ALMA MATER STUDIORUM
UNIVERSITA' DEGLI STUDI DI BOLOGNA
SCUOLA DI INGEGNERIA E ARCHITETTURA SEDE DI CESENA
CORSO DI LAUREA MAGISTRALE
IN INGEGNERIA E SCIENZE INFORMATICHE

TuCSoN on Cloud:

"Elastic" deployment of TuCSoN nodes in the Cloud

Progetto di Sistemi Distribuiti

realizzato da

Casadei Richiard

Indice

Abstract

Se ci venisse chiesto di riassumere, in quattro parole chiave, in cosa consiste il Cloud Computing potremmo senza dubbio scegliere 'web', 'elasticità', 'utility' e 'scalabilità'[1]. E' la natura stessa del paradigma Cloud e delle sue tecnologie a farci scegliere la prima parola, poiché esso consiste in un servizio offerto da un provider al cliente mediante l'utilizzo di risorse hardware/software distribuite e virtualizzate in rete rese accessibili tramite browser web. Introducendo il concetto di virtualizzazione, ovvero il processo in cui vengono utilizzate risorse astratte o virtuali per simulare quelle fisiche, è possibile spiegare la seconda e la terza parola chiave. Attraverso il processo di virtualizzazione si può fornire l'elasticità necessaria per offrire il Cloud Computing come utility che promette l'accesso e l'utilizzo facilitato di un ampio pool di risorse come hardware, piattaforme e/o servizi di sviluppo, che possono essere fornite in modo dinamico per adattarsi ad un carico di lavoro variabile attraverso un load balancing automatico, consentendo anche l'utilizzo delle risorse in modo ottimale scalando automaticamente e rapidamente verso l'esterno o verso l'interno proporzionalmente alla domanda dell'utente. Proprio la scalabilità, ovvero la quarta parola chiave, consiste in un altra caratteristica fondamentale del paradigma Cloud e può esser definita come la capacità di un particolare sistema nell'adattarsi e modificarsi nel caso di variazioni notevoli della mole o della tipologia dei dati trattati o di variazioni notevoli del numero di computazioni su di essi.

Tutte queste caratteristiche dovranno essere ereditate dal sistema TuCSoN nel momento in cui introduciamo il concetto di coordinazione intesa come servizio(CaaS, Coordination as a Service), e volendo quindi portare tale sistema su un'infrastruttura Cloud, offrendo così la sua coordinazione non più come modello di interazione fra normali host, ma come servizio cloud. Tale soluzione avrà come effetto sul sistema TuCSoN una maggiore affidabilità, scalabilità ed efficienza, senza considerare che, data la natura stessa del Cloud Computing, sarà più semplice distribuirlo ed offrirlo a possibili utenti. I meccanismi di scalabilità, elasticità e automazione di base, da far ereditare al sistema TuCSoN on Cloud, sono già stati ampiamente attuati a livello di infrastrutture Cloud, ovvero IaaS, però nei livelli di distribuzione "superiori", come PaaS, SaaS e conseguentemente anche CaaS, sono ancora in fase di sviluppo un diverso numero di tecniche per aiutare a scalare e a gestire quindi carichi di lavoro variabili che devono esser attuate dallo sviluppatore stesso della piattaforma o, nel nostro caso, del sistema TuCSoN on Cloud.

A tal proposito proseguiremo introducendo alcune delle più famose tecniche e modelli di programmazione utilizzati per fornire scalabilità e le scelte attuate per garantire affidabilità, fault tolerance, semplicità ed efficienza tramite load balancing all'interno del Cloud.

Successivamente vedremo quali di questi concetti e di quelli esistenti in varie piattaforme Cloud esistenti potranno essere utilizzati o adattati all'attuale modello di coordinazione TuCSoN, quali concetti di TuCSoN dovranno esser resi disponibili all'infrastruttura Cloud per poter cooperare correttamente ed in armonia e concluderemo il tutto progettando gli interventi necessari per ottenere il sistema TuCSoN on Cloud.

Capitolo 1

Cloud computing: scalabilità ed elasticità

Il Cloud Computing si propone di migliorare i sistemi informatici diminuendo il coinvolgimento umano nel loro funzionamento e vuole raggiungere l'obiettivo finale di fornire sistemi che si gestiscono autonomamente e si adattano ai cambiamenti imprevedibili che possono occorrere al sistema stesso[1].

La caratteristica dei sistemi Cloud di adattarsi, o più precisamente di scalare, in modo elastico ed autonomo si basa su tre principi fondamentali:

- Virtualizzazione: riduce la complessità dei sistemi, standardizzando la piattaforma hardware, riducendo i costi di gestione delle risorse e migliorandone la disponibilità così da consentirne un rapido ed elastico rilascio. Rende i sistemi agili e flessibili ma sopratutto ci permette di disaccoppiare l'infrastruttura informatica della infrastruttura fisica.
- Condivisione delle risorse: la condivisione delle risorse di calcolo tra diverse applicazioni e/o organizzazioni consente di ottimizzare il loro uso.
- Rilascio dinamico: le risorse dovrebbero essere fornite in modo on-demand, ma ciò implica la necessità di monitorare le prestazioni del servizio.

La scalabilità e questi tre principi dipendono fortemente dalla robustezza dell'architettura Cloud sottostante. Nei sistemi Cloud, sopratutto quelli enterprise, una architettura orientata ai servizi (SOA) può essere utilizzata per fornire un'interfaccia semplificata per i processi sottostanti. Le richieste in arrivo per il servizio fornito da questo composito SOA devono essere indirizzati ai componenti corretti e ai rispettivi server, e tale percorso deve essere scalabile per supportare un gran numero di richieste. Solitamente ogni livello SOA distribuisce più server per la distribuzione del carico(load balancing) e la fault tolerance e tale scenario è definito "distribuzione orizzontale" del carico o scalabilità orizzontale, ma esso presenta una limitazione. Nel momento in cui tutti i server sono saturi non è più possibile distribuire ulteriormente il carico e per questo motivo tali architetture si sono evolute aggiungendo più livelli, diventando così multi-tier ed inserendo al loro interno più implementazioni di un determinato servizio offrendo così l'opportunità per una "distribuzione verticale" del carico, ovvero la scalabilità verticale, su più livelli.

Tecniche per la scalabilità: MapReduce

Modello di programmazione, originariamente sviluppato dai fondatori di Google[1] e ora utilizzato per fornire scalabilità a molte delle applicazioni distribuite tipiche del Cloud MapReduce[2] è stato originariamente introdotto da Google come un modello di programmazione distribuita utilizzando grandi cluster di server per elaborare enormi quantitativi(multi-terabyte) di dati. Il modello può essere applicato a molti problemi di calcolo di grandi dimensioni e offre una serie di caratteristiche interessanti come la parallelizzazione

[1] www.google.com

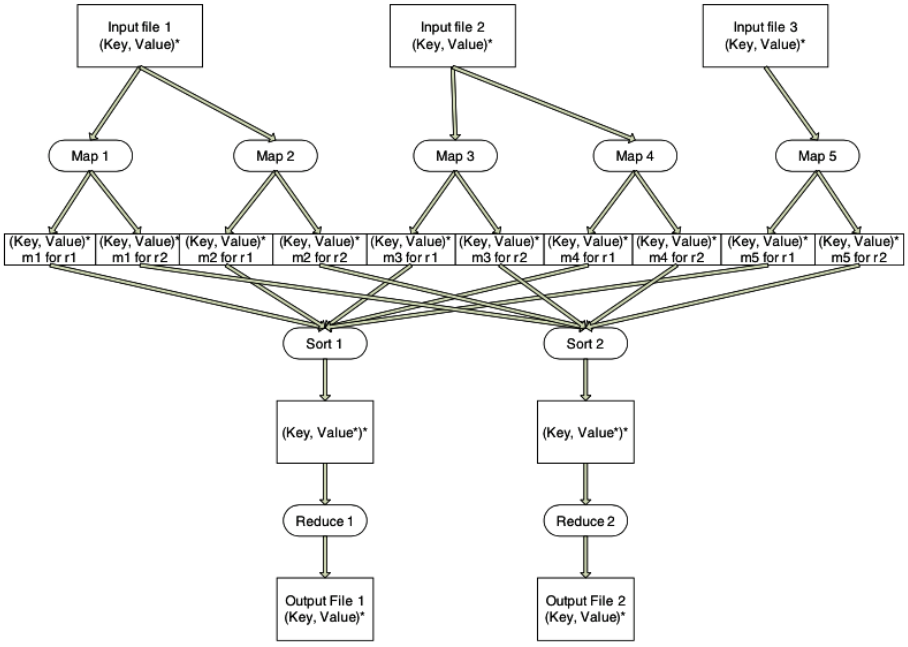
automatica, il load balancing, l'ottimizzazione della rete e una robusta gestione dei guasti alla macchina. Il modello MapReduce si pone di fronte ad un problema scomponendolo in parti più piccole, risolvendo ogni parte in parallelo e quindi ne ricombina i risultati per produrre la risposta finale.

MapReduce gira su file system specializzati, come il Google File System (GFS) o il Hadoop File System (HDFS). I dati vengono caricati, suddivisi in pezzi (comunemente 64 MB) tali che ogni pezzo può esser replicato. Una caratteristica fondamentale di MapReduce è che sia l'elaborazione sia la memorizzazione dei dati risulta distribuita, portando così una serie di vantaggi decisamente più evidenti rispetto all'approccio tradizionale in cui si devono spostare i dati per il calcolo[2]. Tali vantaggi includono:

- Scalabilità
- Affidabilità
- Fault Tolerance
- Semplicità
- Efficienza

Questi vantaggi portano tale modello di programmazione ad esser ampiamente utilizzato in ambienti di Cloud Computing per l'elaborazione di grandi quantità di dati in un modo altamente parallelo.

Nel modello MapReduce, le funzioni non sono così rigidamente definite, ma i programmatori possono specificare ancora il calcolo in termini di due funzioni map e reduce, che possono essere effettuate in parallelo sui sottoinsiemi dei dati totali. La funzione map viene usato per generare un elenco di coppie <chiave,valore> intermedie, mentre la funzione reduce unisce tutti i valori intermedi associati con la stessa chiave intermedia[2].



Quando sono state completate tutte le attività, il risultato viene restituito all'utente. Il modello MapReduce è progettato per funzionare in un file system distribuito e utilizza la scalabilità orizzontale ottenendola mediante l'aggiunta di nodi di calcolo.

Tecniche per la scalabilità: QoS Quality-of-Service on Cloud

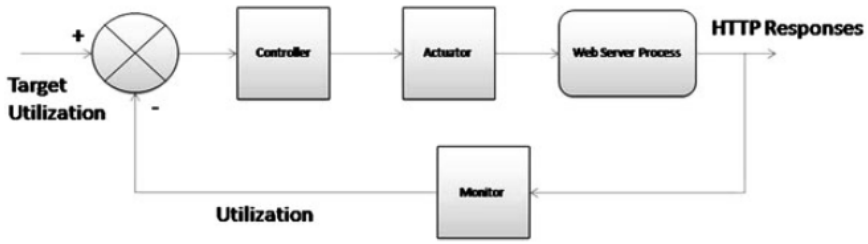
Alcune tecniche e concetti utilizzati per ottenere una qualità di servizio desiderata all'interno di un sistema possono essere estese al Cloud Computing in modo da fornire soluzioni per la gestione e l'adattamento dell'infrastruttura Cloud a fronte di importanti cambiamenti in quest'ultima.

