ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

TESI DI LAUREA
in
Sistemi Distribuiti M

INFRASTRUTTURE BASATE SU EDGE COMPUTING PER SUPPORTO A SERVIZI MOBILI IN AMBIENTI OSTILI

CANDIDATO:
Michele Solimando

RELATORE:
Chiar.mo Prof. Paolo Bellavista

CORRELATORI:
Chiar.mo Prof. Antonio Corradi
Dott. Ing. Alessandro Zanni

Anno Accademico 2015/2016

Sessione I

INDICE

INDICE DELLE FIGURE

CAPITOLO 1: INTRODUZIONE

La stragrande maggioranza dei sistemi informatici, che offrono servizi on line, si appoggia attualmente ad infrastrutture Cloud. Le risorse vengono affittate agli utenti e allocate dinamicamente in base al loro reale utilizzo e al tipo di contratto stipulato. Tra le modalità di connessione abilitate dal Cloud, la più evoluta e pervasiva è rappresentata dall'Internet of Things. Dispositivi eterogenei e con capacità computazionali anche molto limitate, come ad esempio elettrodomestici, mezzi di trasporto, o semplici sensori e attuatori, sono connessi, spesso tramite reti senza fili, e demandano le computazioni più pesanti e l'analisi dei dati raccolti alle centrali Cloud remote, presenti nella rete Internet esterna. L'evoluzione tecnologica dei dispositivi collegati alla rete, insieme al loro impiego in settori molto specializzati e professionali, produce una nuova classe di applicazioni che richiede caratteristiche molto stringenti per quanto riguarda la modalità di interazione e connessione tra la sorgente di produzione dei dati e la risorsa che effettivamente li elabora, estraendone del contenuto. Le specifiche richieste spesso non possono essere garantite lungo il cammino che porta dai dispositivi utente fino alle centrali Cloud; conseguenza dovuta per lo più alla lontananza fisica dei partecipanti, la cui comunicazione attraversa molteplici reti. Le applicazioni riguardanti la realtà aumentata, o quelle dedicate all'intrattenimento online e alle connessioni veicolari sono solo alcuni esempi delle aree applicative che necessitano di una

latenza molto bassa, di supporto alla mobilità e di conoscenza del contesto.

Un nuovo paradigma, affacciatosi recentemente sul panorama tecnologico delle architetture di rete, si pone come obiettivo quello di ampliare le caratteristiche del Cloud, rendendo il modello che ne consegue particolarmente adatto all'esecuzione dei servizi e delle applicazioni dagli elevati requisiti. Spostare le risorse in un punto della rete più vicino alla produzione dei dati è la soluzione che questa trattazione propone per sopperire alle elevate latenze che impediscono l'esecuzione di applicazioni che richiedono tempistiche vicine al real-time e che generano un alto traffico di dati. Il modello Fog/Edge Computing a tre livelli inserisce delle risorse utilizzabili sull'edge della rete, come succede per le risorse rese disponibili dal Cloud nella core network. I vantaggi derivanti dall'adozione di un siffatto modello comprendono una bassa latenza, il supporto alla mobilità, e la possibilità di estrarre e usare informazioni contestuali agli utenti. Parte dell'infrastruttura presente sull'edge, dedicata allo svolgimento dei servizi, si appoggia al Cloud per le computazioni complesse e per il backup dello stato pesante del sistema. Il modello risultante è il tassello abilitante che permetterà alle infrastrutture esistenti di adempiere alle richieste delle moderne applicazioni, soprattutto relative all'IoT, che necessitano processare i dati vicino alla sorgente per minimizzare la latenza ed evitare frequenti connessioni con il Cloud, contenendo il traffico di rete generato. Le centraline di collegamento wireless potranno esse stesse ospitare le risorse necessarie alle continue micro operazioni prima demandate al

Cloud, di modo da poter fornire una bassa latenza a tutte quelle applicazioni che hanno bisogno di un responso real-time circa l'analisi della grossa mole di dati che collezionano e inviano in rete.

L'elaborato è organizzato in due livelli; il primo teorico, riguardante i modelli architetturali trattati e il secondo incentrato sulla presentazione di un prototipo utilizzato per la misurazione delle prestazioni delle risorse sull'edge. Nel Capitolo 2 si introduce l'architettura Cloud, le sue classiche componenti e gli scenari abilitati; per poi evidenziarne l'inadeguatezza della sua applicazione nei moderni scenari mobili. Il Capitolo 3 elenca i modelli proposti per rimediare ai punti deboli del Cloud e presenta alcune delle più importanti piattaforme attualmente presenti sul mercato, con particolare attenzione al modello a tre livelli caratterizzato dall'introduzione di un middleware intermedio tra il livello Cloud e i dispositivi IoT. Nel Capitolo 4, la trattazione prosegue presentando l'architettura generale e le scelte progettuali intraprese durante la progettazione del prototipo. Il Capitolo 5 approfondisce i dettagli implementativi della piattaforma proposta e ne mostra l'utilizzo. Nel Capitolo 6 viene descritto l'ambiente di test e riportati i risultati ottenuti dall'utilizzo della piattaforma. L'elaborato si conclude con il Capitolo 7 che riporta le conclusioni e alcune proposte di possibili estensioni future della piattaforma presentato.

CAPITOLO 2: CLOUD COMPUTING

Questo capitolo ripercorre le più importanti tappe dell'evoluzione dei moderni sistemi informatici, che hanno portato alla formazione dell'attuale architettura di riferimento per ambienti Cloud. Passando attraverso le idee di illustri precursori, si arriverà alla definizione attuale e standardizzata dei componenti principali dell'architettura Cloud. Verranno elencati i punti di forza che questa nuova tecnologia ha introdotto, e le inevitabili debolezze da cui è afflitta.

2.1 Cenni Storici

L'evoluzione delle tecnologie informatiche, prima dell'affermarsi del Cloud Computing, ha attraverso vari stadi architettonici, diventati nel tempo delle vere e proprie pietre miliari per quanto riguarda i modelli su cui costruire infrastrutture informatiche complesse. Cercando in letteratura una definizione ancestrale che meglio rappresentasse la maggior parte dei concetti che il Cloud Computing attualmente incorpora, si arriva alla famosa agenzia americana *ARPA* (*Advanced Research Projects Agency Network*) operante per conto del *Dipartimento della Difesa* (*DoD*). Ebbene un'idea di *rete globale* [1] fu introdotta proprio da *J.C.R. Licklider*, uno degli artefici di *ARPAnet*, prima rete ad aver implementato con successo la famiglia di protocolli *TCP/IP* e che sarà in seguito, negli anni

novanta [2], soppiantata da Internet come lo si conosce oggi. L'idea originale di Licklider inglobava parecchie caratteristiche che oggi Internet offre; ad esempio, l'autore parlava dello strumento stesso come futuro mezzo di comunicazione e commercio [3], e ipotizzava la necessità di interfacce utente che, facilitandone l'utilizzo, avrebbero ampliato il bacino di utenza. Praticamente la sua visione era quello che oggi il Cloud stesso si prefigge di raggiungere come obiettivo; ovvero abilitare la connessione e la comunicazione tra clienti, situati in qualsiasi parte del mondo, e programmi, dati e servizi presenti in siti a loro remoti. Licklider non era il solo ad avere una visione così avveniristica di quello che sarebbe diventata la rete globale. Negli anni sessanta, l'informatico statunitense *John McCarthy*, Premio Turing nel 1971 per i suoi contributi nel campo dell'intelligenza artificiale [4], teorizzava un utilizzo futuro di Internet molto aderente alla tendenza attuale. Egli concepì per primo un modello di computazione informatica distribuita; pensando alle capacità di calcolo come ad una vera e propria risorsa pubblica, da poter vendere ai clienti richiedenti esigendo un compenso determinato da statistiche di utilizzo.

Prima dell'implementazione reale di questo tipo di modelli passeranno effettivamente degli anni, ma un primo grosso cambiamento si identifica con il subentro del paradigma *Utility Computing* rispetto al precedente *Grid Computing*. In realtà i due modelli non sono diametralmente opposti, ma uno è l'evoluzione naturale dell'altro. Senza entrare in dettagli inutili, concettualmente il Grid Computing si concretizza nell'unione di diversi calcolatori che contribuiscono alla soluzione di un

problema comune. Un provider possedeva allora l'intera infrastruttura che realizzava il servizio proposto agli utenti. Naturalmente, questo approccio fa crescere notevolmente i costi, tanto quelli di ingresso nel mercato del provider stesso, in quanto occorre un forte investimento iniziale per poter competere con i fornitori attualmente presenti; quanto quelli di utilizzo. Possedere una infrastruttura preclude l'adattabilità del sistema in quanto occorre acquistarla tutta, prevedendo il massimo picco di utilizzo delle applicazioni e risorse che andranno su di essa installate. Tutto questo porta a dover effettuare una stima precisa dell'effettivo utilizzo, che l'esperienza insegna essere sempre molto complessa, in quanto esistono molteplici fattori che possono cambiare repentinamente, generando esigenze diverse da quelle previste, come ad esempio aumenti improvvisi del numero di utenti o dell'utilizzo di risorse. Inoltre, occorre farsi carico di tutte le operazioni necessarie per manutenere la struttura, e questo potrebbe voler dire, in certi casi, duplicarne delle parti per aumentare la sicurezza dell'intero sistema in caso di fault. Una evoluzione si è avuto con l'Utility Computing che ha perfezionato l'idea dietro alla disposizione di risorse informatiche cooperanti, trasportandola in ambiente pubblico grazie alla rete. Questo modello abilita un tipo di fatturazione *on-demand*, inserendo nel sistema dei misuratori di utilizzo che permettono ai clienti delle risorse di pagare quanto effettivamente stanno utilizzando, senza doversi sobbarcare costi inutili relativi alla infrastruttura e al sovradimensionamento.

L'unione di queste filosofie ha portato alla definizione del ventaglio di modelli che aderiscono alla più generica definizione

di Cloud Computing. Dopo questa doverosa introduzione storica, il discorso si concentrerà su una definizione più tecnica di quello che il Cloud Computing offre, analizzando cosa specifica la definizione standard e come vengano implementate effettivamente le varie risorse proposte.

2.2 Aspetti Architettonici

L'interesse principale degli attuali fornitori di servizi appartenenti alla sfera dell'*Information Technology* è sicuramente quello di assicurare agli utenti una possibilità di utilizzo costante e un adattamento rapido a variazioni contestuali, spesso riguardanti la richiesta variabile di risorse. La filosofia alla base del Cloud stabilisce che le risorse di cui l'utenza ha bisogno, che spaziano dalla semplice capacità di memorizzazione dei dati alla predisposizione di interi sistemi operativi, debbano trovarsi in rete; in una località non obbligatoriamente nota all'utente, in quanto indipendenti dalle applicazioni che andranno ad ospitare. Per esplicitare meglio questi concetti, ci si riconduce alla definizione standard rilasciata dall'istituto più autorevole in materia di modelli e architetture Cloud, ovvero il *National Institute of Standards and Technology* (*NIST*), un'agenzia del *Dipartimento del Commercio* degli *Stati Uniti d'America*. L'istituto si occupa di misurazioni e di standard architetturali in campo tecnologico e definisce il Cloud Computing un modello conveniente per l'abilitazione, l'accesso e la configurazione di risorse in rete facenti parte di un pool condiviso [5]. Per risorse

si intende la rete, intere macchine server, semplici locazioni di memori o applicazioni e servizi in generale. La definizione esalta la facilità di scalabilità del modello, affermando che la fornitura di risorse, effettuata al momento dell'effettivo bisogno, viene eseguita con minimi oneri gestionali da parte dei provider dei servizi. Usare una architettura di tipo Cloud, oltre alla elevata scalabilità, ci garantisce anche una serie di altre caratteristiche durante tutta l'operatività dei servizi che ospita:

- *On-demand self-service*: non occorre interazione umana per fornire risorse ai consumatori. Il processo è automatizzato e non c'è bisogno della comunicazione tra consumatore e fornitori dei vari servizi.
- *Broad network access*: le funzionalità sono fornite attraverso la rete, dando la possibilità a diversi tipi di clienti di usufruirne.
- *Resource pooling*: le risorse computazionali fornite ai consumatori, sono organizzate in un insieme condiviso fruibile da più utenti contemporaneamente. L'assegnazione delle risorse virtuali e fisiche ai vari utilizzatori è dinamica e automatizzata in base esigenze delle applicazioni. Le risorse, inoltre, sono indipendenti dalla locazione, in quanto l'utente, di solito, ne usufruisce naturalmente, senza sapere come esse sono organizzate o dove fisicamente risiedono.
- *Rapid elasticity*: la scalabilità è dinamica, rapida e eseguita automaticamente appena si ha necessità di nuove risorse o quando, alcune di esse precedentemente utilizzate, si liberano. Questo meccanismo permette al consumatore di

non essere conoscenza dei meccanismi che si celano dietro all'illusione di possedere una quantità infinita di risorse.

- *Measured service*: tipicamente nei sistemi cloud si usano basi di misurazione della fruizione dei servizi per effettuare una fatturazione ai diversi utenti. Di solito si parla di modelli, *pay-per-use* o *charge-per-use*, ad un livello appropriato di astrazione a seconda del tipo di servizio che si sta fatturando. Si potrebbe considerare come base per la misurazione l'utilizzo della memoria, della capacità di calcolo dei processori o, questi fattori in combinazione con l'utilizzo della rete e il numero di account attivi. Il controllo sull'utilizzo delle risorse è effettuato in modo trasparente, e noto sia ai fornitori che agli utilizzatori dei servizi.

Elencati i principali punti di forza introdotti dal Cloud Computing, si può affermare che questo modello si concentra sui metodi più economici possibili per fornire alta qualità e servizi veloci al minor costo per l'utente. Alle fondamenta delle scelte progettuali che hanno portato alla classica definizione del Cloud e dei suoi obiettivi, risiede l'architettura di base, mostrata nel prossimo paragrafo, che esplicita meglio la nozione di servizi e del loro utilizzo.

2.2.1 Architettura Orientata ai Servizi

Il paradigma teorico di riferimento per la tecnologia Cloud è senza dubbio quella identificata dall'**architettura orientata ai servizi** (**SOA**, *Service Oriented Architecture*). Con questo termine ci si riferisce al generico middleware che fornisce servizi di tipo business ai consumatori. Questo tipo di logica orientata ai servizi, si basa sul principio secondo il quale decomporre un'applicazione in servizi elementari rende questi ultimi riutilizzabili e di più semplice realizzazione [6]. Ogni servizio si occuperà quindi esclusivamente delle sue incombenze; l'organizzazione dei servizi nel processo produttivo, le loro modalità di interazione e composizione possono essere agevolmente modificate e ispezionate perché sono specificate sotto forma di contratti astratti. Queste interfacce di servizio specificano le caratteristiche del componente, come agire su di esso, le dipendenze di cui ha bisogno e come interagisce con gli altri componenti del sistema. I servizi diventano indipendenti dalla piattaforma e rappresentano astrazioni di elementi, come processi, risorse o intere applicazioni, che si possono standardizzare sotto forma di interfacce formali, il cui contratto di utilizzo, non deve essere in alcuno modo ambiguo [7]. Operando in questo modo si raggiunge un accoppiamento molto basso con la tecnologia a supporto dell'esecuzione; abilitando il programmatore, anche se non a conoscenza dell'ambiente di utilizzo, a poter comunque implementare servizi che espongono interfacce e ne utilizzano delle altre; le quali potrebbero anche descrivere realizzazioni in linguaggi di programmazione diversi. L'esposizione dinamica delle caratteristiche dei servizi,

nasconde quindi i dettagli della particolare logica realizzativa a chi li utilizza; l'utente ha bisogno di conoscere solo la semantica di invocazione, esposta pubblicamente dai componenti che offrono funzionalità e non i loro dettagli implementativa o la loro posizione. I servizi saranno realizzati in maniera il più possibile svincolata dal contesto e priva di stato, di modo da risultare come agenti autonomi e auto gestiti, indipendenti sia dall'ambiente di esecuzione che da invocazioni precedenti. Un'architettura di questo tipo deve prevedere un servizio di discovery che permetta il ritrovamento di componenti attivi, in base all'interfaccia specificata dai clienti che vogliono usufruire di un certo servizio.

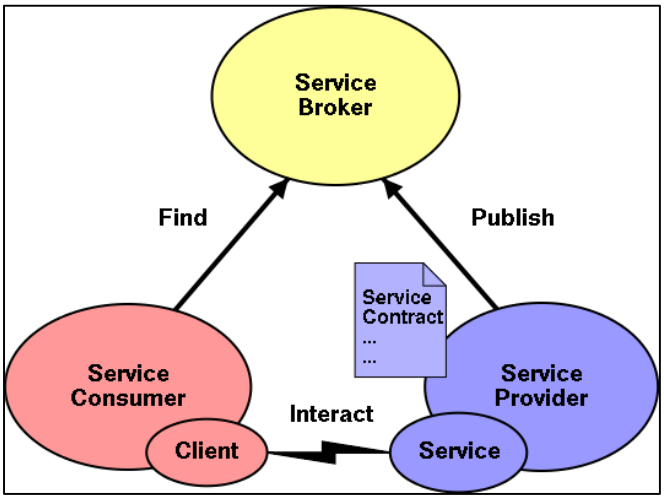

Figura 1 Componenti del modello Service Oriented Architecture.

Ricapitolando, le interazioni in un siffatto modello architetturale prevedono che gli utenti cerchino una specifica interfaccia di utilizzo pubblica, cedendola come parametro di ricerca ad un servizio di discovery. Presso il servizio di discovery si registrano i service provider che memorizzano le interfacce dei servizi che espongono. L'agenzia di discovery mette in comunicazione l'utente con il service provider che fornisce il

servizio richiesto, in modo da consentire un'interazione dirtela tramite interfaccia.

Il modello appena introdotto, per molti anni, ha costituito l'approccio principale da seguire durante la progettazione e lo sviluppo di applicazioni interagenti con sistemi distribuiti. Nei prossimi paragrafi verranno esaminati i dettagli delle architetture di base per Cloud Computing, vero protagonista della scena degli ultimi anni in campo di modelli architetturali per sistemi informatici distribuiti.

2.2.2 Architettura Cloud

L'architettura principale dei sistemi Cloud, deriva da una tassonomia generale, redatta dal NIST, in cui si elencano in maniera completa tutte le parti interagenti nel modello. Anche non essendo la sola [8], essa classifica in modo esauriente tutte le principali tipologie di Cloud, a seconda di varie caratteristiche prese in esame come parametro principale. Bisogna precisare che l'architettura di riferimento proposta dall'istituto identifica *cosa* i servizi Cloud debbano fornire, non *come*; ovvero le diverse implementazioni potrebbero differire nelle proprie intrinseche componenti anche rispettando l'architettura generale di riferimento. Questi documenti rappresentano degli strumenti per discutere, descrivere, categorizzare e sviluppare piattaforme

Cloud, avendo a disposizione come base un framework comune di riferimento.

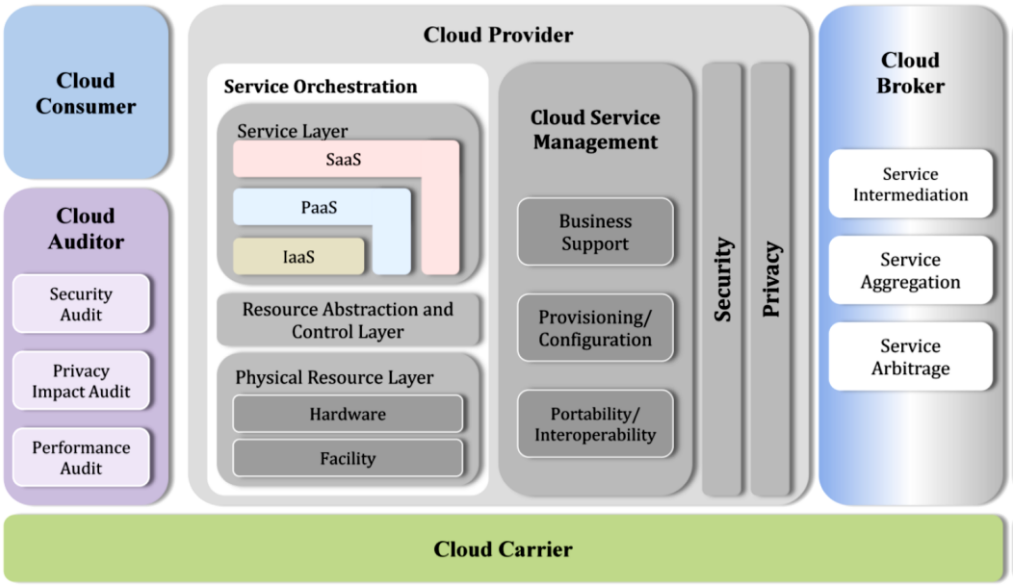

Figura 2 Architettura Cloud di riferimento.

Il modello di riferimento presenta una architettura di alto livello, il cui scopo è facilitare l'apprendimento dei requisiti e delle caratteristiche standard di sistemi di tipo Cloud Computing. Dalla figura si possono notare cinque ruoli fondamentali, interpretati da altrettante entità, che rappresentano persone o organizzazioni che partecipano a transazioni o svolgono attività all'interno del sistema. Intuitivamente un *Cloud Consumer* è l'entità che intrattiene un rapporto commerciale con *il Cloud Provider*, il quale rappresenta il fornitore dei servizi utilizzabili. Un *Cloud Broker* invece si occupa del rapporto tra i fornitori e i consumatori dei servizi, negoziandone l'accordo e gestendo le performance e il recapito dei servizi agli utenti. Ad esempio un consumatore potrebbe chiedere la disponibilità di un servizio ad un broker invece che direttamente al provider. Il broker potrebbe soddisfare la richiesta intersecando servizi provenienti da diversi

fornitori, creando un nuovo servizio combinato, oppure semplicemente migliorando le caratteristiche del servizio stesso fornito all'utente. Il ruolo del *Cloud Carrier* è invece quello di veicolare i servizi, in termini di connessione e trasporto, dai provider ai consumer. Un provider potrebbe in alcuni casi intrattenere due diversi contratti di servizio, uno con il cliente e l'altro con il corriere. In quest'ultimo contratto, il fornitore del servizio dovrà specificare al Cloud Carrier quali sono le specifiche quantitative e funzionali che vuole siano garantite per poter rispettare il contratto che detiene con l'utente. Infine le entità *Cloud Auditor* sono quelle che si occupano di valutazioni e accertamenti su performance e sicurezza, in forma slegata dagli altri agenti e in riferimento a particolari implementazioni.

2.2.3 Cloud Provider

L'entità abilitatrice e centrale dell'architettura è senza dubbio rappresentata dai fornitori di servizi. Il provider, persona fisica o organizzazione che sia, è responsabile della disponibilità del servizio; esso stesso acquista e provvede alla manutenzione del comparto hardware infrastrutturale che è richiesto per fornire i servizi. Anche per quanto riguarda la parte software le responsabilità sono le medesime, ovvero il provider esegue il software cloud che gestisce i servizi e si occupa delle corrette impostazioni che permettono un'adoperabilità dei servizi tramite la rete.

Il livello proprio dei Servizi è anche preso come modello di categorizzazione dei sistemi Cloud; esso sancisce quindi anche quali sono le responsabilità, per ogni tipologia, che i fornitori hanno. Per quanto concerne le piattaforme di tipo *Software as a Service* (*SaaS*) viene fornita al consumatore la possibilità di usufruire delle applicazioni del provider in esecuzione su un'infrastruttura Cloud. Questo significa che il fornitore stesso deve provvedere all'installazione, alla configurazione e all'aggiornamento del software che mette a disposizione.

Parlando delle *Platform as a Service* (*PaaS*), le responsabilità dei fornitori si identificano con la manutenzione dell'infrastruttura e l'esecuzione del software Cloud, configurato in modo che possa fornire le piattaforme di cui i clienti necessitano. Di solito si propongono ai consumatori anche degli strumenti che facilitano l'utilizzo delle piattaforme, quali ad esempio ambienti e kit di sviluppo (*IDE* e *SDK*) che aiutano nella gestione e nel deployment delle applicazioni utente. In questo tipo di contratto Cloud, gli utenti hanno il controllo sulle loro applicazioni, su come vengono messe in operatività e su alcune configurazioni dell'ambiente che le ospita, ma una visibilità molto ridotta, o addirittura assente, circa l'infrastruttura sottostante come la rete, le macchine server, i sistemi operativi e la memoria.

Si arriva infine all'ultima tipologia, ovvero quella delle *Infrastructure as a Service* (*IaaS*). In questo caso è sempre il provider ad acquisire l'infrastruttura generale del sistema, ma il suo compito nei confronti del consumer è quello di fornirgli

l'accesso diretto alle risorse computazionali. Questo è possibile grazie a interfacce di servizio e ad astrazioni logiche, quali macchine e reti virtuali, che gli danno il controllo diretto su risorse computazionali rilevanti come ad esempio il sistema operativo e la rete. Il fornitore mantiene sempre il controllo sulle risorse fisiche effettive e sul software Cloud che gestisce le richieste, ovvero il server, gli apparati di rete, i dispositivi di memoria, il sistema operativo ospitante e i gestori delle macchine virtuali (*hypervisor*).

Prima di addentrarci ulteriormente nella descrizione di tutte le parti componenti o utilizzate dai Cloud Provider, saranno presentate le restanti entità preseti nel modello generale delle architetture Cloud. Questo permetterà di poter affrontare un discorso più approfondito quando si tornerà a parlare di cosa i fornitori di servizi devono offrire e di quali strumenti si servono per abilitare queste concessioni.

2.2.4 Cloud Consumer

Un cliente di qualsiasi servizio Cloud può, in alternativa, contattare direttamente il fornitore o consultare un catalogo propostogli da un broker. Prima di poter usare il servizio, il cliente si accorda con il fornitore tramite un contratto chiamato *Service Level Agreement* (*SLA*). Questo documento sancisce le performance tecniche assicurate dal Cloud Provider per quel tipo di servizio, durante la sua normale operatività. All'interno del

contratto si trovano specifiche riguardanti qualità del servizio, aspetti di sicurezza e rimedi alla mancanza del rispetto dei requisiti da parte del provider. In questo modo il cliente può orientarsi tra le varie proposte riguardanti lo stesso servizio, ma aventi costi e contratti SLA diversi, così da poter decidere di acquistare non solo quello più conveniente, ma anche con termini contrattuali più favorevoli. Raramente i parametri di una fornitura, come contratto e costo, sono negoziabili; a parte rari casi in cui il consumatore sia una entità economicamente rilevante e preveda un elevato utilizzo della piattaforma.

A seconda delle diverse tipologie di Cloud acquistate si hanno a disposizione servizi diversi. Per esempio se un consumer decide di acquistare un servizio di tipo *SaaS* potrebbe trattarsi di una agenzia che demanda l'utilizzo di alcuni comparti software ai propri dipendenti, oppure semplicemente un privato che vuole usare un'applicazione on-line, o ancora un amministratore che usa il sistema SaaS per configurare le applicazioni per gli utenti finali. È possibile fatturare l'utilizzo di questo tipo di servizi basandosi sul numero di utenti finali, sull'utilizzo temporale, sulla banda consumata e/o sui dati memorizzati, sia in termini di durata che di dimensione. I Cloud Consumer di un *PaaS* di solito rappresenta un utente, o una organizzazione, che cerca di sviluppare, testare o configurare delle applicazioni pensate per ambienti Cloud. Come al solito la fatturazione può tener conto di tempo di utilizzo della piattaforma e delle risorse come ad esempio la rete, i database o la memoria. Per quanto riguarda le *IaaS*, i consumatori che hanno accesso diretto a risorse virtualizzate, di solito sono sviluppatori, amministratori di

sistema o dirigenti del settore dell'Information Technology che sono interessati alla creazione, alla gestione o al monitoraggio di servizi per infrastrutture. La possibilità di accedere alle risorse computazionali apre nuove forme di fatturazione che, oltre ai parametri validi per gli altri tipi di infrastrutture Cloud, possono includere anche altri termini di valutazione, come l'utilizzo della CPU, misurato in ore, e il numero di indirizzi IP.

2.2.5 Cloud Auditor, Broker e Carrier

Gli *Auditor* svolgono un ruolo molto rilevante nella scena del Cloud. Essi sono predisposti ad esaminare l'andamento dei servizi per esprimerne un giudizio in termini di sicurezza, impatto sulla privacy degli utenti e performance. I Cloud Auditor sono molto importanti per le agenzie governative perché essi deliberano anche su come si rapportano i vari servizi con il comparto giuridico del paese in cui vengono ad operare.

Una entità di mediazione è rappresentata dai *Cloud Broker*. Nel panorama sempre crescente di servizi Cloud, diventa difficile per un utente orientarsi, così esso può rivolgersi ad un broker, invece che contattare direttamente il fornitore, il quale media la conversazione tra le parti. Il broker di solito si occupa di intermediazione, aggregazione e arbitraggio. Per intermediazione e aggregazione si intende l'arricchimento delle caratteristiche del servizio; ovvero il Cloud Broker potrebbe migliorare o aggiungere delle funzionalità al servizio per aumentarne il

valore. Parlando di arbitraggio invece ci si riferisce alla possibilità di scegliere di volta in volta il servizio da aggregare a quello richiesto, con la possibilità ad esempio di impostare un tool di selezione che opti per un servizio compatibile, che presenta il miglior punteggio.

Il *Cloud Carrier* rappresenta un intermediario per quanto riguarda il trasporto e la consegna del servizio dal fornitore al cliente. Tra esso e il Cloud Provider viene sugellato un contratto SLA i cui termini devono essere rispettati per favorire un soddisfacente utilizzo del servizio ai clienti finali. Di solito ad interpretare questo ruolo sono delle organizzazioni che posseggono infrastrutture fisiche che occorrono al trasporto, come ad esempio apparati di rete performanti e dispositivi di memoria con elevate capacità. Il contratto che intercorre tra lui e il fornitore può prevedere anche che, durante l'utilizzo del servizio, vengano create delle connessioni dedicate tra il consumer ed il provider.

2.2.6 Deployment e Orchestration

Dopo aver discusso dei vari ruoli che assumono le diverse entità partecipanti alla scena Cloud e di quali siano i rispettivi obblighi e obiettivi, in questo paragrafo si avanzerà di un ulteriore livello nella tassonomia, analizzando il modello architetturale più da vicino e concentrando l'attenzione su quelle

che sono le funzionalità, laterali e principali, di un Cloud Provider.

Un importante trade-off presente nella definizione di Cloud data dal NIST, insieme all'altro riguardante la tipologia di servizio offerto, è quello che si basa su quanto sono esclusive le risorse per i clienti finali che le richiedono. Un'infrastruttura viene indicata come *Private Cloud* nel caso in cui essa sia resa disponibile ad un pubblico generico attraverso una rete pubblica. Di solito è posseduta, configurata e manutenuta dalla stessa entità che ne mette a disposizione i servizi sovrastanti, e che potrebbe essere impersonata da un'azienda, una comunità scientifica, un'agenzia governativa o una combinazione dei precedenti. Al contrario, una infrastruttura di tipo *Private Cloud* è così soprannominata per l'esclusività dell'utilizzo che fornisce ai membri di una singola organizzazione. Infrastrutture di questo tipo sono manutenute direttamente dall'organizzazione che fa capo ai clienti beneficiari dei servizi oppure da terze parti; esse inoltre possono essere ospitate negli stessi locali dell'organizzazione consumatrice (*on-site Private Cloud*) oppure esternalizzate verso terze parti che offrono questo tipo di servizio (*outsourced Private Cloud*). Un'altra tipologia dettata dal tipo di utilizzo, è quella rappresentata dal *Community Cloud*. Di solito, diversamente dal Private Cloud, l'infrastruttura viene messa a disposizione di diverse organizzazioni che hanno in comune degli aspetti del proprio lavoro come obiettivi di progetto, sicurezza, privacy o specifiche contrattuali. Similmente al Private Cloud, l'infrastruttura può essere ospitata negli edifici di una delle organizzazioni che ne fanno da consumer (*on-site Community*

Cloud) o esternalizzata (*outsourced Community Cloud*). Come per il possedimento, anche la manutenzione può essere effettuata dalla comunità direttamente oppure da terze parti. Una composizione, di due o più tipologie di Cloud sopra presentate, va a formare il cosiddetto *Hybrid Cloud*. Essendo una tipologia molto versatile, è molto utilizzata; essa lascia i differenti tipi di Cloud che ingloba concettualmente separati, ma ne prevede delle interazioni e provvede a fornire dei collegamenti che abilitano la portabilità di dati e applicazioni.

Parte fondamentale dei componenti appartenenti all'architettura Cloud è rappresentata dai membri della sezione soprannominata *Service Orchestration*. La parte superiore di questo livello è il *service layer* contente la dichiarazione delle interfacce di servizio messa a disposizione dai provider. Nella figura che mostra l'architettura di riferimento, i vari tipi di Cloud, rispettivamente SaaS, PaaS e IaaS, sono stati volutamente incolonnati perché tra di essi potrebbe esistere una relazione di dipendenza; nondimeno il bordo di ognuno di loro arriva fino al livello inferiore, ciò sta a significare che essi sono anche utilizzabili indipendentemente dall'esistenza degli altri livelli. In ambito produttivo non è raro interfacciarsi con sistemi complessi, uno dipendente, concettualmente al di sopra (*on-top-of*), di un altro già esistente che lo ospita e gli fornisce un livello di astrazione maggiore rispetto alle risorse reali.

