Università degli Studi dell'Aquila
Facoltà di Ingegneria

Tesi di Laurea Specialistica in Ingegneria Informatica e Automatica

Realizzazione di un prototipo della versione Cloud SaaS della suite IBM BigFix: Automazione del deployment e del testing

Relatore interno:
Prof. Serafino Cicerone

Laureando:
Beniamino Negrini

Correlatore:
Dott. Marco Secchi

Relatore esterno:
Dott. Bernardo Pastorelli

Anno Accademico 2016/2017

Dedica a piè pagina

Indice

1 Introduzione **7**

2 IBM BigFix **8**

2.1 BigFix 8

2.1.1 Architettura 8

2.1.2 BigFix Platform 11

2.1.3 BigFix Applications 11

2.1.4 Fixlets 12

2.2 IBM 13

2.3 Saas Exploration Project 13

2.3.1 Il framework SCRUM 14

2.3.2 Sistemi di controllo di versione 14

2.3.3 RTC 15

3 Il Cloud e le sue sfaccettature **17**

3.1 Cloud Computing 17

3.1.1 Vantaggi del Cloud Computing 18

3.2 Tipologie di servizi Cloud 19

3.2.1 IaaS, Infrastructure as a Service 20

3.2.2 PaaS, Platform as a Service 20

3.2.3 SaaS, Software as a Service 21

3.3 SOA, Service Oriented Architecture 26

3.4 Il software On Premise 26

4 SaaS e i suoi requisiti 28
4.1 Availability 28
4.2 Reliability 29
4.3 Dependability 29
4.3.1 Availability e Reliability nel contesto Cloud 30
4.4 Scalability 30
4.5 Monitoring 31

5 SaaS, le tecnologie che ne consentono la realizzazione 33
5.1 Microservizi 33
5.1.1 Il modello monolitico a layer 34
5.1.2 Confronto tra l'architettura a microservizi e il modello monolitico 35
5.1.3 Vantaggi e svantaggi di un'architettura a microservizi 35
5.2 Containers 38
5.2.1 I containers e le Macchine Virtuali 39
5.2.2 Vantaggi dei container 40
5.2.3 I container e i microservizi 41
5.2.4 Docker 41
5.2.5 Un'esuberante 45
5.3 Multitenancy 48
5.4 Monitoring Tools 49
5.4.1 Prometheus 49
5.4.2 Grafana 50
5.5 BlueMix Services 50

6 IBM BigFix on SaaS, la progettazione 52
6.1 Interaction Design 52
6.1.1 Design Thinking 53
6.1.2 BigFix SaaS Interaction Design 54
6.2 Requisiti Non Funzionali 59
6.2.1 Dependability 59

6.2.2 Scalabilità 61
6.2.3 Monitoring 62
6.3 Gap con il prodotto on premise 62
6.4 Scelta dei tool e dei servizi da utilizzare 63
6.5 Definizione Architetturale 63
6.5.1 Viste architetturali 64
6.6 Definizione dei processi di gestione 66
6.6.1 Novità rispetto al prodotto già esistente 67
6.6.2 Disaster Recovering 67

7 IBM BigFix on SaaS, l'implementazione del prototipo 68
7.1 Relazione tra il prodotto BigFix e i container 69
7.2 Generazione delle immagini 69
7.3 Automazione del Deployment 69
7.3.1 Releases vs DevOps vs Continuous Delivery 69
7.3.2 Bash Scripting 69
7.3.3 Jenkins 69
7.3.4 UrbanCode 69
7.3.5 Ansible 69
7.3.6 Scenario di onboarding di un nuovo cliente 69
7.4 Automazione del Testing 69
7.4.1 Functional Test 69
7.4.2 Security Test 69
7.4.3 Performance Test 69
7.4.4 Penetration Test 69
7.4.5 Rielaborazione degli output del testing 69

8 Conclusioni 70
8.1 Sviluppi futuri 70
8.2 Considerazioni 70

9 Ringraziamenti 71

A Tecnologie Utilizzate(template di prova - ANCORA DA SCRIVERE) 72

A.1 Linguaggi di programmazione 73

A.2 Linguaggi di Markup e Stile 73

A.3 Framework 73

A.4 Ambiente di Sviluppo 73

A.4.1 Eclipse 73

A.4.2 Piattaforma Web 74

A.4.3 Browser Testing 74

Capitolo 1

Introduzione

(prima bozza)

E' sempre più evidente che il cloud computing è il futuro del software. La rivoluzione consta nella distribuzione dei servizi di calcolo e nella virtualizzazione delle risorse, dando così all'utente la sensazione di un utilizzo centralizzato. Tutto ciò si è reso realizzabile dal momento in cui l'accesso alla rete è divenuto possibile da sempre più dispositivi e con velocità di connessione sempre maggiore.

La tematica del cloud computing è stata centrale nel mio lavoro di tesi presso l'azienda IBM (International Business Machines Corporation) nella sua sede di Roma. Ho partecipato attivamente alla realizzazione di un prototipo software, ossia la versione SaaS della suite IBM BigFix. BigFix è una suite di prodotti dedicati alle aziende che risolvono problematiche di Endpoint Security e di compliance di dispositivi a determinate politiche aziendali. Tramite questi prodotti si ottiene pieno controllo su tutti i dispositivi aziendali. Si possono ad esempio rilevare eventuali attacchi o si possono distribuire aggiornamenti e patch.

La sfida da me raccolta è quindi proprio quella di portare tutto questo arsenale di strumenti nella leggerezza del cloud. Rendendolo disponibile, nel giro di pochi minuti, anche a chi è sempre stato intimorito dalla difficoltà di installazione di uno strumento così potente, ma allo stesso tempo complesso.

Capitolo 2

IBM BigFix

2.1 BigFix

I prodotti della suite IBM BigFix consentono di monitorare e gestire in tempo reale un elevato numero di dispositivi connessi (fino a 250.000). Questi possono essere sia fisici che virtuali, come ad esempio server, desktop, notebook, dispositivi mobili, tablet, POS, ATM e chioschi self-service. Gli utenti principali di questi prodotti sono gli amministratori di sistema. Tramite le applicazioni BigFix possono avere il pieno controllo sugli endpoint. Possono ad esempio distribuire software, applicare delle patch, effettuare il deploy di sistemi operativi, proteggere da attacchi di rete e molto altro.

2.1.1 Architettura

L'architettura di BigFix può essere, per sua natura, molto articolata, nel caso si debba gestire un numero elevato ed eterogeneo di dispositivi. Essa si basa sul consolidato pattern stilistico client/server, ma con una struttura leggermente variata, prevedendo l'inserimento di un ulteriore layer frapposto tra client e server, i Relay, i quali sono fondamentali per bilanciare il carico.

Ma partiamo subito con un esempio per avere un ponto di riferimento. Come possiamo notare, l'elemento centrale è il server di BigFix, il quale ha lo scopo di

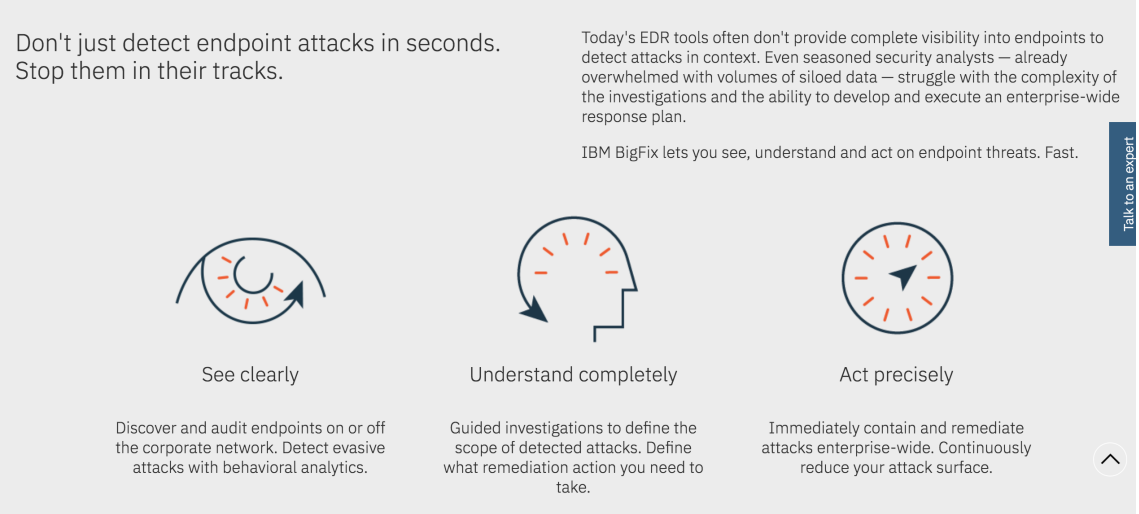

Figura 2.1: Breve header descrittivo di BigFix dal sito web di IBM Security

raccogliere dei particolari messaggi chiamati Fixlet. Questi messaggi dovono poi venire inoltrati ai Relay. E' competenza dei Relay interagire con i singoli client e assicurarsi l'esecuzione delle Fixlet. Le Fixlet, infatti, altro non sono che delle azioni che devono essere necessariamente compiute dai client. Una sorta di catalogo" delle Fixlet più utili può essere trovato sul Content server che è accessibile via internet da gli utilizzatori di BigFix. In tutto ciò la finestra dalla quale opera l'amministratore è ovviamente la Console, la quale monitora tutto il flusso di lavoro di BigFix. Andiamo ora ad analizzare le singole componenti dell'architettura.

Servers Il server coordina tutto il flusso di informazioni e si preoccupa di salvare le informazioni sul database. Al tempo stesso però, lascia agli Agent il compito di effettuare analisi ed eseguire azioni specifiche. Ciò consente di liberare il server da un pesantissima computazione. Per questo motivo il server stesso può gestire un altissimo numero di client.

Relays I Relay si comportano come una cache tra i client e il server e sono di numero variabile in base al numero di client. Aiutano il server a gestire i dispositivi anche se funzionalmente non sono altro che client che sono stati promossi a Relay, aggiungendo a loro dei servizi. A questo punto i client non si interfacceranno mai

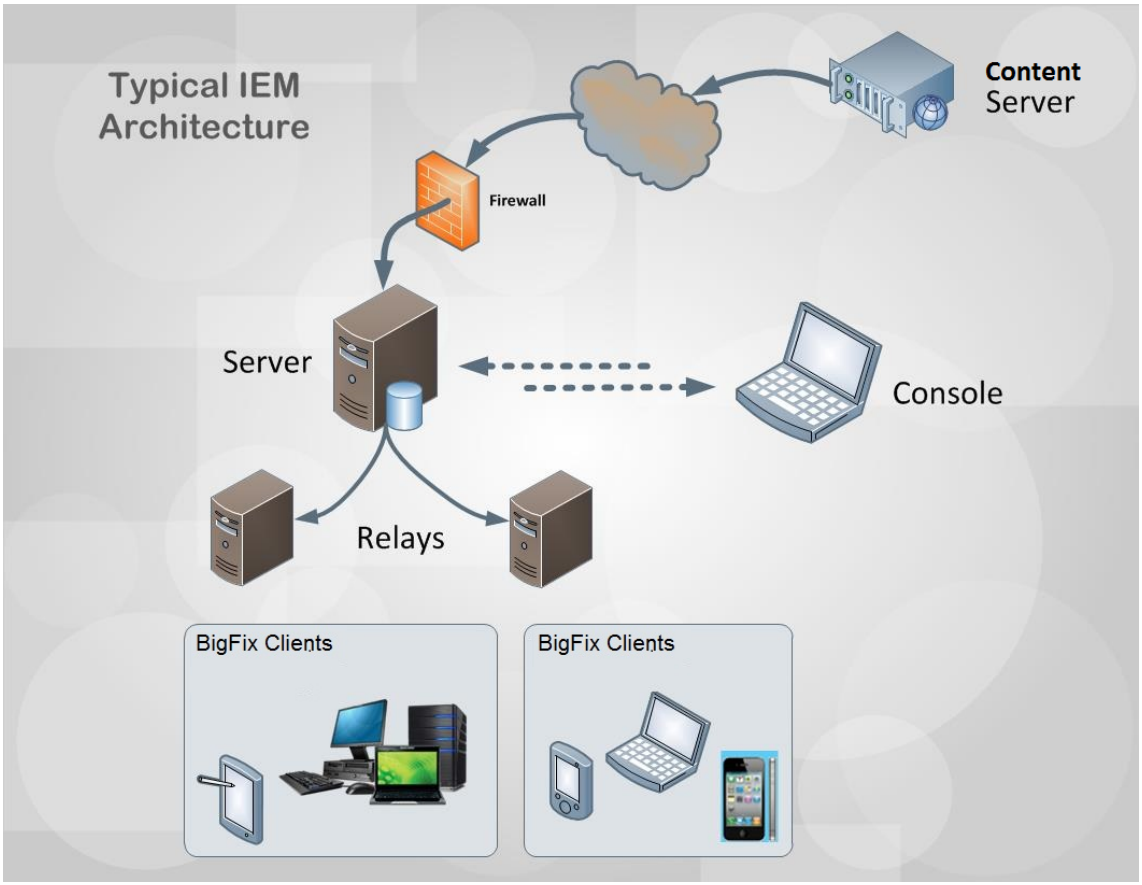

Figura 2.2: Un'architettura BigFix di esempio

con il server, alleggerendone così notevolmente il workload. Possono, ad esempio, più client richiedere un download al Relay, il quale effettuerà un'unica richiesta al server

Agents Un Agent è installato su ogni client facente parte dell'architettura di BigFix. Essi hanno il compito di raccogliere le Fixlet, tramite le quali sono in grado di compiere tutte le azioni necessarie. Un Agent fa dei continui check per confrontare lo stato del dispositivo con le policy stabilite. Appena scopre che il dispositivo è fuori dalla compliance, informa il server ed agisce subito per porre rimedio e, al termine dell'attività, l'Agent informa nuovamente il server sull'esito dell'operazione.

Web Reports I Web Reports costituiscono il componente che consente ad utenti autorizzati di monitorare tutti i dispositivi di BigFix. Si può, in questo modo, tenere traccia di vulnerabilità, azioni richieste e molto altro.

Consoles La Console permette agli amministratori di interagire con tutti i client dell'ambiente BigFix. Gli utenti possono così distribuire velocemente patch e configurazioni.

Content Server Il Content server è una sorta di repository. Contiene Fixlet a bassa personalizzazione che fanno fronte a esigenze più o meno comuni a tutti gli utenti BigFix. Possono essere prelevate ed utilizzati per i propri fini.

BigFix, da un punto di vista logico, si suddivide in due grandi macro-componenti, la Platform e le Applications. La prima svolge la funzione di layer sopra la quale vengono sviluppate tutte le funzionalità dello strato di applications. Questa suddivisione consente una chiara suddivisione delle competenze da parte di progettisti, sviluppatori, tester e assistenti dei clienti. Il team della Platform si concentra quindi nel fornire una solida infrastruttura al team delle Applications, il quale svilupperà i singoli strumenti al servizio dell'utente.

2.1.2 BigFix Platform

La Platform è una tecnologia multi-layer scritta in linguaggio C++ che agisce come colonna portante di tutta l'infrastruttura di BigFix. Essa svolge infatti funzioni fondamentali, spesso utilizzate anche da altre applicazioni dei layer superiori. Le attività della Platform sono fondamentali per la sussistenza dell'architettura che abbiamo descritto poco fa.

2.1.3 BigFix Applications

Tutti i prodotti applicativi che fanno parte di questo componente consentono di gestire in maniera semplice le operazioni inerenti alla security. A differenza della Platform, sono implementate in linguaggio Java o JavaScript ed hanno funzio-

nalità atomiche tra di loro. Sono l'interfaccia principale con il quale interagisce l'amministratore aziendale.

BigFix Lifecycle Questa è l'applicazione che l'amministratore utilizza per gestire il ciclo di vita degli endpoint fisici. Ha una visibilità completa su di essi e pone rimedi immediati. Tra le funzioni principali ci sono quelle di power management, software distribution e OS deployment.

BigFix Patch E' l'applicazione che consente la distribuzione di patch sia a livello di applicativi che a livello di sistema operativo.

BigFix Compliance Si utilizza questa applicazione per garantire la compliance dei dispositivi, identificare irregolarità e risolverle.

BigFix Protection Questa applicazione viene adoperata per garantire una protezione real-time contro qualsiasi genere di malware (virus, trojan, worms, spyware, rootkits e altre minacce web). Ha ovviamente effetto sia sugli endpoint fisici che sulle macchine virtuali.

BigFix Inventory Come dice la parola stessa, questa applicazione si occupa di analizzare gli endpoint e generare un inventario di tutti i software su essi installati, generando dei report ed evidenziando, eventualmente, le irregolarità.

