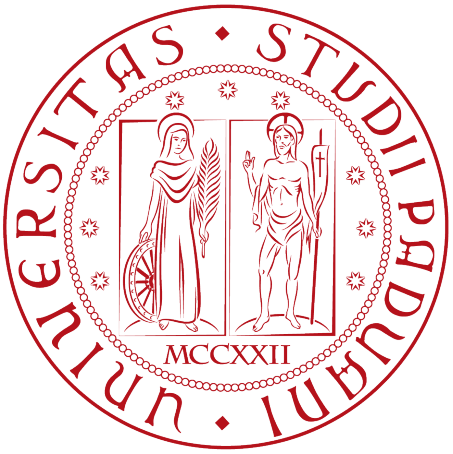
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Civita"

Corso di Laurea in Informatica



Containerizzazione di Malware Dashboard

Relazione finale di stage

Relatore

Prof. Tullio Vardanega

Laureando

Andrei Petrov

Anno Accademico 2016-2017

Sommario

//TODO

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

L'Azienda

1.1 IKS

IKS *(Information Knowledge. Supply)* è un'azienda padovana fondata, dall'attuale Amministratore Delegato Paolo Pittarello, nel 1999.

Nell'insieme, IKS unisce figure di alto profilo con lo scopo di proporre soluzioni innovative alle richieste di mercato dell'Information and Communication Technology (ICT) sia italiano che estero. Le soluzioni offerte interessano in particolare gli ambiti della sicurezza, dell'infrastruttura e della governance Information Technology (IT).

L'azienda è in continua ricerca tecnologica. Investendo sulla formazione del proprio personale, IKS si impegna di portare solo valore aggiunto al business dei propri clienti. Inoltre, l'Azienda crede fortemente nell'innovazione come strumento verso un ambiente digitale comune, Agile (metodologia) e totalmente disponibile.

Il quartier generale aziendale è a Padova. Inoltre, IKS possiede uffici anche nelle seguenti città: Roma, Milano e Trento.

A partire dallo scorso anno (2016), IKS SRL, Kirey SRL, Insirio SPA e System Evolution SRL hanno fondato il Gruppo Kirey. L'obiettivo comune delle quattro aziende è l'unione delle competenze complementari e garantire un portfolio completo di soluzioni ai clienti attuali e futuri.

La creazione del Gruppo Kirey è stata guidata dalla Synergo SGR., società di *private equity*. Il presidente del nuovo Gruppo commerciale è Vittorio Lusvarghi.

A seguito della creazione del Gruppo, IKS SRL e le restanti tre realtà aziendali hanno conservato la propria struttura di governance e management, con il fine di garantire la propria continuità gestionale.

1.2 Profilo dell'azienda

1.2.1 Servizi e prodotti offerti

Nel corso degli anni, IKS si è fatta notare per gli enormi contributi innovativi nell'ambito della sicurezza informatica. Tuttavia, essa non è limitata a questo ambito. Infatti, gli altri ambiti di applicazione sono: infrastruttura e governance IT.

Di seguito vengono presentati i servizi offerti da IKS per ciascun ambito di applicazione:

* **IT Security**

  - **Risk analysis e vulnerability assessment**
    È importante garantire la sicurezza dell'infrastruttura informatica nel suo complesso. A questo scopo, IKS offre un servizio orientato alla ricerca di eventuali vulnerabilità e analisi dei rischi a esse collegate;

Figura 1.1: Visione a processo della gestione del rischio.Immagine tratta da: http://bit.ly/2rh3V0A.

  - **Audit management**
    Le aziende di continuo sono sottoposte a controlli di vario genere; il loro scopo è l'accertamento della regolarità delle aziende con: certificazioni, normative, bilanci ed ecc. IKS offre un servizio di supporto per le aziende con il fine di agevolare le attività di *auditing* ed eventualmente per migliorare i loro processi interni;

Figura 1.2: Flusso di lavoro durante lo svolgimento di un *audit*. Immagine tratta da: http://bit.ly/2rdFhfv.

  - **Difesa perimetrale**
    Sempre in ambito della sicurezza è importante prendere le giuste misure per garantire a priori uno specifico livello di sicurezza e limitare a zero le intrusioni dall'esterno di un'infrastruttura IT aziendale. A questo scopo, IKS offre un'insieme di soluzioni orientare al monitoraggio degli accessi a sistemi aziendali, dei permessi sulle operazioni che un utente può effettuare, e molto altro;

Figura 1.3: Visione grafica del concetto di difesa perimetrale. Immagine tratta da: http://bit.ly/2s834O2.

* **IT Infrastructure**

  - **Business continuity**
    In ambito bancario, le infrastrutture informatiche sono molto complesse. La manutenzione delle infrastrutture informatiche non è semplice. La sfida più difficile è garantire che questi sistemi siano operativi al 100%. Una simile percentuale nella pratica è impossibile. IKS con il proprio gruppo di esperti sono alla continua ricerca di soluzioni per incrementare la percentuale di continuità operativa di questi sistemi. Infatti, le soluzioni offerte dall'azienda sono orientate nel concreto all'infrastruttura del cliente richiedente supporto;

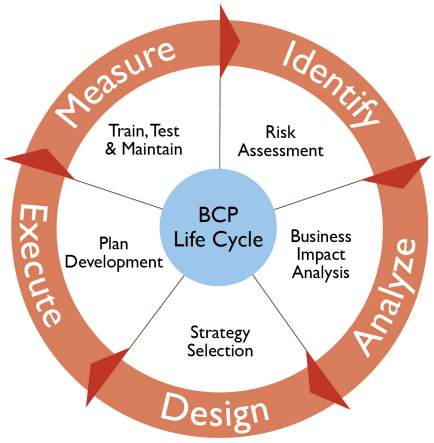


Figura 1.4: Visione del ciclo di vita del processo di *business continuity*. Immagine tratta da: http://bit.ly/2qvCmgP.

  - **Virtualization technology**
    Ogni prodotto software di business per portare valore aggiunto deve essere eseguito. Eseguire un prodotto software per server fisico richiede la disponibilità di un cospicuo numero di server. A questo scopo la tecnologia di virtualizzazione permette la creazione di server virtuali che eseguono

programmi e questi vengono eseguiti da server fisici. I benefici di una simile infrastruttura è l'ottimizzazione delle risorse di calcolo, agilità di gestione e sicurezza. Alcune delle soluzioni di virtualizzazione offerte da IKS sono: VMWare, RHEV ed ecc. Un'evoluzione della tecnologia di virtualizzazione è il *Cloud*. In questo ambito, IKS propone soluzioni di migrazione e supporto verso il Cloud dell'infrastruttura IT classica di un'azienda;

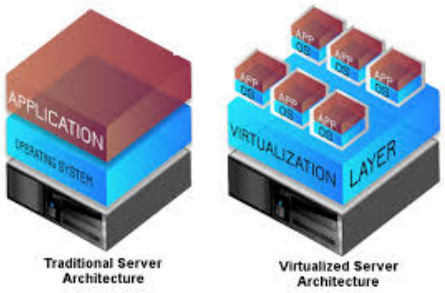


Figura 1.5: Vista a confronto: ambiente server bare metal e virtualizzato. Immagine tratta da: http://bit.ly/2qvtLLk.

* **IT Governance**

  – **Service management**
  Un servizio informatico di business, a causa della sua criticità, richiede costante attenzione. Il monitoraggio del servizio informatico presenta la necessità di enormi investimenti economici. IKS offre piani di gestione per soddisfare anche i più esigenti clienti;

Figura 1.6: Visione della gestione di servizio in prospettiva del Framework ITIL. Immagine tratta da: http://bit.ly/2qvNryk.

  – **Application and performance monitoring**
  Ogni prodotto software ha il proprio specifico ciclo di vita. Concluso il ciclo

di sviluppo, il prodotto è rilasciato in produzione. La seconda parte del ciclo di vita di un prodotto software è la manutenzione. Il monitoraggio di un applicativo è importante per avere una costante visione dello stato del prodotto e prevenire eventuali esigenze di manutenzione generica oppure di basso profilo a livello di codice sorgente. In questo dominio, grazie a partnership strategiche, IKS offre soluzioni mirate a garantire la miglior possibile esperienza di monitoraggio applicativo;

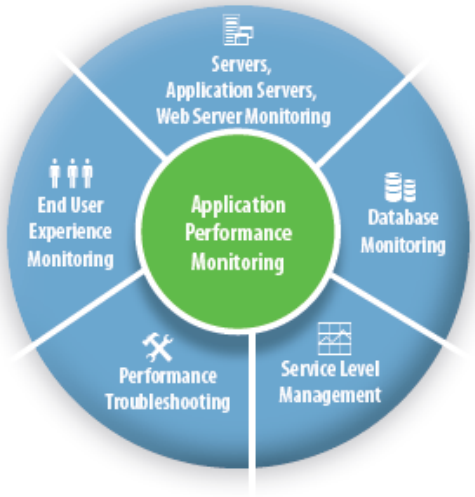


Figura 1.7: Oggetto del monitoraggio sono: gli utenti, i server, le applicazioni ed ecc. Immagine tratta da: http://bit.ly/2v3MuE2.

- **System and networking management**
  Gestire sistemi e reti informatiche è un compito complesso. L'utilizzo di strumenti adeguati permette di semplificare il lavoro e garantisce un stato consistente del sistema nel tempo. Le soluzioni che IKS offre sono orientate alla flessibilità e facilità d'uso dei prodotti offerti in questo contesto;

* **Innovation & Project**

  - **Architetture applicative distribuite**
    I sistemi informatici diventano sempre più di natura distribuita. IKS offre in questo ambito soluzioni architetturali orientate a microservizi, utilizzando le ultime tecnologie orientate alla containerizzazione e orchestrazione di container;
  - **Sviluppo di applicazioni cloud native**
    È comune sentire parlare di cloud. Le classiche architetture applicative non riescono a beneficiare della flessibilità del cloud, perché in organizzazione e struttura non sono scalabili e sono difficilmente modularizzabili. Applicazioni che vengono gestite nel complesso come un'unica unità prendono il nome di monolite. La diretta conseguenza di una simile organizzazione è il carattere statico e poco flessibile dell'applicazione. Paradigmi nuovi, per la messa in

esercizio di applicazioni, mancano l'intigrazione con architetture software tradizionali. Per questo motivO, le applicazioni devono essere sviluppate fin dal principio con un'architettura orientata al Cloud. Una buona guida, di sviluppo di applicazioni orientate al Cloud, è la segeunte: *Twelve Factor-Factor App.*, Heroku, servizio PaaS per applicazioni cloud native, promuove continuamente l'importanza dei 12 principi alla base della filosofia *cloud native*. In questa direzione IKS propone servizi di sviluppo di applicazioni di business orientate all'affidabilità, resilienza, scalabilità orizzontale ed ecc.

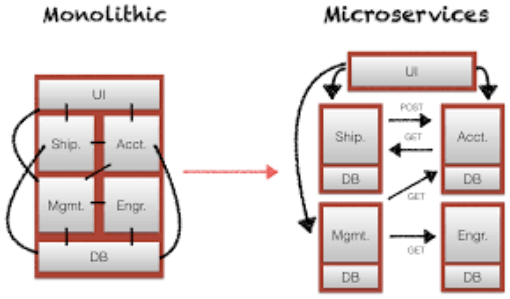


Figura 1.8: Visione architetturale a monolite e microservizi a confronto. Immagine tratta da: http://bit.ly/2rh1niY.

La clientela tipica di IKS sono le aziende operanti nei seguenti ambiti: pubblica amministrazione, bancario, assicurativo e servizi.

Una lista dettagliata delle referenze può essere consultata sul sito di IKS (https://www.iks.it/referenze.html).

1.2.2 Struttura organizzativa

Ad oggi, IKS conta più di 100 dipendenti. La sua organizzazione interna è riassunta nel diagramma in **Figura 1.9**.