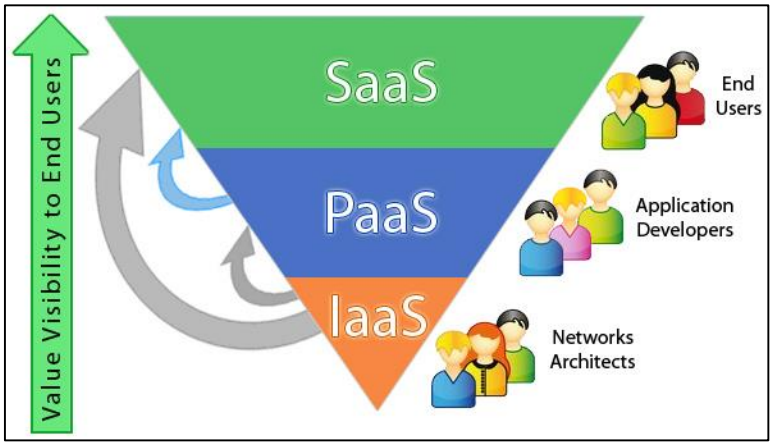

Figura 3 Stack dei sistemi Cloud.

Applicando questo discorso al Cloud ed osservando la figura precedente, si può notare come un sistema di tipo SaaS potrebbe essere stato costruito al di sopra di un PaaS o direttamente su un IaaS; mentre un PaaS potrebbe ospitare delle applicazioni di tipo SaaS e a sua volta essere eseguito su un IaaS. Praticamente, la particolare composizione dei vari livelli del service layer può dare origine a diverse tipologie di piattaforma. Per chiarire si propone il seguente caso di utilizzo: le applicazioni di un sistema SaaS potrebbero essere eseguite su una macchina virtuale fornita da una piattaforma IaaS, oppure formare un livello a se stante e autosufficiente e attingere direttamente dalle risorse Cloud, senza intermediari.

Il livello intermedio dei *servizi di orchestrazione* è occupato da due tipologie di componenti che forniscono accesso al livello sottostante. Si parla di *risorse astratte* indicando tutti quei servizi che assicurano l'efficienza, la sicurezza e l'affidabilità per quanto riguarda l'utilizzo del livello fisico sottostante. Ad esempio in questo insieme vanno incluse le

macchine virtuali, i volumi di dati virtuali e gli hypervisor. Dello stesso livello logico fanno parte anche le *risorse di controllo* che sono strumenti adibiti all'allocazione delle risorse, al dispatching, all'autenticazione sugli accessi e al monitoraggio.

Come consuetudine, l'ultimo livello è occupato dalle vere e proprie *risorse fisiche*. Principalmente queste includono tutte le risorse computazionali, i dispositivi di memoria, i comparti di rete e il resto della strumentazione utile al corretto funzionamento delle macchine.

2.2.7 Service Management

Questi servizi collaterali facilitano la gestione di tutta l'infrastruttura e si dividono in tre gruppi fondamentali. Iniziando dai *servizi business*, essi comprendono la gestione dei consumatori, di contratti e del catalogo dei servizi. La gestione dei committenti va a coprire il rapporto totale con la clientela dei servizi; ovvero bisogna considerare la creazione, la modifica, la distruzione dei profili utente e le loro interconnessioni. Senza dimenticare che di questo comparto logico fanno parte anche tutti quegli strumenti utili a risolvere i problemi tecnici in cui incorrono i clienti.

Oltre i servizi sopra elencati ci sono quelli di *configurazione e approvvigionamento* delle risorse. Questi

strumenti sono utili al deploy automatico del sistema, alla modifica run-time della configurazione e del numero di risorse, al monitoraggio e alla tariffazione.

Altro compito dei fornitori di piattaforme Cloud è quello di fornire meccanismi per una facile *portabilità e interoperabilità* dei servizi. Per portabilità si fa riferimento sia al semplice trasferimento di dati, con estrazione in un sistema e inserimento in un altro; che alla portabilità dell'intero sistema, come ad esempio avviene nel caso della migrazione completa di intere macchine virtuali. Quando si parla di interoperabilità invece ci si concentra sull'adattabilità di un servizio ad essere eseguito su diverse piattaforme, di modo che si possa relazionare con i servizi esistenti, costruendone di nuovi e più evoluti.

2.2.8 Sicurezza e Privacy

L'ultima parte che ci resta da descrivere del modello architetturale Cloud di riferimento è formata da servizi di tipo generico, *cross-cutting*, che migliorano la gestione dei componenti sia fisici che virtuali, fornendo funzionalità trasversali utili a tutti i livelli. La *sicurezza*, come spesso accade per sistemi complessi, rappresenta un aspetto fondamentale anche in questo tipo di architettura, e viene implementata in vari livelli, a seconda del requisito da assicurare che si prende in esame. Ad esempio prendendo in esame il service layer, occorre concentrarsi su componenti diversi a seconda del diverso modello di servizio

esaminato: in un sistema di tipo SaaS, normalmente acceduto via Web Browser, è quest'ultimo lo strumento su cui bisogna concentrare gli aspetti di sicurezza; diverso da quello preso in considerazione in caso di IaaS, che diventerebbe l'hypervisor presente sull'host, considerato che è il punto di accesso alle macchine virtuali. Prendendo in esame invece il modello di deploy, è evidente che alcuni aspetti sono fondamentali per dei sistemi, diversamente per altri. Si pensi all'isolamento del lavoro di una organizzazione da quello delle altre; requisito questo fondamentale in un Cloud pubblico, meno richiesto per quelli privati. Oppure, per quanto concerne la classica strategia di aumento della sicurezza ai bordi della rete, per un Cloud ospitato all'interno dell'organizzazione stessa è meno richiesto come requisito, diversamente da quanto si predispone per i Cloud ospitati su infrastrutture appartenenti a fornitori di terze parti. In ogni caso l'aspetto di sicurezza non è una prerogativa esclusiva dei fornitori di servizi, ma, siccome essi cooperano con i consumatori a diversi livelli, dovrebbero dividersi gli oneri di protezione dell'intero sistema basato sul Cloud. Fornire il controllo di un profilo amministratore, per quanto riguarda un sistema di tipo IaaS, fa parte dei compiti del fornitore della piattaforma, mentre assicurare i clienti dei servizi che vanno ad essere eseguiti su di essa di solito è demandato all'utilizzatore.

La *privacy*, altro servizio trasversale per tutti i livelli, deve tenere conto dei dati relativi ai clienti posseduti dall'infrastruttura. Di solito occorre memorizzare delle strutture dati che rappresentano le informazioni personali di identificazione dei clienti (*Personal Identifiable Information*,

PII), che comprendono dati come l'identità, il nome, dati biometrici e altri parametri simili. Occorre quindi che i fornitori di servizi prestino attenzione anche a come i dati sono condivisi e registrati, per garantire all'utente il livello massimo di affidabilità e riservatezza.

Come per ogni innovazione che si va affermando nel panorama tecnologico, anche il Cloud Computing presenta i propri limiti e introduce dei nuovi quesiti che, tanto i fornitori di servizi che i relativi consumatori, sono chiamati a risolvere. La sempre maggiore adozione di questo modello per l'impostazione architetturale relativa alla maggior parte dei servizi in rete, ha diversificato molto le alternative disponibili sul mercato, ampliandone quindi i relativi campi di utilizzo. Il Cloud abilitando la possibilità di poter condividere dati e infrastrutture tra diverse organizzazioni, impone innegabilmente dei requisiti molto stringenti. Si vedrà come, in alcune situazioni, si assiste ormai ad una inadeguatezza del paradigma, la quale esige un tempestivo aggiornamento per evitare un calo di interesse da parte di quelle organizzazioni che potrebbero essere degli ottimi partner per il suo stesso sviluppo.

2.3 Vantaggi del Cloud

Al di là di questa breve introduzione, i lati positivi riguardanti l'utilizzo del Cloud Computing sono molteplici.

Come si evince, si ottiene innanzitutto un abbattimento dei costi; sia dal punto di vista dell'utenza, non più costretta a possedere costosi comparti tecnici, che per quanto concerne la fornitura, che in questo modo può operare un più efficiente utilizzo delle risorse. Oltre alle entità partecipative al modello Cloud, a beneficiare di un utilizzo più responsabile delle risorse è anche l'ambiente. La condivisione crea una diminuzione della domanda relativa alle realizzazioni fisiche dei vari componenti, ed evita che questi rimangano inutilizzati; abbattendo in questo modo sia le emissioni relative alla produzione che i consumi inerenti l'utilizzo [9].

Per gli utenti finali, i vantaggi derivanti dall'utilizzo di servizi basati su infrastrutture Cloud sono molto rilevanti anche dal punto di vista dell'*esperienza finale di utilizzo*. Una infrastruttura Cloud infatti, avendo di solito delle capacità superiori rapportate a quelle dei normali computer e dispositivi dedicati all'utenza, permette delle performance esecutive migliori duranti lunghi compiti. Inoltre, per alcuni tipi di Cloud, l'acquisto, l'installazione, l'aggiornamento e la manutenzione del software non rappresentano affatto un problema per gli utenti; i quali, esenti da questi compiti, beneficiano direttamente del supporto fornitogli dagli stessi provider di servizio. Lo stesso vale per la quantità di memoria che gli viene fornita, ovvero la sensazione data all'utente è quella che essa sia disponibile in quantità illimitata, a patto di scegliere un canone adatto alle proprie esigenze.

Per le aziende il passaggio verso una tecnologia Cloud è dettato soprattutto dagli elevati *costi di gestione* derivanti dal possedimento e dalla manutenzione, sia hardware che software, di una infrastruttura interna proprietaria [10]. In ambienti enterprise è di importanza economicamente rilevante evitare che le risorse rimangano inutilizzate perché esse rappresentano una perdita monetaria in tutti gli intervalli in cui non vengono sfruttate al massimo del loro carico. Con il Cloud Computing, le imprese utilizzano solo le risorse di cui in quel momento hanno effettivamente bisogno, al cui approvvigionamento penseranno le entità adibite alla scalabilità automatica del sistema Cloud in esecuzione. Un risparmio monetario da parte dei fornitori di servizi, si traduce anche in un maggiore investimento delle aziende nel proprio campo produttivo; queste, evitando di dedicare risorse, sia umane che artificiali, alla continua gestione del sistema, sono maggiormente libere di dedicarsi al miglioramento dei propri prodotti.

Dal punto di vista tecnologico, i vantaggi derivanti dall'utilizzo di una piattaforma Cloud sono stati così rilevanti da aver rivoluzionato l'intero orizzonte dell'Information Technology; aprendo la strada a nuovi modelli operativi prima impensabili. Una *scalabilità* senza precedenti permette di far fronte agevolmente ai picchi di richieste e abilita l'ingresso in grossi mercati anche alle piccole aziende; le quali, tramite dei piani di abbonamento flessibili, possono affrontare la loro futura potenziale crescita in maniera flessibile e senza i costi eccessivi dovuti all'ammodernamento delle proprie infrastrutture. Il trasparente modello di abbonamento che permette di pagare solo

per le risorse effettivamente utilizzate consente, inoltre, alle aziende che lo sottoscrivono di redigere anticipatamente il proprio piano di consumo annuo.

La scalabilità generata dall'utilizzo di sistemi virtualizzati, permette all'utenza di percepire le *risorse come se fossero illimitate*. Spazio di archiviazione, memoria e capacità computazionali possono essere facilmente aggiunte, o requisite, alle macchine virtuali che eseguono i servizi, in base al loro attuale carico. La presenza di un pool di risorse a nostra disposizione permette agli sviluppatori di non preoccuparsi dell'esecuzione di task molto complessi, anche su dispositivi limitati. Se il tempo non è un requisito estremamente stringente si possono demandare queste operazioni all'infrastruttura Cloud di supporto che provvederà ad eseguirle senza interruzioni dovute alla scarsità di risorse.

2.3.1 Internet of Things

Ad accompagnare lo scenario del Cloud Computing verso una ampia adozione è stata l'unione del paradigma Cloud con quello relativo ai dispositivi mobili. Il moderno panorama dei dispositivi mobili si è massicciamente allargato inglobando non solo smartphone, tablet e portatili, ma anche parecchie altre categorie di accessori come ad esempio elettrodomestici e indossabili. Attualmente i device con capacità di compartecipazione a diverse tipologie di rete durante la loro

operatività, rappresentano una considerevole fetta di mercato [11]. In pratica una miriade di oggetti viene ad essere inserita in una sorta di modello interattivo, automatizzato da sensori e attuatori, che crea una rete autonoma di micro dispositivi sensienti, appellata con il neologismo di *Internet of Things* (*IoT*). Gli oggetti di uso comune, riguardanti l'elettronica di consumo e comprendenti anche elettrodomestici ed indossabili, vengono connessi alla rete per essere monitorati, controllati a distanza, o fungere loro stessi da controllori, come ad esempio avviene con occhiali e orologi intelligenti. L'avvento dell'Internet of Things ha aperto la strada ad una serie di prospettive di guadagno derivanti dai più variegati campi di utilizzo. Nel settore pubblico si assiste ad una graduale conversione verso questo tipo di rete; i servizi, presenti nelle moderne Smart City, vengono automatizzati e resi disponibili agli utenti, che ne beneficiano a seconda della località o del tempo in cui vengono ad interagire con essi. In ambito industriale la tendenza è quella di installare fitte reti di sensori e attuatori che prendano decisioni critiche in tempi molto limitati e che automatizzino alcune operazioni di gestione, auto coordinandosi con il minor intervento esterno possibile. Il campo della mobilità crea infinite opportunità di interazione tra gli utenti e l'ambiente a loro circostante. Si assiste all'abilitazione di innovativi servizi, migliorati dalle capacità computazionali delle infrastrutture Cloud, nonché dalla possibilità di immagazzinare un'enorme mole di dati. L'allusione al grosso volume di dati si riferisce sia al classico significato quantitativo, che a quello temporale. Le infrastrutture Cloud, infatti, garantiscono agli utenti IoT un adattamento automatico ai picchi di utilizzo in un ristretto spazio temporale, come spesso

avviene in presenza ad esempio di manifestazioni sportive, aggregazioni sociali in generale ed eventi catastrofici.

Il Cloud risulta essere la tecnologia abilitante fondamentale per l'IoT perché permette l'esecuzione di compiti altrimenti impossibili da portare a termine con l'utilizzo delle sole risorse presenti sui dispositivi mobili. La limitata autonomia, dovuta ad una capacità esigua delle batterie, crea un limite superiore per quanto riguarda i componenti hardware del dispositivo stesso. I circuiti integrati di cui sono dotati i dispositivi mobili non possono che essere di potenza computazionale ridotta per permettere un utilizzo nel tempo il più lungo possibile, senza doverli ricaricare. Dato che La stragrande maggioranza dei dispositivi IoT è dotata di risorse decisamente limitate rispetto alle classiche postazione da lavoro fisse su cui sono installate le piattaforme Cloud, quest'ultimo viene ad essere considerato un modello indispensabile per permettere ai clienti mobili di eseguire operazioni onerose dal punto di vista delle risorse. Spostare parte della computazione sul Cloud è una scelta applicativa fondamentale per mitigare gli effetti negativi della scarsità di risorse. Inoltre, la complessità sempre crescente dei dispositivi, insieme al loro elevato numero, crea una propensione a demandare la loro gestione a piattaforme Cloud.

Stando a parecchi studi statistici che analizzano il tema dell'oggettistica collegata alla rete, i dispositivi mobili connessi globalmente si aggireranno intorno ai 20 miliardi entro il 2020; con un mercato di circa 235 miliardi di dollari, in riferimento a tutti i tipi di servizi che l'IoT abilita [12]. Il mondo dell'IoT sta

gradualmente modificando il nostro modo di rapportarci al mondo esterno; immagini, suoni ed eventi significativi circostanti vengono raccolti e analizzati con l'obiettivo di migliorare lo stile di vita di tutti. Non solo in ambito industriale, ma all'interno dell'intera società si assiste ad un allargamento del bacino di utenza, determinato dal relativamente basso costo degli strumenti utilizzati e dalla facilità di gestione. L'unione di una tecnologia matura come il Cloud e di una innovazione così promettente come l'IoT preannuncerebbero un roseo futuro per la ricerca di nuovi metodi di ottimizzazione del lavoro e di facilitazione della vita quotidiana.

La realtà, però, va a scontrarsi con i naturali limiti tecnologici del Cloud Computing, dovuti in parte alla sua non predisposizione iniziale all'esecuzione di compiti per i quali l'IoT è stato creato, ovvero l'elaborazione continua e veloce di voluminosi dati, preferibilmente in real-time, senza interruzioni, e con tempi di risposta molto bassi. Questo gap tecnologico viene ad affliggere aree molto delicate del Cloud Computing, già di per loro oggetto di studi e miglioramenti. I picchi improvvisi del numero di connessioni, uniti alla non affidabilità della rete globale spesso conducono ad una interruzione della comunicazione che porta ad una sospensione di tutti i servizi eseguiti su piattaforme Cloud remote.

2.4 Limiti del Cloud

Le infrastrutture Cloud soffrono di problemi noti sin dalla loro nascita, come ad esempio quelli relativi alla *sicurezza e alla legislazione*. Per quanto concerne la sicurezza, si può affermare che la questione principale ruota intorno alla preoccupazione degli utenti relativa all'ubicazione dei propri dati relativi ai servizi e al trattamento di quelli personali. L'aspetto di sicurezza del Cloud Computing ingloba un numeroso insieme di tecnologie e modelli assegnati alla protezione di dati, delle applicazioni e dell'infrastruttura su cui sono eseguite. I moderni studi a riguardo mostrano come uno sforzo da tutte le parti è inevitabile per evitare falle nella sicurezza generale [13]. Sia i gestori delle infrastrutture che gli utenti devono attenersi strettamente a regole e protocolli concordati. Da parte dei fornitori ci sono già in atto degli sforzi per aumentare il livello di fiducia dei propri clienti, in particolare si può citare un autorevole istituto, il *Trusted Computing Group*, che produce strumenti che abilitano gli utenti del campo dell'Information Technology sia a scegliere l'offerta più consona fra la vasta gamma di quelle proposte dai fornitori di piattaforme Cloud, che, soprattutto, a valutarne la conformità delle norme di sicurezza. Studi forensi [14] dimostrano come buchi legislativi portino a comportamenti scorretti delle varie parti. I dipendenti che cambiano lavoro spesso continuano ad usufruire dei servizi Cloud, non correttamente disattivati, per scopi personali. Mentre i fornitori potrebbero violare la privacy dei consumatori trattando in modo improprio i dati personali e di utilizzo, che, al contrario, dovrebbero essere utilizzati solo per migliorare la qualità dei

servizi offerti. Occorre uno sforzo legislativo e di standardizzazione delle normative a livello internazionale, al di fuori purtroppo delle competenze tecniche.

Una problematica che genera apprensione durante l'orientamento dei clienti verso una particolare piattaforma tipo Cloud è anche quella di doversi confrontare con problemi di compatibilità tra diverse soluzioni. Una volta trasferito la parte principale del lavoro, un'azienda potrebbe venirsi a trovare legata in maniera indissolubile a quel tipo di tecnologia, privatizzata tramite la genesi di interfacce costruite appositamente per quel servizio. L'interoperabilità tra i diversi provider è una questione ancora aperta che necessita ovviamente di una soluzione tempestiva che standardizzi principalmente le interfacce di programmazione e che permetta una facile migrazione per i clienti.

2.4.1 Limiti del Cloud Applicato all'Internet Of Things

Dal punto di vista dell'Internet of Things, la vera limitazione del Cloud Computing deriva in realtà dalla tipologia di rete per la quale era stato pensato circa dieci anni fa. Si è precedentemente anticipato il ruolo fondamentale che il Cloud viene ad assumere in campo mobile, ma bisogna analizzare attentamente le problematiche tecniche derivanti da questa affermazione. Data l'inevitabile convergenza di Mobile e Cloud Computing, l'attenzione si sposta sulle condizioni reali di

utilizzo di questi modelli, che spesso si vedono inseriti in un ambiente operativo ostile. L'ulteriore carenza di risorse dei dispositivi IoT, rispetto agli attuali smartphone e tablet, è dovuta alla loro diffusione anche in domini molto specializzati dove spesso occorre una miniaturizzazione della componentistica che non permette elevate prestazioni. Per tutta la categoria degli oggetti connessi indossabili, per sensori, attuatori e piccoli gateway IoT, le esigue risorse disponibili obbligano a dover spostare una quantità di dati, e relative computazioni, sul Cloud; con una frequenza e quantità superiore rispetto a quanto avviene per i dispositivi di categoria intermedia come tablet, smartphone e portatili.

Con l'avvento dell'IoT sorge la necessità di dover operare su una network veloce, che fornisca topologie di connessione end-to-end e risposte in real-time. Si pensi alle continue disconnessioni e riconnessioni da parte dei dispositivi, ma anche alle notifiche relative ad un disastro o ad un imminente collasso del sistema. Occorre prendere decisioni in breve tempo e potersi basare su una connessione affidabile tra il cliente e il corrispondente servitore che ne esegue task complessi. In molte situazione, soprattutto dettate dal sovraccarico delle comunicazioni nelle reti WAN multi-hop, queste qualità non sono garantite dal Cloud; infatti esso si pone come obiettivo quello di gestire in maniera sicura dati e applicazioni degli utenti, in tempistiche più rilassate delle moderne esigenze di reattività real-time di cui l'IoT ha bisogno. Inoltre il Cloud è pensato per agire su reti estese di tipo WAN, che potrebbero contenere, lungo il percorso dall'utente all'effettiva locazione Cloud, dei colli di

bottiglia o addirittura delle interruzioni. L'assenza di connessione non è effettivamente il solo motivo di interruzione del servizio. Si pensi al caso in cui sia presente una risorsa ad accesso condiviso per tutti gli utente connessi alla stessa WAN. Come spesso avviene nelle reti che convogliano un gran numero di utenti, sia umani che artificiali, e come sempre più frequentemente avverrà nei moderni scenari IoT, in corrispondenza di picchi di utilizzo c'è una elevata probabilità che la risorsa in condivisione divenga effettivamente inservibile da parte degli utenti, che si vengono a trovare di fronte ad un sovraccarico delle comunicazioni di rete. Il Cloud non è stato concepito per la ricezione di dati con frequenze e velocità con i quali essi vengono prodotti dai numerosi dispositivi IoT; oltretutto far risiedere l'analisi di questi dati su piattaforme Cloud implica lo spostamento continuo di una grossa mole di dati. Il tutto porta alla congestione, che si traduce direttamente in negazione del servizio, rallentamenti o disconnessioni [15].

Oltre ai problemi di congestione, occorre sottolineare come il Cloud sia inadatto agli utenti mobili, in quanto non è raro che essi cambino il proprio indirizzo, durante l'utilizzo dello stesso servizio, allo spostarsi da una rete ad un'altra. Ad ulteriore tesi dell'inadeguatezza del Cloud in campo IoT, si prenda in esame il caso in cui un'azienda voglia interfacciare la propria dotazione di dispositivi proprietari con una qualsiasi piattaforma in rete; essa si scontrerà con l'inventabile incompatibilità dei propri protocolli interni con quelli comunemente usati per gestire i dati in Cloud. Occorre allora provvedere anche ad una conversione, quando si inviano, e ad una traduzione quando si ricevono dati;

operazioni che vanno ad aumentare i tempi di risposta che già rappresentano una criticità del sistema.

Ad aumentare l'impraticabilità dell'utilizzo esclusivo del Cloud per architetture IoT contribuiscono anche i modelli di fatturazione. Spesso l'utilizzo di una piattaforma Cloud è regolamentato dalla stipulazione di un contratto che prevede una fatturazione relativa al numero di connessioni o di dati in transito nel sistema. Appoggiarsi completamente ad una infrastruttura Cloud diventa economicamente insostenibile per le piccole aziende ed, in generale, per tutte quelle architetture che affidano la maggior parte della computazione a risorse esterne.

In sostanza, non è sempre possibile, o conveniente, spostare il carico computazionale in rete verso il Cloud perché non sono garantite le performance di utilizzo in alcune situazioni critiche di particolare interesse. Si pensi, ad esempio, ad agenti mobili che hanno bisogno di servizi residenti in reti WAN multi-hop al momento irraggiungibili, perché compromesse o distrutte; oppure si immaginino situazioni di emergenza, di aggregazione improvvisa o di eventi contestuali rilevanti, che porterebbero un singolo apparato di rete, impostato per lavorare con un certo carico, a trovarsi impossibilitato a gestire tutte le connessioni a causa del sovraffollamento. Questa, a tesi del presente elaborato, diventerà ben presto una limitazione inaccettabile, cui bisogna porre rimedio per evitare che l'imminente ed esponenziale aumento delle connessioni di rete generi l'interruzione dei fondamentali servizi di cui necessitano tanto le aziende quanto la comunità stessa in generale.

2.4.2 Applicazioni dai Requisiti Stringenti

I limiti prima descritti si concretizzano nell'impossibilità di utilizzo, su piattaforme Cloud, di applicazioni altamente accoppiate, che usano intensivamente le risorse. Per questo tipo di applicazioni le classiche infrastrutture Cloud, usate per espandere le capacità dei dispositivi mobili, risultano inadeguate e non garantiscono il mantenimento dei requisiti stringenti di latenza e banda disponibile richiesti. Spesso i device mobili operano in un ambiente ostile, caratterizzato dall'elevato numero di connessioni, dalla presenza di radio collegamenti, certamente più instabili delle reti via cavo, e dalla loro intrinseca scarsità di risorse quali alimentazione e computazione. Per il tipo di servizi che offrono, e che usano, le applicazioni residenti sui dispositivi necessitano di caratteristiche molto stringenti riguardo la latenza e la larghezza di banda; caratteristiche che rendono difficile un deploy efficace di un siffatto sistema su una classica piattaforma Cloud. Diversi contributi letterari provano che questa nuova classe di applicazioni sta guidando un cambiamento radicale dell'infrastruttura Cloud [16].

Il Cloud risulta limitante soprattutto per quanto riguarda gli scenari con reti operanti in ambienti ostili. La parte cliente dei servizi risente della lontananza logica del server, che come conseguenza ha quasi sempre un degrado non accettabile delle prestazioni, con casi limite che arrivano fino alla negazione del servizio stesso. Si pensi a servizi contestuali messi a disposizione

degli utenti in ambienti densamente frequentati, come aeroporti, strutture sportive e teatri, in cui ci sono frequenti picchi di connessioni. Risulta particolarmente difficile prevedere staticamente l'elevato carico e sarebbe insensato fornire l'infrastruttura di capacità che resterebbero inutilizzate durante la normale operatività in assenza di congestione. I collegamenti di rete verso il Cloud, in queste situazione di elevata sollecitazione, si intaserebbero portando, in casi limite, al collasso le comunicazioni con i servizi richiesti. Anche se la comunicazione non viene interrotta, la forte latenza provocata dalla congestione crea gravi problemi soprattutto alle infrastrutture informatiche che svolgono compiti in edifici pubblici; queste ultime, infatti, spesso sono chiamate a decidere della sicurezza fisica degli utenti. Non è raro che i dispositivi IoT, presenti in ambienti pubblici, siano destinati a prendere decisioni in tempi molto ristretti, ad esempio, per coordinare le operazioni di evacuazione della struttura stessa in caso di disastro.

Un'altra area a soffrire della latenza prodotta dalla comunicazione con il Cloud è l'intrattenimento mobile, uno dei mercati in maggiore crescita attualmente. Soprattutto per quanto riguarda il gaming, i moderni dispositivi mobili potrebbero in realtà eseguire qualunque gioco presente su altre piattaforme, se si riuscisse a mantenere le caratteristiche di comunicazione volute. Il grosso del carico computazionale verrebbe eseguito su un server Cloud, mentre il cliente sul mobile si occuperebbe solo di inviare dei comandi e ricevere l'output da mostrare sullo schermo. Attualmente sono in atto delle ricerche [17] [18] [19]

per rendere accettabile la latenza in questo complesso scenario, in cui l'esperienza finale è determinante al fine di un giudizio positivo degli utilizzatori sulla qualità del servizio, e pertanto non può essere degradata.

Tutto l'ammontare dei dati generati dai dispositivi IoT può superare molti TB al giorno, soprattutto in casi critici come il monitoraggio dei veicoli connessi e quello industriale. Anche in questi casi il requisito fondamentale di latenza bassa e di risposte in real-time vanifica i benefici del Cloud. I sistemi di trasporto attuali di solito usano tecnologie di connessione di tipo Dedicated Short Range Communicationst (DSRC) per comunicare sulle corte distanze e Long Term Evolution (LTE) per le lunghe. Dotare i veicoli di dispositivi DSRC o di risorse computazionali per elaborare autonomamente i dati e i messaggi, ha un costo, in termini monetari, troppo elevato per il settore pubblico; inoltre risulterebbe degradante per le performance nelle applicazioni connected vehicle private, come ad esempio nelle competizioni sportive ove il peso è un requisito fondamentale. Per la classe di veicoli in mobilità occorre distribuire messaggi in real-time e ad un basso costo per l'infrastruttura e per il committente, dato l'elevato numero di connessioni tra i veicoli e le centrali di analisi dei dati. Il contenuto dei messaggi può variare a seconda che si trattino di dati o di semplici avvisi. Il requisito che i messaggi siano notificati tempestivamente in real-time assume rilevante importanza ai fini della sicurezza dei viaggiatori che ad esempio potrebbero aver chiesto di essere notificati in base a incidenti e deviazioni vicine alla loro posizione.

Una latenza estremamente bassa è richiesta dalle applicazioni che operano analisi cognitive dell'ambiente, tipo quelle facenti parte del gruppo di software per la realtà aumentata. A questo insieme appartengono tutte le applicazioni che, tramite sensori, microfoni e videocamere, inviano un flusso di informazioni verso un server centrale con risorse illimitate; il quale, dopo aver elaborato le informazioni e averle arricchite di ulteriori dettagli, invia delle risposte al cliente. Il cliente riceve una reinterpretazione della realtà, che però deve essere tempestiva e di solito seguire il flusso di informazioni senza variazioni o rallentamenti. Piattaforme per la realtà aumentata sono già in commercio, non solo per applicazioni industriali, ma anche per quanto riguarda l'elettronica di consumo [20]. Uno degli sviluppatori di una importante azienda dietro alle ricerche sulla realtà aumentata [21], afferma che, per quanto concerne le applicazioni di tipo realtà aumentata e realtà virtuale, la latenza è la chiave abilitante indispensabile. Prendendo, ad esempio, l'elaborazione di un video, la latenza, da quando il cliente muove la visuale del video a quando lo stream gli viene proposto rielaborato, dovrebbe essere tra i 7 e i 15 millisecondi circa per non deteriorare troppo l'esperienza di utilizzo. Diminuire la latenza è il grosso obiettivo che si pongono le aziende che progettano sistemi sia hardware, che relativi software, che operano nel campo cognitive assistance [22]. Il Cloud non riesce a garantire, in qualsiasi situazione, il rispetto di questi requisiti; inoltre le elaborazioni non possono sempre essere fatte in locale, sui dispositivi clienti, che spesso sono di tipo thin, con modeste capacità computazionali e risorse di memoria. Dei test [23] dimostrano la necessità e l'efficacia di una eventuale estensione

del Cloud, a favore di questo tipo di applicazioni, che permetterebbe di abbassare notevolmente la latenza della rete.

Nel prossimo capitolo verrà esposta la tendenza attuale [24] da seguire per mitigare questo tipo di criticità. Per il nuovo modello, una importante azienda nel campo delle telecomunicazioni, cioè *Cisco*, ha coniato il termine di *Fog Computing*; rapportandolo all'esistente Cloud, ma con l'idea di avvicinarne le risorse, quindi la computazione, alla zona della rete più prossima alla sorgente di produzione dei dati, ovvero limitrofa ai dispositivi ad essa direttamente collegati.

CAPITOLO 3: EDGE COMPUTING

La presenza preponderante del concetto di mobilità all'interno dei moderni sistemi informatici, sia aziendali che pubblici che privati, ha permesso l'avverarsi di previsioni, le quali anticipavano questo scenario già qualche anno fa, in cui si parla di come l'ambiente venga ad essere dominato da dispositivi facenti parte della categoria relativa all'*Ubiquitous Computing*, o più in generale *Pervasive Computing*. L'*Edge Computing* è il tassello mancante utile al Cloud per coprire anche queste aree di utilizzo. Questa tecnologia innovativa non intende sostituirsi al Cloud, ma ampliarne la possibilità di utilizzo in certe aree delicate come l'Internet of Things [25]. Il suo scopo è infatti spostare parte delle risorse, che attualmente si trovano nella Core Network, più vicino ai dispositivi mobili che effettivamente le usano, di fatto diminuendo di molto la latenza della comunicazione e alleggerendo il carico di trasmissioni sulla rete centrale.

Il presente capitolo fornisce una definizione di cosa il Fog Computing rappresenta e di come viene implementato. Successivamente prende in esame un ramo particolare del Fog, soprannominato *Mobile Edge Computing*, modello di interesse fondamentale nel panorama tecnologico attuale e futuro. La trattazione proseguirà con un concetto fondamentale nel panorama delle tecnologie situate all'Edge della rete, ovvero il *Cloudlet*; strumento centrale di questo lavoro di tesi, utilizzato, come si vedrà, per comprovare la validità dell'Edge Computing

all'interno di ambienti ostili. Infine, verranno proposte, a scopo illustrativo, alcune implementazioni esistenti e abilitate al Fog Computing, per farne comprendere al lettore l'auspicabile massiccia adozione che il paradigma potrebbe avere in un futuro molto prossimo.

3.1 Fog Computing

Con la definizione di Fog Computing, si intende spostare il paradigma Cloud Computing verso l'edge delle reti [25], in particolare di quelle che collegano in modo wireless tutti i dispositivi asserenti alla definizione di Internet of Things. L'edge della rete è identificato con la subnet in cui i dispositivi risiedono; essa può avere una estensione ridotta o molto ampia a seconda che si tratti di apparati wireless interni o a banda larga. In questo luogo della rete, i servizi Cloud vengono ad operare direttamente, o al massimo tramite un singolo intermediario, con i clienti mobili che consumano o producono dati.