2.1.4 Fixlets

Le Fixlet sono il metodo attraverso il quale si svolgono tutte le operazioni, come distribuzione di software, installazioni di patch e configurazioni. Esse sono dei messaggi inoltrati ai client di BigFix e utilizzano un linguaggio di query specifico, il Relevance.

Il linguaggio Relevance

Con una Fixlet si può anche ispezionare un desiderato aspetto di un client. A tale scopo viene adoperato il linguaggio Relevance. Esso, infatti, consente di interrogare

il client, identificandone caratteristiche dell'hardware o del software tramite particolari costrutti, gli Inspectors. Una necessità può essere infatti quella di applicare una Fixlet solamente a dei client con determinate caratteristiche hardware/software oppure che si trovano in stati ben definiti. Si può, in questo modo, facilmente identificare il corretto sottoinsieme di client ai quali è destinata una nuova Fixlet ed applicarla solo ad essi.

2.2 IBM

Come detto, il lavoro di tesi si è svolto nell'ambito di un progetto formativo stipulato tra l'Università dell'Aquila e IBM Italia Spa. Questo progetto ha previsto un tirocinio svolto nella seda di Roma con obiettivo: "Esplorazione e prototipazione di metodi per portare prodotti BigFix su cloud", per l'appunto la realizzazione del prototipo di BigFix Saas.

La storia dell'IBM ha inizio nei primi decenni del novecento, ma è dagli anni settanta che entra nel mercato dell'informatica, soprattutto nel settore hardware. Negli ultimi venti anni il business si è spostato sempre più sul software. In particolare soluzioni cognitive e piattaforme cloud.

IBM Security Presso la sede di Roma presente il più importante laboratorio IBM italiano, il Rome Software Lab. Nella divisione italiana ci si concentra prevalentemente sullo sviluppo back-end. Una grossa fetta del laboratorio fa parte della divisione Security di IBM. Il portfolio di Security contiene per l'appunto prodotti che si occupano di diversi aspetti della security aziendale, tra questi BigFix è uno dei più consolidati.

2.3 Saas Exploration Project

Lo scopo del progetto al quale ho partecipato con il mio lavoro è quello di esplorare le tecnologie esistenti nel panorama cloud e realizzare il prototipo della versione SaaS di BigFix. A questo scopo, oltre a me, sono state allocate altre tre persone full time al conseguimento del progetto, sotto la guida dell'Architect Bernardo Pastorelli.

2.3.1 Il framework SCRUM

Il team adotta il framework agile SCRUM. Questo modo di operare è di sempre maggiore diffusione ed è basato su un approccio iterativo e incrementale nello sviluppo software. Il design e lo sviluppo sono divisi in iterazioni, denominate "Sprint", della durata fissa di due settimane. Queste due settimane terminano sempre con una versione funzionante del prodotto, il quale viene mostrato in una demo che evidenzia le nuove features implementate.

SCRUM utilizza un approccio empirico alla progettazione. La filosofia di fondo del framework è quella che la conoscenza deriva dall'esperienza, e quindi tutte le scelte che si prendono nel corso della progettazione devono avvenire alla luce di una sempre maggiore esperienza, la quale si ottiene avendo a disposizione il prima possibile un sottoinsieme del prodotto a se stante, testabile ed usabile. Dì qui l'approccio fortemente iterativo e incrementale massimizzando le opportunità di feedback.

All'inizio del progetto vengono definiti i requisiti del prodotto (item), i quali vengono da un'attenta analisi dei bisogni dell'utente. Ogni bisogno viene modellato con una Epica, che a sua volta viene prioritizzata e aggiunta al Product Backlog che le indicizza. Le Epiche vengono poi scomposte in User Stories, le quali si suddividono a loro volta in Task, ossia l'elemento atomico del progetto la cui implementazione viene presa in carico da un singolo componente del team.

L'inizio di uno Sprint è sempre caratterizzato da un meeting in cui si pianificano gli obiettivi. In questo contesto si fa sempre riferimento al Backlog incentrandosi sulle User Stories ancora non coperte. Si cerca quindi di suddividersi i Task in modo tale da avere a fine Sprint quelle nuove funzionalità usabili e dimostrabili. Demo che viene svolta sempre con la presenza di tutto il team e anche di colleghi di altri laboratori IBM.

2.3.2 Sistemi di controllo di versione

Da un'organizzazione di questo tipo ne scaturisce la necessità di tool di controllo di versione che permettano una fluida gestione del codice e della programmazione in

parallelo tra i diversi componenti del team.

GitLab A tal scopo si è adottato ormai da tempo, da tutto il team di BigFix, il software di controllo di gestione distribuito Git e una repository aziendale che consiste in una versione enterprise ad hoc per IBM di GitLab, un hosting service molto simile a GitHub.

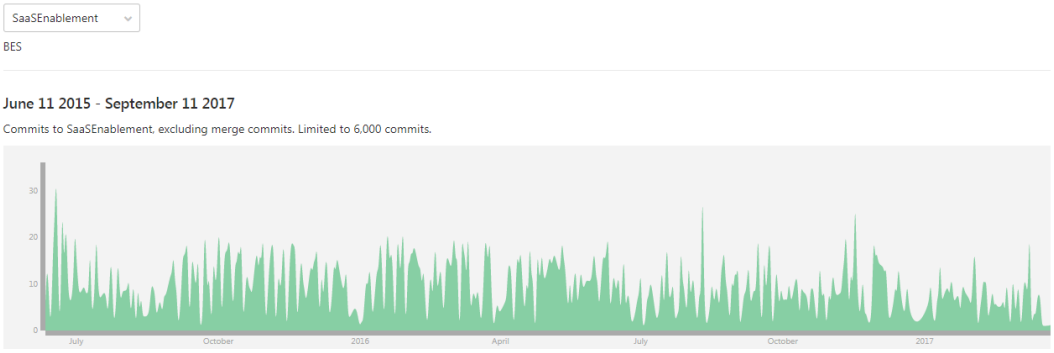

Figura 2.3: Panoramica degli ultimi contributi nel branch del progetto SaaS

Il flusso di lavoro è il seguente. Quando inizia il proprio task, il componente del team si pone su un proprio branch personale sul quale effettua i propri commit. Al termine del task viene fatta una merge request sul branch principale, sono una volta che si è testato il codice, per aggiungere i propri contributi al progetto. A questo punto, dopo una review effettuata da componenti del team accreditati, il nuovo branch verrà mergiato con il branch principale.

2.3.3 RTC

E' ovviamente necessario un tool che coordini anche la suddivisione dei task all'interno del team. A tal fine abbiamo utilizzato un software di IBM, ovvero Rational Team Concert (RTC), il quale è disponibile anche esternamente e gratuito nel caso di piccoli team. Esso offre comodi strumenti di agile planning e gestione di ciclo di vita del software. Ogni componente del team può così tracciare facilmente le aree di sua competenza. E' possibile inoltre usufruire di tool per il source control, controllo dei difetti e gestione delle build.

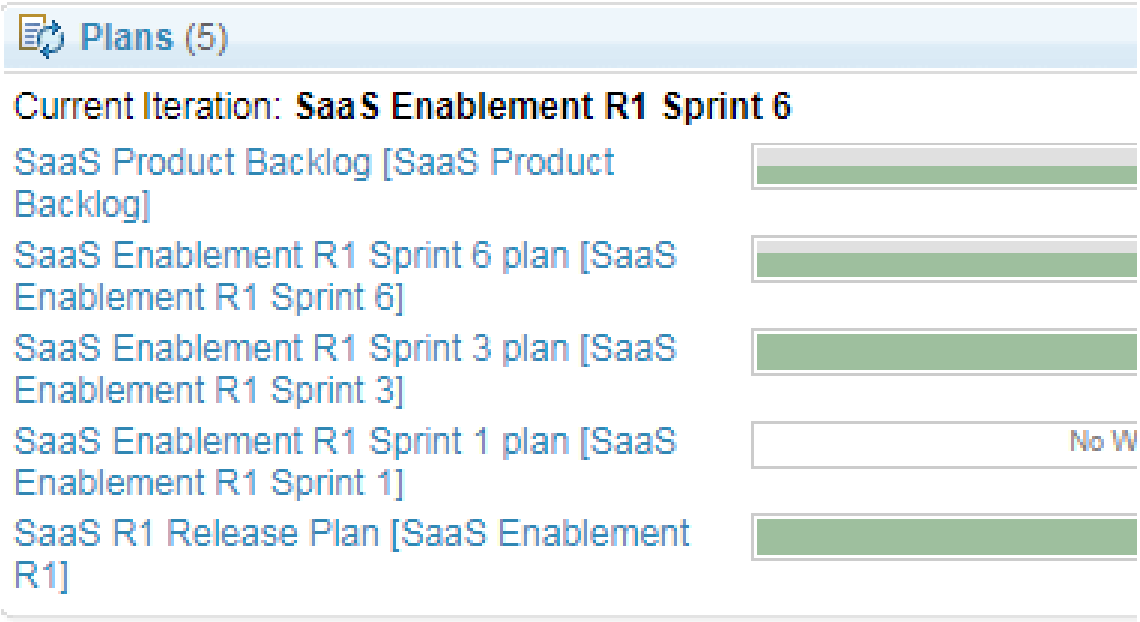

Figura 2.4: RTC. Un'esempio di come viene monitorato il completamento dei diversi sprint

Capitolo 3

Il Cloud e le sue sfaccettature

3.1 Cloud Computing

La differenza tra il possedere e l'utilizzare. E' questo l'aspetto cruciale del cambiamento apportato dal cloud computing rispetto al software tradizionale. Le risorse, che siano esse stesse archiviazione, elaborazione o qualsivoglia risorsa informatica, non sono mai ad hoc per un singolo utente, ma vengono assegnate on demand a diversi utenti e appartengono ad un insieme condiviso da tutti gli altri utenti del prodotto. Attraverso internet ogni utente può accedere a queste risorse in qualsiasi momento. Tali risorse vengono opportunamente allocate all'utente in maniera dinamica e completamente automatizzata. L'utente può utilizzare così anche software non installati sul proprio computer o usufruire di una memoria di massa accessibile da parte sua da qualsiasi dispositivo.

L'esperienza utente che si vuole fornire però è quella di un utilizzo esclusivo della risorsa, come nei software tradizionali, mentre in realtà la risorsa viene solo sapientemente distribuita tra gli utenti. Ciò fa si che, potenzialmente, un singolo utente possa acquisire risorse notevolmente maggiori nel caso medio.

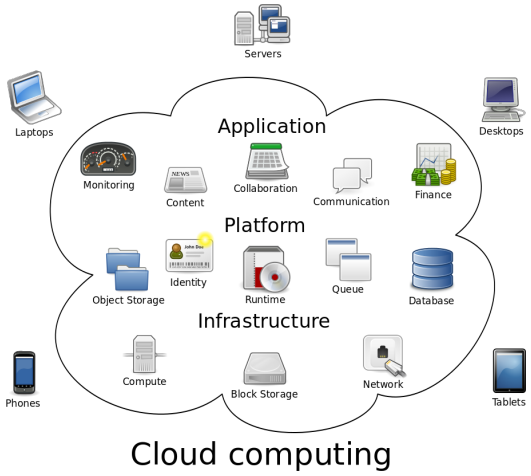

Figura 3.1: Diagramma logico di una rete Cloud Computing

3.1.1 Vantaggi del Cloud Computing

- Costo:
  Con l'avvento del cloud tutta la gestione dell'infrastruttura sottostante al software diviene a carico del provider. Vengono eliminate spese per la gestione dei data center locali. Facendo riferimento alla versione SaaS di BigFix ad esempio, il cliente viene sollevato dal pesante onere di utilizzare server locali e gestirne le relative connessioni. Il provider detiene tutto l'hardware di cui il cliente ha bisogno.

- Velocità:
  Anche qui ci risulta molto utile prendere come esempio la suite di BigFix. quando un nuovo cliente acquista il prodotto nella sua versione on-premises, un'incaricato di IBM si reca presso il cliente e lì inizia un lungo processo di installazione della suite che può impiegare diverse ore. Nello scenario SaaS il cambiamento è radicale. E' sufficiente che il cliente compili una form online, dopo alcuni minuti poi riceve una mail con il link per accedere al servizio.

- Prestazioni
  Una delle motivazioni principali per la quale si sceglie di fare uso del clud computing sono proprio le prestazioni, soprattutto se si adotta il paradigma PaaS.

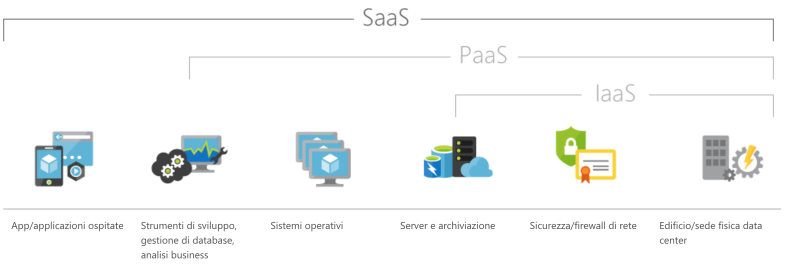

Figura 3.2: Panoramica delle principali tipologie Cloud

Esteriorizzando le risorse di calcolo, si può fare affidamento a dei provider che fanno dei server ad alte prestazioni il loro punto di forza. L'utente può, in questo modo, abbattere dei bottleneck che altrimenti risulterebbero di grande impedimento. Nel 2016 IBM mette per la prima volta a disposizione pubblicamente un computer quantico, proprio attraverso una piattaforma cloud (IBM Q). Questo può rappresentare un esempio estremo in ottica prestazioni, ma che può rendere un'idea di quale potrà essere il trend nei prossimi anni.

- Affidabilità
  Operazioni di mirroring da parte dei provider dei servizi cloud fa sì che il backup dei dati sia continuo ed economico.

3.2 Tipologie di servizi Cloud

Il termine Cloud risulta in realtà molto generico. Esso comprende diverse tipologie di fornitura dei servizi, a seconda della risorsa che viene offerta dal provider. La maggior parte dei servizi di Cloud Computing rientrano in tre tipologie principali: Infrastruttura distribuita come Servizio (IaaS, Infrastructure as a Service), Piattaforma distribuita come Servizio (PaaS, Platform as a Service) e Software come un Servizio (SaaS, Software as a Service). Oltre a queste tipologie, annoveriamo anche soluzioni minori come il DaaS (Data a a Service) e l'HaaS (Hardware as a Service). Andiamo a vedere nel dettaglio come, a seconda della modalità di utilizzo del paradigma Cloud, questi paradigmi si differenziano.

3.2.1 IaaS, Infrastructure as a Service

E' la tipologia più basilare. Vengono messe a disposizione piattaforme di elaborazione. Utilizzando un IaaS si affittano le infrastrutture utili ai propri scopi, come ad esempio server, macchine virtuali (VMs), risorse di archiviazione, reti e sistemi operativi. Può, inoltre, essere messo a disposizione anche hardware in remoto. Il provider di servizi cloud gestisce l'infrastruttura, mentre l'utente acquista, installa, configura e gestisce il software, tra cui sistemi operativi, middleware e applicazioni.

Vantaggi Una soluzione IaaS è quella che garantisce maggiore flessibilità. Tra i vantaggi principali ricordiamo:

- Elevata scalabilità
  Il modello IaaS permette una scalabilità verticale rapida ed economica
- Rapidità di innovazione
  Nel caso del lancio di un nuovo prodotto basato sulla piattaforma IaaS, il tempo di attesa per le nuove configurazioni infrastrutturali è solamente dell'ordine di pochi minuti.
- Adattabilità alle richieste
  Un modello IaaS è estremamente flessibile alle variazioni delle richieste. Si possono facilmente aumentare le risorse nei momenti di picco e ridurle quando non è necessario, risparmiando quindi denaro.

3.2.2 PaaS, Platform as a Service

Una piattaforma distribuita come servizio (PaaS, Platform as a Service) è un ambiente cloud di sviluppo completo. Una soluzione PaaS è progettata per consentire il ciclo completo dello sviluppo delle applicazioni: creazione, test, distribuzione, gestione e aggiornamento. L'utente ha tutta la libertà di sviluppare gli applicativi a proprio piacimento, ma lavora con componenti software già pronti all'utilizzo (microservices). Questi componenti non sono localizzati presso chi utilizza il cloud, bensì presso il provider, il quale si occupa del loro mantenimento e aggiornamento. Il modello PaaS consente di evitare le spese e la complessità legate all'acquisto e alla

gestione di licenze software, middleware e infrastruttura delle applicazioni sottostanti o strumenti di sviluppo. L'utente gestisce le applicazioni e i servizi che sviluppa e il provider cloud si occupa di tutto il resto.

