Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Strumenti di gestione per infrastrutture IT

Tesi di Laurea in Architettura degli Elaboratori

Relatore:
Chiar.mo Prof.
Vittorio Ghini

Presentata da:
Andrea Stella

Sessione II
Anno Accademico 2013/2014

A Nello e Nella.

Indice

Elenco delle figure

Elenco delle tabelle

Prefazione

Nonostante il mio elaborato non tratti prettamente il cloud computing inteso come tale, bensì i tools utilizzati per gestire l'intera infrastruttura IT, mi piacerebbe fare un'osservazione critica riguardo ad una *curiosa* citazione che mi è capitato di leggere durante la stesura di questo lavoro:

> *C'era un tempo in cui ogni casa, città, fattoria o villaggio aveva il suo pozzo dell'acqua. Oggi, i servizi pubblici danno accesso all'acqua "**pulita**" semplicemente girando il rubinetto.*
> *Il cloud computing funziona in modo simile.*
>
> *Vivek Kundra*

Ma allora, se quest'acqua non fosse così pulita come molti vogliono farci credere?
Parecchie e contrastanti sono le opinioni degli esperti proprio sull'utilità del cloud computing.
Richard Stallman, fondatore della Free Software Foundation e pioniere del copyleft[1], definisce l'uso massiccio di cloud computing "peggio della stupidità" in quanto sinonimo di perdite di controllo sui dati:

> "Una stupidaggine. Anzi, peggio di una stupidaggine: una campagna marketing. Qualcuno dice che è inevitabile e quando sentite qualcuno dire così, è molto probabile che si tratti di una

[1]Il copyleft è un metodo generico per rendere un programma (o altro lavoro) libero ed imporre che tutte le modifiche e versioni estese del programma siano anch'esse software libero. Per approfondimenti http://www.gnu.org/copyleft/copyleft.it.html

> strategia d'affari per renderlo vero. Una ragione per non usare le web application è la perdita del controllo: i dati fluiscono liberamente tra qualsiasi postazione o thin client in giro per il mondo e i datacenter, ma a scapito della capacità del legittimo proprietario di disporne a suo piacimento. Basti pensare a cosa accadrebbe nel caso in cui un account venisse sottratto al suo titolare: da quello stesso account potrebbe partire una reazione a catena, che coinvolgerebbe tutti gli altri servizi ad esso collegati, stravolgendo le attività personali e lavorative di quello stesso individuo. È un male proprio come usare programmi proprietari. Fate il vostro lavoro su un vostro computer con un programma che rispetti le vostre libertà: usando un programma proprietario sul server di qualcun altro si è senza difese. Vi state mettendo nelle mani di chiunque abbia sviluppato quel software. Il rischio è che se all'inizio questi servizi possono apparire più economici (o addirittura gratuiti) rispetto alle abitudini attuali, nel lungo periodo possano invece rivelarsi oltremodo costosi. E soprattutto, l'intera mole di informazioni personali (foto, appunti, appuntamenti in agenda) o aziendali (budget, bilanci, piani strategici) sarebbe affidata alla onestà e alla solidità di una azienda, esponendosi a tutti i rischi di boicottaggio o incidenti che questo comporta."[2] [3]

Inoltre Stallman ha criticato anche il sistema studiato da Google, Chrome OS, in quanto a suo avviso andrebbe ulteriormente a peggiorare le cose poichè esplicitamente pensato e disegnato (tra l'altro su sistema GNU/Linux) per conservare localmente quante meno informazioni possibili, lasciando il grosso dei dati alle "nuvole" Google, collocate in luoghi ignoti:

> "Credo che a chi lavora nel marketing piaccia il termine "cloud computing" in quanto privo di sostanziale significato. Il significato del termine non é una sostanza, ma un atteggiamento: "Lascia che Tom, Dick e Harry tengano i tuoi dati, lascia che qualsiasi Tom, Dick e Harry si occupino dei tuoi dati (e li controllino)".

> Probabilmente il termine "careless computing"[2] sarebbe più calzante. [...] Il Governo potrebbe incoraggiare le persone a piazzare i dati dove esso sia in grado di raggiungerli senza mostrare mandati di perquisizione, piuttosto che in luoghi di proprietà propria. A ogni modo, finché abbastanza di noi continueranno a tenere i propri dati sotto il proprio controllo, potremo ancora continuare a farlo. E faremmo bene a continuare così, o questa opzione potrebbe scomparire".[4]

E allora mi chiedo: considerando che l'88% di chi naviga su Internet usa almeno un servizio di cloud computing, spesso in modo inconsapevole e per memorizzare dati sensibili, quanti di questi si è mai posto il problema della privacy? Sempre secondo Stallman vi è un problema strisciante di fondo:

> "Penso che molte persone continueranno a spingersi verso il careless computing, *perchè nasce un nuovo babbeo ogni minuto*."

[2] Careless vuol dire incurante.

Introduzione

Con l'introduzione e l'estensione di tecniche informatiche e telematiche si è avuta una vera e propria rivoluzione in parecchi settori, dall'economia alla politica, alla società. Soprattutto nel settore economico, ma non solo, l'informatizzazione ha cambiato sia i ritmi produttivi che i consumi, automatizzando gran parte del lavoro e portando l'uomo ad occuparsi prevalentemente di funzioni di controllo. Sempre più aziende, enti o organizzazioni hanno bisogno di una propria infrastruttura IT, e soprattutto nel periodo iniziale non è proprio semplice definirne le reali necessità. Un'attenta e scrupolosa pianificazione delle risorse, quindi, risulta essere necessaria: investire in molte risorse significherebbe avere dei costi iniziali molto elevati con il rischio che molte di queste non vengano mai effettivamente utilizzate, così come limitarne la quantità iniziale porterebbe ad un possibile reinvestimento nel caso in cui l'infrastruttura non soddisfacesse le necessità di quel momento. Questo rappresenta un punto chiave difficile da prevedere.
Inoltre, la gestione di un'infrastruttura IT non è un compito semplicissimo, soprattutto quando bisogna configurare e gestire un gran numero di macchine.
L'avvento di nuove tecnologie come la virtualizzazione e il Cloud Computing semplifica sia il processo di pianificazione, poichè ad esempio il consumatore ha la possibilità di scalare orizzontalmente l'intera infrastruttura pagando solo ciò che utilizza, che quello di gestione e manutenzione della stessa poichè vengono decentralizzate al di fuori dell'azienda. Con gli anni un gran numero di provider hanno iniziato ad offrire i propri servizi al pubblico, ognuno utilizza le proprie piattaforme di gestione ed API, creando così dei problemi di comunicazione tra di essi. Per questo, diverse community si sono impegnate

nell'implementazione di progetti open source e di open standards proprio per rendere più omogeneo l'utilizzo di questi servizi e in modo da fornire un'interfaccia comune di facile gestione. Inoltre molte compagnie hanno investito il loro denaro per lo sviluppo di alcuni tools, i cosiddetti Software Configuration Management, che vengono incontro alle esigenze degli amministratori di sistema: facilitano ed automatizzano la gestione delle configurazioni di software e servizi di un'infrastruttura informatica.

Questo elaborato è dunque così suddiviso.
Nel primo capitolo faremo riferimento ai cenni storici e daremo una breve panoramica delle caratteristiche e dei modelli del cloud computing.

Il secondo tratterà l'analisi delle piattaforme open source più utilizzate per la gestione di sistemi cloud: dopo una breve panoramica dei software esistenti, faremo uno studio più approfondito dell'architettura, dei componenti e delle principali caretteristiche di OpenStack, OpenNebula, ed Eucalyptus.

Nel terzo capitolo ci focalizzeremo nello studio e nell'analisi dei tools di gestione delle configurazioni che monopolizzano il mercato attuale: Puppet di Puppet Labs e Chef di Opscode; faremo anche una breve descrizione di due altri software che negli ultimi anni stanno prendendo sempre più piede in questo settore: Salt di SaltStack ed Ansible di AnsibleWorks.

E per concludere, nell'ultimo capitolo verranno introdotti gli standards, alcuni in fase di sviluppo altri già definiti, sviluppati negli ultimi anni con lo scopo di risolvere le problematiche legate alla comunicazione tra diversi provider. Verrà analizzato lo standard OCCI (Open Cloud Computing Interface) ed in particolare i tre moduli di cui è composto: l'OCCI Core, l'OCCI Infrastructure e l'OCCI RESTful HTTP Rendering.

Capitolo 1

Cloud Computing

Il marchio di garanzia di ogni termine gergale è la capacità di suggerire un'apparenza di significato, senza tuttavia darne alcuno. Per molti, il termine *cloud computing* è certamente un termine gergale.
Questo termine è utilizzato in contesti diversi e discordanti, ad esempio spesso per riferirsi al servizio di posta elettronica di Google oppure per parlare di EC2 di Amazon. Ma il cloud computing non è un termine gergale più di quanto lo sia la parola *Web*. Il cloud computing è l'evoluzione e la convergenza di una serie di tecnologie che si sono sviluppate negli ultimi trent'anni, in grado di rivoluzionare le modalitá con cui le organizzazioni costruiscono le proprie infrastrutture informatiche.[1]

1.1 Storia del cloud computing

L'espressione "cloud computing" è stata usata per la prima volta nel mondo del marketing per evidenziare il passaggio da dati e/o servizi disponibili in un numero limitato di dispositivi alla strutturazione dinamica tipica del web, dove questi ultimi sono sparsi all'interno di un sistema di grandi dimensioni, ma sempre localizzabili e raggiungibili dall'utente.
Nell'ultimo decennio il processo di evoluzione dell'informatica e della microelettronica ha alterato completamente le nostre abitudini, il nostro modo di lavorare e di interagire con gli altri. Quello che oggi reputiamo necessario, ad

esempio computer e cellulari, un tempo era veramente un'esigenza che pochi potevano permettersi economicamente e ancor più che pochi sapevano utilizzare. Le origini di questa nuova tecnologia sono da ricercarsi intorno agli anni 50-60, quando John McCarthy[1] immaginava un futuro in cui l'elaborazione dei calcoli sarebbe potuta essere distribuita ed organizzata su diversi sistemi pubblici d'accesso.[6]
Quando negli anni '80 l'uso di mainframe e minicomputer ("*mini*" per l'epoca, ovviamente) viene soppiantato dall'avvento dei personal computer, le aziende iniziano a porsi il problema di come condividere facilmente le risorse tra i pc dei propri dipendenti: nasce l'architettura client/server.
Con la comparsa delle connessioni dialup[2] le aziende iniziarono ad interconnettere le loro filiali in modo semplice, ma solo dopo con l'avvento del Web negli anni '90 nasce l'idea dell' Application Service Provider (ASP), ovvero un modello architetturale per l'erogazione di servizi informatici tramite accesso remoto. Tutte le soluzioni proposte[3], però, non ebbero un seguito per gli stessi motivi per i quali molta gente ancora oggi non vede di buon occhio il cloud computing, tra i quali citiamo i problemi che riguardano la proprietà dei dati, la qualità del servizio, e la sicurezza.
Successivamente, negli anni 2000 alcune compagnie produttrici di software presentarono alle aziende dei prodotti per la virtualizzazione, subito accolti positivamente perchè visti come la possibilità concreta di ottimizzare l'infrastruttura IT ed abbassare il TCO[4]. Infatti si ha un risparmio sull'acquisto, installazione, manutenzione e dismissione di hardware e software. Qualche anno dopo nasce una nuova architettura, la Service Oriented Architecture (SOA), che unita alle tecnologie di virtualizzazione hanno permesso lo sviluppo di applicativi altamente portabili e ad architettura distribuita, oltre alla possibilità di interconnettere ed utilizzare facilmente software eterogenei ed eventualmente gestiti da altre aziende.

[1]Informatico statunitense, conosciuto come il padre fondatore dell'Intelligenza Artificiale (AI). Vince il Premio Tuning nel 1971.

[2]Connessioni tra computer realizzate con l'utilizzo di modem tramite la composizione di una normale numerazione telefonica, cioè dunque utilizzando l'usuale banda fonica a bassa frequenza, grazie a opportuni programmi detti dialer

[3]Ricordiamo Citrix WinFrame

[4]Total cost of ownership

Negli anni a seguire con l'aumento del numero degli *internauti* nacquero i primi servizi on-demand, ovvero basati sul modello Software as a Service (SaaS), che attirarono subito l'interesse delle grandi aziende dell'IT mondiale. Microsoft infatti consolidò i suoi web service mentre IBM diede vita nel 2001 ad un'iniziativa, l'Autonomic Computing[8], che aveva lo scopo di fornire ai computer gli strumenti necessari per autogestirsi senza l'intervento umano.
Probabilmente la vera svolta si è avuta quando Amazon, grande colosso mondiale della vendita online, ha modernizzato i propri datacenter secondo la tecnologia cloud e, grazie all'avvento nel 2006 degli Amazon Web Sevice (AWS), ha offerto la possibilità agli utenti di usufruire dei propri servizi, nonostante questi ultimi avessero già avuto inconsapevolmente delle esperienze con la nuova tecnologia, grazie all'utilizzo di Hotmail e Gmail.
In Italia, invece, è Telecom ad offrire i primi servizi basati sul cloud computing grazie alla sua "Nuvola Italiana".

1.2 Caratteristiche principali

Nonostante non esista una definizione accettata da tutti per il cloud computing, una un pò più *classica* è quella redetta dal NIST[5]:

> " Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models "[5]

ovvero:

[5]National Institute of Standards and Technology. Agenzia del governo degli Stati Uniti d'America che si occupa della gestione delle tecnologie

> "Il cloud computing è un modello di elaborazione che abilita un accesso in rete, su richiesta, ubiquo e conveniente, ad un insieme di risorse di calcolo (CPU, storage, reti, sistemi operativi, servizi e/o applicazioni) condivise e configurabili, e che possono essere acquisite e rilasciate rapidamente e in modo dinamico. Questo comporta uno sforzo di gestione minimo, o comunque un'interazione minima con il fornitore del servizio. Questo modello di cloud è caratterizzato da cinque aspetti chiave, tre modelli di servizio e quattro modelli di deployment."

Gli aspetti fondamentali di questo livello sono:

1. *servizi su richiesta:* un consumatore può richiedere i servizi in maniera autonoma, senza alcun intervento umano da parte del fornitore di servizio

2. *accessibilità globale:* queste funzionalità di calcolo sono disponibili in rete e vengono accedute tramite l'utilizzo di meccanismi standard da dispositivi diversi (come ad esempio smartphone, tablets, laptops), da piú luoghi e in ogni momento.

3. *raggruppamento delle risorse:* le risorse fisiche e virtuali messe a disposizione dal provider ed assegnate e riassegnate in maniera dinamica agli utenti sulla base delle loro richieste, sono riunite per servire una molteplicitá di consumatori tramite un modello multi-tenant[6], senza che questi abbiano nè il controllo nè la conoscenza esatta della locazione delle risorse che gli sono state assegnate[7].

4. *elasticità immediata:* queste funzionalità possono essere ottenute in modo rapido ed elastico, e qualche volta questo può avvenire automaticamente. Agli occhi degli utilizzatori la quantità di risorse a disposizione appare quasi illimitata. Queste possono essere acquisite in qualsiasi quantità ed in qualunque momento.

[6] Letteralmente, "con più affittuari".

[7] A volte si ha un controllo sulla locazione ad un livello di astrazione più alto, ad esempio la nazione, e questo succede per motivi legislativi

5. *misurabilità dei servizi:* i sistemi di cloud computing controllano e ottimizzano in modo automatico l'uso delle risorse sulla base di misure appropriate per il tipo di servizio. Sia il fornitore che il consumatore possono monitorare in maniera trasparente l'utilizzo del servizio, così da poter essere applicati dei modelli di *pay per use*.

1.3 Modelli di infrastruttura a nuvola

Se un'organizzazione decide di gestire la propria infrastruttura deve affrontare e risolvere una serie di problemi legati a costi, risorse hardware, e dati. La pianificazione delle risorse è da sempre una questione critica. Un'infrastruttura basata sulla novola, scelta in base alle specifiche esigenze dell'azienda, semplifica notevolemente queste problematiche perchè è il fornitore di servizio a prendersi carico della gestione e configurazione del sistema cloud. Basti pensare a cosa succederebbe se ad esempio sbagliamo a pianificare le nostre risorse oppure se non abbiamo abbastanza capitale per comprare nuovo hardware. Inoltre, in caso di malfunzionamenti bisognerebbe ripristinare il sistema nel minor tempo possibile per evitare un blocco completo del servizio, oppure si rischierebbe di avere a bilancio dei costi extra, se un server non è più necessario e rimane inutilizzato. Se si adotta invece una soluzione basata sul cloud computing non dovremmo preoccuparci di quanto sopra elencato poichè avremo la possibilità di aggiungere risorse hardware nel momento in cui queste si rendono necessarie e disabilitarle quando non ne avremo più bisogno, non avremo spese di startup ma solo d'esercizio e non dovremo preoccuparci di gestire l'hardware.
Di seguito verranno descritti i principali modelli infrastrutturali del cloud computing.

1.3.1 Software as a Service (SaaS)

In questo modello ad essere erogati come servizi sono proprio i software. I SaaS sono un modello di distribuzione del software che permette l'utilizzo delle applicazioni da remoto (ad esempio utilizzando un server web).