Stando alla definizione comunemente accettata di Fog Computing [26], esso rappresenta uno scenario in cui un elevato numero di dispositivi wireless comunicano e potenzialmente cooperano tra di loro e con i servizi in rete per supportare la memorizzazione dei dati e i processi computazionali senza l'intervento di terze parti. Cisco aggiunge alla definizione anche situazioni in cui è possibile che i dispositivi stessi diano supporto

all'intera infrastruttura di rete; affittando parte delle loro capacità in cambio di incentivi, di tipo monetario in caso si stia venendo fatturati, o viceversa tramite altre forme di premio come accessi esclusivi e privilegiati a certe applicazioni.

3.1.1 Caratteristiche del Fog Computing

Gestire una miriade di dispositivi eterogenei, sfruttando le tecnologie Cloud centralizzate, risulta molto complesso; tanto che spesso vengono a perdersi i vantaggi derivanti dall'adozione del modello IoT. Il Fog Computing cerca di venire incontro a requisiti quali la minimizzazione della latenza; fondamentale in ambienti ostili per la segnalazione tempestiva degli eventi rilevanti e potenzialmente dannosi che accadono nel sistema. Il goal principale del Fog non è solo la latenza, ma si cerca anche di essere incisivi dal punto di vista del ridimensionamento dei dati inutili e ridondanti inviati alla rete centrale. La computazione all'edge abbatte il traffico ridondante e inutile, conservando la banda a favore di processi più significativi, ed inviando solo periodici consuntivi (*clustering*) di quello che è lo stato del sistema in esame.

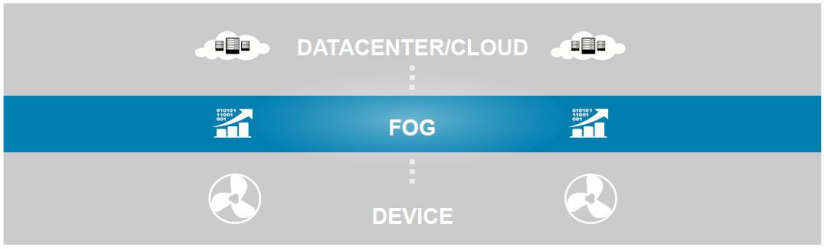

Figura 4 Livelli del Modello Fog Computing.

Il modello concettuale che descrive l'architettura Fog inserisce tutti gli apparati che estendono le funzionalità del Cloud, in un livello dedicato intermedio di connessione ai dispositivi finali di produzione dei dati. Gli agenti attivi propri del modello prendono il nome di *nodi Fog*; essi hanno la caratteristica di essere lontane dal data center Cloud con cui comunicano, ma molto prossime ai dispositivi utente; sono, inoltre, in numero elevato e sparsi sul territorio secondo una distribuzione geograficamente conveniente che copra il maggior numero di utenti. I nodi, anche essendo meno performanti delle macchine usate per ospitare le piattaforme Cloud, hanno un comparto tecnologico adatto all'esecuzione di compiti complessi e alla memorizzazione dei dati; inoltre sono scalabili e adattabili a diversi tipi di deployment. Le applicazioni che risiedono sui nodi Fog non rappresentano comparti software a se stanti, ma fanno parte di soluzioni più ampie che coprono anche i livelli Cloud e utente. I dati prodotti dall'utenza dei servizi Cloud, passano dal Fog che seleziona le computazioni che richiedono tempi stringenti ed esegue il lavoro localmente senza interpellare il livello Cloud. Gestire i dati nella loro interezza non è compito del livello Fog in quanto esso non possiede tutte le capacità del livello Cloud sovrastante. Il Cloud, in questo modo, verrà

sollecitato solo da richieste che prevedono vincoli temporali poco stringenti; come, ad esempio, l'elaborazione statistica dello storico del sistema o l'analisi e lo storage a lungo termine di Big Data. Lo stato, e i dati in generale, custodito dai nodi Fog è di dimensioni ridotte rispetto a quello che deve conservare il Cloud e mediamente è preservato per un tempo inferiore, quello utile all'esecuzione del servizio IoT real-time [27]. Questa separazione di responsabilità, oltre ad alleggerire il carico rappresentante la mole di dati inviati al Cloud, permette di aumentare la sicurezza delle informazioni sensibili che transitano per il livello Fog. Dal punto di vista software, i nodi Fog ereditano le caratteristiche degli apparati server del Cloud, ovvero sono ambienti altamente virtualizzati. I vantaggi derivanti dal modello software virtualizzato sono gli stessi dell'ambiente cloud; essi assicurano un isolamento logico delle componenti condivise, sia tra i vari processi in esecuzione che tra le aree dove vengono allocati i dati.

Il livello intermedio del modello, oltre a ridurre la latenza e ad aumentare la banda, fornisce ai dispositivi, e alle applicazioni che ne hanno bisogno, dati contestuali. I nodi Fog essendo apparati intelligenti posti sull'edge della rete, riescono a estrapolare informazioni di infrastruttura in modo più dettagliato di come succede per i data center Cloud; i servizi godono di informazioni aggiuntive circa lo stato del canale radio usato per la comunicazione e la locazione geografica della centralina, nonché dei dispositivi che ne usufruiscono. Tra i servizi che il Fog offre ai dispositivi IoT, c'è anche il supporto alla mobilità; quando un nodo si accorge che la connessione con

il device sta diventando troppo debole e la latenza ne risente, può spostare l'applicazione verso un altro nodo Fog, ora in prossimità del dispositivo che si sta spostando, di modo da non avere interruzioni di servizio rilevanti.

3.2 Mobile Edge Computing

Come già successo con il Cloud, l'unione del paradigma Fog con il campo strettamente mobile, crea un nuovo vantaggioso modello, il *Mobile Edge Computing* (*MEC*). Di Questa nuova tecnologia, che sta recentemente prendendo piede all'interno del panorama architetturale dei sistemi mobili distribuiti, ne è in corso un tentativo di standardizzazione, da parte dell'autorevole *European Telecommunications Standards Institute* (*ETSI*) [28]. L'idea alla base del modello è il trasporto delle possibilità offerte dal Cloud sull'edge delle reti mobili, rappresentato dalla *Radio Access Network* (*RAN*), che concettualmente è quella parte dell'infrastruttura di rete che connette i dispositivi finali alla rete Internet esterna. Praticamente il MEC può essere visto come un server Cloud, installato in prossimità dei dispositivi mobili, in una rete ad accesso radio, che effettua delle operazioni specifiche che non è possibile portare a termine con una tradizionale infrastruttura Cloud [29].

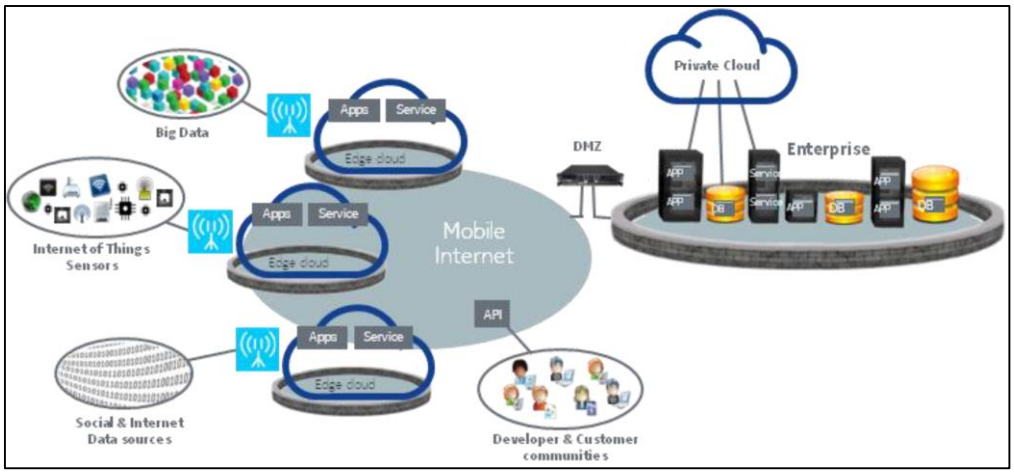

Figura 5 Panoramica sull'architettura Mobile Edge Computing.

Lo sviluppo tecnologico della componentistica installata sugli apparati di rete ha permesso un ampliamento considerevole delle possibilità di guadagno conseguenti all'abilitazione di una serie di servizi residenti sull'edge network. Una nuova catena di valore viene a creare diverse opportunità per gli operatori mobili, che possono aprire la loro infrastruttura a terze parti; come, ad esempio, agli sviluppatori che riusciranno a progettare le loro applicazioni consapevoli di potersi interfacciare con un ambiente caratterizzato da una latenza molto bassa e una banda passante elevata. I provider provvederanno, invece, all'evoluzione degli apparati RAN, i quali andranno ad arricchire le funzionalità dei servizi ospitati, fornendo ai dispositivi mobili supporto real-time e informazioni contestuali sulla rete. I server MEC potranno essere eseguiti tanto sulle attuali centraline *LTE* (*Long Term Evolution*) che si occupano delle connessioni mobili di quarta generazione, che sugli apparati delle reti 3G. All'interno, invece, i server MEC potranno essere installati sugli access point Wi-Fi che aggregano clienti presenti in siti specifici, come ospedali, aeroporti o uffici aziendali, sia pubblici che privati.

L’arricchimento delle funzioni di rete risulta trasparente all’utenza finale che ne beneficia assistendo ad un aumento della percezione positiva (*QoE, Quality of Experience*) sui servizi utilizzati. L’accesso ai contenuti e alle applicazioni è accelerato; in quanto esse risiedono in una parte della rete prossima all’utenza che massimizza la velocità dell’interazione. La contestualità di cui viene arricchita la comunicazione, sia geografica che per quanto riguarda le condizioni di rete, viene ad assumere un valore aggiuntivo per le applicazioni, che possono specializzarsi automaticamente a seconda del valore di questi parametri e ottimizzare la corrispondenza tra le parti.

3.2.1Architettura Mobile Edge Computing

Come si nota dall’immagine raffigurante l’architettura del modello MEC, molti elementi sono al di fuori dello scopo della trattazione e verranno rappresentati in forma astratta. Una piattaforma server MEC è composta da una infrastruttura ospitante affiancata ad una piattaforma applicativa, che contiene quei servizi utili al miglioramento delle caratteristiche del sistema e alla generazione delle informazioni contestuali.

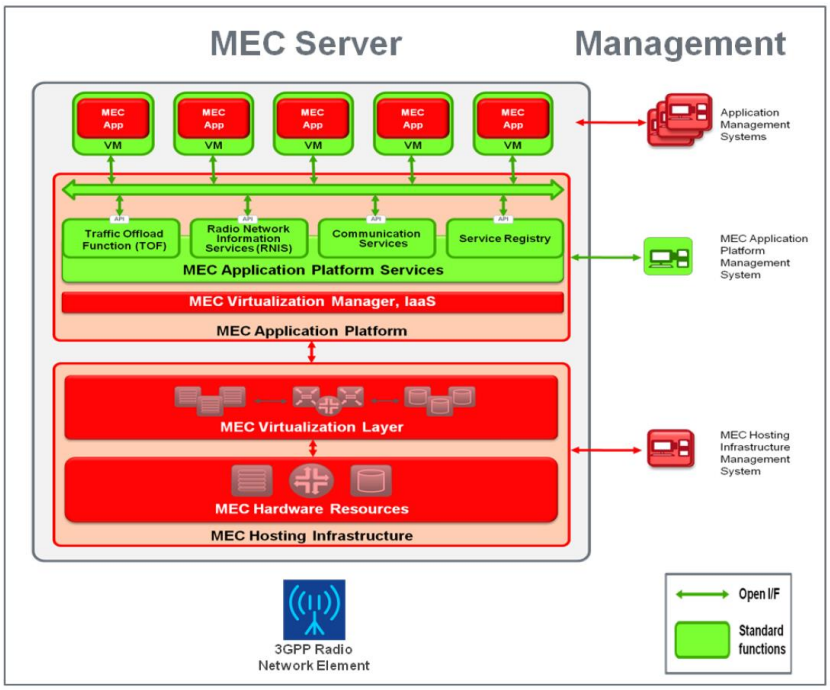

Figura 6 Architettura Mobile Edge Computing.

Per quanto riguarda l'infrastruttura *MEC Hosting*, i componenti hardware intrinsechi con cui essa viene realizzata, cosi come i software di virtualizzazione, non sono specificati perché il modello ne è indipendente. Dei servizi appartenenti alla *MEC Application Platform*, invece, si mette in evidenza il *Virtualization Manager* che crea una piattaforma virtuale IaaS, la quale, separando gli ambienti di utilizzo, provvede all'indipendenza di servizi e applicazioni sovrastanti. Come anticipato, questa infrastruttura è agnostica delle applicazioni che andrà ad eseguire; anche il top-level del modello, quindi, viene raffigurato in maniera astratta, dando la massima flessibilità alla tipologia di macchine virtuali che possono essere eseguite. Prescindono sia dalle applicazioni che dalle relative interfacce di comunicazione, anche i servizi *MEC Application Platform*, che incarnano il middleware software di supporto alle applicazioni sovrastanti.

Una parte dei servizi middleware, costituita da servizi di comunicazione e registry, è stata realizzata seguendo i principi relativi all'architettura SOA. Per quanto concerne i *Communication Services*, essi rappresentano il punto di collegamento tra le applicazioni e i servizi middleware sottostanti. Abilitano, tramite API ben definite, la comunicazione della macchine virtuali con l'ambiente MEC e tra di loro. Di solito questo componente viene implementato da un broker inserito in una comunicazione di tipo publisher/subscriber che permette la distribuzione dei messaggi uno a molti e un basso accoppiamento tra i partecipanti. Si possono naturalmente prevedere meccanismi di autenticazione che controllino e filtrino i messaggi pubblicati o prelevati. Il *service registry* fornisce anch'esso flessibilità al sistema dando visibilità dinamica dei servizi offerti dal server MEC. Le applicazioni possono eseguire una ricerca dei servizi e riceverne la locazione per usarli, o pubblicizzarne di propri esponendo gli end-point dove è possibile contattarli.

Ad affiancare questi tipici componenti SOA ci sono i servizi *Radio Network Information* che forniscono ai clienti informazioni dettagliate e real-time sullo stato del canale radio e della rete in generale. Le applicazioni possono farne un uso speculativo cercando di adattare il proprio comportamento alla condizione attuale della rete. Queste informazioni contengono delle indicazioni intrinseche come l'identificativo e la locazione geografica della cella a cui si è collegati, il carico di utenza attualmente presente sul nodo e di conseguenza la banda disponibile. Il contesto può essere arricchito in fase di deploy

fornendo al sistema informazioni aggiuntive da consegnare ai clienti che ne facciano richiesta. Questo tipo di informazioni sono reperibili perché fornite dai protocolli presenti nelle reti *3GPP*, ma il modello non specifica né quali né come esse debbano essere esposte agli utenti, lasciando libertà di scelta implementativa agli sviluppatori. Avendo a disposizione informazioni temporali e identificative degli utenti, a questo livello è demandata l'analisi statistica delle performance e le misurazioni relative all'utilizzo delle risorse.

L'ultimo servizio da descrivere è incarnato nella funzione *Traffic Offload*, che si occupa di assegnare priorità ai vari flussi e alle rotte memorizzate. Ingloba meccanismi di controllo per autenticare gli utenti ai diversi flussi. Attualmente il filtraggio viene operato sulla base di una tupla di tre elementi quali l'indirizzo IP del dispositivo mobile, l'indirizzo IP della rete e il tipo di protocollo IP; ma non è escluso un ampliamento delle politiche di filtering nelle versioni successive del protocollo.

Fanno parte del middleware di supporto anche le relative *interfacce di gestione*, che gli operatori di rete usano per orchestrare le varie componenti, il ciclo di vita e alcune opzioni di operabilità delle applicazioni e dei servizi che il server MEC ospiterà. Le operazioni cui ci si riferisce a questo punto della descrizione del modello prescindono dai servizi business veri e propri; esse sono delle regole che vengono fornite all'infrastruttura che in questo modo può operare una gestione ottimale delle macchine virtuali e della piattaforma in generale, a seconda delle esigenze del sistema. Di questo insieme di

interfacce, fa parte anche la descrizione degli standard usati per impacchettare le applicazioni, che contengono descrittori dell'applicazione, delle performance richieste da fornire alla macchina virtuali e dei meccanismi di controllo degli accessi. Adottare un sistema di descrittori per eseguire le macchine virtuali, permette di astrarre dalle varie tipologie di applicazione e servizio, permettendo un deployment delle risorse virtuali su piattaforme MEC multi-vendor.

In conclusione il MEC abilita servizi e applicazioni all'esecuzione su server, posti all'edge delle reti mobili, forniti da diversi provider. Il MEC trasforma le base-station delle reti mobili in piattaforme Cloud, arricchendone le caratteristiche con informazioni contestuali. Le piattaforme MEC trovano spazio applicativo sia in scenari outdoor che indoor. Già dalle prime stesure del protocollo si concepirono diverse modalità di deployment [30] che abilitano la presenza di apparati MEC tanto sulle base station esterne, come quelle utilizzate per le connessioni 3G e LTE, tanto sugli access point interni, che servono ambienti circoscritti come ospedali, stazioni e edifici aziendali. Per l'ambiente outdoor, la presenza di implementazioni MEC permette di fornire proprietà di sicurezza e virtualizzazione direttamente all'interno delle centraline radio di accesso. Anche il deployment indoor delle piattaforme MEC ha i suoi vantaggi; facilitando soprattutto le comunicazioni Machine-to-Machine (M2M), esso consente di distribuire contenuti contestualizzati e aumentare la velocità di risposta.

Il modello presentato di seguito risulta essere molto aderente ai principi introdotti per il Mobile Edge Computing, ovvero esso si propone come supporto alle applicazioni interattive che consumano molte risorse, ospitate su dispositivi mobili in un ambiente ostile. Come si vedrà il Cloudlet promette di essere una tecnologia che, tramite un modello a tre livelli classico del Fog Computing, fornisce risorse ai device mobili con una latenza molto bassa; facendosi carico di tutte le operazioni proprie del campo della mobilità.

3.3 Cloudlet

L'obiettivo principale del Cloudlet è quello di *avvicinare il Cloud* ai dispositivi che ne usufruiscono. Seguendo la basi poste dai modelli MEC e Fog Computing, il Cloudlet incarna il livello intermedio di un'architettura a 3 livelli. Esso si pone tra i dispositivi mobili e i servizi in Cloud centralizzati, all'edge della rete Internet, alla quale è connesso tramite un link veloce e affidabile. Le entità decentralizzate risiedono ad un solo hop wireless di distanza dai dispositivi finali associati. Il Cloudlet, come la definizione suggerisce, permette di costruire un *data-center in a box*, ovvero di ricreare l'ambiente di invocazione dei servizi Cloud tramite risorse presenti sulla rete locale, che rappresenta l'edge della core network che collega al Cloud remoto.

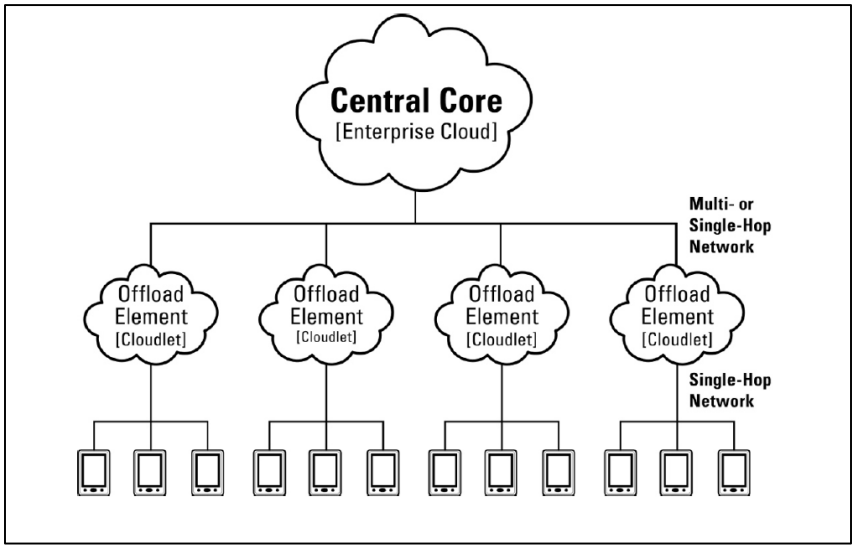

Figura 7 Architettura Cloudlet a tre livelli.

Le caratteristiche rappresentative del modello sono le seguenti:

- *soft state*: i servizi Cloudlet sono intesi come completamente auto gestiti, per questo si limiteranno a contenere sono un certo tipo di stato temporaneo, proveniente dai dispositivi o dal Cloud in caso di cache dei contenuti; lo stato significativo viene dirottato direttamente sui server Cloud di supporto. Data l'operabilità in ambienti ostili, la distruzione di una centralina non deve considerarsi un evento catastrofico per il sistema in generale.
- *powerful, well-connected and safe*: dovendo ricevere un notevole carico computazionale demandatogli dai dispositivi mobili pertinenti alla propria area, le centrali Cloudlet devono essere dotate di una tecnologia adatta all'esecuzione di tali computazioni resource-intensive, inclusa un'adeguata alimentazione. Tipicamente sono

connesse ad un canale Internet, tramite il quale accedono al Cloud, via cavo con tecnologie stabili e veloci.

- *close at hand*: è essenziale, ai fini del corretto dispiegamento dei servizi Cloudlet, la vicinanza dei server agli utilizzatori. La prossimità logica dei clienti alle macchine che contattano si traduce in una connessione a bassa latenza, con alta banda disponibile e con la possibilità di fare delle stime e previsioni precise delle sue performance.
- *builds on standard cloud technology*: non essendo una tecnologia sostitutiva, si adotteranno i classici meccanismi che vengono usati nelle piattaforme Cloud; ovvero la gestione dei task provenienti dai dispositivi mobili viene demandata alle macchine virtuali, che sono a loro volta amministrate da classiche infrastrutture Cloud come *Amazon EC2* oppure *Openstack*. La differenza con le piattaforme Cloud è che queste hanno funzionalità aggiuntive derivanti dalla loro natura Cloudlet.

Il Cloudlet incarna tutte quelle caratteristiche che sono diventate essenziali per l'esecuzione di task che usano intensamente le risorse disponibili in una comunicazione fortemente accoppiata. L'utilizzo delle macchine virtuali permette ai servizi di essere transienti; ovvero venire creati sovrapponendo specifici metadati ad immagini base dei vari sistemi operativi, per poi essere facilmente dismessi facendo tornare il sistema, inteso anche come risorse fisiche, alla sua versione vanilla. La transitorietà dei servizi è di fondamentale supporto all'eterogeneità e alla mobilità dei clienti che una

centralina Cloudlet può servire. La segmentazione del software permette una netta separazione di ciò che è l'ambiente attuale di esecuzione da quello che la stazione deve conservare come contenuto permanente.

Il Cloudlet rielabora due dei concetti fondamentali presenti in tutte le tecnologie mobili che utilizzano la virtualizzazione come meccanismo di astrazione delle risorse. In generale esistono due approcci tramite i quali una macchina virtuale viene messa in esecuzione. Nella *migrazione* di una macchina virtuale si ha il completo trasferimento dello stato della memoria, dei registri della CPU, e dello stato temporaneo proprio della macchina in esecuzione. L'altro approccio è rappresentato dalla *sintesi*, che rappresenta l'allocazione, secondo diverse strategie, delle risorse necessarie ad un consumatore. I prossimi paragrafi spiegheranno qual è il criterio che il Cloudlet usa per realizzare questi importanti meccanismi di sintesi e di handoff che permettono, rispettivamente, la fruizione e il trasferimento dei servizi.

3.3.1 Sintesi

Le piattaforme di cyber-foraging, che forniscono risorse aggiuntive ai clienti mobili dalle capacità ristrette, usano diverse strategie, più o meno speculative, per restringere al minimo il tempo di sintesi di una macchina virtuale utile all'utente del servizio. In sistemi di questo tipo le applicazioni sono partizionate tra la parte cliente in esecuzione sui dispositivi

mobili e il server messo in esecuzione dalla struttura remota. Di solito è il consumatore a contattare il provider dei servizi per richiedere direttamente la risorsa, se disponibile, o il suo reperimento qualora non lo fosse. Come strategia di esempio molto dinamica, verrà presentata quella utilizzata dal modello Cloudlet, che ingloba molte delle caratteristiche del meccanismo generale di provisioning di una macchina virtuale.

Un host che riceve richieste dai clienti ospita al proprio interno una piattaforma di gestione delle macchine virtuali, un front-end per la comunicazione con gli utenti e delle immagini base di sistemi operativi maggiormente utilizzati. Per esempio, come elemento statico presente negli host Cloudlet sin dal set-up, oltre alle immagini basi e al manager delle macchine virtuali, si trova anche un servizio di discovery; quest'ultimo permette il reperimento dell'indirizzo della centralina stessa da parte di utenti che richiedono caratteristiche specifiche del Cloudlet sull'edge della rete. Il servizio di discovery è responsabile dell'advertisement in multicast della presenza del Cloudlet Host nella rete locale. I servizi di discovery sono strumenti leggeri ed appropriati per ambienti locali perché evitano la comunicazione tra i clienti mobili, che vogliono usare servizi presenti nella propria sottorete, e la rete esterna. Sui dispositivi mobili è in esecuzione un Cloudlet Client sensibile alla ricezione di messaggi in multicast che lo notificano della presenza del server Cloudlet sulla rete locale a cui è collegato wireless. Nelle classiche piattaforme di supporto, il cliente mobile, venuto a conoscenza dell'indirizzo di una centralina per il cyber foraging, può avviare la comunicazione dei dati relativi al servizio da

eseguire verso il server; il quale provvederà al provisioning di una macchina virtuale segnalando la richiesta alla piattaforma che le gestisce sul nodo host. Quando la risorsa sarà pronta, il cliente ne verrà informato e potrà demandare le proprie computazioni pesanti alla macchina virtuale appena fornita. L'architettura Cloudlet, ad esempio, come scambio di informazioni relative al servizio da eseguire, prevede la consegna, dal cliente mobile al Cloudlet Server, di un file overlay. Il servizio è avviabile solo se l'host possiede l'immagine base del sistema dal quale l'overlay è stato generato; viene quindi eseguito questo controllo che, se dà esito positivo, abilita il gestore di macchine virtuali ad avviare una VM relativa al servizio richiesto. La figura seguente mostra le componenti fondamentali che intervengono nella comunicazione tra il server Cloudlet, inserito nella edge network, e i clienti che chiedono di eseguire dei task con le risorse della centralina

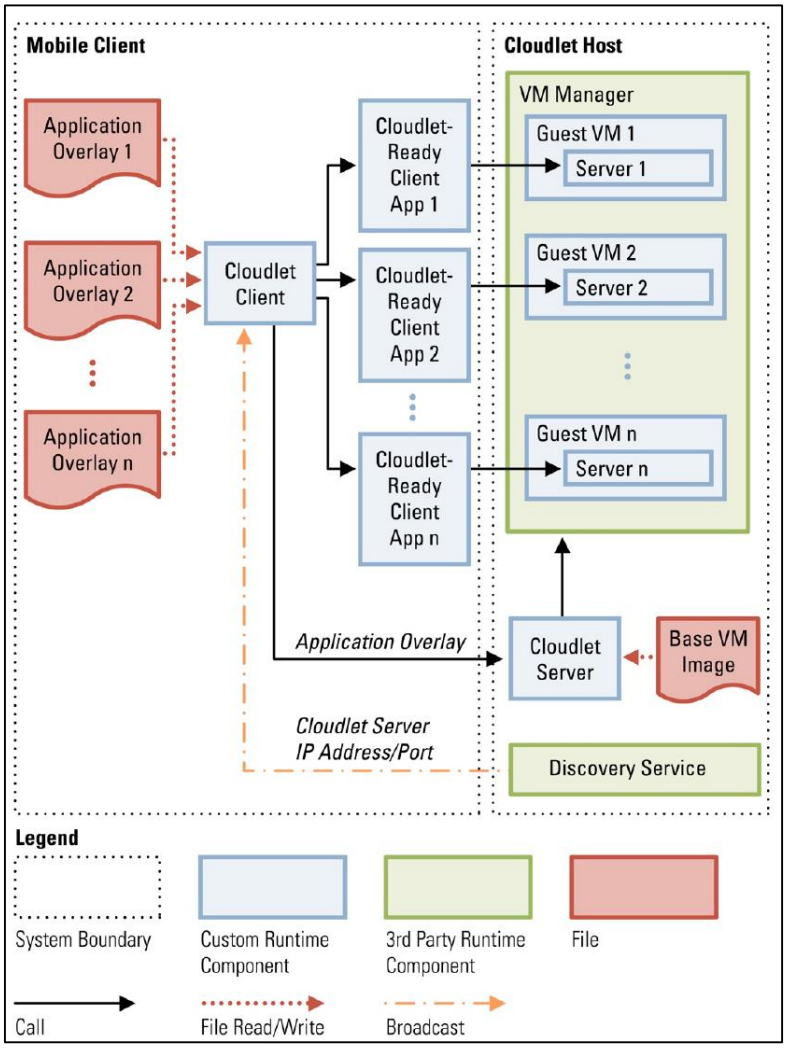

Figura 8 Architettura cloudlet per il cyber-foraging.

Il cliente mobile, situato ad un hop wireless di distanza dal server, memorizza un certo numero di overlay che contengono il codice e i dati da mandare al server per l'esecuzione del relativo servizio. Se la connessione al Cloud è disponibile, per ottimizzare lo spazio disponibile e il consumo di energia del dispositivo mobile durante il processo di sintesi, è possibile memorizzare solo un URL relativo all'overlay del servizio di interesse; di modo da consegnare questo link al Cloudlet che in autonomia provvede a scaricare l'overlay. La modalità, introdotta

dal Cloudlet, di consegna locale della componente utile all'esecuzione di un servizio, rende la velocità di questo processo dipendente solo dalla tecnologie wireless installate sulle centrali e non più dalle caratteristiche attuale della rete WAN. Una modifica al comparto hardware di un server di questo tipo coincide direttamente con una maggiore velocità nella sintesi di nuove macchine virtuali, indipendentemente dall'avanzamento tecnologico che subisce parallelamente la rete Internet.

Per quanto riguarda la generazione delle informazioni utili al server per avviare gli specifici servizi richiesti dai clienti, per il modello Cloudlet, il processo di creazione di un overlay è eseguito offline, in modo da non pesare sulla comunicazione run-time.

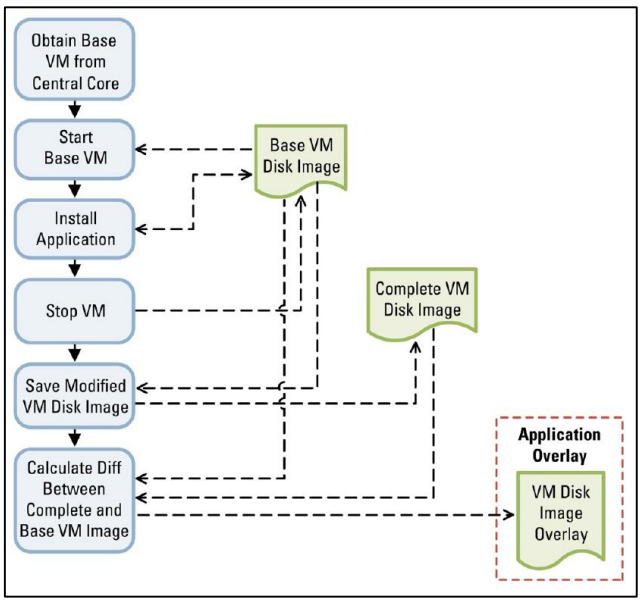

Figura 9 Processo di creazione di un Overlay.

In prima analisi, l'host si procura una immagine di una Base VM contenente il sistema operativo di base sul quale andare ad installare successivamente il servizio. L'immagine base può

essere contenuta staticamente all'interno del server, oppure essere scaricata dalla rete esterna. Non è improponibile seguire la strada statica durante questo passo; dato che la stragrande maggioranza di servizi sono eseguiti solo su un numero ristretto di sistemi operativi, è possibile fornirli tutti anticipatamente al server per evitare connessioni inutili se non in casi estremamente particolari che non involvono uno dei convenzionali sistemi maggiorente usati. Una macchina virtuale viene quindi creata a partire da un'immagine base. L'applicativo server, e le relative librerie, vengono installate sulla macchina virtuale appena creata. Alla fine delle operazioni di set-up del servizio, la macchina viene sospesa. Quando la macchina viene sospesa vengono creati due file che faranno parte dell'overlay: il primo corrisponde alla differenza del disco tra la macchina virtuale appena sospesa è quella originale di base; il secondo contiene invece la differenza binaria tra la memoria della VM sospesa e la memoria dell'immagine base. I delta, relativi a disco e memoria, vengono compressi insieme all'interno di un unico file rappresentante l'overlay.

