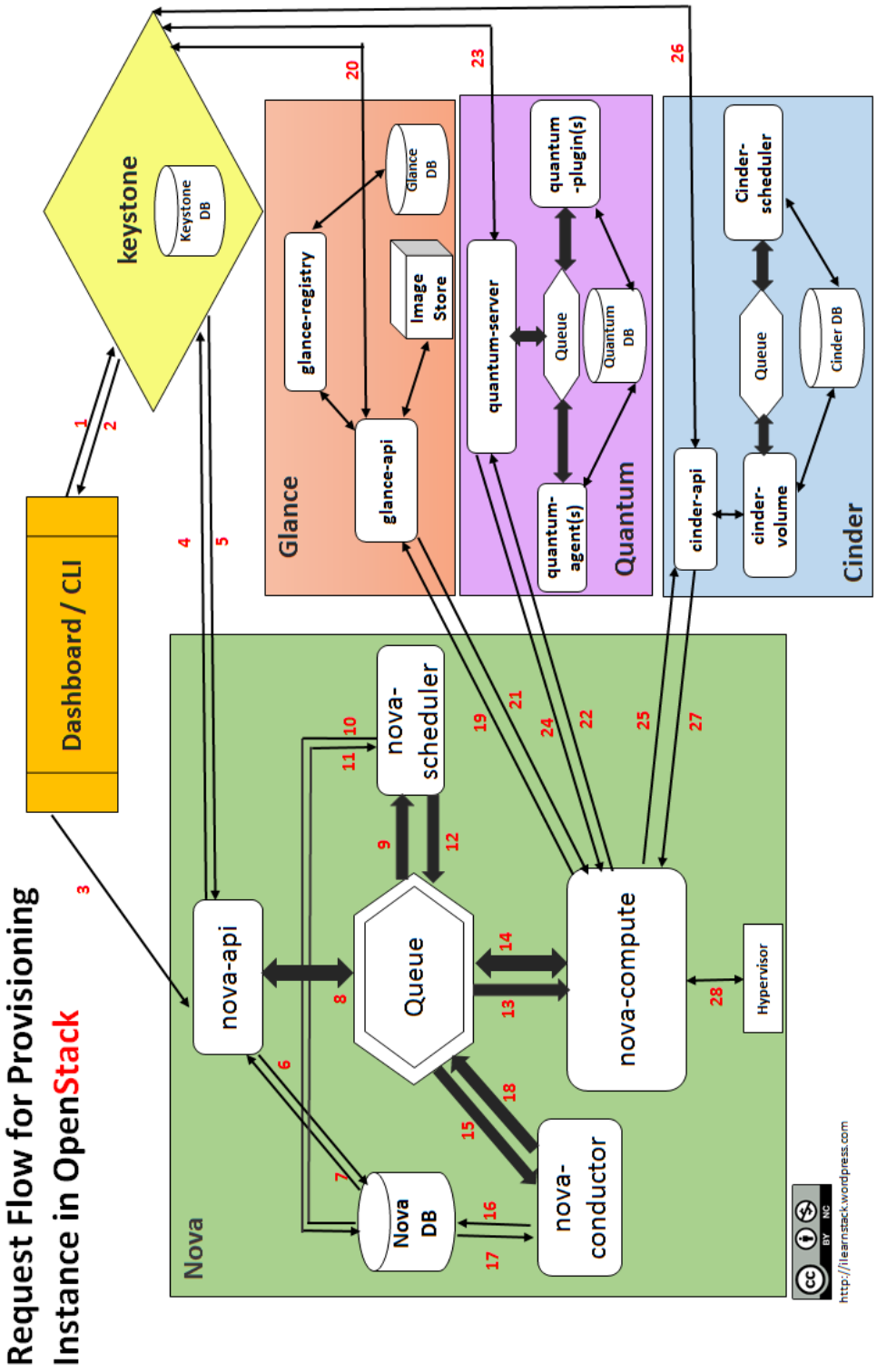

Figura 4.4: *Request Flow* per l'istanziazione di una macchina virtuale

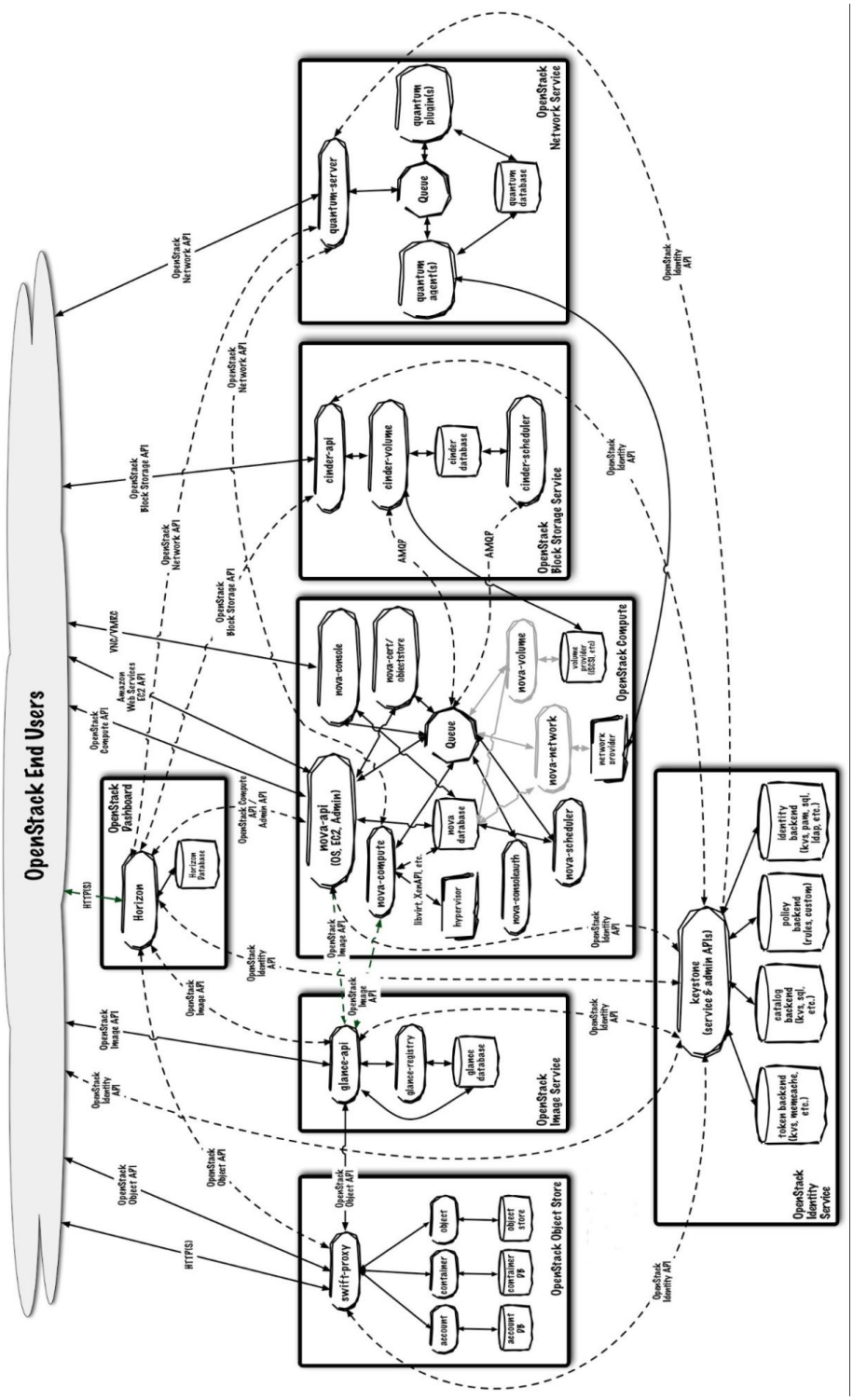

Figura 4.5: Visione completa di una piattaforma OpenStack

Capitolo 5

Conclusioni

Negli anni '50, gli ormai storici *mainframe*, che occupavano stanze intere dei laboratori di ricerca, venivano acceduti da piccoli terminali senza potenza di calcolo dedicati esclusivamente all'impartizione di ordini. Onde incrementare la produttività e giustificare l'immensa spesa che comportava non solo l'acquisto, ma anche la manutenzione di queste macchine, furono sviluppate le prime tecniche di scheduling e l'accesso concorrente tramite più *dummy terminals*. Nacque in questo modo il concetto di *time-sharing* che oggi è alla base di ogni CPU. Risulta curioso notare la piega presa dopo oltre sessanta anni di evoluzione. Inizialmente si puntava ad incrementare sempre di più la potenza dei singoli terminali, anche quelli divenuti *home computers* grazie alla visione di tante note aziende del settore informatico. Al giorno d'oggi s'impone nel mercato sempre di più un modello che curiosamente ha molti punti in comune con i suddetti *mainframe*. Non risulterebbe sorprendente se tra dieci anni vi fosse uno scenario in cui per ogni casa ci sia un terminale con una potenza di calcolo limitata ed una semplice connessione ad internet, ma in quanto potenzialità ed offerta, non abbia nulla da invidiare ai computer