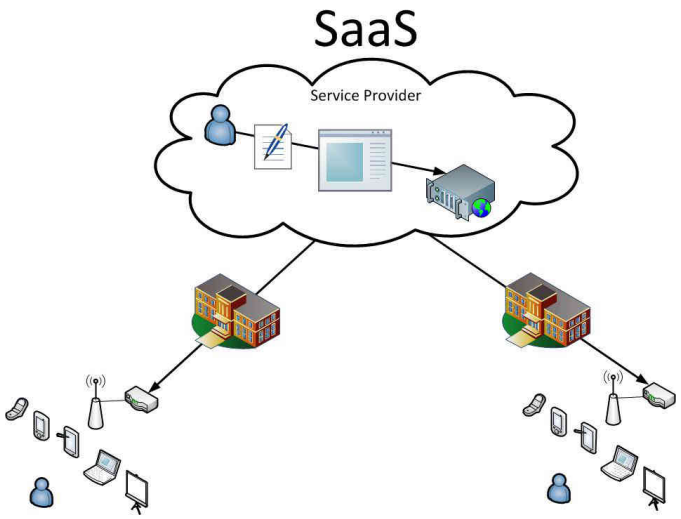

Figura 1.1: Modello architetturale SaaS

Dal punto di vista dell'utilizzatore è del tutto trasparente dove il software è ospitato, su che sistema operativo gira e in che linguaggio di programmazione è scritto. Ma soprattutto l'utente, oltre a non dover installare alcuna applicazione in locale, non deve preoccuparsi di dover gestire l'applicazione (talvolta la può configurare) nè l'infrastruttura per la sua esecuzione.
I benefici di questo modello sono diversi e possono essere così riassunti:

1. *sempicità di amministrazione*
2. *aggiornamenti e gestione delle patch*[8] *automatici*
3. *compatibilità e facile collaborazione*, in quanto tutti i consumatori hanno la stessa versione del software
4. *accessibilità globale*
5. *riduzione dei costi di licenza*

Nonostante questo, però, rimangono i soliti problemi che riguardano la gestione della privacy, poichè i dati delle aziende sono reperibili sui server dei fornitori di servizi SaaS. Proprio per questo vengono attuate delle politiche

[8]Modifica del codice

di sicurezza in modo che si possano avere i giusti permessi per amministrare il database ma nessuno di questi per accedere al contenuto. Inoltre, per maggiore sicurezza, vengono utilizzati meccanismi di crittografia nella comunicazione tra fornitore e clienti.

1.3.2 Database as a Service (DBaaS)

Database as a Service è un approccio architetturale ed operativo che consente ai fornitori di servizi IT di offrire a più consumatori la possibilità di memorizzare dati online, sui propri server.

Figura 1.2: Modello architetturale DBaaS

Le caratteristiche essenziali di questo modello sono:

1. fornitura e gestione di ogni istanza del database da parte del consumatore basata su meccanismi on-demand e self-service

2. monitoraggio automatizzato e conformità con quanto definito dal provider in termini di qualità del livello di servizio

3. controllo capillare sull'utilizzo del database in modo da offrire funzionalità di reportistica e fatturazione per ogni utente

Naturalmente in aggiunta a queste peculiarità ci si aspetta che questo modello supporti l'accesso da una vasta gamma di dispositivi e meccanismi, e garantisca la multi-tenancy e la gestione automatizzata delle risorse. Basta pensare al fatto che gli operatori cloud lavorano su centinaia, migliaia o

addirittura decine di migliaia di database allo stesso tempo, per capire che i servizi DBaaS devono mettere a disposizione per questi ultimi delle API[9] che appunto garantiscano, in maniera molto flessibile, l'automazione di tutte queste funzionalità di gestione. Alcuni, come Kurt Mackey[10], parlano del DBaaS come il futuro dell'app economy, la soluzione ai problemi di sviluppo delle applicazioni, ma anche per questo tipo di modello il problema è sempre lo stesso: quello della privacy.

1.3.3 Hardware as a Service (HaaS)

Nel *Grid Computing*[11] l'Hardware as a Service è un modello pay-as-you-go che garantisce l'accesso all'infrastruttura e alla potenza di calcolo di un provider. Il cliente dapprima invia la mole di dati ad una macchina del servizio. Successivamente queste informazioni verranno elaborate dalla *griglia* di computer del provider, che li restiruirà all'utente finale non appena la lavorazione sarà terminata.
Quest'architettura al momento risulta essere la più difficile da implementare per gli eccessivi costi iniziali di investimento.

1.3.4 Platform as a Service (PaaS)

Platoform as a Service è una categoria del cloud computing che fornisce un ambiente operativo completo per consentire agli sviluppatori, tramite l'apposita piattaforma di sviluppo messa a disposizione dal fornitore, di creare applicazioni e servizi su internet. È proprio quest'ultimo ad occuparsi dei dettagli della messa in produzione. Questi servizi risiedono nella *nuovola* e

[9]Il modello API standard utilizzato per le funzioni di cloud è REST (Representational State Transfer). Un'interfaccia di questo tipo che sta iniziando a guadagnare popolarità è *Trove* per OpenStack, che fornisce un meccanismo standard per automatizzare molte delle funzioni DBaaS tramite MySQL.

[10]Fondatore di MongoHQ. MongoHQ fornisce soluzioni per la gestione di database agli sviluppatori che utilizzano MongoDB come la loro tecnologia di archiviazione dati.

[11]Tra le architetture applicative è la più semplice da implementare nella nuvola. Un'applicazione in modalità grid computing è un software molto intensivo in termini di utilizzo di CPU che suddivide le proprie procedure di calcolo in modo da poterle eseguire contemporaneamente su più macchine.

sono accessibili agli utenti semplicemente tramite l'uso di un comune web browser, tutto questo azzerando i costi e la complessità associati all'acquisto, alla configurazione, all'ottimizzazione e alla gestione dell'hardware e del software di base.

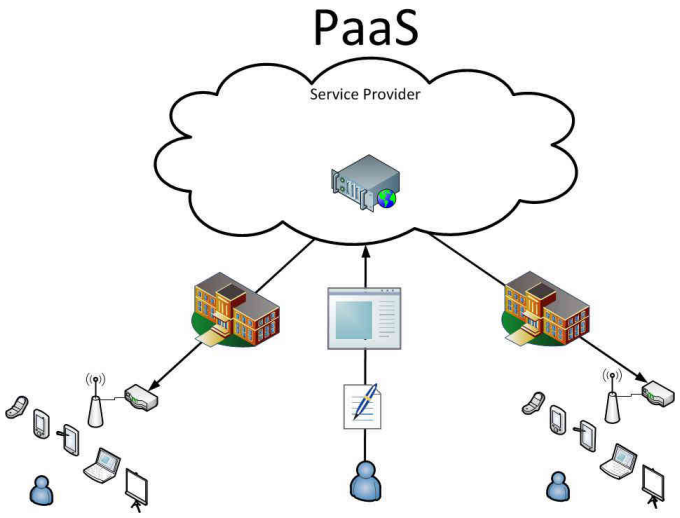

Figura 1.3: Modello architetturale PaaS

Inoltre i servizi vengono costantemente aggiornati e gli sviluppatori vengono assistiti dai provider sin dal concepimento della loro idea fino alla creazione dell'applicazione tramite strumenti di testing ed implementazione, e come con la maggior parte delle offerte cloud anche per i servizi PaaS l'utente paga solo ciò che usa.
Alcuni benefici di questa architettura sono:

1. *Nessun costo di investimento in infrastrutture fisiche.* Poichè viene virtualizzata l'intera infrastruttura si azzerano le spese relative alla parte hardware e alle "competenze" per gestirlo. L'utente sarà libero di concentrarsi sullo sviluppo e pagherà solo le risorse di cui avrà realmente bisogno, minimizzando lo spreco di eventuali risorse non utilizzate.

2. *Flessibilità.* È il consumatore a scegliere gli strumenti da installare all'interno della propria piattaforma, che viene adattata secondo specifiche esigenze.

3. *Adattabilità.* In qualunque momento l'utente può decidere di modificare qualsiasi funzionalità del servizio.

4. *Sicurezza.* Il provider "garantisce" sicurezza, backup e ripristino dei dati.

Uno degli esempi più noti di PaaS è Google App Engine. Per sfruttare questo servizio le applicazioni devono essere scritte in Python o Java utilizzando gli strumenti di sviluppo messi a disposizione da Google, tra cui quelli per utilizzare il Google filesystem e i data repository. Lo svantaggio di questa soluzione, però, è che si tende a diventare *prigionieri* del fornitore. Ad esempio, con Google, l'utente sarà costretto a scrivere le applicazioni solo nei linguaggi supportati, utilizzando delle API proprietarie. In questo modo verrà negata qualsiasi possibilità d'esecuzione di queste applicazioni al di fuori dell'infrastruttura offerta da Google.

1.3.5 Infrastructure as a Service (IaaS)

Il modello architetturale IaaS (vedi figura 1.4) rappresenta invece il livello di astrazione più basso. Si riferisce all'utilizzo dell'intera infrastruttura IT che l'utente può configurare con un livello di granularità molto fine, riuscendo così ad adattarla alle proprie esigenze. Le risorse hardware di questo tipo di cloud vengono assegnate e rilasciate al momento della richiesta in base alle reali esigenze.
Secondo il NIST:

> "L'infrastruttura IaaS definisce quel servizio che fornisce all'utente funzionalità di elaborazione, storage, connettività alla rete ed altre risorse di calcolo fondamentali."

Tutto questo si traduce nel concetto di *virtualizzazione dell'hardware.* Tramite la virtualizzazione è possibile suddividere una singola macchina fisica in varie macchine virtuali completamente indipendenti che, come tali, possono utilizzare sistemi operativi diversi e propri spazi di memoria, propri dischi e proprie CPU. Alcune tecnologie di virtualizzazione permettono persino di

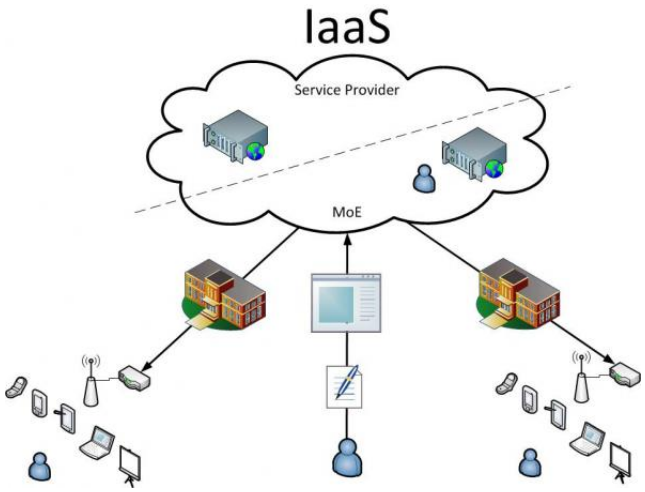

Figura 1.4: Modello architetturale IaaS

spostare un'istanza di un server virtuale, da un server fisico ad un altro, senza che l'istanza smetta di funzionare.
Per implementare la virtualizzazione vengono adottati i cosiddetti *sistemi di virtualizzazione*, che forniscono un gestore di macchine virtuali (chiamato in gergo *hypervisor*[12]) entro cui possono operare uno o più sistemi operativi ospiti. Una critica che spesso viene fatta a questi sistemi di virtualizzazione è che essi spesso, a parità di hardware, forniscono prestazioni inferiori. In ambiente di cloud computing questo non è sempre così per diversi motivi. Primo perché l'hardware utilizzato dai provider è decisamente potente e quindi le prestazioni che si ottengono sono generalmente migliori di quelle di un server fisico con minori prestazioni e gestito in maniera tradizionale. Inoltre, le tecnologie di virtualizzazione di fascia alta[13] utilizzate dai fornitori di servizi cloud si basano su funzionalità di para-virtualizzazione[14] direttamente a livello di CPU, e questo eguaglia le prestazioni delle macchine virtuali con quelle di server fisici di analoghe caratteristiche.

[12]Il compito dell'hypervisor è quello di astrarre l'hardware in maniera tale da permettere ai sistemi operativi ospite di condividere le risorse di uno stesso server fisico mantenendo quelli virtuali completamente isolati l'uno dall'altro.
[13]vedi Xen e VMware.
[14]Nel caso della paravirtualizzazione non c'è l'emulazione del processore e l'overhead è molto basso, ma un crash del sistema porterebbe in crash anche tutte le virtual machine

Un cliente di servizi IaaS non gestisce l'infrastruttura cloud sottostante la nuvola, ma ne controlla tutto il resto. Grazie ai diversi tipi di contratto messi a disposizione dai provider l'utente finale può scegliere l'hardware e il software, compreso il sistema operativo[15], che più si addicono alle sue esigenze.

1.3.6 Il modello SPI

Dei modelli sopra descritti, il Software as a Service (**S**aaS) insieme al Platform as a Service (**P**aaS) e all'Infrastructure as a Service (**I**aaS) costituiscono la classificazione più comune dei servizi cloud e formano il modello SPI, che sembra suggerire una classificazione netta dei servizi cloud ma che in realtà non lo è, poichè esiste un'ampia varietà di offerta di infrastrutture e piattaforme.

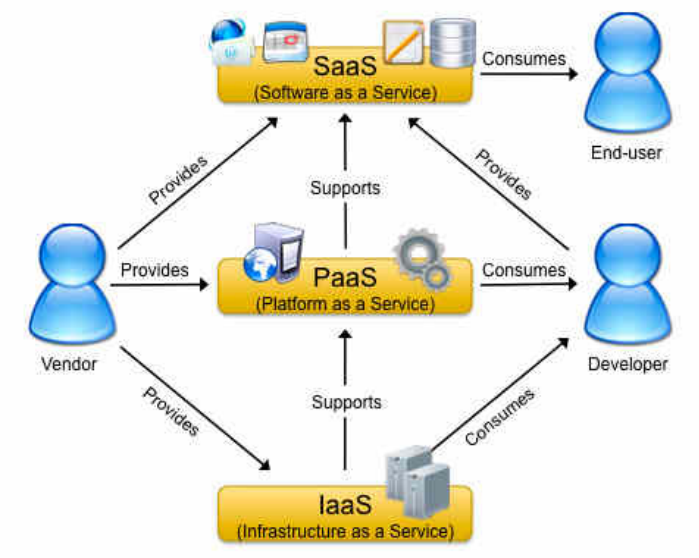

Figura 1.5: Relazione tra SaaS, PaaS, IaaS, fornitori e sviluppatori.

[15]Spesso i fornitori di servizi mettono a disposizione dell'utente delle immagini di macchine predefinite.

1.3.7 Business Process as a Service (BPaaS)

Anche se il NIST non lo considera appartenere alla categoria dei principali modelli di servizi cloud, negli anni è andato a svilupparsi anche un altro tipo di cloud: il Business Process as a Service (BPaaS). Questo si colloca in cima al modello SPI, è considerato l'evoluzione del SaaS ed è stato pensato per erogare funzionalità di business o di processo.
Le caratteristiche principali del BPaaS sono:

1. incapsulare ed offrire gran parte di quello che riguarda un modello di business
2. aggiungere pratiche industriali al livello di soluzioni SaaS
3. attuare modifiche al flusso di lavoro in maniera dinamica

1.4 Modelli di distribuzione

I cloud provider erogano i propri servizi secondo vari modelli in relazione alla tipologia di rete utilizzata dai consumatori. Dopo aver presentato il concetto di cloud computing, le sue caratteristiche e i modelli di infrastruttura, andremo ad elencare quali sono i modelli di distribuzione, ai cui estremi troviamo il cloud privato ed il cloud pubblico.

1.4.1 Cloud privato

Per cloud privato[16] si intende un sistema la cui infrastruttura è di proprietà di chi utilizza quel servizio. L'organizzazione stessa può decidere se lasciare la gestione del cloud in mano a terze parti, occuparsene personalmente all'interno dei propri locali oppure alle volte combinare queste due alternative. Tra i vantaggi di questo modello citiamo la sua scalabilità, il maggior rendimento della struttura IT ed una maggiore sicurezza della privacy. Ricordiamo infatti che l'azienda puó memorizzare i dati sensibili all'interno della propria struttura e questo implica una maggiore percezione di riservatezza

[16] Conosciuto anche come "*Internal Cloud*"

di questi ultimi. Al contrario abbiamo dei costi elevati di startup e relativi alla gestione dell'infrastruttura IT (alimentazione, condizionamento dei locali, sicurezza, controllo accessi, connettività). Un esempio di cloud privato è VmWare.

1.4.2 Cloud pubblico

Il cloud pubblico eroga servizi, tramite la rete, per aziende e privati. Questi servizi sono di proprietà del provider che provvede anche alla configurazione e gestione dell'infrastruttura cloud.
I vantaggi di questo modello riguardano l'efficienza dell'infrastruttura IT e minor costi, dovuti all'annullamento delle spese iniziali, al minor consumo di energia elettrica, all'eliminazione dei costi di gestione. Infatti è il provider ad assicurare l'assistenza di eventuali problemi tecnici.
Come già accennato nei paragrafi precedenti, uno dei maggiori problemi del cloud pubblico è quello relativo alla privacy delle informazioni che risiedono su server di proprietà del fornitore.

1.4.3 Cloud ibrido

Il modello ibrido è una soluzione intermedia ai precedenti. L'infrastruttura è composta da due o più modelli di cloud diversi (privato, di comunità, pubblico) che rimangono logicamente entità a se stanti, ma possono essere integrati tra loro tramite una tecnologia standardizzata o proprietaria che permette la portabilità di dati e applicazioni.
Questo modello rappresenta la soluzione ottimale sia per quelle aziende che possiedono già un'infrastruttura consolidata e che vogliono migrare verso un cloud pubblico, sia per quelle che vogliono mantenere alcune delle loro informazioni all'interno della propria struttura delegando la gestione di altre ad organizzazioni esterne.

1.4.4 Cloud di comunità

Il cloud di comunità rappresenta un'infrastruttura multi-tenant condivisa da più enti di una specifica organizzazione, che perseguono gli stessi obiettivi operativi e strategici, e che sono accumunati dalle stesse esigenze nei confronti dei provider, indipendentemente dai servizi richiesti. L'obiettivo di questo modello è quello di usufruire sia dei vantaggi del cloud pubblico, ad esempio il modello di fatturazione pay-as-you-go, sia dei vantaggi del cloud privato, come il maggior livello di privacy e sicurezza.
Può essere di proprietà, gestito dall'organizzazione in prima persona, da terze parti o da una combinazione di questi e può risiedere o all'interno o all'esterno dell'organizzazione.