La sintesi può seguire diverse strategie [31] per ridurre il tempo di avvio dell'applicazione, che rappresenta l'intervallo che separa l'arrivo di una richiesta di provisioning al server e la relativa risposta di VM Ready, che segnala l'effettiva ripresa della macchina virtuale. La metodologia più intuitiva è quella rappresentata dalla *sintesi tramite comunicazione dell'overlay*, che rappresentate la parte server di una applicazione di servizio costruita su un modello cliente/servitore. Esso, una volta consegnato al server, viene applicato ad una macchina virtuale

avviata dall'immagine base da cui l'overlay è stato generato. Il controllo, come anticipato, è fatto sulla compatibilità tra l'overlay e le immagini base che il server conosce. Oltre alla classica sintesi tramite ricezione e decompressione di un overlay, è possibile adottare un metodo di tipo *Cached VM*. I server vengono approvvigionati anticipatamente di macchine virtuali corrispondenti ai servizi che richiederanno i clienti mobili. A run time, il dispositivo mobile verifica che il server abbia una macchina virtuale di servizio operativa e corrispondente all'applicazione che intende eseguire. Se esiste una macchina di questo tipo, il server in esecuzione sull'host cloudlet crea una copia di tale macchina e restituisce indirizzo e porta di accesso al cliente. È possibile alleggerire il carico di dati presenti sul dispositivo mobile tramite la tecnica *Push*: il server, in questo caso, non solo contiene già le macchine virtuali relative ai servizi, ma anche le rispettive parti cliente. Il mobile, entrando in contatto con una centralina di cyber-foraging, la interroga sui servizi che essa fornisce e di conseguenza ne scarica la parte da eseguire sul dispositivo, similmente a quanto accade quando si accede ai classici app-store. Il controllo che viene effettuato in questo algoritmo è relativo alle possibilità, del sistema operativo in esecuzione sul device, di eseguire la parte cliente che gli viene consegnata dal server. In caso positivo, il cliente riceve l'applicativo e lo installa; contemporaneamente il server host avvia una macchina virtuale corrispondente e, quando pronta, rimanda indirizzo e porta al dispositivo. Il metodo più flessibile di tutti, ma che ha anche qualche svantaggio è denominato *On-Demand VM Provisioning* [32]. Il cliente in questo caso consegna al server non un overlay, ma un file di configurazione, ovvero un

file generato da un tool automatico che è usato per assemblare un servizio partendo da una immagine base. Il server crea come al solito una copia dell'immagine base per poi applicare lo script di configurazione che scarica, installa e avvia le componenti software. Lo svantaggio di questa variante è rappresentato non solo dalla compatibilità richiesta con lo strumento che genera lo script di configurazione, ma anche dalla obbligatoria presenza, in una repository locale o su un sito di distribuzione del codice affidale, di tutti gli elementi utili al set-up del servizio.

3.3.2 Handoff

La mobilità dei clienti determina la necessità di una procedura per spostare tutto il carico computazionale da una centralina ad un'altra più vicina alla nuova posizione dell'utente. L'*handoff* è il meccanismo, trasparente alle applicazioni, che avvicina il servizio generico in corso, ad una centralina più conveniente di quella attuale, dalla quale l'utente si sta allontanando, per evitare un eccessivo degrado delle prestazioni dell'applicazione in esecuzione. L'handoff di una macchina virtuale operativa sull'edge della rete, anche essendo simile a quanto succede con la *live migration* per le piattaforme Cloud, differisce da questo processo in alcuni punti [33]. Mentre la migrazione a caldo cerca di minimizzare l'ultimo passo del procedimento, ovvero quando la macchina virtuale è inutilizzabile, l'handoff ha come obiettivo di base la riduzione di tutto l'intervallo utile al completamento dell'operazione. La

migrazione a caldo propria del Cloud, per minimizzare il tempo di inoperatività della macchina virtuale, continua l'esecuzione sulla sorgente anche durante il trasferimento. Se in questo frangente viene modificato lo stato della memoria, il processo si ripete, con iterazioni sempre di minor durata; fin quando, durante l'ultimo passo, si ha la sospensione della macchina e il trasferimento dell'ultima piccola parte di memoria modificata, per poi eseguire la riattivazione sul data center destinazione. Bisogna notare come la migrazione sia limitante rispetto all'handoff perché non esegue il trasferimento dello stato del disco, ma solo della memoria. In ambiente Cloud si può considerare accettabile questa situazione perché si suppone che questo stato sia accessibile ad una locazione condivisa tra i server. Per trasferire una macchina a caldo non occorre una connessione ad alta banda come per il Cloud, ma occorre la presenza, nell'host di destinazione, di una immagine base compatibile con la macchina virtuale che si sta trasferendo.

Il processo dell'handoff, utilizzato dalla nuova architettura Cloudlet, prende spunto dalla sintesi, ovvero coinvolgendo l'overlay, ma questa volta a run time. Invece che considerare la creazione di un overlay solo un passo esterno all'esecuzione, esso viene eseguito molteplici volte, sul Cloudlet sorgente, durante l'handoff. Se per la sintesi il tempo necessario per la generazione di un overlay era trascurabile, perché offline, nell'handoff diventa un fattore di fondamentale importanza, volendo prendere in considerazione il tempo totale del trasferimento. Ridurre questo intervallo è di fondamentale importanza per garantire una esperienza soddisfacente nell'utilizzo di servizi in mobilità. Sono

state suggerite, a tal proposito, differenti soluzioni che accelerano la creazioni di un overlay in base alle risorse hardware disponibili [33]. Durante le continue creazioni di overlay che avvengono in corrispondenza di un handoff, la macchina virtuale continua ad essere operativa e potrebbe ricevere delle richieste dai clienti che ne modificano lo stato. Le modifiche attuali verranno inviate nella iterazione successiva; fino a quando la durata dell'iterazione diventa sufficientemente corta da poterla considerare priva di comunicazioni. A questo punto il sistema sospende l'esecuzione della macchina virtuale e completa l'operazione di handoff.

Al fine di ottimizzare il tempo totale occorrente per la completa esecuzione di un handoff, le risorse principali, quali la capacità computazionale e la banda di rete disponibile, vengono monitorate continuamente per adattare il processo all'attuale condizione operativa. L'euristica che selezione la migliore modalità operativa che massimizza il *throughput* totale del sistema è dinamica e reagisce ai cambiamenti relativi alla banda e alle risorse di elaborazione disponibili.

3.4 Elijah

Una delle implementazione open source più complete riguardanti l'architettura Cloudlet a tre livelli è senza dubbio la *piattaforma Elijah*. Essendo stata concepita come un'estensione

di una diffusa piattaforma IaaS Cloud, essa adotta lo stesso tipo di licenza *Apache Version 2.0*. Elijah è un progetto collaborativo il cui maggiore contributo è dovuto al gruppo di ricerca della *Carnegie Mellon University* (*CMU*), ma che sta ricevendo parecchio interesse da parte di tutte le aziende leader nel campo del Cloud open source [34]. Le estensioni create per il Cloudlet dal team di Elijah, applicate alla piattaforma Openstack, la trasformano in *Openstack++*; strumento capace di eseguire tutti i passi che contraddistinguono il modello Cloudlet illustrato precedentemente [35]. Uno deli obiettivi principali del progetto è quello di standardizzare le API Cloudlet, come precedentemente è successo con Openstack per quanto concerne invece le interfacce Cloud. In questo modo, i provider di software e hardware con capacità Cloudlet potranno contribuire alla fornitura di servizi su diversi livelli mantenendo una compatibilità globale.

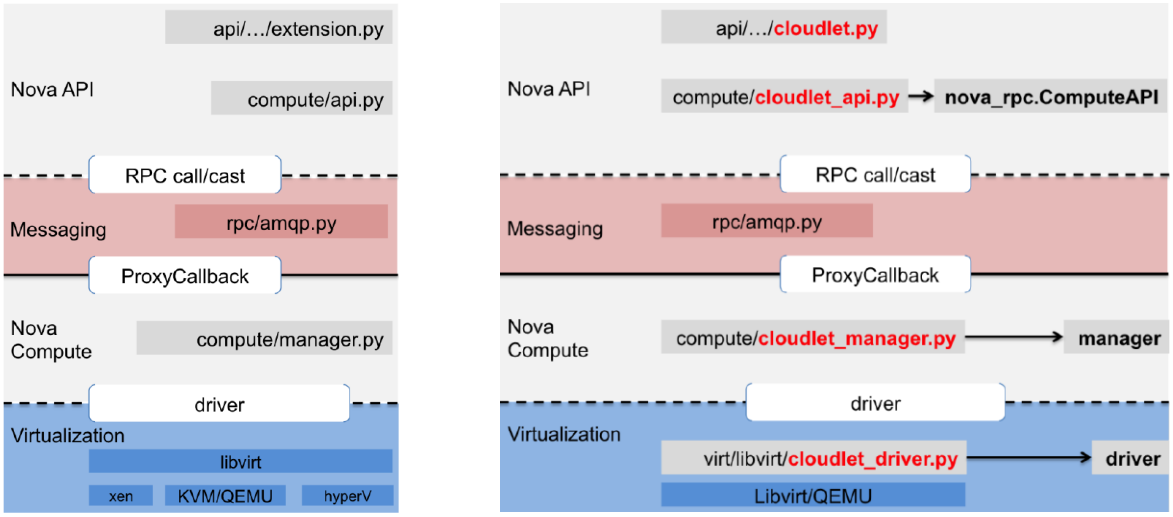

Figura 10 Confronto tra la catena di chiamate per una richiesta di API in Openstack e in Openstack++.

La lungimiranza insita nello strumento Openstack ha lasciato spazio, tramite *meccanismi di estensione*, ad approcci

diversi da quelli previsti inizialmente; così da dare la possibilità di sperimentare nuove opzioni, senza preoccuparsi di modificare le API standard. Le estensioni, essendo interrogabili, possono essere scoperte su richiesta dagli utenti; i quali, contattando un cluster Openstack, ne verificano run time l'abilitazione alle funzioni Cloudlet. Il meccanismo delle estensioni è realizzato permettendo agli sviluppatori di utilizzare le proprie *classi Extension*. La classica catena di chiamate per Openstack prevede che una richiesta di API da parte di un utente arrivi prima ad una classe extension, che di conseguenza sceglie a chi demandare l'interrogazione. Alcune chiamate, se necessario, attraversano il livello di messaggistica per arrivare al corrispondente nodo Compute. I comandi riguardanti le funzioni da eseguire sulle macchine virtuali vengono inviati dal nodo Compute all'hypervisor che li eseguirà tramite un driver di comunicazione. L'esito dell'operazione viene consegnato al cliente ripercorrendo la catena delle chiamate all'inverso. L'implementazione delle funzionalità Cloudlet segue questa stessa filosofia; come si può notare dalla figura che mette a confronto la gerarchia di chiamate, sono state inserite delle estensioni per intercettare quelle di interesse per il Cloudlet. Le chiamate possono essere dirette su dei file custom cambiando dei parametri nei file di configurazione di Openstack. Le classi modificate per il Cloudlet sono speculari a quelle classiche per Openstack, ma con nomi e funzioni diverse: *cloudlet.py*, *cloudlet_api.py*, *cloudlet_manager.py*, *cloudlet_driver.py*. La gestione dell'intero sistema, grazie a questo meccanismo di aggiunta dinamica, diventa relativamente semplice; è possibile abilitare o disabilitare le funzioni Cloudlet semplicemente copiando dei file

in percorsi standard creati da Openstack. Le chiamate verranno indirizzate ai file modificati, invece che a quelli standard, come indicato nei file di configurazione di Openstack. Le funzionalità aggiuntive sono state realizzate in linguaggio di programmazione *Python*, essedo quello usato da Openstack.

Di per se Openstack è uno strumento complesso che comprende tutte le funzionalità standard richieste in ambienti Cloud. Gli sviluppatori di Elijah hanno previsto un utilizzo più leggero delle funzionalità Cloudlet, slegato da quelle Openstack, fornendo una versione stand-alone. Questo permette di testare in modo autonomo le caratteristiche Cloudlet, rendendo il debug più agevole.

3.4.1 Openstack++

Le funzioni implementate da Openstack++ sono quelle presentate per il modello Cloudlet. Si può *importare una macchina virtuale di base* tramite il gestore di immagini Glance, proprio degli ambienti Openstack. Questa funzione serve a popolare anticipatamente le centrali Cloudlet con le immagini base dei sistemi operativi maggiormente usati; caratteristica essenziale del modello che, come anticipato, prevede di eseguire un controllo, preventivo alla messa in funzione di una macchina virtuale, circa la presenza della relativa macchine base. L'importazione dell'immagine base è eseguita off line,

dall'amministratore di rete, al momento dell'installazione della piattaforma Elijah su una centralina Cloudlet.

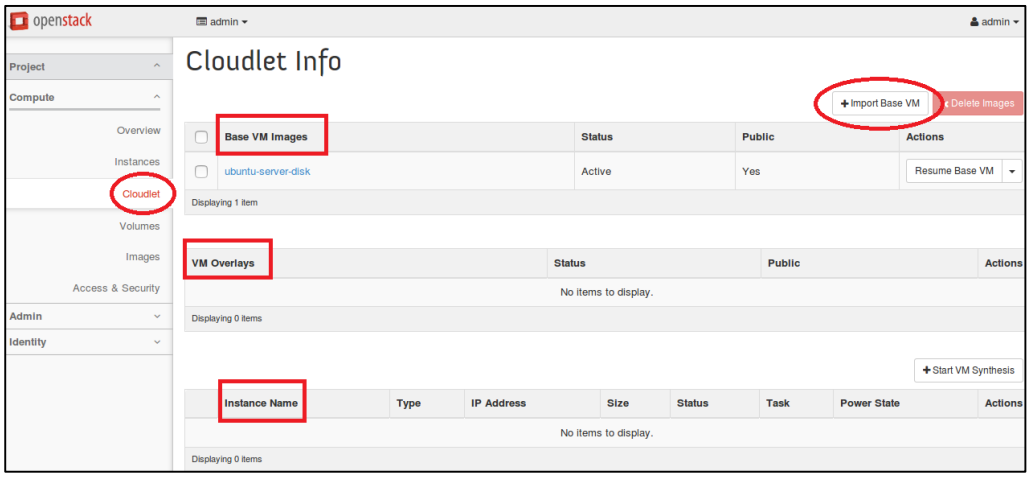

Figura 11 Interfaccia utente Openstack relativa al Cloudlet.

L'interfaccia utente principale della piattaforma mostra una visuale con tre tabelle indicanti, dall'alto verso il basso, rispettivamente le basi, gli overlay e le macchine virtuali Cloudlet attualmente disponibili. Da questa schermata si può eseguire l'importazione di una immagine base.

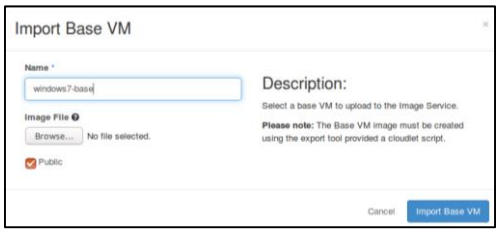

Figura 12 Importazione di una immagine base.

L'importazione di una immagine base in Openstack++ è equivalente al salvataggio di un *blob* tramite il gestore Glance di Openstack; le API utilizzate, infatti, sono uguali a quelle di Glance, con l'unica differenza di essere eseguite diverse volte, invece che una sola come avviene per le tipiche immagini Cloud.

Una immagine base è, in realtà, un file compresso contenente quattro componenti al proprio interno: un'immagine del disco di base e il relativo valore hash, uno snapshot della memoria e il relativo hash. Questi quattro file vengono prima decompressi e poi salvati singolarmente in Glance. Essi, anche rappresentando entità separate per il sistema di storage, sono legate dal comune valore della proprietà *base_sha256_uuid*, che ne identifica l'appartenenza ad una stessa immagine base avente quel particolare identificati univoco. Oltre a questa proprietà aggiuntiva, le immagini Cloudlet salvate in Glance contengono anche un particolare flag, *is_cloudlet*, che le identifica come Cloudlet, a differenza di quelle Cloud, che potrebbero comunque coesistere all'interno dello stesso gestore di immagini. Un'altra proprietà degna di nota, che viene usata successivamente dal driver, è *base_resource_xml_str*. Questa contiene l'intera configurazione *libvirt* che indica le caratteristiche che devono avere le macchine virtuali generate da questa base. Alla ripresa, o sintesi, di una macchina virtuale viene creato un corrispondente *flavor*, se esso non è già presente sotto stesso nome; ovvero un profilo Openstack che indica le caratteristiche delle nuove istanze virtuali avviate.

Di solito l'importazione di una macchina base è seguita dalla sua ripresa, eseguita tramite l'opzione *Resume Base VM*. Questa operazione, anch'essa eseguita off line, rappresenta il momento in cui gli sviluppatori del servizio Cloudlet settano la propria macchina virtuale con le librerie e i server utili all'esecuzione del particolare servizio.

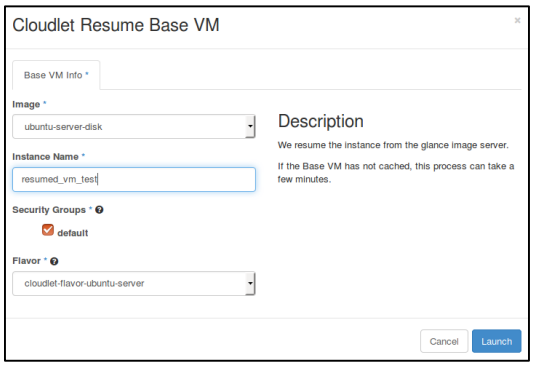

Figura 13 Ripresa di una VM a partire da un'immagine base.

L'istanza resumed verrà visualizzata nella tabella inferiore presente nella scheda principale di Openstack++ relativa al Cloudlet, mostrata nella relativa figura. Questa operazione è analoga alla ripresa di una macchina virtuale da uno snapshot per quanto riguarda la versione classica di Openstack, con la differenza che adesso la chiamata, prima di arrivare all'hypervisor, passa dal *CloudletDriver*, classe che eredita le proprie capacità dal driver di base *LibvirtDriver*. Il driver modificato esamina la richiesta proveniente dall'utente e interroga l'immagine richiesta sulla sua natura Cloudlet. Se il controllo è positivo, invece di avviare l'istanza dal boot, il driver Cloudlet la riprende dallo snapshot.

Successivamente all'installazione delle librerie e dei server necessari all'esecuzione dei servizi Cloudlet presenti sulla macchina virtuale, di solito lo sviluppatore chiede ad Openstack di *generare un overlay* a partire da una istanza ripresa e ancora in esecuzione. Questa operazione genera degli snapshot utili a riprendere la macchina da questo punto e successivamente li comprime in un file overlay scaricabile, memorizzandolo in

Glance e cancellando la macchina virtuale da cui è stato creato il file. Questa nuova funzionalità viene aggiunta tramite il meccanismo delle estensioni, definendo una nuova classe, *CloudletAPI*, che eredita il proprio comportamento da *nova_rpc.ComputeAPI*. L'overlay, o un link utile al suo download, viene consegnato, dal provider del servizio, agli utenti mobili; i quali a questo punto sono abilitati all'esecuzione dei servizi Cloudlet per cui l'overlay è stato generato.

L'operazione di *sintesi*, presentata in precedenza solo in maniera teorica, è eseguita run time. Tramite questa procedura, un utente mobile può avviare il provisioning di una macchina virtuale, contenente un back-end server utile all'esecuzione del servizio, su un vicino Cloudlet. Per sintetizzare una macchina virtuale occorre indicare l'URL relativo all'overlay da utilizzare per la sintesi. Il risultato è una macchina virtuale mostrata sempre nella tabella relativa alle istanze, ma, questa volta, di tipo provisioned, invece che resumed come nel caso Resume Base VM. Il metodo è molto simile a quello usato per l'avvio di istanze in Openstack, con la differenza che il messaggio *HTTP POST* generato dall'utente contiene la parola chiave *overlay_url* che ne identifica la diversità. Oltre a questa nota distintiva, il payload JSON del messaggio usato per la sintesi, contiene anche il nome,

l'indicazione di quale immagine base del disco si sta usando e il flavor da adottare in fase di avvio della nuova istanza.

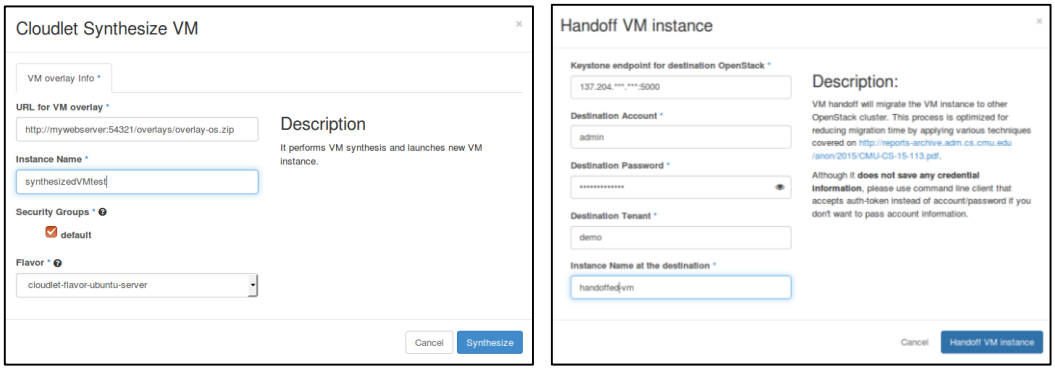

Figura 14 Interfacce utente per la Sintesi e l'Handoff di una macchina virtuale in Openstack.

Una macchina virtuale sintetizzata può essere terminata, qualora l'utente avesse esaurito le proprie richieste, o in alternativa trasferita tramite un *handoff*. Per trasferire, verso una nuova piattaforma Openstack, un'istanza di una macchina virtuale in esecuzione, occorre che il richiedente possegga i permessi di accesso sia dell'ambiente Openstack sorgente, che di quello destinazione. Le credenziali occorrono per richiedere le API e vengono inserite nel payload del messaggio, ove naturalmente il cliente indica anche l'URL di destinazione della macchina virtuale. Le macchine sorgente e destinazione dove risiedono le centrali Cloudlet possono essere molto eterogene tra loro in termini di capacità computazionali, e non godere delle stesse condizioni di rete. Un trasferimento di una macchina virtuale in esecuzione porta con se dei rischi relativi alla compatibilità dell'hardware installato sulla destinazione. Non è raro incappare in incompatibilità a livello delle caratteristiche del microprocessore [36] che potrebbe presentare, alla sorgente, delle caratteristiche richieste superiori a quelle che la

destinazione può fornire alle macchine virtuali create o, nello specifico, trasferite. In questo caso, sarebbe problematico per l'hypervisor fornire alla macchina virtuale queste proprietà, di solito rappresentate dai flag della CPU, non ereditabili dal microprocessore del sistema host. Fortunatamente gli hypervisor più comuni supportano un set predefinito di modelli di CPU assegnabili alle virtual machine, all'interno del quale esiste una catena gerarchica di compatibilità, con le relative caratteristiche per ogni modello. Confrontando il tipo dei processori reali installati sulle centrali Cloudlet in esame, se ne possono identificare i punti in comuni e quelli di incompatibilità, individuando un modello comune che possa essere riprodotto virtualmente su entrambi i sistemi.

Il team di lavoro di Elijah ha proposto una modalità che accelera la produzione di un overlay. L'esigenza di trasferire nel minor tempo possibile una macchina virtuale verso una centrale Cloudlet più prossima è richiesta soprattutto dal processo di handoff, più che dalla creazione dell'overlay per i clienti, che avviene invece off line. Con l'obiettivo di ridurre la finestra temporale che occorre al completamento di un handoff; la procedura è stata arricchita di parametri configurabili run time a seconda delle caratteristiche attuali della rete e del carico computazionale del server Cloudlet [33]. Si ricordi che l'overlay è la differenza tra lo stato di disco e memoria di una macchina virtuale in esecuzione, e la corrispondente immagine base. Dovendo trasmettere questi dati al Cloudlet di destinazione, tralasciando le parti della macchina virtuale che sono rimaste inalterate rispetto alla macchina base, occorre ottimizzare il

processo di creazione dell'overlay. Lo stream dei dati modificati è diviso in chunk di dimensione minore, ognuno dei quali è confrontato con la rispettiva parte della macchina base e scartato se uguale ad essa. Per questi confronti viene usata una funzione di hash, di solito *SHA-256*. Tramite il processo di *de-duplicazione* ogni chunk modificato viene confrontato con tutti i chunk del disco, della memoria della macchina base e con tutti i precedenti chunk di questo passo. In questo modo ogni duplicato viene sostituito da un link che punta alla sua copia originale. L'ultima elaborazione che i dati subiscono prima di essere immessi in rete è quello della *compressione*. Per non pesare troppo sulla computazione del nodo host Cloudlet, che potrebbe diventare un collo di bottiglia al pari della rete, si useranno diversi algoritmi a seconda della disponibilità computazionale attuale. Uno degli strumenti utilizzati è *GZIP*, che ha una modesta frequenza di compressione e usa poco le risorse. In alternativa, si ha l'algoritmo *LZMA*, ottimizzato per alte frequenza di compressione e decompressione a discapito della potenza utilizzata. Un metodo mediano è rappresentato invece dall'utilizzo di *BZIP2*. Una ulteriore ottimizzazione di tutti questi passi è la loro *esecuzione in parallelo*, che utilizza le capacità multi core dei moderni processori, attivando simultaneamente tutti i passi precedentemente illustrati. Il vantaggio principale rispetto alla serializzazione dei vari passi risiede nel poter iniziare l'invio in rete dei dati relativi all'overlay, parallelamente alla loro elaborazione. Iniziare il trasferimento il prima possibile ottimizza le risorse temporaneamente utilizzate per memorizzare i dati intermedi, che i vari passaggi di lavorazione creano.

3.4.2 Applicazioni Elijah

Per incentivare l'utilizzo della piattaforma Elijah e mostrarne i soddisfacenti risultati, il team responsabile del progetto, ha sviluppato e messo a disposizione anche un set di applicazioni dimostrative di esempio. L'applicazione di maggior peso, dal punto di vista dell'interesse mediatico, è sicuramente *Gabriel* [37]. Il team di Elijah ha pensato un servizio che sfrutta la possibilità di catturare le immagini dai dispositivi indossabili, per poi elaborarle in remoto, fornendo una risposta real-time all'utente. Il dominio applicativo scelto si concentra sull'assistenza cognitiva a persone che, a causa dell'età avanzata, di malattie o incidenti, sono affette da disabilità, che non permettono loro un regolare svolgimento delle funzioni quotidiane. Gabriel viene proposto come un software che guida questa categoria di utenti durante il loro declino cognitivo. Gli umani sono molto sensibili ai ritardi che si verificano nella comunicazione, tanto più se essa involve un processo grafico. Gabriel si è dovuto scontrare con i problemi derivanti dalla richiesta di una latenza molto bassa, di poche decine di millisecondi, unita alla necessità di trasferire il carico su una macchina più performante del dispositivo indossabile che produce effettivamente i dati da analizzare. Il team di Elijah propone il Cloudlet come piattaforma che fornisce bassa latenza e risorse aggiuntive ai dispositivi indossabili che eseguono il servizio di assistenza cognitiva Gabriel [38].

Un'altra applicazione che sfrutta le centrali Cloudlet per l'off-load delle operazioni onerose in termini di risorse è

GigaSight. Gli sviluppatori hanno pensato ad un framework che si occupasse di tutto il ciclo che attraversano i video caricati dai dispositivi indossabili; dalla loro creazione, alla modifica, fino alla condivisione. La miriade di immagini riprese da telecamere sparse in ambienti privati, ma anche pubblici, nelle Smart City ad esempio, prima di essere memorizzata e condivisa ha bisogno di attraversare il cosiddetto processo di denaturazione, che elimina dalle immagini qualunque riferimento ad informazioni private, come volti, targhe di veicoli o altre informazioni distintive, per tutelare la privacy dei soggetti ripresi. GigaSight, basandosi sulla località e su altre informazioni contestuali fornite tanto dalle centraline radio quanto dai sensori sui dispositivi indossabili connessi, condivide ed elabora video di interesse per quella zona. Essi possono essere distribuiti agli interessati in tempo reale e vengono memorizzati e modificati sulle centrali Cloudlet stesse per non consumare le risorse dei dispositivi [39]. Tutti i video e la relativa indicizzazione sono ora sotto il controllo della centrale sull'edge della rete, solo i più significativi o quelli che hanno bisogno di una condivisione geografica più ampia andranno a sincronizzarsi con la rete esterna su piattaforme Cloud.

Queste soluzioni mirano ad ampliare l'interesse suscitato dalla nuova tecnologia Cloudlet, fornendo una prova di come un ammodernamento dell'infrastruttura Cloud esistente possa apportare enormi vantaggi all'esecuzione di applicazioni dai requisiti elevati, in ambienti ostili.

3.5 Implementazioni Edge Computing

Data la necessità di un livello intermedio tra i dispositivi mobili, soprattutto della famiglia IoT, e il livello Cloud, le comunità scientifiche e industriali stanno gradualmente proponendo il Fog Computing come un passo necessario da compiere e un modello vantaggioso da adottare. Il Fog Computing fornisce una nuova catena di valore che permette agli operatori di aprire l'edge delle loro Radio Access Network a terze parti fidate per permettere il deploy semplice e veloce delle loro applicazioni. Una migliore esperienza si traduce in un valore aggiuntivo da proporre ai clienti, che vedono le loro applicazioni arricchite da una latenza molto bassa e da informazioni contestuali. Non solo gli sviluppatori di applicazioni beneficiano delle nuove opportunità economiche introdotte dall'Edge Computing, ma anche gli stessi operatori e fornitori di apparati di rete, che potranno offrire nuove tecnologie abilitate all'esecuzione di servizi Edge Computing. Le previsioni future ritengono che il Fog [40], ed in particolare il MEC Computing [41] diventeranno una chiave abilitante fondamentale per il nuovo standard di rete mobile 5G. Un nuovo *Industry Specification Group* (*ISG*), formatosi nel 2014 [42], è al lavoro sulla standardizzazione, a livello industriale, delle piattaforme, sia hardware che software, riguardanti il MEC. Il gruppo tiene incontri ad intervalli di tempo regolari, come è prassi per la redazione di un standard di questa portata; ai quali contribuiscono importanti compagnie impegnate nel campo dell'Information Technology, come ad esempio Huawei, IBM, Intel, Nokia Networks, NTT DOCOMO e Vodafone [43].

Intel sta collaborando con *Saguna* per usare il MEC al fine di migliorar le performance delle proprie reti mobili [44]. Saguna personalizza il modello 3-tier inserendo, nel livello occupato dal Fog Computing, un server MEC e, come estensione del livello Cloud, un gateway MEC. *Saguna Open-RAN* è una piattaforma software virtualizzata che consente il deploy di applicazioni MEC di terze parti.

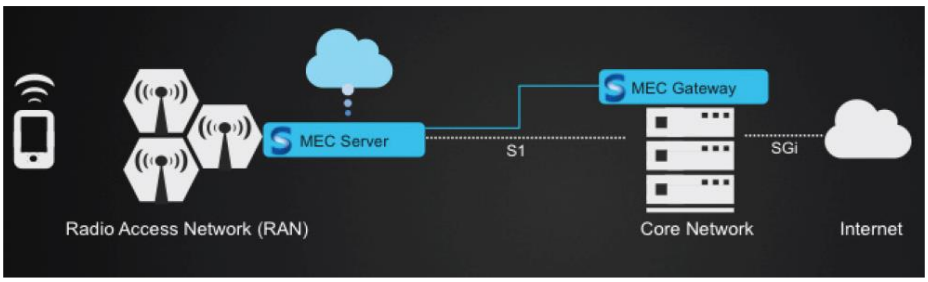

Figura 15 Infrastruttura MEC proposta da Saguna.

Nel modello proposto da Saguna, i *MEC Server*, che gestiscono le richieste, vengono aggregati alle classiche centrali RAN, mentre il *MEC Gateway* viene posto nella rete centrale. Un MEC Server, a seconda delle capacità e dei servizi proposti, può servire una sola base station o anche un gruppo di access point che condividono un qualunque contesto. Il modello di riferimento non specifica a che tipo di stazione i MEC Server debbano essere connessi proprio per lasciare spazio a estensioni che supportino qualunque tipo di protocollo di comunicazione mobile e di rete eterogenea. In questa architettura il MEC server che si occupa, oltre che delle richieste proveniente dai clienti, anche della fornitura delle informazioni che la prossimità offre, come, ad esempio, il carico attuale delle varie stazioni radio e la potenza del segnale. Il MEC Gateway è installato in una macchina virtuale eseguita su un server presente sulla Core Network, invece che

sull'Edge della rete. Il gateway proposto da Saguna, è un ponte di collegamento tra le nuove tecnologie e i comparti esistenti; è di supporto alla mobilità dei clienti e assicura ai MEC Server le classiche funzionalità di rete come politiche di controllo e meccanismi di integrazione con le architetture esistenti, soprattutto per quanto riguarda il livello Cloud. Le versioni attuali dei MEC Server sono basate su software Linux e sui potenti processori *Intel Xeon*. Purtroppo, anche essendo costruito su protocolli open-source, l'implementazione Saguna prevede attualmente solo una licenza commerciale.

Allo sviluppo della tecnologia MEC, partecipano anche vendor affermati, produttori di apparati di rete. Mentre *Huawei* propone l'importanza strategica della transizione su sistemi Edge Computing [45], *Cisco* realizza una vera e propria piattaforma compatibile con i servizi MEC. Cisco, come accennato precedentemente, è l'azienda leader che ha presentato il Fog Computing come soluzione abilitante dei nuovi scenari Internet of Everything. L'architettura *IOx* [46] è l'implementazione del livello Fog proposta da Cisco, che, allo stato attuale, permette il deploy di applicazioni, sviluppate da Cisco e da aziende partner, su una grande varietà di piattaforme hardware situate sull'edge della rete. Il framework si fa carico della comunicazione con il Cloud e della gestione delle varie applicazioni; abbracciando non solo il deployment, ma anche la possibilità di monitorare e gestire l'intero ciclo di vita.

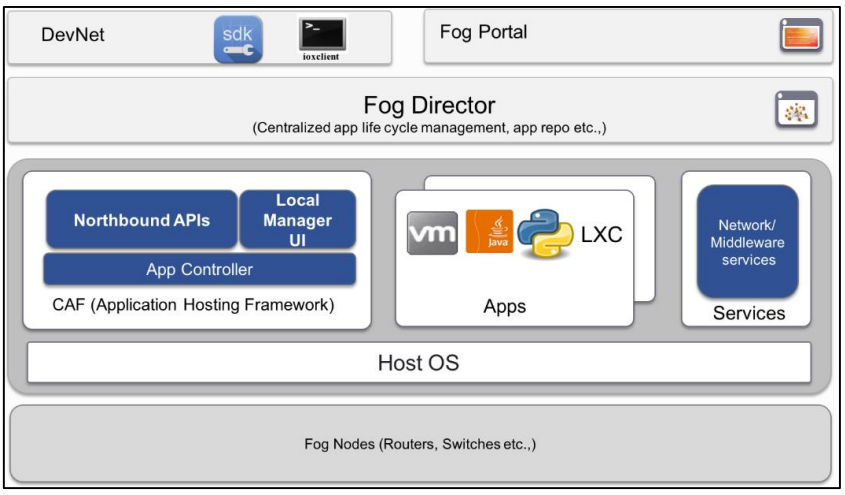

Figura 16 Framework Fog Computing proposto da Cisco.