In seguito, descrivo le unità operative che costituiscono il nucleo decisionale dell'azienda. Queste unità sono:

* **Direzione**
  Definisce gli orientamenti e le politiche aziendali, gli obiettivi per la qualità, riesamina periodicamente il sistema di qualità e gestisce il piano di formazione dei dipendenti in funzione alle esigenze e motivazioni personali;

* **Direzione Commerciale**
  Definisce le politiche commerciali, gli obiettivi e le risorse necessarie. Promuove i servizi e prodotti dell'azienda. Gestisce i clienti, i fornitori e le offerte contrattuali;

* **Direzione tecnica o Operation**
  Supporta la Direzione Commerciale nella valutazione commerciale di prodotti e/o offerte dal punti di vista tecnico. Gestisce a livello tecnico i progetti e servizi. Pianifica le risorse necessarie per i prodotti/servizi. Verifica lo stato del prodotto/servizio offerto;

* **Amministrazione & Finanza**
  Gestisce la documentazione di progetto, su coordinamento della direzione commerciale e tecnica. Gestisce l'archiviazione della documentazione;

* **Acquisti**
  Su coordinamento della Direzione, gestisce i fornitori di prodotti e servizi. Gestisce il processo di acquisizione di nuovi prodotti o servizi. Il processo di acquisizione è guidato dalle necessità interne aziendali oppure da quelle dei clienti;

* **Assicurazione Qualità**
  È a stretto contatto solo con la Direzione. Gestisce il piano di qualità, coordina le attività di ispezione, misura e stima il livello della qualità aziendale;

* **Business Unit (BU)**
  Gestisce i progetti o servizi concordati con il Cliente. Rendiconta direttamente alla Direzione Tecnica e gestisce l'emissione delle fatture verso il Cliente.

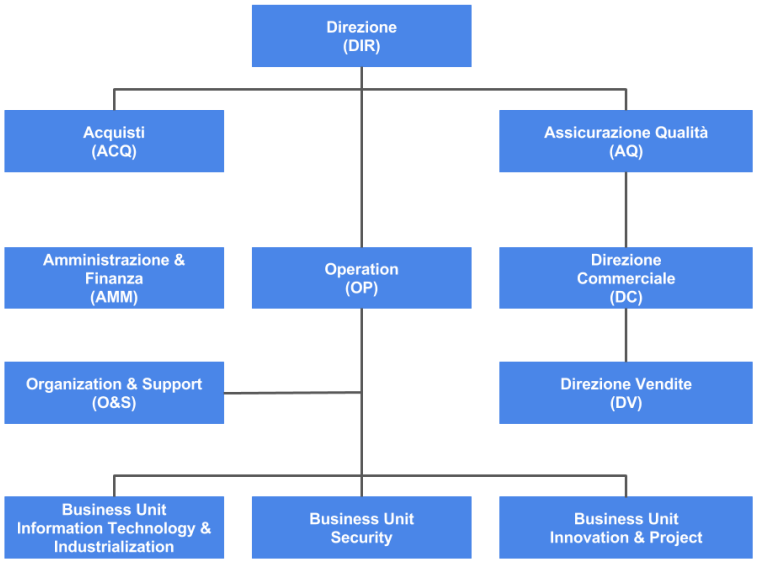


Figura 1.9: Organigramma aziendale

1.2.3 Processi aziendali

IKS, a partire dal 2003, è certificata UNI EN ISO 9001. Questo certifica che l'azienda cura molto la qualità del proprio lavoro. Infatti, il miglioramento continuo permette all'azienda di rimanere competitiva e consolidare la propria posizione di leader sul mercato del ICT italiano. Riporto, di seguito, alcuni obiettivi di qualità dell'azienda:

* Mantenere e aumentare il livello di soddisfazione del Cliente;

* Operare in modo efficiente ed efficace per soddisfare i requisiti contrattuali, norme e regolamenti;

* Monitorare i propri processi per: garantire azioni correttive tempestivamente e permettere un comportamento pro attivo, anticipare i bisogni e predire le risorse aziendali necessarie prima dell'effettivo bisogno;

∗ Assicurare una adeguata formazione al Personale.

Il Cliente copre un ruolo importante nella quotidinità di IKS. Infatti, l'azienda cerca di coinvolgere i propri clienti il più possibile., questo è necessario per comprendere meglio i bisogni attuali del cliente e cogliere esigenze future. In azienda, il passo successivo alla formalizzazione del bisogno del Cliente segue un'attività di analisi dei requisiti. L'obiettivo dell'attività è la dettagliata comprensione del contesto applicativo, quali sono le parti interagenti e quali possono essere i rischi, durante l'attività di progetto, per implementare i bisogni del Cliente.

A progetto concluso, il Cliente valuta criticamente la soluzione presentata. La valutazione, eventualmente, coinvolge un reclamo. Questo è rivolto alla Direzione dell'Azienda.

IKS organizza il proprio lavoro per processi: primari, direttivi e di supporto. Ciascuna categoria di processo definisce delle responsabilità e compiti. Per esempio, i processi organizzativi interessano le attività per: definire la politica e strategia aziendale, pianificare e allocare le risorse, riesaminare la gestione del sistema di qualità. Invece, i processi primari ricoprono attività che garantiscono un diretto ricavo economico per l'azienda e danno un valore aggiunto al prodotto o servizio fornito. Esempi di attività in questa categoria sono: proporre offerte commerciali ai clienti, progettare e sviluppare prodotti software, erogare servizi IT. L'ultima categoria di processi sono i processi di supporto. Le consone attività giornaliere riguardano: gestire le risorse umane, l'infrastruttura e gli ambienti di lavoro., monitorare e analizzare la qualità aziendale.

In **Figura 1.10** segue una presentazione dello schema organizzativo utilizzato dall'Azienda, durante il ciclo di vita di un progetto.

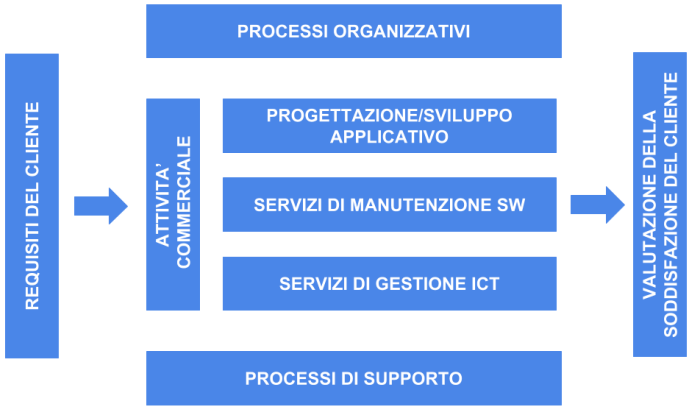


Figura 1.10: Rappresentazione grafica del coinvolgimento del Cliente e i corrispettivi livelli degli interventi del gruppo commerciale, tecnico, direzionale e di supporto nella gestione di un'offerta di progetto.

Periodicamente, il responsabile della qualità, incaricato della Direzione, attua attività d'ispezione. L'obiettivo dell'attività è il controllo del livello di qualità fornita

dai dipendenti aziendali. A posteriori, segue un'attività di analisi e misura dei livelli di qualità organizzati per BU, servizio e prodotto offerto da IKS.

1.3 Rapporto con l'innovazione

L'innovazione è il processo di gestione dell'intero ciclo di vita di un'idea. L'obiettivo è: portare un miglioramento di processo aziendale, di prodotto e/o di servizio. Le conseguenze dirette del miglioramento sono: valore aggiunto, per l'azienda, in termini di rientro economico e soddisfare un bisogno, per il Cliente, in modo efficace ed efficiente.

L'approccio innovativo induce l'utilizzo dell'informazione, della creatività e dello spirito d'iniziativa per raccogliere maggior valore aggiunto dalle risorse a disposizione. L'azienda utilizza l'innovazione per soddisfare in modo pro attivo le richieste del Cliente. Questo principio è pienamente il linea con la strategia di qualità aziendale: *client first*.

La modalità di innovazione di IKS è un approccio incrementale. Inizialmente l'azienda cerca di soddisfare i bisogni principali e raggiungere il prima possibile gli obiettivi minimi pre-fissati. In seguito, l'azienda migliora la propria offerta mediante incrementi continuativi di dettaglio.

Per supportare l'innovazione, IKS ha creato una cultura aziendale che permette ai propri dipendenti di scambiarsi idee, sperimentare, imparare in gruppo e mettere in atto la propria creatività. Non manca la comunicazione con i propri responsabili. Questi sono i primi a motivare di continuo le risorse umane a loro disposizione. Il dialogo dipendente-responsabile non è verticale. La cultura aziendale in questa direzione è molto drastica: favorire uno scambio di idee in modo che esso sia equo, semplice e non orientato alle gerarchie aziendali.

In questo contesto, per l'intera durata del mio periodo di stage e dopo un primo momento di ambientamento, io ho beneficiato molto del clima aziendale. Infatti, non è mancato il libero confronto con il tutor aziendale, il quale ha mostrato disponibilità e apertura al mio spirito d'iniziativa. Sempre mio tutor aziendale ha supportato me in ogni scelta decisionale che io abbia motivato e ritenuto significativa per il beneficio del mio progetto.

Le idee sono una parte del processo di gestione dell'innovazione: la realtà è molto più complessa. IKS non possiede un effettivo processo di gestione a livello aziendale. Questo viene gestito da un gruppo di persone con competenze trasversali e a livelli organizzativi differenti.

Capitolo 2

L'azienda e gli stage

L'azienda investe costantemente sui propri dipendenti. Questo approccio permette all'azienda soddisfare, in modo agile, i bisogni tecnologici del mercato con le giuste competenze. Questo permette ai dipendenti dell'azienda di dedicare parte della propria giornata lavorativa in approfondimenti personali e attività di formazione.

A supporto dell'attività di laboratorio, il gruppo IT interno ha installato un ambiente virtuale. Disporre in azienda di un simile *environment* permette ai dipendenti di sperimentare con: tecnologie, integrazione di sistemi ed ecc. Inoltre, l'ambiente virtuale offre un'infrastruttura dedicata per i progetti di stage. I temi comuni di sperimentazione sono: il *Cloud*, *Machine Learning* e *Analytics* ed ecc.

Di recente, l'Azienda ha concluso una partnership strategica con AWS (*Amazon Web Services*). L'accordo di collaborazione, permetterà a IKS di migrare verso il Cloud la propria infrastruttura informatica e proporre offerte commerciali orientate al cloud, con attore principale Amazon.

Ogni anno, l'azienda partecipa a StageIT: un evento completamente dedicato agli studenti universitari dei Corsi di Laurea in Scienze ed Ingegneria Informatica. Infatti, questo evento permette allo studente di mettersi in contatto diretto con le aziende e queste ultime di promuovere i propri progetti di stage. StageIT prevede, inoltre, un concorso per premiare il miglior progetto di stage, svolto nell'edizione dell'anno precedente. Il vincitore del concorso, scelto dagli studenti, ottiene come premio un buono d'acquisto del valore di 500 Euro.

2.1 Il valore aggiunto di uno stagista

IKS è un partecipante attivo a StageIT; annualmente l'azienda propone fino a 6 progetti di stage. Gli argomenti degli stage non sono verticali su un'unica tematica, ma coinvolgono temi come:

* Sviluppo di applicazioni basate su web, Cloud, mobile;
* Progettazione di ambienti, metodologie e strumenti di sviluppo software.

Lo stagista è una risorsa importante per IKS., infatti, l'azienda percepisce lo stagista come portatore di idee nuove e spirito di innovazione per consolidare il valore aggiunto aziendale. In principio, l'azienda impiega lo stagista su progetti di

sperimentazione. Questi ultimi hanno come obiettivo l'analisi di fattibilità e la loro, eventuale, integrazione nell'offerta commerciale dell'azienda. In seguito, lo stagista assume maggiore indipendenza fino a essere completamente integrato nel mondo del lavoro.

Per l'intera durata dello stage, lo stagista opera in un ambiente vero simile alla realtà aziendale. Il tutor esterno, oltre a guidare nel lavoro lo stagista in funzione di un PdL (Piano di Lavoro), osserva le attività dello stagista. Le osservazioni contribuiscono alla valutazione finale dell'attività di stage del tirocinante.

Riassumendo, i progetti di stage offerti dall'azienda sono importanti alle ambi le parti. Lo stagista assume esperienza lavorativa sul campo e l'azienda sperimenta in modo agile temi innovativi e di tendenza sul mercato sia nazionale che internazionale.