odierni. L'approccio *cloud* sta portando l'intero mondo dei computer proprio in questa direzione. Proprio come se la famigerata "nuvola" altro non fosse che un insieme di *mainframe* con tanti *dummy terminals* ad effettuarvi accesso e richiedere servizi. Insomma, tornare ai modelli del passato, ma sfruttando tutte quelle tecnologie sviluppate nel frattempo. Si è visto come gran parte del merito vada alle tecnologie di virtualizzazione, che hanno permesso di materializzare su una dimensione alternativa elementi una volta fisici e tangibili. Tuttavia, nell'ottica di avere un sistema di facile ac-

cesso, manutenzione, estensione ed uso, bisogna andare incontro a non pochi problemi.

Nel presente lavoro di tesi si è discusso di uno dei principali tra i problemi nel forte affidamento a infrastrutture virtuali. Ciò nonostante, si è anche visto, studiando una piattaforma collaudata ed in continuo sviluppo, come tale aspetto sia soltanto una piccola parte dell'insieme di interazioni tra gli elementi di una piattaforma *cloud*. E ad ogni tipo diverso d'interazione, ad ogni risorsa richiesta, ad ogni elemento interrogato, altre problematiche salgono in superficie. Ricordando la celebre frase attribuita al premio nobel irlandese George Bernard Shaw, *"La scienza è sempre imperfetta. Ogni volta che risolve un problema, ne crea almeno dieci nuovi"*. Ebbene la *Computer Science* non fa eccezione. Tuttavia tutto ciò spinge ricercatori e sviluppatori a trovare nuove soluzioni, a coniare nuovi concetti e teorie, cercando di spingersi sempre oltre il limite e spostare la soglia del progresso sempre un po' più lontana.