Vantaggi Uno scenario PaaS riduce quindi notevolmente la quantità di codice da scrivere semplificando quindi il lavoro dello sviluppatore e aumentandone la produttività. Inoltre risulta molto più semplice così il porting di un prodotto da web a mobile e viceversa. Componenti molto complessi e costosi possono inoltre essere messi a disposizione, con un utilizzo limitato, anche per sviluppatori che altrimenti non potrebbero permetterselo.

IBM Bluemix

Troviamo, sempre all'interno di IBM, uno dei principali servizi cloud PaaS presenti sul mercato: IBM BlueMix. L'utente può usufruire di un'astrazione di molte componenti utili allo sviluppo. Si può, ad esempio, fare uso di Database specifici, di moduli dedicati all'IoT, di tecnologie Blockchain e molto altro. Ne vediamo alcuni esempi nella figura 3.3

IBM Watson

Tra le componenti sviluppate da IBM merita una menzione anche Watson. Watson è un sistema di intelligenza artificiale in grado di rispondere a domande espresse in un linguaggio naturale. Tra le funzionalità ci sono quelle di elaborazione del linguaggio naturale, information retrieval, rappresentazione della conoscenza, ragionamento automatico e tecnologie di apprendimento automatico.

3.2.3 SaaS, Software as a Service

Il Software as a Service (SaaS) è un modello di distribuzione del software in cui l'applicativo e gli eventuali servizi collegati sono eseguiti in un ambiente centralizzato e gli utenti vi accedono via rete, quasi sempre via Internet e usando un browser come interfaccia. I SaaS sono ormai sempre più diffusi. Tra i maggiori si ricordano le Google Apps (ad esempio Gmail, Google Drive, Google Calendar) e la suite Microsoft

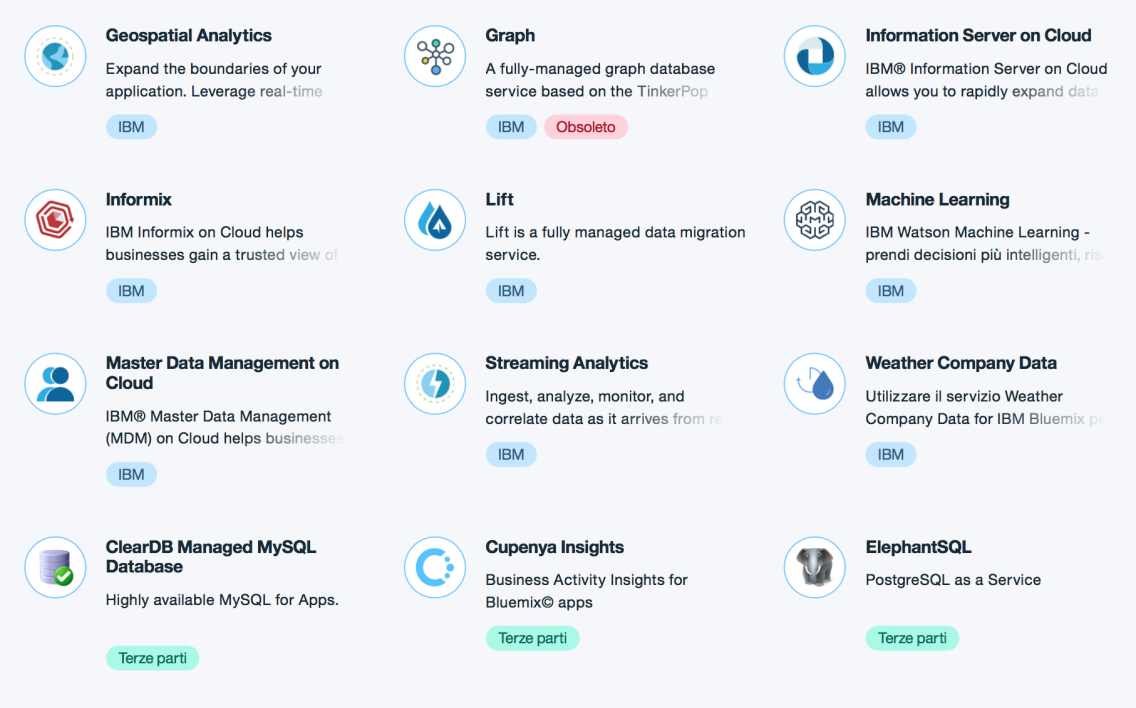

Figura 3.3: Esempi di moduli presenti nel catalogo BlueMix

Office 365. Con la metodologia SaaS il provider fornisce tutto il software direttamente all'utente, al quale non resta che usarlo senza preoccuparsi di installazioni e configurazioni. L'utente non paga per il possesso del software, ma per il suo utilizzo. Spesso vengono infatti applicate tariffe in base all'utilizzo del prodotto stesso.

Spesso un'architettura SaaS è multi-tenant, ossia una sola applicazione server viene utilizzata da più utenti mantenendone, al tempo stesso, separati gli ambienti e i dati. Il concetto di multi-tenancy verrà approfondito maggiormente nei prossimi capitoli.

Attualmente le principali software house attive nel mondo aziendale traggono dalle offerte SaaS circa il venti percento del loro fatturato, nel giro di tre-quattro anni la maggior parte di loro prevede che il mercato SaaS diventi sempre più fondamentale.

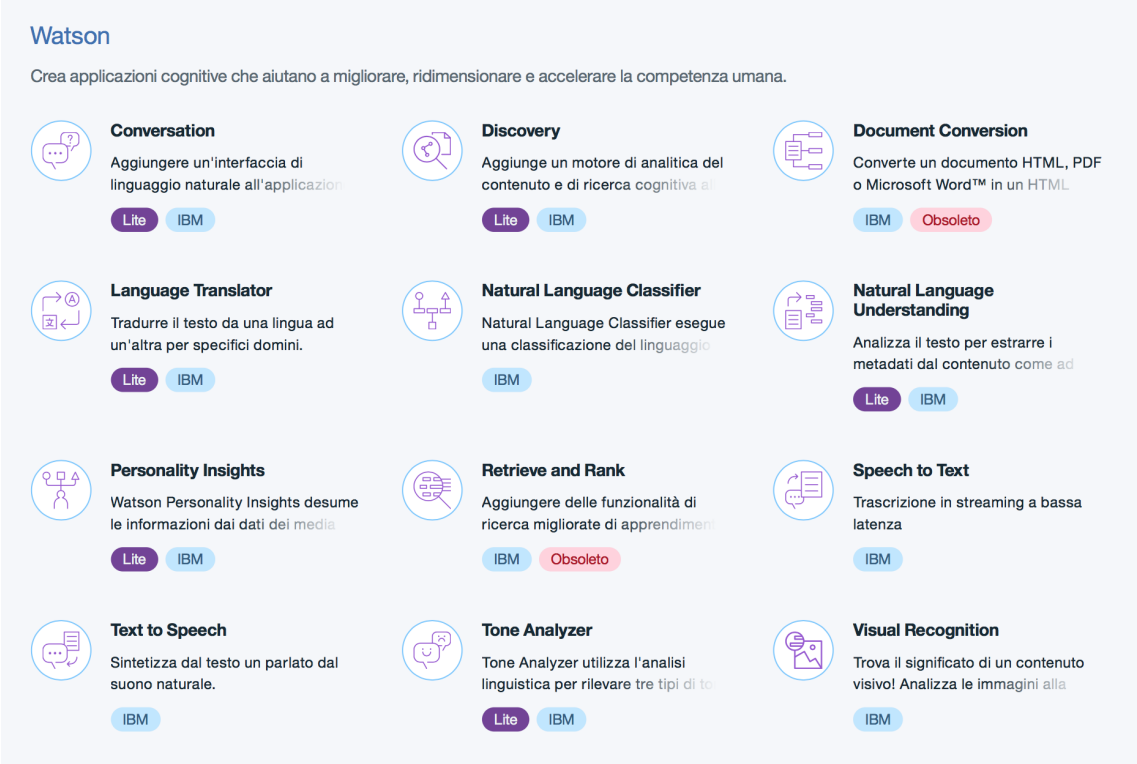

Figura 3.4: Alcuni moduli Bluemix appartenenti a Watson

Vantaggi

Con la metodologia SaaS il provider fornisce il software all'utente già pronto all'uso, l'utente non si deve quindi preoccupare di nient'altro. I clienti, a questo punto, non pagano più per il possesso del software, ma per l'utilizzo del software stesso. In questo modo spesso possono essere notevolmente abbattute le spese iniziali per l'installazione e la configurazione di un prodotto, sostituendole con un costo di funzionamento. Il SaaS comporta quindi una spesa inferiore, ma ricorrente. E' il vendor però a preoccuparsi della manutenzione.

La caratteristica preponderante del cloud SaaS è proprio quella di poter raggiungere i propri dati personali da qualunque luogo e con qualunque dispositivo. E' questa la vera rivoluzione del cloud nel senso più comune del termine. Quando si ha bisogno di lavorare con i propri dati personali, non si ha più la necessità di avere

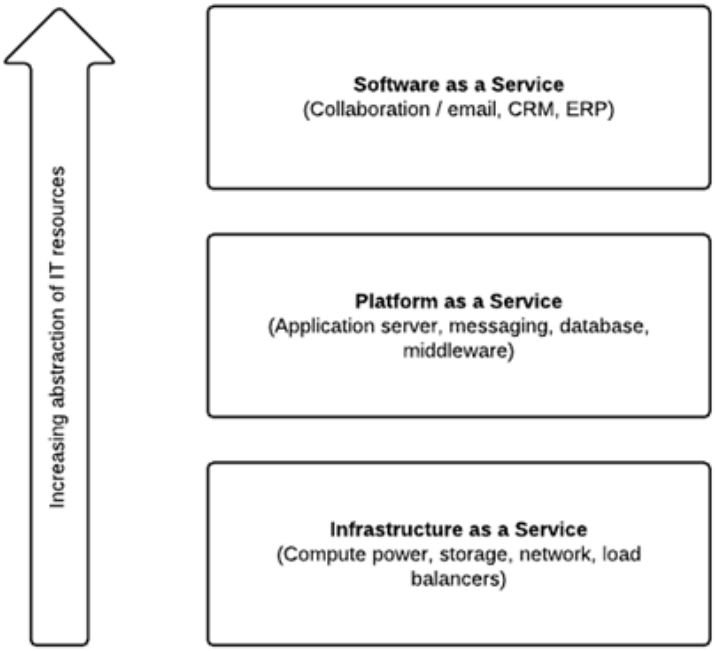

Figura 3.5: Livelli di astrazione nei modelli Cloud Computing

con se hardware specifico, il proprio laptop o la propria usb key, ma basta avere le proprie credenziali al accedere così allo spazio dedicato.

L'accesso alle tecnologie più sofisticate diviene inoltre alla portata di tutti. Software come gli ERP (Enterprise Resource Planning) e i CRM(Customer Relationship Management), possono essere utilizzati anche da quelle organizzazioni che prima non potevano permettersi un investimento di questa portata.

Un elemento che contraddistingue il SaaS è il notevole grado di apertura verso altre componenti, il che le rende altamente riusabili e flessibili, ciò è ovviamente un grande punto di forza in quanto i requisiti dell'utente spesso cambiano in continuazione.

Gli scenari di aggiornamento cambiano radicalmente. La distribuzione degli aggiornamenti è pressoché immediata e tutti gli utenti hanno la certezza di operare con l'ultima versione del software. Le fasi di upgrading non comportano più l'assenza del servizio come nella versione on premise. L'utente non si accorge neanche del processo di aggiornamento e si ritrova ad utilizzare il software aggiornato.

Svantaggi

Il modello ha ovviamente anche i suoi svantaggi, o perlomeno alcune criticità che vanno tenute presenti. La principale sta nella gestione dei dati aziendali, che sono localizzati nei data center del cloud provider e questo può essere giudicato un rischio per la privacy delle informazioni o addirittura costituire una violazione delle norme che devono osservare le aziende operanti in settori particolari come, ad esempio, la sanità e la difesa.

Tra gli altri aspetti di cui tenere conto ci sono sicuramente le prestazioni e l'affidabilità delle connessioni, che risultano essere i veri bottleneck non solo delle soluzioni SaaS, ma di tutte le soluzioni cloud.

Infine altre criticità riguardano il provider dei servizi. Bisogna tenere conto quanto sia affidabile e se presenta, ad esempio, il rischio che esca dal mercato o ritiri il prodotto software SaaS.

3.3 SOA, Service Oriented Architecture

Il concetto di Service Oriented Architecture è affine a quello di SaaS. Un SaaS può rappresentare la percezione da parte dell'utente della modalità di utilizzo di una Architettura Service Oriented. Un servizio ha l'obiettivo di incapsulare una ben precisa funzionalità, per renderla disponibile e accessibile come servizio software sul web. L'Architettura Orientata ai Servizi è quindi uno stile architetturale per la costruzione di una molteplicità di sistemi o applicazioni sulla base della composizione di un insieme di servizi. Spesso, quindi, queste applicazioni non fanno altro che comporre un SaaS, il quale trova in un'architettura di questo tipo una appropriata implementazione.

3.4 Il software On Premise

Abbiamo appena analizzato le peculiarità delle singole tipologie di servizi cloud. Anche il classico modello on premise però ha i suoi punti di forza.

Vantaggi del Software On Premise

- Controllo esclusivo su sistemi e dati
- Alta personalizzazione
- Gestione interna dei dati sensibili
- Alto investimento iniziale ammortizzato nel lungo periodo

Nel caso in cui si abbiano già a disposizione tutte le infrastrutture necessarie, non risulta più vero che la soluzione cloud sia la più economica per far fronte alla necessità di un determinato applicativo. E' anche necessario però che il software sia abbastanza centralizzato per adottare una soluzione on premise.

Occorre quindi fare un'analisi di quanto sia necessario personalizzare il software di cui si ha bisogno e averne il pieno controllo, tenendo conto che un software on premise richiede molta più cura, manutenzione e lavoro di una soluzione basata su

cloud. Il paradigma di fornitura on premise risulta ancora essere la soluzione più adatta nel caso in cui la gestione diretta dei dati sia fondamentale per policy aziendali oppure sia necessaria una maggiore flessibilità di configurazione per l'integrazione con altre architetture software. Un'altro requisito che ne può richiedere l'adozione è la necessità che l'architettura fisica del software sia geograficamente centralizzata.

Capitolo 4

SaaS e i suoi requisiti

Prima di conoscere a fondo le tecnologie proprie della realizzazione di un SaaS, occorre essere consapevoli di quali siano le esigenze di un software di questo tipo. Ovviamente, per la sua natura, il SaaS ci pone davanti ad esigenze del tutto nuove rispetto ai tradizionali paradigmi, in quanto la modalità di accesso al sistema è del tutto nuova. Ad esempio non possiamo sapere a priori quanti utenti dovranno essere serviti dal provider in un determinato istante, o quale carico di lavoro richiederanno al sistema. Andiamo a vedere nel dettaglio quali sono i principali aspetti critici di cui tenere conto nella fornitura di un SaaS.

4.1 Availability

Il concetto di Availability è ben definito dallo standard ITU-T E.800: "E' l'abilità di un sistema di essere in uno stato che soddisfa un determinato requisito, in determinati istanti di tempo, assumendo che le risorse a lui necessarie siano disponibili."

Dobbiamo tenere conto inoltre che l'Availability rappresenta la porzione di tempo in cui il sistema si comporta secondo le proprie specifiche. Va tenuto in considerazione che, quando si verifica un fallimento, al tempo di non-Availability si aggiunge il tempo per porre rimedio al fallimento e far ripartire il sistema.

4.2 Reliability

La Reliability è definita anch'essa dalla International Telecommunications Union (ITU-T) recommendations E.800, come segue: "E' l'abilità di un sistema di soddisfare una funzione richiesta, sotto determinate condizioni e per un certo intervallo di tempo". La Reliabiity è un concetto affine all'Availability, con la differenza che la Reliability si riferisce all'abilità del sistema di compiere i suoi scopi durante un'intervallo di tempo, e non un istante preciso. Essa infatti si quantifica con una probabilità.

4.3 Dependability

I concetti di Availability e Reliability si possono astrarre insieme nel concetto di Dependability. Esso consiste proprio nella capacità di un sistema di fare in modo che si possa "dipendere" da esso, ossia di mostrarsi affidabile ai propri utilizzatori. Come possiamo notare la Dependability è un concetto ben definito, ed ha quindi le sue metriche ben definite che permettono di quantificarlo nei vari aspetti. La Reliability stessa, può rappresentare una metrica per quantificare l'affidabilità di un sistema. Andiamo a descriverne qualcuna.

MTTF, Mean Time To Failure Misura l'intervallo di tempo tra due eventi di "Failure" nei quali il sistema non è riuscito a portare a termine il proprio compito.