1.4.5 Cloud privato virtuale

Con questo modello si presuppone che i servizi di cloud siano erogati in maniera isolata e completamente dedicata, pur avvalendosi di elementi infrastrutturali o di rete parzialmente condivisi.

Capitolo 2

Cloud Management Platforms

Dopo aver ripercorso la storia del cloud computing, averne dato una definizione e aver visto i vari modelli di infrastuttura e di distribuzione, in questo capitolo daremo una panoramica delle principali piattaforme di monitoraggio di sistemi di cloud computing ed analizzeremo più da vicino alcune tra quelle open source utilizzate per la gestione di Infrastructure as a Service, quali ***OpenStack***, ***OpenNebula*** ed ***Eucalyptus***.

2.1 Panoramica

Come già accennato nel precedente capitolo, negli ultimi anni il cloud computing è entrato in maniera invasiva nella vita quotidiana di molte persone grazie anche alla continua diffusione dei dispositivi mobili, quali smartphone, tablets etc.
Oggigiorno, molte sono le aziende che sviluppano ed offrono i propri servizi attraverso la nuvola, sia che questi siano di tipo SaaS, sia PaaS, che IaaS. Tra le più conosciute ed utilizzate ricordiamo Amazon, VMware, Microsoft Windows Azure, GoGrid, Google.

2.1.1 Amazon Elastic Compute Cloud

Amazon.com è stata una delle prime aziende che ha iniziato, nel 2006, ad offrire servizi di cloud computing. L'*Amazon Elastic Compute Cloud*, no-

to anche come “EC2”, è il sistema di erogazione di VMs on-demand con la possibilità di scegliere sistema operativo da utilizzare, ed applicazioni eventualmente pre-installate. Tramite la semplice interfaccia utilizzata, l'AWS Management Console, è possibile configurare con estrema facilità tutte le funzionalità offerte: abbiamo il controllo completo delle risorse di calcolo utilizzate e la possibilità di ottenere ed avviare in tempi brevi nuove istanze, nel momento in cui ce ne fosse bisogno. Il cliente paga soltanto ciò che usa e gli sviluppatori vengono forniti degli strumenti necessari per la realizzazione di applicazioni, in modo da evitare scenari di errori comuni.

Amazon EC2 fornisce una serie di funzionalità avanzate per la costruzione di applicazioni altamente scalabili e resistenti ai guasti:

- *Amazon Elastic Block Store*: è un servizio che permette di allocare volumi di spazio variabile, da associare poi ad un'istanza in esecuzione. Questo è uno storage persistente e rimane invariato tra un' esecuzione e l'altra di un'istanza. Un volume EBS può essere montato come secondo o ennesimo volume su un'istanza instance-store, oppure può essere esso stesso il disco di avvio di una istanza EC2 e, in tal caso, si parla di istanza EBS-backed.

- *EBS-Optimized Instances*: abilitano le istanze EC2 a sfruttare al meglio le operazioni di input/output su volumi EBS, fornendo un throughput dedicato tra Amazon EC2 e Amazon EBS che varia dai 500 ai 2000 Mbps.

- *Multi locazione*: garantisce continuità da fallimenti del datacenter ed una miglior latenza.

- *Indirizzo IP elastico*: sono progettati per il cloud computing dinamico ed associati non ad una singola istanza, ma al proprio account. Consentono di mascherare fallimenti d'instanze o Availability Zone rimappando l'indirizzo IP pubblico con un'altra istanza.

- *Amazon Virtual Private Cloud*: permette di configurare delle VPN per la gestione delle politiche di sicurezza durante l'esecuzione di risorse.

- *Amazon CloudWatch*: strumento che permette il monitoraggio di risorse ed applicazioni.
- *Auto Scaling*: è possibile aumentare o diminuire la quantità di risorse necessarie in un determinato istante, ottimizzando i livelli di performance o riducendo i costi.
- *Elastic Load Balancing*: il traffico in arrivo viene distribuito attraverso multiple istanze EC2.
- *High Performance Computing (HPC) Clusters*: clienti con carichi di lavoro molto pesanti possono personalizzare l'infrastruttura secondo le proprie esigenze, in modo da poter godere delle performance necessarie ad affrontare un determinato lavoro.
- *GPU Instances*: utilizzate dai clienti che necessitano di elevate performance di calcolo parallelo: applicazioni 3D, chimica computazionale, progettazione ingegneristica.
- *High I/O Instances*: utilizzate dai clienti che richiedono elevate operazioni di I/O, es. oltre le 100.000 IOPS. Queste istanze sono supportate dalla tecnologia Solid State Disk (SSD) e sono ideali per coloro che necessitano di alte prestazioni in database relazionali e NoSQL.
- *High Storage Instances*: per coloro che richiedono un'altissima densità di memorizzazione per istanza ed elevate operazioni di I/O sequenziale.
- *VM Import/Export*: per importare/esportare macchine virtuali.
- *AWS Marketplace*: negozio online per acquistare ed implementare rapidamente software che gira su AWS.
- *Enhanced Networking*: migliora le prestazioni nella comunicazione attraverso la rete (maggiori packet per second (PPS), riduce il jitter[1] di rete e la latenza.)

Per maggiori dettagli si rimanda alla documentazione ufficiale [16].

[1] Variazione statistica nel ritardo di ricezione dei pacchetti trasmessi, causata dalle code interne ai router congestionati.

2.1.2 VMware vSphere

VMware vSphere è una "suite" di software, funzionalità e servizi pensati per il cloud computing. Primo sistema operativo cloud del settore, VMware vSphere sfrutta la potenza della virtualizzazione per trasformare i datacenter in infrastrutture di cloud computing semplificate, consentendo alle organizzazioni IT di erogare servizi affidabili e flessibili di nuova generazione.
VMware vSphere riduce la complessità di gestione dell'hardware mediante la virtualizzazione totale di server, storage e hardware di rete, e fornisce semplici funzioni per l'High Availability in modo da fronteggiare i tempi di inattività non pianificati, ad esempio per guasti server.
Il "centro di comando" di VMware vSphere è il VMware vCenter Server, che consente l'amministrazione dei servizi applicativi e di infrastruttura, nonchè l'automazione delle attività operative quotidiane, assicurando massima visibilità su ogni aspetto dell'ambiente virtuale.
Per maggiori dettagli sulle funzionalità e principali servizi di VMware vSphere, si rimanda alla sezione 1.4 di [17].

2.1.3 Apache CloudStack

Apache CloudStack è una piattaforma open source capace di monitorare insiemi di risorse di storage, di rete e di eleborazione, al fine di realizzare sistemi IaaS pubblici e privati. Questa piattaforma supporta diversi hypervisor, tra cui BareMetal, Hyper-V, KVM, LXC, vSphere (via vCenter), Xenserver.
CloudStack permette la gestione di infrastrutture altamente scalabili, riuscendo a controllare decine di migliaia di server fisici distribuiti in diversi datacenter tramite l'utilizzo di API RESTful. Inoltre, fornisce delle API compatibili con quelle EC2, in modo da poter utilizzare gli strumenti più diffusi nella gestione degli AWS.
CloudStack fornisce due interfacce grafiche: una per gli amministratori ed usata per monitorare il sistema cloud, l'altra per gli utenti finali utilizzata per controllare le macchine virtuali.
Infine CloudStack mette a disposizione svariate funzionalità per incrementa-

re l'High Availability del sistema in caso di guasti.
Per maggiori dettagli si rimanda alla documentazione ufficiale.[18]

2.1.4 I servizi offerti da Google

Anche Google ha presentato i suoi servizi relativi al cloud computing:

1. ***Google App Engine***. Permette lo sviluppo e l'hosting di applicazioni web gestite dai Google Data Center; offre due ambienti di sviluppo, quali Java e Python.

2. ***Google Compute Engine***. Permette di creare ed avviare macchine virtuali con diverse configurazioni.

Inoltre Google ha sviluppato ***Ganeti***, un software scritto in Python per gestire in maniera semplice e ridondata cluster di macchine virtuali. Questo tool utilizza come hypervisor KVM o Xen, ed LVM (Logical Volume Management) per la gestione dei dischi.

2.1.5 GoGrid

Altra soluzione per cloud computing, con licenza proprietaria, è GoGrid. Fornisce un servizio di hosting mettendo a disposizione delle macchine virtuali di tipo Windows e Linux, in cui di default sono già preinstallate delle applicazioni, come ad esempio Java, C#, Apache, PHP, Microsoft SQL Server, e MySQL. La gestione dei server avviene mediante pannello di controllo multi-server e API con tecnologia RESTful.

2.1.6 Microsoft Azure

Denominata *Windows Azure* fino al 25 Marzo 2014, si tratta della piattaforma di cloud computing sviluppata da Microsoft, volta alla creazione, alla distribuzione e alla gestione di applicazioni e servizi sia PaaS che IaaS.
Per ulteriori informazioni si rimanda alla documentazione ufficiale [19].

Nelle seguenti sezioni analizzeremo tre piattaforme open source, quali l'emergente OpenStack, che insieme ad VMware sono le piattaforme più chiacchierate nelle grandi aziende o ISP, OpenNebula ed Eucalyptus. Tutte e tre abbracciano l'Open Cloud implementando le specifiche dello standard OCCI che verrà analizzato nel capitolo 4.

2.2 OpenStack

OpenStack è un progetto open source altamente scalabile pensato per l'amministrazione di servizi IaaS, rilasciato sotto licenza Apache e gestito dall'OpenStack Foundation. Lanciato nel 2010, viene sviluppato sia della NASA sia dell'azienda di hosting Rackspace. Tutto ha inizio quando Joshua McKenty[2] scrive nel suo blog:

> "*Lanciato Nova*[3]*. Apache-Licensed Cloud Computing, in Python. È vivo, è pieno di bug, è beta. Date un'occhiata* "

Tra quelli che leggono quel post c'è Rick Clark della Rackspace, che, caso vuole, stava lavorando proprio sullo sviluppo di una piattaforma open source[4]. Dopo aver contattato i suoi capi, tra cui il vicepresidente dell'azienda Jim Curry, figlio di un ingegnere della NASA, questi fissano un incontro con Chris C. Kemp, direttore del centro Ames dell'agenzia spaziale americana. A fine giornata si arriva ad un patto: avrebbero unito i due codici in un unico progetto e lo avrebbero chiamato OpenStack. Successivamente anche altre importanti aziende si sono offerte a supportare il progetto.
OpenStack è basato su di un'architettura a moduli(vedi figura 2.1), ciascuno indipendente dall'altro e pensato per una specifica funzionalità. Questi moduli sono lanciati su server distribuiti per ottimizzare la tolleranza ai guasti e la disponibilità di servizio, sono stati progettati secondo le linee guida della OpenStack Foundation in modo che possano lavorare insieme per realizzare

[2]Consulente della Nasa

[3]Sistema, rilasciato con licenza Apache, che permette di lanciare programmi senza scaricarli fisicamente sul proprio computer, molto simile all'EC2 di Amazon.

[4]Swift, uno storage virtuale (che faceva le veci del Simple Storage Service, S3, di Amazon).

un servizio completo di cloud computing. Questa cooperazione è garantita dall'utilizzo di specifiche API pubbliche.
Di seguito una piccola panoramica di ciascun modulo.

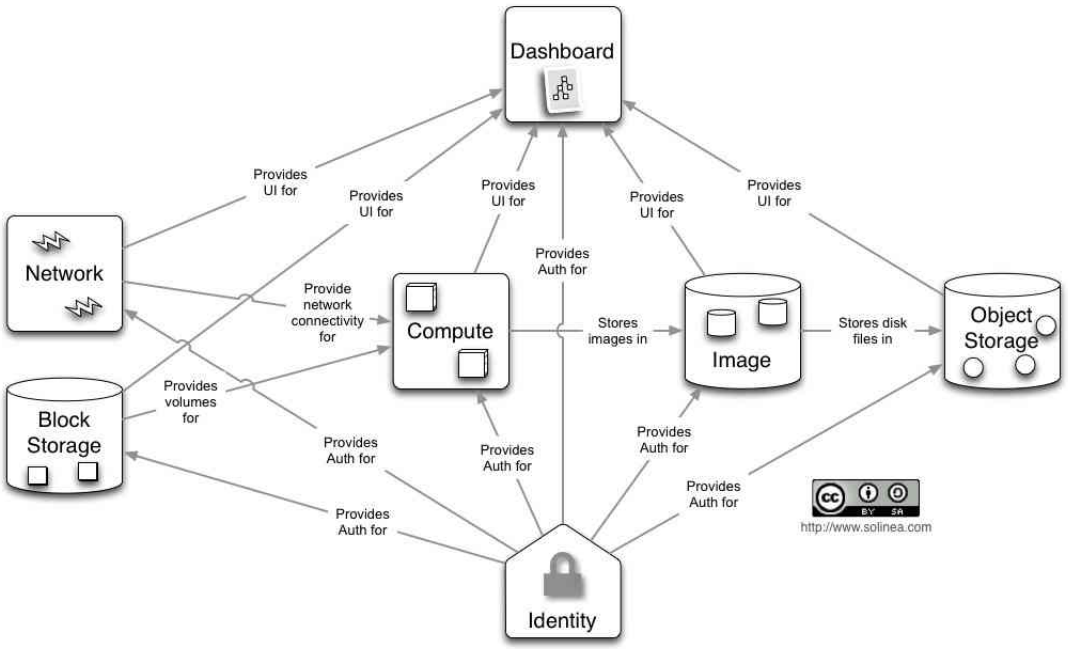

Figura 2.1: Diagramma concettuale dell'architettura di OpenStack

2.2.1 OpenStack Dashboard (*Horizon*)

È un'applicazione web sviluppata su Django[5] con lo scopo di dare supporto agli altri componenti di OpenStack. Fornisce ad amministratori e ad utenti un'interfaccia grafica per l'accesso, la fornitura e l'automatizzazione delle risorse dell'infrastruttura: ad esempio, con Horizon, si possono creare e gestire delle istanze virtuali, gestire la rete e l'archiviazione dei dati, etc. Inoltre gli sviluppatori possono implementare strumenti aggiuntivi sfruttando le API native di OpenStack oppure quelle compatibili con EC2.

[5]Framework Python open source

2.2.2 OpenStack Compute (*Nova*)

Nova è un controller scritto in Python, nonchè il componente più importante di un sistema IaaS. È stato progettato per la gestione e l'automazione di insiemi di risorse, come ad esempio le VMs, ed è in grado di lavorare con svariate tecnologie di virtualizzazione (KVM, XenServer, Hyper-V), così come con configurazioni di High Performance Computing (HPC). Tiene traccia sia delle risorse utilizzate sia quelle disponibili nell'intera infrastruttura: quando arriva ad esempio la richiesta di istanziazione di una macchina virtuale, Nova comunica con l'hypervisor sottostante, indicando a quest'ultimo di allocare le risorse necessarie. Stessa cosa vale sia per il rilascio delle risorse, che per la distruzione della macchina virtuale. Inoltre utilizza la libreria Eventlet per la programmazione concorrente, Kombu per la comunicazione AMQP ed SQLAlchemy per l'accesso al database.

2.2.3 OpenStack Networking (*Neutron*, in precedenza Quantum)

È il sistema che gestisce la rete, in particolare gli indirizzi IP (statici o dinamici). Fornisce diversi modelli di rete per diverse applicazioni e gruppi di utenti: i modelli standard sono le reti flat e le VLANs, utilizzate per la separazione di servers e traffico di rete. Gli utenti possono creare le proprie reti, controllarne il traffico e collegare servers e dispositivi. Gli amministratori, invece, possono utilizzare la tecnologia SDN (Software Defined Networks) per ottenere alti livelli di multi-tenancy e scalabilità di rete. Inoltre Neutron permette di estendere i propri servizi di rete, fornendo all'infrastruttura sistemi di rilevamento delle intrusioni (IDS), servizi di load balancing, firewall e VPN. Diverse sono le configurazioni di rete offerte da Neutron:

Single Flat Network

Risulta essere lo scenario più semplice in cui esiste una sola rete Openstack Neutron. Questo significa che la rete è condivisa ed è visibile a tutti gli utenti tramite le API di Openstack Networking. Gli utenti delle VMs hanno

un'unica scheda di rete e ricevono un indirizzo IP statico dalla sottorete associata a quella rete. Questo può essere associato ai modelli FlatManager e FlatDHCPManager forniti da OpenStack Compute.

Multiple Flat Network

Essenzialmente simile al precedente, eccetto che gli utenti hanno a disposizione più reti condivise da poter utilizzare.

Mixed Flat and Private Network

Questa configurazione è un'estensione delle due precedenti. Oltre ad avere l'accesso ad una o più reti condivise tramite Neutron, ogni tenant può creare delle reti aggiuntive condivise e visibili solo agli utenti di quella determinata istanza. Quando vengono create delle nuove istanze di VMs, queste possono avere accesso su una qualsiasi delle reti condivise su Neutron e/o su qualsiasi rete privata creata da un determinato tenant. Questo consente la creazione di topologie "multi-tier", utilizzando VMs con più schede di rete.

Provider Router with Private Networks

In questo scenario ogni tenant possiede una o più reti private tutte collegate al mondo esterno tramite un router in Neutron, gestito e di proprietà dell'amministratore. Il router mappa gli indirizzi pubblici della rete esterna in indirizzi fissi sulle reti private e fornisce connettività L3 tra quest'ultime in modo da consentire a tenant differenti di raggiungere gli altri, eccetto se non espressamente dichiarato da apposite regole. Questo è l'equivalente di VlanManager in Openstack Compute.

Per-tenant Routers with Private Networks

In questo scenario ogni tenant è collegato ad almeno un router e tramite le API di Neutron può decidere se crearne degli altri. Poichè ci sono più router gli indirizzi IP delle relative sottoreti si possono sovrapporre senza che questo crei dei conflitti, dal momento che l'accesso alla rete esterna avviene tramite SNAT o floating IP.