2.2 Alcuni temi di stage

2.2.1 AIOps e Machine Learning

Il progetto di stage tratta l'integrazione del *Machine Learning* con strumenti di Application Performance Monitoring. L'obiettivo principale è la sperimentazione di un nuovo approccio di monitoraggio. Integrando diverse soluzioni in questo ambito e il tirocinante deve fornire uno studio del prodotto finale. Una conseguenza importante di questo progetto è lo sviluppo di un pensiero critico per affrontare le più difficili sfide del monitoraggio di applicazioni e infrastrutture.

Il presente progetto si colloca nell'ambito del *application and performance monitoring* che è un servizio offerto dall'azienda al supporto della governance IT.

2.2.2 DevOps Automazione

L'automazione è fondamento di ogni realtà aziendale contemporanea. Infatti, il numero di macchine da gestire molto spesso non è piccolo. Per semplificare i compiti di gestione si devono utilizzare strumenti di configurazione e automazione. Queste tecnologie permettono di automatizzare tutte le operazioni manuali che un sistemista spesso compie durante le attività di manutenzione giornaliere. L'obiettivo di questo progetto è l'integrazione di alcuni strumenti che semplificano il Patching dei server e sperimentare con nuove tecnologie del settore. Il presente progetto si colloca nell'ambito del *system management*.

2.2.3 Sviluppo moduli evolutivi in ambito antifrode

IKS ha grande esperienza in ambito della sicurezza informatica bancaria. Uno dei prodotti risultati di questa esperienza è SMASH. L'obiettivo dello stage è estendere il prodotto con qualche funzionalità di monitoraggio di azioni sospette. Oltre allo sviluppo di moduli evolutivi lo stagista ha la possibilità di apprendere delle competenze forti nell'ambito della sicurezza informatica. La presente proposta di stage è un progetto inter business unit dell'azienda. Esso si colloca nell'ambito dello sviluppo di prodotti software e della sicurezza informatica nel settore bancario.

2.3 Il progetto proposto

2.3.1 Motivazioni

E' sempre più comune, nelle realtà aziendali, l'approccio agile. La metodologia agile promuove la comunicazione e concentra l'attenzione di tutti i stakeholder sul valore finale di prodotto e/o strategia da raggiungere.

Se gli sviluppatori hanno come obiettivo primario lo sviluppo di un prodotto software allora i professionisti dell'IT hanno come priorità la garanzia di servizio e manutenzione periodica del prodotto software realizzato.

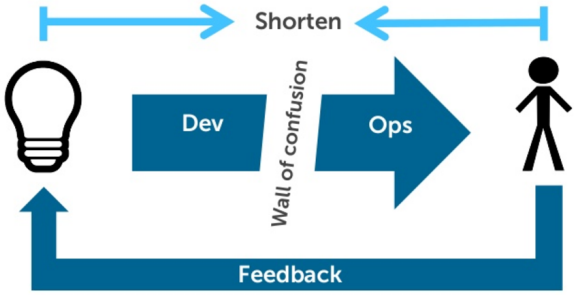


Figura 2.1: Sia gli sviluppatori che i professionisti IT sono portatori di valore: un feedback che coinvolge ambe le parti è essenziale. Immagine tratta da: http://bit.ly/2rM9lBQ.

Tra i due gruppi esiste un muro di incomprensione. La mancanza di comunicazione ed interazione rafforza questo fenomeno. Le problematiche, che avvengono dopo il rilascio del prodotto software, sono responsabilità dei professionisti IT. Di conseguenza, le loro attività sono orientate alla risoluzione dei problemi. Una simile organizzazione della distribuzione delle responsabilità è normale in contesti di realtà aziendali con rilasci sporadici.

Un'azienda informatica, caratterizzata da un numero elevato di rilasci giornalieri, necessita di un approccio diverso. A questo scopo il movimento culturale, chiamato DevOps, è orientato all'unione degli sviluppatori e sistemisti. L'unione promuove un cambio di mentalità, creazione di nuove competenze e sviluppo di nuovi strumenti che possano diminuire la distanza tra le due realtà.

Il DevOps ha conseguenze più profonde del semplice cambio culturale. Esso introduce un cambiamento interno orientato alla modifica del modello di qualità. I benefici di questo cambiamento sono i seguenti: maggiore innovazione, qualità di prodotto, processo e agilità nel cogliere i bisogni di mercato del momento.

Una tipica rappresentazione del ciclo di vita DevOps segue in figura.

L'abilità di cambiare in modo agile è un grande beneficio per le aziende., il modo di lavorare diventa più flessibile e i dipendenti mostrano un maggior coinvolgimento nella quotidianità aziendale.

Lato infrastrutturale, invece, la gestione diventa: disciplinata, sistematica e standardizzata. Con un approccio standardizzato e ben strumentalizzato è più difficile individuare server nomadi. Può succedere che, in caso di operazioni sofisticate e mirate alla manutenzione, un server scompaia dall'orizzonte di visibilità. In assenza di opportuni strumenti è difficile individuare l'accaduto, di conseguenza l'azienda nota un incremento del tasso di spreco delle risorse.

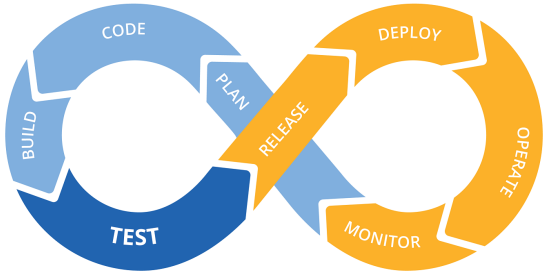


Figura 2.2: Il DevOps abilita l'automazione del processo di rilascio del software e i cambi dell'infrastruttura IT. Immagine tratta da: http://bit.ly/2rsw9nm.

Sebbene il DevOps possieda uno scopo più ampio, il *continuous delivery* è un approccio che promuove l'automazione di tutti i processi coinvolti durante il rilascio di un prodotto software. Questo permette di abbreviare i tempi, aumentare il numero dei rilasci e migliorare la gestione del cambiamento.

Essere veloci nel *delivery* di un prodotto software non è sufficiente. E' importante prevedere una *pipeline* di *deployment* che coinvolga ogni *stage* del ciclo di *operation*. Automatizzare il *deployment* implica minor intervento manuale, minor numero di errori e conseguentemente maggiore formalità nelle attività complessive coinvolte.

Un *application container* permette di confezionare le applicazioni e dipendenze esterne in unità singole. Implementare il *packaging* in questo modo le applicazioni facilita lo scambio di artefatti tra tutti i gruppi coinvolti nel ciclo di vita del prodotto software. Di conseguenza; sia il professionista IT che lo sviluppatore instaurano un protocollo di comunicazione e scambio di esperienze basato su qualcosa di concreto, stabile e funzionante. Il classico detto "funziona sul mio computer" perde di significato.

Al momento la comunità open source offre molte tecnologie di containerizzazione. La tecnologia più stabile e famosa nel comunità Linux è LXC (*Linux Kernel Container*); essa aggiunge il supporto a livello del kernel dei container applicativi per i sistemi operativi Linux. Docker, invece, è un'altra soluzione di containerizzazione. Le prime versioni di Docker interagivano con la tecnologia LXC; le versioni recenti di Docker hanno rimosso la dipendenza software verso LXC e ha implementato una soluzione di containerizzazione proprietaria.

La containerizzazione è una tecnologia molto attraente. Visti i vantaggi tecnici, la containerizzazione porta a un livello superiore il modello concettuale rappresentante un'applicazione. L'architettura dei prodotti software, dal punto di vista dei container, risulta essere più componibile. In un ambiente dinamico, caratterizzato dall'automazione, verifiche e *deployment* automatici, le classiche architetture software non sono pensate per beneficiare di questa flessibilità. Lo stesso non vale per i microservizi.

I microservizi rappresentano uno stile architetturale in sintonia con la filosofia Unix., ogni microservizio implementa una sola funzionalità - basso accoppiamento.

La figura mostra, tramite l'utilizzo dell'esagono, diversi microservizi. Ciascuno dei quali ha una responsabilità ben definita. La rappresentazione ad esagono di un microservizio deriva dal *hexagon design pattern*. Questo modello di progettazione è dovuto ad *Alistair Cockburn*.

Per garantire un basso accoppiamento tra i microservizi, il sistema deve utilizzare un servizio infrastrutturale e di supporto, chiamato *Service Discovery*, utilizzato come

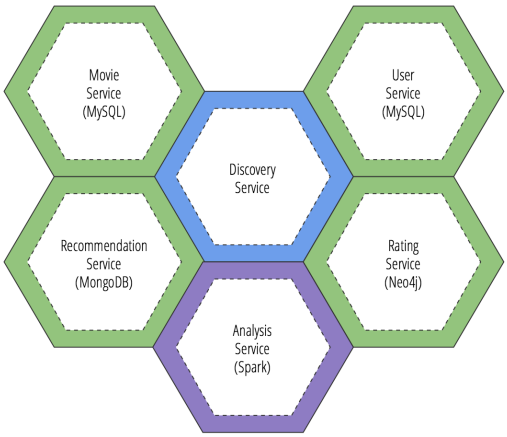


Figura 2.3: Esempio di un sistema a microservizi. Immagine tratta da: http://bit.ly/2qNXKxj.

un DNS (*Domain Name System*). In questo modo, i microservizi continuano ad essere autonomi, mentre la gestione della comunicazione inter microservizio è separata dall'evoluzione dei singoli microservizi. In aggiunta, i microservizi beneficiano della *space transparency*. Se un microservizio X, in esecuzione su una macchina A, migra per eseguire su una macchina B, allora un microservizio Y, che vuole comunicare con X, deve contattare il *Service Discovery* per ottenere l'indirizzo di X. L'effetto ottenibile è un alto tasso di mobilità dei servizi.

E' usuale incapsulare un microservizio in una capsula - il container software. Per la proprietà di isolamento: nello stesso ambiente possono coesistere due o più copie dello stesso microservizio, riducendo quasi a zero l'interferenza di un microservizio su un'altro. Con la scalabilità orizzontale i microservizi beneficiano dell'incremento in robustezza e alta affidabilità. Per implementare il *routing* delle richieste verso i microservizi, i professionisti IT configurano un microservizio di *load balancing* a livello applicativo del modello OSI (*Open System Interconnection*).

I microservizi semplificano l'analisi, progettazione e implementazione dell'applicazione complessiva, scomponendo il prodotto complessivo in sotto applicazioni indipendenti; inoltre, i microservizi complicano l'applicazione a causa di: un elevato numero di componenti da gestire, difficoltà di versionare i microservizi in modo consistente e indipendente, politiche complesse di monitoraggio ed ecc. Per la tracciabilità delle richieste inter servizio è usuale utilizzare strumenti specializzati nel monitoraggio. Esempi di applicazioni, a questo scopo, sono: Appdynamics, Dynatrace, Instana ed ecc.

I container e microservizi aprono nuove sfide sia per gli sviluppatori che per i sistemisti. E queste sfide caratterizzano il contratto di collaborazione tra i due gruppi.

2.3.2 Obiettivi aziendali

Un team interno di IKS ha sviluppato una soluzione di Executive e Malware Dashboard basata sullo stack applicativo: Elasticsearch, Logstash e Kibana.

La mia attività di stage ha avuto la containerizzazione della soluzione precedentemente implementata come obiettivo principale.

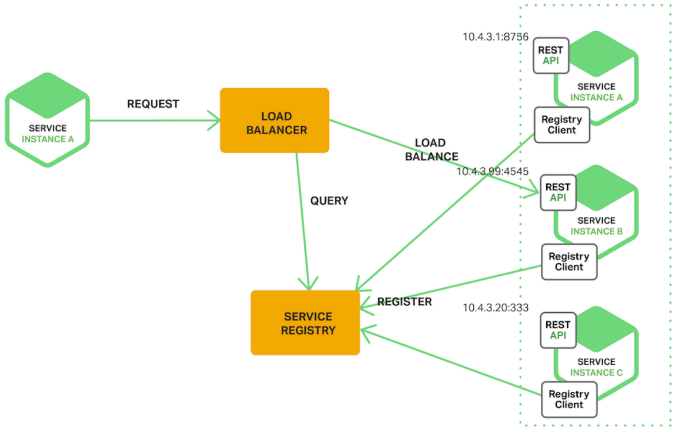


Figura 2.4: I microservizi permettono di scalare orizzontalmente per reggere ai più esigenti carichi di lavoro. Immagine tratta da: http://bit.ly/2qI50LR.