Alla base dell'architettura di Cisco, a livello fisico, si collocano i *Fog Node*, che forniscono le risorse fisiche alle applicazioni inserite negli ambienti virtualizzati sovrastanti. L'*Application Hosting Framework* di Cisco (*CAF*) si occupa della gestione dei servizi sui nodi Fog, fornendo supporto al ciclo di vita e al reperimento delle risorse fisiche. Questo componente possiede anche degli strumenti per il troubleshooting e il debug. Il middleware IOx fornisce inoltre degli strumenti, sotto forma di API, utili ad accelerare lo sviluppo delle applicazioni. Mentre il CAF si occupa della gestione di un singolo nodo Fog, il *Fog Director* è il servizio di gestione centralizzato che orchestra tutte le applicazioni e i servizi sui nodi Fog presenti. *DevNet* invece è l'ambiente di sviluppo che consente di progettare applicazioni che possono essere ospitate sulle piattaforme IOx. Comprende *ioxclient*, un cliente utilizzabile da linea di comando, e un kit di sviluppo. L'ultimo componente pensato da Cisco è il *Fog Portal*, interfaccia principale per interagire con il sistema e con le applicazioni che si vogliono installare all'interno. Cisco ha reso disponibile per gli sviluppatori *IOx Sandbox* [47], una macchina

virtuale Linux che esegue il framework sul quale si appoggiano le applicazioni e i servizi IOx. Essendo un ambiente simulato ha naturalmente meno potenzialità rispetto ai device reali, ma permette l'acquisizione di una maggiore dimestichezza con il framework IOx, nonché lo sviluppo di applicazioni IoT in un ambiente protetto. Le versioni del framework sono anch'esse scaricabili [48], ma occorre possedere un contratto con l'azienda.

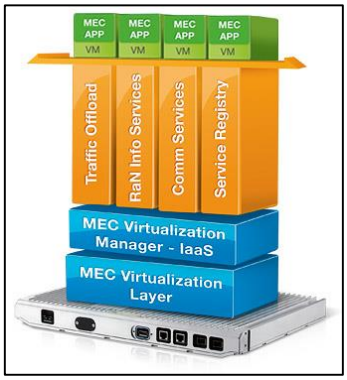

Figura 17 Piattaforma MEC AdLink.

Altra azienda leader nella produzione di hardware di rete è *AdLink*, che propone una tecnologia ad alte performance, compatibile con il Mobile Edge Computing, progettata per ambienti esterni estremi. L'apparato di rete fornisce i classici servizi MEC all'edge della rete, basati su ambienti virtualizzati. La parte inferiore dell'infrastruttura host è costituita dall'hardware, dal livello di virtualizzazione e dal gestore IaaS. Il livello dei servizi sovrastanti fornisce funzionalità utili alle applicazioni che l'infrastruttura ospiterà, come ad esempio il controllo del traffico e dello stato attuale dei vari canali radio e un meccanismo di registry, utile per reperire velocemente la locazione dei vari servizi. La soluzione AdLink si basa su

processore Intel Xeon e ha delle caratteristiche manifatturiere che la rendono resistente a traumi e vibrazioni, con possibilità di utilizzo in ambienti la cui temperatura sia compresa tra i -40°C e i +55°C. La piattaforma hardware ad alte prestazioni SETO-1000 è attualmente venduta solo da pochi distributori [49].

Una soluzione aziendale che minimizza la latenza nella comunicazione tra clienti mobili e centrale di elaborazione dati, è quella proposta da *Ducati* in collaborazione con *EMC* [50]. L'analisi dei dati riguardanti le competizioni sportive è stata sempre al centro dell'attenzione mediatica, ma con le nuove piattaforme Edge Computing è possibile spingerla ad un livello professionale che rispetti i requisiti real-time richiesti. L'applicazione del modello al reparto corse aiuta a velocizzare lo sviluppo di tecnologie che successivamente verranno applicate a tutta la categoria dei connected vehicle. Il progetto, in utilizzo nelle competizioni di MotoGp e Superbike, è volto alla rapida ottimizzazione del veicolo in base alle informazioni contestuali sulla sua condizione e su quella della pista, nonché degli avversari. EMC ha realizzato un sistema di elaborazione per Big Data chiamato *Isilon scale-out Network-attached Storage* (*NAS*) che Ducati usa per analizzare la miriade di dati che arriva dalle prove su strada, come ad esempio le prestazioni, la fluido-dinamica e la telemetria. La controparte adibita alla condivisione real-time è lo strumento Syncplicity, che permette una comunicazione sicura e veloce all'interno di ambienti eterogenei [51], quindi compatibile con la maggior parte delle tecnologie esistenti, incluse quelle Ducati. Le piattaforme utilizzate da

Ducati sono proprietarie della EMC, per le quali è possibile stipulare contratti esclusivamente economici.

Anche nel campo delle telecomunicazioni si assiste ad una spinta dei servizi verso l'Edge delle reti. Le aziende stanno lavorando sulle opportunità offerte dal MEC riguardanti le Smart City. Ci si concentra sui contenuti contestuali che possono essere forniti agli utenti in base alla loro posizione. Inoltre, dalle varie connessioni ai canali radio e alle centraline sparse per la città, si possono operare una serie di raccolte analitiche che suggeriscono, ad esempio, quali sono i mezzi pubblici maggiormente utilizzati o le linee cittadine troppo affollate. *Nokia* propone il modello MEC come soluzione principale per velocizzare la consegna dei contenuti agli utenti finali [52]. In collaborazione con *Siemens* [53] hanno lanciato un framework chiamato *Liquid NET*, che viene presentato come la prima piattaforma mobile-edge installabile all'interno di una base station. La piattaforma utilizza la piattaforma IBM *WebSphere Application Service Platform for Networks* come tecnologia per la virtualizzazione delle macchine virtuali ospitanti i servizi. Il server è installabile sulle base station *Nokia Flexi Multiradio 10*. Anche Telecom Italia contribuisce a divulgare l'importanza della virtualizzazione, ed in particolare del modello MEC [54]. Telecom sottolinea le crescenti opportunità economiche che l'esigenza delle attuali e future reti mobili crea a favore degli operatori di rete. Intercettando questa domanda è possibile agire tempestivamente, secondo Telecom, per proporre uno standard che soddisfi tutti i requisiti richiesti dal modello MEC proposto dall'istituto ETSI. Gli operatori, dopo lo sforzo per

l’affermazione della tecnologia, ne ricaverebbero dell’utile proponendo nuovi e ottimizzati contenuti ai propri clienti.

In questo capitolo si è assistiti ad una panoramica che mostrasse come importanti nomi nel mondo dell’IT stiano proponendo soluzioni alternative al modello Cloud, in alcuni casi tramite vere e proprie implementazioni attualmente utilizzabili. Il prossimo capitolo mostra l’architettura ideata per esporre gli scenari abilitati dalla nuova tecnologia sull’Edge della rete, introdotti fino a questo punto solo teoricamente. Il modello di riferimento scelto implementa i concetti ribaditi dal Cloudlet, tramite un’architettura a tre livelli per i quali viene proposto un pacchetto di software che, coprendoli tutti, assegna ad ogni strumento una funzione specifica a seconda della sua posizione all’interno della rete mobile.

CAPITOLO 4: PROGETTAZIONE DEL PROTOTIPO

Si è realizzato un prototipo del modello edge implementando i 3 livelli previsti dall'architettura mobile presentata nei capitoli precedenti. Il capitolo, dopo aver introdotto i requisiti utili allo sviluppo e alla realizzazione del prototipo, spiega in maniera generale come interagiscono le varie componenti della piattaforma. Dopo la panoramica sull'architettura, si passa alla presentazione per livelli dei vari agenti, quindi delle loro responsabilità e delle modalità di comunicazione con le altre entità del sistema. Per terminare sono presentate le scelte progettuali intraprese per personalizzare e adattare i meccanismi di sintesi ed handoff alle esigenze della piattaforma.

4.1 Introduzione

L'architettura a tre livelli introdotta dall'Edge Computing risulta di fondamentale importanza negli ambienti ostili, caratterizzati dalla scarsità di risorse dei clienti partecipanti e dalla non affidabilità della connessione alla rete esterna. Per la comunicazione tra le applicazioni software presenti sui dispositivi mobili e le macchine che effettivamente eseguono le computazioni pesanti per il servizio richiesto, è necessaria una

latenza minima che solo le risorse presenti sull'Edge della rete possono garantire. Si propone di seguito un prototipo che mostra gli scenari abilitati da questo tipo di architettura e l'implementazione dei classici meccanismi che essa descrive.

4.1.1 Requisiti

L'attuazione delle varie funzionalità previste per un modello architetturale che comprende tre livelli, contempla la presenza all'interno della rete di diversi strumenti utili all'attuazione dei servizi e alla loro gestione. Verranno di seguito elencate le entità che occorrono all'esecuzione del prototipo, rimandando la lista dei relativi componenti intersechi, al capitolo della trattazione che si occupa dei test valutativi.

- Un dispositivo mobile;
- Due macchine server comunicanti;
- Un Access Point Wireless di collegamento tra dispositivi mobili e macchine server;
- Una macchina Cloud in esecuzione.

I ruoli delle varie entità all'interno dell'architettura generale del prototipo verranno spiegati in maniera generale durante il resto di questo capitolo; per poi approfondire la parte implementativa e operativa delle singole componenti

successivamente, all'interno del capitolo relativo all'implementazione del prototipo.

4.2 Architettura Generale

Per l'implementazione di un prototipo che valutasse il guadagno prestazionale derivante dall'adozione di un terzo livello intermedio tra i dispositivi mobili e il Cloud, si è deciso di mantenere un carattere open-source per quanto riguarda gli strumenti utilizzati; a questo scopo si sono scelte tecnologie certificate con licenze free o open-source. Tutto il software utilizzato fa parte della famiglia dei prodotti open-source; per la quale è consentita la distribuzione, la visualizzazione e modifica del codice sorgente e la condivisione [55]. I termini di distribuzione del software open-source sono un approccio etico allo sviluppo di nuove tecnologie; essi prevedono la non discriminazione di nessun gruppo di utenti né dei campi di utilizzo dei prodotti a cui si applicano. Le nuove piattaforme, adottando una licenza di questo tipo, si assicurano una adozione molto più ampia e rapida, dovuta soprattutto alle possibilità di collaborazione e condivisione fornite agli sviluppatori. Le licenze maggiormente usate per proteggere un prodotto open-source sono quella redatte da *Apache* [56] e la *GNU General Public License* [57], e coprono tutti gli strumenti e le piattaforme utilizzate nello sviluppo del prototipo presentato in questa tesi.

Di seguito è mostrata una panoramica sull'architettura del prototipo, realizzato seguendo il modello a tre livelli che propone un middleware intermedio tra il livello Cloud e quello dei dispositivi mobili. Attualmente si è deciso di mostrare la centrale di collegamento radio e il server che esegue il middleware intermedio come entità fisicamente separate; ma, con l'avvento di nuove apparecchiature di rete, si prevede la disponibilità delle risorse direttamente sulla centrale di collegamento radio più prossima ai clienti mobili.

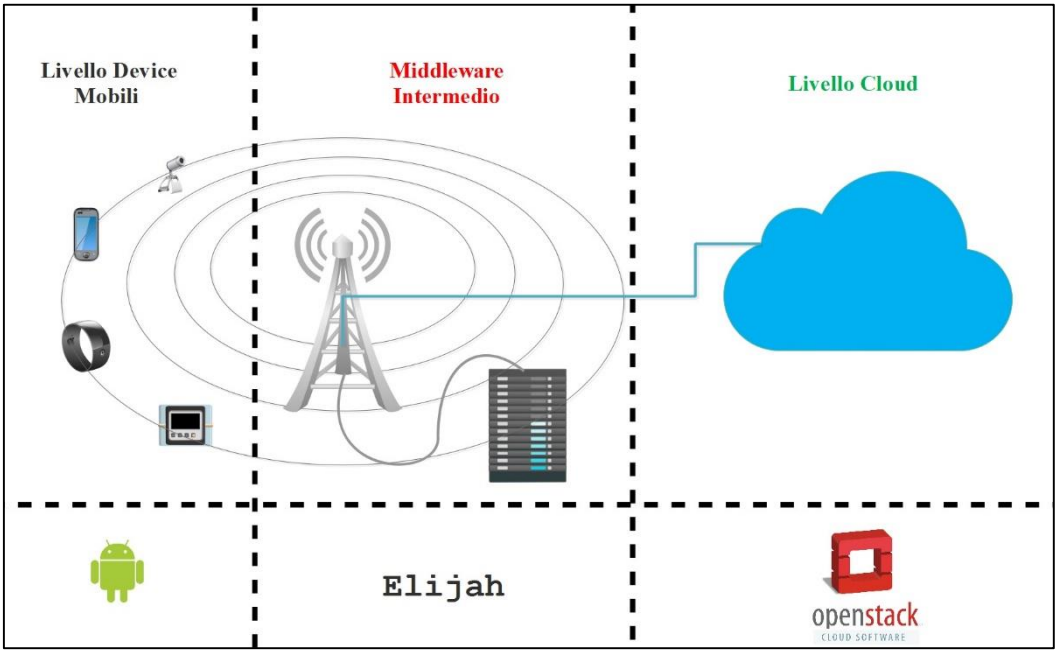

Figura 18 Architettura a tre livelli del prototipo, con relative tecnologie utilizzate.

Per il livello mobile si è deciso di simulare gli agenti operanti in ambiente ostile con device smartphone su cui è in esecuzione il sistema operativo *Android* [58]. La scelta non è vincolante, ma ci si è basati sulle recenti analisi di mercato che ne mostrano la maggiore adozione [59] rispetto ad altre piattaforme alternative.

Il livello intermedio rappresenta il server in esecuzione sull'edge della rete. La piattaforma che gestisce le risorse è *Elijah* [34], realizzata come estensione di Openstack, framework open-source cha ha molto seguito all'interno della comunità di sviluppatori Cloud. La scelta è stata dettata dal carattere open-source e dalla strategia di sviluppo del software, impostata sulla modifica di Openstack, che rappresenta un punto decisamente favorevole per una futura rapida adozione dello strumento rispetto alle alternative.

Il livello Cloud è utilizzato da quello intermedio per scaricare le immagini dei sistemi operativi oppure le componenti utili al servizio richiesto, qualora non possedesse già queste informazioni. La piattaforma del prototipo in esame, per essere operativa, ha bisogno solamente di file, dalle dimensioni contenute, scaricabili dalla rete Internet, nel caso in cui queste informazioni non siano reperibili in locale presso altre postazioni. Per il livello Cloud si è prevista una macchina virtuale con un indirizzo pubblico, su cui è installato un web server Apache da cui poter scaricare i dati di cui il livello intermedio necessita. Si è scelto di utilizzare le risorse fornite dalla piattaforma Openstack, per restare aderenti al carattere open-source del progetto. L'infrastruttura Cloud Openstack su cui è in esecuzione la macchina virtuale utilizzata, è manutenuta e gestita dal team di ricerca del dipartimento DISI dell'Università di Bologna.

4.2.1 Dispositivi Mobili

La funzione dei dispositivi mobili all'interno del prototipo è quella di simulare gli agenti operanti in ambiente ostile; i quali, tramite la loro caratteristica di essere connessi ad Internet, riescono ad eseguire computazioni complesse in remoto, risparmiando le esigue risorse di cui sono provvisti.

Come esempio di comunicazione tra il livello mobile e l'edge della rete, si è scelto un servizio che sfruttasse le risorse computazionali tanto da giustificarne l'esecuzione su macchine dedicate invece che sui dispositivi richiedenti. Il servizio prevede un'interazione tra il dispositivo mobile e un software, che utilizza le librerie *OpenCV* (Open Source Computer Vision Library) [60], installato sulla macchina virtuale fornita dal middleware. OpenCV è un software open-source per la computer vision. Le librerie sono una collezione di algoritmi ottimizzati per l'elaborazione di immagini e video. Oltre alle caratteristiche open-source e al successo che questo software ha raccolto, sia tra le emergenti start-up che tra affermate compagnie come Google, Microsoft, Intel, IBM; a guidare la scelta è stata la modularità del software, il quale per eseguire uno specifico task ha bisogno delle sole librerie che quell'algoritmo utilizza, e non di tutte quelle fornite. Inoltre esso viene fornito con delle interfacce di interazione compatibili con C++, C, MATLAB, Python e Java e supporta tutti i sistemi operativi attualmente più diffusi.

I dispositivi mobili in questione demandano al server sull'edge della rete le computazioni pesanti riguardanti la

modifica delle immagini. Al server è richiesto di identificare il numero di volti presenti in un'immagine inviata dal dispositivo mobile, il quale resta in ascolto della risposta. In un siffatto scenario, si ipotizza che la notifica dell'informazione richiesta debba avvenire con tempi di latenza molto ristretti, per permettere al dispositivo di agire di conseguenza. Il livello intermedio aiuta a ridurre notevolmente questa latenza, fornendo al cliente la risposta prodotta e rendendo disponibili gli ultimi dati modificati, in questo caso la foto più recente, a favore del cliente che li richiedesse in un secondo momento.

Oltre agli eventuali servizi predisposti per l'interazione con il middleware intermedio, si è prevista la presenza sui dispositivi mobili di un loro gestore. L'applicazione cliente che comunica con la centrale intermedia per la richiesta di risorse è separata dalle applicazioni che incarnano i servizi; questo perché essa rappresenta una sorta di gestore che, prima di avviare un servizio, ne verifica l'eseguibilità comunicando con l'host sull'edge della rete. Gli elementi di cui ha bisogno il cliente per contattare il servizio vengono forniti dal discovery. Il gestore di applicazioni, installato sul dispositivo, esegue il servizio Network Service Discovery di Google, impostato per essere sensibile alla presenza di una centrale operativa sull'edge della rete locale.

La richiesta di fruibilità di un servizio, fatta dai dispositivi mobili alle centrali intermedie, può avere una risposta dalle tempistiche variabili; a seconda che le risorse siano già pronte o debbano essere predisposte run-time. Data la limitatezza di risorse di cui sono dotati i dispositivi cliente interagenti con

l’edge della rete, su cui è in esecuzione il middleware, si è previsto un modello di interrogazione a polling sulla effettiva operatività della risorsa richiesta al server. Qualora il servizio necessitasse di essere sintetizzato al momento della richiesta bisogna tener conto delle tempistiche di questo processo; le quali oltre che dalle caratteristiche hardware della centrale su cui viene eseguito, dipendono anche dall’indicazione dell’utente circa la locazione dell’overlay utile alla ripresa del servizio. Esso potrebbe essere nella stessa rete locale ai servizi edge, in caso di applicazioni note o molto utilizzate, o remoto; e trovarsi potenzialmente in reti non attualmente raggiungibili o dall’elevata latenza.

Una possibilità data dalle strategie di sintesi, che non prevede comunicazioni con la rete esterna, è quella di contenere all’interno dei dispositivi cliente stessi i vari overlay utili alla predisposizione del servizio da parte del middleware intermedio. Volendo coprire anche questo caso si è previsto un web server in esecuzione sui device che esponga gli overlay dei servizi installati. Si è scelta a questo scopo l’applicazione kWS [61], dall’intuitiva e semplice interfaccia grafica, gratuita leggera e senza eccessivi oneri di gestione.

4.2.2 Middleware Intermedio

Per il livello intermedio è stato utilizzato il modello Cloudlet che permette di disporre di un *data-center in a box* sulla

rete ove risiedono i dispositivi mobili, di modo che la comunicazione goda delle caratteristiche richieste di latenza e banda che si sono prima esposte.

Si è deciso di realizzare il prototipo curando maggiormente l'aspetto più innovativo del protocollo a tre livelli, ovvero la comunicazione tra il cliente mobile, caratterizzato da risorse limitate, e un server, in esecuzione sul livello intermedio, che ne riceve le richieste e opera all'edge della rete. Per orchestrare le varie parti in maniera dinamica si sono previste diverse componenti software ulteriori sulla centrale a cui si collegano i dispositivi mobili. Di seguito viene riportato lo schema di interazione principale tra i due livelli.

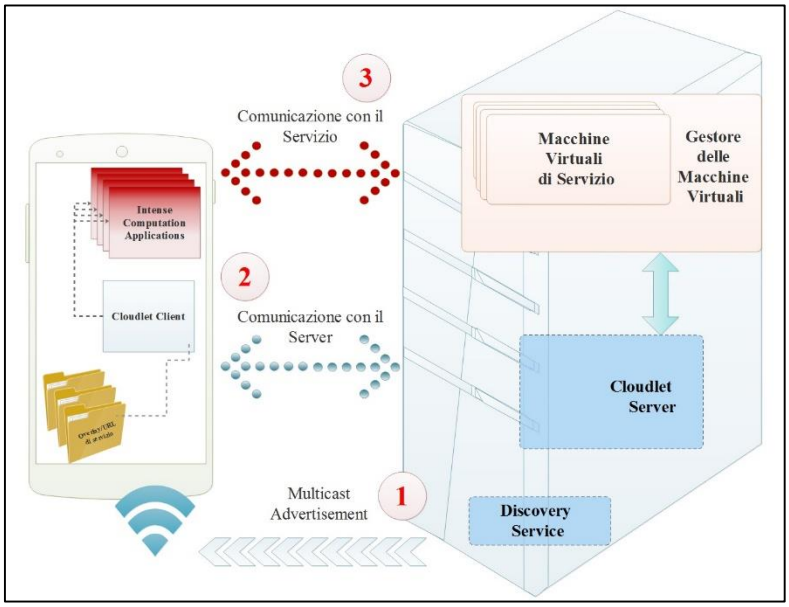

Figura 19 Panoramica sulla comunicazione tra dispositivi mobili e livello intermedio.

Il Server sull'edge della rete può essere automaticamente scoperto dai dispositivi facenti parte della stessa rete locale. Il

server pubblicizza la sua natura tramite l'invio di messaggi multicast contenenti il tipo e la descrizione del servizio middleware offerto. Gli utenti mobili sono stati resi sensibili alla ricezione dei messaggi multicast trasmessi dal Discovery Server presente sulla centrale. Il servizio di discovery, adibito a questo invio di informazioni, che si è scelto è Avahi, anch'esso open-source e presente di default in quasi tutte le distribuzioni Linux. Durante la fase di discovery,

Una volta ottenute dinamicamente le informazioni su come contattare il server, viene disposto un canale di comunicazione tra il server che comunica con il middleware intermedio e il gestore dei clienti dei servizi edge presente sul dispositivo. In questo momento, al livello intermedio pervengono le richieste riguardanti i servizi che i dispositivi vorrebbero eseguire presso la presente centrale. Se, dopo gli eventuali controlli, il server decide che è possibile avviare il servizio richiesto, comanda la creazione di una macchina virtuale al gestore, qualora non ne fosse già presente una che può eseguire il lavoro. Quando una nuova macchina virtuale è pronta, il server Cloudlet ne consegna gli estremi di connessione al cliente che, nelle invocazioni successive, tramite la parte cliente del servizio scelto, comunicherà direttamente con la macchina virtuale selezionata o appena creata.

Il middleware gestisce macchine virtuali su cui è installato il servizio di rilevamento dei volti che comunica direttamente con il livello mobile. La comunicazione con il servizio OpenCV prevede il trasferimento di una immagine dal dispositivo alla

macchina virtuale. Il server ospite esegue delle modifiche sull'immagine che mettono in risalto i volti presenti al suo interno, contornandoli con un rettangolo rosso. Per non allungare inutilmente i tempi della comunicazione, la risposta alla richiesta del numero di volti viene inviata il prima possibile al dispositivo richiedente; il quale, su richiesta, può scaricare l'ultima foto modificata, presente sulla macchina virtuale. Queste immagini, quella originale e quella modificata con il rilevamento dei volti, rappresentano l'unico stato del servizio presente sull'edge della rete. Se quest'ultimo dovesse essere interrotto e si volesse riprenderlo in un altro luogo, sarebbe possibile trasferirlo su un'altra centrale tramite procedura di handoff. A differenza della sintesi, che provvede solo all'installazione ed esecuzione del comparto operativo del servizio, l'handoff trasferisce anche lo stato temporaneo di quest'ultimo, ovvero quello rappresentato dalle immagini memorizzate. La comunicazione è possibile grazie all'informazione aggiuntiva, riguardante l'indirizzo pubblico, che il server di discovery invia al cliente. Il cliente possiede gli indirizzi pubblici di tutte le centrali edge visitate.

Ulteriore accorgimento che ottimizza l'utilizzo delle risorse e la reattività del sistema è stato caricare anticipatamente le centrali con le immagini base dei vari sistemi operativi. Per la ripresa o il trasferimento di un servizio serve solo il reperimento di un overlay da applicare alla specifica immagine base tra quelle memorizzate. Si ricordi che l'overlay contiene le modifiche da applicare all'immagine base del sistema operativo da cui è stato generato per riportarlo nella condizione di eseguire il particolare servizio. Le modifiche a memoria e disco di un'immagine base,

memorizzate all'interno di un overlay, hanno dimensione ridotte rispetto alle corrispondenti basi. Nel prototipo, ad esempio, si utilizzeranno file compressi sull'ordine del Gigabyte per le immagini base e di poco superiori ai cento Megabyte per l'overlay. Oltre ad utilizzare le risorse in maniere più efficiente viene ridotto il traffico di rete a beneficio della banda disponibile e viene accelerata la ripresa dei servizi.

4.2.3 Livello Cloud

Il livello Cloud superiore verrà interrogato durante le operazioni iniziali di set-up, durante la manutenzione, e per il download degli overlay dalle locazioni fornite dai produttori dei servizi. Durante la disposizione delle centrali, l'amministratore di rete provvede a scaricare dal Cloud le immagini base dei vari sistemi operativi che si vuole il server in questione supporti. Queste verranno precaricate durante la fase iniziale, di modo da averle pronte alle richieste dei clienti. Il Cloud partecipa al prototipo anche con un ruolo di locazione di backup, ove vengono memorizzati tutti gli overlay relativi ai servizi che gli utenti possono richiedere. Ai clienti viene rilasciato un URL, che punta ad un sito Cloud, dal quale poter scaricare l'overlay relativo al servizio che intendono utilizzare.

4.3 Funzionalità della soluzione integrata

Per il prototipo proposto, sono state progettate e realizzate le seguenti funzionalità:

- Sintesi di una macchina virtuale di servizio: azione eseguita tramite la consegna di un overlay dal dispositivo mobile alla centrale intermedia sull'edge.
- Servizio di rilevamento dei volti: attuato tramite la consegna di un'immagine ad una macchina virtuale di servizio in esecuzione sull'edge. La risposta consegnata al cliente comprende il numero di volti rilevati nell'immagine.
- Handoff di una macchina virtuale di servizio: il dispositivo, in seguito alla mobilità, può richiedere il trasferimento di una macchina, in esecuzione presso un sito edge precedentemente visitato, per continuare ad usare il servizio con lo stesso stato pregresso.
- Supporto Cloud: nel livello Cloud è inserito un web server che funge da repository per gli overlay utilizzati dai clienti.

Di seguito sono riportate nel dettaglio le scelte progettuali effettuate per modellare e personalizzare le modalità di sintesi e di handoff.

4.3.1 Sintesi

Il diagramma di sequenza seguente ripercorre i passi da compiere e i il significato dei messaggi inviati dai vari agenti durante il provisioning di un servizio nel prototipo. Il diagramma presuppone che il servizio sia avviabile e già pronto sull'host al quale arriva la richiesta.

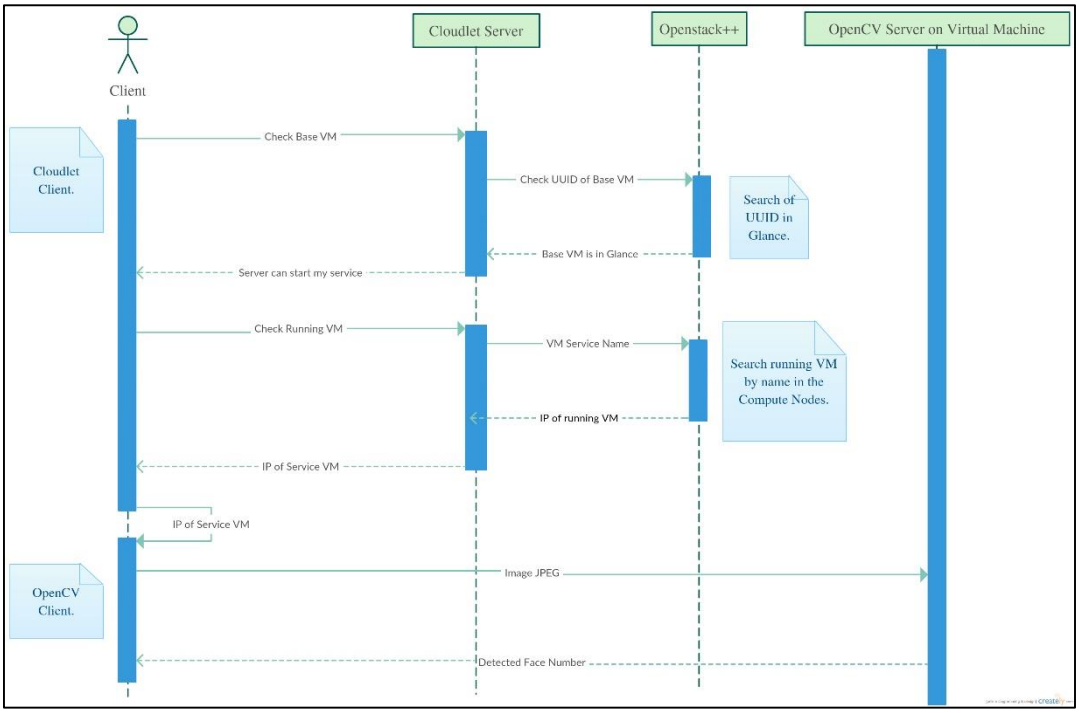

Figura 20 Diagramma di sequenza della sintesi.

Il cliente, mostrato come entità astratta nel diagramma, è implementato nelle applicazioni Cloudlet Client e OpenCV Client; che comunicano, rispettivamente, con il Cloudlet Server sulla centrale host e l'OpenCV Server sulla macchina virtuale di servizio. Come mostra lo schema, il gestore si occupa delle richieste di risorse e della comunicazione con il server sull'edge. Il server, comunicando con il gestore di macchine virtuali

presente sull'host, restituisce una risposta al cliente circa lo stato della sua richiesta; e, in caso di avvenuta sintesi, gli consegna gli estremi di connessione utili a colloquiare direttamente con la macchina virtuale che espleta il servizio.

4.3.2 Handoff

L'esecuzione del servizio potrebbe subire una interruzione a causa di una caduta del server, di una disconnessione o semplicemente del cambiamento di posizione da parte dell'utente; il quale, spostandosi, potrebbe non ricevere più il segnale radio che lo collega alla centrale Cloudlet. A questo punto occorre eseguire un nuovo discovery; quando una nuova centrale Cloudlet verrà rilevata il cliente potrà chiedere l'handoff della sua vecchia macchina virtuale di servizio presso la centralina attuale per continuare l'esecuzione del servizio interrotto. Il dispositivo consegna alla nuova centrale l'indirizzo di quella su cui è in esecuzione il servizio e ne richiede l'avvicinamento. Per chiarezza i passi compiuti dalla piattaforma sono riportati in un diagramma di sequenza, nella figura seguente, che mostra i messaggi scambiati tra il cliente e le due centrali che devono effettuare un handoff per riportare una macchina virtuale di servizio in esecuzione sul server che attualmente risponde alle richieste del dispositivo mobile.

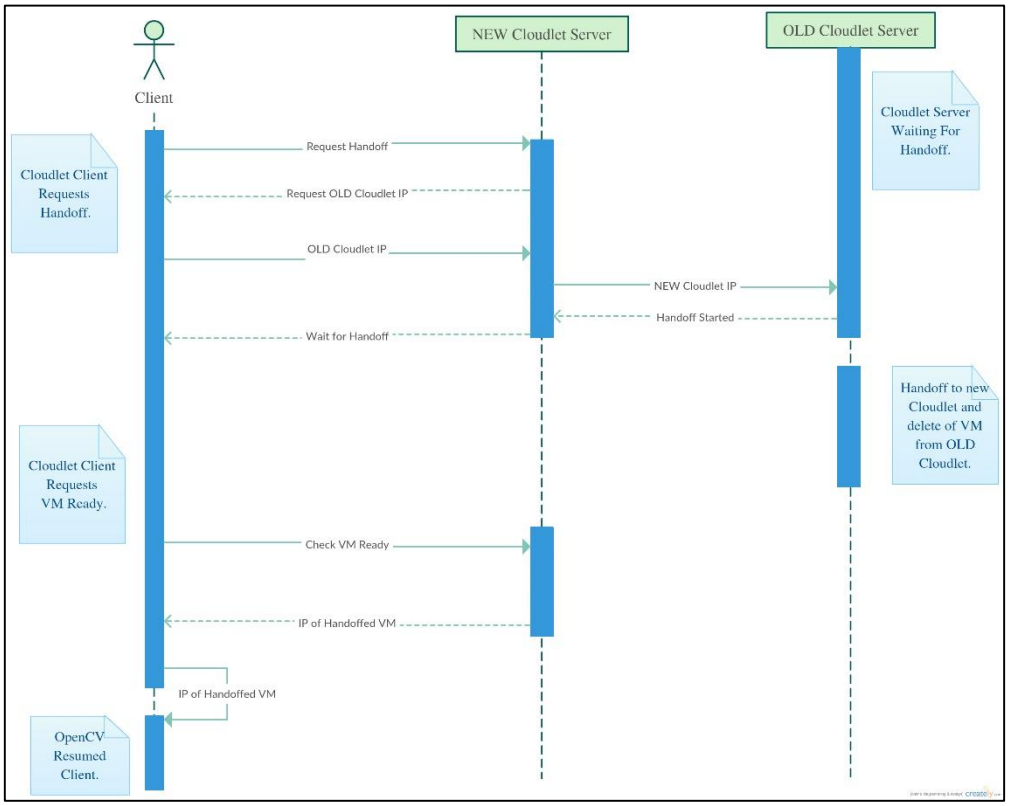

Figura 21 Diagramma di sequenza dell'handoff.