Soluzioni di QoS[2] e gestione per le attuali applicazioni Web possono essere inserite ed adattate in una qualsiasi applicazione Cloud all'interno di una infrastruttura IaaS, utilizzando un ipotetico anello di controllo che cerca di evitare sovraccarichi del sistema, soddisfare il tempo di risposta individuale e garantire il throughput[2]. I componenti di tale anello sono:

- Application Service: l'applicazione di servizio da eseguire in una o più macchine virtuali nel Cloud.
- Monitor: entità che offre risposte circa l'utilizzo delle risorse in base a misure disponibili ed interrogabili tramite le API dell'infrastruttura Cloud, come CPU, memoria su disco e larghezza di banda della rete.
- Controller: entità che data la differenza tra la QoS desiderata e il reale utilizzo delle risorse (misurato dal componente Monitor), decide quali azioni correttive attuare per raggiungere l'obiettivo di QoS. e massimizzare l'utilizzo delle risorse del sistema.
- Actuator: componente che traduce le decisioni correttive prese dal Controller in azioni concrete inviate al middleware Cloud col fine di scalare su o giù, attraverso gli strumenti messi a disposizione dall'infrastruttura Cloud alla base del sistema, i componenti del servizio in modo da cambiare il suo carico.

L'anello di controllo ha quindi il compito di modellare il sistema Cloud in base alle esigenze del servizio e realizzare una metodologia valida per fornire scalabilità ed elasticità, attraverso un costante controllo su di esso e apportando differenti cambiamenti a run-time.

Di seguito è fornito uno schema dell'anello di controllo di un processo Web Server.



[2] http://en.wikipedia.org/wiki/Quality_of_service

Altre tecniche per la scalabilità

Un altro approccio per fornire ad un sistema adattabilità ai cambiamenti, elasticità e scalabilità è quello di avere più opzioni di implementazione che danno possibilità sia di distribuzione del carico verticale su più livelli sia quella orizzontale. La possibilità di scalare un sistema può dipendere dalla sua struttura, i tipi di strutture dati, algoritmi o meccanismi di comunicazione utilizzati per implementare i componenti del sistema[2].

La scalabilità automatica a livello di applicazione può essere implementata in diversi modi:

- Gli utenti forniscono un insieme di regole in un linguaggio ben definito che alimenta un "controller" che agisce per conto dell'utente ed è incaricato di far rispettare le regole di scalabilità specificate.
- Progettare ed implementare algoritmi specifici o metodi statistici per far sapere al controller quando l'applicazione deve scalare, senza dover ricorrere ad azioni dirette da parte di un utente sul sistema interessato.

Si può osservare che i sistemi basati su regole forniscono descrizioni significative sulle condizioni adeguate alla scalabilità ed un eventuale utilizzo di tecniche di datamining può essere impiegato per estrarre regole pertinenti e aiutare gli utenti (fornitori di servizi) ad acquisire a runtime conoscenze sulle loro tecniche di esecuzione e ottimizzazione delle applicazioni, mentre le tecniche algoritmiche (consideriamo anche strumenti statistici, reti neurali, o tecniche di teoria di controllo tradizionali) offrono un grado di controllo e una reattività a run-time che ancora non può essere raggiunta con sistemi basati su regole[1].

Capitolo 2

TuCSoN ed il Cloud

Diverse motivazioni ci hanno spinto a pensare di poter avere il sistema TuCSoN[3] sul Cloud. Un suo inserimento in tale contesto porterebbe in principio ad una più semplice distribuzione ed offerta all'utente, poiché grazie alla natura stessa del Cloud sarà possibile fornire coordinazione a utenti, processi o software senza dover installare piattaforme dedicate contenenti modelli di coordinazione sulle macchine fisiche, ma la coordinazione sarà eseguita all'interno della "nuvola".

Anche dal punto di vista tecnico si otterrebbero numerosi vantaggi come una maggiore affidabilità del servizio, scalabilità, efficienza e fault tolerance realizzabili mediante l'estensione di strategie e tecniche già ampiamente utilizzate nel Cloud che rappresentano caratteristiche proprie di questa tecnologia. Efficienza, scalabilità e fault tolerance possono essere realizzati mediante un deployment elastico dei nodi, su risorse fisiche o virtuali all'interno del Cloud, utilizzando le strategie di scaling offerte dal provider del servizio Cloud e sui quali saranno creati o replicati i centri di tuple. Tale operazione, regolata da opportune strategie di l'utilizzo, apporterebbe, inoltre, al sistema TuCSoN on Cloud la proprietà di load balancing che dovrà essere supportata da entità di monitoring dei nodi e delle risorse che gli fanno da host, le quali ricoprono il ruolo di manager delle risorse e applicheranno le azioni suggerite dalle strategie di gestione.

TuCSoN: il sistema

Per poter realizzare il sistema TuCSoN on Cloud è necessario fare un'analisi preventiva delle caratteristiche principali dell'attuale sistema TuCSoN, di come esse possano giocare un ruolo fondamentale durante il suo inserimento nel contesto Cloud e di come potrebbero coesistere con le caratteristiche tipiche del Cloud.

TuCSoN è un modello per la coordinazione dei processi distribuiti, oltre che di agenti autonomi, intelligenti e mobili le cui entità principali sono rappresentati da[3]:

- Agenti TuCSoN, entità pro-attive che interagiscono tra di loro scambiandosi, mediante primitive di coordinazione, tuple attraverso i centri di tuple. Rappresentano le entità coordinabili ed a ciascuno di essi è assegnato un nome `aname` ed un identificativo UUID che insieme ne compongono il nome completo: `aname : uuid`
- Nodi TuCSoN, rappresentano l'astrazione topologica in cui risiedono i centri di tuple. Ciascuno di essi è identificato univocamente dalla coppia `< netid : portno >` che corrispondono rispettivamente al numero IP o il nome di dominio (DNS) del device che ospita il nodo e al numero di porta dove il servizio TuCSoN è in ascolto.

[3] http://apice.unibo.it/xwiki/bin/view/TuCSoN/

- Centri di tuple ReSpecT, entità reattive che rappresentano il mezzo di coordinazione e forniscono lo spazio condiviso per la comunicazione basata su tuple. Siccome ogni nodo contiene al massimo un centro di tuple quest'ultimo è identificato univocamente tramite un nome ammissibile associato all'identificatore del nodo che lo ospita, il nome completo di un centro di tuple chiamato tname risulta essere: `tname @ netid : portno`

Un sistema TuCSoN è quindi caratterizzato dall'insieme di nodi TuCSoN (possibilmente distribuiti) che ospitano un servizio TuCSoN, ciascun nodo è identificato dal dispositivo di rete che ospita il servizio e dalla porta di rete in cui il servizio TuCSoN ascolta richieste in ingresso. Il nodo quindi è incaricato di rimanere in ascolto di invocazioni di operazioni sulla porta associata al device, smistarle al centro di tuple specificato e ritornare il risultato dell'operazione.

Il linguaggio di coordinazione di TuCSoN permette agli agenti di interagire con i centri di tuple attraverso l'esecuzione di operazioni di coordinazione, costituite da primitive di coordinazione e linguaggi di comunicazione[3]. Un'operazione di coordinazione è invocata da un agente su uno specifico centro di tuple, il quale è incaricato della sua esecuzione, e la sua sintassi completa è composta da `tname @ netid : portno ? op` , dove `op` rappresenta l'operazione da eseguire e costituita dalle primitive di coordinazione. Siccome nell'invocazione si devono specificare le informazioni di rete del nodo(il suo nome) si parla di invocazione esplicita dell'operazione.

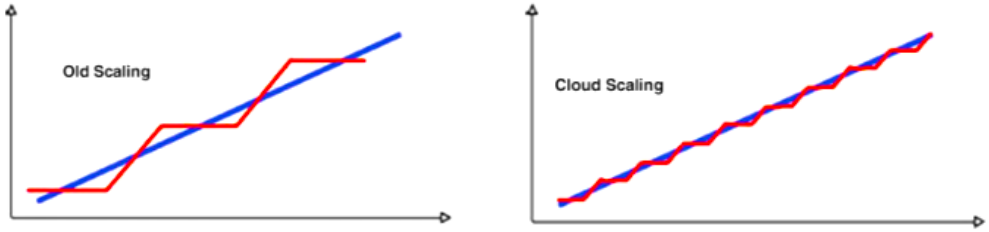
Per poter eseguire operazioni sui centri di tuple appartenenti ad una specifica organizzazione, agli agenti è rilasciato un Agent Coordination Context(ACC), una interfaccia runtime e stateful che fornisce le operazioni disponibili ed eseguibili tu tali centri di tuple. Un ACC ricopre il ruolo di mediatore dell'interazione in quanto un agente non può invocare un'operazione direttamente sul nodo, ma esso dovrà utilizzare "l'interfaccia" delle operazioni disponibili fornitagli dall'ACC. Dualmente il sistema di nodi interagisce con gli agenti solo tramite l'ACC.
Gli ACC sono le astrazioni runtime che permettono di gestire in modo uniforme all'interno del sistema TuCSoN i problemi di coordinazione, organizzazione e sicurezza e forniscono un disaccoppiamento tra gli agenti ed il loro ambiente, in modo che ciascun agente può essere progettato e sviluppato indipendentemente dagli altri agenti, risorse e servizi.

Il sistema TuCSoN inoltre supporta la persistenza, il quale fornisce la possibilità di spostare il contenuto di un centro di tuple dalla memoria volatile ad una memoria persistente, attraverso una richiesta di attivazione[3]. Tale richiesta memorizza un'istantanea del contenuto del centro di tuple, che viene "fotografato" nel momento esatto in cui viene attivata la persistenza, e tale operazione è ripetuta per tutti gli aggiornamenti che si verificano in seguito, fino a quando la persistenza non viene disattivata.

TuCSoN to Cloud, Cloud to TuCSoN

Dopo aver effettuato un'analisi del sistema TuCSoN nella quale sono state riportate le principali specifiche che lo caratterizzano, proseguiamo analizzando quali di questi concetti e quali di quelli appartenenti alla tecnologia Cloud debbano essere estesi o uniti al fine di realizzare il sistema TuCSoN on Cloud, partendo da queste due tecnologie inizialmente distinte. L'analisi è focalizzata sui concetti e su cosa in concreto entrambe le tecnologie dovranno fornire l'una all'altra per poter funzionare in sinergia, così da poter realizzare una sorta di linea guida sul "cosa ci serve" per la scelta di una piattaforma Cloud adatta al nostro scopo e sulle scelte architetturali ed implementative che decideremo di prendere.

Il Cloud fornisce scalabilità, efficienza e affidabilità mediante l'utilizzo di alcune delle tecniche spiegate nel capitolo precedente e applicandole quando necessario, in modo tale da ottimizzare l'utilizzo delle risorse ed evitando quindi sopra/sotto-fornitura di quest'ultime.