POFOD, Probability Of Failure On Demand anche questa è una probabilità. In particolare si vuole misurare la probabilità che un sistema fallisca facendo fronte ad una richiesta che gli è stata sottoposta. Differisce con l'Availability infatti perché questa misurazione si basa sulle richieste ricevute, mentre l'Availability è su base temporale. POFOD è una metrica molto importante per quei sistemi che vengono chiamati in causa raramente, ma all'interno di processi critici.

4.3.1 Availability e Reliability nel contesto Cloud

Possiamo immaginare, a questo punto, quanto sia fondamentale un'altissima Availability e Reliability per i servizi Cloud. In caso di Failure, infatti, possono potenzialmente essere coinvolti tutti gli utenti serviti dal provider che ha subito il guasto. I servizi erogati via Cloud dovrebbero essere disponibili da chiunque li richieda e da qualunque parte del mondo ventiquattro ore su ventiquattro. Ovviamente una affidabilità massima non è verosimile, ma ci si aspetta una Reliability di molto vicina al 100. Ad esempio, BlueMix dichiara una Reliability del 99,95 percento. Per rendere l'idea, una Reliability del 99,95 percento sta a significare che, sulla base annuale, il servizio può non essere disponibile per circa 4 ore.

Upgrade Sono gli scenari di Upgrade un aspetto maggiormente critico. Come si può immaginare, molti servizi SaaS hanno bisogno di essere continuamente aggiornati e modernizzati. All'uscita di una nuova versione del software occorre che questa venga distribuita a tutti gli utenti del servizio. Distribuire un software su scala Cloud non è semplice come si possa pensare. Non si può infatti interrompere l'erogazione del servizio per far partire il processo di aggiornamento del prodotto, perché infatti questo processo può essereanche molto lungo. Occorrerà quindi adottare delle tecniche che diano l'impressione all'utente di una continuità del servizio. Vedremo nei prossimi capitoli quale strategia abbiamo adottato con BigFix SaaS.

4.4 Scalability

Definiamo invece la Scalabilità come la capacità di un software di adattarsi all'aumento del carico di lavoro senza un decadimento delle prestazioni. Ci aspettiamo, ad esempio, che BigFix SaaS non abbia difficoltà ad operare con un numero considerevolmente alto di endpoint, quali possono essere i 250.000 client supportati dalla suite ne versione on premises. Il tutto ovviamente mantenendo gli stessi standard prestazionali che si osservano con l'interazione con pochi Endpoint. Questa qualità può essere in realtà merito di due diversi aspetti della Scalabilità:

- Scalabilità Verticale
  Consiste nell'incrementare le risorse del sistema, aumentando ad esempio le

risorse hardware del server. SI può aggiungere RAM o CPU e rendere il sistema più performante sotto l'aumento del carico di lavoro.

- Scalabilità Orizzontale
  Con la Scalabilità Orizzontale si ha un approccio diverso. Al verificarsi di un maggiore carico di lavoro non vengono incrementate le risorse di un server, ma vengono aggiunti nuovi server in parallelo, andando a formare tutti insieme un sistema unico. E' questo l'aspetto più interessante al punto di vista Cloud, ma ovviamente occorre che anche a livello software ci siano meccanismi intelligenti di ripartizione del carico nei diversi nodi che compongono il sistema.

Ovviamente la Scalabilità Orizzontale è un aspetto centrale nel Cloud Computing. Le richieste di utilizzo di un componente software possono crescere notevolmente e su scala mondiale al crescere degli utenti. Al tempo stesso le risorse di calcolo però possono essere anch'esse distribuite geograficamente in luoghi diversi e non è necessario che il sistema risieda nello stesso luogo. Introduciamo così notevole flessibilità ed agilità al sistema per far fronte ai bisogni di scaling causati dall'aumentare delle risorse richieste dagli utenti. Il tutto deve rientrare in un processo automatico di adattamento alle richieste, non possiamo pensare infatti che ci sia un intervento umano per far scalare il sistema secondo le esigenze.

4.5 Monitoring

Quello di Monitoring più che requisito andrebbe definito una necessità del Cloud Computing. La necessità di verificare che il servizio erogato dal provider si mantenga al di sopra di livelli ben stabiliti di qualità del servizio (QoS). Questi livelli vengono stabiliti con il Service Level Agreement tra le due parti. Occorre definire delle metriche e stabilire delle metodologie accurate per compiere le misurazioni, effettuando opportunamente delle medie dei valori ottenuti o calcolare picchi dei valori, come nel caso della latenza.

Tra i più significativi parametri da misurare troviamo sicuramente il tempo di latenza, il tasso di perdita dei pacchetti Ip e i disturbi sul segnale. Bisogna considerare

che dal punto di vista dell'esperienza utente il sistema si presenta come un servizio unico. All'utente interessa che questo servizio gli garantisca affidabilità, facilità di utilizzo e tempi di risposta rapidi. Dietro questa astrazione sappiamo che ci sono in realtà molte componenti e spesso anche sistemi diversi che interagiscono tra loro. Il Provider deve quindi essere in grado di tracciare eventuali problemi, per risalire alla causa e individuare le relative responsabilità. E' fondamentale quindi riconoscere e tenere sotto controllo i fattori fondamentali che determinano la qualità percepita dall'utente.

Capitolo 5

SaaS, le tecnologie che ne consentono la realizzazione

Nel corso degli ultimi anni, con il proliferare delle piattaforme e dei servizi di Cloud Computing, sono nate e si sono sviluppate molte tecnologie per soddisfare le nuove esigenze e i nuovi requisiti appena visti che questa rivoluzione della fruizione del software ha comportato.

5.1 Microservizi

Un concetto fondamentale, di cui il Cloud Computing fa largamente uso, sono i mircoservizi. Cominciamo col darne una definizione abbastanza formale: "Lo stile architetturale a microservizi è un approccio allo sviluppo di una singola applicazione come insieme di piccoli servizi, ciascuno dei quali viene eseguito da un proprio processo e comunica con un meccanismo snello, spesso una HTTP API.(Martin Fowler)".

L'approccio è quello di dividere le funzionalità del sistema in più microservizi. Ad ogni microservizio corrisponde una necessità dell'utente. La filosofia di dividere il software in base alle responsabilità è già presente da tempo nell'ingegneria del software. Una suddivisione modulare del sistema in base ai casi d'uso dell'utente è già presente da tempo nell'Object Oriented Design, ma la novità apportata dai

microservizi è che il sistema con essi risulta scomposto in realtà in piccoli servizi completamente indipendenti tra loro. Ogni microservizio si preoccupa infatti di risolvere un particolare problema del cliente, un unico scenario. La comunicazione tra i servizi avviene attraverso la rete al fine di garantire l'indipendenza tra i servizi ed evitare ogni forma di accoppiamento. Ogni microservizio, infatti, rappresenta un'entità separata che generalmente viene pubblicata come un modulo di una Platform as a Service.

5.1.1 Il modello monolitico a layer

Secondo il classico modello a layer le funzionalità vengono suddivise in base al grado di astrazione tra i vari livelli, usando delle tecnologie proprie di ogni livello. Questi sono separati a livello logico e comunicano tra di loro. Con questa architettura però, nonostante ci sia questa suddivisione a strati, il software risulta essere un unico sistema monolitico, sebbene molti componenti possano essere comunque riusabili.

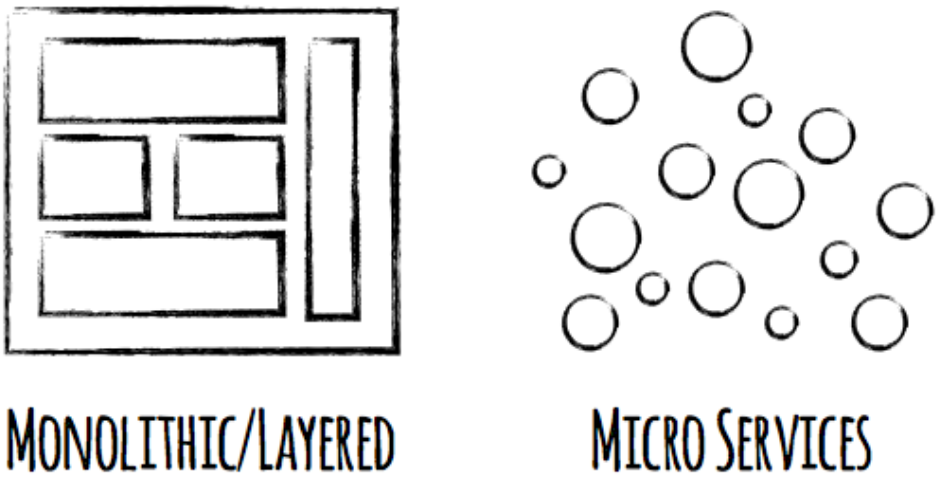

Figura 5.1: Il modello monolitico e i microservizi

5.1.2 Confronto tra l'architettura a microservizi e il modello monolitico

La scelta di adottare uno o l'altro approccio viene dopo un'attenta analisi dei requisiti che il sistema deve soddisfare. In questo studio però va anche tenuto conto quanto le esigenze possano cambiare nei futuri utilizzi del software.

- La struttura interna di tutto un sistema monolitico è composta principalmente dai layer di interfacciamento con l'utente, logica di business e persistenza dei dati. In un'architettura a microservizi non troviamo questa divisione a livello di sistema, ma la ritroviamo semmai all'interno di un singolo microservizio atomico. Ogni microservizio, ad esempio, si occuperà di preservare il suo stato tramite l'utilizzo di un proprio database non condiviso con gli altri microservizi.

- Uno dei fattori chiave da considerare è la scalabilità. Per scalare un'applicazione monolitica occorre necessariamente clonarla in più server, macchine virtuali o contenitori.

- Quando occorre scalare orizzontalmente un'architettura a microservizi, si creano e si distribuiscono indipendentemente tra loro repliche dei microservizi in più server o container.

5.1.3 Vantaggi e svantaggi di un'architettura a microservizi

Un'architettura di questo tipo porta con se ovviamente anche i suoi svantaggi e i suoi vantaggi. Sono proprio questi ultimi che la rendono molto adatta al Cloud Computing.

Vantaggi

- Velocità
  L'architettura a microservizi è quella che si sposa meglio con la metodologia agile. un microservizio deve avere sempre delle dimensioni ridotte e il suo sviluppo dovrebbe avere una durata di circa due settimane. Ciò porta ad

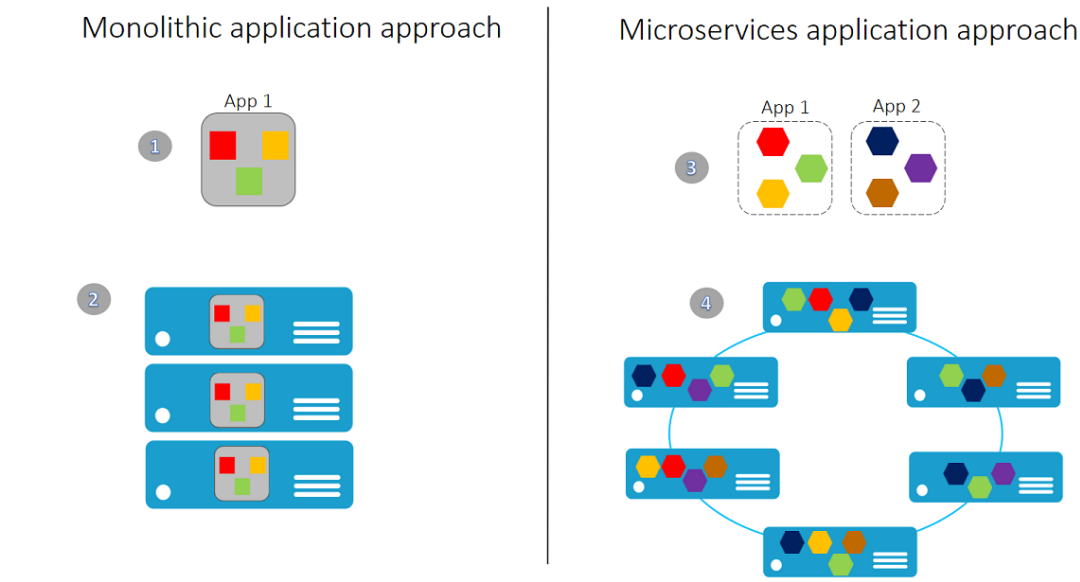

Figura 5.2: Il modello monolitico e i microservizi

avere fin da subito piccole porzioni del sistema (i servizi appunto), pronte, testabili ed utilizzabili. Ogni microservizio inoltre è autonomo e può quindi giungere in ambiente di produzione indipendentemente dagli altri. In questo modo si riesce a reagire molto velocemente alle esigenze di mercato.

- Sperimentazione
  Con i microservizi la modularità del sistema è un notevole punto di forza. Sperimentare nuove tecnologie all'interno di un singolo microservizio ha un impatto nullo su tutti gli altri. Si è in questo modo molto più invogliati a ricercare sempre più nuove tecnologie da inserire nel proprio prodotto. Il rischio è minimo in quanto, anche nel caso l'esperienza risulti fallimentare, la mole di lavoro che comporta la modifica di un microservizio e veramente molto ridotta.

- Tecnologie ad hoc
  Altro fattore da considerare è la possibilità di differenziare le tecnologie a seconda del microservizio. Si prenda come esempio la vastità di database che

sono disponibili all'uso. Un database che è appropriato per un microservizio potrebbe non essere la scelta migliore per un altro.

- Scalabilità
  I software con un'architettura a microservizi sono pensati per scalare orizzontalmente in maniera estremamente agevole. Si possono replicare a piacimento tramite l'utilizzo di containers e hanno un comportamento distribuito anche nel caso si trovino sulla stessa macchina, in quanto i container li isolano da tutti gli altri.

- Facilità di Deployment
  Le modifiche hanno un impatto molto ridotto nell'interno sistema. Grazie a ciò è possibile rilasciare sul mercato il software aggiornato con frequenze molto maggiori. Potenzialmente ogni modifica può subito essere pubblicata e non occorre attenersi a lunghi processi di release in cui si cerca il più possibile di accumulare modifiche da effettuare per poi applicarle tutte insieme.

- Portabilità
  Il software a microservizi è facilmente componibile e portabile su più contesti e dispositivi, come web,mobile ma anche sistemi embedded o dispositivi indossabili ad esempio.

Svantaggi

- Dipendenza dalla rete
  E' questo il principale punto di critico di un'architettura a microservices. Abbiamo quanto l'interazione tra i singoli microservizi faccia affidamento su una comunicazione attraverso internet. Ovviamente questo deve essere un requisito fondamentale. In mancanza di una connessione adeguata tutto il sistema smette di funzionare.

- Identità e autenticazione
  Una volta che un utente del software effettua il login occorre garantire che la sua autenticazione, e soprattutto la sua identità, venga mantenuta in tutti i microservizi che andranno a comporre la sua esperienza utente.

5.2 Containers

I container sono l'habitat naturale dei microservizi. Essi forniscono al software tutto ciò che gli è necessario, garantendogli un ambiente estremamente leggero e flessibile.

Figura 5.3: Containers nell'accezione dell'industria dei trasporti

Già la parola stessa container vuole alludere ad una analogia con l'attrezzatura specifica dei trasporti. Da dove è nata lì la necessità dell'utilizzo dei container? Immaginiamo lo scenario in cui alcune merci andassero trasportate prima via terra, magari con un camion, e poi via mare per raggiungere un altro stato. Prima dell'avvento dei container il camion arrivava al porto, andava aperto e il contenuto passato in un mercantile pronto sul molo. L'operazione era effettuata manualmente e i singoli imballaggi erano trasbordati uno alla volta con grande dispendio di tempo e mezzi. Si iniziò allora a considerare più pratica l'idea di trasbordare sulla nave l'intero corpo del camion. Nacquero così i container: contenitori multiuso, realizzati in formati standard che possono essere passati con facilità da un camion a una nave, poi magari su un treno merci e così via.

E così anche nel nostro caso abbiamo bisogno di uno strumento che possa contenere i microservizi e le applicazioni, che ci consenta di manovrarle con facilità senza sapere cosa facciano esattamente. I containers sono infatti un metodo di virtualizzazione del sistema operativo che ha come obiettivo quello di isolare il software che ospita, permettendo di eseguire le applicazioni e le loro dipendenze in processi completamente isolati. L'infrastruttura del container interagisce direttamente con il kernel della macchina che lo ospita, scavalcando gli altri layer. In qualsiasi sistema operativo collochiamo il container, le sue configurazioni interne rimarranno sempre separate, e l'applicazione contenuta avrà garantito il suo ecosistema necessario all'esecuzione. Non ci si deve preoccupare, ad esempio, di produrre una versione software per Windows e una per sistemi UNIX, ma la soluzione a container e unica e portabile.