In questa fase, dopo che il dispositivo ha esplicitato al Cloudlet attuale la necessità di un handoff, la comunicazione avviene solo tra i due Cloudlet. La centrale, a cui il cliente mobile è collegato attualmente, riceve da quest'ultimo l'indirizzo pubblico del Cloudlet precedente. Il Cloudlet attuale, tramite questa indicazione, contatta la sorgente e le consegna il proprio indirizzo pubblico. Il Cloudlet sorgente comanda l'handoff verso l'indirizzo ricevuto dal Cloudlet destinazione e, ad operazione finita, cancella la macchina virtuale dalla sorgente e la rende disponibile alla destinazione, come previsto dal protocollo Cloudlet al termine di un handoff.

Elencate le principali scelte progettuali fatte in fase di sviluppo, nel prossimo capitolo si approfondiranno i passi compiuti per implementare la piattaforma proposta e la comunicazione tra le diverse componenti.

CAPITOLO 5: IMPLEMENTAZIONE E FUNZIONAMENTO DEL PROTOTIPO

Il capitolo spiega come installare e configurare le varie parti della piattaforma. Partendo dal set-up personalizzato di un host sull'edge, si mostrerà come è possibile creare un servizio tramite il middleware Elijah. In seguito verrà mostrato, anche tramite schermate di utilizzo, il funzionamento pratico del prototipo.

5.1 Installazione e Configurazione di Elijah

Per rendere operativi i due server Cloudlet, occorre installare la piattaforma Elijah su entrambe e modificare Openstack, con le librerie Cloudlet, affinché diventi Openstack++. La piattaforma Elijah viene fornita in due versioni: quella principale distribuita come estensione di Openstack [62], più una versione stand-alone per scopi di testing [63]. Il sistema che implementa totalmente le funzionalità del modello Cloudlet è solo quello rilasciato come plug-in di Openstack, caricato in git con il nome di *elijah-openstack*. La repository *elijah-provisioning* fornisce gli strumenti per effettuare la sintesi delle macchine virtuali a partire da un overlay in autonomia da altre piattaforme; ma il progetto è privo dell'implementazione del meccanismo di handoff. Per interagire con i servizi Cloudlet resi

disponibili da Elijah sull'edge della rete, il prototipo implementato opera esclusivamente tramite Openstack++, azzerando in questo modo gli oneri di gestione del sistema, durante la sua esecuzione, da parte degli amministratori di rete. Oltre alla mancanza di un meccanismo per l'handoff, la scelta di prediligere Openstack++ alla versione stand-alone è dovuta alla dinamicità della piattaforma nella ricca gestione di tutte le risorse del sistema. Inoltre chiunque ha dimestichezza con la piattaforma Cloud Openstack, non dovrebbe incontrare difficoltà nell'accogliere e gestire le nuove funzionalità messe a disposizione dall'espansione Openstack++ fornita da Elijah. I passi da eseguire per effettuare il provisioning di una macchina virtuale tramite la versione stand-alone verranno comunque illustrati per completezza durante la trattazione e perché questa metodologia operativa è stata utilizzata per la gestione iniziale dell'infrastruttura e per la verifica delle compatibilità in fase di test, in quanto più leggera del comparto di componenti avviato da Openstack.

5.1.1 Installazione di Openstack++

Per installare le librerie Cloudlet, contenute nella versione stand-alone, occorre eseguire questi comandi:

```
sudo apt-get install git openssh-server fabric
git clone https://github.com/cmusatyalab/elijah-provisioning
cd elijah-provisioning
fab install
```

L’installazione prevede la presenza sul server delle seguenti applicazioni: git, openssh-server, fabric. Tramite git ci si può collegare alla repository contenente le librerie e scaricarle per poi successivamente eseguirne l’installazione. Per testare il successo di questo primo passo si può eseguire il comando `cloudlet list-base`, che visualizza le immagini base presenti nel sistema. Alla prima esecuzione, il comando mostrerà solo una lista vuota, ma la mancata segnalazione di errori è utile alla verifica che il processo di installazione è terminato con successo. A questo punto il team di Elijah consiglia di installare Openstack, nella sua versione Kilo standard, tramite Devstack.

```
echo "$USER ALL=(ALL) NOPASSWD: ALL" | sudo tee -a /etc/sudoers
git clone https://github.com/openstack-dev/devstack
cd devstack
git checkout stable/kilo
cp samples/local.conf local.conf
./stack.sh
```

Come prima cosa, per l’utente viene aggiunta una regola che permette l’esecuzione dei comandi senza il prompt di richiesta della password. Il passo successivo è il download da git di Devstack e la sincronizzazione sulla versione `stable/kilo` tramite comando `git checkout`. Il file *local.conf* [64] viene utilizzato dallo script *stack.sh* per impostare le caratteristiche di gestione di Openstack in modo automatico durante l’installazione. La verifica della riuscita dell’installazione della versione originale di Openstack può essere attuata sia esaminando i numerosi file di log, presenti nel direttorio di default `/opt/stack/logs`, sia collegandosi all’indirizzo *http://localhost/* per osservare, dall’interfaccia grafica, se tutte

le componenti sono operative. Le operazioni di verifica e gestione possono essere preformate anche utilizzando le API messe a disposizione per i vari strumenti cooperanti in Openstack. Per utilizzare queste funzioni da linea di comando occorre eseguire il login tramite i meccanismi di autenticazione messi a disposizione da Openstack. Accertata la corretta operatività della piattaforma Openstack, si procede con l'installazione dell'estensione relativa alle funzioni Cloudlet e al riavvio di Openstack, questa volta tramutato in Openstack++. Si noti che spesso, dopo la sospensione e il riavvio dell'intero comparto di servizi Openstack, occorre riavviare manualmente sia il server `apache2` che lo strumento di autenticazione `keystone-all`.

```
git clone https://github.com/cmusatyalab/elijah-openstack
cd elijah-openstack
fab localhost devstack_single_machine
cd ~/devstack
./unstack
./rejoin-stack.sh
sudo service apache2 start && sudo service keystone-all start
```

Come si è già sottolineato, lo script di installazione previsto dal team di Elijah non cambia il codice Openstack esistente, ma sostituisce le proprie librerie a quelle pensati per la versione di default. Esso rappresento uno strumento che viene innestato come estensione, dando la possibilità di ritornare alla versione originale, qualora lo si desiderasse, semplicemente invertendo i passi necessari all'installazione. La configurazione di Openstack viene modificata dal processo di installazione agendo sulle proprietà degli strumenti presenti nella piattaforma, che sono elencate nel file `/etc/nova/nova.conf`. Oltre alle classiche

modifiche al file nova.conf introdotte per l'utilizzo di Openstack++, si è ritenuto vantaggioso aggiungere la possibilità di poter assegnare in automatico alle istanze un indirizzo IP, prelevato da un pool definito staticamente; azione questa che contribuisce all'automazione della comunicazione nel sistema. Per assegnare automaticamente un indirizzo *Floating IP*, tramite il quale poter contattare le istanze, occorre eseguire la procedura mostrata nell'elenco seguente, prima di riavviare i servizi nova.

1. Predisporre un pool di indirizzi, preferibilmente che non comprenda quelli assegnati in automatico dagli altri meccanismi di assegnazione dinamica eventualmente presenti nel sistema. Di seguito è mostrato il comando utilizzato per le impostazioni del prototipo:

```
nova-manage floating create --pool my_floating_pool –ip_range 192.168.1.224/27
```

2. Occorre modificare, con valori conformi a quelli usati nel comando precedente, il file di configurazione nova.conf. All'interno della sezione `[DEFAULT]` inserire le seguenti opzioni:

```
default_floating_pool = my_floating_pool
floating_range = 192.168.1.224/27
auto_assign_floating_ip = True
```

Tra le buone norme da seguire per evitare errori comuni in fase di set-up del sistema. Si ricordi che, per rendere effettive le modifiche alle impostazioni riguardanti Openstack, occorre

riavviare i rispettivi servizi il cui comportamento si è modificato tramite file di configurazione. Inoltre, si noti che l'installazione di Openstack è stata eseguita tramite lo script fornito da Devstack. Se per qualsiasi motivo lo sviluppatore si trovasse a dover eseguire nuovamente lo script `stack.sh`, le modifiche riguardanti Openstack++ verrebbero annullate e occorrerebbe installare di nuovo l'estensione contenente le librerie di Elijah, tramite il comando `fab localhost devstack_single_machine`.

5.1.2 Compatibilità Hardware

Terminata l'installazione del comparto riguardante la piattaforma Elijah, si illustra di seguito come adattarne le impostazioni alle caratteristiche dell'ambiente di sviluppo utilizzato.

Caratteristica fondamentale per la ripresa di una macchina virtuale in un nuovo server diverso da quello su cui è stata creata, ovvero quanto accade durante l'handoff, è la compatibilità della CPU di destinazione. Quando si esegue il deploy una macchina virtuale su un server, questa assorbe delle caratteristiche dalla CPU della macchina host, per poterle simulare nel proprio ambiente, il tutto grazie ai moderni hypervisor e ai nuovi flag per la virtualizzazione hardware dei microprocessori. Sebbene entrambi i vendor abbiano fornito i propri processori di flag dedicati alla virtualizzazione, alcune caratteristiche proprietarie rendono impossibile la simulazione completa e reciproca delle

caratteristiche dei processori installati sulle due centrali utilizzate. Si è scelto di simulare una tipologia di processore, frutto dell'intersezione tra le due CPU in gioco, che potesse essere compatibile con gli ambienti utilizzati. Grazie alla presente scelta si è anche dimostrato come, tramite opportune modifiche, sia possibile adeguare la piattaforma Elijah a lavorare con hardware altamente eterogenei; nel caso della piattaforma realizzata, si è superata l'incompatibilità tra microprocessori prodotti da vendor diversi.

La strategia che ha portato all'estrazione di una tipologia base di processore che potesse essere simulato da entrambi gli host è partita dall'esame di un particolare file di default presente nel file system di Linux alla locazione di default `/usr/share/libvirt/cpu_map.xml`. Dalle configurazioni classiche di Openstack, si può notare che il driver responsabile dell'interazione con KVM/QEMU è libvirt. Al di sopra di questo strumento è stata costruita una comoda interfaccia a linea di comando chiamata virsh. Tramite virsh è possibile chiamare le API di gestione messe a disposizione da libvirt per invocare operazioni sulle macchine virtuali. Lasciando la gestione del ciclo di vita delle macchine ospiti ad Openstack, il comando è stato utilizzato per indagare sulle caratteristiche che gli host potevano simulare e per intersecarle cercando un modello che meglio si adattasse alla nostra situazione. Invocando, da linea di comando Linux, `virsh capabilities`, si ottiene una lista di tutti i modelli di CPU che l'host può simulare in aggiunta alle caratteristiche della CPU host stessa. Qualora si possedesse un gruppo di macchine molto simili dal punto di vista delle

caratteristiche dei microprocessori si potrebbe impostare l'hypervisor in modo che veda tutte ed esattamente le stesse caratteristiche della CPU host. Nel caso in esame, e spesso come si prevede avverrà per le tecnologie eterogene dispiegate sull'edge della rete, le caratteristiche delle macchine che devono operare una migrazione sono molto diverse; il problema quindi consiste nel determinare il giusto modello di CPU che soddisfa il pool di host a disposizione. Il driver viene in aiuto fornendo due utili comandi: il primo, `virsh cpu-compare`, accetta un file di configurazione con all'interno una descrizione XML di un processore e lo compara con le caratteristiche della CPU host, dandoci una conferma sulla compatibilità o meno; il secondo, `virsh cpu-baseline`, prende anch'esso come parametro un file, questa volta contenente due modelli di CPU e risponde con l'indicazione di quali flag mascherare per creare un modello di CPU simulabile su entrambi. L'esito del controllo effettuato sugli host utilizzati ha estratto come compatibile il modello di CPU che nel file `cpu_map.xml` corrisponde al nome KVM64.

Date queste informazioni di compatibilità si è potuta modificare la configurazione di Openstack++ affinché avviasse macchine virtuali simulando sempre il processore del tipo prescelto. La modifica ha interessato i file della versione stand-alone e di quella Openstack di Elijah che vanno a configurare il file di Openstack `nova.conf`. All'interno di questo file, che tiene traccia di tutte le configurazioni del nodo Compute host, si può notare la presenza della configurazione relativa al modo operativo della CPU per quanto riguarda le macchine virtuali

gestite con libvirt; con la relativa impostazione, frutto dell'indagine sulla compatibilità:

```
[libvirt]
libvirt_cpu_mode=custom
libvirt_cpu_model=kvm64
```

A causa dei suddetti problemi di compatibilità si è presentata anche la necessità di ricreare una immagine base della macchina virtuale che ospita il servizio Cloudlet. La base fornita dagli sviluppatori di Elijah è stata costruita sulla versione di *Linux Ubuntu 12.04 LTS 32 bit*, mentre quella scelta per il presente progetto è la versione successiva *Linux Ubuntu 14.04 LTS a 64 bit*. La scelta è stata dettata dalle limitate capacità della versione a 32 bit del sistema Linux, che viene sostituita con quella più recente, a 64 bit. Quasi tutti i moderni microprocessori dotati di virtualizzazione hardware riescono ad emulare le caratteristiche di una CPU a 64 bit; sulla quale è utile installare un sistema operativo, anch'esso a 64 bit, abilitato ad eseguire le applicazioni più complesse che oggi vanno per la maggiore, come ad esempio librerie grafiche. Per creare una immagine base adatta all'importazione in Openstack++, si è scaricata una classica immagine base del sistema operativo, della suddetta versione, e si è creata una macchina virtuale ove installarlo. Per questo passo, si è deciso di utilizzare lo strumento *virt-manager*, utile alla gestione delle macchine virtuali. Esso si presenta con una comoda interfaccia grafica che visualizza tutte le macchine virtuali presenti nel sistema, a prescindere dal loro stato attuale, e le loro caratteristiche fondamentali. Date le non eccelse caratteristiche hardware degli host utilizzati, si è cercato un

compromesso nelle caratteristiche della macchina virtuale base, che poi saranno quelle ereditate dalle macchine create a run-time. Le macchine possiedono le seguenti caratteristiche:

- CPU = 1 VCPU;
- RAM = 1GB;
- HDD = 8GB;
- Software preinstallati: SSH Server.

Le macchine virtuali create tramite librerie standard di Linux e tramite questo tool grafico, vengono salvate come file img nella cartella di default `/var/lib/images/`. Per la creazione della base compatibile con il prototipo Cloudlet realizzato, si è utilizzata la versione stand-alone di Elijah che prevede questa utile funzione. Tramite il client a linea di comando si può avviare una macchina virtuale a partire da un file raw passato come parametro, eseguire le proprie operazioni e, in seguito alla chiusura, generare un pacchetto contenente la macchina virtuale base pronta per l'importazione in Openstack++. Il comando in questione è:

```
cloudlet base /var/lib/images/base_disk.img
```

Una volta avviata la macchina virtuale tramite Elijah, si è provveduto all'installazione di pacchetti software utili in molteplici ambiti applicativi. Ad esempio, si è installata la versione 1.8 dell'ambiente Java SE Runtime, utile all'esecuzione del back-end server previsto per l'interazione con l'utente mobile. Una volta concluse le installazioni preliminari, e

approdati ad una versione generica e completa del sistema operativo, contenente non librerie specifiche, ma applicativi presenti in quasi tutte le versioni dei sistemi adibiti allo svolgimento di servizi, è possibile chiudere la finestra di interazione con la macchina virtuale e tornare in automatico a quella che esegue l'originale comando `cloudlet base`. L'operazione finale è la genesi automatica degli snapshot dello stato di memoria e disco e il salvataggio di questi file all'interno del direttorio di default `~/.cloudlet/`. Questo percorso contiene cartelle, i cui nomi sono l'hash stesso delle immagini, formate ognuna da tutti i file relativi ad una stessa immagine base. Possiamo verificare la corretta creazione di una voce all'interno del database delle immagini Cloudlet tramite il comando `cloudlet list-base`. Per generare un unico file portabile da poter importare all'interno di Openstack++ occorre usare il comando:

```
cloudlet export-base ~/.cloudlet/hash/base_disk.img baseVM.zip
```

Al comando occorre il path che porta all'immagine base, visionabile tramite il comando che le elenca tutte prima citato, e il nome che si vuole attribuire al file risultate dall'operazione di packaging.

5.1.3 Software di Supporto al Middleware Intermedio

Completate le configurazioni relative all'ambiente Openstack++, di seguito viene presentato il comparto software di

supporto al sistema presente in entrambe le centrali. Si è precedentemente accennato al servizio di discovery che permette ai clienti mobili che arrivano nella rete ove risiede la centralina di scoprirla automaticamente, e riceverne di conseguenza indirizzo IP e porta con la quale mettersi in comunicazione con il Cloudlet. Si è fatto in modo, tramite Avahi Server, che le due centrali pubblicizzassero la propria natura Cloudlet inserendo nel percorso `/etc/avahi/services` una definizione del servizio Cloudlet con i seguenti parametri:

```
<?xml version="1.0" standalone='no'?>
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">
<service-group>
    <service>
          <type>_cloudlet._udp</type>
          <port>22222</port>
    </service>
</service-group>
```

Listing 1 Definizione XML del servizio Cloudlet pubblicizzato da Avahi.

Infine, per permettere alle due centrali una corretta comunicazione reciproca, è stato attivato un sistema di sincronizzazione temporale tramite protocollo *Network Time Protocol* (*NTP*). Il protocollo, facente parte della famiglia TCP/IP, è utile alla sincronizzazione in rete tramite richieste inviate a server specifici. I server in questione sono impostati dal file di configurazione di default `/etc/ntp.conf`, che non si è ritenuto necessario modificare. In realtà il sistema NTP è molto più complesso di come sembri in quanto si hanno diversi livelli di sincronizzazione, alla base dei quali troviamo quello collegato direttamente a degli orologi atomici che ne mantengono l'informazione temporale. Fortunatamente gli strumenti di gestione astraggono dalla sottostante complessità; si pensi che il

cliente tiene conto del ritardo della comunicazione con il server e corregge autonomamente l'informazione oraria. Sulle centrali Cloudlet comunicanti nel prototipo, è eseguito un cliente NTP, sulla porta di default UDP 123, che le sincronizza agli stessi server NTP.

Con le installazioni eseguite fino a questo punto, si sono rese operative le centrali Cloudlet. Openstack++, insieme agli altri strumenti presentati, permetterà la gestione delle risorse hardware e la comunicazione reciproca e con i clienti. Gli oneri di gestione non sono ancora terminati; i fornitori di servizi devono provvedere a fornire il loro software, sotto forma di overlay, agli utilizzatori. La creazione di un overlay avviene tramite la piattaforma Openstack++, a partire da una fra le immagini base disponibili per le centrali Cloudlet. Alla fine del processo, i provider dovranno provvedere a rendere disponibile un link sul Cloud da dove poter scaricare gli overlay relativi ai loro servizi.

5.2 Implementazione di un Servizio

La creazione di un overlay può avvenire sia tramite Openstack++, che con il software Elijah in versione stand-alone. Creare un overlay presuppone la presenza di una immagine base all'interno del gestore di immagini del middleware; ipotesi che si è precedentemente data per scontata come operazione

preliminare, al fine di velocizzare i passi di ripresa e sintesi delle macchine virtuali. Per quanto concerne la versione stand-alone di Elijah, non essendo garantita la presenza dell'immagine base, i passi necessari all'importazioni di una macchina base, al fine di generare un overlay, vengono comunque presentati. Si noti che l'uso di questa versione è consigliato solo in fase di test.

Con la versione stand-alone possiamo importare un'immagine base all'interno del database Cloudlet, qualora questa non fosse già disponibile con lo stesso identificativo univoco.

```
cloudlet import-base ./myBaseVM.zip
INFO create directory for base VM
INFO Decompressing Base VM to temp directory at
    /tmp/cloudlet-base-k7ANqB
INFO Place base VM to the right directory
INFO Register New Base to DB
INFO ID for the new Base VM:
          abda52a61692094b3b7d45c9647d022f5e297d1b788679eb93735374007
          576b8
INFO Success
```

L'output del comando `cloudlet import-base` mostra i passi eseguiti durante l'importazione di una immagine base. La base viene decompressa in una cartella temporanea dal prefisso *cloudlet-base* nel percorso di default `/tmp/`. Successivamente è memorizzata nel database e le viene assegnato un hash univoco con cui poter essere selezionata.

5.2.1 Ripresa di una Immagine Base

Come primo passo nel processo di creazione di un overlay in generale occorre importare l'immagine base su cui vogliamo costruire il nostro servizio. L'importazione, anche essendo un processo molto lungo perché comprende la decompressione dell'immagine e degli snapshot della macchina virtuale, non rallenta l'operatività dei servizi in quanto è eseguita offline, prima che il sistema sia effettivamente reso operativo per l'interazione con i clienti.

Su Openstack++ occorre eseguire il *Resume* dell'immagine base, in seguito al quale viene avviata un'istanza di un macchina virtuale. La macchina virtuale viene ripresa dal punto in cui si sono salvati gli snapshot presenti all'interno del file compresso importato come base. La prima volta che si esegue questa operazione, la sua durata può essere parecchio elevata, a seconda delle caratteristiche hardware della macchina host; durante i test effettuati, si è arrivati ad attendere fino a circa venti minuti, prima che una nuova macchina virtuale venisse ripresa su una centrale che non avesse mai avviato quella specifica immagine base. Questo inconveniente non grava sulla comunicazione con l'utente, in quanto eseguito off line, ma velocizza le eventuali riprese successive della stessa base, perché la importa nella cache del nodo compute. Alla fine di questo passo si ottiene una macchina virtuale di base pronta a ricevere le modifiche utili all'esecuzione del servizio previsto.

5.2.2 Creazione di un Overlay da una Immagine Base

Qualora si stesse utilizzando la versione stand-alone di Elijah, una volta importata o creata una immagine base, occorre eseguire il comando `cloudlet overlay`, fornendo come parametro la locazione del file immagine. Se l'applicativo che intendiamo installare all'interno della macchina virtuale prevede la comunicazione tramite delle porte specifiche, che vogliamo siano accessibili, occorre impostare l'opzione del comando overlay che permette il tunneling dei pacchetti da una porta sull'host ad un'altra sulla macchina virtuale.

```
cloudlet overlay /path/to/base_disk.img -- -redir tcp:2222::22 -redir
tcp:8080::80
```

Nell'esempio proposto si creano due tunnel; rispettivamente dalla porta 2222 dell'host si accede alla 22 sulla macchina virtuale, mentre dalla 8080 dell'host si viene indirizzati verso l'80 della macchina virtuale. Questo si traduce nella seguente modalità operativa in fase di creazione dell'overlay: se si vuole accedere al server ssh presente sulla macchina virtuale appena ripresa, si deve fornire al comando ssh l'indirizzo dell'host e la porta 2222; allo stesso modo volendo accedere al web server della macchina virtuale presente sulla porta 80, si deve inserire nel browser l'indirizzo della macchina host seguito dal numero di porta 8080.

In presenza di una macchina virtuale appena ripresa da quella di base, le operazioni da eseguire sono le medesime in

entrambi i casi di gestione delle risorse, ovvero con Openstack oppure con la versione stand-alone. Per prima cosa si è installato il back end server OpenCV che risponderà direttamente alle richieste del cliente, poi si sono copiate le librerie utili all'esecuzione del servizio di grafica e, infine, si è avviato il servizio, il quale si mette in attesa delle richieste degli utenti sulla porta 22222. Occorre notare che, il server viene distribuito insieme alle librerie utili alla sua esecuzione. Eseguire delle operazioni con le librerie OpenCV, non presuppone l'installazione preliminare di software aggiuntivo, ma solo la copia delle librerie utili all'interno del percorso standard in Linux Ubuntu, che contiene anche le librerie grafiche, ovvero nella cartella `/usr/lib/x86_64-linux-gnu/`. L'installazione di OpenCV è stata necessaria invece durante lo sviluppo, in quanto viene generato il file contenente l'interfaccia Java di comunicazione con le librerie, chiamato *libopencv_java310.so* per la versione 3.1 di OpenCV. Le librerie sono state estrapolate dal suddetto file tramite l'utilizzo congiunto di alcuni strumenti che fanno parte della bash di Linux. Il comando sottostante ne mostra l'uso fatto:

```
sudo ldd libopencv_java310.so |
    grep "=> /" |
          awk '{print $3}' |
                xargs -I '{}' cp -v '{}' my_lib_path/
```

Di seguito si riportano le parti più significative della progettazione dell'interazione tra il back end server e le librerie OpenCV importate nel sistema, utile al riconoscimento del numero di volti presenti in una immagine inviata al server da un

dispositivo mobile. Il pacchetto software con cui viene distribuito il server OpenCV contiene anche il file di libreria utile al rilevamento dei volti all'interno di una immagine; esso è chiamato `haarcascade_frontalface_alt.xml`. Questo file viene usato per costruire un'istanza della classe `CascadeClassifier`, specializzandola nel detecting dei volti. Di questo oggetto si usa il metodo `detectMultiScale`, al quale viene fornita sia l'immagine ricevuta, che è quella da modificare, che una matrice, oggetto della classe `MatOfRect`, in cui verranno immagazzinati i dati relativi ai vari volti rilevati. Il conteggio è immediatamente inviato al cliente, mentre l'immagine verrà modificata dal server, inserendo un rettangolo rosso intorno ad ogni volto, e poi salvata nella stessa directory in cui è stata precedentemente scaricata l'immagine originale proveniente dal device.

```
....
URLlibSOurl=getClass().getResource("/resources/libopencv_java310.so");
File libFile = new File (libSOurl.getFile());
String libSOPath = libFile.getAbsolutePath();
System.load(libSOPath);
System.setProperty("java.library.path", path);
....
Field fieldSystPath;
try{
    fieldSystPath = ClassLoader.class.getDeclaredField("sys_paths");
    fieldSystPath.setAccessible(true);
    fieldSystPath.set(null, null);
}catch(SecurityException | NoSuchFieldException |
           IllegalArgumentException | IllegalAccessException e) {
    e.printStackTrace();
}
....
URL resourceXml=getClass().getResource(
"/resources/haarcascade_frontalface_alt.xml");
File file=new File (resourceXml.getFile());
CascadeClassifier faceDetector = new CascadeClassifier(
                                          file.getAbsolutePath());
Mat img = Imgcodecs.imread(IMG_DIR+SOURCE_IMG_NAME);
MatOfRect faceDetections = new MatOfRect();
faceDetector.detectMultiScale(img, faceDetections);
System.out.println(String.format("Detected %s faces",
                                  faceDetections.toArray().length));
MainOpenCV.setFaceNumResult(faceDetections.toArray().length);
for (Rect currRect: faceDetections.toArray()){
   Imgproc.rectangle(img, new Point (currRect.x, currRect.y),
    new Point(currRect.x+currRect.width,
    currRect.y+currRect.height),
    new Scalar (0,0,255));
}
    ....
```

Listing 2 Interazione tra server Java e librerie OpenCV.

Dopo l'installazione delle librerie e del back end server sulla macchina virtuale, è possibile continuare con la gestione delle risorse del sistema. Se si sta utilizzando Openstack++, tornando alla pagina principale del Cloudlet, si può operare per creare un overlay selezionando la voce corrispondente dal menu delle opzioni relative alla macchina virtuale in esecuzione. Se, invece, si è usata la versione stand-alone, per comandare la creazione di un overlay, occorre semplicemente chiudere la console di interazione con la macchina virtuale ripresa. In entrambi i casi, vengono catturate tutte le modifiche apportate alla macchina ripresa, rispetto a quella base; successivamente ne

viene generato un overlay e infine la macchina virtuale viene distrutta. L'overlay è impacchettato in un file zip contenente i file binari minimi per ricostruire la macchina modificata a partire da quella base. Il file compresso contiene due tipi di file, i veri blob che hanno estensione *xz* e i meta-file informativi con suffisso invece *overlay-meta*. In Openstack++ gli overlay generati vengono memorizzati, con un hash univoco, all'interno di Glance per un successivo download. Invece per quanto riguarda la versione stand-alone di Elijah, l'overlay viene salvato in una cartella temporanea di default da dove è possibile prelevarlo per consegnarlo ai futuri utenti del servizio.

5.3 Operatività della piattaforma

Gli overlay creati con Openstack++ sono utilizzabili per la sintesi tramite la versione stand-alone di Elijah, e viceversa. Il prototipo proposto realizza la sintesi dinamica dei servizi in base alle richieste dei clienti. Prima di mostrare questa classica procedura, si illustra come sia possibile sintetizzare una immagine base a partire da un overlay con la versione stand-alone della piattaforma Elijah.

5.3.1 Sintesi

Viene di seguito riportata la procedura di sintesi di una macchina virtuale a partire da un overlay utilizzando la versione stand-alone della piattaforma Elijah. Questa è l'ultima funzionalità che la versione stand-alone mette a disposizione; dopo della quale, la trattazione si concentrerà esclusivamente sul funzionamento dinamico tramite piattaforma Openstack++. Per effettuare il provisioning occorre avviare un server per la sintesi e un relativo cliente.

```
synthesis_server
INFO ------------------------------------------------
INFO * Base VM Configuration
INFO 0: /tmp/cloudlet-base-k7ANqB/abda****76b8/myBaseVM.raw (Disk
    8192 MB, Memory 1040 MB)
INFO ------------------------------------------------
INFO * Server configuration
INFO - Open TCP Server at ('0.0.0.0', 8021)
INFO - Disable Nagle(No TCP delay) : 1
```

L'avvio del server è effettuato sulla porta di default 8021 dell'host. È possibile avviare il cliente anche su un altro host per poi indicare l'indirizzo dell'host remoto su cui eseguire la sintesi della macchina virtuale. In ogni caso, il comando per lanciare il cliente ha bisogno, oltre alla locazione del server, anche di un altro parametro indicante l'URL dell'overlay. Nell'esempio seguente, il server è in esecuzione sull'host locale al client, il quale possiede nel proprio file system anche l'overlay.

```
synthesis_client -s localhost –u
                    http://localhost:54321/overlays/overlay-os.zip
```

Se la sintesi ha successo verrà avviata una console che mostra l'esecuzione della macchina virtuale ripresa a partire dall'overlay indicato. Il prototipo in esame non è stato pensato come interagente con la versione stand-alone, principalmente a causa delle sue mancanze nell'implementazione di tutti i meccanismi del modello Cloudlet. Di seguito verrà presentata la classica operatività del sistema in fase produttiva, quindi con il supporto della piattaforma Openstack++.

La sintesi implementata da Openstack++ è stata automatizzata, all'interno del prototipo, per rispondere dinamicamente alle richieste di servizio da parte degli utenti. Occorre installare manualmente le due applicazioni di test, il gestore Cloudlet e il cliente del servizio OpenCV. Esse non sono state pubblicate sullo store, quindi prima di poterle usare occorre compiere dei passi preliminari per installarle sul dispositivo; passi che potrebbero differire a seconda della versione di Android utilizzata. Di seguito viene riportata la semplice procedura che abilita il deploy di applicazioni dalle origini sconosciute, sulla versione Android Lollipop:

1. Caricare i file apk sul dispositivo.
2. Nelle impostazioni di sistema selezionare *Sicurezza*.
3. Sotto la voce *Amministrazione Dispositivo* abilitare l'opzione *Origini Sconosciute*.
4. Per entrambe le applicazioni, cliccare sul file apk e acconsentire all'installazione.

Avviando l'applicazione che funge da gestore dei servizi edge e utilizzando il pulsante Start Discovery, è possibile abilitare la ricezione dei messaggi multicast che segnalano la presenza di un server edge nella rete locale. Il Cloudlet Client scansiona quelli pertinenti e ne restituisce le informazioni riportate, ovvero indirizzo IP e porta, se tali messaggi fanno parte di una pubblicazione da parte di un server sull'edge della rete.

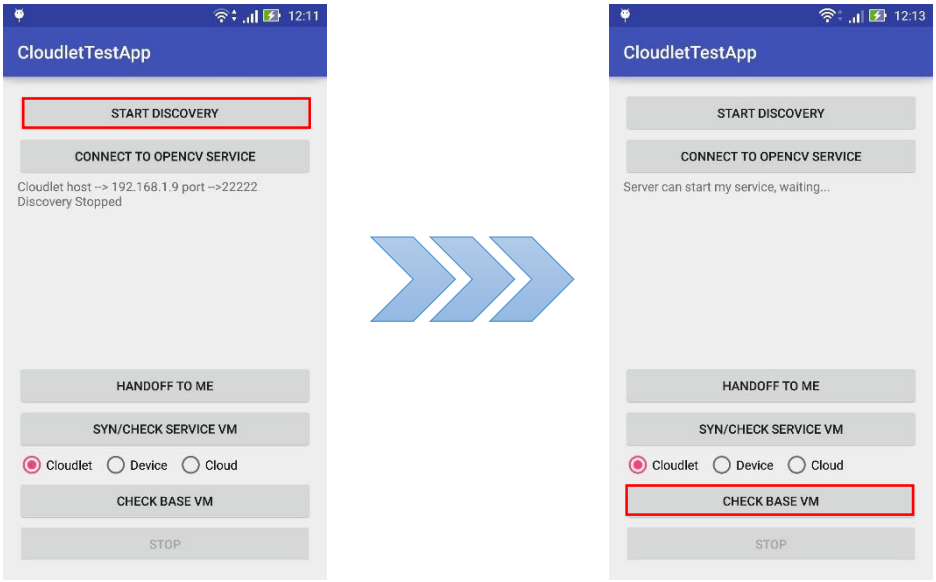

Figura 22 Discovery e verifica dell'immagine base.