Sviluppi futuri concreti dopo la soluzione alla problematica d'interazione tra elementi eterogenei potrebbero benissimo trovarsi in piattaforme di gestione e controllo che offrano tutte le funzionalità già presenti nei sistemi omogenei, rendendo completamente trasparente all'utilizzatore la natura della piattaforma, comportando d'altro canto immensi vantaggi economici per i *service provider* ed *infrastructure provider*. Studiando i possibili scenari ed applicando opportunamente i *pattern* più opportuni nei punti giusti, si può arrivare ad un modello con risultati ottimali in termini di *performance* ed affidabilità, donando al *cloud*, ma anche in generale ai sistemi eterogenei, punti in più a loro favore.

Bibliografia

[1] Amazon AWS. `http://aws.amazon.com/`:*Overview, ec2, s3.*

[2] Dmtf. `http://www.dmtf.org/`:*cim, ovf.*

[3] Eucalyptus. `https://www.eucalyptus.com/`: *Overview.*

[4] libvirt. `http://libvirt.org/`: *Overview, API reference, Architecture, Goals, Language bindings, Drivers, Secure Usage, Domains, Network, Applications.*

[5] Opennebula. `http://opennebula.org/`: *Overview.*

[6] OpenStack. `https://www.openstack.org/`: *Overview.*

[7] OpenStack docs. `http://docs.openstack.org/`: *Overview, Nova, Drivers, Operations Guide, API quick start, Training guides, Command-line Interface.*