Per fare ciò sono utilizzate tecniche di monitoraggio delle risorse impiegate sul Cloud, le quali hanno il compito di controllare periodicamente il livello delle metriche specificate dal progettista e di compiere azioni correttive in caso di necessità, così da avere sempre il massimo livello di performance del sistema. Solitamente nei sistemi Cloud, le metriche di riferimento sono quelle corrispondenti alle risorse delle macchine fisiche o virtuali come CPU, RAM ecc. ma per quanto riguarda un sistema TuCSoN on Cloud tali metriche non sono adatte dato che, da test di performance effettuati su TuCSoN, si evince che quest'ultimo non ha un grosso impatto su CPU, RAM e altre risorse fisiche del computer che lo ospita. Per quanto riguarda la misurazione delle performance su un sistema TuCSoN on Cloud la metrica più consona ad un monitoraggio e controllo è il numero di richieste di operazioni su un nodo, maggior è il numero di richieste da gestire maggior sarà la latenza di risposta ed il sistema dovrà agire e scalare per mantenere un alto livello di performance. Scalare nel sistema TuCSoN on Cloud significa aver la possibilità di creare/replicare/splittare e rendere attivo un nodo TuCSoN uguale a quello in cui le performance, secondo la metrica stabilita, stanno iniziando a degradarsi e su di esso indirizzare una parte di richieste di operazioni che prima erano destinate al primo nodo. Per fare ciò è necessario capire se sarà onere del sistema TuCSoN apportare queste azioni e fare in modo che il nuovo nodo contenga già al suo interno il centro di tuple o le tecniche già offerte del Cloud basteranno e/o dovranno essere estese. La creazione/replicazione di nodi TuCSoN dovrà inoltre essere registrata all'interno del sistema in modo tale da avere sempre una panoramica dell'intero sistema e per aver inoltre la possibilità di ridurre il numero di eventuali nodi inutilizzati, individuabili mediante

un controllo di tipo heart beat o sul numero di richieste fatte su di esso, scalando verso il basso.

Avendo il Cloud alla sua base il principio di trasparenza totale per l'utente, è normale pensare che nella visione di quest'ultimo la nozione di nodo non sarà più disponibile, mentre all'interno della "nuvola" e quindi a livello di sistema tale concetto esisterà ancora. L'utente quindi non avrà a disposizione tutte le informazioni sulla locazione fisica del centro di tuple, necessarie per l'invocazione esplicita di un'operazione e per tanto dovrà essere possibile assegnare da parte dell'utente un nome al centro di tuple con il quale potrà interrogarlo. All'interno della nuvola invece rimarrà il concetto di nodo identificato dalla coppia

`< netid : portno >` così da non dover stravolgere la sintassi e la semantica utilizzate fino ad ora in TuCSoN per l'invocazione di un operazione di coordinazione. Pe far la coordinazione con un nodo Cloud ogni utente/agente TuCSoN dovrà prendere un determinato ACC, al quale sarà assegnato un nome univoco per agente così da identificarlo all'interno di una lista interna all'agente, sfruttando la funzione `addContext` ed inserendo il nome del nodo al quale si vuole agganciare. Si pensa quindi che sarà l'ACC a dover risalire ai parametri del nodo tramite l'associazione agente-nomeNodo. Tali parametri dovranno essere recuperati mediante alcune API(Cloud o TuCSoN). Successivamente tramite il comando `setContext`, sarà possibile settare l'ACC configurato e che l'agente potrà utilizzare ottenendo così un contesto Cloud da poter interrogare per ottenere coordinazione.

Nel sistema TuCSoN on Cloud dovrà inoltre essere estesa l'attuale implementazione della persistenza, fornendo all'utente la possibilità di salvare il contenuto di un centro di tuple presente sulla nuvola nel proprio computer. L'attuale implementazione può essere utilizzata per fornire un backup preventivo all'interno della nuvola, con la possibilità di sfruttarlo per la replicazione o creazione di ulteriori nodi durante lo scaling e per garantire una sorta di fault tolerance, consistenza e funzione di recovery nel sistema TuCSoN on Cloud.

Questi concetti e idee di soluzioni su TuCSoN applicato al Cloud ci permettono di stilare una lista preventiva delle cose su cui ci dovremmo focalizzare per la creazione del sistema TuCSoN on Cloud e ci permettono di proseguire e fare un'analisi di alcune delle piattaforme PaaS in commercio e capire così quale possa essere la più adatta al nostro scopo e fin dove il Cloud ci sarà di aiuto con le sue tecniche, dove l'attuale sistema TuCSoN arriva e quindi dove esso dovrà essere cambiato/esteso. Tale lista prevede:

- metriche di monitoring per TuCSoN
- dove arriva il monitor del cloud e dove dovremo intervenire noi in TuCSoN
- scaling out
- durante lo scaling out viene replicato anche il centro di tuple assieme a tutta la risorsa(fisica e servizio nodo TuCSoN)
- funzione di heart beat e checking status nodo per eventuale scaling in
- naming dei nodi per l'utente
- forwarding delle operazioni su nodi in base ai loro nomi

- associazione agente-nomeNodo per ricavare ip : porto ed effettuare le operazioni
- esecuzione delle operazioni su tutti nodi replicati così da avere consistenza nei dati e non nodi non consistenti
- salvataggio dati persistenza su macchina fisica dell'utente
- gestione della persistenza per nodi replicati e nodi splittati

Capitolo 3

In questo capitolo analizzeremo alcune delle PaaS presenti in commercio, in modo tale da avere tutti i dati necessari e le caratteristiche di ciascuno di esse per effettuare una scelta accurata della piattaforma che utilizzeremo per la realizzazione del sistema TuCSoN on Cloud.

Google App Engine

Google App Engine è uno stack di sviluppo offerto da Google di tipo Platform-as-a-Service progettato per applicazioni Web distribuite[4]. Come accade con altre offerte PaaS, Google App Engine gestisce molti dei dettagli di implementazione circa il mantenimento dei server virtuali, sistemi operativi, strumenti di sviluppo e politiche di scaling(se si seguono le regole stabilite da Google).

In Google App Engine l'elenco dei linguaggi di programmazione supportati prevede Java, Python, PHP e GO, inoltre l'ambiente Java supporta anche altri linguaggi che fanno uso del JRE, ad esempio, Rubin attraverso il progetto JRuby di Google App Engine. E' presente un SDK per ciascuno dei quattro principali linguaggi supportati così come un plugin per Eclipse. Java e Python sono pienamente supportati, ma Go e PHP sono ancora considerati in fase sperimentale. E' possibile inoltre associare le applicazioni con Compute Engine per integrare altre tecnologie come Node.js, C ++, Scala, Hadoop, MongoDB, Redis e altre ancora.

I programmi in Google App Engine vengono eseguiti all'interno di una sandbox che è logicamente isolata dalle altre applicazioni, dal sistema operativo e dall'hardware sottostante[4]. Esse non possono scrivere sul file system locale e possono accedere alle altre macchine su Internet tramite fetch URL e funzioni e-mail fornite nell'ambiente di sviluppo. Altri metodi di connessione e comunicazione sono ancora in fase beta o sono sviluppate da terze parti. Il codice, inoltre, viene eseguito solo in risposta a una richiesta Web, a fronte di un evento nella coda dei task o tramite un'operazione pianificata. Quando il codice viene eseguito in risposta ad uno di questi eventi, deve restituire un risultato entro 60 secondi. Tale limitazione è dovuta al fatto che il modello sandbox è progettato per isolare processi e ridurre il rischio che un processo malevolo su un server fisico possa interrompere le operazioni di altri processi sullo stesso server. Google sostiene che il limite dei 60 secondi sul calcolo incoraggi la modularità, l'utilizzo di unità limitate di computazione sulla sua piattaforma PaaS e che l'autoscaling è previsto solo per le applicazioni con bassa latenza, cioè quelle che presentano la necessità di rispondere alle richieste in meno di 1 secondo.

Google App Engine include data store transazionali e schema-less basati su coppie identificative chiave-valore[4]. Conosciuto con il nome di Datastore, questo servizio gestisce la complessa gestione dei dati che devono essere accessibili a più istanze di macchine e prevede che le entità siano organizzate in modo gerarchico. Gli sviluppatori e gli amministratori di applicazioni possono interrogare le entità del Datastore utilizzando GQL, un linguaggio di query SQL-like che gira nella console di amministrazione e nel runtime di Python. Gli

sviluppatori di Google App Engine hanno anche accesso ai servizi di database relazionali in Google Cloud SQL, che si basa su MySQL.

Gli sviluppatori hanno la possibilità di gestire le loro applicazioni e lo storage attraverso la Dashboard di Google App Engine, la quale permette loro un accesso ai dettagli delle loro istanze, code di attività e presenta anche dettagli sul Datastore, compresi i dettagli sugli indici, ricerche di testo, memcache e statistiche varie.

In Google App Engine i servizi sono fatturati in base all'utilizzo, ma Google offre gratuitamente fino a 28 ore di istanza, 1 GB di spazio di archiviazione datastore e 1 GB di traffico in entrata ed in uscita per applicazione al giorno.

Amazon AWS

Amazon Web Services (AWS) non è probabilmente il primo servizio di cloud che viene in mente quando si pensa ad un servizio Cloud di tipo Platform-as-a-Service. AWS è conosciuto principalmente come piattaforma Infrastructure-as-a-Service(IaaS) ed è praticamente considerabile come sinonimo di cloud computing pubblico in generale, ma molti dei servizi disponibili in AWS sono comparabili a quelli disponibili in offerte PaaS. Amazon[4] infatti offre un mix si servizi che comprendono la fornitura di istanze Cloud, realizzate principalmente tramite macchine virtuali configurabili, la fornitura di tecnologie di database, servizi di load balancing e auto-scaling, la fornitura anche di Cloud privati e tanti altri servizi. Dal momento che Amazon è in gran parte un servizio IaaS non c'è praticamente alcun limite ai linguaggi di programmazione e alle tecnologie server-side che è possibile installare ed eseguire. I linguaggi usati comunemente come Java, Python, Ruby, Perl ed altri, sono pienamente compatibili.

In AWS è presente un un vero ambiente di elaborazione virtuale, Amazon EC2, che consente di utilizzare le interfacce dei servizi Web per lanciare istanze con una varietà di sistemi operativi, caricarli con l'ambiente dell'applicazione, gestire le autorizzazioni di accesso della rete, ed eseguire l'immagine utilizzando uno o più sistemi in base alla volontà dell'utente[5]. Amazon EC2 inoltre fornisce una serie di funzionalità avanzate per la costruzione di applicazioni di classe enterprise scalabili e con proprietà di fault tolerance. Quando si avvia un istanza si avrà la possibilità di configurare il volume di root ed un Elastic Block Store (EBS). Il volume di root è associato con l'istanza e per impostazione predefinita viene eliminato al termine dell'istanza, mentre se si ha la necessità di memorizzare i dati persistentemente nel cloud si utilizza un EBS. Esso infatti offre uno storage persistente per le istanze Amazon EC2, collegato alla rete, che risulta indipendente dalla vita di un'istanza e consente di memorizzare gli oggetti fino a 5 TB di dimensione.