Di seguito presento gli obiettivi principali della mia attività di stage:

* Containerizzare le componenti applicative, costituenti la soluzione implementata dal team interno di IKS;
* Garantire la non regressione al livello funzionale della soluzione;
* Predisporre un ambiente containerizzato con e senza orchestratore di container.

In conclusione, la containerizzazione garantisce che la soluzione di executive e malware dashboard sia portabile su ambienti diversi, dal portatile dello sviluppatore al Cloud.

2.3.3 Obiettivi personali

Come attività preliminare alla ricerca di un progetto di stage per la Laurea ho attuato uno studio individuale di mercato. Lo scopo era capire: tendenze tecnologiche, architetturali e metodologiche. Se da un lato le mie ricerche hanno cercato di cogliere le novità del momento, dall'altro a livello personale queste erano mirate alla ricerca di un contesto in cui potermi applicare e maturare.

Con il presente progetto gli obiettivi personali erano:

* Apprendere conoscenze e competenze in ambito di:
  - Virtualizzazione basata sulla tecnologia a container;
  - Sistemi distribuiti;
  - Amministrazione di sistema Linux;
* Acquisire esperienza pratica nella gestione delle reti di calcolatori in ambito dei sistemi, nello specifico le reti definite in modo programmatico per le tecnologie orientate alla containerizzazione;

- Acquisire esperienza nell'analisi, progettazione e implementazione di sistemi orientati ai microservizi;
- Famigliarizzare con la piattaforma Kubernetes e i principi del Cloud.

2.4 Piano di lavoro

L'azienda ha pianificato un piano di lavoro (PdL) per un totale di 300 ore complessive. Ho consegnato il documento del PdL all'Ufficio degli Stage presso l'Ateneo dell'Università di Padova; una seconda copia del PdL ho consegnato al tutor interno; l'ultima copia, invece, controfirmata dall'ufficio stage dell'Università ho consegnato all'azienda.

Descrivo brevemente di seguito il contenuto del PdL inerente all'insieme delle attività svolte:

- Fase 1 - Formazione (56 ore)
  - Docker: la tecnologia per la containerizzazione;
  - Kubernetes: la tecnologia per l'orchestrazione;
  - Elasticsearch, Logstash e Kibana (ELK): lo stack applicativo;
  - Verifiche delle competenze acquisite;
- Fase 2 - Analisi e progettazione (56 ore)
  - Analisi delle funzionalità della soluzione non containerizzata di dashboard;
  - Analisi delle modalità di containerizzazione delle componenti;
  - Analisi delle modalità di *deployment*;
  - Progettazione delle modalità di verifica della non regressione;
  - Progettazione architetturale della soluzione;
  - Progettazione della modalità di *deployment*;
  - Documentazione;
- Fase 3 - Implementazione (188 ore)
  - Installazione e configurazione dell'orchestratore;
  - Implementazione della soluzione in un contesto con e senza orchestratore;
  - Verifica di non regressione;
  - Documentazione.

2.5 Vincoli

2.5.1 Vincoli temporali

Lo stage ha avuto una durata di 8 settimane, per un complessivo di 310 ore di lavoro. Ho lavorato a tempo pieno con il seguente orario: 9.00-18.00. Con la pausa pranzo di 1 ora dalle 12.30 alle 13.30. Come pianificato nel PdL, ho seguito le attività in ordine seguendo le fasi della pianificazione. Qualche volta ho alterato l'ordine delle attività per adattare le esigenze e ridurre gli effetti del cambio di contesto. Infatti, ogni fase ha coinvolto attività mirate al raggiungimento di specifici obiettivi. Per maggior dettaglio sul contenuto del PdL riferire la sezione Piano di Lavoro.

2.5.2 Vincoli tecnologici

Fin dal primo giorno di lavoro l'azienda mi ha fornito un portatile dedicato per l'itero periodo di stage. Inoltre, mi è stato vietato di collegare alla rete aziendale qualsiasi dispositivo personale. Inoltre, il portatile di lavoro non poteva essere portato a casa. Per comunicare internamente sono stati utilizzati strumenti di messaggistica istantanea, come Skype, per comunicazioni informali e la posta elettronica. Per le comunicazioni via email ho utilizzato il contatto personale universitario.

Oltre a questo vincolo, a livello tecnologico sono state fissate le seguenti tecnologie:

* CentOS7: il sistema operativo installato sulle macchine di laboratorio. CentOS7 è la versione open source di RHEL7 (Red Hat Enterprise Linux versione 7);
* Docker: lo strumento che permette in modo estremamente facile la creazione, il *deployment* e l'esecuzione di applicazioni utilizzando la tecnologia a container. In questo modo l'utente focalizza l'attenzione su questioni diverse dall'installazione e configurazione dell'applicazione. L'architettura di Docker segue in figura.

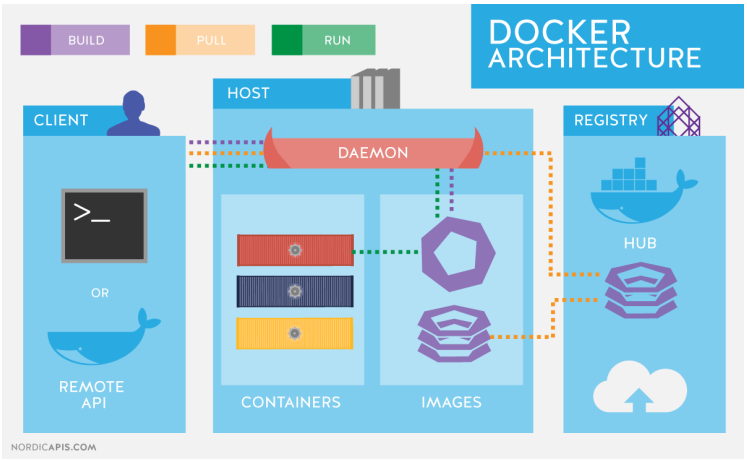


Figura 2.5: Le parti costituenti la piattaforma Docker sono: il demone, il client e il registry Docker. Immagine tratta da: http://bit.ly/2rmkt7g.

Le componenti architetturali costituenti la piattaforma Docker sono il: demone, client e registry. L'architettura di alto livello è un architettura client/server. Il server di Docker è il demone che ha la responsabilità di gestione dei container sulla macchina locale. Mentre il client si interfaccia tramite un'interfaccia REST al demone e permette di interagire in modo agile con i container, creare reti virtuali, gestire i dati che devono essere condivisi tra i container e il file system locale della macchina. Infine, il registry di Docker è una repository che può essere pubblica o privata e ha la responsabilità di abilitare la condivisione di immagini utili alla creazione dei container. Come modello mentale, in relazione con il paradigma ad oggetti, è possibile paragonare le immagini Docker a classi che devono essere istanziate per la creazione di oggetti, ovvero container.

Per favorire il libero scambio di immagini Docker, l'azienda Docker Inc ha messo a disposizione degli utenti un hub: spazio web per la condivisione delle immagini Docker. Nella modalità privata di una repository, Docker Inc. rende disponibile il servizio di *scanning* delle vulnerabilità delle immagini.

Quando l'utente esegue il comando run tramite la CLI di Docker, il demone controlla che l'immagine da istanziare, per la creazione del container, sia presente sul *file system* locale. In caso affermativo, il demone Docker crea e mette in esecuzione il container, altrimenti esso scarica prima l'immagine dal registry e al termine, di questa attività, istanzia il container;

* Kubernetes: è un sistema open source per automatizzare il *deployment*, la scalabilità e gestione di applicazioni containerizzate. La tecnologia è il risultato di 15 anni di esperienza in Google con la tecnologia a container. Kubernetes garantisce la portabilità delle applicazioni e l'indipendenza dall'ambiente fisico di esecuzione.

  Kubernetes è un sistema distribuito per l'orchestrazione di container applicativi. Il modello architetturale dell'orchestratore è master/slave.

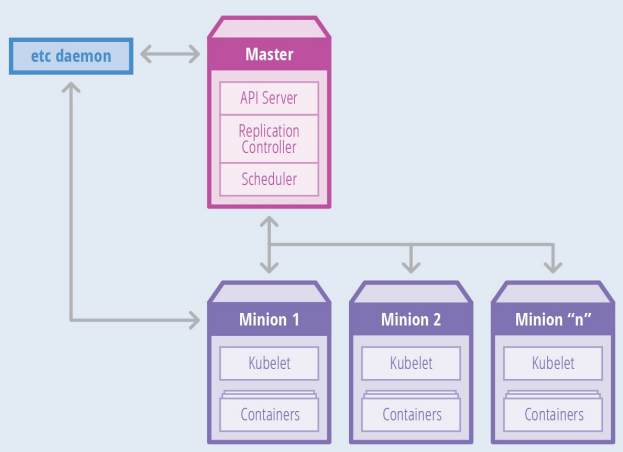


Figura 2.6: Le componenti architetturali di Kubernetes sono: API server, Scheduler, Replication Contoller, Kubelet, Kube proxy, Database Etcd. Immagine tratta da: http://bit.ly/2s4eKUR.

Il master è un pannello di controllo del sistema K8s (Kubernetes). Lui gestisce il *workload* dei container. Inoltre, esegue le componenti critiche del sistema: il database chiave valore ad alta consistenza **etcd**, il gestore delle repliche per la scalabilità orizzontale - **replication controller** e lo **scheduler**.

La componente slave esegue il carico di lavoro. Essa comunica solo con il master e salva le informazioni di servizio, tramite il API server, nel database etcd.

Ogni componente master e slave eseguono un agente locale chiamato Kubelet. L'agente collega le varie componenti e interagisce con il demone Docker.

Infine, il Kube Proxy è la componente che gestisce il traffico di rete dell'intera infrastruttura.

Kubernetes, essendo un sistema fin dall'inizio pensato per essere componibile si può integrare bene con soluzioni di terzi parti, come per esempio: diverse soluzioni per lo storage, diversi plugin per la rete ed ecc;

* Elasticsearch, Logstash e Kibana: Le tre componenti sono comunemente conosciute con l'acronimo ELK. Esse vengono utilizzate insieme come una soluzione

open source in progetti che hanno forti esigenze di ricerca e analisi di dati. Elasctisearch è il cuore dello stack applicativo. Esso è un database NoSQL e distribuito implementato in Java. Orientato all'immagazzinamento di dati non strutturati, Elasticsearch permette di effettuare ricerche complesse impiegando millisecondi contro i secondi necessari utilizzando un classico DB SQL.

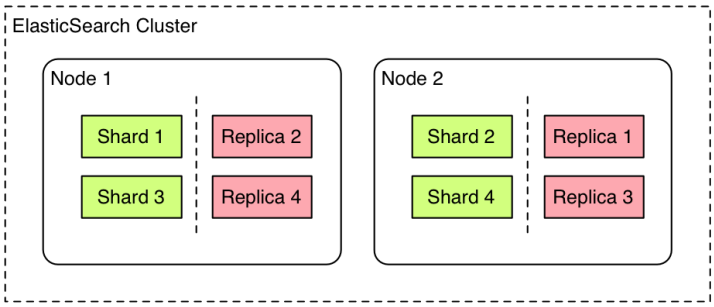


Figura 2.7: Vista di alto livello del meccanismo di partizione dei dati immagazzinati in Elasticsearch. Immagine tratta da: http://bit.ly/2r0HTQL.