[8] OpenStack wiki. `https://wiki.openstack.org/wiki/`: *Overview, Nova, Neutron, Swift, Cinder, Keystone, Horizon.*

[9] vmware. `http://www.vmware.com/`:*Overview, vCloud Suite, vSphere, ESXi, vCenter.*

[10] Wikipedia. `http://en.wikipedia.org/`:*Hypervisor, Cloud Computing, Windows Azure, Virtual Private Cloud, Hardware Virtualization, DDS.*

[11] Windows Azure. `http://www.windowsazure.com`:*Overview.*

[12] K. An, S. Pradhan, F. Caglar, and A. Gokhale. A Publish/Subscribe Middleware for Dependable and Real-time Resource Monitoring in the Cloud. *ACM Digital Library*, 2012.

[13] J. Araujo, F. Schneider, W. Lisandro, et al. A New Approach to the Design of Flexible Cloud Management Platforms. *IFIP*, 2012.

[14] S. A. Baset. Open Source Cloud Technologies. *ACM Digital Library*, 2012.

[15] L. Beernaert, M. Matos, R. Vilaça, and R. Oliveira. Automatic Elasticity in OpenStack. *ACM Digital Library*, 2012.

[16] M. Bolte, M. Sievers, G. Birkenheuer, et al. Non-intrusive Virtualization Management using libvirt. *EDAA*, 2010.

[17] Y. Cho, J. Choi, and J. Choi. An Integrated Management System of Virtual Resources based on Virtualization API and Data Distribution Service. *ACM Digital Library*, 2013.

[18] DMTF. Open Virtualization Format Specification. Technical report, DMTF, 2012.

[19] IEEE, editor. *Towards an Agent-Based Symbiotic Architecture for Autonomic Managements of Virtualized Data Centers*, 2012.

[20] Y. Jegou, P. Harsh, R. Cascella, et al. Managing OVF Applications Under SLA Constraints on Contrail Virtual Execution Platform. *ACM Digital Library*, 2012.

[21] H. Lee and J. Lee. Design for Management Software of Desktop Virtualization Solutions. *IEEE*, 2010.

[22] E. Maldini. Modelli e Tecnologie Per La Gestione di Macchine Virtuali Tramite API Software.

[23] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. Elastic Management of Cluster-based Services in the Cloud. *ACM Digital Library*, 2009.

[24] K. Nagin, D. Hadas, Z. Dubitzky, et al. Inter-Cloud Mobility of Virtual Machines. *ACM Digital Library*, 2011.

[25] D. Niyato, K. Zhu, and P. Wang. Cooperative Virtual Machine Management for Multi-Organization Cloud Computing Environment. *ACM Digital Library*, 2011.

[26] OpenStack. *OpenStack - Admin User Guide*, November 2013.

[27] OpenStack. *OpenStack - Cloud Administrator User Guide*, November 2013.

[28] OpenStack. *OpenStack - End User Guide*, November 2013.

[29] W. Shi, J. Lee, S. Taeweon, et al. Architectural Support of Multiple Hypervisors over Single Platform for Enhancing Cloud Computing Security. *ACM Digital Library*, 2012.

[30] Z. Song and L. Xiao. Research and Design of Inter-Communication in DVMM. *IEEE*, 2009.

[31] V. Soundararajan and K. Govil. Challenges in Building Scalable Virtualized Datacenter Management. *vmware*, 2011.

[32] U. Steinberg and B. Kauer. NOVA: A MIcrohypervisor-Based Secure Virtualization Architecture. *ACM Digital Library*, 2010.

[33] G. Vallée, T. Naughton, and S. L. Scott. System Management Software for Virtual Environments. *ACM Digital Library*, 2007.

[34] VMware Inc. and XenSource Inc. The Open Virtual Machine Format Whitepaper for OVF Specification. Technical report, DMTF, 2007.

[35] S. Wu, L. Deng, H. Jin, et al. Virtual Machine Management Based on Agent Service. *IEEE*, 2010.

[36] F. Wuhib, R. Stadler, and H. Lindgren. Dynamic Resource Allocation with Management Objectives - Implementation for an OpenStack Cloud. *IFIP*, 2012.

Elenco delle figure