AWS presenta anche un servizio, denominato Amazon CloudWatch[5], con il quale è possibile monitorare le risorse cloud AWS e le applicazioni, a partire da Amazon EC2. Esso fornisce visibilità sull'utilizzo delle risorse, sulle prestazioni operative, e modelli globali sulle metriche quali l'utilizzo della CPU, del disco, di lettura e scrittura e del traffico di rete

[4] www.amazon.com

aggregando e memorizzando i dati di monitoraggio e rendendoli consultabili utilizzando le API dei servizi Web o strumenti su riga di comando offerti da Amazon. Con tale strumento è quindi possibile ottenere statistiche e visualizzare grafici. Per utilizzare Amazon CloudWatch, è sufficiente selezionare le istanze Amazon EC2 che si desidera monitorare ed inoltre è possibile inviare e memorizzare, tramite le API, parametri personalizzati ed importanti per le prestazioni operative della propria applicazione e che consentono di risolvere ed individuare le tendenze. È possibile utilizzare Amazon CloudWatch anche per inviare allarmi col fine di innescare attività di Elastic Load Balancing per aiutare a distribuire il traffico alle istanze create dall'Auto Scaling. Con CloudWatch, e senza costo aggiuntivo infatti è presente il servizio Auto Scaling[5], il quale consente di scalare automaticamente la capacità di Amazon EC2 in alto o in basso in base alle condizioni definite dall'utente. Con Auto Scaling, è possibile garantire che il numero di istanze EC2 di Amazon che si sta utilizzando scali senza problemi durante i picchi di domanda per mantenere alto il livello delle prestazioni, e che si ridimensioni automaticamente durante le pause della domanda per ridurre il numero di istanze inutilizzate e minimizzare i costi. Il servizio Elastic Load Balancing in modo automatico distribuisce il traffico in entrata all'applicazione su più istanze Amazon EC2[5], fornendo la capacità di bilanciamento del carico necessarie in risposta al traffico in entrata e consentendo di ottenere una maggiore fault tolerance nelle applicazioni. Elastic Load Balancing rileva le istanze che per qualche motivo non sono funzionanti o funzionano malamente e reindirizza automaticamente il traffico verso le istanze "sane" finché il funzionamento delle altre istante non è stato ripristinato e/o corretto.

AWS presenta inoltre altri due principali servizi, AWS Lambda e Amazon VPC[5].
AWS Lambda è un servizio di calcolo nel quale il codice di un applicazione viene eseguito, nell'ordine di pochi millisecondi, in risposta agli eventi e automaticamente gestisce le risorse di calcolo, rendendo più semplice la costruzione di applicazioni focalizzate sulla reattività di risposta a nuove informazioni. È inoltre possibile utilizzare AWS Lambda per creare nuovi servizi di back-end in cui le risorse di elaborazione vengono attivati automaticamente in base a richieste personalizzate.
Amazon VPC (Virtual Private Cloud), invece, consente di disporre di una sezione isolata e privata di AWS Cloud con la quale è possibile definire una topologia di rete virtuale[5]. Si può facilmente personalizzare la configurazione di rete per il proprio sistema, ad esempio, è possibile creare una sottorete rivolta al pubblico per i server web che permette l'accesso a Internet, e posizionare i sistemi back-end come database o applicazioni server in una sottorete privata senza accesso a Internet. È possibile sfruttare più livelli di sicurezza, compresi i gruppi di protezione ed elenchi di controllo di accesso di rete, per controllare e monitorare l'accesso alle istanze Amazon EC2 in ogni sottorete.

AWS prevede anche la compatibilità con svariate tecnologie di database, Oracle, MySQL e SQL Server possono essere istituite e gestite senza alcun problema ed inoltre Amazon mette a disposizione un proprio servizio chiamato Relational Database Service di Amazon (RDS)[5]. Esso consiste in un servizio in cui invece di creare la propria istanza database, è possibile utilizzare il servizio web RDS per fornire l'accesso e la gestione di un database di Amazon. Ci

sono un certo numero di vantaggi a questo approccio, in quanto Amazon si occupa di tutte le operazioni di amministrazione del database, come l'applicazione di patch software e la creazione di backup. RDS supporta anche la replica multi-zona in modo da ottenere caratteristiche di alta disponibilità senza il livello di sovraccarico amministrativo che si avrebbe se esso fosse eseguito su un proprio database.

AWS, quindi, offre davvero una vasta gamma di servizi di cloud computing. Per ogni servizio, si paga esattamente la quantità di risorse effettivamente necessarie senza sottostare a nessun impegno minimi o a contratti a lungo termine. AWS offre anche una varietà di servizi, alcuni forniti in combinazione ad altri senza alcun costo aggiuntivo. Il costo di utilizzo di Amazon EC2 si stima in base a diversi aspetti e configurazioni, esso comprende il monitoraggio ed il controllo di base delle risorse tramite Amazon CloudWatch affiancato da Auto Scaling, ma con un supplemento mensile è possibile attivare un monitoraggio più dettagliato. Il servizio Elastic Load Balancing che può essere utilizzato per distribuire il traffico tra le istanze Amazon EC2 incide sul costo mensile del servizio EC2 in base alle ore di utilizzo ed il servizio Amazon EBS fornisce tre principali tipi di volume nel momento di creazione di un istanza: General Purpose (SSD), Provisioning IOPS (SSD), e magnetici. I tre tipi di volume si differenziano per prestazioni e costi, in modo da poter scegliere lo storage con le migliori prestazioni ed il prezzo giusto per le esigenze dell'utente.

Se si utilizza AWS Lambda si paga solo per le richieste servite ed il tempo di calcolo necessario per eseguire il codice. La fatturazione del costo del servizio è misurata con incrementi di 100 millisecondi, il che rende conveniente e facile da scalare automaticamente alcune richieste al giorno a migliaia al secondo.

Amazon, per avvicinare nuovi clienti, offre però anche un livello gratuito di AWS nel quale è possibile eseguire una micro istanza gratuita di Amazon EC2 per un anno, e selezionare qualsiasi degli altri servizi proposti da Amazon, come Amazon Elastic Block Store, Amazon Elastic Load Balancing, e tanti altri, all'interno dei 12 mesi e rispettando certi limiti di utilizzo.

Heroku

Heroku è stato uno dei primi fornitori di servizi Platform-as-a-Service (PaaS) e tutt'ora continua ad essere quasi sinonimo di PaaS. La piattaforma Heroku offre ambienti di elaborazione astratti chiamati dynos, contenitori in stile Unix virtualizzati che eseguono i processi in ambienti isolati[6].

Heroku inizialmente è nato come una piattaforma per sviluppatori Ruby, ma da allora ha ampliato il suo supporto per altri linguaggi di programmazione come Python, Java, Scala, Clojure, e Node.js. Le applicazioni in Herkou sono una combinazione di codice e descrizioni di tutte le dipendenze, come ad esempio pacchetti, moduli e librerie, che devono essere presenti nell'ambiente per far eseguire correttamente l'applicazione[6]. Le dipendenze sono descritti in formato dichiarativo su un specifico file, Java per esempio specifica le dipendenze

nei file pom.xml, Ruby utilizza gemfiles, e le applicazioni Python definiscono le dipendenze nei file requirements.txt. Se l'applicazione dipende da un framework di uso comune, come Django o Rails per Ruby, Heroku può determinare come eseguirlo in modo autonomo. In altri casi, invece, si dovrà specificare i processi che devono essere eseguiti elencando i comandi in un file di processo, denominato Procfile da inserire nella cartella root dell'applicazione[6]. Il file contiene la maggior parte delle informazioni necessarie per istanziare i processi in dyno, ovvero le istruzioni e i comandi da eseguire e i dettagli di configurazione completa dei componenti necessari per distribuire un'applicazione. Le informazioni di configurazione descrivono i dettagli specifici della distribuzione, come ad esempio le credenziali e stringhe di connessione ad eventuali database. Mantenere queste informazioni separate dal codice sorgente rende più facile distribuire le applicazioni in ambienti di sviluppo, test e produzione. Le informazioni di configurazione vengono catturate in variabili di configurazione e sono visibili dall'applicazione attraverso variabili d'ambiente. Tutte le applicazioni su Heroku utilizzano un modello di processo (via Procfile) che consente loro di scalare verso l'alto o verso il basso istantaneamente da riga di comando o Dashboard e ciascuna di esse possiede una serie di dyno in esecuzione, gestita dal dyno manager.

La piattaforma Heroku utilizza git come mezzo principale per la distribuzione di applicazioni, ma sono forniti anche altri metodi per trasportare il codice sorgente per Heroku che utilizzano API proprietarie. Quando si crea un'applicazione su Heroku, esso associa un nuovo repository git remoto, tipicamente chiamato Heroku, con il repository git locale per l'applicazione[6]. Dopo aver inserito l'applicazione nel PaaS Heroku, quest'ultimo gestisce le prossime fasi del processo di distribuzione. Compila il codice sorgente (se necessario), recupera le dipendenze, e integra queste risorse in una struttura chiamata slug che contiene tutte le risorse necessarie per eseguire l'applicazione. Quando si distribuisce una nuova versione di un'applicazione, tutti i dyno attualmente in esecuzione vengono uccisi, e quelli nuovi (con la nuova versione) vengono avviati per sostituirli.

Le applicazioni che richiedono uno storage permanente possono richiedere accesso a Heroku Postgres. I database Postgres in Heroku sono accessibili da qualsiasi client Postgres, sia che esso sia nel cloud o sul dispositivo locale. Heroku fornisce anche una comoda utility di connessione per generare specifiche stringhe di connessione al database. L'ambiente Postgres, inoltre, comprende registri di ripristino che mantengono le informazioni riguardanti tutte le transazioni, i registri di ripristino vengono copiati su più data center in modo che in caso di un guasto hardware, lo stato del database potrebbe essere ricostruito.

La piattaforma Heroku comprende la gestione dei processi e dei servizi di database, ma talvolta ciò non è sempre sufficiente e quindi sono messe a disposizione alcune applicazioni di terze parti, chiamati add-ons, come servizi all'interno della piattaforma Heroku. I tipi di add-ons vanno da archivi di dati e servizi di ricerca a supporti per i dispositivi mobili e la messaggistica SMS. Se il database Postgres non è una buona soluzione per le esigenze di archiviazione dei dati persistenti, si ha ad esempio la possibilità di utilizzare Amazon RDS. Heroku realizza anche registri come log costituiti da flussi di eventi, e raccoglie il flusso di log

prodotti da tutti i processi in esecuzione in tutti i dyno, e le componenti della piattaforma Heroku, nel Logplex, un sistema real-time ad alte prestazioni per la realizzazione di registri di performance. È inoltre possibile monitorare i registri di un singolo dyno, mantenendo il canale aperto, e rimanendo in ascolto per ulteriori eventi[6].

Le tariffe della piattaforma Heroku sono stabilite in base all'uso delle risorse richieste e tutti gli add-ons disponibili sono acquistabili tramite un marketplace proprietario di Heroku. L'unico profilo gratuito prevede l'utilizzo di un singolo dyno(non vi è possibilità di scalare quindi) e una configurazione di Heroku Postgres molto limitata.

Cloudify

Cloudify è una piattaforma software open source di orchestrazione cloud scritta in Python e YAML(precedentemente in Java) basato su standard TOSCA(Topology and Orchestration Specification for Cloud Applications, ovvero Topologia e Orchestrazione Specifica per applicazioni cloud) che permette di automatizzare il processo di installazione, implementazione e anche post-distribuzione come il monitoraggio, correzione e auto-scaling dello stack dell'applicazione[7]. Per poter raggiungere questo livello di automazione Cloudify riceve input, noti come blueprints(progetti), che vengono compresi dal software di configurazione file e che descrivono come l'applicazione interagisce con il centro dati. Un blueprint consiste quindi in un piano di orchestrazione di un'applicazione, ispirato allo standard OASIS TOSCA, contenuto in un file YAML che descrive la topologia dell'applicazione come nodi e le loro relazioni/dipendenze, l'implementazione di ogni evento del ciclo di vita di ciascun nodo, la configurazione di ciascun nodo e facoltativamente i flussi di lavoro che descrivono l'automazione dei diversi processi, come l'installazione, l'aggiornamento e tanti altri.