Il meccanismo di gestione dei dati di Elasticsearch è complicato. In figura è rappresentato un cluster di due nodi. Elasticsearch permette di organizzare l'insieme complessivo di dati in sottoinsiemi (*shard*) e aggiungere un numero y di copie (*replica*) per ogni sottoinsieme primario. Una simile organizzazione permette di garantire un'alta affidabilità di dati a livello applicativo. Inoltre, in funzione ai casi d'uso Elasticsearch è altamente personalizzabile. Per incrementare le performance di lettura, l'utente incrementa il numero delle repliche. Per incrementare le performance di scrittura, l'utente diminuisce il numero di sottoinsiemi primari dei dati. Kibana, invece, è la componente dello stack che offre la funzionalità di visualizzazione dei dati presenti in Elasticsearch. Una peculiare caratteristica di Kibana è l'interfaccia di creazione dei cruscotti. Kibana sfrutta l'integrazione nativa con Elasticsearch per esprimere ricerche molto complesse e renderizzarle a video tramite effetti grafici accattivanti. Infine, Logstash è la componente di estrazione, trasformazione e caricamento dei dati dalla sorgente in Elasticsearch. Con Logstash risulta semplice filtrare l'informazione utile per l'analisi dei dati ed eliminare il rumore di fondo. La comunità open source di Logstash ha sviluppato un insieme molto ricco di strumenti che estendono il suo insieme di funzionalità. Per esempio, tramite uno plugin esterno è possibile programmare Logstash a interagire con le API (Application Program Interface) del social network Twitter, per cercare, scaricare e filtrare l'informazione che soddisfa uno specifico criterio di ricerca. Dal punto di vista architetturale la soluzione ELK è flessibile e permette di scalare facilmente in orizzontale, proporzionalmente al carico di lavoro. A seguire allego una rappresentazione grafica delle dipendenze inter componenti a livello architetturale. Infatti, Kibana legge da Elasticsearch (ES) mentre Logstash scrive in ES.

Inizialmente sono state fissate anche le rispettive versioni delle componenti sopra citate. Tuttavia, nel corso dello stage ho realizzato che questo può bloccare l'evoluzione dell'infrastruttura e comportare qualche problema in futuro. A questo scopo ho predisposto un ambiente tollerante agli aggiornamenti. Durante la personalizzazione dell'ambiente mi sono ispirato al principio *self driven infrastructure* di CoreOS. In

Figura 2.8: Vista di alto livello del legame sussistente inter componente. Il verso della freccia nell'immagine indica una dipendenza.

questo modo gli aggiornamenti delle componenti possono essere effettuati in modo completamente trasparente.

Infine, come vincolo per le immagini dei container Docker, l'azienda ha imposto a me di utilizzare solo immagini ufficiali provenienti dal hub di Docker. Questo vincolo è dovuto alla presenza di vulnerabilità di sicurezza nelle immagini di terzi parti.

Capitolo 3

Lo svolgimento dello stage

3.1 Metodo di lavoro

Le aziende, indifferentemente dal settore, hanno un proprio metodo di lavoro. Ovvero, ciascuna azienda, nel corso della propria carriera, crea e mette assieme numerose tecniche utili al raggiungimento degli obiettivi di mercato. Un sottoinsieme di queste aziende impone agli stagisti, come vincolo, il proprio metodo di lavoro. Un'ulteriore sottoinsieme di aziende, tramite una negoziazione, sceglie quello più opportuno e conveniente per lo specifico progetto di stage ed affine al tirocinante. Infine, un restante sottoinsieme di aziende offre libertà e non impone alcun vincolo sul metodo del lavoro. IKS, il proponente del mio progetto di stage, ha lasciato a me decidere come organizzare il lavoro di dettaglio del PdL.

Per l'intero periodo dell'attività di stage, mi sono ispirato a SEMAT (*Software Engineering Methods and Tools*). SEMAT è un'iniziativa che promuove la composizione di tecniche e favorisce la creazione di metodi personalizzati di lavoro e su misura per qualsiasi specifico gruppo di persone e/o organizzazione. Il concetto fondamentale, dell'iniziativa SEMAT, è quello del *alpha*. Un *alpha* rappresenta: un punto di vista, un insieme di obiettivi e una *checklist* utile per la valutazione dello stato di avanzamento del progetto. L'insieme degli *alpha*, che SEMAT promuove, rappresenta l'essenza di qualsiasi progetto. Oltre all'essenza, SEMAT propone il kernel, un vocabolario. Lo scopo del kernel è di agevolare la comunicazione tra i membri del gruppo di lavoro. Il kernel può essere utilizzato, inoltre, per la comunicazione tra i membri di gruppi diversi.

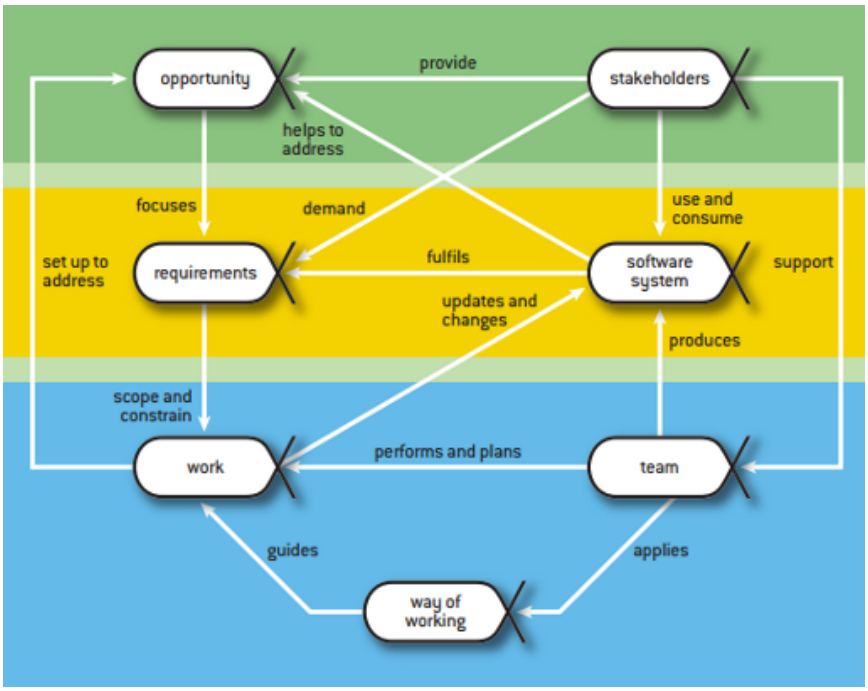


Figura 3.1: Rappresentazione, di alto livello, dei legami sussistenti tra differenti concetti alpha. Immagine tratta da: http://bit.ly/2g5oV7b.

La **Figura 3.1** presenta l'insieme degli alpha e le relazioni sussistenti tra gli alpha adiacenti, su proposta di Ivar Jacobson, il pioniere dell'iniziativa SEMAT.

I colori, invece, rappresentano le aree di competenza per la valutazione dello stato del progetto. Ciascun colore corrisponde a un obiettivo finale; questi obiettivi sono:

* Livello di fidelizzazione del cliente - colore verde;
* Maturità della soluzione e della rispettiva implementazione - colore giallo;
* Il *backlog* di lavoro - colore blue.

Un'interpretazione simile della realtà offre una maggior tracciabilità dello stato di avanzamento del progetto. Per esempio, in **Figura 3.2**, l'alpha dei requisiti offre un ottimo spunto di riflessione sulla maturità e stabilità dei requisiti, individuati nella fase di AdR (Analisi dei Requisiti). Inoltre, esiste per ciascun alpha un insieme di indicatori, che guidano, rispetto al piano iniziale di lavoro, l'andamento e la qualità del lavoro complessivo. Tramite una lista di controllo, SEMAT propone un modo indipendente da processi aziendali di valutazione del completamento degli obiettivi.

La rappresentazione dei temi frequenti in un progetto software tramite gli alpha incrementa la comprensione generale del gruppo di lavoro.

Per esempio, il alpha dei requisiti è caratterizzato dal seguente insieme di obiettivi da raggiungere:

* *Conceived*: Dimostrata necessità per un nuovo prodotto;
* *Bounded*: Lo scopo del nuovo prodotto sono chiari;

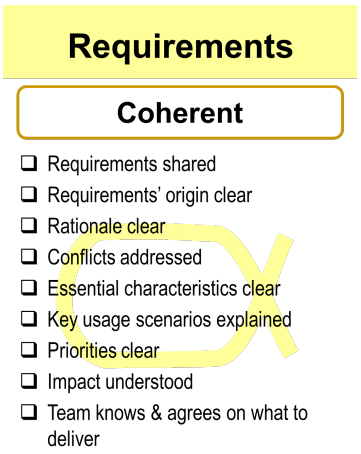


Figura 3.2: Ogni concetto alpha è caratterizzato da un insieme di elementi descrittivi, come: nome, l'obiettivo e la lista di controllo. Immagine tratta da: http://bit.ly/2vgPqJf.

* *Coherent*: I requisiti offrono una chiara prospettiva del nuovo prodotto software;
* *Acceptable*: I requisiti descrivono il prodotto che è accettato dal cliente;
* *Addressed*: Una buona parte dei requisiti sono stati implementati, il sistema soddisfa la visione che il cliente ha del sistema;
* *Fulfilled*: I requisiti che sono stati affrontati sono in accordo con gli obiettivi pattuiti con il cliente.

Durante l'analisi dei requisiti, ho utilizzato questi obiettivi per valutare lo stato del mio lavoro, rispetto alla pianificazione iniziale. Inoltre, quando ho raggiunto l'obiettivo *"Coherent"* dei requisiti, ho acquisito una buona visione del sistema da sviluppare nel complesso Infatti, la lista di controllo, in allegato a ciascun alpha, mi ha guidato nell'attività di consuntivazione. Ho trovato d'aiuto questa lista di controllo anche come fonte di requisiti e domande critiche sul progetto e sistema.

Ho utilizzato, per esempio, questa griglia per valutare la mia situazione di progresso rispetto alla pianificazione di periodo.

Per un maggior controllo e visibilità del grado di completamento del progetto, ho utilizzato l'applicazione "SematAcc", che è raggiungibile al seguente indirizzo `http://sematacc.herokuapp.com/`. L'applicazione mi ha permesso, grazie al cruscotto intuitivo da essa offerto, di concentrare la mia attenzione su obiettivi di valore aggiunto al progetto e temporeggiare con le attività di priorità più bassa. In questo modo, ho ottenuto un modo agile e semplice per gestire i rischi.

A progetto concluso, la situazione del progetto è come in *Figura 3.3*. I dati, che presento, mostrano uno stato salutare per il progetto. Questi dati non rappresentano il massimo valore; visto che, gli obiettivi sono in continua evoluzione, come le tecnologie.

Ho pianificato e trattato il margine rimanente come un margine di evoluzione e miglioramento del sistema.

Aderire all'iniziativa SEMAT e seguire approcci metodologici flessibili è vantaggioso. Nel mio caso, ho composto un proprio metodo di lavoro; ho ottenuto una semplice

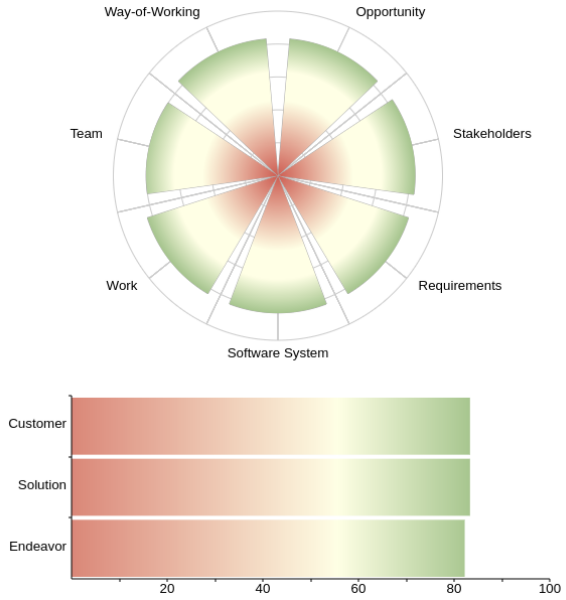


Figura 3.3: Rappresentazione dello stato del progetto, a seguito dell'ultima consegna ufficiale e superamento della revisione di accettazione.

gestione dei rischi e ho migliorato la mia sensibilità per recepire lo stato del progetto in modo naturale ed intuitivo. Inoltre, ho migliorato la metodologia di lavoro e, con disciplina, ho acquisito le capacità di controllo sull'incertezza, in momenti di decisioni importanti.

3.2 Pianificazione

Durante lo stage ho lavorato 316 ore. Complessivamente, queste ore corrispondono poco più di 8 settimane di lavoro effettivo. Nel mio caso, lo stage è iniziato il 12/12/2016 e terminato il 20/02/2017. In questo arco di tempo, è incluso anche il periodo delle festività natalizie. Causa assenze impreviste da parte mia, nell'ultima settimana di lavoro, ho esteso la terminazione dello stage dal giorno 20/02/2017 al 23/02/2017.