2.2.4 OpenStack Block Storage (*Cinder*)

Fornisce i volumi virtuali che vengono associati alle varie istanze di elaborazione. Gestisce la creazione, l'allocazione e la deallocazione dei dispositivi virtuali per i server. Questi interagiscono con l'OpenStack Compute e la Dashboard; proprio quest'ultima consente agli utenti di gestire le proprie esigenze di storage non solo in locale, ma anche tramite l'utilizzo di svariate piattaforme d'archiviazione, tra cui Ceph, CloudByte, Coraid etc.
Inoltre, questo modulo si occupa anche della gestione degli snapshot di dischi virtuali, funzionalità che risulta molto utile in caso di backup. La snapshot creata viene vista come un normale volume.

2.2.5 OpenStack Image Service (*Glance*)

È la componente che gestisce e recupera, tramite API RESTful, le immagini delle macchine virtuali ed anche gli snapshot di quest'ultime, nel caso in cui a seguito di eventuali problemi le immagini debbano essere ripristinate.
Il provider può inoltre creare dei template di immagini, dando la possibiltà a qualsiasi utente di avviarle come nuove istanze di elaborazione: queste sono le cosiddette immagini pubbliche. Le immagini private invece sono gli snapshot che l'utente crea e che potrà riutilizzare dal proprio account. Le immagini possono essere arhiviate in molti formati, tra cui raw, AMI, VHD (Hyper-V), VDI (VirtualBox), qcow2 (Qemu/KVM), VMDK (VMWare), OVF (VMWare, others), in modo da mantenere la compatibilità con svariati hypervisor.
Ogni immagine è contrassegnata da un identificativo univoco, detto *uuid*, e da uno stato di esecuzione tra queued, saving, active, killed, deleted, pending_delete.

2.2.6 OpenStack Object Storage (*Swift*)

È un sistema altamente scalabile e ridondante per l'archiviazione di dati. Questi vengono memorizzati su diversi cluster di server standardizzati, in modo da garantirne la replicazione e la distribuzione del loro contenuto in

caso di guasti al server o ai dischi. Swift non è visto come un file system tradizionale, ma appunto come un sistema di storage distribuito per oggetti, i quali hanno un proprio URL per essere reperiti.
Swift, come riportato in figura 2.2, è costituito da diversi componenti.

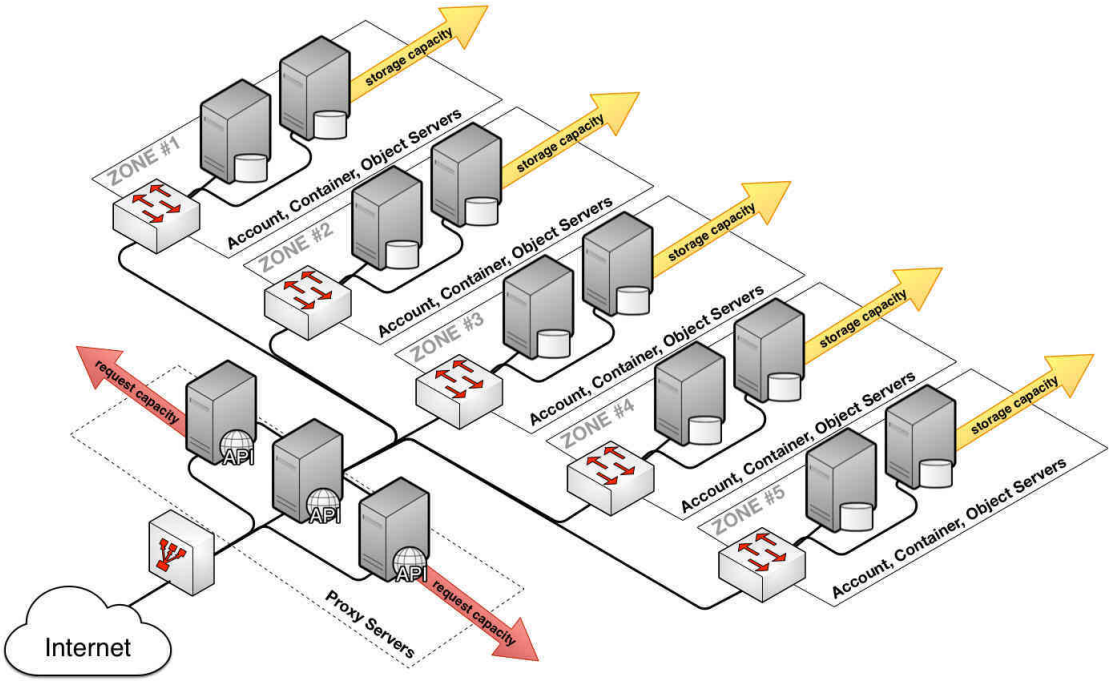

Figura 2.2: Architettura Swift

Possiamo distinguere tra:

- Proxy Servers
- Accounts & Containers
- Object Servers

I Proxy Servers rappresentano l'interfaccia pubblica di Swift e si occupano della gestione di tutte le richieste API in entrata. Quando arriva una richiesta per un oggetto il proxy determina la sua locazione e ne gestisce la risposta. Tutti gli oggetti vengono raggruppati in strutture logiche denominate *ring*, che si occupano di mapparli alla loro reale locazione utilizzando

come coordinate le zone, le partizioni e le repliche. La partizione è una collezione di dati, tra cui il database degli account, il database dei container e gli oggetti. Queste sono fondamentali per la replica dei dati, in quanto ogni partizione viene memorizzata per default su 3 dischi in zone diverse per evitare eventuali perdite di dati nel caso di guasti al sistema. Gli account e i container non sono altro che database SQLite distribuiti nel cluster. Un database di account contiene l'elenco di tutti i contenitori (es. le cartelle) di quell'account, mentre un database di container contiene la lista degli oggetti (es. i files) presenti sullo storage. Gli Object Servers, invece, rappresentano le macchine dedicate all'effettivo immagazzinamento dei dati.
Inoltre sia i cluster d'archiviazione sia i proxy servers sono scalabili orizzontalmente semplicemente aggiungendo nuovi server e senza nessuna interruzione di servizio.

2.2.7 OpenStack Identity (*Keystone*)

Questo modulo gestisce l'autenticazione e il controllo degli accessi degli utenti al sistema. Mantiene un elenco centralizzato degli utenti e dei privilegi che essi hanno sui diversi servizi del sistema di cloud computing. Supporta diverse modalità di autenticazione e, grazie alla sua architettura modulare, è possibile estenderle aggiungendo dei moduli appositamente sviluppati.
I meccanismi per l'autenticazione sono gestiti principalmente utilizzando tre entità, quali *users*, *tenants*, e *roles*. Gli users non sono altro che rappresentazioni digitali di persone, sistemi o servizi che si avvalgono dei servizi dell'infrastruttura cloud di OpenStack. La struttura dei tenants, invece, serve a raggruppare gli utenti e i servizi in un contesto comune, permettendo così di associare regole differenti agli stessi utenti in diversi tenants.
Keystone conferma una richiesta in ingresso tramite la validazione delle credenziali fornite dall'utente. Queste credenziali possono essere username/password o username/API key. Una volta verificata l'identità, l'Identity Service fornisce all'utente un token che sarà presentato ad ogni richiesta successiva. Il token può essere revocato e una volta scaduto l'utente dovrà riesibire le sue credenziali.

Riassumendo, Keystone fornisce i seguenti tipi di servizi:

- *Indentity Service*, per la convalida delle credenziali;
- *Policy Service*, che fornisce un motore di autorizzazione basato su regole;
- *Token Service*, che gestice e convalida i token;
- *Catalog Service*, che fornisce un endpoint registry per il discovery degli stessi.

Piccola nota sugli sviluppi di Keystone, è che si pensa di integrare, nelle prossime versioni di OpenStack, il supporto aggiuntivo di altri protocolli di autenticazione.

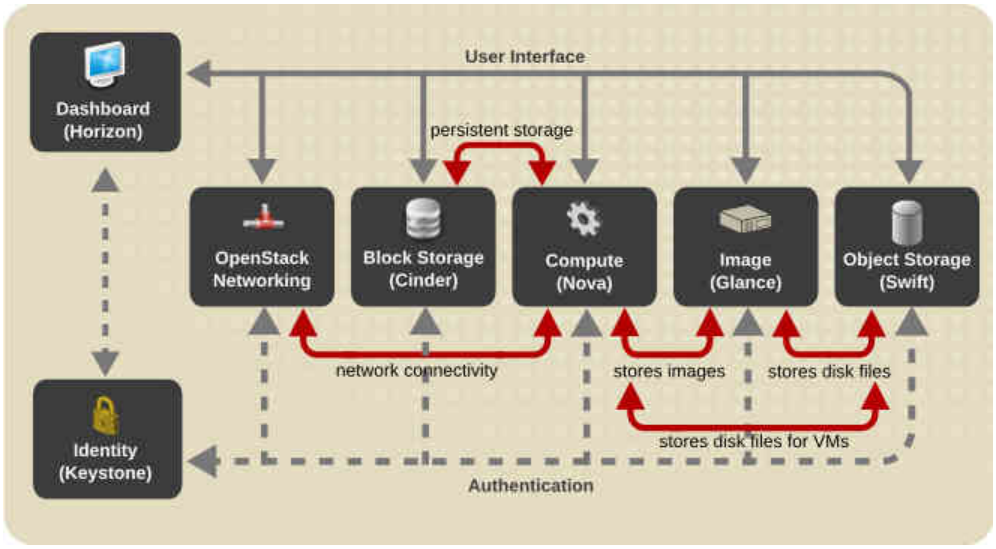

Figura 2.3: Interazione tra moduli

2.2.8 Altri moduli

Un sistema IaaS potrebbe essere realizzato semplicemente utilizzando i moduli sopra esposti. OpenStack, però, ne include degli altri, alcuni già rilasciati nelle ultime release, altri che devono essere dichiarati come ufficiali, ed altri ancora allo stato sperimentale.

Telemetry (*Ceilometer*)

È il programma che traccia tutto quello che succede nel cluster OpenStack: inizialmente pensato per fornire tutti i contatori necessari alla fatturazione del cliente, questo framework è comunque in grado di raccogliere informazioni da utilizzare per qualsiasi altra esigenza. Ceilometer, ad esempio, raccoglie e monitora diversi dati sulle attività di rete, di storage, di input/output, sulle risorse di tipo Compute.

Orchestration (*Heat*)

Questo modulo si avvale di template per creare la maggior parte dei tipi di risorse: strutture di macchine virtuali, volumi, utenti, ma anche progetti molto complessi.

Database (*Trove*)

Trove gestisce il provisioning di Database as a Service.

Altri moduli sono ***Sahara*** che implementa il Big Data, fornendo un'API comune per progetti tipo Hadoop; ***Ironic***, che verrà ufficializzato nella prossima release, Kilo, gestirà il Metal as a Service, cioè il provisioning di server fisici. ***Murano***, per gestire il catalogo delle applicazioni, ***Zaqar*** per gestire il queueing, ***Designate*** per il DNS as a Service, ***Manila*** per il Filesharing as a Service, e tanti altri.

2.3 OpenNebula

La prima versione open source di OpenNebula è stata rilasciata nel marzo del 2008. OpenNebula è uno strumento per la gestione di infrastrutture virtuali che utilizza varie tecnologie di storage, network, virtualizzazione e autenticazione per la realizzazione di modelli IaaS pubblici, privati ed ibridi.
Questa piattaforma scritta in C++, Ruby e Java, fornisce ad utenti e amministratori diverse interfacce ed hypervisor per garantire standardizzazio-

ne, interoperabilità e portabilità. OpenNebula è dotata di quattro diverse interfacce, dalle quali si possono gestire le risorse fisiche e virtuali:

- un'interfaccia per utenti, facilita loro l'esecuzione delle tipiche operazioni di gestione delle infrastrutture private ed ibride.
- un'interfaccia di amministrazione a linea di comando (CLI, command line interface) o grafica (Sunstone per amministratori) per utenti ed operatori esperti.
- API estensibili di basso livello per gli sviluppatori in Ruby, JAVA e XMLRPC.
- un marketplace con un proprio catalago di applicazioni virtuali pronte ad essere eseguite in ambienti OpenNebula.

L'architettura di questa piattaforma è altamente modulare in modo da offrire ampio supporto a livello aziendale per quel che riguarda l'hypervisor, il monitoraggio, lo storage, il networking e i servizi di gestione degli utenti. Inoltre, si presuppone che l'infrastruttura fisica adotti la classica architettura di *cluster computing* ed un insieme di hosts sui quali girano le VMs. Tutti gli hosts sono collegati al front-end tramite una rete fisica.
I componenti principali di un sistema basato su OpenNebula sono:

- la ***macchina front-end*** che contiene l'installazione di OpenNebula e che abilita l'utente ad eseguire i rispettivi servizi. Questa deve essere connessa con tutti gli altri hosts della rete e deve avere accesso ai datastores.
- gli ***hosts*** che eseguono gli hypervisors e forniscono le risorse di cui necessitano le VMs.
- i ***datastores*** che contengono le immagini delle VMs.
- le ***reti fisiche*** usate per supportare i servizi base come l'interconnessione dei server di archiviazione e le operazioni di controllo di OpenNebula.

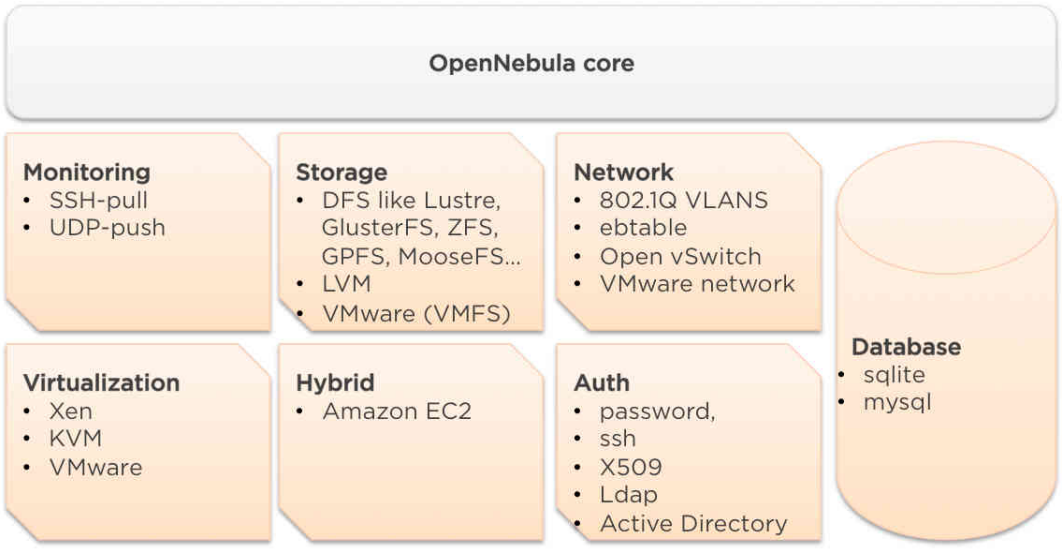

Figura 2.4: Panoramica dell'architettura di OpenNebula

2.3.1 Monitoraggio

Il sottosistema di monitoraggio raccoglie informazioni relative agli hosts del sistema e alle macchine virtuali e le invia alla macchina front-end in due differenti modalità:

- ***Modello UDP-push***. Ogni host manda periodicamente dei dati di monitoraggio alla macchina front-end, che a sua volta li raccoglie e li elabora tramite un apposito modulo.

- ***Modello Pull***. La macchina front-end interroga periodicamente gli hosts e successivamente esegue l'indagine tramite ssh.

2.3.2 Virtualizzazione

Il sottosistema di virtualizzazione comunica con l'hypervisor installato negli hosts ed esegue le azioni necessarie per ogni fase del ciclo di vita delle VMs. Nativamente OpenNebula supporta 3 hypervisor: Xen, KVM, VMware.

2.3.3 Storage

Per quanto riguarda il sottosistema di storage, OpenNebula utilizza dei Datastores per gestire le immagini disco delle VMs. Quando vengono caricate le VMs, le immagini sono trasferite dal Datastores alle macchine hosts e, a seconda dell'attuale tecnologia di memorrizzazione utilizzata, si ha un trasferimento reale, un link simbolico o la creazione di un LVM (Logical Volume Management).
OpenNebula gestisce 3 differenti classi di datastore:

- ***System Datastore***. Contiene le immagini per l'esecuzione delle macchine virtuali.

- ***Image Datastore***. Memorizza i repository delle immagini del disco. Queste immagini sono spostate, o clonate nel/dal system datastore quando le VMs vengono avviate/spente e a seconda della tecnologia utilizzata può essere di diversi tipi:

  - ***File-system***. Memorizza le immagini in formato file.
  - ***vmfs***. Formato utilizzato con hypervisor VMware.
  - ***LVM***. Permette di utilizzare i volumi LVM per contenere le immagini disco, invece che semplici file.
  - ***Ceph***. Le immagini disco vengono memorizzate tramite l'utilizzo di dispositivi a blocchi Ceph.

- ***File Datastore***. Utilizzato per memorizzare semplici file e non immagini.

2.3.4 Networking

Il sottosistema di rete è facilmente adattabile e personalizzabile al fine di una migliore integrazione con gli specifici requisiti di rete del Datacenter. Sono necessarie almeno due tipologie di reti fisiche:

- ***rete di servizio***, necessaria ai demoni della macchina front-end per accedere agli hosts. Questo è necessario per gestire e monitorare gli hypervisor e per spostare i file immagine.

- ***rete di istanza***, necessaria ad offrire connettività alle macchine virtuali tra i vari hosts.

L'amministratore può associare differenti driver per ogni host a seconda delle relative specifiche:

- ***dummy***, driver di default. Non esegue alcuna operazione di rete e anche le regole per il firewall vengono ignorate.

- ***fw***, vengono applicate le regole per il firewall ma l'isolamento della rete viene ignorato.

- ***802.1Q***, limita l'accesso alla rete tramite le VLANs. Necessita di supporto hardware.