Per distribuire e gestire un'applicazione con Cloudify. è necessario descriverla in un progetto basato su TOSCA, la recipe, "la ricetta", descrive il modello dell'applicazione che include i dettagli necessari per installare e gestire l'applicazione, come ad esempio i componenti delle applicazioni e delle loro dipendenze, la posizione dei binari, l'installazione e la configurazione di monitoraggio. Una tipica distribuzione comprende la fase di impostazione dell'infrastruttura, comprese configurazioni di rete (router, gruppi di protezione, IP pubblici), le risorse di calcolo e di storage, e assegnarli al dispositivo di rete e di storage rilevanti, la fase di assegnazione dei ruoli di ogni singola macchina e installazione dei relativi livelli applicazione di queste macchine tramite SSH o shell script, o anche chiamando gli strumenti di gestione della configurazione, come Chef o Puppet, la fase di avvio dei servizi competenti nel giusto ordine in base alle loro dipendenze, la fase di monitoraggio delle applicazioni e di registrazione, e per ultima la fase di gestione della domanda. Cloudify infatti continuamente monitora l'applicazione, anche sulla base di eventuali metriche definite, per verificare che essa soddisfi le richieste, ed in caso di violazione, guasto o di ridimensionamento eventi prenderà azioni correttive. Cloudify, inoltre, consente di aggiungere tecnologie di monitoraggio proprietarie, fornendo un mezzo per installare gli agenti di monitoraggio come parte

dell'installazione dell'applicazione, e riportare i suoi parametri al Cloudify policy engine e al database delle metriche. Cloudify poi permette di visualizzare le metriche di monitoraggio e le statistiche e fornisce le API per interrogarle da uno dei sistemi dell'utente.

Cloudify scalerà automaticamente l'applicazione utilizzando i criteri e le regole di ridimensionamento personalizzate, valutati in base a metriche personalizzate definite. Quando il Cloudify Orchestrator, un nuovo componente nell'architettura della piattaforma, individua una violazione delle regole, utilizzando le regole di scaling personalizzate, richiama un flusso di lavoro scale-out che può creare nodi aggiuntivi, applicare il bilanciamento del carico o eseguire qualsiasi altra serie di passaggi desiderati (con un flusso di lavoro personalizzato)[7].

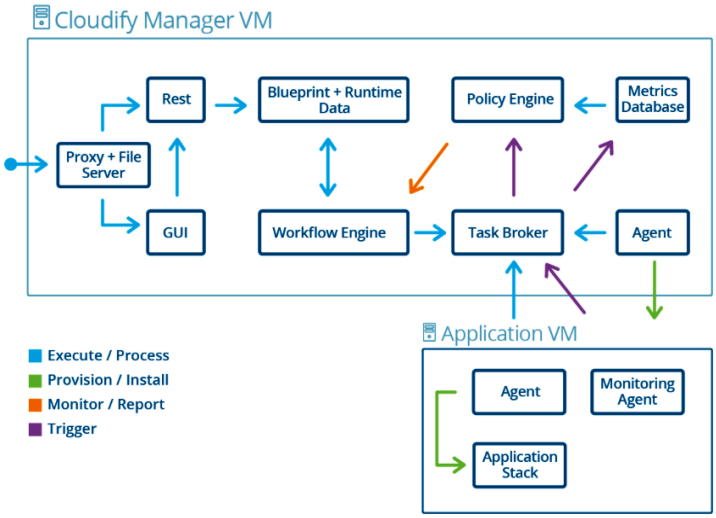
Cloudify è infine strettamente integrato con OpenStack, e, allo stesso tempo, include il supporto per altre piattaforme cloud così da consentire un alto livello di l'interoperabilità tra di esse. Alcune delle piattaforme cloud più popolari supportati da Cloudify sono AWS, CloudStack, Microsoft Azure e VMWare. Cloudify però supporta anche ambienti non cloud infatti può esser collegato ad un insieme predefinito di macchine e può gestire la distribuzione delle applicazioni entro i suoi confini. Questa configurazione potrebbe essere estremamente vantaggiosa per le organizzazioni che non dispongono di un ambiente di cloud-based o che sono in una transizione verso di esso.

Novità della nuova versione

Cloudify 3.1 ha una nuova architettura e si compone delle seguenti parti principali[7]:

- client CLI, un eseguibile scritto in Python che può essere eseguito su Windows, Linux e Mac OS e che offre 2 funzioni principali, manager bootstraping e gestione delle applicazioni. La prima funzione si tratta di una funzione opzionale in quanto è possibile installare un proprio tool se preferito e si occupa del bootstrapig, il processo di installazione del gestore di Cloudify che viene eseguito attraverso il CLI utilizzando un blueprint (denominato blueprint manager). Questo progetto, descrive le risorse cloud a disposizione compreso il gestore di VM. Il piano descrive inoltre l'installazione del manager utilizzando sia i pacchetti Deb o un contenitore Docker. La seconda funzione consiste nel fornire all'utente il set completo di funzioni per la distribuzione e la gestione di applicazioni, tra cui la navigazione log/eventi, il CLI funge da client REST verso l'interfaccia di gestione REST Cloudify.
- Manager (Orchestrator), è un "orchestratore" stateful che distribuisce e gestisce le applicazioni descritte nei piani di orchestrazione chiamati blueprints. La responsabilità principale del manager è quella di eseguire i processi di automazione descritti nei flusso di lavoro(script) e far eseguire agli agenti i comandi di esecuzione durante l'occorrenza di un problema.
- Agents(agenti), sono i responsabili della gestione dell'esecuzione dei comandi del manager utilizzando una serie di plugin. C'è un agente manager-side per la distribuzione delle applicazioni e un agente opzionale su ciascuna applicazione VM.

Gli agenti manager-side sono responsabili della gestione delle attività relative ad aspetti IaaS, ad esempio, la creazione di una macchina virtuale o di un network, l'associazione di un IP ad una VM. Possono essere utilizzati anche con altri strumenti come REST per eseguire task in modalità remota.
Gli agenti application-side sono opzionali e si trovano nelle VM delle applicazioni. L'utente può indicare nella blueprint la presenza di un agente e vengono installati dall'agente lato manager come parte della attività di creazione di macchine virtuali. Una volta in esecuzione, l'agente lato applicazione può installare plugin ed eseguire operazioni a livello locale. Le attività tipiche saranno di tipo middleware e di tipo implementativo di moduli applicativi.



Cloudify usa Nginx come frontend reverse proxy e file server, ed è controllato tramite API REST. Tali API coprono tutte le funzioni Cloud di orchestrazione e gestione ed è possibile utilizzarle attraverso il CLI non interattivo di Cloudify o scrivendo un proprio client REST.

E' presente anche una Web GUI che aggiunge visibilità e valore alla piattaforma Cloudify, e presenta diverse schermate:

- Blueprints, un catalogo di tutti gli schemi caricati
- Blueprint, un insieme di finestre per un progetto particolare, tra cui Blueprint Topology, Blueprint Network Topology, Blueprint Node, Blueprint Source
- Schermata delle distribuzioni
- Schermata della topologia di distribuzione
- Schermata della topologia del Network Deployment
- Schermata degli eventi di distribuzione
- Schermata delle metriche di performance
- Workflow Engine

Componenti dell'architettura

Cloudify utilizza un workflow engine per consentire qualsiasi processo di automazione attraverso flussi di lavoro già compresi o personalizzati[7]. Il motore di workflow è responsabile per il conteggio e l'orchestrazione delle attività di creazione/manipolazione delle componenti dell'applicazione. Per ottenere ciò il workflow engine interagisce con i dati Blueprint e i dati di runtime per ottenere le proprietà ed le informazioni de i plugin e scrivere task al broker task. Il workflow engine di Cloudify è costruito sulla base di Celery task broker, una coda/job asincrona dei task basata su scambio di messaggi distribuiti, e l'utente può scrivere flussi di lavoro personalizzati in Python utilizzando le API che forniscono l'accesso alle componenti della topologia e consentire passaggi di esecuzione e reporting di stato.

Cloudify usa Elasticsearch come archivio di dati dello stato della distribuzione ed il modello di distribuzione e dei dati di runtime vengono memorizzati come documenti JSON, ed usa InfluxDB come repository metriche di monitoraggio, Metrics Database[7].

Cloudify offre, inoltre, un policy engine, che gestisce le politiche di gestione personalizzate al fine di prendere decisioni di esecuzione sulla disponibilità, ecc. Cloudify usa Riemann.IO CEP come nucleo del policy engine e non è necessario alcun accesso o necessità di configurazione da parte dell'utente. Le politiche, scritte in Clojure, sono registrate, attivate, disattivate e cancellate dal motore di Workflow come parte del processo di orchestrazione.

Per quanto riguarda il Task Broker, Cloudify utilizza Celery con un bus di messaggi RabbitMQ per gestire le attività di distribuzione ed esecuzione. Ciascun task contiene: il blueprint e le proprietà di esecuzione (se applicabili) del nodo in questione, il plugin (nome e URL) che eseguirà l'operazione e il nome dell'operazione che il plugin deve eseguire[7].

Gli agenti Cloudify che si basano sulle entità worker di Celery ascoltano le code RabbitMQ per ottenere i compiti da eseguire. Una volta che un messaggio arriva, invocano il compito e riportano il risultato. Un agente può essere posizionato sia in remoto al Nodo che manipola (di default sul VM Cloudify Manager) sia sullo stesso host e può eseguire le seguenti operazioni:

- Installare plugin, l'agente riceve istruzioni su i plugin che dovrebbe utilizzare e li carica. I plugin possono essere aggiunti dinamicamente a runtime.
- Eseguire operazioni, gli agenti ottengono dei task dal workflow engine che rappresentano le istruzioni da eseguire su un plugin specifico. L'agente assegna uno dei suoi lavoratori per gestire l'operazione.

Infine plugin sono tool scritti in python per strumenti di terze parti che si desidera utilizzare con qualsiasi workflow Cloudify. Esempi notevoli sono i plugin per le API IaaS, come gli strumenti di gestione di configurazione dell'infrastruttura e anche i plugin per l'installazione e la configurazione di agenti di monitoraggio. Ciascun plugin dispone di metodi che corrispondono alla interfaccia delle operazioni di nodo.

Osservazioni

Google App Engine

Google App Engine sembra presenti una limitazione fondamentale, infatti sembra non sia possibile fornire parametri personalizzati per poter eseguire monitoring ed eventuali operazioni correttive di conseguenza. Google infatti specifica che tutte le applicazioni devono seguire una serie di regole ben precise per poter funzionare al meglio sulla sua piattaforma. Inoltre trovo che la restituzione di un risultato entro 60 secondi dopo la compilazione del codice a fronte sia un po' limitante per tutte quelle operazione che riguardano l'estrazione, la trasformazione ed il caricamento di grandi quantità di dati, e che quindi in questo caso un servizio di tipo IaaS sarebbe probabilmente una scelta migliore per voi. Google giustifica questa scelta dicendo che essa incentiva la modularità ed infatti l'alternativa al passaggio ad un servizio IaaS sarebbe nel configurare il programma affinché suddivida la grande mole di dati in subunità che possono essere elaborate entro il limite di 60 secondi.