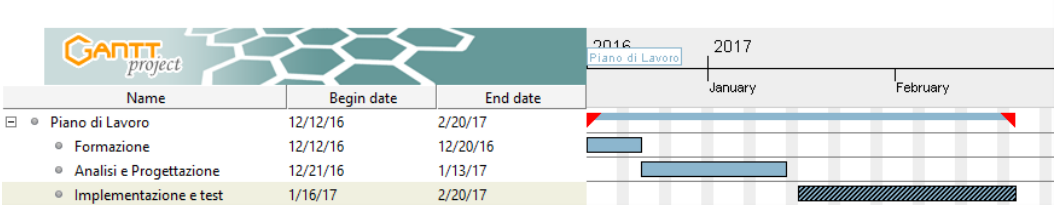


Figura 3.4: Piano di lavoro ufficiale.

In **Figura 3.4** presento il diagramma di Gantt del piano di lavoro, i cui dettagli ho descritto sezione Piano di Lavoro.

Il diagramma di Gantt presenta una pianificazione generica del PdL; tuttavia, essendo io libero nell'organizzazione del lavoro, ho rivisto e riprogrammato le attività del PdL come in **Figura 3.5**.

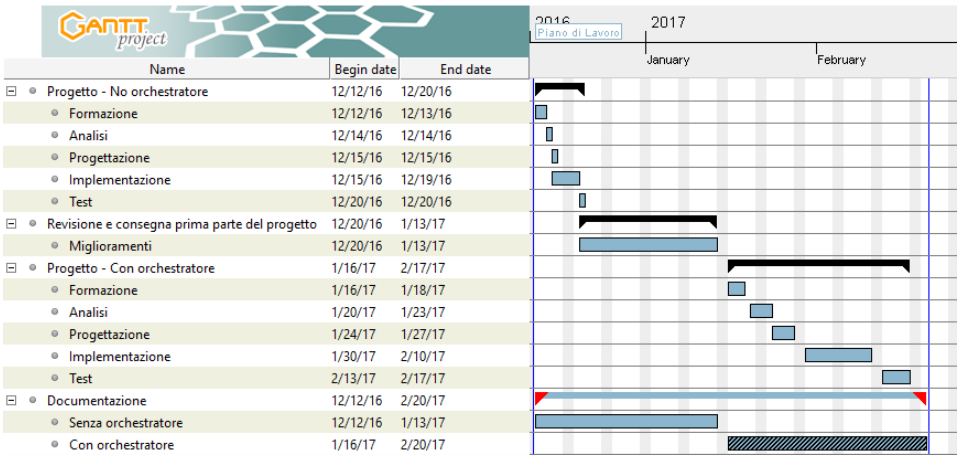


Figura 3.5: Piano di lavoro rivisto e riorganizzato.

Il progetto prevede la containerizzazione di una soluzione di "*Malware Dashboard*" e il rilascio del sistema containerizzato in un ambiente con e senza orchestratore. In relazione con le modalità richieste di messa in esercizio del sistema, ho organizzato il lavoro complessivo in due mini progetti. Ho orientato il primo progetto al rilascio del sistema in un contesto containerizzato senza orchestratore, qui ho utilizzato unicamente la tecnologia Docker, e un secondo progetto orientato all'orchestrazione di container, qui ho utilizzato la tecnologia Kubernetes. Il secondo progetto ha esteso il primo da un punto di visto di un incremento migliorativo.

Come parte di riuso tra i due progetti, ho utilizzato l'architettura del sistema complessivo e le componenti containerizzate. Una simile organizzazione del progetto di stage, mi ha permesso di pianificare i momenti di *sprint*; uno sprint è un insieme di compiti, usualmente, di breve durata e molto intensi di lavoro. Ho scelto di operare in questo modo perché ho ritenuto importanti i *feedback* del tutor aziendale. Inoltre, ho utilizzato questi feedback come una metrica di avanzamento, raggiungimento degli obiettivi e maturità della soluzione. Inoltre, conoscere il punto di vista del tutor ha garantito stabilità alla soluzione per l'implementazione del sistema.

3.3 Attività di formazione

Ho seguito un metodo attivo di formazione. Ho arricchito le attività di studio teorico con attività di laboratorio. Durante le sessioni pratiche, ho svolto piccoli progetti. Tramite questi piccoli progetti, ho toccato con le mani le tecnologie alla base dello stage.

3.3.1 Docker

Ho imparato la tecnologia, Docker, allo stato dell'arte della containerizzazione. Durante la formazione su Docker, ho appreso i concetti alla base della tecnologia, come utilizzare

lo strumento e come pensare i sistemi orientati ai container.

Tramite il frammento di codice, che allego in seguito, illustro come estendere un'immagine Docker della componente d'esempio Elasticsearch con una estensione di terzi parti; questo frammento di codice è chiamato Dockerfile.

```
FROM elasticsearch:2.4
MAINTAINER Andrei Petrov <apetrov.ya@yandex.ru>
ADD hq.tar /usr/share/elasticsearch/plugins
```

Il Dockerfile rappresenta la configurazione di componente e le sue dipendenze software. In un altro modo, il Dockerfile descrive in modo dichiarativo le dipendenze esterne del processo. Il Dockerfile è necessario per la creazione dell'immagine. Dall'immagine, invece, si possono istanziare, possibilmente, infiniti processi. Inoltre, un'informazione curiosa sulla chiave **ADD** è la seguente: la direttiva permette, in modo automatico, l'estrazione di file da un archivio con estensione tar. Questo *hack* è utilissimo e molto amato dagli esperti di containerizzazione Docker. La direttiva è uno *shortcut* che permette di ridurre a una sola registrazione sul *Augmented File System*, utilizzato da Docker per la memorizzazione della configurazione dei container. Altrimenti, per copiare un archivio da un contesto all'altro durante la creazione dell'immagine, devo copiare l'archivio e successivamente estrarre i dati. Anche io preferisco utilizzare la direttiva "ADD".

Quando un prodotto software complesso è costituito da un insieme di componenti containerizzate, un modello mentale utile per la comprensione delle relazioni tra le immagini Docker è il modello ad ereditarietà singola dei linguaggi di programmazione ad oggetti, come per esempio il linguaggio di programmazione Java. Le immagini, come in alcuni casi le classi, possono essere organizzate tramite estensioni. Questo modello favorisce il riciclo delle componenti.

Ho utilizzato i Dockerfile come template per la creazione di immagini e container di servizi. Questo approccio mi ha permesso di codificare in un file, da un lato, un insieme di configurazioni parametrizzate; dall'altro, includere fisicamente nel package le dipendenze esterne. In seguito alla codifica dei Dockerfile ho notato un notevole incremento della mia produttività.

La tecnologia Docker ha rivoluzionato il mondo dello sviluppo del software. Questa tecnologia ha introdotto uno standard per il confezionamento del software.

Un esempio di istanziazione di un container applicativo Elasticsearch è il seguente:

```
docker run -d \
-p $(ES_MASTER_CONTAINER_HTTP_PORT):$(ES_MASTER_CONTAINER_HTTP_PORT) \
-p $(ES_MASTER_CONTAINER_TCP_PORT):$(ES_MASTER_CONTAINER_TCP_PORT) \
--name $(MASTER_CONTAINER_NAME) \
--volumes-from $(ES_DATA_CONTAINER_NAME) \
-v limits.conf:/etc/security/limits.conf \
$(ES_DOCKER_IMG):$(ES_DOCKER_IMG_VERSION) \
-Des.node.name="$(MASTER_ES_NODE)" \
-Des.cluster.name="$(ES_CLUSTER_NAME)" \
-Des.node.master="true" \
-Des.node.data="false"  \
-Des.index.number_of_shards=$(ES_SHARDS_NUM) \
-Des.index.number_of_replicas=$(ES_REPLICAS_NUM) \
-Des.network.host=_site_ \
...
```

```
-Des.gateway.recover_after_nodes=2 \
-Des.gateway.expected_nodes=2 \
...
```

Qui, tramite la CLI (Command Line Interface) di Docker, creo e ed eseguo un container in modalità demone (processo eseguito in background). Successivamente, configuro le porte logiche per abilitare la comunicazione interprocesso via rete; specifico un *bucket* logico per la condivisione dei dati tra il server e il container, tramite le opzioni "–volumes-from" e "-v"; fornisco un alias mnemonico al container, tramite l'opzione "–name" ed ecc. Le opzioni con il prefisso "-Des" rappresentano le configurazioni per il processo Java di Elasticsearch. In questo frammento di codice ho omesso alcune parti.

Per verificare le mie conoscenze preliminari su container e Docker, ho implementato uno script Bash con l'obiettivo di creare un cluster di `N` nodi di Elasticsearch in modo automatico ed autonomo. Il cluster creato è rilasciato in modalità locale alla macchina virtuale ospitante. Su un insieme di `N` nodi, `N-M` nodi sono di tipologia "data" e il resto sono di tipologia "master". Il numero minimo di nodi master è dato dalla seguente formula `M_min = M/2 + 1`. La quantità `M_min` specifica il *quorum* iniziale di nodi, necessario per la procedura di elezione di un nuovo leader del cluster.

3.3.2 Kubernetes

L'orchestratore è indispensabile per una efficiente gestione di una flotta di container. Durante il periodo di formazione, ho affrontato diversi problemi legati al presente orchestratore. Alcuni di questi problemi riguardano la creazione di un cluster Kubernetes in configurazione semplice. Questa configurazione interessa la creazione di un cluster formato da nodi master e worker. La configurazione del cluster è sprovvista di alta affidabilità. Ho effettuato questa scelta causa la difficoltà della procedura di installazione e configurazione di un cluster Kubernetes in alta affidabilità. Un cluster Kubernetes può essere creato tramite strumenti di terzi parti della tecnologia, fornite come moduli esterni che estendono il prodotto base. Un simile strumento, per esempio, è **kubeadm**. Inoltre, è possibile creare un cluster Kubernetes a mano. Quest'ultima modalità è macchinosa e lunga. Durante lo stage ho favorito la semplicità; con lo strumento *kubeadm* ho effettuato il *deployment* di un cluster Kubernetes in breve tempo. Lo strumento gestisce in modo automatico tutti i passi di configurazione iniziale. Inoltre, *kubeadm* offre una procedura di distruzione del cluster.

Ad esempio, tramite il semplice comando

```
kubeadm init
```

ho installato e configurato un nodo master di Kubernetes. Successivamente, con il token generato, ho aggiunto ulteriori nodi worker al nodo master esistente. Per aggiungere un nuovo nodo worker ho usato, invece, il seguente comando:

```
kubeadm join <token> <indirizzo IP del master>
```

Durante l'attività di formazione su Kubernetes ho svolto attività di studio mirate all'amministrazione dell'infrastruttura Kubernetes e all'organizzazione delle risorse containerizzate tramite gli oggetti nativi dell'orchestratore, per esempio: Replication-Controller, Services, Deployments, Job ed ecc., affinché l'orchestratore possa gestire i container in esecuzione. L'oggetto base di Kubernetes è il Pod. Questa risorsa è una capsula dove al centro risiede un container Docker. Un Pod è anche l'unita atomica di schedulazione.

Per la codifica degli oggetti di Kubernetes ho utilizzato dei file. Questi file sono un modo dichiarativo per codificare l'infrastruttura. Nella comunità di Kubernetes essi vengono chiamati *manifest*.