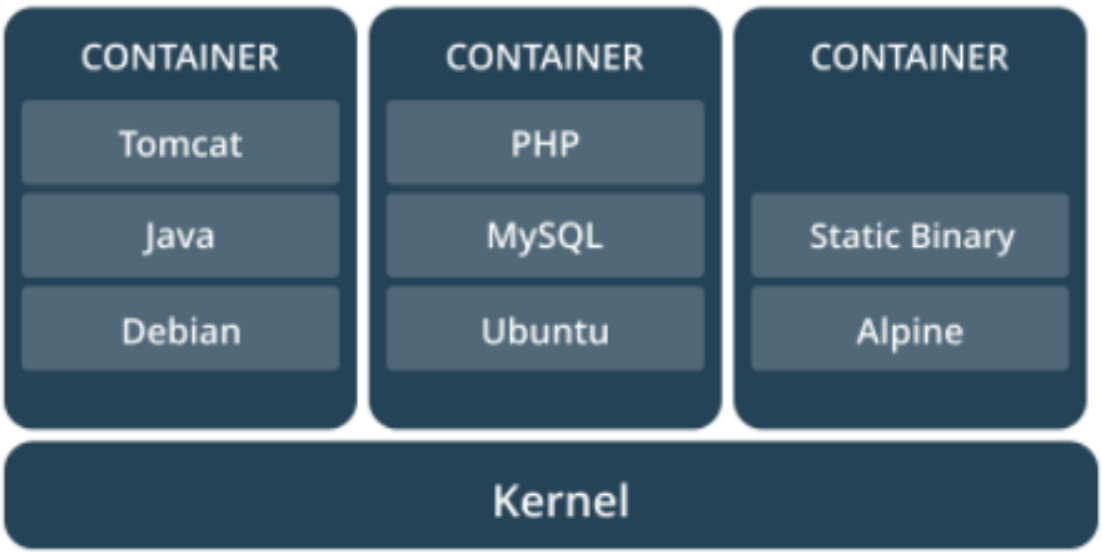

Figura 5.4: Schematizzazione di tre container sulla stessa macchina ospitante

5.2.1 I containers e le Macchine Virtuali

Stando alla descrizione dei container che abbiamo dato fino a questo punto potrebbe sorgere una domanda: perché utilizzare i container se potrebbero essere usate delle

macchine virtuali? Una Virtual Machine è per sua natura già isolata dalla macchina che la ospita. La risposta sta nella leggerezza d'uso dei container. I container infatti non virtualizzano l'hardware della macchina, ma solamente il layer applicativo, rendendoli più portabili ed efficienti.

Non avendo un proprio sistema operativo, il peso di un container e dell'ordine di qualche Megabyte, contro i Gigabyte di una macchina virtuale che sovrappone il proprio sistema operativo a quello della macchina ospitante. A differenza delle macchine virtuali inoltre, i container non necessitano dell'Hypervisor, un componente che svolge delle attività di controllo e coordinamento sulle macchine virtuali.

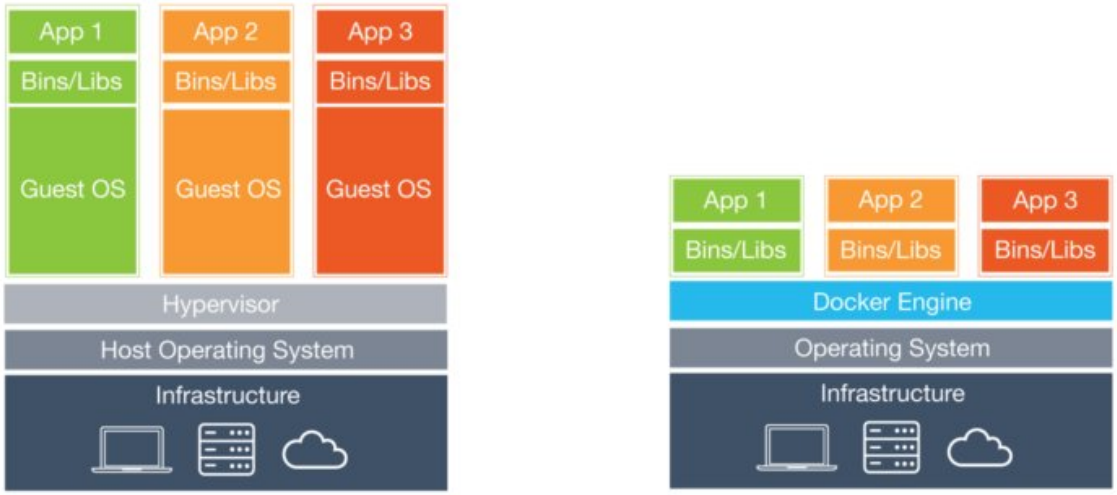

Figura 5.5: Confronto tra macchine virtuali e container, nell'esempio con l'utilizzo di un Docker Engine

5.2.2 Vantaggi dei container

Andiamo a ricapitolare quali sono le principali motivazioni che possono spingerci all'adozione dei container.

- Coesione dell'ambiente
  L'ambiente di un container è fortemente disaccoppiato dalla macchina in cui

si trova. Questo fa si che il proprio contenuto sia facilmente replicabile e portabile ovunque.

- Gestione delle risorse
  Con i containers si ha una gestione delle risorse di calcolo molto più efficiente. Richiedendo poche risorse alla macchine ospitante, si ha la possibilità di eseguire molti più container contemporaneamente.

- Produttività
  Diminuendo le dipendenze, ci si alleggerisce di tutta la mole di lavoro necessaria per configurare correttamente un prodotto.

- gestione degli aggiornamenti.
  Molto spesso la gestione degli aggiornamenti è un meccanismo già compreso nei container engine, come Docker.

5.2.3 I container e i microservizi

Possiamo quindi comprendere come i container siano estremamente appropriati ad ospitare del microservizi. Ponendo un microservizio dentro ogni container, abbiamo la garanzia che ogni microservizio operi come un sistema separato, anche nell'eventualità che due servizi risiedano nella stessa macchina fisica. Ogni microservizio inserito in un container gode automaticamente di tutti i vantaggi di portabilità e flessibilità che sono tra i requisiti fondamentali che ci spingono verso l'architettura a microservizi.

5.2.4 Docker

Docker è una piattaforma open source nata nel 2013 e scritta in linguaggio Go. Il suo scopo è quello di facilitare ed automatizzare il deployment delle applicazioni all'interno dei container. Il tool offre allo sviluppatore delle comode API di gestione che consentono agli sviluppatori di testare le applicazioni, effettuare build e distribuire il proprio prodotto. Docker fa utilizzo di numerose librarie come, ad esempio, libcontainer che ha lo scopo di interagire con il kernel di Linux. In questo modo,

Figura 5.6: Logo di Docker

isolando risorse, servizi e processi, dalla prospettiva dell'utilizzatore del container si ha l'impressione di un utilizzo esclusivo del sistema operativo.

Il contiene diviene con Docker così l'unità di distribuzione del prodotto. Quando il servizio è stato correttamente implementato, è pronto per essere distribuito e, attraverso il container, può essere inserito in un orchestratore per garantirne il ciclo di vita.

Docker Engine L'Engine di Docker è una struttura a strati, troviamo:

- Il server, un demone chiamato dockerd che è sempre in running. E' il componente che maneggia gli oggetti Docker.
- Le REST API, che rappresentano l'interfaccia per interrogare il server.
- La Command Line Interface, che è quella con la quale si interfaccia l'utente e si aggancia alla REST API.

Architettura di Docker L'architettura di Docker è la classica architettura client-server. Il client parla direttamente con il Docker deamon che si trova sul server. E' proprio il dockerd che si prende l'onere di fare gran parte del lavoro, quello di far partire e distribuire i container.

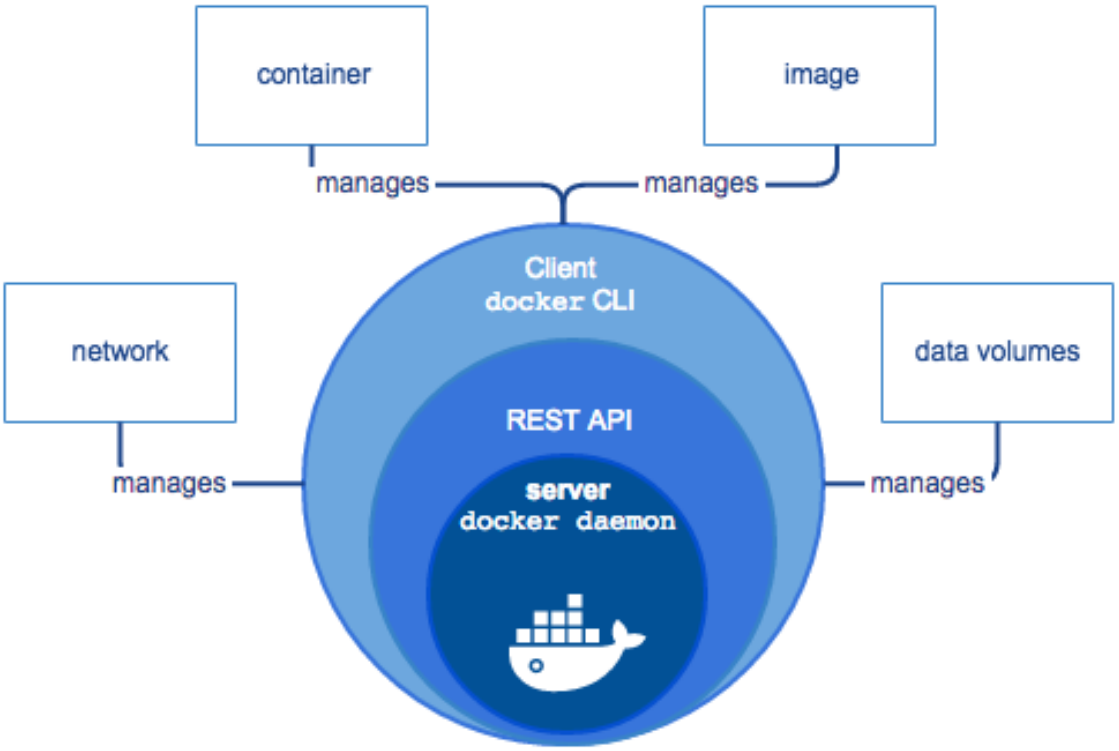

Figura 5.7: Docker engine

I componenti che vanno a costituire l'architettura di Docker sono i seguenti:

- Docker deamon
  Dockerd è in ascolto di chiamate da parte della REST API. E' sua responsabilità far funzionare tutto l'engine della gestione dei container.

- Docker client
  Lo scopo principale del client è interagire con chi sviluppa il contenuto dei container. Questo componente recepisce i comandi da parte dello sviluppatore e li traduce in chiamate per il deamon.

- Docker registries
  E' un registro che consente di conservare le immagini di Docker. Immaginiamo

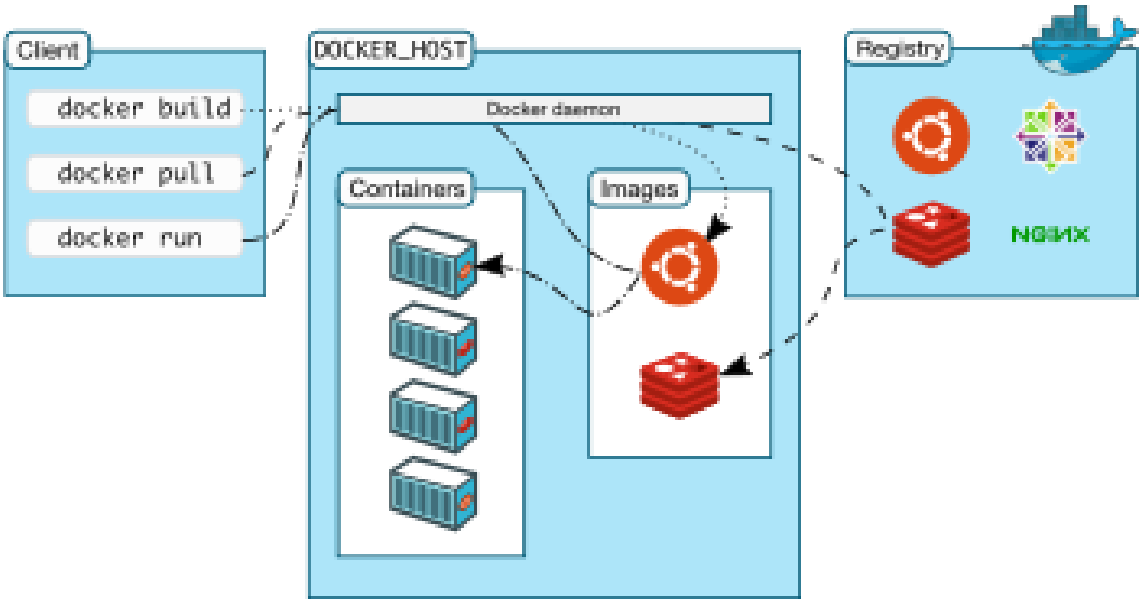

Figura 5.8: Architettura di Docker

che sia interesse di chi li sviluppa poter conservare e mettere in vendita servizi implementati all'interno dei container. Vengono offerte anche delle comode API e uno store nel quale mettere in mostra le proprie immagini.

- docker objects
  In questa categoria rientrano più tipologie di oggetti Docker, come quelli che andiamo ad analizzare qui di seguito.

- Images
  un immagine è un template standard di sola lettura che si utilizza per parametrizzare un nuovo container che si sta per mettere in piedi. Per creare la propria immagine con i parametri personalizzati, si dichiarano nel Dockerfile gli step necessari. Successivamente ad ogni step corrisponderà a un layer del container.

- Container
  Un container è proprio l'istanziazione di una immagine, è infatti definito univocamente dalla propria immagine. E' un elemento che può essere gestito a

piacimento e può essere collegato ad una o più reti, fattore importantissimo per far interagire il proprio contenuto con quello di altri container.

- Services
  I servizi permettono di scalare i container attraverso più Docker deamon. In questo modo si riesce a dare ugualmente l'impressione all'utente di utilizzare comunque un servizio esclusivo e centralizzato.

5.2.5 Un'esuberante

Figura 5.9: Logo di Un'esuberante

Abbiamo appena tessuto le lodi di Docker. Ora però poniamoci alcune domande: come possono essere coordinati, distribuiti e gestiti i container e il loro workload, man a mano che vengono consumate le risorse disponibili nell'infrastruttura sottostante? Come operano i container in un ambiente network multi-tenant? Quale livello di sicurezza propone Docker? E chi decide quale sia il giusto livello di astrazione?

Per far fronte a queste necessità è nato da Google nel 2014 Kubernetes. Esso offre un layer di astrazione per migliorare le performance dei container stessi eliminando molti dei processi manuali coinvolti nel deployment e nella scalabilità di applicazioni containerizzate. Consente di far cooperare opportunamente insiemi di container componendo delle unità logiche utili nelle nostre applicazioni.

Perché utilizzare Kubernetes? Le applicazioni di produzione si espandono su più container, che devono essere distribuiti su più server host. Kubernetes ti offre

le capacità di orchestrazione e gestione necessarie per distribuire i container, in modo scalabile, al fine di gestire i carichi di lavoro. L'orchestrazione di Kubernetes consente di creare servizi applicativi che si estendono su più container, programmare tali container in un cluster, gestirne la scalabilità e l'integrità nel tempo.

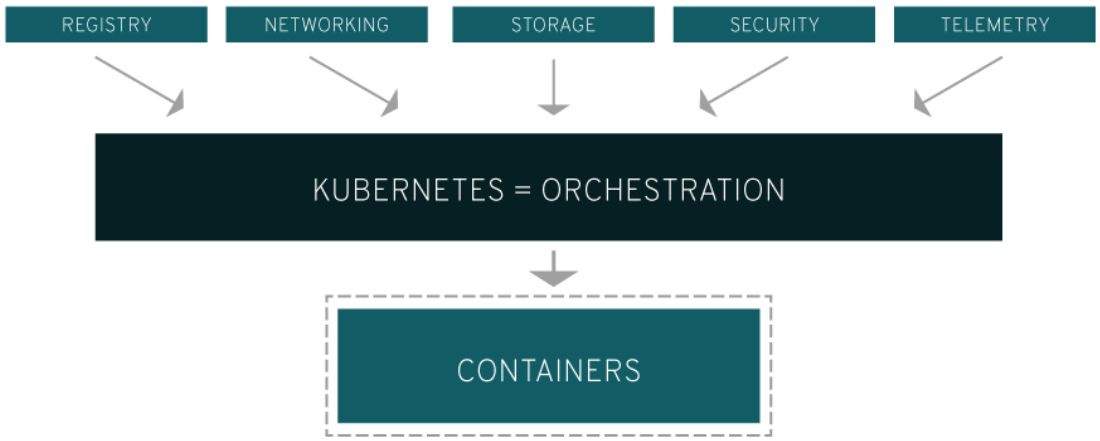

Figura 5.10: Ruolo di orchestratore di Kubernetes

I Pod di Kubernetes, cosa sono? Kubernetes risolve molti dei noti problemi relativi alla proliferazione dei container, raggruppandoli in un Pod. I pod aggiungono quindi un livello di astrazione ai cluster di container. La loro funzione è quella di alleggerire i carichi di lavoro e fornire i servizi necessari, tra cui rete e storage, ai container stessi. Kubernetes agevola, inoltre, il bilanciamento del carico all'interno dei pod e garantisce l'utilizzo di un numero di container adeguato per supportare i tuoi carichi di lavoro.