Amazon AWS

I fornitori di servizi PaaS offrono in genere un mix di servizi di calcolo, storage, applicativi e la gestione del sistema. Tutti questi servizi sono disponibili anche nelle AWS di Amazon, nonostante esso sia un servizio di tipo IaaS e ciò permette di avere un certo livello di controllo anche sull'infrastruttura sottostante, oltre ai già sopracitati servizi forniti simili a quelli di un PaaS. Detto ciò si desume che Amazon AWS potrebbe esser un ottimo candidato su cui spostare TuCSoN.

Heroku

Heroku è una piattaforma poliglotta poiché consente di creare, eseguire, scalare le applicazioni in modo analogo utilizzando le dipendenze e i Procfile. Probabilmente sarebbe anch'essa un ottimo candidato per la sperimentazione con TuCSoN, ma la "scarna" configurazione di base ottenibile sottoscrivendo un contratto free non permetterebbe il test di operazioni di scaling automatico in quanto si disporrà solamente di un singolo dyno.

Cloudify

Cloudify non è un tradizionale framework PaaS, anche se può essere utilizzato per costruirne uno. Esso è stato progettato per gestire applicazioni non banali, che in genere hanno più livelli di complessità che spesso si discostano anche dai soliti stack Ruby, Node, Java. Utilizzando le recipe ed i blueprints per modellare l'applicazione, Cloudify consente agli utenti di automatizzare il deployment e la gestione di qualsiasi stack applicativo esistente. Inoltre, offre all'utente un grado di controllo sull'applicazione stessa molto più alto. Includendo il supporto per altre piattaforme cloud, consente di avere un alto livello di l'interoperabilità tra quelle scelte e quindi si presta come ottimo candidato per l'inserimento di TuCSoN nel Cloud anche a fronte dei risultati ottenuti nei precedenti studi e del fatto che risulta opensource.

Capitolo 4

Terminata l'analisi delle caratteristiche dell'attuale sistema TuCSoN e di alcuni servizi PaaS Cloud presenti sul mercato, in questo capitolo ci appresteremo a fornire alcune possibili implementazioni e soluzioni concettuali da applicare a parte dei componenti TuCSoN col dine di estenderli e soddisfare la lista dei requisiti stilata precedentemente, cercando inoltre di affrontare tutti gli aspetti fondamentali dell'elasticità e le eventuali problematiche che potrebbero insorgere.

Metriche di monitoring

Nel capitolo 2 abbiamo dimostrato che le classiche metriche di monitoring utilizzate negli attuali sistemi Cloud non possono essere del tutto adeguate per il nostro sistema TuCSoN on Cloud, in quanto le risorse di utilizzo degli host(fisici e virtuali) del sistema non saranno soggette a grossi carichi o a consumi elevati. Metriche come CPU, RAM e storage non dovranno essere monitorate poiché rimarranno sempre su percentuali di lavoro normali e non ci permetterebbero di attuare strategie di elasticità. A tal proposito è stato necessario pensare su quali altri fattori il sistema TuCSoN on Cloud potrebbe esser sensibile e sui quali potremmo riscontrare picchi di carico elevati che potrebbero andare ad influire sulle performance. A fronte della analisi precedente si pensa che le metriche di monitoring più consone ad un monitoraggio e controllo possano essere:

- Le richieste TCP verso ACCNode_Side e verso TransducerNode_Side.
  Il controllo delle prime permetteranno di attuare strategie di scalabilità atte ad avere elasticità dei nodi mentre il controllo delle seconde porterà l'elasticità dei Probes, o meglio delle risorse ambientali.
- Il numero di reaction ed eventi nella RespectVM che triggherano a fronte di un'operazione. Queste metriche ed il loro controllo hanno lo scopo di mantenere alte le prestazione della parte ReSPecT del sistema TuCSoN. Le reaction le troviamo nel package `alice.tuplecenter.core` nelle classi `AbstractEvent` e `Reaction`, mentre gli eventi nella RespectVM nel `package alice.respect.core` nella classe `InternalEvent`.
- Il numero di TupleCenter TC presenti su un nodo. Esso infatti da alcuni test fatti si è concluso che può incidere sulle performance del nodo stesso
- Memoria consumata nel JVM per Nodo e TCContainer. A causa di TuProlog si possono verificare situazioni in cui vi è un alto consumo di memoria della JVM, di conseguenza monitorare tale risorsa permetterebbe di avere ulteriore elasticità a livello di nodo e tuple center.

A livello concettuale ed implementativo non è possibile definire in che modo TuCSoN debba essere esteso per fornire le metriche al meccanismo di monitoring del Cloud, poiché quest'ultimo e le relative tecniche e funzionalità per attuare lo scaling out/in e annessa

replicazione o ripartizione del nodo scalato dipendono dalla piattaforma Cloud utilizzata per TuCSoN on Cloud.

Funzione di heart beat/checking status del nodo/TC

Nella situazione attuale, per come è progettato il sistema TuCSoN potrebbero incorrere problemi, come alta latenza o un nodo cade in modo inaspettato, e solamente nel momento in cui un agente/utente effettua un'operazione si verrebbe a sapere ciò poiché viene lanciata un'eccezione. Questa situazione rappresenta una caratteristica limitante e negativa di TuCSoN quindi si è pensato che in un sistema TuCSoN on Cloud si debba realizzare una funzionalità di controllo e verifica dello stato dei nodi, la quale dovrà essere eseguita in modo autonomo dal sistema e che, in caso di malfunzionamento, permetterà di evitare questa problematica richiamando determinate strategie di scaling. Per realizzare tale funzionalità si è pensato a due possibili soluzioni:

- Viene creata una nuova entità, esterna(mediante l'implementazione di un nuovo componente) o all'interno dell'ACCNode_Side(tramite un Thread), che ad intervalli regolari ed in modo parallelo alle classiche operazioni di coordinazione esegua una operazione(ad esempio di semplice lettura) per verificare la "salute" del nodo/TC.
- Ad intervalli regolari ed in modo automatico viene previsto uno scambio di pacchetti UDP di pochi byte tra i due ACC associati(Agent e Node Side) per innescare la più semplice operazione da eseguire su un centro di tuple e controllare così il corretto funzionamento del nodo. Tale scambio di pacchetti può essere eseguito anche sui due Transducer associati(Node e Probe Side).

Con questa nuova funzionalità anche nel caso in cui per un lungo periodo l'agente/utente non effettui operazioni, il sistema riuscirà a verificare lo stato dei nodi ed in caso di eccezioni potrà richiamare meccanismi elastici di correzione in base alle strategie di elasticità definite.

Naming dei nodi

La caratteristica di trasparenza totale del Cloud rappresenta un requisito importante da soddisfare per la creazione del sistema TuCSoN on Cloud. Nell'attuale implementazione di TuCSoN l'utente per poter effettuare una operazione deve essere in possesso di informazioni di rete che in un ipotetico contesto Cloud non sarebbero accessibili o rintracciabili per questioni di sicurezza. Per soddisfare quindi il requisito di trasparenza totale delle informazioni dei nodi si è pensato di creare un meccanismo di naming di quest'ultimi così da non avere più un'invocazione esplicita delle operazioni sui nodi e che consenta all'utente di richiedere un'operazione nella forma `op ? tcName @ nodeName`. Per non stravolgere la sintassi e la semantica delle operazioni utilizzate attualmente in TuCSoN si è deciso di lasciare inalterata l'invocazione delle operazioni all'interno del sistema effettuate dall'ACCNode_Side e che le informazioni di rete, ovvero `ip` e `portno`, saranno ricavate dal nome del nodo. Si pensa che l'assegnazione di un nome da parte dell'utente al nodo

interessato potrà avvenire attraverso il CLI, ed il salvataggio di tale informazione associata ad indirizzo IP e porta di ascolto del servizio, dovrà essere a carico di una nuova entità, chiamata ElasticityController, e potrà essere realizzata all'interno del sistema mediante le seguenti soluzioni:

- L'implementazione di un nuovo oggetto chiamato NodeInfo nei quali compaiono i campi NodeName, ip, portno che rappresenteranno le informazioni di ciascun nodo.

```
1  public class NodeInfo{
2
3      String nodeName, ip, portno;
4
5      public NodeInfo(String nodeName, String ip, portno){
6
7          // i parametri ip e portno passati al costruttore saranno
8          // ricavati da metodi delle API Cloud che ci permetteranno
9          // di recuperare indirizzo IP e porta di ascolto del nodo
10         // appena creato/replicato
11
12         this.nodeName = nodeName;
13         this.ip = ip;
14         this.portno = portno;
15
16     }
17
18     // Di seguito dovranno essere implementati i metodi
19     // per restituire i singoli campi
20
21 }
22
```

  L'insieme di tutti questi oggetti sarà creato e gestito dalla nuova entità, la quale nel momento di prima creazione del nodo si occuperà della creazione dell'oggetto e del suo inserimento in una struttura apposita, come ad esempio un ArrayList<NodeInfo>(anche una Map o una HashMap andrebbero bene), e nel momento in cui saranno attuate politiche di scaling provvederà, in caso di replicazione e/o split di un nodo a creare un nuovo oggetto NodeInfo con le nuove informazioni del nodo e ad inserirlo nella struttura, mentre nel caso di scaling in a recuperare l'oggetto corrispondente al nodo eliminato in base al suo campo NodeName e ad eliminarlo dalla struttura. Tale struttura sarà interrogata dall'ACCNode_Side e dal tusconNodeService per il forwarding delle operazioni. Le operazioni dell'utente quindi diverranno della forma `tcName @ nodeName` e nell'ACCNode_Side verrà confrontato il nodeName inserito da CLI con quelli presenti negli oggetti della struttura ed in caso di match, l'ACCNode_Side provvederà a recuperare le restanti informazioni e ad invocare l'operazione vera e propria. Stessa cosa avverrà da parte del TucsonNodeService.

- Un db che sarà interrogato dall'ACCNode_Side e dal TucsonNodeService per recuperare i campi ip e portno. Se nodo verrà replicato, splittato o eliminato basterà intervenire sul db inserendo o eliminando la voce corrispondente.
- Un file xml con una struttura ben definita, che sarà aggiornato in caso di scale in/out.

Tutte queste soluzioni mantengono separata la logica della coordinazione da quella della gestione dell'elasticità, fornendo anche una sorta di modularità al sistema.