Un esempio di manifest, che ho codificato, è il seguente

```
---
apiVersion: v1
kind: Service
metadata:
namespace: elk-dev
name: elasticsearch
labels:
  component: elasticsearch
  role: client
spec:
 selector:
 component: elasticsearch
 role: client
 ports:
 - name: http
   port: 9200
   protocol: TCP
---
apiVersion: v1
kind: ReplicationController
metadata:
  name: es-client
  namespace: elk-dev
  labels:
   component: elasticsearch
   role: client
spec:
 replicas: 1
 ...
```

Il linguaggio che ho utilizzato per la stesura dei manifest di Kubernetes è il linguaggio di markup YAML (Yet Another Markup Language). I manifest possono essere stesi in due formati: JSON e YAML. Ho utilizzato il secondo per facilitare la lettura. Il manifest allegato è un esempio parziale di configurazione in codice di una componente Elasticsearch dell'infrastruttura. La prima risorsa è un Service. Un Service implementa la funzionalità di *Service Discovery*. Infatti, i microservizi utilizzano i nomi dei Service per abilitare la comunicazione. La seconda risorsa, invece, rappresenta un ReplicationController. Il ReplicationController dichiara che il numero di copie per lo specifico microservizio deve essere sempre pari a uno; lo stato di questa risorsa è gestito dalla componente controller di Kubernetes. In caso che, lo stato della risorsa inizierà a divergere , il controller riporterà allo stato specificato la risorsa. Questa funzionalità permette, in caso di errore, di schedulare nuovamente il Pod e risanare lo stato della componente tramite una sequenza di riavvi continui. E in questo caso il controller di Kubernetes segnalerà il motivo di errore della risorsa. Questa funzionalità si chiama *self-healing*. I dati dei pod possono essere salvati su volumi di memorizzazione esterni.

La specifica di qualsiasi risorsa è costituita da quattro parti fondamentali: apiVersion, kind, template e spec. La chiave "apiVersion" determina la versione delle API di Kubernetes da utilizzare quando dobbiamo specificare una risorsa di tipo "kind". Ogni chiave-valore presente internamente alla sezione template deve essere interpretata come una meta informazione necessaria all'amministrazione delle risorse. Infine, il contesto "spec" descrive la specifica per la risorsa. Le specifiche riguardano la codifica dello stato della risorsa.

La tecnologia Kubernetes offre una macchina a stati per la gestione dei container. Questo modello a stati, introdotto da Kubernetes, permette diversi problemi legati alla qualità dei servizi, alla quantità di risorse utilizzate da ciascuna risorsa ed ecc.

3.3.3 Clustering di Elasticsearch

Una delle caratteristiche più importanti di Elasticsearch è la funzionalità di *clustering*. Questa funzionalità permette di eseguire in parallelo un insieme di processi Elasticsearch sotto uno spazio di nomi comune. La singola componente, di un cluster, prende il nome di nodo. I nodi di un cluster condividono dati. L'organizzazione dei dati e la politica della loro gestione, in modalità cluster, è complessa. Durante il periodo di formazione ho appreso come creare, organizzare i dati e gestire la componente Elasticsearch in modalità *clustering*.

Approfondito il concetto di clustering in Elasticsearch, ho implementato degli script per automatizzare e velocizzare le procedure di manutenzione del cluster Elasticsearch.

Se da un lato, il clustering incrementa la percentuale di *High Availability*, da un altro lato è necessario analizzare i requisiti d'uso del cluster. L'analisi d'uso è necessaria per comprendere come configurare i livelli di ridondanza dei dati e l'organizzazione di partizionamento degli indici gestiti da Elasticsearch. Nella terminologia Elasticsearch un indice è un database. Durante la formazione su Elasticsearch, ho realizzato che esigenze di scrittura possono impattare le performance del cluster se il numero delle repliche è elevato. Questo fenomeno è dovuto alla grande latenza di propagazione delle scritture su ciascuna copia dei dati memorizzati. Al contrario, in caso di esigenze di lettura, ho notato che un alto numero di repliche migliora di molto i tempi di risposta del cluster Elasticsearch.

3.4 Analisi dell'alta affidabilità

Una delle più importanti garanzie che il settore IT deve garantire al business è la continuità operativa dei servizi informatici a supporto. I servizi informatici sono complessi e distribuiti. A causa della loro natura distribuita, i sistemi informatici possono essere sottoposti a problemi, per esempio, di rete, che degradano la qualità di erogazione di questi servizi.

L'alta affidabilità è un'abilità, del sistema, che permette di garantire la continuità operativa, anche se le singole componenti non sono in uno stato di buona salute. L'analisi dell'alta affidabilità ha l'obiettivo di comprendere a fondo i dettagli della proprietà di alta affidabilità dell'intero sistema da realizzare.

In questa fase ho individuato due livelli esclusivi che interessano l'alta affidabilità: un livello fisico/virtualizzato e uno logico/containerizzato. Su raccomandazione del mio tutor aziendale, ho tralasciato lo studio dell'alta affidabilità a livello fisico e mi sono concentrato su quello logico. A questo livello, ho analizzato quali sono le componenti che possono mettere a rischio lo stato complessivo del sistema.

Nella tabella a seguire, riporto le tecnologie utilizzate e per ciascuna di esse caratterizzo la tipologia di applicazione e il rispettivo livello di supporto per il clustering.

Componente	Tipologia	Supporto nativo clustering
Elasticsearch	Applicazione Java	Si
Kibana	Applicazione Javascript	No
Logstash	Applicazione Java	No
Nginx	Applicazione C++	Si

Tabella 3.1: Elenco delle tecnologie utilizzate con il supporto nativo per il clustering

Dalla tabella segue che, alcune componenti non offrono un supporto nativo per il clustering. Per garantire la continuità di servizio, ho utilizzato il concetto di *redundancy*. Tramite l'esecuzione parallela di più processi dello stesso tipo, posso garantire la continuità di erogazione del servizio, anche se al più tutti meno uno di questi processi ,all'improvviso, falliscono. Ad alto livello, l'utente può, eventualmente, risentire un degrado di *performance*. Un ulteriore elemento della mia analisi è stata la garanzia del basso accoppiamento tra le componenti del sistema.
Come strumento a supporto dell'analisi dell'alta affidabilità, ho utilizzato il modello a cubo della scalabilità. Il modello offre tre dimensioni X,Y e Z. Ciascuna dimensione rappresenta una oggetto di studio. Per esempio, Elasticsearch è un'applicazione distribuita che offre nativamente il supporto per la scalabilità orizzontale e il partizionamento dei dati. Una conseguenza importante da notare in Elasticsearch è il livello della consistenza. Infatti, Elasticsearch è un database NoSQL e non garantisce l'alta consistenza. Questo è il prezzo da pagare quando l'obiettivo del sistema è la scalabilità su larga scala.
Sebbene la scalabilità implica l'alta affidabilità, per esempio tramite la ridondanza, ho ritenuto il modello della scalabilità a cubo un concetto importante per comprendere le dimensioni dell'alta affidabilità del sistema.

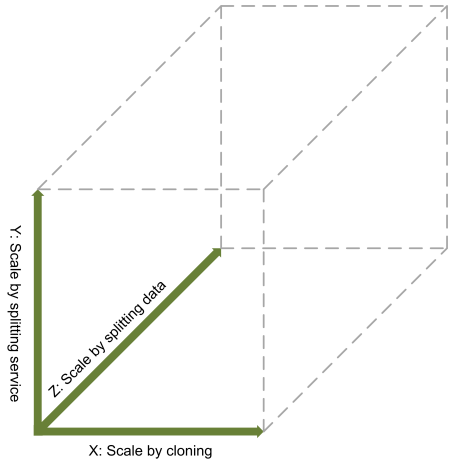


Figura 3.6: Interpretazione grafica del modello a cubo della scalabilità. Immagine tratta da: http://bit.ly/2wPA42R

La dimensione del sistema determina la difficoltà di gestione. La difficoltà di gestione incrementa la probabilità che a regime (sistema operante a tempo pieno) il sistema

possa rivelare un comportamento non atteso. La garanzia di un appropriato livello di servizio favorisce un accordo equo tra il fornitore del servizio e l'utilizzatore. Una metrica che ho fissato per la qualità di erogazione del servizio è la classe 3 di alta affidabilità. Ho calcolato questa metrica tramite i test di carico e i test di durata.
La possibilità di valutare a regime la qualità del servizio è un obiettivo dei prossimi incrementi evolutivi del progetto. Al momento dello stage ho ritenuto questo obiettivo un requisito opzionale.
Per la misura dell'alta affidabilità, ho utilizzato la seguente scala espressa in tabella.

Tipo di sistema	Disservizio (minuti per anno)	Affidabilità	Classe
Non gestito	50000	90	1
Gestito	5000	99	2
Bene gestito	500	99.9	3
Tollerante ai guasti	50	99.99	4
Altamente affidabile	5	99.999	5

Tabella 3.2: Categorizzazione delle classi di affidabilità dei sistemi informatici. Materiale tratto da: http://bit.ly/2wlGrbn.

L'incremento della classe di alta affidabilità è un'attività che richiede costanza e automazione. I sistemi caratterizzati da alte classi di alta affidabilità richiedono che essi vengano progettati per essere il più possibile autonomi.

3.5 Progettazione ed implementazione

Con la presente sezione descrivo l'architettura di alto livello del sistema.

3.5.1 Vista architetturale

Per la modellazione di alto livello del sistema ho utilizzato lo stile architetturale a Microservizi. Questo stile promuove la dimensione Y del modello a cubo della scalabilità. Infatti, lo stile architetturale a Microservizi porta all'estremo il concetto del partizionamento delle funzionalità. In questo contesto, una funzionalità individua un processo. Con l'ausilio dei container i microservizi vengono containerizzati. Per ottenere la scalabilità orizzontalmente è sufficiente aggiungere un nuovo processo, copia del processo containerizzato in esecuzione e bilanciare il traffico verso l'insieme di processi cosi ottenuto.
In figura che segue, presento i microservizi costituenti il sistema da me realizzato.

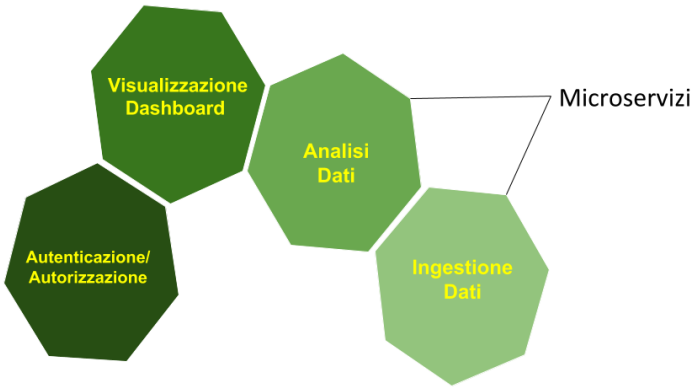


Figura 3.7: Visione di alto livello dell'architettura del sistema.

Ogni microservizio è responsabile di una sola funzionalità. Le figure esagonali sono una convenzione diffusa per rappresentare i singoli microservizi costituenti un'architettura a Microservizi. Questa notazione ha origine nel pattern architetturale "Hexagon Design Pattern", promosso da Alistair Cockburn.
In tabella, per ogni microservizio riporto una breve descrizione della funzionalità e la tecnologia utilizzata per la sua implementazione.

Microservizio	Funzionalità)	Tecnologia
Autorizzazione/Autenticazione	Gestione degli accessi	Nginx
Visualizzazione	Gestione delle dashboard	Kibana
Analisi dati	Gestione dei dati in analisi	Elasticsearch
Ingesting	Raccolta ed inserimento dei dati	Logstash

Tabella 3.3: Elenco dei microservizi costituenti il sistema.

3.5.2 Flusso operativo

Il sistema che ho progettato è caratterizzato da un alto tasso di coesione e basso accoppiamento. In figura a seguire, con l'ausilio dell'indicazione numerica, presento i flussi di dati che il sistema deve gestire.

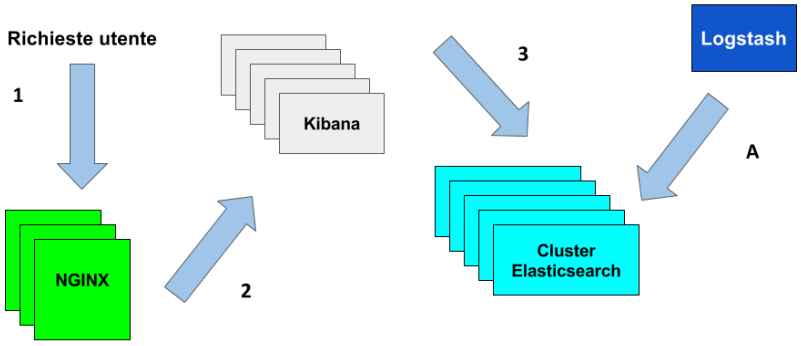


Figura 3.8: Indicazione dei flussi di dati e delle dipendenze dei microservizi.