Architettura di Kubernetes Andiamo a definire quelli che sono i termini di riferimento in Kubernetes:

- Master
  E' la macchina che controlla i nodi Kubernetes. È il punto di origine di tutte le attività assegnate.

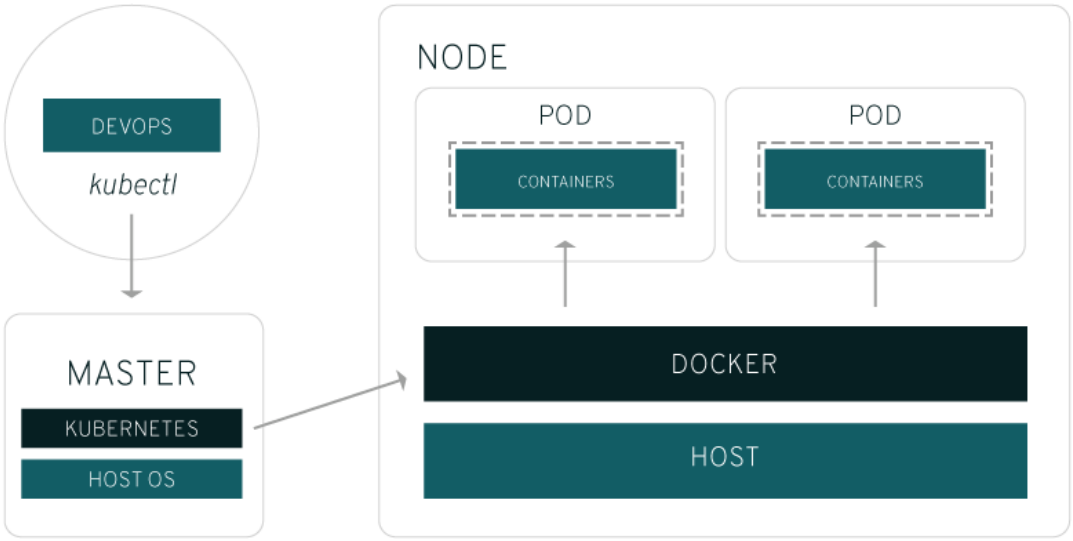

Figura 5.11: Architettura di Kubernetes

- Nodi
  Queste macchine eseguono le attività assegnate richieste. Sono controllate dal nodo master di Kubernetes. Possono essere assimilate con gli host dei container.

- Pod
  Rappresenta un gruppo di uno o più container distribuiti su un singolo nodo. Tutti i container presenti in un pod condividono indirizzo IP, IPC, nome host ed altre risorse. I pod astraggono la rete e lo storage dal container sottostante, consentendo di spostare i container nei cluster con maggiore facilità.

- Servizio
  Questa componente ha la funzionalità di disaccoppiare le definizioni del lavoro dai pod. I proxy di servizio di Kubernetes ricevono automaticamente le richieste di servizio al pod corretto, indipendentemente dagli spostamenti nel cluster, anche nel caso in cui esso sia stato riposizionato.

- Kubelet
  Questo servizio viene eseguito sui nodi, legge i manifest del container e garan-

tisce che i container definiti vengano avviati ed eseguiti.

- Kubctl
  Questo componente si interfaccia direttamente con il programmatore. Presenta una riga di comando per la creazione e gestione dei pod.

E Docker? La piattaforma docker mantiene le proprie funzioni. Quando Kubernetes assegna un pod ad un nodo, il kubelet su quel nodo chiede a docker di lanciare i container specificati. Quindi, il kubelet legge continuamente lo stato di quei container da docker e aggrega le informazioni nel nodo master. Docker invia i container sul nodo, avvia e arresta normalmente i container. La differenza è che un sistema automatizzato con Kubernetes chiede a docker di eseguire queste operazioni, anziché assegnarle ad un amministratore, il quale deve eseguirle manualmente su tutti i nodi, in tutti i container.

5.3 Multitenancy

La multitenancy è un paradigma che, andando a braccetto con il cloud computing, si sta diffondendo sempre di più nel panorama delle architetture software. Esso prevede che una singola istanza del software in questione, situato su un server, sia utilizzato da molti utenti, chiamati tenant. I tenant condividono l'accesso comune al servizio con privilegi differenziati sulle istanze software. L'obiettivo principale è quello di garantire scalabilità al servizio e dare al tenant la percezione di un possesso esclusivo del software.

Vantaggi della Multitenancy

- Costi
  Scalare un'architettura multitenant è più economico, in quanto si interviene solamente in un'unica macchina fisica e acquistare hardware più performante risulta meno dispendioso. Inoltre in questo modo si riduce anche il numero di licenze da acquistare, nel caso siano necessarie, il che può comportare un notevole risparmio.

- Data Mining
  Nel caso sia necessario estrapolare dei dati da tutti i clienti, avere dei tenant tutti sulla stessa macchina rende l'operazione molto più immediata.

- Release Management
  Al rilascio di una nuova versione il processo risulta essere molto semplificato. Ciò viene ovviamente dal fatto che dovendo aggiornare una sola istanza del software su una sola macchina fisica la mole di lavoro è sicuramente minore.

Requisiti Ovviamente un'architettura multitenant ha anche dei requisiti. Sicuramente, nonostante l'unicità del software, si deve garantire un alto grado di "personalizzazione" ai diversi tenant, proprio come se avessero un utilizzo esclusivo del servizio. Da un altro lato ci si aspettano elevati parametri di security, robustezza e prestazioni.

5.4 Monitoring Tools

La centralizzazione portata dal SaaS comporta anche un radicale cambiamento delle modalità di monitoring del sistema. Non è pensabile che una persona possa andare a controllare tutti i log dei servizi e misurare le prestazioni degli stessi manualmente. E' necessario l'utilizzo di appositi tool che hanno come finalità quella di raccogliere questa mole di informazioni, elaborarle e presentarle al personale addetto del provider nel modo più intellegibile possibile. In questa ottica andiamo a presentare due tool utilizzati nel progetto: Prometheus e Grafana.

5.4.1 Prometheus

Figura 5.12: Logo di Prometheus

prometheus è un tool di monitoring open-source nato nel 2012 e scritto in Go. Si adatta sia alle architetture centralizzate che a quelle distribuite. La struttura del tool e pensata per garantire un validissimo supporto nel caso il servizio monitorato abbia dei malfunzionamenti. Vediamo un elenco delle caratteristiche principali.

- Un data model multidimensionale che ha lo scopo di raccogliere tutti i dati
- Un query language flessibile per far fronte a questi dati
- Engine per la rielaborazione dei dati
- Meccanismi sofisticati per collezionare serie storiche
- Supporto per le dashboard

5.4.2 Grafana

Figura 5.13: Logo di Grafana

Grafana consente di avere un portale centralizzato da cui monitorare tutti i servizi. Consente di visualizzare dati, creare degli alert e visualizzare tutte le informazioni in comode interfacce per gli utenti.

5.5 BlueMix Services

Come abbiamo accennato precedentemente, IBM BlueMix è una PaaS che offre molti microservizi utili per il cloud computing. Il vantaggio dei microservizi è proprio

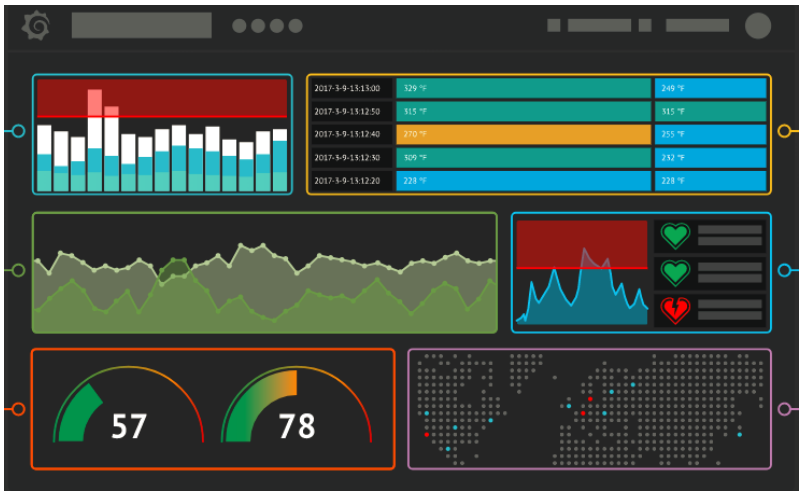

Figura 5.14: Una delle interfacce di Grafana

Figura 5.15: Logo di IBM BlueMix

questo, possono essere utilizzati a piacimento in qualsivoglia progetto. Alcuni dei servizi BlueMix sono stati anche utilizzati nel nostro progetto e altri lo saranno in seguito, con degli sviluppi futuri.

Ne sono un esempio il database DB2 e Kubernetes, che nel nostro caso sono stati proprio integrati nelle loro versioni presenti su BlueMix. E' possibile inoltre che in futuro si adotti un'altro database IBM sempre presente su BlueMix, Cloudant, un database NoSQL nativo cloud.

Capitolo 6

IBM BigFix on SaaS, la progettazione

Per la realizzazione di un prototipo di questo tipo, è necessaria un'attenta progettazione. Quella di cui andremo a parlare a breve è la fase dell'ideazione che va dall'identificazione dei requisiti funzionali, dei bisogni dell'utente, alla definizione dei parametri qualitativi che deve avere il prodotto finale, determinando scelte molto importanti dal punto di vista architetturale. Al termine di questa fase il team inizia gli sprint di development, raffinando sempre di più il modello del servizio che si vuole realizzare. Andiamo ora a vedere quali sono i primi passi che si sono mossi nella realizzazione del progetto.

6.1 Interaction Design

Non si può prescindere dal fatto ce l'interaction design sia la primissima fase da affrontare all'inizio del progetto. Se non si parte dai reali bisogni dell'utente finale, si rischia inevitabilmente di sbagliare strada e realizzare un prodotto che non avrà mai successo sul mercato. Nell'affrontare questo tipo di progettazione l'IBM ha adottato un framework sempre più diffuso nel panorama dello studio dell'usabilità, il Design Thinking.

6.1.1 Design Thinking

Empatia. E' questa la parola chiave della filosofia del Design Thinking. E' un processo creativo che ha come scopo quello di mettere al centro del progetto le necessità dell'utente. Ma proprio per meglio comprendere queste necessità è indispensabile instaurare un rapporto di empatia con gli utenti stessi. Capire i loro reali bisogni, ma anche osservarli durante la loro vita quotidiana per comprendere quelle necessità che non vengono direttamente esternate. Al tempo stesso si vogliono massimizzare le occasioni di feedback cercando di produrre il prima possibile degli output che permettano di comparare più soluzioni alternative. Aumentando per quanto possibile gli input, si riducono le probabilità di fallimento. Ma analizziamo i diversi step che accompagnano un percorso di design thinking.

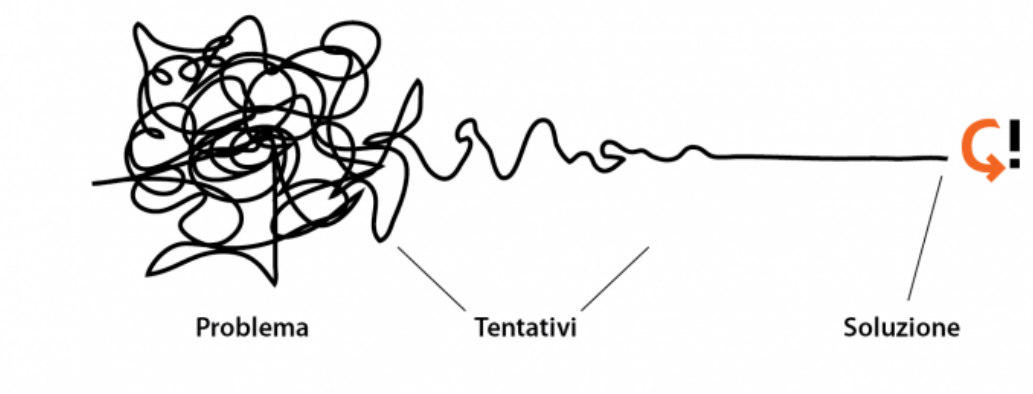

Figura 6.1: Schematizzazione molto basilare del Design Thinking

- Definire il problema
  In questa fase del lavoro è fondamentale osservare, capire le abitudini degli utenti ed immedesimarsi in loro.

- Divergere
  Qui entra in gioco la creatività, cercando di mettere sul piatto il maggior numero di soluzioni possibili, evitando però preconcetti sulle soluzioni e concentrandosi solamente sul problema.

- Testare
  E' necessario ora produrre dei prototipi testabili in modo che si possano comparare le diverse soluzioni e ricevere feedback dagli utenti.

- Convergere
  A questo punto è il tempo di dirigersi verso una soluzione, utilizzando anche la creatività per attuare dei compromessi tra quelle soluzioni parziali che intersecano al meglio i requisiti.

6.1.2 BigFix SaaS Interaction Design

Il nodo cruciale di questa fase del lavoro e stato per noi quello di capire realmente quale fosse il target del nuovo servizio SaaS e quali siano i reali bisogni che possano spingere i clienti ad adottare un prodotto SaaS, siano essi già degli utenti della versione on premise o no. Il problema principale prima dell'avvento del paradigma design thinking era che i requisiti funzionali dei prodotti che venivano realizzati per le aziende erano stabiliti tramite delle contrattazioni svolte tra i progettisti e gli addetti agli acquisti delle aziende clienti, spesso trascurando i reali beneficiari del prodotto, ossia i tecnici dell'azienda cliente. Questo spesso porta a realizzare dei prodotti che non fanno fronte ai reali bisogni dell'utente.

Stakeholder Map L'obiettivo principale di questa fase del lavoro è stata per noi quella di allineare tutti gli interessati, sviluppatori, dirigenti e potenziali utenti, sugli obiettivi del progetto SaaS. Per fare ciò si è fatto uso della Stakeholder Map, un artefatto che raffigura per l'appunto tutti gli Stackeholder interessati alla realizzazione del progetto. Possiamo notare come gli input vengano da figure dirigenziali, che dettano le direttive aziendali, e da clienti del panorama cloud. E' inoltre necessaria una stretta interazione tra il team di sviluppo e il Security Operation Team, ovvero il team che dovrà garantire la manutenzione del servizio SaaS, monitorando le prestazioni del servizio e intervenendo se necessario.

User Research Il coinvolgimento della figura dell'utente finale è avvenuto fin dalla progettazione. Questo è stato fatto tramite interviste strutturate, ma anche osservando gli utenti nella loro routine lavorative, cercando di captare necessità e

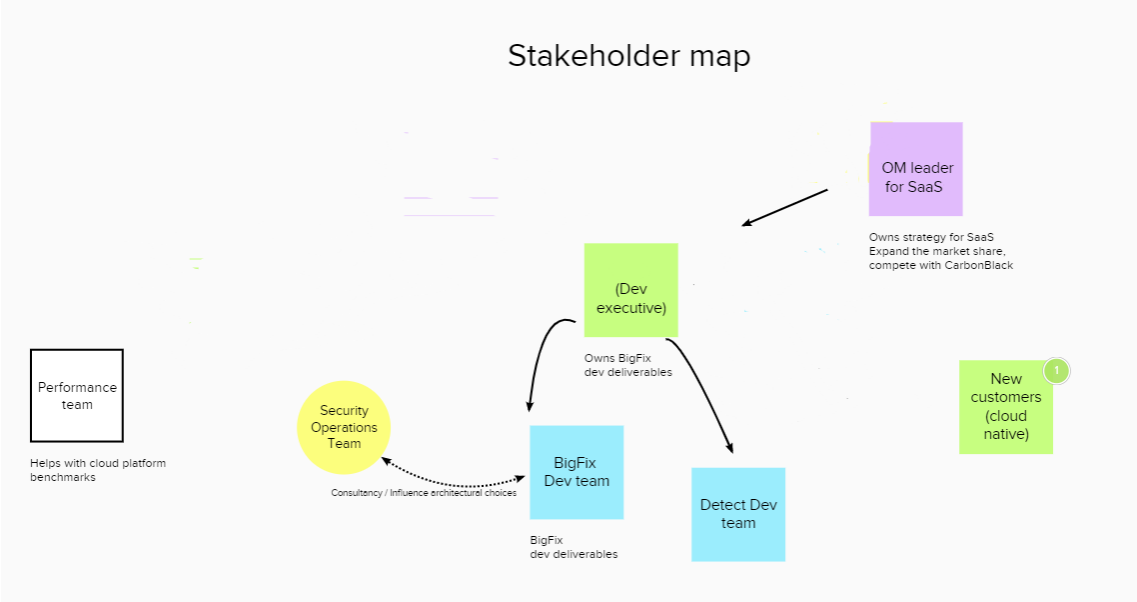

Figura 6.2: BigFix on SaaS Stakeholder Map

frustrazioni che vogliono essere eliminate e annotarle. L'obiettivo di fondo di questa fase del lavoro era quello di instaurare un rapporto di empatia tra gli utenti e chi realizza il progetto.