Il passo successivo si attiva tramite la pressione del pulsante che verifica la presenza di una immagine base sulla centralina della quale il dispositivo ha appena scoperto l'indirizzo. La verifica ha esito positivo nel caso in cui sul server sia presente l'immagine base utilizzata per lo sviluppo del nostro servizio. Il server Cloudlet, scritto in Java, comunica con Openstack tramite le API messe a disposizione dal progetto open-source JCloud [65], invece è abilitato ad eseguire i comandi relativi ad Openstack++ tramite un cliente python fornito dagli sviluppatori di Elijah, che risponde al percorso di default `elijah-`

`openstack/client/cloudlet_client.py`. La scelta di JCloud è guidata dallo stesso team di Openstack [66], che indica questa distribuzione open-source come completa e portabile. È stato scelto il cliente python per comunicare con Elijah perché comprensivo del controllo sugli errori relativi alle chiamate ad Openstack++. Bisogna notare che l'utilizzo prevede la fornitura al comando di un file di autenticazione, di cui se ne mostra un esempio che riporta il caso di autenticazione con nome utente e password, utilizzata nel prototipo.

```
{
    "account": "admin",
    "password": "passwordforadmin***",
    "tenant": "demo",
    "server_addr": "192.168.1.***"
}
```

La verifica di una immagine all'interno di Glance viene eseguita tramite JCloud e restituisce al cliente una risposta di attuabilità o meno del servizio richiesto. Un responso positivo abilita la sintesi vera e propria della macchina virtuale di servizio. La richiesta di sintesi arriva al server Cloudlet che, prima di consegnarla al gestore Openstack++, verifica, sempre tramite JCloud, la presenza di una macchina virtuale, già creata in precedenza, utilizzabile per il lavoro richiesto. Se ce ne dovesse essere una in esecuzione, viene restituito al cliente mobile direttamente l'indirizzo e la porta su cui è possibile contattarla tramite l'applicazione dedicata al servizio. Se invece occorre sintetizzare una nuova macchina virtuale, il server Cloudlet provvederà a scaricare l'overlay dall'URL indicatogli

dal cliente e inizierà le operazioni di provisioning. Per il prototipo sono state previste tre diverse forme di provisioning.

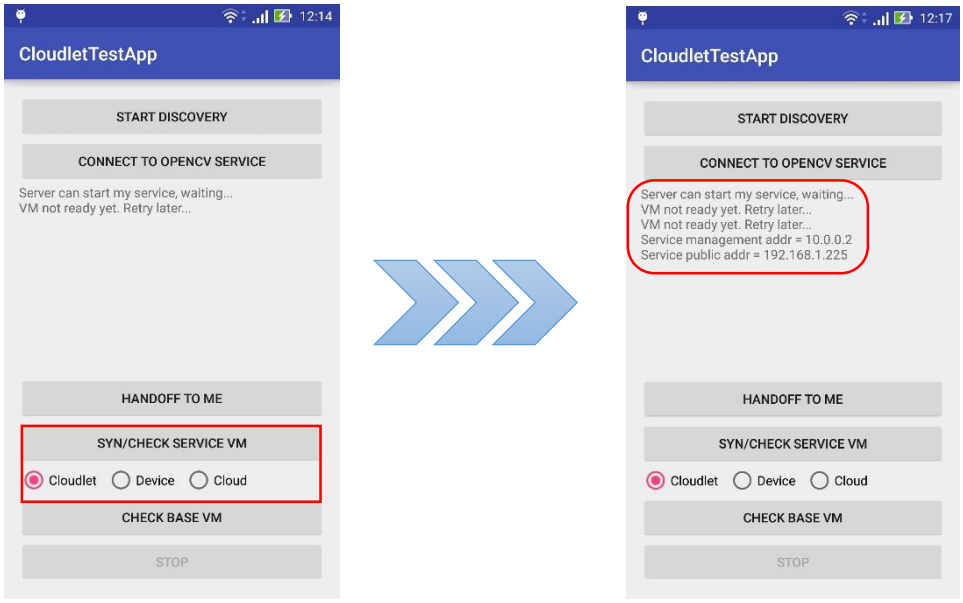

Figura 23 Attesa della sintesi di una macchina virtuale e consegna degli estremi di connessione.

La sintesi impiega il minor tempo possibile nel caso in cui l'overlay sia memorizzato localmente alla centrale Cloudlet. Il servizio potrebbe essere stato utilizzato in precedenza o potrebbe essere stato fornito alla centrale anticipatamente dagli amministratori di rete, prevedendo le richieste dei clienti che visitano quella stazione. In questo modo il Cloudlet non subisce rallentamenti dovuti alla rete perché evita di scaricare il file contenente l'overlay. Memorizzare l'overlay sul dispositivo è anch'essa una strategia abbastanza veloce per eseguire la sintesi, ma presuppone che sul dispositivo ci sia abbastanza spazio inutilizzato da permettere la presenza permanente degli overlay dei servizi che si vogliono utilizzare. Bisogna notare che per queste due prime modalità operative, non occorre la connessione

alla rete Internet esterna, ma basta che sia assicurata quella locale tra la centralina e il dispositivo richiedente. L'ultima modalità implementata per la sintesi, che prevede di scaricare l'overlay da un web server installato su una piattaforma Cloud, è l'unica per la quale occorre un collegamento funzionante alla rete esterna. I tempi di esecuzione dall'arrivo della richiesta alla effettiva sintesi della macchina si allungano, ma l'occupazione dello spazio di memoria è minimizzata. Mentre, per le prime due modalità operative, un miglioramento delle caratteristiche del comparto hardware della centralina e del collegamento radio con il dispositivo corrispondono direttamente ad una maggiore velocità di sintesi, questo non è assicurato con la strategia del download dell'overlay dal Cloud. L'attraversamento di reti multi hop e le relative latenze, fuori dal controllo di amministratori di rete e provider di servizi, potrebbero ritardare di molto le operazioni necessarie a rendere operativa una macchina virtuale. In caso si possegga fisicamente un overlay è bene utilizzare per la sintesi le prime due modalità proposte, e lasciare la terza, che coinvolge il Cloud, solamente a situazioni critiche in cui si è inseriti in ambienti in cui è impossibile o impraticabile memorizzare dei dati in modo sicuro. L'overlay utilizzato per il prototipo è della dimensione di circa 106 MB, avendo ridotto al minimo il comparto software installato. Si noti che memorizzare e scaricare l'overlay dal Cloud è una soluzione auspicabile in caso la dimensione del file diventi critica in conseguenza ai numerosi software installati nella macchina virtuale per eseguire il servizio. In tale situazione, è opportuno prevedere un collegamento robusto alla rete esterna.

5.3.2 Servizio OpenCV e Handoff

Quando la macchina virtuale di servizio è in stato *Ready* ed il server Openstack++ viene interrogato su di essa, quest'ultimo consegna al cliente tutte le informazioni necessarie per utilizzare il servizio OpenCV. Il dispositivo dispone di indirizzo e porta della macchina virtuale, ospite della centrale Cloudlet, ove è in esecuzione il server OpenCV. Tramite la pressione del pulsante che avvia l'applicazione cliente del servizio richiesto, le informazioni di connessione vengono passate al nuovo software. L'applicazione cliente di OpenCV proposta come esempio, per motivi di test e debug, permette di inserire manualmente un indirizzo su cui si è sicuri che ci sia un server in ascolto; ma, se avviata dal gestore Cloudlet, questi campi vengono automaticamente compilati. Il cliente del servizio OpenCV non deve fare altro che selezionare una foto dalla propria galleria, cliccando sul pulsante *Choose Pic*. Il server sulla macchina virtuale, riceve la foto, rileva i volti presenti al suo interno e ne memorizza una copia modificata con l'evidenziazione delle facce rilevate. Al cliente viene inviata la risposta contenente il numero di volti trovati; dopo di che il server si rimette in ascolto per una nuova richiesta.

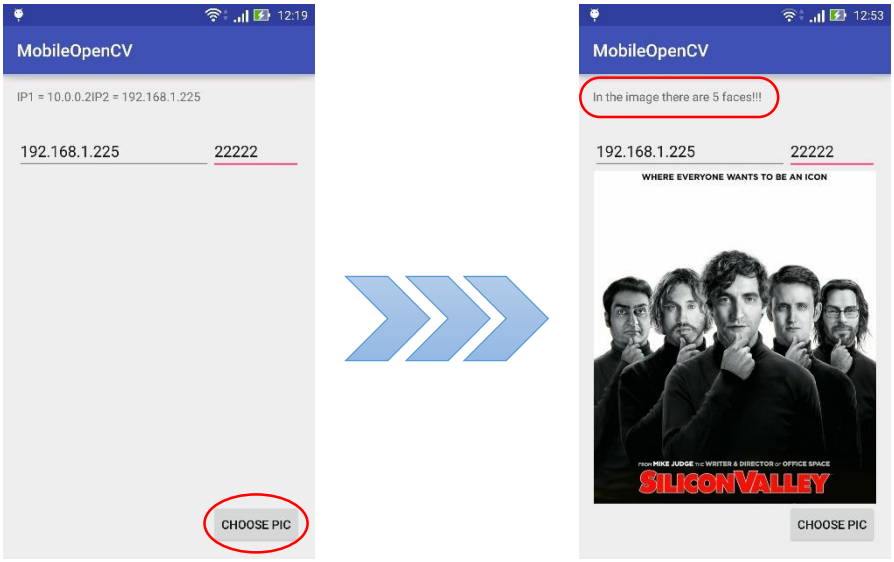

Figura 24 Funzionamento dell'applicazione OpenCV.

Per riprendere l'utilizzo del servizio OpenCV interrotto a causa della mobilità del cliente, occorre effettuare un handoff. Prerequisito è che il cliente abbia visitato un sito edge in precedenza, sul quale è ancora in esecuzione una macchina con il servizio OpenCV e del quale possiede l'indirizzo pubblico. Dopo aver eseguito un discovery e aver scoperto un nuovo server edge, occorre cliccare sul pulsante *Handoff to me*, il quale apre la lista dei siti edge visitati dal dispositivo richiedente. Selezionare quello di interesse contenente la macchina virtuale che si desidera trasferire e attendere il completamento dell'operazione. Lo schema seguente riassume l'implementazione del servizio di handoff, modificato secondo le necessità della soluzione integrata presentata, ovvero dando al cliente mobile la capacità di scatenare un trasferimento quando desidera e verso la centrale che ritiene opportuna.

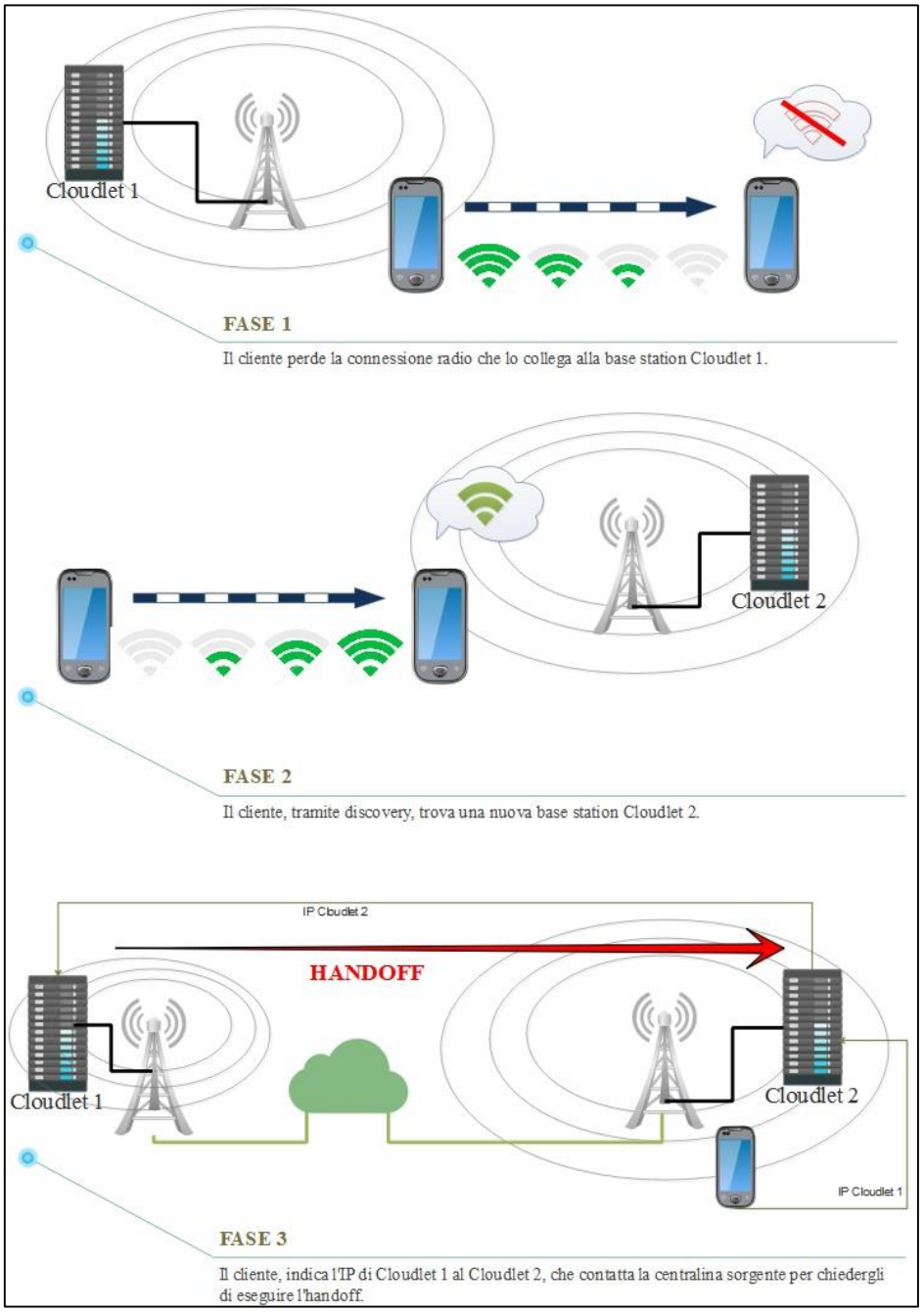

Figura 25 Esecuzione di un Handoff tra due centrali Cloudlet.

Si noti che i server Cloudlet sono sempre in ascolto di richieste di handoff sulla porta 22221; in risposta alle quali comandano ad Openstack++ l'handoff di una macchina virtuale locale verso la centralina richiedente. Allo stesso modo della

sintesi, possiamo verificare, con il pulsante adibito, lo stato della macchina virtuale che si sta trasferendo; e, qualora pronta all'esecuzione, riceverne indirizzo e porta per poter riprendere il servizio dal punto in cui era stato interrotto sulla centrale Cloudlet precedente.

Tramite la progettazione e l'implementazione di una piattaforma edge comprensiva di tutti e tre i livelli previsti dal protocollo, nel prossimo Capitolo verranno presentati i risultati ottenuti nelle misurazioni effettuate durante l'operatività del prototipo.

CAPITOLO 6: RISULTATI

In questo capitolo si analizzano le performance dell'implementazione proposta. Prima di riportarne i risultati, si descrivono le caratteristiche dell'hardware utilizzato. La trattazione prosegue con le misurazioni ottenute durante le varie prove eseguite con la piattaforma, ovvero la sintesi, l'handoff e la latenza; quest'ultima rapportata all'operatività del servizio su piattaforma Cloud.

6.1 Configurazione Hardware Utilizzata

Di seguito sono riportati tutti gli strumenti hardware utilizzati nel progetto e nello sviluppo dell'architettura del prototipo, utili a gestire l'infrastruttura e a eseguire i servizi. Per la parte applicativa in esecuzione sui dispositivi mobili con sistema operativo Android si sono sviluppate un'applicazione che gestisce i servizi Cloudlet conosciuti dal cliente e un'applicazione che implementa un servizio di esempio. Entrambe le applicazioni sono fornite come file apk di debug, non firmate elettronicamente, data l'attuale assenza sul territorio di centrali Cloudlet pronte ad eseguire il servizio, tranne quella provvista per i test. Per lo sviluppo delle applicazioni si è usato l'ambiente di sviluppo integrato standard per Android, ovvero Android Studio [67]. Per il deploy e l'esecuzione di tali applicazioni si è usato uno smartphone di fascia media con

sistema operativo Android versione 5.0 di marca Asus, precisamente il modello ZE551ML [68].

Per quanto riguarda il comparto hardware, si è palesata la necessità di provvedere all'installazione di due centrali Cloudlet, utili al test della procedura di trasferimento del servizio. Si sono utilizzate due macchine general-purpose, dalle discrete caratteristiche hardware, ma con meno capacità computazionali delle classiche macchine server Cloud. Il seguente elenco ne mostra le caratteristiche rilevanti ai fini del presente elaborato:

- Server Cloudlet uno:
  - CPU: AMD Phenom II X4 965 [69].
  - RAM: 8GB.
  - Hard Disk: 100 GB.
- Server Cloudlet due:
  - CPU: Intel Core i7 2640M [70].
  - RAM: 8GB.
  - Hard Disk: 500GB.

Entrambe le postazioni eseguono il sistema operativo Linux Ubuntu Desktop 14.04 LTS 64 bit, e su entrambe è installata la piattaforma Elijah che si occupa di svolgere le mansioni dettate dal protocollo Cloudlet. Per lo sviluppo sia dell'applicazione server che gestisce le richieste dei cliente, che di quella che realizza il servizio, installata all'interno di una macchina virtuale, si è optato per il linguaggio di programmazione Java. Totalmente compatibile con Android, Java è molto utilizzato, anche a livello aziendale. Come tool di sviluppo si è utilizzato il

progetto open-source Eclipse [71], che contiene un insieme di strumenti di sviluppo e un IDE grafico per Java. Come si è anticipato, le centrali, non prevedendo obbligatoriamente una posizione assegnata staticamente, devono pubblicizzare la loro natura Cloudlet, e di conseguenza il loro indirizzo, per poter entrare in comunicazione con i clienti. Il collegamento delle centrali alla rete esterna è fornito tramite un singolo access point wireless al quale le centrali sono collegate con cavo Ethernet CAT7.

6.2 Risultati Ottenuti

Questa sezione del documento mostra i risultati raggiunti nella fase di test a cui il prototipo è stato sottoposto. Per ogni prova si riportano le caratteristiche della rete su cui è stata effettuata, essendo queste molto influenti sui risultati e sulle latenze misurate. La rete è stata monitorata con il software gratuito open-source e multi piattaforma *iperf3*. Per le misurazioni effettuate per le operazioni proprie di Openstack++, si sono esaminati i file di log presenti nella cartella di default `/opt/stack/logs`. Invece, per confrontare le latenze di esecuzione del servizio creato, si è fatto in modo che il server mostrasse degli opportuni messaggi timestamp per ogni avvio e completamento delle operazioni di rete effettuate con i clienti mobili. La misurazione delle risorse utilizzate è stata eseguita monitorando il tool di Linux *System Monitor* in congiunzione all'utilizzo del free software *htop*, e facendo una media dei picchi

massimi di utilizzo per ciascuna risorsa e del numero di thread creati.

6.2.1 Sintesi

Il provisioning di una macchina può seguire diverse strategie. Come precedentemente mostrato, durante l'operatività della piattaforma, un cliente potrebbe richiedere la sintesi di una macchina non attualmente in esecuzione. La centralina riceve un indirizzo dal quale scaricare l'overlay utile ad eseguire il servizio richiesto. L'indirizzo, a seconda della tipologia di richiesta, potrebbe condurre ad uno storage locale, ad un Cloud fidato, usato come backup per i servizi, oppure ad una locazione Cloud remota sconosciuta, utilizzata dal produttore del servizio per condividere il proprio applicativo. In ogni caso l'overlay va prima scaricato poi decompresso e successivamente applicato ad una macchina vergine lanciata per questo scopo. Nel caso in cui l'overlay sia da scaricare da una sorgente Cloud, la componente di maggior peso nella somma delle tempistiche risulta lo scambio del file in rete. Essendo questa una statistica fuori dal controllo dell'amministrazione del prototipo presentato, i risultati seguenti, circa la sintesi di una macchina virtuale di servizio, sono relativi ad un overlay memorizzato localmente al server che ha il compito di avviare il servizio. L'indicazione di quale rete è stata utilizzata è in questa sezione inutile. Nella realtà operativa, le tempistiche riportate sarebbero da sommare a quelle

dell’eventuale scaricamento da remoto dell’overlay, che, si ricordi, è della dimensione di 106 Megabyte circa.

SINTESI	
Log	**Time [ms]**
Avvio dell’istanza	0
Richiesta di risorse accettata	288
Creazione immagine	2322
Inizio – Fine sintesi della VM	98619
VM Ripristinata	1540
VM Ripresa	94
Istanza in esecuzione	2278

Tabella 1 Media delle tempistiche riguardanti i passi di sintesi.

I valori mostrati si riferiscono ad una media delle numerose prove effettuate. Essi mostrano gli eventi fondamentali che avvengono durante la ripresa di una macchina virtuale con Elijah e la loro durata media espressa in millisecondi. Bisogna considerare che durante le varie prove, le tempistiche si discostano dai valori riportati con una varianza di circa il 2%. L’avvio dell’istanza, evento che segue la ricezione di una richiesta di creazione di una macchina virtuale, è pressoché istantaneo; viene considerato l’evento dal quale partono le misurazioni. Come prima cosa, anche durante la ripresa di una macchina in Openstack nella sua versione originale, avviene la richiesta delle risorse che si vogliono riservare alla macchina virtuale che verrà creata sul nodo Compute selezionato. Il nodo risponde in maniera affermativa in caso possegga abbastanza

risorse da soddisfare la richiesta. Alla creazione dell'immagine con le caratteristiche richieste, segue la sintesi che è il processo più lungo in quanto comprende i passi di decompressione e applicazione dell'overlay. Gli ultimi tre eventi espressi in tabella fanno parte del normale ciclo di vita delle macchine virtuali ed esprimono rispettivamente il ripristino, la ripresa e l'effettiva operatività della macchina. L'ultimo passo segna la piena funzionalità del servizio, ovvero il corretto set-up delle risorse come disco, memoria e collegamento in rete, quindi la possibilità di ricevere richieste dai clienti. Tramite questo test, oltre alle basse tempistiche di sintesi, si è verificato come gli overlay generati con la versione stand-alone di Elijah e con Openstack++ siano totalmente interscambiabili. Inoltre, come spiegato durante la trattazione, modellando opportunamente le caratteristiche delle macchine virtuali create dalle risorse presenti sull'edge della rete a partire dall'overlay, è possibile superare le incompatibilità hardware tra host molto diversi tra loro.

RISORSE – SINTESI			
Nome Processo	**N° Thread**	**MEM [MB]**	**CPU [%]**
cloudlet_vmnetfs	6	65	25
cloudlet_qemu-system-x86_64	4	240	3

Tabella 2 Processi e rispettive risorse usate durante la sintesi.

I dati relativi alle risorse utilizzate durante la sintesi sono mostrati nella tabella seguente; essi sono stati estrapolati monitorando i due principali processi in esecuzione durante la sintesi, ovvero `cloudlet_vmnetfs` e `cloudlet_qemu-system-x86_64`.

6.2.2 Handoff

Gli obiettivi dell'handoff tra centrali sull'edge della rete sono differenti da quelli della migrazione a caldo delle macchine virtuali, usata invece in ambienti Cloud. Sebbene si ottimizzi il tempo totale di trasferimento, anche il tempo in cui la macchina resta inutilizzabile viene mantenuto molto limitato. Ancora una volta, i limiti tecnologici sono di rilevante importanza in questa fase di test. In questo caso, utilizzando la rete locale per trasferire una macchina virtuale, le tempistiche di trasferimento sono trascurabili in confronto al tempo di elaborazione occorrente alle centraline per eseguire il processo di migrazione. In corrispondenza di un handoff si ha la creazione al volo di diversi overlay e il loro trasferimento. Il trasferimento è spezzato in chunk di dimensioni ridotte rispetto ad un overlay nella sua interezza; questo per favorire il trasferimento in rete e per abilitare l'esecuzione delle operazioni in parallelo. Per questo test, le azioni che danno il contributo temporale maggiore sono la compressione e la decompressione delle varie parti dell'overlay. Queste sono dipendenti direttamente dalle capacità computazionali disponibili e libere sulle centrali tra cui viene effettuato l'handoff. La tabella seguente riporta i tempi misurati durante l'esecuzione di un handoff di una macchina di servizio operativa, quindi con il suo soft state all'interno. Le misurazioni sono state tratte dal principale file di log utilizzato da Devstack, ovvero `n-cpu.log`.

La Banda di rete disponibile durante questo test è di 93.8 Megabit al secondo.

SENDER	
Log	**Time [ms]**
Inizio – Fine Handoff	165744
Flusso di Compressione	121641
Flusso di invio dei Blob	120018

Tabella 3 Media delle tempistiche riguardanti i passi di handoff alla sorgente.

RECEIVER	
Log	**Time [ms]**
Inizio – Fine ricezione Handoff	125516
Inizio – Fine Spawning	168370
Inizio – Fine Decompressione	120234

Tabella 4 Media delle tempistiche riguardanti i passi di handoff alla destinazione.

Le prime due tabelle descrivono gli eventi fondamentali che avvengono durante la procedura di handoff e le loro durate medie estratte da diverse prove. I trasferimenti sono stati eseguiti migrando macchine virtuali che hanno processato lo stesso numero di richieste prima dell'inizio dell'handoff. Le medie sono state ottenute misurando i valori su entrambe le centrali ed invertendo la sorgente e la destinazione per simulare la casualità nel comportamento degli utenti. I valori reali si discostano dalla media con un errore di circa 2%, in condizioni di sistema a regime. L'evento di inizio handoff, per il mittente, rappresenta la pervenuta richiesta di trasferimento; che viene inviata dalla

centrale di destinazione, futura ricevente della macchina virtuale. Mentre la fine dell'handoff coincide con la terminazione, da parte della centrale sorgente stessa, della macchina virtuale di servizio appena trasferita. Per la centrale destinataria, invece, l'handoff viene misurato dal momento in cui essa riceve la richiesta dal dispositivo mobile; mentre la fine coincide con la ripresa della macchina virtuale di servizio trasferita. Come si può vedere, le azioni che impiegano più tempo per essere portate a termine sono rappresentate dalla compressione e dalla trasmissione in rete dei vari blob. Gli eventi presentati nelle prime due tabelle, inoltre, avvengono in parallelo; soprattutto per quanto riguarda compressione-invio e ricezione-decompressione.

Risultati Notevoli HANDOFF	
Action	**Value**
Downtime della VM (ms)	1596
Blob trasferiti (n°)	85
Dati Trasferiti (Megabyte)	132,608
Tempo Totale (secondi)	172,523

Tabella 5 Misurazioni notevoli durante la procedura di handoff.

Per motivi di chiarezza si riporta una terza tabella che descrive altre importanti misurazioni durante l'handoff. Dato significativo è il ristretto tempo di interruzione, in cui la macchina non riceve richieste ed il servizio è sospeso. Si ha una media di circa un secondo e mezzo di interruzione. Dei dati trasferiti si può notare come essi abbiano una dimensione

maggiore dell'overlay originale generato per il servizio: si parla di una differenza di circa 30 Megabyte, 106 per l'overlay originale utile alla sintesi e 130 per una macchina che ha processato già delle richieste. Questo dislivello è dovuto sia al carico di dati che la macchina trasferita ha acquisito durante la sua operatività sulla prima centrale edge, rappresentato dalle foto, sia dalle modifiche che si sono operate a disco e memoria. Essendo molte delle azioni riportate eseguite in parallelo, si è voluta esplicitare la media della tempistica del processo nella sua interezza come ultimo valore degno di nota, il quale si aggira attorno ai 172 secondi, con una varianza del 2% circa.

Per misurare l'utilizzo delle risorse sono stati monitorati i processi `handoff_server_proc` sull'host destinazione, e `handoff_proc`, su quello sorgente. I valori mostrati nella tabella seguente ne riassumono la media.

RISORSE – HANDOFF			
Nome Processo	**Numero Thread**	**MEM [GB]**	**CPU [%]**
handoff_server_proc	6	1,04	25
handoff_proc	14	2,06	25

Tabella 6 Processi e rispettive risorse usate durante l'handoff.

6.2.3 Latenza

L’obiettivo principale della trattazione è abbattere la latenza del Cloud, per permettere l’utilizzo dei modelli da esso derivanti, in ambienti in cui ci sono esigenze tempistiche molto stringenti. I risultati seguenti ritraggono a confronto una stessa applicazione, eseguita su una risorsa presente sull’edge della rete, in rapporto alla sua esecuzione su un server remoto, in Cloud. Le misurazioni riguardano il servizio OpenCV creato per la piattaforma presentata. L’applicazione mobile OpenCV, nella prima parte della sua operatività, invia una foto al server con cui ha stabilito un contatto. Nella seconda parte riceve le informazioni contestuali circa il numero di volti presenti all’interno dell’immagine elaborata dal server. Essendo, in generale, l’invio di parecchi dati verso il Cloud l’area maggiormente colpita dalle alte latenze, si è misurato l’intervallo di tempo utile alla ricezione, da parte del server, dei dati da elaborare. Come si evince dal grafico, sono state fatte parecchie misurazioni con carichi diversi. Per ogni carico sono stati effettuati diversi invii e calcolata la media dei tempi, per tenere conto della variabilità del collegamento radio tra i dispositivi mobili e le risorse.

La Banda di rete disponibile durante questo test è di 53.3 Megabit al secondo nel caso della comunicazione con l’edge; mentre si abbassa fino a 2.44, durante la comunicazione con il Cloud.

Data Size	Cloud	Cloudlet
100 KB	153 ms	59 ms
200 KB	416 ms	82 ms
1 MB	3039 ms	306 ms
2,5 MB	8692 ms	441 ms
5 MB	19871 ms	986 ms
25 MB	93947 ms	3982 ms

Tabella 7 Confronto tempistiche di esecuzione del servizio sul Cloud e sul Cloudlet.

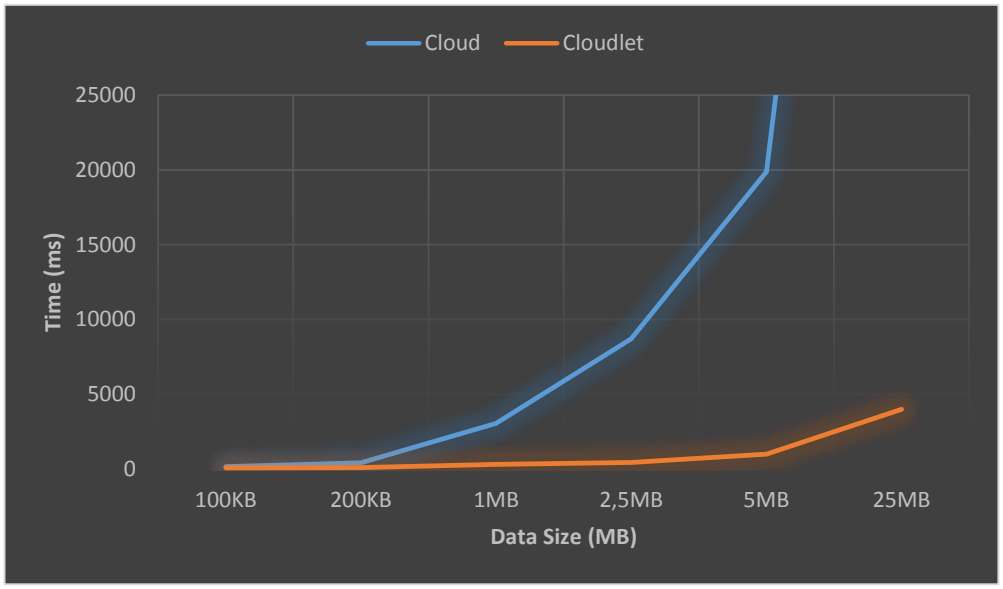

Figura 26 Grafico a linea che mette a confronto le prestazioni di Cloud e Cloudlet.

I risultati non tradiscono le aspettative, ovvero alle risorse sull'edge i dati sono resi disponibili prima che nel Cloud. Il divario delle tempistiche tra le due strategie si allarga con l'aumentare della dimensione del carico, questo è dovuto principalmente alle alte latenze nella comunicazione con la macchina remota Cloud.