Consistenza dei nodi

L'esecuzione di tecniche di scaling all'interno del sistema TuCSoN on Cloud potrà portare a problemi di consistenza dei nodi. Infatti in base alla tipologia di scaling effettuato si potrebbero presentare scenari in cui il sistema dovrà intervenire per mantenere consistenti le tuple all'interno dei centri di tuple attraverso un forwarding multiplo delle operazioni da effettuare. Le principali tecniche di scaling out consistono in replicazione e separazione/split, mentre quelle di scaling in nella aggregazione ed eliminazione(di risorse inattive). Oltre alla tipologia della tecnica di scaling utilizzata per fornire elasticità anche la natura della operazione da svolgere andrà ad incidere sulle fasi da attuare per mantenere consistenti i nodi del Cloud. Affrontando le varie tecniche di scaling out una ad una andremo ad individuare a fronte di determinate operazioni quali dovrebbero essere le soluzioni per mantenere la consistenza all'interno del sistema:

- Scaling out: replicazione - A seguito di un operazione di replicazione di un nodo nel sistema andranno a trovarsi due o più copie dello stesso nodo e stesso tuple center, tale situazione non risulta critica se l'utente si limita ad eseguire semplici operazioni di lettura, come la `rd, rdp, get, rd_all, no_all, urd, urdp, uno, unop,` quindi tali operazioni possono essere eseguite su un solo nodo. Nel momento in cui l'operazione di coordinazione da eseguire comporterà una modifica del tuple space, invece, tale operazione dovrà essere eseguita su tutte le copie del nodo replicato. Le operazioni in questione sono la `out, in, inp, set, out_all, in_all, uin, uinp`. Facendo riferimento alla soluzione del naming presentata precedentemente l'ACCNode_Side dovrà risalire attraverso il nome a tutti i nodi presenti nella struttura che raccoglie le informazioni dei nodi ed inoltrare ad essi l'operazione richiesta dall'utente. Successivamente solo uno dei risultati ricevuti saranno poi inviati all'utente.
- Scaling out: separazione/split - Se un nodo viene diviso in due o più parti per mantenere elevate le prestazioni, qualsiasi operazione che l'utente vorrà eseguire dovrà essere eseguita su tutti i nodi. I risultati, in caso di operazioni che richiedono la restituzione di una lista di tuple, dovranno essere aggregati e inviati all'utente, mentre se si tratta di operazioni che richiedono la restituzione una singola tupla che ha corrispondenza con il template richiesto in caso di successo sarà mandata all'utente la tupla/o le tuple che ha/hanno fatto match poiché esse possono essere distribuite tra le varie parti del nodo originale mentre nel nodo in cui non è presente alcun match verrà sospesa l'esecuzione e ripresa solamente quando sarà trovata una corrispondenza.
- Le tecniche di scaling in, in sostanza possono creare le stesse situazioni dello scaling out, ma con una complessità minore a livello di istanze poiché esse sono mirate a ridurre il numero di nodi presenti nel sistema aggregando alcune parte dei nodi(e riconducendomi quindi alla situazione creatasi dopo uno split) o eliminando nodi replicati. Solo nel caso in cui l'aggregazione o la eliminazione dei nodi comporti come risultato la finale presenza di

un singolo nodo non si presenterebbero problemi di consistenza vista l'unica istanza del nodo presente nel sistema.

Persistenza

La persistenza nell'attuale sistema TuCSoN avviene attraverso due principali fasi:

- Nel caso in cui non sia già presente un file si procede alla creazione di un file xml ed in esso viene salvato lo stato attuale del nodo con sue le caratteristiche possedute al momento dell'abilitazione della persistenza. Nel file creato le tipologie di strutture delle quali si tiene traccia sono Tuple, SpecTuples e Predicates e la struttura del file xml segue la seguente struttura:

```
<persistency>
    <snapshot tc="'@'(default,':'(localhost,20504))" time="2014-10-13_21.06.53">
        <tuples>
            <tuple> ... </tuple>
        </tuples>
       <specTuples>
            <spec> ... <spec>
        </specTuples>
        <predicates>
            <predicate> ... </predicate>
        </predicates>
    </snapshot>
<updates time="2014-10-13_21.07.00">
        <update action="addition" subject="tuple"> ... </update>
</updates>
</persistency>
```

- Durante tutto il periodo in cui rimane abilitata la persistenza per ogni update(inserimento, cancellazione di tuple) si controlla prima se è presente il file e successivamente si procede al suo aggiornamento andando ad indicare le tuple che hanno subito un cambiamento.

L'utente può abilitare la persistenza mediante l'invocazione di una operazione di `out(enable_persistency)`e allo stesso modo la disabilitazione avviene eseguendo una operazione di `out(disable_persistency)`.

L'attuale salvataggio del file xml avviene all'interno della directory "./persistent/" (vedi campo privato `PERSISTENCY_PATH` della classe `TucsonNodeService`) e tale directory risiederà all'interno della nuvola rendendo inaccessibile tale file all'utente, mentre si pensa che in un contesto Cloud sia utile estenderne il salvataggio in modo tale che esso possa essere a disposizione dell'utente. Per realizzare questa estensione si pensa che sia necessario affiancare alle attuali primitive di abilitazione e disabilitazione un'altra coppia di operazioni di tipo `out(enable_phpersistency)`e `out(disable_phpersistency)` con lo stesso fine, ma che permetteranno il salvataggio del file xml su un servizio di storage persistente offerto dal provider del Cloud e accessibile dall'utente o su altri servizi di storage ampiamente disponibili in commercio(vedi Dropbox[5], Google Drive[6] o Box[7]) che mettono a disposizione delle API

[5] https://www.dropbox.com

[6] https://www.google.com/intl/it_it/drive/

[7] https://www.box.com/it_IT/

per il salvataggio di dati sui loro storage. In caso di salvataggio della persistenza su dischi accessibili all'utente all'interno del file xml si dovranno andare a modificare le informazioni del figlio snapshot poiché, per questioni di sicurezza e trasparenza, non dovranno essere visualizzate le informazioni di rete del nodo ma sarà semplicemente visualizzato il suo nome.

Dualmente alla consistenza anche per lo sviluppo della persistenza si dovranno inoltre tenere conto delle problematiche derivanti dalle operazioni di scaling sui nodi dato che anche per questa funzionalità del sistema si presenteranno nuovi scenari e relative questioni che non saranno risolvibili con l'attuale implementazione della persistenza. Le situazioni analizzate sono:

- Scaling out: replicazione - Se un nodo viene replicato e su di esso non è abilitata la persistenza nel momento in cui l'utente esegue una operazione di `out(enable_persistency)` tale operazione dovrà essere inoltrata su tutti i nodi replicati, proprio come si è pensato per una semplice `out`, e ciò porterà ad avere di più file xml uguali, ciascuno per il relativo nodo. Si è pensato che sia necessario utilizzare tale modalità poiché è impossibile determinare a priori quale nodo, ed il relativo file, poi potrà essere eliminato a causa di una scale in e così si disporrà sempre di una copia del file xml. In caso di abilitazione della persistenza con salvataggio del file su storage persistente il file che verrà salvato dovrà essere uno solo per evitare di avere multiple copie dello stesso file sul disco.
- Scaling out: replicazione - Se un nodo viene replicato e su di esso è già stata abilita la persistenza anche sul nuovo nodo dovrà essere abilitata in modo automatico dal sistema ed il successivo comportamento risulterà analogo a quello visto in precedenza.
- Scaling out: separazione/split - Se un nodo viene diviso in più istanze e su di esso non è abilitata la persistenza quando verrà eseguita una operazione di abilitazione `out(enable_persistency)` della persistenza essa dovrà essere eseguita su tutte le parti distribuite del nodo e così si andranno a creare un numero pari alle istanze del nodo di file xml differenti che rappresenteranno lo snapshot della relativa porzione di nodo. In caso di salvataggio fisico del file e di eventuale scale in con aggregazione di tutte le istanze o di una sottoporzione di esse, i relativi file dovranno essere uniti in uno unico e poi salvati sullo storage Cloud dedicato.
- Scaling in: aggregazione ed eliminazione - tali scenari sono già stati analizzati nei punti precedenti come parte integrante delle soluzioni per lo scaling out.

Capitolo 5

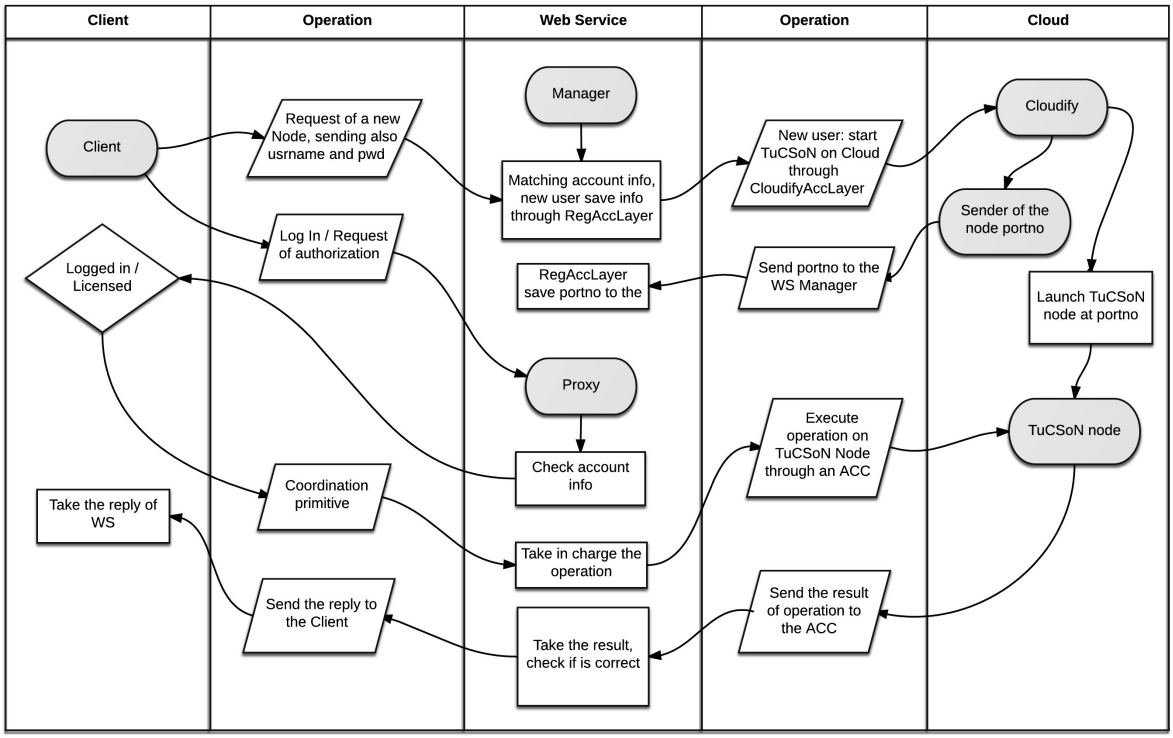
Caso di studio

In quest'ultimo capitolo è proposto un caso di studio nel quale si verificherà la correttezza del lavoro svolto da Guerra[8] su TuCSoN on Cloud, andando prima ad installare su alcune macchine del dasLab la piattaforma Cloudify(versione 2.7.1 GA)[8], poi inserendo l'applicazione nella piattaforma.

L'applicativo è formato da diversi componenti, ciascuno dei quali ricopre un ruolo differente all'interno dei livelli dell'architettura del sistema, che è suddivisa in:

- Client: entità che utilizza la coordinazione offerta dal livello Web Service
- Web Service: entità che, attraverso i suoi componenti interni, gestisce gli account registrati e regola l'interazione dei Client verso i nodi comportandosi da proxy e nascondendo la visione del nodo al client per rispettare il vincolo di trasparenza richiesto dal Cloud
- Cloud: qui sono compresi i nodi TuCSoN, programmati tramite i propri centri di tuple, ed il componente che ha il compito di spedire il numero di porta del nodo TuCSoN creato al Web Service, il quale poi lo salverà nelle informazioni dell'account specifico. I vari componenti sono istanziati attraverso la piattaforma Cloudify e che fornisce l'infrastruttura Cloud sul quale risiederanno i nostri nodi.

Il comportamento del sistema e la sequenza delle operazioni svolte da ciascun componente dell'architettura nel suo complesso può essere riassunto come segue:



[8] http://apice.unibo.it/xwiki/bin/view/Courses/SdLm1314ProjectsUIAccountingTucsonOnCloud

Modifiche nel codice

L'intero progetto è stato eseguito e progettato su un sistema Windows per tanto si è ritenuto necessario modificarlo in modo che a runtime ed in modo dinamico esso possa eseguire le opportune operazioni anche nel caso in cui sia testato su sistemi Unix-like. Di seguito riporto le modifiche eseguite all'interno delle varie classi, alcune necessarie per aggiornare i path delle risorse utilizzate, e quelle eseguite nella relativa recipe di Cloudify(non sono riportati i metodi o i costruttori completi per questioni di praticità).