- ***ebtables***, limita l'accesso alla rete tramite le regole di Ebtables. Non richiede nessuna configurazione hardware particolare.

- ***ovswitch***, limita l'accesso alla rete tramite Open vSwitch Virtual Switch[6].

- ***VMware***, questi driver utilizzano l'infrastruttura di rete di VMware per fornire alle VMs avviate con l'hypervisor VMware una rete isolata e compatibile con 802.1Q.

2.3.5 Autenticazione

Il sottosistema di autenticazione mette a disposizione i seguenti tipi di modello:

- Autenticazione tramite Username/Password (default).

- Autenticazione tramite ssh.

[6]Maggiori informazioni reperibili al sito http://openvswitch.org/

- Autenticazione tramite certificati x509.
- Autenticazione tramite protocollo LDAP (Lightweight Directory Access Protocol).

Inoltre, OpenNebula consente l'installazione di componenti aggiuntivi nel momento in cui il sistema cloud è pronto ed avviato.
Per maggiori informazioni si rimanda alla documentazione ufficiale[13].

2.4 Eucalyptus

Eucalyptus, acronimo di "Elastic Utility Computing Architecture for Linking Your Programs To Useful System", è un software open source per la realizzazione di sistemi cloud compatibili con gli Amazon Web Service, privati ed ibridi. È possibile creare delle infrastrutture IaaS astraendo risorse di elaborazione, di memorizzazione e di rete, le quali possono essere scalate dinamicamente a seconda del carico di lavoro dell'applicazione. Eucalyptus è stato acquistato a settembre 2014 da Hewlett-Packard.

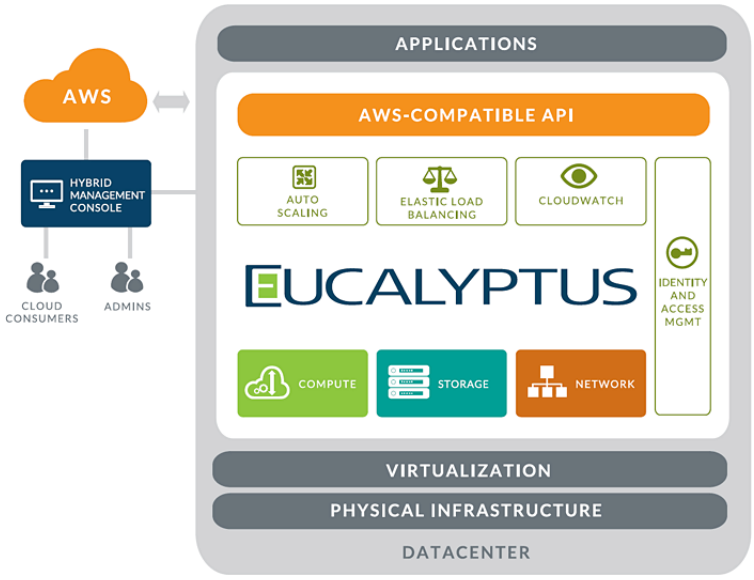

Figura 2.5: Sistema basato su piattaforma Eucalyptus

2.4.1 Architettura e componenti

L'architettura di Eucalyptus risulta essere altamente scalabile a causa della sua natura distribuita e sufficientemente flessibile per supportare aziende di qualsiasi dimensione.
Dalla figura 2.6 si possono contare tre differenti livelli, ognuno dei quali costituito da diversi componenti.

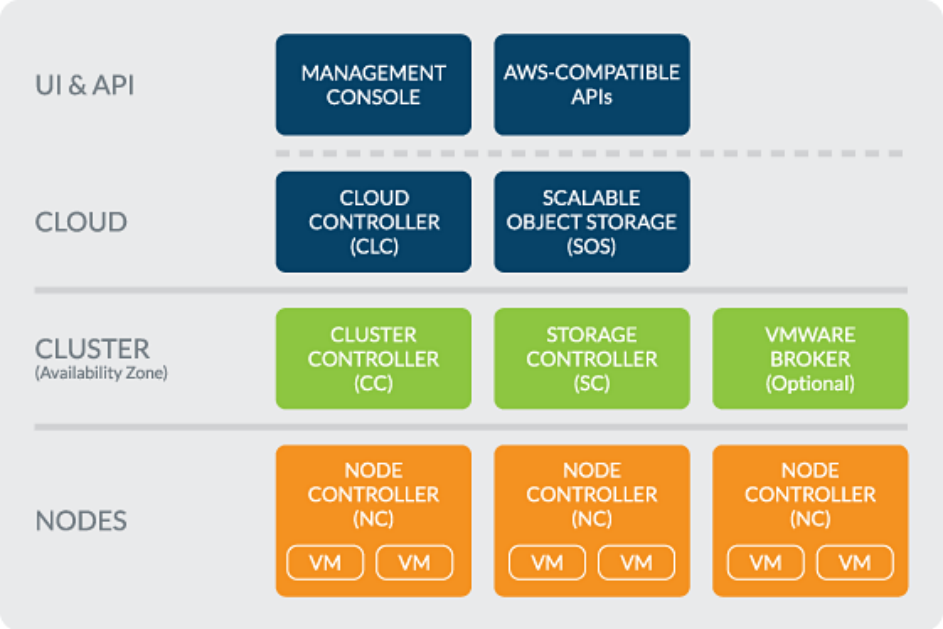

Figura 2.6: I componenti di Eucalyptus

Cloud Level

Questo livello è costituito da due componenti, quali il Cloud Controller(CLC) e lo Scalable Object Storage(SOS).
Il Cloud Controller può essere considerato come il punto d'ingresso al sistema per amministratori, sviluppatori e utenti normali. Offre un'interfaccia web, compatibile con l'EC2 di Amazon, ed accetta richieste API da parte dell'utente da linea di comando, come euca2ools, per la comunicazione all'esterno e all'interno dell'infrastruttura. Si occupa di comunicare con i Node Controller (NC) per reperire informazioni sulle varie risorse; inoltre, prende

ed attua decisioni di scheduling ad alto livello elaborando richieste per i Cluster Controller (CC). In poche parole gestisce le risorse virtuali sottostanti. Ogni sistema cloud accetta un solo CLC, le cui funzionalità ad alto livello si possono così riassumenere:

- Autenticazione
- Accounting
- Reportistica
- Gestione delle quote

Lo Scalable Object Storage (SOS), invece, è il sistema di archiviazione, l'equivalente al servizio Simple Storage Service (S3) di Amazon. Eucalyptus fornisce sia un'implementazione di archiviazione base, conosciuta come ***Walrus***, che soddisfa implementazioni di sistemi di piccole dimensioni, sia un'implementazione più sofisticata, conosciuta con il nome di ***RiakCS***, per sistemi di larga scala e che esiggono maggiori prestazioni.

Cluster Level

Il Cluster Level è composto dai Cluster Controller (CC), dagli Storage Controller (SC) e opzionalmente dal VMware Broker.
Un cluster è l'equivalente di una zona di disponibilità[7] di AWS, ed un sistema basato su Eucalyptus può averne diversi. Il CC è considerato il front-end che gestisce i diversi clusters del sistema e comunica con lo Storage Controller ed il Node Controller. Inoltre, gestisce l'esecuzione delle istanze di elaborazione, ad esempio le VMs, e si occupa di definire il Service Level Agreements (SLAs)[8] del cluster.
Lo Storage Controller è l'equivalente dell'Elastic Block Store(EBS) degli AWS. Questo comunica con il Cluster Controller ed il Node Controller e

[7]Amazon mette a disposizione un numero preciso di datacenter sparsi per il mondo: vengono usati i termini di zona geografica e zona di disponibilità. La prima identifica la zona fisica di dove si trovano i datacenter delle risorse che vogliamo acquistare, mentre la zona di disponibilità sono i datacenter ridondanti presenti nella stessa zona geografica.

[8]Governa l'uso di un servizio.

gestisce volumi e snapshots alle istanze facenti parte del suo cluster.
Infine, il VMware Broker fornisce un'interfaccia compatibile con AWS per ambienti VMware: media le interazioni con il CC e trasforma le Eucalyptus Machine Image (EMIs) in dischi virtuali supportati da VMware.

Node Level

Il Node Level, il livello più basso nell'architettura di Eucalyptus, è formato dai Node Controllers (NC). Il Node Controller ospita le VMs e gestisce gli endpoints delle reti virtuali. Inoltre, scarica e memorizza le immagini dallo Scalable Object Storage, così come crea e memorizza istanze di VMs.

2.4.2 Architetture di riferimento

Eucalyptus mette a disposizione due tipi di architettura a seconda dello specifico caso d'uso. Nonostante questo, però, non tutti i progetti di implementazione di un sistema potranno attuarle esattamente così come descritte, ma si prevede comunque che queste siano un buon punto di riferimento per avviare l'infrastruttura precedentemente studiata. I due modelli architetturali disponibili sono:

1. *General Purpose Large*
2. *General Purpose Small*

Fondamentalmente la differenza sostanziale tra queste due architetture sta nelle dimensioni dell'infrastruttura IT che si vuole realizzare, poichè a seconda di quale si sceglie si è condizionati a determinati limiti infrastrutturali. Comunque sia i benefici di entrambe sono gli stessi:

- Implementazioni robuste e ad alte prestazioni.
- Scalabilità su richiesta del carico di lavoro.
- Fornisce funzionalità di High Availability per una maggiore stabilità nel caso di malfunzionamenti delle risorse fisiche.

Per maggiori informazioni riguardo limitazioni/requisiti/configurazioni delle due architetture si rimanda alla documentazione ufficiale.[15]

2.4.3 Altre note

Eucalyptus permette di avviare multiple istanze di VMs che utilizzano diverse versioni sia del sistema operativo Linux che Windows. Le Amazon Machine Images (AMIs) sono compatibili con questo sistema, così come anche le VMware Images e le vApps, che possono essere convertite per essere avviate. Supporta diversi tipi di hypervisor, quali Xen, KVM e dal 2009 anche VMware.
Per quanto riguarda l'autenticazione degli utenti, Eucalyptus mette a disposizione delle funzionalità di controllo delle risorse virtuali che si basano su meccanismi di "*controllo degli accessi basato sui ruoli (RBAC)*".
Le interfacce per la gestione dell' autenticazione sono anche compatibili con le IAM API di Amazon.

Capitolo 3

Software Configuration Management

La nascita di nuove tecnologie ha fatto si che aumentasse anche il numero di server di un'organizzazione. Questo inevitabilmente ha incrementato la necessità di automazione del software e dei servizi in esecuzione nell'infrastruttura IT. In questo capitolo parleremo dei principali *Software Configuration Management* che accompagnano l'intero ciclo di vita di un sistema.

3.1 Concetto di DevOps

Innanzitutto, prima di cominciare ad analizzare da vicino i vari strumenti di gestione, diamo una breve definizione del concetto di *DevOps*. DevOps (da Development e Operations) è il principio secondo cui *sviluppatori* e addetti alle *operations* comunicano e collaborano nello sviluppo del software, per aiutare le organizzazioni a migliorare la produttività e la velocità di distribuzione del software, automatizzandone i flussi di lavoro. Questo movimento viene reso popolare tramite una serie di DevOps Day, iniziati nel 2009 in Belgio, per discutere sul divario operativo e lasciare che ingegneri del software potessero parlare dei modi migliori per raggiungere dimensioni e velocità ottimali nella distribuzione del codice. DevOps trova campo fertile in ambiente cloud: numerose aziende come SaltStack, AnsibleWorks, Opscode e Puppet

Labs incentivano il proprio lavoro nell'implementazione di strumenti software che rispecchiano le metodologie portate avanti dal movimento, la cui idea fondamentale è quella di "descrivere un'infrastruttura come codice".[20]

3.2 Software Configuration Management

Configuration Management è il processo utilizzato per la configurazione di server, e non solo, tramite la definizione a priori dello stato di questi ultimi. I *Software Configuration Management* leggono le configurazioni da file sorgenti e le applicano ai nodi in questione in maniera automatica, prevedibile ed idempotente. In questo modo l'ammistratore del sistema può replicare la stessa configurazione su più server, oppure ricostruirla in poco tempo nel caso in cui si riscontrassero dei problemi su uno o più nodi. Diversi sono gli strumenti per la gestione delle configurazioni, anche se non tutti hanno gli stessi obiettivi e/o lo stesso insieme di funzionalità.
Salt, rilasciato nel 2011 da SaltStack, è un tool molto scalabile in grado di gestire decine di migliaia di server. Si basa su CLI e la comunicazione è abbastanza semplice poichè avviene tramite SSH. I client, detti *minions*, ricevono i comandi dal server master e rispondono con i relativi risultati. I server master possono essere più di uno e distribuiti su più livelli in modo da distribuire il carico delle richieste, anche se così facendo si rischia di aumentare la ridondanza dei dati. Salt si può eseguire sia in modalità Standalone, nelle piccole infrastrutture o per ragioni di testing, sia in modalità Master/Agent, tramite la quale il client riceverà la propria configurazione attraverso i Salt States (SLS), inviati dalla macchina master. Salt supporta diverse distribuzioni: Arch Linux, Debian, Fedora, FreeBSD, Gentoo, OS X, RHEL e derivate, Solaris, Ubuntu, Windows, Suse. Per maggiori informazioni riguardo alle specifiche installazioni vedi http://docs.saltstack.com/en/latest/topics/installation.
Un altro strumento open source utilizzato per la gestione delle configurazioni e per l'automazione di un'infrastruttura IT è **Ansible**. Lanciato nel 2012, gestisce i nodi tramite SSH e necessita di Python (2.4 o successivi) per essere

installato su questi ultimi. Ansible ha un'architettura modulare che consente di poterlo estendere all'infinito: ogni modulo, che può essere scritto in qualsiasi linguaggio di programmazione, svolge una determinata funzione e gestisce un singolo aspetto di ogni sistema. I componenti principali sono:

- ***Inventario***. È la lista di server sui quali Ansible applica, a comando, le configurazioni di sistema e le istruzioni di automazione. Di default è contenuto all'interno del file */etc/ansible/hosts*, anche se è possibile specificare un percorso a piaecre.

- ***Tasks***. Sono una serie di istruzioni che Ansible esegue in ordine di apparizione.

- ***Handlers***. Sono delle istruzioni che vengono eseguite a seguito di una determinata azione. Gli Handlers si basano sul modulo *service*, utilizzato per gestire i servizi di sistema.

- ***Playbook***. Sono delle collezioni di Tasks e devono essere definiti in linguaggio YAML.

- ***Moduli***. Vengono utilizzati per la gestione di servizi cloud. Facilitano l'installazione di pacchetti software su server Linux con apt e yum, la gestione di file di configurazione e database etc.

A differenza di altri tools, Ansible, non richiede l'installazione di alcun agent[1] sui server, in quanto utilizza di default il trasporto SSH, eliminando così anche la necessità di installare software estraneo sui nodi. Può essere distribuito in ambienti di virtualizzazione e in ambienti di cloud pubblico e privato, quali VMware, OpenStack, AWS, Eucalyptus Cloud, KVM, e CloudStack. La macchina da cui vengono lanciati i comandi richiede l'installazione di Python 2.6 e supporta diverse distribuzioni basate su Linux e Unix, tra le quali ricordiamo Red Hat, Debian, CentOS, OS X e BSD; sistemi Windows non supportati. I nodi invece che devono essere gestiti richiedono l'installazione

[1]Piccolo software che rimane in esecuzione permanente su uno o più server ed ha lo scopo di attendere comandi e configurazioni impartite dal server master.

di Python 2.4 (o successivi). L'installazione di *python-simplejson* è necessaria se si eseguono versioni di Python precedenti alla 2.5. Inoltre viene fornito supporto per macchine Windows.
Attualmente i software maggiormente utilizzati sono due: ***Puppet*** e ***Chef***.

3.2.1 Puppet

Puppet, rilasciato nel 2005 sotto licenza Apache 2.0 dalla Puppet Labs, è un'utility open source scritta in Ruby per l'automazione IT: consente di gestire l'infrastruttura automatizzando task come il provisioning, la configurazione, la conformità e la gestione del software.
La configurazione di sistemi tramite Puppet avviene in due fasi principali: la prima è quella di redigere un catalogo e la seconda è quella di applicare le eventuali modifiche allo stato delle risorse descritte in esso, nel caso in cui queste non fossero nello stato desiderato. Il catalogo non è altro che un documento che descrive lo stato desiderato per un determinato nodo. Qui vengono elencate tutte le risorse che bisogna gestire, nonchè eventuali dipendenze tra di esse. Il funzionamento di Puppet si basa su due diverse modalità:

- ***Architettura Agent/Master (default)***. In questa architettura i nodi che devono essere gestiti lanciano in background il demone *puppet agent*, mentre la/e macchina/e server esegue/ono il *puppet master*. Periodicamente il puppet agent invierà al puppet master una richiesta di configurazione del nodo, il quale redigerà un catalogo secondo cui il nodo dovrebbe essere configurato e lo rimanderà indietro al client. Una volta ricevuto questo catalogo, il client verificherà che ogni risorsa descritta in esso si trovi nello stato desiderato, e se così non fosse apporterà le modifiche necessarie. Successivamente segnalerà al puppet master tutti i cambiamenti che sono stati effettuati alla configurazione del nodo.
  La comunicazione tra master e agent è crittografata: il client infatti genera una chiave autofirmata e la manda al master, che la verifica. Dopo di che, sempre il client, inoltra una richiesta di certificato che il master,

che funge anche da "autorità di certificazione", deve convalidare prima di stabilire la connesione sicura.

- ***Architettura Standalone***. Con questa modalità, invece, ogni server possiede la copia completa di tutte le informazioni di configurazione e quindi redige il proprio catalogo. Sul nodo da gestire viene eseguito il demone *puppet apply*. Questo per redigere il catalogo ha bisogno dei permessi necessari per accedere ai dati di configurazione, proprio come il puppet master. Redatto il catalogo, il server controllerà le risorse descritte in esso e ne modificherà lo stato, nel caso in cui queste non fossero nello stato desiderato. Successivamente il puppet apply memorizzerà tutti i cambiamenti su disco. È anche possibile configurarlo in modo che questi "report" vengano mandati ad un servizio centrale.