Nascita delle personas A questa punto occorreva fare un'operazione di astrazione. Cercare di identificare degli elementi chiave dal lavoro precedente e impersonare i bisogni e le caratteristiche scoperte in delle personas. Le personas sono personaggi fittizi che vengono creati per rappresentare i diversi tipi di utenti in base alle loro caratteristiche comportamentali. Vengono utilizzate per creare degli scenari e capire meglio il target del lavoro che si sta per compiere. Nel nostro caso sono state individuate le seguenti personas:

- Rick - BigFix Operator
  Rick è l'amministratore che utilizza BigFix nella sua versione SaaS. Vuole poter gestire gli endpoint della sua azienda come farebbe con la versione on premise, distribuire contenuti e forzare le policy aziendali. usa le applications di BigFix.

- James - Content Creator
  James crea i contenuti per BigFix, crea Fixlet e task ad hoc per la propria azienda e crea pacchetti da deployare su diversi endpoint aziendali.

- Scott - BigFix Architect
  Scott è l'architetto dell'azienda cliente. Si occupa di installare la suite BigFix, che nel caso della versione SaaS risulta essere molto più agevole. Deve stabilire quale sia l'architettura azienda aziendale, relay e la rete di agent.

- Rafael - Security Analyst
  Rafael è la figura che si occupa di controllare che gli endpoint rispettino le policy aziendali e può mandare dei messaggi per sollecitare l'attuazione della compliance.

- Lucy - IT Manager
  Si occupa della gestione dell'infrastruttura IT dell'azienda. Ha bisogno di accedere a molti contenuti dell'operator e si interfaccia spesso con Rick.

- Hugo - OPS engineer
  Hugo è un dipendente IBM che si occupa della manutenzione del servizio SaaS.

Empathy Map A questo punto è stato necessario definire gli aspetti caratteriali dal punto di vista lavorativo delle personas appena individuate. Questo si formalizza attraverso una Empathy Map per ogni peronas che si vuole analizzare. In questo artefatto vengono poste al centro e ne vengono appuntate le peculiarità. Abbiamo scritto cosa fa durante la giornata e le sue necessità, ma anche dopo un'analisi quelli che possono essere i suoi sentimenti durante lo svolgimento dei task quotidiani. tra le caratteristiche che abbiamo cercato di delineare nei personaggi ci sono:

- Profilo professionale
- Attività
- Attitudini
- Bisogni

- Obiettivi
- Cosa possiamo fare per aiutarlo

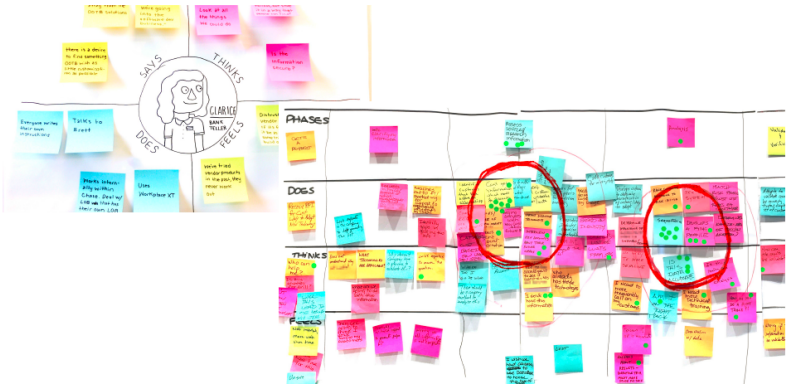

Figura 6.3: Esempio di Empathy Map

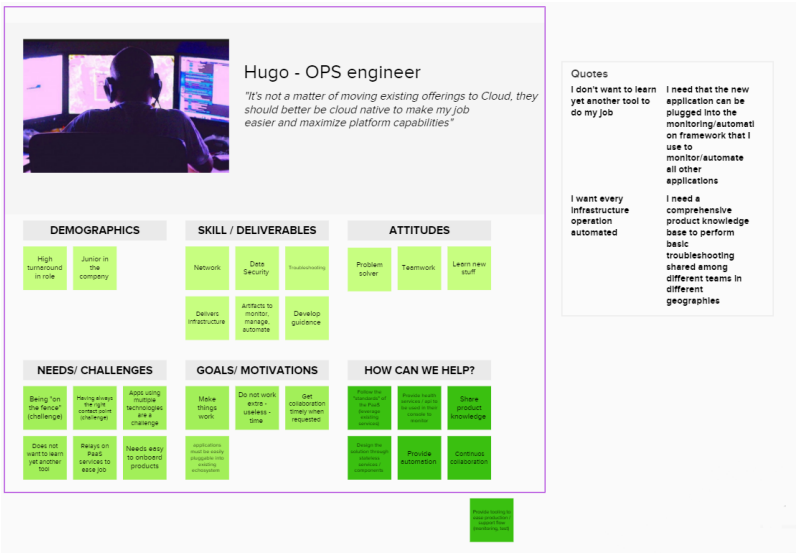

Figura 6.4: Empathy Map di una delle nostre personas

Scenario Map In questo nuovo elaborato le personas con le quali siamo entrati in confidenza nella fase precedente iniziano ad essere inserite nel loro ciclo di vita quotidiano. Con lo scenario viene descritto un flusso di lavoro tipico del personaggio

in questione. Nel succedersi degli stages, ovvero i passi in cui si divide lo scenario, si annotano quelli che possono essere i sentimenti dell'utente, cercando di individuare gli elementi di frustrazione e le opportunità per intervenire nella progettazione risolvendo i problemi dell'utente tipo. Vediamo qui di seguito una scenario map per Rafael.

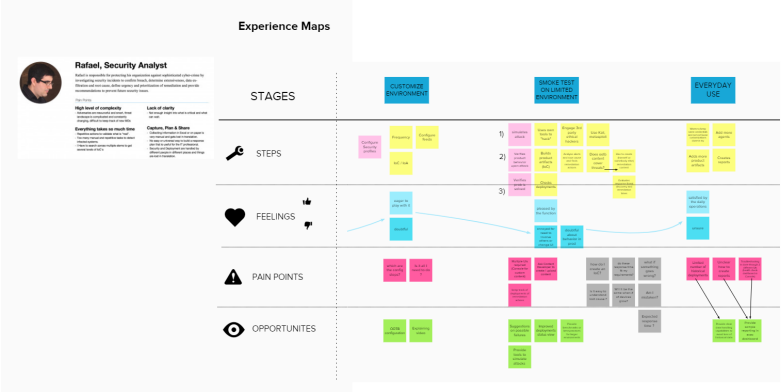

Figura 6.5: Scenario che rappresenta il primo utilizzo del servizio da parte di Rafael, il Security Analist

Vediamo qui di seguito uno schizzo riassuntivo degli step del Design Thinking che abbiamo adottato:

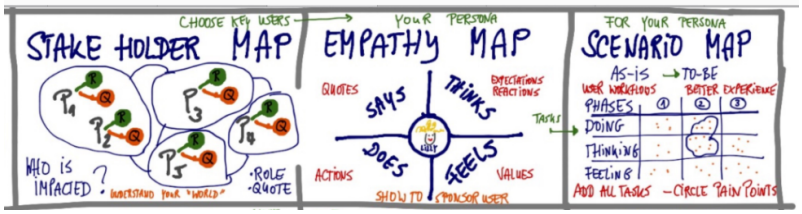

Figura 6.6: Step del Design Thinking adottati nel nostro progetto

Gli scenari in questo modo individuato vanno a rappresentare la base per la stesura delle Epiche. Queste, come abbiamo descritto nella sezione 2.3 verranno poi raffinate con la stesura delle Storie e infine divide nei Task implementabili dagli sviluppatori.

Figura 6.7: Relazione tra gli scenari individuati in questa fase e le Epiche del framework SCRUM

6.2 Requisiti Non Funzionali

Abbiamo già parlato nel capitolo 4 di quali sono le nuove problematiche alle quali una SaaS application deve far fronte. Ovviamente nel mio lavoro di tesi questo aspetto è stato un argomento cruciale delle prime fasi del lavoro. Soddisfare questo tipo di requisiti comporta infatti fare scelte architetturali molto impattanti e in quanto tali occorre definirle prima possibile nel design di un sistema software.

6.2.1 Dependability

Il servizio di BigFix SaaS è stato progettato per garantire, quando sarà in produzione, un'availability che si mantenga sempre su valori superiori al 99. Ovviamente si prevedono carichi di utilizzo che possono essere anche molto elevati. La suite di BigFix è utilizzata contemporaneamente da clienti di tutto il mondo, alcuni dei quali possiedono una rete di endpoint composta da un numero considerevole di nodi. Tutto ciò può portare a picchi di carico molto elevati nonostante i quali il servizio deve continuare a essere disponibile con prestazioni sopra delle soglie minime di accettabilità.

Microservizi e container Come abbiamo potuto osservare nei capitoli precedenti, l'adozione di microservizi e container è un must per i servizi cloud. Grazie a questa scelta possiamo garantire agli utenti di BigFix SaaS un'alta Dependability, fattore fondamentale nel contesto della security aziendale in cui si va a calare questa suite di prodotti. I microservizi di BigFix, infatti, verranno replicati tramite i container in data center IBM in tutto il mondo, ciò potrà garantire anche tolleranza ai guasti che possono presentarsi. Il grado di replicazione dei diversi microservizi sarà ovviamente proporzionale all'importanza del microservizio stesso. Ci saranno ovviamente dei microservizi con dei ruoli più centrali di altri.

Rolling Update

Un'altro aspetto critico nel garantire un'alta availability è quello dell'aggiornamento del servizio. Facendo un paragone con i servizi SaaS che utilizziamo quotidianamente per consultare la posta elettronica, notiamo che non assistiamo mai a fenomeni di mancanza del servizio quando il prodotto si aggiorna, ma, all'occorrenza, troviamo già il prodotto nella sua versione aggiornata. Vogliamo che questo comportamento si verifichi anche con la suite SaaS di BigFix e per questo occorre attuare una politica di Rolling Update. Silentemente, vengono aggiornate a turno tutte le repliche dei microservizi interessanti dall'aggiornamento. Nel fare ciò però, l'esperienza utente non risente di peggioramenti, in quanto le repliche che rimangono in servizio garantiscono l'efficienza del servizio.

Utilizzo di DB2

Anche la persistenza dei dati può risultare essere un elemento critico per la dependability. Occorre uno strumento che garantisca l'integrità dei dati, la resistenza ai guasti con adeguate misure di ripristino e soprattutto la riservatezza dei dati che, in un contesto come la security aziendale, possono essere molto sensibili. Si è scelto di utilizzare come DBMS DB2, un database relazionale prodotto da IBM. Una peculiarità di questo prodotto è la HADR (High Availability and Disaster Recovery). Diamo un'occhiata all'architettura di DB2 per capire di cosa si tratta.

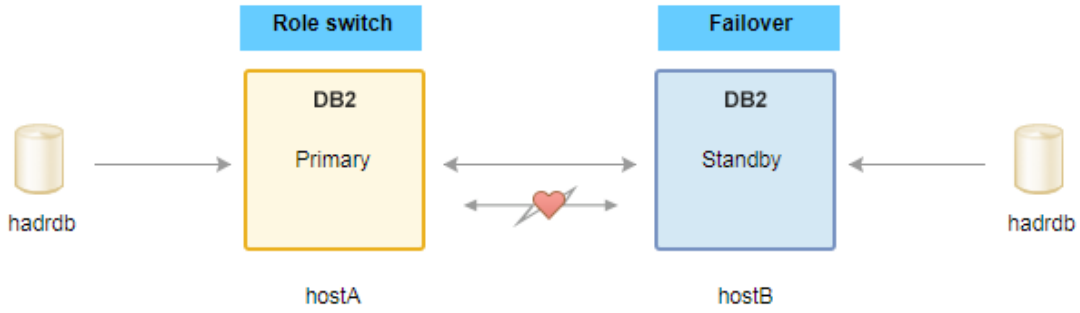

Figura 6.8: Architettura del DBMS IBM DB2

DB2 replica tutto il contenuto del suo database, chiamato Primary Database, in un secondo database detto Standby Database, il quale svolge anche il ruolo di backup. I dati di questi due sono consistenti e vengono sincronizzati costantemente. Qualunque malfunzionamento del database principale normalmente comporterebbe dei tempi di non availability più o meno lunghi. Con questa architettura HADR, invece, nel momento in cui il Primary Database presenta un guasto, lo Standby Database assume il suo ruolo (Failover) finché il database primario non torna disponibile e, a qual punto, i due database tornano a svolgere il loro compito originario (Role switch). Una prerogativa importante però è che i due database risiedano in due data center distinti, o comunque provengano da due fonti di energia distinte nel caso si trovino nello stesso luogo geografico, per evitare che dei guasti possano colpirli entrambi.

6.2.2 Scalabilità

Per quanto riguarda la Scalabilità ci prefiggiamo di garantire le stesse specifiche del prodotto in versione on premise, quindi di supportare fino a 250.000 endpoint per server. Il soddisfacimento di questa specifica, nel contesto SaaS, sposta l'attenzione ovviamente sul nuovo concetto di server, ossia una serie di microservizi distribuiti che svolgono le funzionalità che nella versione on premise era svolta dal server presso il client. Ancora una volta sta nella ridondanza dei microservizi la chiave per garantire la scalabilità prefissata.

6.2.3 Monitoring

Sotto l'aspetto del monitoring ci siamo dovuti scontrare con una nuova complessità nel saper monitorare un servizio così diffuso come quello di SaaS. La necessità è quella di sostituire l'intervento umano nella consultazione dei log di tutti i servizi. Il requisito che abbiamo è quello di analizzare i risultati, saper effettuare delle medie e calcolare dei picchi di parametri come il throughput o la latenza. Vogliamo infine che questa mole di dati fosse facilmente consultabile agli occhi di chi effettua la manutenzione del prodotto, magari sotto forma di grafici facilmente intellegibili. Per soddisfare queste necessità abbiamo individuato i tool Prometheus e Grafana che si sono rivelati molto utili nelle fasi successive al deployment, come vedremo in seguito.

6.3 Gap con il prodotto on premise

La natura di un servizio SaaS porta con se alcune differenze strutturali importanti con il prodotto già esistente. La modalità di fruizione del prodotto è completamente diversa dal prodotto on premise infatti e gli accorgimenti sono da prendere subito in considerazione in quanto impattano pesantemente sulle scelte architetturali.

Introduzione della multitenancy Uno di questi è sicuramente la multitenancy. Nel modello SaaS può capitare che sulla stessa macchina fisica risiedano più server di clienti diversi. Dalla prospettiva utente però si deve dare l'impressione di un possesso esclusivo del server tramite strategie di multitenancy. E' di fondamentale importanza che un cliente non entri in contatto con dati afferenti al server di altre organizzazioni, anche se queste risiedono sullo stesso server fisico. Tra gli accorgimenti attuati c'è la modifica della modalità di archiviazione dei dati, permettendo di filtrare i dati appartenenti al tenant corretto e speciali privilegi di utilizzo dei servizi server.

Introduzione dei microservizi L'introduzione dei microservizi è un elemento centrale della conversione a SaaS. Per attuarla è necessario un attento percorso di refactoring del codice del prodotto, suddividendolo in servizi coesi che possano rappresentare delle entità separate che cooperino tra loro.

6.4 Scelta dei tool e dei servizi da utilizzare

Nel mio caso mi sono mosso in un ambito, quello del cloud computing, si privo di una consolidata storia alle spalle, ma anche ricco di continui nuovi contributi. Infatti compaiono sempre nuove tecnologie dedicate esclusivamente all'ambiente cloud, molte delle quali open-source con frequentissimi contributi da parte degli sviluppatori.