CAPITOLO 7: CONCLUSIONI

La presente trattazione sottolinea l'importanza di una tecnologia complementare al Cloud e abilitante per quanto riguarda le moderne applicazioni dai requisiti altamente stringenti circa la latenza di comunicazione con le risorse richieste. Spostare la computazione sull'edge è la soluzione proposta per abbattere i ritardi che affliggono le reti multi-hop. La sintesi dinamica tramite overlay elimina gli oneri di gestione e di amministrazione del sistema e, sotto alcune ipotesi, permette di continuare ad eseguire i servizi richiesti anche in particolari casi in cui il collegamento tra la centrale all'edge che li ospita e il livello Cloud, è instabile o assente. I modelli implementativi che seguono il nuovo paradigma del Fog Computing, permettono non solo un rapido provisioning delle risorse, ma favoriscono la mobilità degli utenti mantenendo bassi i tempi di sospensione del servizio durante il trasferimento delle risorse. I clienti che hanno intenzione di usufruire dello stesso servizio, utilizzando il medesimo stato, possono richiedere la comunicazione tra due centrali per avvicinare il servizio alla centrale di accesso radio a cui si è attualmente collegati. Durante la trattazione si è presentato un prototipo che esaltasse le qualità del Fog Computing, utilizzando l'implementazione del modello Cloudlet fornita dalla Carnegie Mellon University. Si sono apportati miglioramenti al progetto Elijah stesso; ampliandone le potenzialità. Si è riusciti a trasferire macchine virtuali tra due centrali con caratteristiche intrinseche molto diverse, identificando, tra le loro proprietà hardware, un sottoinsieme totalmente simulabile, tramite virtualizzazione, da entrambe le

postazioni. In ambienti operativi dalle caratteristiche ostili, in cui la rete non è sempre affidabile, spesso costituita da canali radio, e i dispositivi compartecipanti hanno risorse limitate, superare questa mancanza è di fondamentale importanza. Ad esempio, potrebbe occorrere trasferire tutto il contenuto computazionale su centrali temporanee, dalle caratteristiche esigue, in caso di guasto o di connessione mancante. Oltre a risolvere l'annoso problema delle compatibilità hardware, si è anche realizzata una forma originale di trasferimento automatico delle macchine virtuali in funzione. Si è automatizzato il meccanismo di handoff originale, affidando la responsabilità della ripresa del servizio al dispositivo mobile. Quest'ultimo, in base alle esigenze di mobilità, può comandare il trasferimento di una macchina di servizio, utilizzata in precedenza presso un sito edge diverso, verso la centrale attuale di collegamento; per riprendere la comunicazione e lo stato antecedenti. Al prototipo si è aggiunta la possibilità di comunicazione con il livello Cloud, che si occupa di salvare i file utili al setup di una centrale sull'edge della rete. Questi miglioramenti hanno condotto a risultati soddisfacenti non solo per quanto riguarda la latenza misurata, ma anche dal punto di vista dei nuovi scenari abilitati. La latenza è risultata di molto inferiore a quella che occorre per eseguire le stesse operazioni appoggiandosi al Cloud. Inoltre, le tempistiche misurate per l'handoff risultano favorevoli a scenari in cui i partecipanti sono caratterizzati da un'elevata mobilità. Al Cloud potranno in futuro essere affidate mansioni di gestione, ad esempio politiche di failover e manutenzione automatica delle centrali. Di fondamentale importanza per l'evoluzione del modello a tre livelli è sicuramente il dispiegamento sul territorio

di molte centrali intermedie tra Cloud e dispositivi cliente. Con una copertura territoriale completa, si potrebbero realizzare, per la piattaforma presentata, altre strategie di handoff e sintesi che andrebbero a velocizzare l'approvvigionamento delle risorse, caratteristica essenziale per limitare temporalmente le interruzioni di servizio e le attese. Si potrebbero prevedere le prossime centrali sull'edge che l'utente visiterà in base alla direzione dei suoi spostamenti; o caricare anticipatamente le centrali di macchine operative, che svolgono determinati servizi, in base alla tipologia di dispositivi connessi.

L'Edge Computing è una tecnologia ancora agli albori, ma si auspica che possa arrivare in breve tempo ad un'adozione pervasiva da parte dei fornitori di infrastrutture di rete. La partecipazione aziendale allo sviluppo di uno standard comune dimostra come l'interesse per questa tecnologia sia vivo all'interno delle comunità scientifiche e industriali. L'abbattimento dei costi della componentistica hardware utile ad eseguire servizi sull'edge delle reti, permetterà ai Service Provider di commercializzare appartati in grado di soddisfare le esigenze dei clienti mobili, senza eccedere nelle loro capacità hardware, e quindi nel loro costo, come succede per le macchine Cloud. La qualità del servizio aggiunta che potrà essere proposta ai clienti e l'attitudine ad un impiego in settori come la sicurezza pubblica e l'intrattenimento, promettono un immediato ritorno sugli investimenti da parte di tutte le entità che si faranno portavoce di questa necessaria evoluzione del Cloud, la quale ne amplierà il ventaglio di utilizzo.

APPENDICE A: SISTEMI DI DISCOVERY

All'interno di reti ove ai dispositivi è data possibilità di mobilità, diventa essenziale una tecnologia che permetta di scoprire le risorse presenti nelle vicinanze senza un esplicito intervento umano. Nelle reti mobili, caratterizzate da molteplici partecipanti, è impensabile concepire operazioni di manutenzione per ogni ingresso e uscita, che vanificherebbero i vantaggi derivanti dall'utilizzo di un siffatto modello. I sistemi di discovery permettono ai dispositivi che entrano in una nuova rete di chiedere quali siano i servizi disponibili e riceverne una descrizione, senza l'intervento di un amministratore di sistema. Le risorse dal loro canto si rendono disponibili in rete registrando i propri servizi e fornendo eventualmente informazioni di utilizzo, come interfacce e/o indirizzo IP.

A.1 Zeroconf

Il protocollo *Zero Configuration Networking* (*zeroconf*) è diventato uno standard adottato a livello globale già da qualche anno [72]. Il modello è molto utilizzato in ambienti relativamente ristretti ove il lavoro di amministrazione è impossibile o impraticabile. Si pensi ad esempio a piccole reti domestiche formate da sensori ed elettrodomestici, oppure a uffici di modeste dimensione dove ci si collega con stampati, proiettori e altri dispositivi per cui non è previsto un intervento per ogni

connessione. Zeroconf è particolarmente adatto anche per piccole reti con clienti mobili, e per collegamenti estemporanei tra dispositivi wireless. Le reti complesse, soprattutto quelle che mettono in comunicazione server con indirizzi e nomi globalmente unici, richiedono configurazioni statiche e durature nel tempo; zeroconf, invece, si occupa della manutenzione in reti effimere, dove le informazioni di configurazione hanno significato locale e temporaneo. Non aggiungendo nessuna modifica alle interfacce esistenti, il protocollo si propone di migliorare la risoluzione dei nomi a livello applicativo e la configurazione delle interfacce IP a livello di rete. Di seguito sono riassunti i principali compiti che le implementazioni del protocollo Zeroconf devono eseguire.

Gli host collegati alla rete vengono configurati manualmente o usando protocolli come DHCP e PPP. Nelle reti estemporanee, formate da dispositivi altamente eterogenei, non sempre è possibile l'intervento umano oppure l'utilizzo di protocolli complessi che comunicano con server potenzialmente irraggiungibili. Sebbene la configurazione dinamica sia una proprietà insita nel protocollo IPv6, non lo è in quello IPv4. Zeroconf propone un metodo di *assegnazione dinamica di un indirizzo IP*, di qualunque versione, valido per la comunicazione locale [73], che permette ai dispositivi di determinare la sotto maschera della rete in cui sono inseriti, configurare le loro interfacce con un indirizzo unico, e risolvere autonomamente le collisioni.

Zeroconf ingloba un meccanismo di risoluzione dei nomi. In generale, l'importanza di questo strumento è fondamentale nelle reti perché permette di invocare un servizio con il nome piuttosto che specificarne l'indirizzo; il quale al contrario del precedente, potrebbe cambiare con una frequenza maggiore. Utilizzare un DNS per assegnare un nome univoco a un dispositivo di solito prevede sempre l'utilizzo di un server dedicato; anche perché il protocollo di risoluzione dei nomi è stato originariamente pensato per riferimenti a gruppi di dispositivi, per lo più server, facenti parte dello stesso ambito di gestione. Il protocollo Zerconf utilizza il multicast DNS per risolvere nomi locali tramite protocollo DNS, ma senza la necessità di dover contattare un server DNS. La richiesta di traduzione di un nome in un indirizzo, invece di essere inviata al server DNS viene inviato ad un noto indirizzo IP multicast, 224.0.0.251 per il protocollo IPv4 e FF02::FB per quello IPv6, sulla porta UDP 5353. Ogni dispositivo è in ascolto delle richieste; se l'interfaccia su cui arriva la richiesta è impostata con il nome da risolvere, il dispositivo che l'ha ricevuta risponde.

L'ultima area di rilevanza in cui si inserisce il protocollo è il discovery dinamico dei servizi, ovvero la possibilità per i clienti di richiedere e recuperare l'indirizzo di un servizio senza configurazioni statiche preesistenti né interventi di manutenzione. *DNS SRV Resource Record* è un meccanismo standardizzato da IETF, che permette ai clienti di scoprire servizi via protocollo DNS. La richiesta del cliente contiene il tipo del servizio, il protocollo di trasporto utilizzato e il dominio; mentre

la risposta consiste in una lista di host compatibili con le specifiche richieste.

A.2 Avahi e Network Service Discovery

Si è precedentemente introdotto l'utilizzo di due protocolli, usati dal prototipo, per la scoperta dinamica delle specifiche Cloudlet in un rete locale. L'algoritmo di advertisement e discovery, pensato per l'implementazione realizzata in questa tesi, prevede l'utilizzo rispettivamente di Avahi, su macchina server Linux e del Network Service Discovery di Google, su dispositivo mobile.

Avahi è un'implementazione del protocollo Zeroconf realizzata per il sistema operativo Linux sotto licenza free *GNU Lesser General Public License* (*LGPL*). Avahi include un sistema multicast DNS e permette alle applicazioni di pubblicizzare e scoprire servizi che sono eseguiti su host collegati alla rete locale. La distribuzione di Avahi, oltre a vari wrapper, che nascondono la complessità dei sistemi sottostanti, e alle librerie, reali implementazioni del servizio, esegue un processo demone principale. *Avahi-daemon* usa le librerie core per implementare uno stack multicast DNS accessibile tramite la pubblicazione di file XML nella cartella */etc/avahi/services*, che espongono ognuno un servizio e le sue caratteristiche. Esistono altre librerie che rendono accessibile il sistema di discovery come *nss-mdns* e

avahi-dnsconfd, che interagiscono con il DNS tramite linea di comando. La libreria *libavahi-client* riscrive le API dell'originale *libavahi-core* che realizza il servizio multicast DNS, per nasconderne l'originale complessità. *Avahi-browse*, *avahi-publish* e *avahi-resolve* sono un esempio dei comandi più comuni utilizzati da libavahi-client. La libreria *avahi-discover-standalone* è lo strumento adibito al reperimento dell'elenco di tutte le risorse e i servizi pubblicizzati sulla LAN locale. Quest'ultima contiene il proprio stack multicast DNS ed è fornita solo per scopi di test e debug dato che è sconsigliato eseguire più di un servizio DNS sullo stesso host. Ne esiste anche una implementazione python chiamata *avahi-discover*.

Sul dispositivo mobile invece è in azione il servizio di Google *Network Service Discovery* (*NSD*). Il servizio permette alle applicazioni di identificare i dispositivi all'interno della rete locale che svolgono i compiti richiesti. Incapsulando il comportamento del servizio in API prestabilite, come succede spesso con le tecnologie della piattaforma Android, evita agli sviluppatori di costruire manualmente il discovery. Il servizio dà anche la possibilità alle applicazioni di registrare le proprie competenze a favore di altri richiedenti.

```
public void registerService(int port) {
    NsdServiceInfo serviceInfo  = new NsdServiceInfo();
    serviceInfo.setServiceName("ServiceName");
    serviceInfo.setServiceType("_protocol._transportlayer");
    serviceInfo.setPort(port);
    ....
    mNsdManager = Context.getSystemService(Context.NSD_SERVICE);
    mNsdManager.registerService(serviceInfo,NsdManager.
PROTOCOL_DNS_SD,mRegistrationListener);
}
```

Listing 3 Snippet di codice per impostare i parametri di un servizio.

La classe `NsdServiceInfo` serve per impostare i parametri del servizio pubblicizzato. Il servizio risolve autonomamente le collisioni; dato che un nome deve essere unico all'interno di una stessa rete, se ce ne dovessero essere più di uno uguali, verrebbe aggiunto un suffisso identificativo. Oltre al nome va settato anche il tipo di servizio, rappresentato per convenzione come una stringa ove il protocollo applicativo è seguito dal tipo di protocollo usato a livello di trasporto, ad esempio per un servizio web in tcp si userà `_http._tcp`, mentre per una stampante collegata in rete si può utilizzare la dicitura `_ipp._tcp`. Infine del servizio pubblicizzato se ne specifica una porta di utilizzo. È possibile impostare questo valore, in maniera dinamica, selezionando la prima porta disponibile sul device, senza prevedere questo valore a compile time, perché potrebbe collidere con altri servizi che usano la stessa porta a run time. L'interfaccia `RegistrationListener`, da implementare, racchiude tutti gli eventi notevoli del protocollo di discovery.

```
public void initializeRegistrationListener() {
mRegistrationListener = new NsdManager.RegistrationListener() {
    @Override
    public void onServiceRegistered(NsdServiceInfo NsdServiceInfo) {
           mServiceName = NsdServiceInfo.getServiceName();
    }

    @Override
    public void onRegistrationFailed(NsdServiceInfo serviceInfo,
                                                        int errorCode)
    {....}

    @Override
    public void onServiceUnregistered(NsdServiceInfo arg0) {....}

    @Override
    public void onUnregistrationFailed(NsdServiceInfo serviceInfo,
                                                        int errorCode)
    {....}
};
}
```

Listing 4 Snippet dell'interfaccia RegistrationListener.

Come quasi tutte le funzioni di questo tipo in Android, il metodo `registerService()` è asincrono, di conseguenza tutte le azioni che l'applicazione deve compiere in seguito alla sua esecuzione devono essere inserite all'interno del metodo `onServiceRegistered()`. Come si nota dalla porzione di codice che riporta l'interfaccia `RegistrationListener`, all'interno del metodo `onServiceRegistered()` viene inserito il codice utile a recuperare il nome, che il sistema potrebbe aver cambiato, rispetto a quello predisposto dallo sviluppatore, in seguito alla collisione con un altro nome equivalente.

Per quanto riguarda la ricerca dei servizi locali, è attuata tramite l'impostazione di un listener, contenete le call back corrispondenti agli eventi rilevanti nel discovery, e la chiamata asincrona al metodo `discoverServices()`. Gli eventi del sistema, nel caso della ricerca di un servizio, notificano l'applicazione rispettivamente in corrispondenza dell'inizio del servizio di

discovery e del suo eventuale fallimento, del ritrovamento di un servizio e di quando esso non è più disponibile.

```
public void initializeDiscoveryListener() {
mDiscoveryListener = new NsdManager.DiscoveryListener() {

    @Override
    public void onDiscoveryStarted(String regType) {....}

    @Override
    public void onServiceFound(NsdServiceInfo service) {
           if (!service.getServiceType().equals(SERVICE_TYPE) &&
                        service.getServiceName().contains("ServiceName")) {
                 mNsdManager.resolveService(service, mResolveListener);
           }
    }

    @Override
    public void onServiceLost(NsdServiceInfo service) {....}

    @Override
    public void onDiscoveryStopped(String serviceType) {....}

    @Override
    public void onStartDiscoveryFailed(String serviceType,int errorCode){
           mNsdManager.stopServiceDiscovery(this);
    }

    @Override
    public void onStopDiscoveryFailed(String serviceType, int errorCode){
           mNsdManager.stopServiceDiscovery(this);
    }
};
}
```

Listing 5 Snippet dell'interfaccia DiscoveryListener.

Come si può notare dal codice della call back `onServiceFound()`, in conseguenza al ritrovamento di un servizio, se esso corrisponde ai parametri di nome e tipo previsti per quel servizio, occorre recuperarne le informazioni di connessione utilizzando la API `resolveService()`. Questo metodo accetta come parametri un servizio, oggetto della classe `NsdServiceInfo`, e un `ResolveListener`.

Il corretto discovery e risoluzione di un servizio, come mostrato all'interno del metodo `onServiceResolved()`,

permettono di recuperare tutte le informazioni utili al collegamento con il servizio cercato, ovvero indirizzo IP e porta.

```
public void initializeResolveListener() {
mResolveListener = new NsdManager.ResolveListener() {
    @Override
    public void onResolveFailed(NsdServiceInfo serviceInfo,
                                                int errorCode)
    {....}

    @Override
    public void onServiceResolved(NsdServiceInfo serviceInfo) {
           mService = serviceInfo;
           int port = mService.getPort();
           InetAddress host = mService.getHost();
    }
};
}
```

Listing 6 Snippet dell'interfaccia ResolveListener.

Infine, compito dello sviluppatore è mantenere la consistenza dei servizi attivi all'interno della rete locale. In corrispondenza delle call back riguardanti il ciclo di vita delle applicazioni, è buona norma effettuare la de-registrazione del servizio alla chiusura dell'applicazione che lo ha registrato, tramite le API `unregisterServicer()` e `stopServiceDiscovery()`.

APPENDICE B: OPENSTACK

Openstack è una piattaforma Cloud di tipo IaaS open-source, con un'architettura modulare che racchiude al proprio interno un insieme di strumenti specializzati in varie aree di gestione. È un progetto nato nel 2010 frutto della collaborazione tra due importanti aziende, Rackspace e NASA.

B.1 Componenti

L'elenco delle componenti riportato di seguito non è esaustivo dello strumento nella sua interezza, ma se ne vogliono riportare le parti fondamentali per una migliore comprensione delle loro funzionalità, utilizzate dal prototipo proposto e citate durante l'esposizione. Si tenga presente che è possibile installare tutte queste componenti sullo stesso host, ma solo durante la progettazione; al contrario, in produzione, si consiglia di disporre ogni elemento su un host diverso e di replicarne le caratteristiche per rendere il sistema robusto ad eventuali guasti o interruzioni di servizio.

- **Compute** (*Nova*): componente centrale, funge da controllore del sistema. È l'entità responsabile della gestione delle risorse disponibili e delle macchine virtuali. È compatibile con molte tecnologie hypervisor, ad esempio con KVM, VMWare, Xen e Linux Container LXC.

- **Block Storage** (*Cinder*): memorizza in maniera persistente i volumi id dati. Contribuisce ad avere un ottima strategia di memorizzazione degli snapshot dei dati all'interno dei volumi, che possono essere ripresi singolarmente e collegati a macchine virtuali in esecuzione. Questo componente risulta particolarmente adatti in scenari che richiedono performance elevate, come in caso di grandi database.
- **Object Storage** (*Swift*): è un sistema scalabile e ridondante per la memorizzazione dei dati. Gli oggetti, diversamente dai file, non sono organizzati gerarchicamente, ma sono tutti allo stesso livello per Openstack, che li caratterizza con un identificativo unico indipendente dalla loro locazione. Openstack si occupa della replicazione dei dati e della gestione del sistema in caso di fallimento dei nodi.
- **Networking** (*Neutron*): crea e organizza l'infrastruttura di rete per la piattaforma, gestisce indirizzi IP, reti, router e reagisce dinamicamente alla loro modifica. È possibile predisporre anche servizi di rete complessi come firewall e VPN. Fornisce agli amministratori la possibilità di poter scegliere esattamente su quali nodi eseguire specifici servizi.
- **Identity** (*Keystone*): fornisce autenticazione e autorizzazione ai componenti Openstack. Per l'autenticazione supporta diverse strategie tipo credenziali e token. Come back end di memorizzazione dei dati è possibile usare diverse tecnologie, anche in maniera concorrente, come MariaDB, LDAP e SQL. Le autorizzazioni sono gestite con i ruoli che determinano i permessi degli utenti.

- **Image** (*Glance*): implementa un registro per le immagini virtuali dei dischi. L'amministratore può registrare, condividere, duplicare immagini e creare snapshot, pronti per essere memorizzati tramite Swift o altri back-end predisposti. Glance supporta una grande varietà di formati per la memorizzazione del disco, come ad esempio aki/ami/ari (formati usati di default da Amazon Web Service), iso, raw, vhd (formato tipico di molti gestori di macchine virtuali come VMware, Xen, Microsoft, VirtualBox), qcow2 (formato di default per QEMU/KVM), oltre alla maggior parte dei formati per container.
- **Dashboard** (*Horizon*): un'interfaccia grafica web per gli amministratori che permette di creare e lanciare e gestire istanze e relative politiche di accesso, in alternativa all'utilizzo del sistema tramite l'invocazione delle API. Fornisce una visione dello stato delle componenti della piattaforma, delle risorse utilizzate e di quelle disponibili; il tutto tramite i tre pannelli di default Project, Admin, e Setting. Il design modulare di questo componente permette di mostrare anche altre schede sulla pagina principale, riguardanti i vari componenti aggiuntivi o le espansioni.

B.2 Devstack

Devstack fornisce strumenti per installare e manutenere un ambiente operativo Openstack. Tramite gli script forniti per l'installazione è possibile scaricare tutte le componenti della

piattaforma Cloud da repository git e impostarne una specifica versione. È spesso usato in ambiente di sviluppo grazie alla facilità di deploy, ma spesso costituisce la base per progetti Openstack più complessi, o aiuta semplicemente a costruire velocemente un ambiente di test per valutare i servizi prima che vengano inseriti in produzione. La configurazione di Devstack, che va ad influire direttamente su quella della piattaforma Openstack nella sua interezza, è modificabile tramite il file `local.conf`, situato di default nella cartella principale; il quale si collega alle proprietà di Openstack stesso.

BIBLIOGRAFIA

[1] J. C. R. Licklider, "DARPA/977.pdf," 25 April 1963. [Online]. Available: http://www.dod.mil/pubs/foi/Reading_Room/DARPA/977.pdf.

[2] L. Barry, C. Vinton, C. David, K. Robert, K. Leonard, L. Daniel, P. Jon, R. Larry and S. Wolff, "Brief History of the Internet," 15 October 2012. [Online]. Available: http://www.internetsociety.org/sites/default/files/Brief_History_of_the_Internet.pdf.

[3] J. C. R. Licklider and R. W. Taylor, "The Computer as a Communication Device," in *Science and Technology*, 1968.

[4] S. University, "The A.M. Turing Award," Stanford University, Stanford, 1971.

[5] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger and D. Leaf, "NIST Cloud Computing Reference Architecture," National Institute of Standards and Technology, 2011.

[6] A. S. Tanenbaum and M. V. Steen, Distributed Systems: Principles and Paradigms, Prentice Hall, Inc, 2007.

[7] D. Barry, Web Services, Service-Oriented Architectures, and Cloud Computing, Morgan Kaufmann, 2013.

[8] F. Polash, A. Abuhussein and S. Shiva, "A Survey of Cloud Computing Taxonomies: Rationale and Overview," The University of Memphis, Memphis, 2014.

[9] L. Garber, "In Brief," IEEE Computer Society, 2014.

[10] A. Gajbhiye and K. M. P. Shrivastva, "Cloud Computing: Need, Enabling Technology, Architecture, Advantages and Challenges," Department of Computer Technology and application, NITTTR, Bhopal, 2014.

[11] J. L. V. Barbosa, "Ubiquitous Computing Applications and Research Opportunities," University of Vale do Rio dos Sinos, São Leopoldo, Brazil, 2015.

[12] R. v. d. Meulen, "Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015," Gartner, 10 November 2015. [Online]. Available: http://www.gartner.com/newsroom/id/3165317.

[13] F. B. Shaikh and S. Haider, "Security Threats in Cloud Computing," Internet Technology and Secured Transactions (ICITST), 2011 International Conference, Abu Dhabi, 2011.

[14] A. Ghafarian, "Foreniscs Analysis of Cloud Computing Services," Science and Information Conference (SAI), Dahlonega, GA, USA , 2015.

[15] A. Papageorgiou, B. Cheng and E. Kovacs, "Real-Time Data Reduction at the Network Edge of Internet-of-Things Systems," NEC Laboratories Europe, Heidelberg, Germany, 2015.

[16] K. Ha, P. Pilla, G. Lewis, S. Simanta, S. Clinch, N. Davies and M. Satyanarayanan, "The Impact of Mobile Multimedia Applications on Data Center Consolidation," Carnegie Mellon University; Intel Labs; Lancaster University, 2013.

[17] MICROSOFT, "Outatime: Using Speculation to Enable Low-Latency Continuous Interaction for Cloud Gaming," Microsoft Research, 21 August 2014.

[Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/outatime_techreport2014.pdf.

[18] Y. Lin and H. Shen, "CloudFog: Towards High Quality of Experience in Cloud Gaming," Department of Electrical and Computer Engineering, Clemson, South Carolina, 2015.

[19] Y. Lin and H. Shen, "Leveraging Fog to Extend Cloud Gaming for Thin-Client MMOG with High Quality of Experience," Clemson University, Clemson, South Carolina, USA, 2015.

[20] L. Oculus VR, "oculus," Oculus VR, 2016. [Online]. Available: https://www.oculus.com/en-us/.

[21] M. Abrash, "Latency – the sine qua non of AR and VR," Valve Corporation, 29 December 2012. [Online]. Available: http://blogs.valvesoftware.com/abrash/latency-the-sine-qua-non-of-ar-and-vr/.

[22] T. R. Agus, C. Suied, S. J. Thorpe and D. Pressnitzer, "Characteristics of human voice processing," Proceedings of 2010 IEEE International Symposium on Circuits and Systems, 2010.

[23] Y. Gao, W. Hu, K. Ha, B. Amos, P. Pillai and M. Satyanarayanan, "Are Cloudlets Necessary?," School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 2015.

[24] H. Chang, A. Hari, S. Mukherjee and T. Lakshman, "Bringing the Cloud to the Edge," Bell Labs, USA, Murray Hill, NJ, 2014.

[25] F. Bonomi, R. Milito, J. Zhu and S. Addepalli, "Fog Computing and Its Role in the Internet of

Things," Cisco Systems Inc., San Jose, CA, USA, 2012.

[26] L. M. Vaquero, "Finding your Way in the Fog: Towards a Comprehensive Definition of Fog Computing," Hewlett-Packard Labs, Bristol, United Kingdom, 2014.

[27] Cisco, "Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are," Cisco Systems, 2015. [Online]. Available: http://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf.

[28] ETSI, "About," European Telecommunications Standards Institute, [Online]. Available: http://www.etsi.org/about. [Accessed 2016].

[29] ETSI, "Mobile-Edge Computing," European Telecommunications Standards Institute, 2014.

[30] ETSI, "Mobile-Edge Computing (MEC); Service Scenarios," ETSI Industry Specification Group, 2015.

[31] G. Lewis, S. Echeverría, S. Simanta, B. Bradshaw and J. Root, "Tactical Cloudlets: Moving Cloud Computing to the Edge," Carnegie Mellon Software Engineering Institute, Pittsburgh, PA, USA , 2014.

[32] S. Echeverr´ıa, J. Root, B. Bradshaw and G. Lewis, "On-Demand VM Provisioning for Cloudlet-Based Cyber-Foraging in Resource-Constrained Environments," Carnegie Mellon Software Engineering Institute, Pittsburgh PA, USA, 2014.

[33] K. Ha, Y. Abe, Z. Chen, W. Hu, B. Amos, P. Pillai and M. Satyanarayanan, "Adaptive VM Handoff Across Cloudlets," Carnegie Mellon University School of Computer Science, Pittsburgh, PA, USA, 2015.

[34] CMU, "Elijah Cloudlet-based Mobile Computing," CMU, [Online]. Available: http://elijah.cs.cmu.edu/. [Accessed 2016].

[35] K. Ha and M. Satyanarayanan, "OpenStack++ for Cloudlet Deployment," Carnegie Mellon University School of Computer Science, Pittsburgh, PA, USA, 2015.

[36] QEMU, "Features/CPUModels," QEMU, [Online]. Available: http://wiki.qemu.org/Features/CPUModels. [Accessed 2016].

[37] CMU, "Press Mention of Gabriel and Wearable Cognitive Assistance," CMU, 2016. [Online]. Available: http://elijah.cs.cmu.edu/press-gabriel.html.

[38] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai and M. Satyanarayanan, "Towards Wearable Cognitive Assistance," Carnegie Mellon University; Intel Labs, 2014.

[39] P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha and M. Satyanarayanan, "Scalable Crowd-Sourcing of Video from Mobile Devices," 2013.

[40] 5G-PPP, "The 5G Infrastructure Public Private Partnership: the next generation of communication networks and services.," 5G Infrastructure Public Private Partnership, 2015.

[41] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher and V. Young, "Mobile Edge Computing A key technology towards 5G," European Telecommunications Standards Institute, Sophia Antipolis, France, 2015.

[42] ETSI, "Executive Briefing – Mobile Edge Computing (MEC) Initiative," European Telecommunications Standards Institute, 2014.

[43] MEC, "Mobile Edge Computing Congress," [Online]. Available: https://meccongress.com/. [Accessed 2016].

[44] INTEL, "Saguna and Intel – Using Mobile Edge Computing to Improve Mobile Network Performance and Profitability," Intel Labs, 2016.

[45] N. Zhong, "Mobile Edge Computing: Unleashing the value chain," HUAWEI TECHNOLOGIES CO., LTD., 2015.

[46] CISCO, "What is IoT?," Cisco Systems, 2016. [Online]. Available: https://developer.cisco.com/site/iox/documents/developer-guide/?ref=fog.

[47] CISCO, "IOx Sandbox," Cisco Systems, [Online]. Available: https://communities.cisco.com/community/developer/networking/internet-of-things/iox/sandbox. [Accessed 2016].

[48] CISCO, "IOx Fog Node Software-1.0.0," Cisco Systems, 2016. [Online]. Available: https://software.cisco.com/download/release.html?i=!y&mdfid=286259484&softwareid=286306224&release=1.0.0&os=.

[49] WDL, "ADLINK SETO-1000 EXTREME OUTDOOR SERVER DUAL INTEL XEON E5-2448L," Wdl Systems, [Online]. Available: http://www.wdlsystems.com/Box-PC/Extreme-Outdoor-Server/ADLINK-SETO-1000-Extreme-Outdoor-Server-Dual-Intel-Xeon-E5-2448L.html. [Accessed 2016].

[50] DUCATI, "Ducati accelerates its racing ambitions with EMC," Ducati, 4 September 2014. [Online]. Available: http://www.ducati.com/news/ducati_accelerates_its_racing_ambitions_with_emc_/2014/09/04/3351/index.do.

[51] EMC, "Secure, Enterprise File Sync and Share with EMC Syncplicity Utilizing EMC Isilion, EMC Atmos, and EMC Vnx," EMC Corporation, 2013.

[52] NOKIA, "Mobile Edge Computing," Nokia Networks, [Online]. Available: http://networks.nokia.com/portfolio/solutions/mobile-edge-computing#tab-highlights. [Accessed 2016].

[53] IBM, "Smarter wireless networks Add intelligence to the mobile network edge," IBM Corporation , 2013.

[54] E. Demaria, A. Pinnola and N. Santinelli, "La Virtualizzazione di Rete: lo Standard NFV," Telecom Italia, 2015.

[55] Opensource.org, "The Open Source Definition," Opensource.org, [Online]. Available: https://opensource.org/osd. [Accessed 2016].

[56] Opensource.org, "Apache License, Version 2.0," Opensource.org, [Online]. Available: https://opensource.org/licenses/Apache-2.0. [Accessed 2016].

[57] Opensource.org, "GNU General Public License, version 2 (GPL-2.0)," Opensource.org, 1991. [Online]. Available: https://opensource.org/licenses/GPL-2.0.

[58] GOOGLE, "Developers," Google Inc., [Online]. Available: https://developer.android.com/index.html. [Accessed 2016].

[59] IDC, "Smartphone OS Market Share, 2015 Q2," International Data Corporation, [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp. [Accessed 2016].

[60] OPENCV, "About," Itseez, [Online]. Available: http://opencv.org/about.html. [Accessed 2016].

[61] K. Zafar, "kWS - Android Web Server," [Online]. Available: https://play.google.com/store/apps/details?id=org.xeustechnologies.android.kws&hl=it. [Accessed 2016].

[62] CMU, "elijah-openstack," Carnegie Mellon University, [Online]. Available: https://github.com/cmusatyalab/elijah-openstack. [Accessed 2016].

[63] CMU, "elijah-provisioning," Carnegie Mellon University, [Online]. Available: https://github.com/cmusatyalab/elijah-provisioning. [Accessed 2016].

[64] OPENSTACK, "All-In-One Single Machine," OpenStack Foundation, [Online]. Available: http://docs.openstack.org/developer/devstack/guides/single-machine.html. [Accessed 2016].

[65] JCLOUD, "Compute Guide," Apache Software Foundation, [Online]. Available: https://jclouds.apache.org/start/compute/. [Accessed 2016].

[66] OPENSTACK, "Software Development Kits," OpenStack Foundation, [Online]. Available: https://wiki.openstack.org/wiki/SDKs#Java. [Accessed 2016].

[67] GOOGLE, "Android Studio The Official IDE for Android," Google Inc., [Online]. Available: https://developer.android.com/studio/index.html. [Accessed 2016].

[68] ASUS, "ZenFone 2 (ZE551ML)," Asus, [Online]. Available: https://www.asus.com/us/Phone/ZenFone_2_ZE551ML/specifications/. [Accessed 2016].

[69] CPUWORLD, "AMD Phenom II X4 965 (125W, BE) specifications," CPUWORLD, [Online]. Available: http://www.cpu-world.com/CPUs/K10/AMD-Phenom%20II%20X4%20965%20Black%20Edition%20-%20HDZ965FBK4DGM%20(HDZ965FBGMBOX).html. [Accessed 2016].

[70] CPUWORLD, "Intel Core i7-2640M (BGA) specifications," CPUWORLD, [Online]. Available: http://www.cpu-world.com/CPUs/Core_i7/Intel-Core%20i7-2640M%20Mobile%20processor%20-%20AV8062700839107.html. [Accessed 2016].

[71] ECLIPSE, "About the Eclipse Foundation," Eclipse Foundation, [Online]. Available: https://eclipse.org/org/#about. [Accessed 2016].

[72] IETF, "Zero Configuration Networking (Zeroconf)," IETF Zeroconf Working Group , [Online]. Available: http://www.zeroconf.org/. [Accessed 2016].

[73] S. Cheshire, B. Aboba and E. Guttman, "Dynamic Configuration of IPv4 Link-Local Addresses," IETF Working Groups , [Online]. Available: https://tools.ietf.org/pdf/rfc3927.pdf. [Accessed 2016].

[74] Cisco, "IOx and Fog Applications," Cisco Systems, [Online]. Available: http://www.cisco.com/c/en/us/solutions/internet-of-things/iot-fog-applications.html. [Accessed 2016].

[75] M. Satyanarayanan, P. Bahl, R. Cáceres and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," Carnegie Mellon University, Pittsburgh, PA, USA, 2009.