In entrambe le architetture se si è in no-op mode (simulazione), verrà simulata la nuova configurazione senza apportare nessun cambiamento al sistema. Puppet utilizza un linguaggio dichiarativo e la sua architettura è modulare: è costituito da piú di 2700 moduli pre-costruiti, tutti disponibili per il download gratuito dal Puppet Forge[2], per automatizzare le più comuni attivita di gestione, quali:

- installazione e configurazione di Apache, oltre alla configurazione e gestione di una vasta gamma di configurazioni di hosts virtuali;
- gestione di sorgenti APT;
- installazione, configurazione ed esecuzione di NTP[3];
- gestione e configurazione di firewalls;
- installazione e configurazione di MySQL;
- e altro ancora.

[2]https://forge.puppetlabs.com/

[3]Network Time Protocol, protocollo per sincronizzare gli orologi dei computer all'interno di una rete a commutazione di pacchetto

È inoltre possibile definire nuovi moduli in base alle proprie esigenze tramite il DSL di Puppet.
Oltre alla versione open source, Puppet è disponibile anche nella versione Enterprise: si tratta di un prodotto commerciale che integra più di 40 altri servizi open source per creare una piattaforma completa per l'automazione IT.
Diverse sono le piattaforme supportate: Puppet può essere eseguito su RHEL, e derivate quali CentOS, Scientific Linux, Oracle Linux, and Ascendos; Debian ed Ubuntu, Fedora, Microsoft Windows e Mac OS X. Supporta inoltre altre distribuzioni, anche se Puppet Labs non fornisce nè pacchetti ufficiali, nè alcun testing automatizzato: citiamo SUSE Linux Enterprise Server (versione 11 e successive), Gentoo Linux, Mandriva Corporate Server 4, ArchLinux, Oracle Solaris (versione 10 e successive), FreeBSD 4.7 (e successive), OpenBSD 4.1 (e successive). Per approfondimenti sui requisiti di sistema e informazioni dettagliate sulle piattaforme supportate, si consiglia la lettura di https://docs.puppetlabs.com/puppet/latest/reference/system_requirements.html#platforms-with-packages. Per maggiori informazioni riguardo la configurazione e l'utilizzo di Puppet si rimanda alla documentazione ufficiale [24].

3.2.2 Chef

Anche Chef, come Puppet, è un tool per la gestione delle configurazioni di un'infrastruttura IT, distribuito sotto la licenza Apache 2.0 e scritto in Ruby (lato client) e in Erlang[4] (lato server). Chef si basa su semplici concetti, quali il raggiungimento dello stato di sistema desiderato, la pianificazione centralizzata dell'infrastruttura IT e l'aggregazione di semplici risorse per realizzare sistemi complessi. Si basa anch'esso su due modalità di funzionamento:

- ***Architettura Standalone***. Permette di utilizzare i cookbooks senza richiedere alcun accesso allo Chef server. Il demone chef-solo viene eseguito in locale e richiede che: sia i cookbooks sia le relative dipendenze

[4]http://it.wikipedia.org/wiki/Erlang_(linguaggio_di_programmazione)

vengano memorizzati sul disco fisico di quel nodo. Non supporta tutto l'insieme di funzionalità che invece mette a disposizione chef-client.

- ***Architettura Client/Server***. In quest'architettura, lo Chef server rappresenta il repository centrale di tutti i dati, le "*ricette*", di configurazione. Qualsiasi nodo che esegue chef-client, invece, può essere gestito tramite il sistema Chef. Chef-client esegue tutte le attività di configurazione specificate dalla run-list, prelevando tutti i dati di configurazione necessari dal server centrale. La comunicazione tra client e server viene resa sicura tramite l'utilizzo di una combinazione di chiavi pubbliche e private.

Componenti di Chef

Come si nota dall'immagine 3.1, numerosi sono i componenti che permettono di gestire un sistema con Chef. Questi lavorano insieme per fornire informazioni e istruzioni a chef-client, affinchè questo possa eseguire il proprio lavoro. I principali componenti di Chef sono:

- ***chef-client***. È l'agent che viene eseguito su ogni nodo che deve essere gestito da Chef. Questo esegue tutte le operazioni necessarie per portare il nodo nello stato richiesto. Rappresenta il componente principale installato sui nodi insieme ad *ohai*, tool utilizzato per individuare gli attributi dei nodi (utilizzo della rete, della memoria, della CPU etc.)

- ***Workstation***. È la macchina che esegue *knife* e che viene utilizzata per controllare un singolo Chef server. Knife è uno strumento a linea di comando utilizzato per gestire nodi, cookbooks e ricette, risorse cloud etc.

- ***Chef server***. Raccoglie tutti i dati di configurazione necessari a chef-client per configurare un nodo. Memorizza cookbooks, le politiche che devono essere applicate ai nodi e i metadati che li descrivono.
  I *cookbooks* costituiscono la parte fondamentale per la distribuzione delle politiche di configurazione. Questi definiscono come deve essere

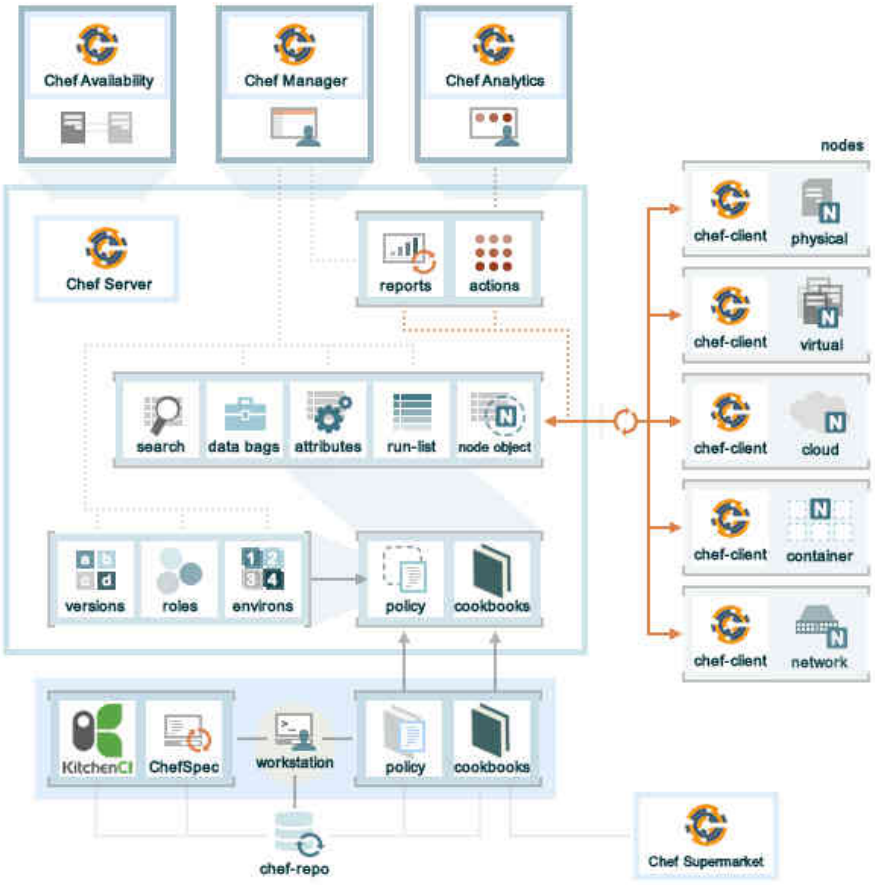

Figura 3.1: Relazione tra i componenti di Chef

configurato un nodo, attraverso l'utilizzo di attributi e "*ricette*"; quest'ultime, scritte in Ruby, devono definire qualsiasi cosa sia necessaria per configurare una specifica parte del sistema. Due componenti molto importanti del server sono gli *attributi* e le *run-list*: i primi rappresentano specifici dettagli di un nodo, quali lo stato corrente, lo stato precedente e successivo all'esecuzione di chef-client e vengono definiti dallo stato del nodo stesso, dai cookbooks, dalle varie regole. Le run-list, invece, definiscono tutte le impostazioni di configurazione da applicare al nodo, un elenco ordinato di regole da eseguire, secondo l'ordine di apparizione.

- ***Chef Supermarket***. È il luogo in cui i cookbooks della comunità

vengono creati e mantenuti.

Inoltre, Chef server mette a disposizione dell'utente delle funzionalità premium che possono essere facilmente abilitate ed integrate, quali:

- ***Chef Manager***. Fornisce un'interfaccia web-based per la gestione di attributi, run-list, roles, cookbooks etc. che sono memorizzati sul server.
- ***Chef Analytics***. Fornisce la possibilità di visualizzare a runtime tutto ciò che avviene sul server.
- ***Chef Availability***. Si occupa di replicare lo Chef server, in modo da fornire supporto per l'High Availability.

Chef può essere integrato con le piattaforme di gestione di sistemi cloud, come Rackspace, Amazon EC2, Google Cloud Platform, OpenStack, Microsof Azure, per configurare automaticamente nuove macchine.
chef-client è supportato da diverse distribuzioni Linux, Unix, e Microsoft Windows, tra le quali troviamo Ubuntu, Debian, RHEL/CentOS, Fedora, Mac OS X, Solaris, FreeBSD, Windows Server. chef-server supporta invece solo alcune versioni di CentOS, Oracle Linux, RHEL ed Ubuntu. Per una lista completa e dettagliata delle piattoforme supportate, vedi https://docs.getchef.com/supported_platforms.html, mentre per i requisiti di sistema si consiglia la lettura di https://docs.getchef.com/chef_system_requirements.html#chef-client. Per maggiori informazioni riguardo la configurazione e l'utilizzo di Chef si rimanda alla documentazione ufficiale [25].

Capitolo 4

Open Standards

Negli anni sempre più provider hanno iniziato ad offrire servizi di cloud computing. L'adozione da parte di ognuno di questi di una propria piattaforma di gestione ha portato a dei problemi di comunicazione tra queste diverse realtà. In risposta a questo problema alcune community non solo hanno iniziato a sviluppare dei progetti open source come OpenStack, che rendono più omogeneo l'utilizzo dei servizi di diversi provider, ma hanno dato vita a dei veri e propri standards in modo da avere un'interfaccia comune di facile gestione. In questo capitolo faremo una panoramica degli standards che sono in fase di sviluppo ed analizzeremo più da vicino uno di questi, ovvero lo standard OCCI (Open Cloud Computing Interface).

4.1 Panoramica

Immaginiamo di dover migrare i nostri servizi da un provider ad un altro: senza alcuno standard il cliente dovrebbe dapprima convertire i propri dati per poi riutilizzarli su di un'altra infrastruttura, così come senza un'interfaccia standardizzata non si avrebbe quell'automazione necessaria, ad esempio, per migrare un gran numero di VMs da un provider all'altro. È necessario quindi garantire interoperabiltà e portabilità per ovviare a queste limitazioni. Gli *Open Standards* hanno l'obiettivo di fornire maggiore libertà operativa ai fruitori di questi servizi, per arrivare ad una condivisa, efficace ed efficiente

soluzione di "*Open Cloud*" non soggetta a soluzioni proprietarie, in modo da abbatterne tutti i relativi costi e problemi di licenze, diritti, etc.
L'Open Cloud dovrebbe rappresentare dati e metadati in un formato Open Standard e fornire un'interfaccia standardizzata per il controllo di tutte le funzionalità. Il cliente in questo modo potrà usufruire di qualsiasi servizio che desidera, e abbandonarlo in qualsiasi momento senza alcuna limitazione. Le risorse dovrebbero essere rappresentate tutte allo stesso modo e gestite tramite queste APIs standard, mentre i dati dovrebbero avere un formato comune in modo tale da poter essere gestiti da diversi servizi di diversi provider. OCCI è considerato, insieme anche all'Open Virtualization Format (OVF), uno standard approvato sul mercato per l'interoperabilità e portabilità dei dati.
Un punto chiave nella scelta di questo standard, oltre al fatto che sia open, è la sua capacità d'integrazione[32] con altri standard: un caso comune è quello di avere un servizio distribuito su più VMs ognuna con il suo volume di storage e tutte interconnesse tramite rete. Lo standard OCCI, ad esempio, può essere integrato con l'Open Virtualizzation Format(OVF) e il Cloud Data Management Interface(CDMI) in modo da fornire l'interfaccia comune che gestisce le suddette risorse.
Open Virtualizzation Format(OVF) è lo standard di DMTF per la pacchettizzazione, la distribuzione e l'impiego di risorse virtualizzate. Viene utilizzato per dispositivi virtuali, applicazioni preconfigurate installate su un insieme di macchine virtuali. Queste risorse virtuali pacchettizzate agiscono come dei modelli o servizi plug and play risultando indipendenti dalla piattaforma su cui girano, quindi non ci sarebbe alcuna differenza se al livello sottostante ci fosse un hypervisor Xen o VMware. Questo è alla base della migrazione di macchine virtuali da un provider all'altro.
Cloud Data Management Interface(CDMI), sviluppato da SNIA, fornisce un formato per descrivere dati, metadati ed operazioni di packaging e trasferimento di questi ultimi attraverso la nuvola. Fornisce l'interfaccia tramite la quale vengono eseguite le operazioni Create, Retrieve, Update e Delete, abilitando l'utente alla gestione di dati.
Inoltre, come vedremo nelle prossime sezioni, lo standard OCCI è già suppor-

tato da molte piattaforme di gestione di ambienti cloud, tra cui OpenNebula ed OpenStack.
Ricapitolando, gli standards già definiti o in corso di completamento sono:

- **DMTF OVF, Open Virtualization Format**. Formato standard grazie al quale è possibile migrare da un ambiente virtuale ad un altro. Anche se accettato dalla maggior parte dei fornitori per la virtualizzazione ed il cloud, per ora la conversione dei dati in questo formato non è così semplice e facile. Per approfondimenti si rimanda a http://www.dmtf.org/standards/ovf.

- **DMTF CIMI, Cloud Infrastructure Management Interface**. Definisce un modello logico per la gestione delle risorse ICT come un "service domain".

- **DMTF CADF, Cloud Auditing Data Federation Work Group**. Altro gruppo DMTF che ha emesso un primo documento di inquadramento sulla federazione tra cloud di fornitori diversi, sia pubblici che ibridi.

- **CDMI, Cloud Data Management Interface**. Sviluppato da SNIA, definisce protocolli ed API per accedere e gestire dati archiviati in un sistema cloud.

- **IEEE P2302, Draft Standard for Intercloud Interoperability and Federation**. Definisce topologie, funzionalità e governance per l'interoperabilità e la federazione cloud-to-cloud.
  Per approfondimenti si rimanda a http://grouper.ieee.org/groups/2302/, http://standards.ieee.org/develop/wg/ICWG-2302_WG.html.

- **OASIS TOSCA Draft 0.2, "Topology and Orchestration Specification for Cloud Applications", Marzo 2012**. Focalizzato sulla portabilità di servizi ed applicazioni cloud.

Per maggiori dettagli vedi sezione 2.2.4 di [26].

4.2 Open Cloud Computing Interface

L' **Open Cloud Computing Interface** definisce un protocollo ed un set di APIs necessari per ogni attività di gestione, da remoto, delle infrastrutture cloud. Comprende un insieme di specifiche, sviluppate attraverso l'Open Grid Forum, che definiscono il modo in cui i diversi fornitori devono erogare i propri servizi, attraverso un'interfaccia comune e standardizzata. Si basa sui principi del World Wide Web offrendo un modello estensibile per l'interazione con i servizi "as a Service".
Fornisce un'API RESTful molto semplice: attraverso un URL l'utente può identificare uno specifico provider e relative risorse da controllare, che possono essere istanze di elaborazione, di storage o reti. Ad ognuna di queste vengono associati degli attributi; insiemi di risorse possono essere collegati tra di loro e gestiti tramite le tipiche operazioni di start, stop, update e delete, utilizzando i metoti GET e POST dell'HTTP.

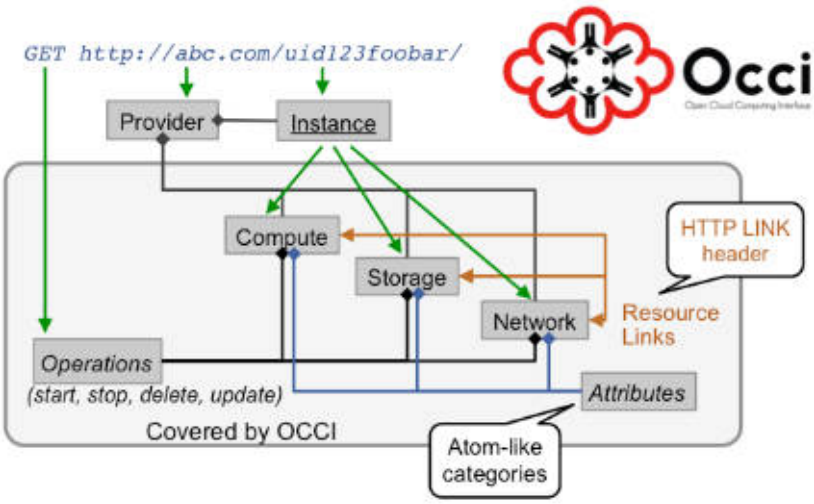

Figura 4.1: Esempio di utilizzo dello standard OCCI

Il progetto OCCI nasce nel marzo del 2009 guidato dagli allora co-presidenti di SUN Microsystem, RabbitMQ e dell'Università Complutense di Madrid, ed inizialmente è pensato unicamente per la gestione remota di servizi basati sul modello IaaS. Ad oggi, il numero delle adesioni al progetto supera i 250

membri, tra industrie, istutuzioni accademiche e di ricerca e semplici individui. Giusto per citarne alcuni, allo sviluppo di questo standard contribuiscono Rackspace, Oracle, Platform Computing, GoGrid, Cisco, Flexiscale, ElasticHosts, CloudCentral, RabbitMQ, CohesiveFT, CloudCentral,SLA@SOI, RESERVOIR, Claudia Project, OpenStack, OpenNebula, DGSI.
Ad oggi si possono contare più di 20 implementazioni, tra cui quelle di OpenNebula.org, RESERVOIR, SLA@SOI, e OpenStack, del quale è diventata parte integrante del progetto a partire dalla release del marzo 2012, OpenStack Essex.
L'architettura di OCCI è modulare ed estensibile. Attualmente lo standard è alla versione 1.1, ed è costituito da tre moduli:

- ***Open Cloud Computing Interface - Core***. Rappresenta il nucleo di questo standard. Fornisce mezzi e semantica necessari per la gestione e la definizione di risorse e classi di risorse. Questo modello è libero dagli altri e può essere utilizzato come componente autonomo in altri contesti (es. architetture ROA).