Trattandosi della realizzazione di un prototipo SaaS, l'attenzione principale non poteva che ricadere sui container e sui più diffusi trend in ambito cloud. Si è scelto di adottare l'utilizzo dei container. Come largamente trattato nella sezione 5.2 essi rappresentano una tecnologia fondamentale per realizzare un servizio SaaS. Avendo scelto di adottare i container occorreva quindi scegliere una piattaforma che automatizzi il deployment dei container in maniera efficace. Da questo punto di vista la scelta è ricaduta su Docker, una scelta quasi obbligata vista l'alta competitività del tool e l'affermazione ormai incontrastata. In questo modo ci si alleggerisce dall'overhead di utilizzare un'intera macchina virtuale per ogni container in quanto tutti i container sulla stessa macchina fisica condividono lo stesso sistema operativo, ma al tempo stesso sono isolati.

A questo punto però bisogna immaginare uno scenario in cui il servizio che sono andato a realizzare e scomposto in un numero piuttosto elevato di container, alcuni andranno creati in delle situazioni specifiche, altri eliminati o modificati. Per queste funzionalità l'IBM ha come direttiva quella di utilizzare Kubernetes, del quale abbiamo parlato anche nella sezione 5.2. Tramite questo tool sono in grado di orchestrare i container e controllarne il flusso di vita tramite i pod.

6.5 Definizione Architetturale

La definizione di un'architettura di un prodotto così complesso è guidata da un'attenta analisi dei requisiti appena descritti. Occorre stabilire quali siano i componenti da implementare, altri da riutilizzare e alcuni da adattare. Inoltre occorre stabilire

come interfacciarsi con nuove tecnologie che devono essere opportunamente inserite nel contesto di applicazione.

6.5.1 Viste architetturali

Vista sull'architettura fisica e sui container

Questa è la più significativa delle viste architetturali. Da questa immagine possiamo chiaramente evincere la struttura a microservizi del SaaS di BigFix. Ogni microservizio è deployato in un container. Volutamente non si sono volute raffigurare macchine che ospitano i container per rendere il senso di portabilità dei container stessi. Potrebbero trovarsi tutti sulla stessa macchina fisica come in parti del mondo differenti. Tutti i container, come abbiamo sottolineato precedentemente, adottano Docker come gestore. Da questa architettura notiamo anche la presenza di nuovi componenti:

- Container Master
  Questo container ospita il componente Kubernetes master, il quale ha il compito di governare il comportamento di tutti gli altri nodi container. Questi ultimi rappresentano infatti dei Kubernetes node.

- BigFix Automation Manager
  Questo componente è fondamentale per tutti i processi di gestione, che vedremo in seguito. SI immagino ad esempio scenari come l'onboarding di un nuovo cliente o la distribuzione di una versione del servizio.

- Load Balancer
  Come suggerisce il nome stesso è il componente che ha la responsabilità di ripartire il carico di lavoro sui microservizi presenti sui container, alcuni di loro replicati. Una buona gestione del workload è fondamentale nelle situazioni di picco di utilizzo da parte dei clienti per garantire l'availability del prodotto.

Viewpoint dell'utente utilizzatore

Da questa immagine invece cerchiamo di ricostruire la struttura tradizionale di BigFix rivisitandola però in ottica SaaS. Le componenti del server e dei relay non sono

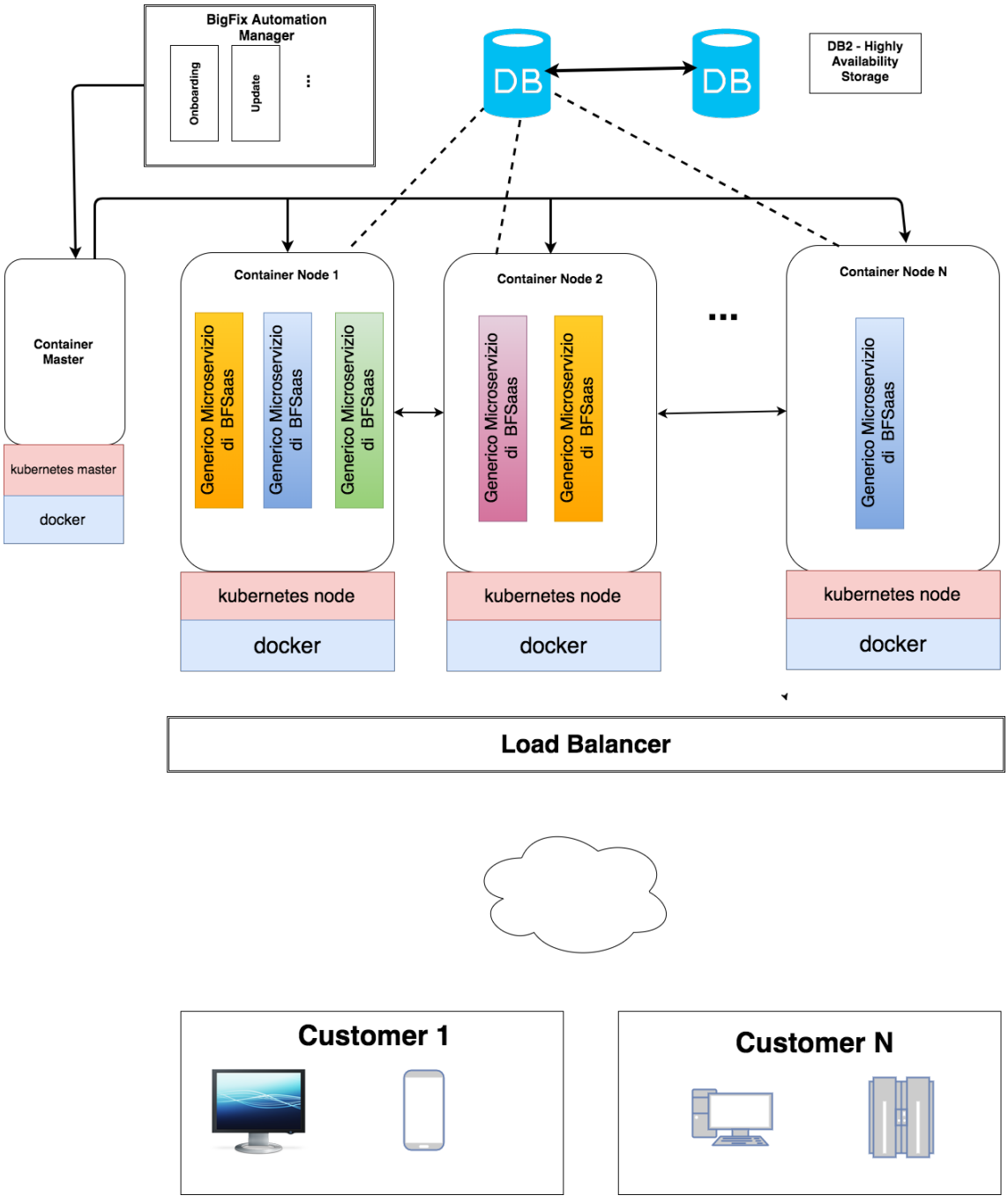

Figura 6.9: Architettura di BigFix SaaS

più macchine fisiche facenti parte dell'infrastruttura del cliente. Il server e il relay sono completamente sotto il controllo del provider e sono composti da opportuni

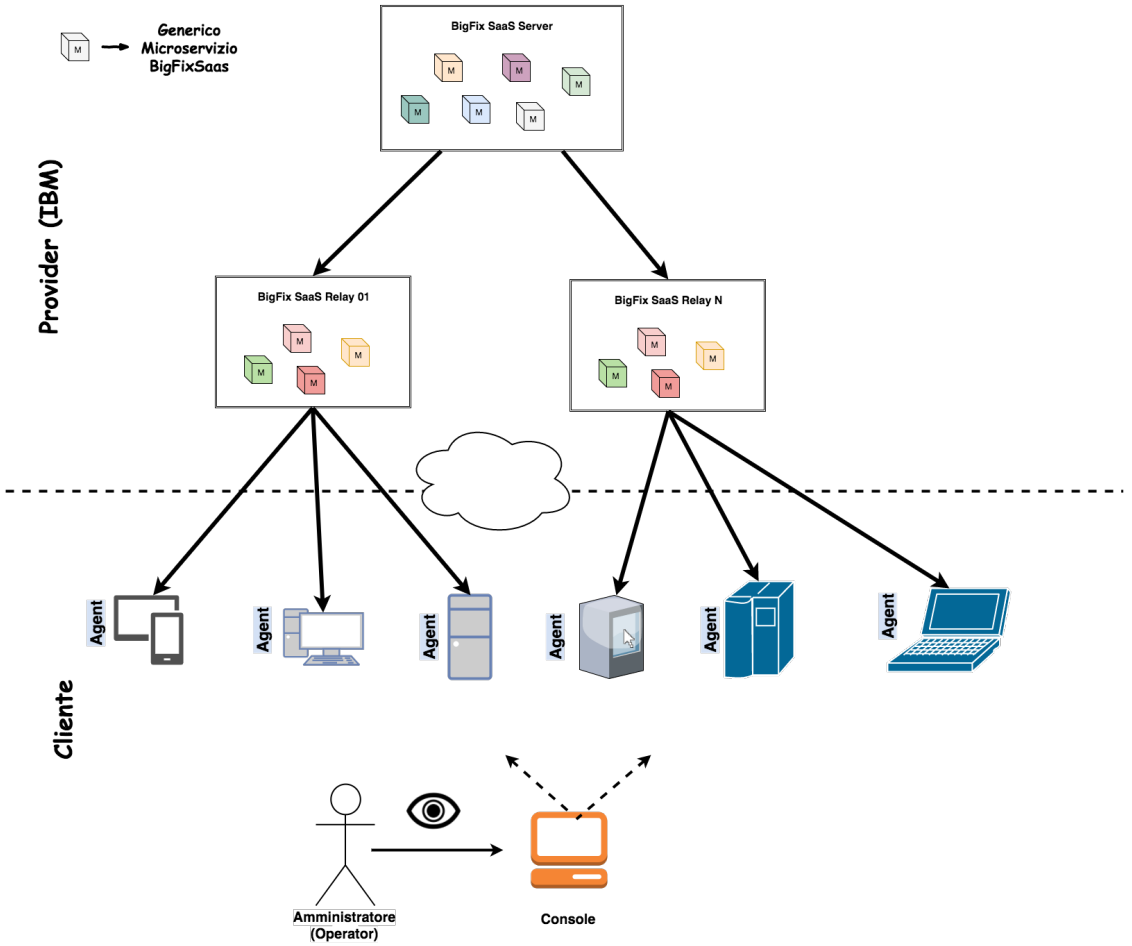

Figura 6.10: Architettura di BigFix SaaS

microservizi che cooperano per garantire le funzionalità del servizio. L'unico componente che rimane fisicamente dal cliente è l'agent, che è presente ovviamente in ogni dispositivo della rete di endpoint.

6.6 Definizione dei processi di gestione

Una delle novità nel distribuire un servizio SaaS riguarda i processi di gestione. Analogamente alla progettazione architetturale, anche i processi di gestione vanno ben definiti e, successivamente, messi in opera.

6.6.1 Novità rispetto al prodotto già esistente

Onboarding Si pensi ad esempio allo scenario dell'onboarding di un nuovo cliente. Il sistema deve garantire che un processo completamente automatico possa portare il nuovo utente dal click sulla form di registrazione a tutto il servizio BigFix pronto all'utilizzo in pochi minuti, al massimo ore. Gli utilizzi del servizio hanno delle dinamiche molto diverse rispetto allo scenario on premise.

Monitoring Anche il monitoring delle attività e delle prestazioni richiede nuovi strumenti perché sposta l'attenzione sull'utilizzo dei container e sul funzionamento dei microservizi. Abbiamo definito dei processi specifici (Prometheus) che sappiano monitorare i pod di Kubernetes e, conseguentemente, i container.

6.6.2 Disaster Recovering

Dal punto di vista del disaster recovery con un servizio SaaS abbiamo notevolmente diminuito il rischio di verificarsi di situazioni di criticità, in quanto le infrastrutture fornite dal provider sono molto più efficienti da questo punto di vista. Al tempo stesso però, nel caso del verificarsi di qualche malfunzionamento il pericolo è notevolmente maggiore. A quel punto infatti potrebbero essere coinvolti tutti i clienti forniti dal servizio. In uno scenario del genere abbiamo progettato misure di emergenza che prevedono a redistribuzione del carico tra i microservizi ma anche l'attuazione del recovery di emergenza di db2.

Capitolo 7

IBM BigFix on SaaS, l'implementazione del prototipo

7.1 Relazione tra il prodotto BigFix e i container

7.2 Generazione delle immagini

7.3 Automazione del Deployment

7.3.1 Releases vs DevOps vs Continuous Delivery

7.3.2 Bash Scripting

7.3.3 Jenkins

7.3.4 UrbanCode

7.3.5 Ansible

7.3.6 Scenario di onboarding di un nuovo cliente

7.4 Automazione del Testing

7.4.1 Functional Test

JUnit

JUTAA

7.4.2 Security Test

AppScan

Image Compliance

7.4.3 Performance Test

7.4.4 Penetration Test

7.4.5 Rielaborazione degli output del testing

Capitolo 8

Conclusioni

8.1 Sviluppi futuri

8.2 Considerazioni

Capitolo 9

Ringraziamenti

Appendice A

Tecnologie Utilizzate(template di prova - ANCORA DA SCRIVERE)

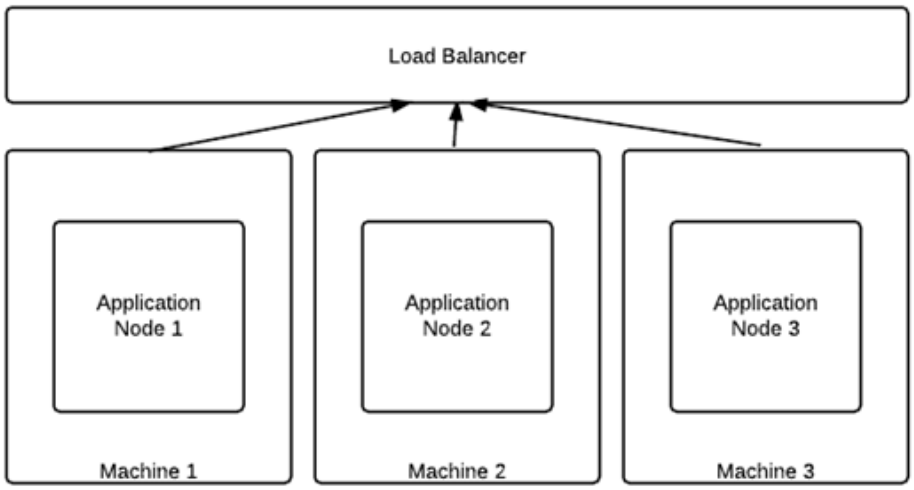

Figura A.1: Una schematizzazione della distribuzione del carico in una architettura Cloud

A.1 Linguaggi di programmazione

- PHP 5.4.7
  http://www.php.net/;
- Javascript
  http://www.w3.org/standards/webdesign/script;

A.2 Linguaggi di Markup e Stile

- HTML4/HTML5;
- CSS/CSS3;

A.3 Framework

- Smarty Template Engine
  http://www.smarty.net/;
- JQuery
  http://jquery.com/;
- JQueryUI
  http://jqueryui.com/;
- beContent
  http://www.becontent.org/;

A.4 Ambiente di Sviluppo

A.4.1 Eclipse

Per Eclipse sono state utilizzate due versioni differenti, la 4.2.2 in ambiente Windows e la 3.8.0 in ambiente Ubuntu/Linux
http://www.eclipse.org/
Inoltre è stato utilizzato il pacchetto

- PHP Development Tools 3.1.1
  http://projects.eclipse.org/projects/tools.pdt;

A.4.2 Piattaforma Web

XAMPP

http://www.apachefriends.org

- Apache Web Server ver. 2.4.3
  http://httpd.apache.org/;
- MySql Database Management System ver. 5.5.27
  http://dev.mysql.com/;

A.4.3 Browser Testing

Mozilla Firefox

- Firebug ver 1.11.2
  http://getfirebug.com/
  - Plug-In Validator ver. 0.0.6
    https://addons.mozilla.org/it/firefox/addon/validator/;
  - Plug-In Google Page Speed ver. 2.0.2.3
    https://developers.google.com/speed/pagespeed/?hl=it-IT;

Google Chrome

- Strumenti per gli sviluppatori integrati

Responsive Testing

- Viewport Resizer- Responsive Design Bookmarklet
  http://lab.maltewassermann.com/viewport-resizer/ ;

Dedica a fine pagina