Progetto TucsonOnCloudRESTful:

- Il file `Registry.xml` è stato inserito all'interno di una cartella nominata `regFile` nel package `base` ed è stato aggiornato il `registryPath` all'interno del `RegistryAccessLayer`
- Nel `CloudifyAccessLayer` nel metodo `newNode` è stato ritenuto opportuno precisare come il path debba essere modificato in base al sistema e ne sono stati forniti due esempi(per Windows e per sistemi Unix-like).
  Il metodo `execCmd` è stato modificato per agire in modo dinamico al sistema su cui esso è in esecuzione verificando prima la natura del sistema operativo e richiamando poi i comandi opportuni per il lancio della recipe TuCSoNCloud su Cloudify.
- E' stato modificato anche il parametro path passato al costruttore dell'oggetto `CloudifyAccessLayer` posto all'interno del costruttore della classe `Manager`.

I progetti SendPort e WSClient sono rimasti inalterati, mentre gli ultimi cambiamenti eseguiti sulle recipe Cloudify sono anch'essi mirati ad eseguire il file .sh(quest'ultimo completamente creato).

Installazione dell'ambiente di sviluppo e procedura di test

Prima dell'installazione della piattaforma è necessario configurare sulla propria macchina la variabile di ambiente `JAVA_HOME` e aggiungerla al `PATH`. Per eseguire tale operazione si è creato il file .bash_profile all'interno della directory root dell'account nella macchina del dasLab assegnata e sono state aggiunte a tale file queste linee:

```
export JAVA_HOME=$(/usr/libexec/java_home)
export PATH=$PATH:JAVA_HOME/bin
```

Il primo export assegna alla variabile di ambiente `JAVA_HOME` il percorso della più recente installazione di Java sulla macchina. Successivamente si è salvato il file e si è utilizzato il comando `source .bash_profile` per rendere effettive le modifiche senza necessitare del riavvio della macchina.

Una volta scaricata e scompattata, la piattaforma è accessibile tramite una Command Line Interface CLI dedicata presente all'interno dei suoi file di distribuzione. La CLI è lanciabile tramite il comando da terminale `<directory's Cloudify folder>/bin $ ./cloudify.sh` .

Per il caso di studio di TuCSoN on Cloud si è deciso di sfruttare l'opportunità offerta da Cloudify di installare sulla propria macchina un Cloud locale tramite il comando `bootstrap-localcloud`. Un Cloud locale consiste in un vero e proprio ambiente di emulazione di una "nuvola" che permette all'utente di eseguire tutta la gestione Cloudify e di servizi applicativi su una singola macchina. Utilizzando il comando `teardown-localcloud` potremmo terminare i processi alla base del Cloud locale.

Siccome il Web Service è esterno al Cloud esso è stato installato in un server "fisico" con installato Apace Tomcat[9], simulato attraverso EclispeEE, e sono stati precedentemente modificati i vari path che puntano a Cloudify. I path da modificare con quelli della propria macchina sono i seguenti(evidenziati in blu):

- Classe RegistryAccessLayer

```
public class RegistryAccessLayer {
   // Windows path example
   // private String registryPath = "C:\\Users\\Luca\\Desktop\\Registry.xml";
   // Unix-like path example
   private String registryPath = "/Users/rc/Desktop/workspace/TucsonOnCloudWSRESTful/TucsonOnCloudRESTful/src/base/regFile/
Registry.xml";
   . . .
```

- Classe CloudifyAccessLayer

```
public String newNode(String username, String password) throws IOException {
   System.out.println("I'm starting the service and creating a Node for:" + username);
   boolean result = false;
   boolean first_node = (RAL.activeAccounts() == 0);
   if(!RAL.addNewUser(username, password))
      return "Problems";
   if(first_node){
      // Path of TuCSoN Node service recipe at the end of command
      // Windows path example C:/cloudify/gigaspaces-cloudify-2.7.0-ga/recipes/apps/TuCSoNCloud/Node
      // I'm running on a Unix-like system
      result = execCmd("connect " + cloudAddress + ";use-application TuCSoNCloud;install-service /Users/rc/Desktop/cloudify/
recipes/apps/TuCSoNCloud/Node/");
   . . .
```

- Classe Manager

```
public Manager() throws ParserConfigurationException, TransformerConfigurationException,
TransformerFactoryConfigurationError {
   super();
   builder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
   transformer = TransformerFactory.newInstance().newTransformer();
   //CAL = new CloudifyAccessLayer("C:\\cloudify\\gigaspaces-cloudify-2.7.0-ga\\bin\\cloudify.bat", "127.0.0.1");
   CAL = new CloudifyAccessLayer("/Users/rc/Desktop/cloudify/bin/cloudify.sh", "127.0.0.1");
   RAL = new RegistryAccessLayer();
   . . .
```

[9] http://tomcat.apache.org

Guida per la sua configurazione su Eclipse : http://www.eclipse.org/webtools/jst/components/ws/1.5/tutorials/InstallTomcat/InstallTomcat.html

I passi eseguiti successivamente sono stati i seguenti:

- Inserimento dei file della recipe TuCSoN on Cloud all'interno della directory apps di cloudify
- Lancio di Cloudify da shell
- Creazione del Cloud locale tramite il comando `bootstrap-localcloud`
- Esecuzione del Web Service TucsonOnCloudRESTful da EclipseEE, attraverso la configurazione Run on a Server
- Esecuzione del TestingEnv del progetto Test come semplice applicazione Java per vedere in azione la coordinazione as a service. Di seguito è proposto il log delle operazioni svolte dal Web Service visualizzato sulla console di EclipseEE.

```
apr 02, 2015 9:55:43 AM org.apache.coyote.AbstractProtocol start
INFORMAZIONI: Starting ProtocolHandler ["ajp-nio-8009"]
apr 02, 2015 9:55:43 AM org.apache.catalina.startup.Catalina start
INFORMAZIONI: Server startup in 573 ms
OP: new-node
I'm starting the service and creating a Node for:luca

Starting Non-Interactive Shell
>>> connect 127.0.0.1
Connected successfully
>>> use-application TuCSoNCloud
Using application TuCSoNCloud
>>> install-service /Users/rc/Desktop/cloudify/recipes/apps/TuCSoNCloud/Node/
Uploading /var/folders/lj/y0zk23mj3jg1gpgl44c3wg8r0000gn/T/ServicePackage9067201366809003175.tmp/Node.zip
Installing service Node with 1 instances
Waiting for life cycle events of service Node
.........
[localhost/127.0.0.1] - Node-1 PRE_START invoked
...
[localhost/127.0.0.1] - Node-1 PRE_START completed, duration: 1,7 seconds [OK]
[localhost/127.0.0.1] - Node-1 START invoked
Nuova porta consegnata
Assegnata
.....
Successfully installed 1 instances out of 1 for service Node


Service "Node" successfully installed
>>> exit
Good Bye!

F1B07DDAF345BCFDD87965B35973AA0B: Documento ricevuto

F1B07DDAF345BCFDD87965B35973AA0B: Autorizzazione in corso ...
F1B07DDAF345BCFDD87965B35973AA0B: Utente = luca loggato correttamente
F1B07DDAF345BCFDD87965B35973AA0B: Documento ricevuto

F1B07DDAF345BCFDD87965B35973AA0B: Username = luca; Operation = out; Template = res(testo)
F1B07DDAF345BCFDD87965B35973AA0B: autenticazione in corso...
F1B07DDAF345BCFDD87965B35973AA0B: autenticato utente
....[OperationHandler (wsAgent)]: requesting op 1, res(testo), '@'(default,':'(localhost,'20506'))
F1B07DDAF345BCFDD87965B35973AA0B: Risposta res(testo)
```

- Verifica della corretta installazione dell'applicazione TuCSoN on Cloud tramite l'accesso all'indirizzo su cui Cloudify fornisce il servizio di management e ai log da console.

Cloudify permette di visionare lo stato della propria applicazione e dei relativi servizi accedendo all'indirizzo IP e porta indicati al momento del bootstraping del Cloud sotto la voce CLOUDIFY MANAGEMENT. In questo caso di studio, una volta terminata l'operazione di bootstrap del local-cloud si è appreso che l'indirizzo di default al quale accedere per il management è `http://127.0.0.1:8099/`. e per il quale non è necessario disporre di credenziali di accesso per potervi entrare.

Conclusioni

Questo progetto si è sviluppato in un iniziale apprendimento delle principali tecniche di scalabilità utilizzate negli attuali sistemi Cloud ed è proseguito in un'analisi di ciò che il sistema TuCSoN dovrebbe acquisire se inserito in un contesto Cloud. Si è cercato di fornire una sorta di linea guida nella quale sono state espresse tutte le possibili estensioni da implementare nell'attuale sistema TuCSoN per renderlo elastico e permettergli di scalare i propri nodi in modo autonomo e del tutto trasparente all'utente, fornendo inoltre diverse soluzioni applicative. Poiché alcune di queste estensioni potrebbero essere realizzate solamente nel momento in cui il sistema viene inserito concretamente in un contesto Cloud, in quanto strettamente dipendenti dall'architettura del Cloud e dei suoi meccanismi di monitoring e di scalabilità, si è fatta anche una analisi di alcune delle attuali e più famose piattaforme presenti sul mercato nella quale si sono evidenziati pregi e difetti e delle quali si è espresso un parere in merito ad un loro utilizzo per il futuro sistema TuCSoN on Cloud.

E' stato proposto anche un caso di studio nel quale si è modificato parte del lavoro di Guerra per rendere il progetto universale, esso infatti in principio era compatibile solamente con sistemi Windows mentre con le modifiche apportate esso è compatibile anche con sistemi Unix-like. Dai Test effettuati si è riscontrato che, nel caso in cui la fase di pre-start del lancio dell'applicazione di TuCSoN on Cloud da parte di Cloudify impieghi un tempo abbastanza maggiore di quello usuale(circa 2 sec), la sincronizzazione tra i componenti sembra non più rispettata producendo un malfunzionamento dovuto ad una corsa critica. Per tanto si ritiene opportuno che i primi sviluppi futuri riguardino la risoluzione di tale problematica sviluppando una forma più robusta di sincronizzazione tra i vari componenti.

Bibliografia

[1] Richard Hill, Laurie Hirsch, Peter Lake, Siavash Moshiri - “Guide to Cloud Computing - Principles and Practice”, 2013

[2] Borko Furht, Armando Escalante - “Handbook of Cloud Computing”, 2010

[3] Andrea Omicini, Stefano Mariani - “The TuCSoN Coordination Model and Technology. A Guide” , TuCSoN v.1.11.0.0209, Guide v.1.2.1, 22 Ottobre 2014
http://www.slideshare.net/andreaomicini/the-tucson-coordination-model-technology-a-guide

[4] Google App Engine - https://cloud.google.com/appengine/docs

[5] Amazon AWS - http://aws.amazon.com

[6] Heroku - https://devcenter.heroku.com/articles/how-heroku-works

[7] Cloudify - http://getcloudify.org

[8] http://getcloudify.org/guide/2.7/qsg/qsg.html#Getting_Started