- ***Open Cloud Computing Interface - Infrastructure***. Rappresenta le risorse dell'infrastruttura e i collegamenti tra quest'ultime, estendendo rispettivamente le classi Resource e Link del modello Core.

- ***Open Cloud Computing Interface - RESTful HTTP Rendering***. Descrive un formato serializzato che viene utilizzato nella comunicazione tra client e servizi.

Nelle prossime sezioni tratteremo in maniera dettagliata i moduli sopra elencati.

Diverse sono le motivazioni che hanno portato all'implementazione dello standard OCCI:

1. ***Interoperabilità***. Consentire ai diversi fornitori di servizi cloud di interagire tra loro, senza dover modificare il formato o lo schema dei propri dati, oppure essere dipendenti dalle proprie API.

2. ***Portabilità***. Il cliente potrà con semplicità muovere i propri servizi da un provider all'altro, secondo le esigenze del momento (ad esempio per una riduzione dei costi).

3. ***Integrazione***. Le implementazioni di queste specifiche possono essere facilmente integrate con middleware[1] esistenti, con software di terze parti e altre applicazioni.

4. ***Innovazione***. L'open standard può essere un motore di innovazioni, così come quest'ultime possono richiedere questi standard.

5. ***Riusabilità***. Questo può essere visto su due livelli: da una parte il riutilizzo del codice attraverso API standardizzate, e dall'altra il riuso dello standard stesso in più campi.

4.3 OCCI - Core

Questo modello definisce rappresentazioni di tipi di istanza, in maniera tale che queste possano essere gestite tramite le implementazioni del modulo di rendering. Si tratta di un'astrazione delle risorse reali che permette l'identificazione, la classificazione, l'associazione e l'estensione delle stesse.
Una caratteristica fondamentale di questo modello è che può essere esteso, in maniera tale che ogni sua estensione sarà visibile al client OCCI a runtime. In poche parole un client OCCI può connettersi ad un'implementazione OCCI con modulo Core esteso, nonostante non abbia in anticipo alcuna informazione riguardo le sue estensioni, ma semplicemente sarà in grado di analizzare le varie risorse facenti parte il sistema in fase d'esecuzione.
Il diagramma di figura 4.2, anche se non rappesenta una completa definizione, dà una panoramica del modulo Core.
Ogni risorsa in OCCI, come ad esempio una macchina virtuale, viene gestita dalla classe ***Resource***, o da una sua sottoclasse. La classe Resource è costituita da una serie di attributi comuni che vengono ereditati anche dalle sue

[1] Con middleware si intende un insieme di programmi informatici che fungono da intermediari tra diverse applicazioni e componenti software. Sono spesso utilizzati come supporto per sistemi distribuiti complessi.

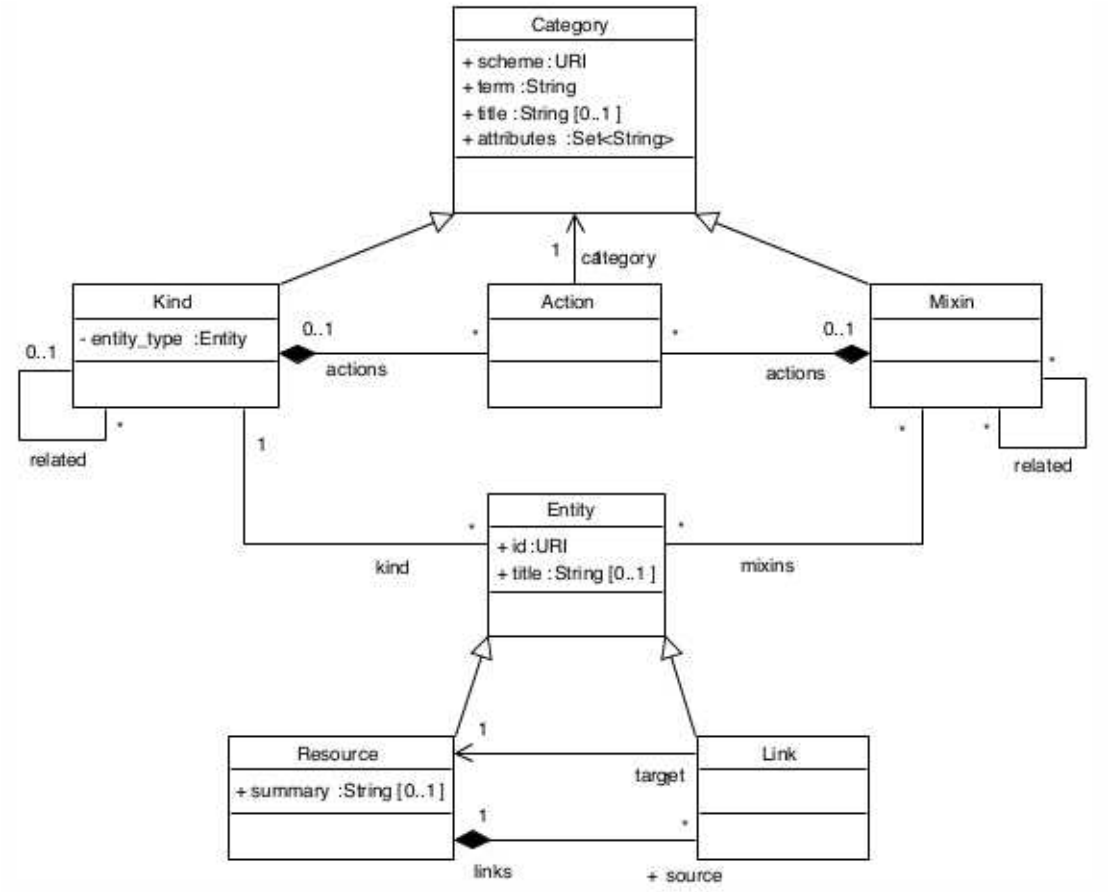

Figura 4.2: Panoramica del modulo OCCI Core

sottoclassi; grazie alla classe ***Link*** è possibile associare più istanze Resource. Anche la classe Link è costituita da attributi comuni che vengono ereditati dalle sue sottoclassi.

Entity è una classe astratta, dalla quale ereditano sia la classe Resource sia la classe Link. Ogni sua sottoclasse è identificata da un'unica istanza Kind. Quest'ultima è la base per il sistema di classificazione: è una specializzazione della classe ***Category*** e introduce funzionalità aggiuntive, operazioni invocabili sulle istanze di risorse rappresentate dalla classe ***Action***. Infine, troviamo la classe ***Mixin***: questa completa la classe Kind nella definizione del sistema di classificazione del modello OCCI Core.

4.3.1 Identificazione e classificazione

L'OCCI Core fornisce un sistema integrato per la classificazione dei tipi. Questo sistema offre gli strumenti necessari affinchè si possa operare in maniera semplice e trasparente (ad esempio per evitare la traduzione del formato dei dati), tramite l'utilizzo sia di protocolli basati su testo sia di protocolli binari. A far parte di questo sistema sono le classi Category, Kind e Mixin.

Category

La classe Category è alla base del meccanismo di identificazione usato dal sistema di classificazione di OCCI. Questo deve essere implementato. Le istanze di questa classe sono utilizzate per identificare le istanze della classe Action, mentre altri usi degli attriburi di Category sono gestiti dalle sue sottoclassi Kind e Mixin.
Category deve implementare i seguenti attributi, in maniera tale da essere conforme alle specifiche di questo modulo:

Tabella 4.1: Attributi definiti dalla classe Category

Attributo	Tipo	Molteplicità	Mutabilità	Descrizione
term	String	1	Immutabile	Identificativo univoco dell'istanza Category all'interno dello schema di categorizzazione
scheme	URI	1	Immutabile	Rapprensenta lo schema di categorizzazione
title	String	0..1	Immutabile	Nome dell'istanza
attributes	String	0..*	Immutabile	L'insieme di tutti gli attributi della risorsa definita dall'istanza della classe Category

L'identificazione univoca delle istanze Category avviene attraverso la concatenazione di *scheme* e *term* (es. http://example.com/category/scheme#term) e questa proprietà viene ereditata anche dalle sue sottoclassi Kind e Mixin. Tutti gli schemi di categorizzazione della specifica OCCI utilizzano l'URL di base "http://schemas.ogf.org/occi/", il quale è riservato per OCCI e non deve essere usato da nessun'altra estensione del fornitore.

Kind

Kind, insieme alla classe Mixin, rappresenta il sistema di classificazione di questo modello, ed implementa il meccanismo di identificazione dei tipi, per tutte le istanze Entity presenti nell'OCCI Core. Ogni sua istanza rappresenta l'identificativo univoco per le sottoclassi di Entity, con il quale rimarranno associate per tutto il loro ciclo di vita. Un modulo Core, inizializzato senza alcuna estensione, è composto da tre istanze Kind: una per Entity, l'altra per Resource e l'ultima per Link.
Per essere conforme alle specifiche, la classe Kind deve implementare gli attributi ereditati dalla classe Categoryi e i seguenti:

Tabella 4.2: Attributi definiti dalla classe Kind

Attributo	Tipo	Molteplicità	Mutabilità	Descrizione
actions	Action	0..*	Immutabile	Insieme di Actions definite dall'istanza Kind
related	Kind	0..*	Immutabile	Insieme delle istanze Kind
entity_type	Entity	1	Immutabile	Tipo di entità identificata dall'istanza
entities	Entity	0..*	Immutabile	Insieme delle risorse instanziate dalla sottoclasse di Entity, identificata da questa istanza Kind.

Anche questa classe deve essere implementata.

Mixin

Questa classe rappresenta un meccanismo di estensione capace di aggiungere nuove funzionalità alle risorse, sia al momento della loro creazione sia a runtime. Un'istanza Mixin può essere associata a qualsiasi risorsa esistente, in modo da aggiungerle nuovi attributi oppure altre Actions.
Per essere conforme alle specifiche, la classe Mixin deve implementare gli attributi elencati in tabella 4.3 e quelli ereditati dalla classe Category.
Nell'istanziazione del modello Core nessuna istanza Mixin è presente, ma anche questa classe deve essere implementata.
Per maggiori dettagli sul sistema di classificazione e identificazione si rimanda alla documentazione ufficiale([27], sezione 4.4).

Tabella 4.3: Attributi definiti dalla classe Mixin

Attributo	Tipo	Molteplicità	Mutabilità	Descrizione
actions	Action	0..*	Immutabile	Insieme di Actions definite dall'istanza Mixin
related	Mixin	0..*	Immutabile	Insieme delle istanze Mixin
entities	Entity	0..*	Mutabile	Insieme delle istanze di risorse associate con l'istanza Mixin

4.3.2 I tipi base

Tra i tipi base, invece, troviamo Entity, Resource, Link ed Action, i quali devono essere tutti implementati.

Entity

Entity è una classe astratta per le sue sottoclassi Link e Resource, le quali ereditano obbligatoriamente l'attributo *occi.core.id* ed opzionalmente *occi.core.title*.
Gli attributi che devono essere implementati affinchè questa risulti conforme con la specifica sono:

Tabella 4.4: Attributi definiti dalla classe Entity

Attributo	Tipo	Molteplicità	Mutabilità	Discov.	Descrizione
occi.core.id	URI	1	Immutabile	Si	ID univoco (all'interno del namespace del fornitore) delle istanze delle sottoclassi di Entity
occi.core.title	String	0..1	Mutabile	Si	Nome dell'istanza
kind	Kind	1	Immutabile	No	Istanza Kind che identifica la risorsa, sottoclasse di Entity
mixins	Kind	0..*	Mutabile	No	Insieme delle istanze Mixin associate a questa risorsa.

Ad ogni sottoclasse di Entity deve essere assegnata un'istanza Kind, mentre ad Entity l'istanza Kind "http://schemas.ogf.org/occi/core#entity" per la sua stessa identificazione, come da tabella 4.5.
Essendo una classe astratta, Entity di per sè non può essere istanziata.
Infine, l'istanza di una sottoclasse di Entity dovrebbe essere associata ad una o più istanze Mixin.

Tabella 4.5: Istanza Kind assegnata al tipo Entity

Attributo	Valore
term	entity
scheme	http://schemas.ogf.org/occi/core#
title	Entity type
attributes	occi.core.id, occi.core.title
actions	-

Resource

Il tipo Resource eredita Entity e rappresenta, tramite specializzazione, le risorse reali da controllate e manipolate: queste possono essere macchine virtuali, reti e servizi.
Questo tipo deve implementare tutti gli attributi ereditati da Entity, più i seguenti:

Tabella 4.6: Attributi definiti dalla classe Resource

Attributo	Tipo	Moltep.	Mutabilità	Descrizione
occi.core.summary	String	0..1	Mutabile	Breve descrizione dell'istanza Resource
links	Link	0..*	Mutabile	Insieme dei Link di cui è composta l'istanza. Essendoci una stretta relazione, la rimozione di un Link dall'insieme deve anche rimuovere l'istanza Link.

Per identificare il tipo Resource viene utilizzata l'istanza Kind così come descritta nella seguente tabella:

Tabella 4.7: Istanza Kind assegnata al tipo Resource

Attributo	Valore
term	resource
scheme	http://schemas.ogf.org/occi/core#
title	Resource
attributes	occi.core.summary
actions	-

Questa classe risulta essere il primo dei tre punti di ingresso per l'estensione del modulo OCCI Core. Per maggiori informazioni a riguardo vedi sezione 4.6 del documento di specifica[27]

Link

Un'istanza di tipo Link definisce l'associazione tra due istanze Resource. Link deve implementare sia gli attributi ereditati dalla classe Entity, sia i seguenti:

Tabella 4.8: Attributi definiti dalla classe Link

Attributo	Tipo	Moltep.	Mutabilità	Descrizione
occi.core.source	Resource	1	Mutabile	Istanza Resource da cui l'istanza Link ha origine.
occi.core.target	Resource	1	Mutabile	Istanza Resource a cui l'istanza Link punta.

Per identificare il tipo Link viene utilizzata l'istanza Kind così come descritta nella seguente tabella:

Tabella 4.9: Istanza Kind assegnata al tipo Link

Attributo	Valore
term	link
scheme	http://schemas.ogf.org/occi/core#
title	Link
attributes	occi.core.source, occi.core.target
actions	-

Gli attributi occi.core.source e occi.core.target devono riferirsi a risorse attraverso il namespace del fornitore; viene fornita anche la possibilità di riferirsi a risorse esterne, di cui quest'ultimo non ha alcun controllo diretto, ma soltanto se questa risorsa risulta essere mappata tramite istanze della sottoclasse di Entity.
Questa classe risulta essere il secondo punto d'ingresso per estendere il modulo OCCI Core. Vedi sezione 4.6 del documento di specifica[27].

Action

La classe Action è un tipo astratto. Ogni sua sottoclasse definisce una funzionalità applicabile su un'istanza di una sottoclasse di Entity: modifica lo stato di risorse tramite determinate operazioni, come ad esempio il riavvio

Tabella 4.10: Attributi definiti dalla classe Action

Attributo	Tipo	Moltep.	Mutabilità	Descrizione
category	Category	1	Immutabile	Istanza Category che identifica quella determinata istanza Action.

di una macchina virtuale. Action definisce i seguenti attributi:
Un'istanza di questa classe deve essere sempre associata per composizione ad un'istanza Mixin o Kind, le quali possono eseguire quella particolare funzionalità che Action definisce, sulle risorse che queste identificano. Tuttavia, l'implementazione OCCI può non autorizzare l'esecuzione di quell'operazione, nel caso in cui non fosse in quel momento applicabile. Questi metodi sono considerati validi soltanto se tutte le risorse del sistema risultano essere associate ad un'istanza Kind o Mixin, collegata a quell'istanza Action.
Per identificare il tipo Action viene utilizzata l'istanza Category così come descritta nella seguente tabella:

Tabella 4.11: Istanza Category assegnata alla classe Action

Attributo	Valore
term	action
scheme	http://schemas.ogf.org/occi/core#
title	Action
attributes	-

Tutti gli argomenti passati ai metodi della classe Action, devono essere definiti attraverso l'attributo *attributes* dell'istanza Category, la quale identifica le sottoclassi di Action. Se ad esempio è presente una sottoclasse Action *resize*, definita per un servizio di storage, questa dovrebbe avere anche un attributo *size* che rappresenti l'argomento di quell'operazione.
Infine, Action è il terzo ed ultimo punto d'ingresso per l'estensione di OCCI Core. Per dettagli vedi sezione 4.6 del documento di specifica[27].

4.4 OCCI - Infrastructure

Questo modulo rappresenta l'estensione di OCCI Core; implementa le APIs necessarie per la realizzazione di un'Infrastructure as a Service. Quest'ultime permettono la creazione e la gestione delle tipiche risorse associate a servizi IaaS, quali ad esempio istanze Compute o di Storage, occupandosi anche di interconnetterle tra di loro, tramite il tipo StorageLink.
I principali tipi infrastrutturali definiti dallo standard OCCI sono:

- ***Compute***. Rappresentano risorse di calcolo.
- ***Network***. Rappresentano risorse di interconnessione.
- ***Storage***. Rappresentano risorse per la memorizzazione d'informazioni.

In supporto ai suddetti tipi ci sono i seguenti sottotipi Link:

- ***NetworkInterface***. Connette un'istanza Compute ad un'istanza Network, con l'ausilio di IPNetworkInterface Mixin.
- ***StorageLink***. Connette un'istanza Compute ad un'istanza Storage.

Questi tipi ereditano i tipi base del modulo Core, insieme a tutti i relativi attributi. L'implementatore sarà libero di scegliere quali sottotipi di Resource e Link implementare; tutto quello che sarà supportato da quella specifica implementazione, sarà analizzato, a suo tempo, attraverso l'apposita interfaccia delle query di OCCI. Importante, è che, come descritto dal modulo Core, ad ogni risorsa creata venga assegnata un'istanza Kind per la sua univoca identificazione.

4.4.1 I principali tipi infrastrutturali

Analizziamo ora da vicino i tipi base di questo modulo.

Compute

Il tipo Compute rappresenta una generica risorsa di calcolo dell'informazione, come ad esempio una macchina virtuale. L'istanza Kind associata a Compute è http://schemas.ogf.org/occi/infrastructure#compute.
Di seguito si elencano attributi ed operazioni applicabili su istanze di questo tipo.

Tabella 4.12: Attributi definiti da Compute

Attributo	Tipo	Moltep.	Mutabilità	Descrizione
occi.compute.architecture	Enum{x86, x64}	0..1	Mutabile	Architettura della CPU.
occi.compute.cores	Integer	0..1	Mutabile	Numero cores della CPU.
occi.compute.hostname	String	0..1	Mutabile	Hostname dell'istanza.
occi.compute.speed	Float,10^9 (GHz)	0..1	Mutabile	Frequenza di clock.
occi.compute.memory	Float,10^9 (GiB)	0..1	Mutabile	Max Ram allocabile.
occi.compute.state	Enum{active, inactive, suspended}	1	Immutabile	Stato corrente.

Tabella 4.13: Operazioni applicabili su istanze di tipo Compute

Operazione	Stato dell'istanza	Attributi
start	active	-
stop	inactive	method={graceful,acpioff,poweroff}
restart	active	method={graceful,warm,cold}
suspend	suspended	method={hibernate,suspend}

Network

Il tipo Network rappresenta un'entità di rete di tipo L2, come ad esempio uno switch virtuale, e viene identificato attraverso l'istanza Kind http://schemas.ogf.org/occi/infrastructure#network.

Definisce i seguenti attributi e le seguenti operazioni:

Tabella 4.14: Attributi definiti da Network

Attributo	Tipo	Moltep.	Mutabilità	Descrizione
occi.network.vlan	Integer:0-4095	0..1	Mutabile	ID VLAN 802.1q.
occi.network.label	Token	0..1	Mutabile	Tag della VLAN.
occi.network.state	Enum{active,inactive}	1	Immutabile	Stato corrente dell'istanza.

Tabella 4.15: Operazioni applicabili su istanze di tipo Network

Operazione	Stato dell'istanza	Attributi
up	active	-
down	inactive	-

Network può essere esteso, tramite l'utilizzo di ***IPNetworking Mixin***[2], affinchè supporti le funzionalità L3/L4.
L'istanza Network associata ad un IPNetworking Mixin deve implementare i seguenti attributi:

Tabella 4.16: Attributi definiti dall'istanza Network associata ad IPNetworking Mixin

Attributo	Tipo	Moltep.	Mutabilità	Descrizione
occi.network.address	Range di indirizzi IPv4, IPv6	0..1	Mutabile	Indirizzo IP.
occi.network.gateway	Indirizzo IPv4 o IPv6	0..1	Mutabile	Indirizzo IP.
occi.network.allocation	Enum{dynamic, static}	0..1	Mutabile	Meccanismo di allocazione degli indirizzi.

Per collegare un'istanza Compute alla rete viene utilizzato il tipo NetworkInterface, il quale eredita dal tipo base Link del modello Core. A NetworkInterface viene assegnata l'istanza Kind http://schemas.ogf.org/occi/infrastructure#networkinterface, connessa all'istanza http://schemas.ogf.org/occi/core#link.

[2]Identificato dall'istanza Kind "http://schemas.ogf.org/occi/infrastructure/network#ipnetwork"

Attraverso quest'ultima, NetworkInterface definisce i seguenti attributi:

Tabella 4.17: Attributi definiti dall'istanza NetworkInterface

Attributo	Tipo	Moltep.	Mutabilità	Descrizione
occi.networkinterface.interface	String	1	Immutabile	ID dell'interfaccia utilizzata per la connessione.
occi.networkinterface.mac	String	1	Mutabile	Indirizzo Mac dell'interfaccia utilizzata per la connessione
occi.networkinterface.state	Enum{active, inactive}	1	Immutabile	Stato corrente dell'istanza.

Per supportare le funzionalità L3/L4 con il tipo NetworkInterface viene definita ed utilizzata l'istanza Mixin ***IPNetworkInterface***[3].
Quest'istanza definisce i seguenti attributi:

Tabella 4.18: Attributi definiti dall'istanza IPNetworkInterface

Attributo	Tipo	Moltep.	Mutabilità	Descrizione
occi.networkinterface.address	Indirizzo IPv4, IPv6	1	Mutabile	Indirizzo IP.
occi.networkinterface.gateway	Indirizzo IPv4 o IPv6	0..1	Mutabile	Indirizzo IP.
occi.networkinterface.allocation	Enum{dynamic, static}	1	Mutabile	Meccanismo di allocazione degli indirizzi.

Per maggiori dettagli riguardo al tipo Network e al collegamento di risorse di rete vedi rispettivamente sezione 3.2 e 3.4 di [28].

Storage

Questo tipo rappresenta risorse che memorizzano informazioni su dispositivi di immagazzinamento dati ed eredita dalla classe base Resource dell'OCCI Core.
Identificato da http://schemas.ogf.org/occi/infrastructure#storage, deve implementare gli attributi elencati in tabella 4.19; tabella 4.13, invece, tiene

[3]Identificata da http://schemas.ogf.org/occi/infrastructure/networkinterface#ipnetworkinterface

traccia delle operazioni applicabili su questa classe di istanze.

Tabella 4.19: Attributi definiti dall'istanza Storage

Attributo	Tipo	Moltep.	Mutabilità	Descrizione
occi.storage.size	Float, 10^9 (GiB)	1	Mutabile	Spazio d'immagazzinamento dell'istanza.
occi.storage.state	Enum{online, offline, backup, snapshot, resize, degraded}	1	Immutabile	Stato corrente dell'istanza.

Tabella 4.20: Operazioni applicabili su istanze di tipo Storage

Operazione	Stato dell'istanza	Attributi
online	online	-
offline	offline	-
backup	None	-
snapshot	None	-
resize	None	size=Flaot 10^9 (GiB)

Per quanto riguarda il collegamento di un'istanza Compute con un'istanza Storage viene utilizzato il tipo StorageLink: fornisce tutte le funzionalità preliminari di basso livello messe a disposizione dall'implementazione OCCI ed è identificato dall'istanza Kind http://schemas.ogf.org/occi/infrastructure#storagelink, connessa a sua volta con l'istanza Kind http://schemas.ogf.org/occi/core#link. Tramite quest'ultima definisce i seguenti attributi:

Tabella 4.21: Attributi definiti dall'istanza StorageLink

Attributo	Tipo	Moltep.	Mutabilità	Descrizione
occi.storagelink.deviceid	String	1	Mutabile	ID del dispositivo.
occi.storagelink.mountpoint	String	0..1	Mutabile	Percorso di montaggio del dispositivo nel sistema ospite.
occi.storagelink.state	Enum{active, inactive}	1	Immutabile	Stato attuale dell'istanza.

Per maggiori dettagli riguardo al tipo Storage e al collegamento di risorse per l'immagazzinamento vedi, rispettivamente, sezione 3.3 e 3.4 di [28].

4.4.2 Templates predefiniti

I templates permettono, ai clienti di un'implementazione OCCI, di utilizzare, in maniera semplice e veloce, delle configurazioni predefinite. Questi vengono implementati attraverso l'utilizzo di un'istanza Mixin.
OCCI supporta l'implementazione di due tipi di template:

- ***OS (Operating System) Template***. Permettono all'utente di specificare quale sistema operativo deve essere installato su di una specifica risorsa Compute.
- ***Resource Template***. È compito del provider mettere a disposizione dell'implementazione una configurazione preimpostata di instanze Resource.

Per maggiori dettagli si rimanda alla sezione 3.5 del documento di specifica[28].

4.5 OCCI - RESTful HTTP Rendering

Questo modulo specifica come deve essere utilizzato il modulo OCCI-Core, attraverso l'utilizzo del protocollo HTTP, e si basa sul concetto di *Resource Oriented Architecture (ROA)*. ROA utilizza l'approccio RESTful per assicurare l'interazione tra il client ed i servizi, la quale è realizzata attraverso il controllo e la modifica di un insieme di risorse, collegate tra di loro, e dei loro stati.
HTTP è il protocollo ideale per implementare un sistema basato su architettura ROA, poichè fornisce sia i mezzi adatti per l'identificazione univoca di singole risorse, attraverso l'utilizzo di URLs, sia un insieme di metodi generici in grado di manipolarle: POST per la creazione, GET per il recupero, POST/PUT per l'aggiornamento e DELETE per la rimozione di risorse.
Per maggiori approfondimenti riguardo al comportamento delle specifiche operazioni, al rendering dei tipi Category, Kind, Mixin, delle istanze Link, Action, Entity e per quel che riguarda sicurezza e autenticazione, si rimanda alla documentazione ufficiale[29] e alla lettura del capitolo 2 di [30].

Conclusioni

Cloud computing e virtualizzazione hanno indubbiamente cambiato il modo di concepire le infrastrutture IT. E non solo. Grazie alla scalabilità offerta da queste tecnologie, aziende ed organizzazioni, pubbliche e private, possono realizzare un sistema per la fornitura di servizi in base alle esigenze di quel momento. Lo scopo di questo elaborato, oltre a fare una panoramica del cloud computing che ormai è entrato a far parte della vita quotidiana della maggior parte di noi, è stato quello di analizzare e far conoscere le piattaforme e i tools maggiormente utilizzati per la gestione ed il mantenimento di piccole e grandi infrastrutture IT. In quanto alle piattaforme di gestione di sistemi cloud, OpenStack senz'altro è uno dei progetti maggiormente utilizzati, ma soprattutto supportati dai colossi dell'informatica: una nuova release viene rilasciata ogni sei mesi e questo lo rende un progetto sempre più all'avanguardia. La OpenStack Foundation il 16 Ottobre 2014 ha rilasciato l'ultima release del software, OpenStack Juno, con diversi nuovi progetti ed un sacco di nuove funzionalità. OpenStack, OpenNebula ed Eucalyptus sono di grande aiuto per chi deve gestire un ambiente cloud, ma quando l'infrastruttura IT cresce, crescono anche i problemi relativi alla configurazione di quest'ultima: come abbiamo visto, negli ultimi anni alcuni tools sono stati sviluppati per automatizzare questo compito. Questi ultimi vengono integrati alle diverse piattaforme cloud per migliorare la gestione delle risorse; nonostante questi offrano simili funzionalità, bisognerebbe prendersi del tempo ed analizzarli singolarmente prima di adottarne uno, per capire quali di questi soddisfino al meglio le proprie esigenze. Oggigiorno Puppet e Chef guidano questo settore, ma anche altri tools vengono costantemente aggiornati e si propongono come valide alternative.

L'adozione di uno standard comune, inoltre, semplificherebbe di gran lunga la comunicazione tra i diversi provider. OCCI è riuscito ad astrarre le risorse dei servizi cloud ed oggigiorno diverse sono le sue implementazioni.
Come accade ai nuovi progetti, però, ancora è troppo presto per dire se questo verrà accettato ed utilizzato da tutti: di sicuro l'Open Cloud farà discutere molto nei prossimi anni.

Ringraziamenti

Un primo ringraziamento va al mio relatore, il Prof. Vittorio Ghini, per la cortesia e la disponibilità dimostratami durante la stesura di questo lavoro.

Ringrazio infinitamente i miei Genitori per avermi sostenuto in tutto e per tutto, dandomi sempre la forza di andare avanti, non solo durante la mia carriera universitaria ma soprattutto nella vita di tutti i giorni. Siete il mio punto di riferimento.

Ringrazio mia sorella Lorena. Hai saputo sempre essermi vicino nonostante la lontananza dell'ultimo anno. Semplicemente sei e sarai sempre una persona speciale. Ti voglio bene.
Un grazie speciale anche ad Enrico, per i momenti piacevoli trascorsi insieme.

Ringrazio "*la*" Zia Anna, una seconda madre. Se oggi ho raggiunto questo traguardo sicuramente è anche merito tuo.

Ringrazio Melissa. Te che mi hai costantemente incoraggiato e sostenuto durante questo mio percorso. Te che hai saputo sempre strapparmi un sorriso. Te, piccola grande donna. Grazie.

Ringrazio Dario, un fratello più che un amico. Grazie al tuo modo di essere hai reso più leggeri e spensierati i momenti di maggiore tensione e difficoltà, incoraggiandomi e supportandomi. Ringrazio anche Silvia, tua complice.

Ringrazio Donny per il continuo supporto morale datomi. Sei una buona

amica.

Ringrazio Maurizio, Lamberto e Francesco, i quali ognuno a proprio modo sono stati stimolo di crescita e conoscenza.

Un ringraziamento speciale anche a tutti i cugini e parenti che mi sono stati vicino; a "*Quei bravi ragazzi di Arnesano*": Angelo, Marco, Sonia, Gabriele, Mattia e a tutti gli Amici conosciuti negli ultimi anni.

Bibliografia

[1] George Reese: Cloud Computing. Architettura, infrastrutture, applicazioni

[2] The Guardian: "Cloud computing is a trap, warms GNU founder Richard Stallman", 2008. Reperibile al sito http://www.theguardian.com/technology/2008/sep/29/cloud.computing.richard.stallman

[3] Luca Annunziata, Stallman: dite no al cloud computing, 2008. Reperibile al sito http://punto-informatico.it/2422084/PI/News/stallman-dite-no-al-cloud-computing.aspx

[4] Manolo De Agostini, "Stallman su Chrome OS, cloud computing pericoloso", 2010. Reperibile al sito http://www.tomshw.it/cont/news/stallman-su-chrome-os-cloud-computing-pericoloso/28604/1.html

[5] Pell Mell, Timothy Grance: The Nist Definition of Cloud Computing, 2009. Reperibile al sito http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[6] L'origine storica del cloud: i principali sostenitori: http://www.hostingtalk.it/lezione-2-lorigine-storica-del-cloud-i-principali-sostenitori_-c000000sh/

[7] Cloud Computing: rebrand o nuova tecnologia? Breve storia del cloud computing dai mainframe al cloud.

http://www.mycloudbeans.com/content/cloud-computing-rebrand-o-nuova-tecnologia-breve-storia-del-computing-dai-mainframe-al-cloud

[8] Jeffrey O.Kephart, David M.Chess: The Vision of Autonomic Computing, articolo reperibile al sito http://users.soe.ucsc.edu/~griss/agent-papers/ieee-autonomic.pdf

[9] Database as a Service: Reference Architecture–An Overview http://www.oracle.com/technetwork/topics/entarch/oes-refarch-dbaas-508111.pdf

[10] Astrid, ResPublica: L'impatto del cloud computing sull'economia italiana

[11] Tiziana Moriconi, Openstack, il software open che rivoluziona il cloud computing, 2012. Reperibile al sito http://daily.wired.it/news/internet/2012/04/06/openstack-open-source-cloud-36241.html

[12] OpenStack Documentation. Reperibile al sito http://docs.openstack.org/

[13] OpenNebula 4.8 Documentation. Reperibile al sito http://docs.opennebula.org/4.10/

[14] Eucalyptus 4.x Documentation. Reperibile al sito https://www.eucalyptus.com/eucalyptus-cloud/documentation

[15] Eucalyptus Reference Architectures. Reperibile al sito https://eucalyptus.atlassian.net/wiki/display/EUCA/Eucalyptus+Reference+Architectures

[16] Amazon EC2 http://aws.amazon.com/ec2/

[17] Alessio Carta: Guida alla virtualizzazione con VMware vSphere 5.1

[18] CloudStack Documentation http://docs.cloudstack.apache.org/en/master/index.html

[19] Microsoft Azure Documentation http://azure.microsoft.com/en-us/documentation/

[20] Pasquale Amoruso: Analisi di strumenti software per l'automazione dei processi di Configuration Management dei sistemi: Puppet Software Tool

[21] Valentino Gagliardi: Ansible per l'automazione IT e Configuration Management

[22] SaltStack Documentation http://www.saltstack.com/

[23] Ansible Documentation http://docs.ansible.com/

[24] Puppet Labs Documentation https://docs.puppetlabs.com/

[25] Chef Documentation https://docs.getchef.com/

[26] Marco Rodolfo Alessandro Bozzetti: Guida al Cloud. Nella Nuvola la Stella Cometa per il Manager, 2012

[27] Ralf Nyrèn, Andy Edmonds, Alexander Papaspyrou, Thijs Metsch: Open Cloud Computing Interface - Core, 2011

[28] Thijs Metsch, Andy Edmonds :Open Cloud Computing Interface - Infrastructure, 2011

[29] Thijs Metsch, Andy Edmonds: Open Cloud Computing Interface - RESTful HTTP Rendering, 2011

[30] Sandro Fiore, Giovanni Aloisio: Grid and Cloud Database Management, 2011

[31] Luigi Vanacore: Analisi degli standard OCCI per il Cloud Computing

[32] Amine Ghrab , Sabri Skhiri , Hervé Kœner, Guy Leduc: Towards A Standards-Based Cloud Service Manager