Il sistema è caratterizzato da due flussi distinti.

1. **Flusso 1-2-3**:
   Il seguente flusso rappresenta il flusso inerente alla parte esposta all'uso degli utenti.

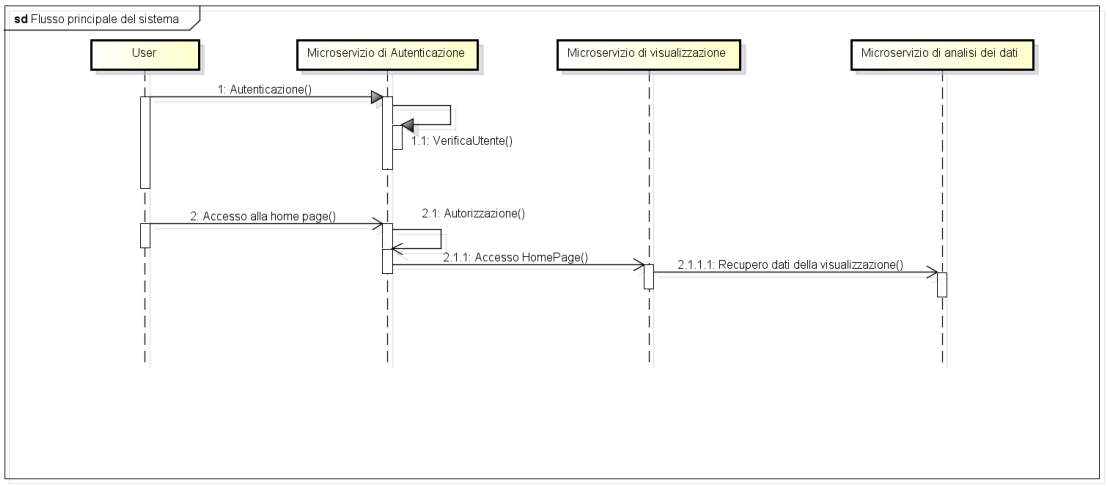


Figura 3.9: Descrizione di alto livello dell'interazione dell'utente con il sistema.

2. **Flusso A**:
   Un'ulteriore flusso interessa il processo di raccolta ed inserimento dei dati nel sistema. Lo strumento ETL (Extract Transform and Load) utilizzato è Logstash. Questo flusso non impatta le perfomance complessive del sistema, in quanto la sorgente dati è statica, ha una dimensione fissa. Il flusso è operativo solo durante la prima installazione del cluster, successivamente esso è disattivato.

3.5.3 Organizzazione del cluster Elasticsearch

Dal punto di vista applicativo e in contesto WEB, Elasticsearch può essere interpretato come un backend, ovvero una sorgente di dati. Questa componente supporta diverse configurazioni topologiche di nodi. L'insieme di nodi che non ho trattato sono i nodi per la comunicazione inter cluster e distribuiti su zone geografiche diverse. Un *use case* comune per l'utilizzo di questo tipo di nodi è la replica dei dati tra due *Data Center* diversi.
La progettazione del cluster che ho trattato riguarda una configurazione di nodi distribuiti nella stessa zona geografica.
In contesto con orchestratore Kubernetes, il cluster minimale è composto dal seguente insieme: 1 nodo data, 1 nodo master e 1 nodo client. L'ultimo nodo ha il compito di bilanciare il traffico esterno proveniente dagli utenti connessi al cluster. Inoltre, i nodi di tipo client mantengono una tabella informativa su quale nodo mantiene uno specifico frammento di dato. Ho progettato in questo modo l'architettura del sistema per agevolare la scalabilità orizzontale delle componenti. Un vincolo forte per la scalabilità dei nodi master è il seguente: il numero dei nodi master deve essere scalato in quantità dispari.

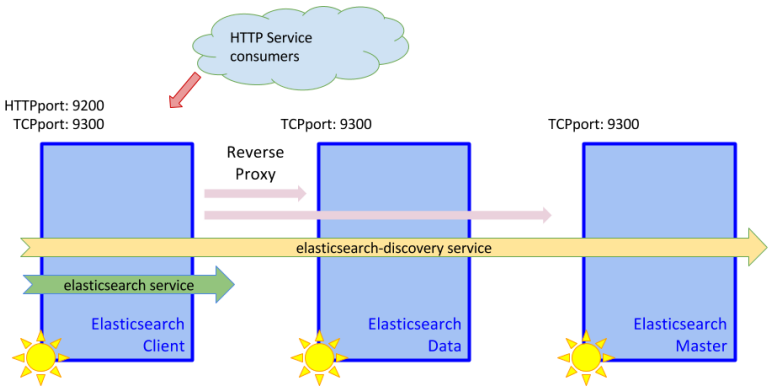


Figura 3.10: Architettura del *backend* applicativo.

Il traffico tra i processi Elasticsearch è di due tipi. Il primo traffico riguarda quello di servizio ed interno del cluster, è un traffico non visibile dall'esterno ed è necessario per coordinare i singoli nodi del cluster, il protocollo utilizzato è HTTP e la porta logica è 9300. Sempre attraverso il protocollo HTTP e porta 9200 ho esposto il servizio di Elasticsearch ad ogni altro microservizio consumatore. Il traffico entrante è diretto verso il nodo client che in modalità *reverse proxy* inoltra le richieste ai membri del

cluster. Il traffico tra le componenti del cluster Elasticsearch è criptato tramite un token. Ogni componente del sistema è associata con un utente che ha determinati privilegi. In figura, ho rappresentato il token di sicurezza tramite un sole di color giallo. L'utente interagisce con il microservizio di gestione delle dashboard. La tecnologia utilizzata è un applicazione web, che nativamente è pensata per comunicare con Elasticsearch. Quest'applicazione, essendo priva di stato, è facilmente scalabile in orizzontale.

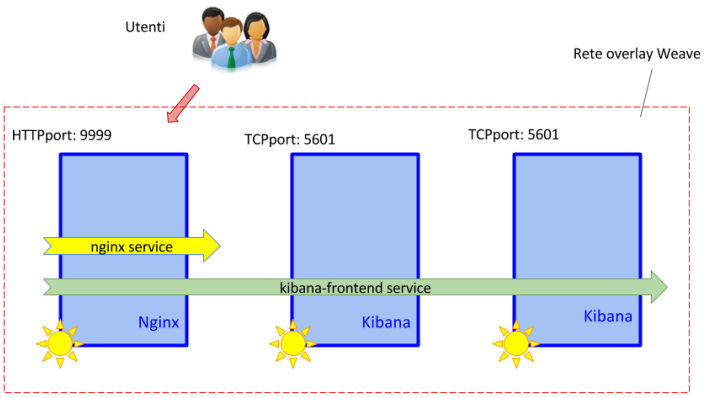


Figura 3.11: Architettura del *frontend* applicativo.

Il sistema espone un unico punto di accesso alle dashboard. Ogni utente per accedere al sistema deve essere registrato. Solo dopo un'autenticazione l'utente può utilizzare il servizio. Inoltre, per garantire che solo determinati utenti possano accedere al servizio, il sistema effettua un filtraggio base sugli indirizzi IP. Infatti, solo gli indirizzi IP nella lista bianca vengono accettati dal sistema. In caso di altri indirizzi IP, il sistema ignora le richieste.
Per l'implementazione del microservizio di autenticazione ed autorizzazione ho utilizzato Nginx. Nginx è un *load balancer* a livello L7 (Layer 7 OSI), bilanciatore del traffico a livello HTTP.
Un'ulteriore questione che ho trattato è il dimensionamento delle componenti del sistema.
Ciascuna componente specifica una quantità fissa di risorse. Questa quantità non è statica e può variare nel tempo. Durante le valutazioni di dimensionamento ho cercato di garantire che il totale delle risorse richieste da un insieme di componenti, in una specifica macchina virtuale, non superi la quantità fisicamente disponibile. In caso di sovrastima, le macchine virtuali subiranno il OOM (Out of memory).

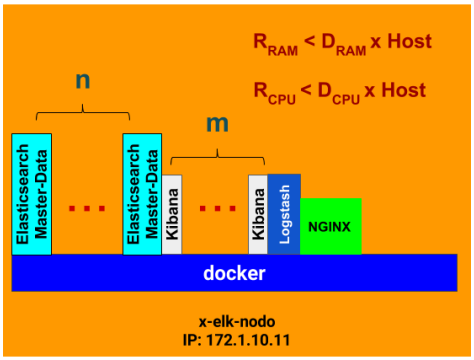


Figura 3.12: Visione d'insieme per macchina virtuale della richiesta di risorse fisiche.

3.5.4 Lo *Storage*

In contesto containerizzato con orchestratore, ho utilizzato le risorse di Kubernetes per isolare le dipendenze delle componenti applicative verso il server fisico che gestisce la memorizzazione dei dati e la loro disponibilità inter macchine virtuali.
Come *tool* per la condivisone fisica dei dati tra le macchine virtuali ho utilizzato il NFS (Networked File System). Ho valutato anche la tecnologia GlusterFS di RedHat. In seguito a una sessione di *brainstorming* con il mio tutor abbiamo deciso che la tecnologia per lo [storage distribuito] è molto valida e adatta per i container., tuttavia, aggiunge complessità al progetto. Come conseguenza di questa sessione, io e il mio tutor abbiamo confermato l'utilizzo della tecnologia NFS.

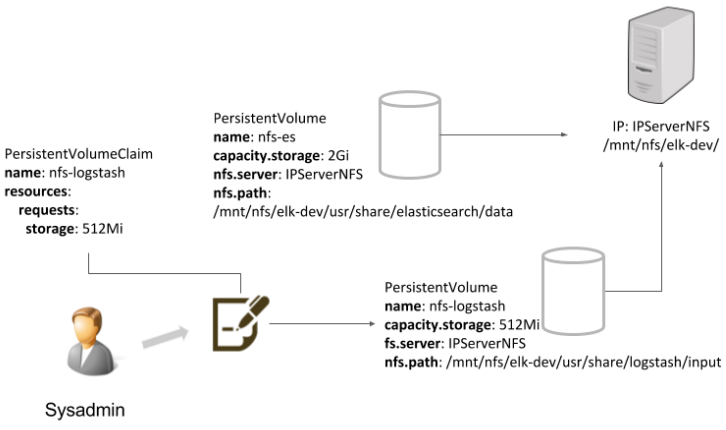


Figura 3.13: Organizzazione logica dei dati e i livelli di astrazione degli accessi ai dati.

Kubernetes ha il compito di orchestrare i container. Molto spesso, Kubernetes schedula un container su una VM e in seguito, in caso di non disponibilità di risorse fisiche (CPU, RAM), lo scheduler della piattaforma schedula l'esecuzione del container su un'altra macchina virtuale. Questo fenomeno è comunemente chiamato migrazione. Il modo con cui Docker e Kubernetes esportano i dati dal container sono i volumi. Tuttavia, i volumi introducono una dipendenza forte con il host che ospita il container in esecuzione. Inoltre, i volumi creati su un host non sono visibili e raggiungibili da qualunque altro host nella rete.
Kubernetes, tramite gli oggetti per la gestione dei volumi, elimina l'accoppiamento dei container con l'ambiente virtuale specifico di esecuzione.
In figura presento i livelli di astrazione usati da Kubernetes per garantire la visibilità dei dati tra VM durante la migrazione dei container. Gli oggetti per la gestione dei dati sono i PersistentVolume e PersistentVolumeClaims.
Con i PersistentVolume definisco una capacità di dati che può essere fisicamente allocata sul disco fisico di una macchina virtuale. E questa capacità è utilizzabile globalmente da tutti i microservizi che hanno accesso. Invece, con i PersistentVolumeClaim definisco un sottoinsieme della capacità specificata dai PersistentVolume. Questo modo di gestire i dati mi permette di rendere indipendente la modalità di memorizzazione dei dati su disco dalla modalità di consumo dei dati da parte delle componenti containerizzate. Una simile razionalizzazione dello storage mi offre la possibilità di portare i dati in qualsiasi ambiente, da un volume NFS a un volume sul cloud di Amazon oppure Azure ed ecc.
Con l'orchestratore la gestione dei dati è semplice. In assenza dell'orchestratore la complessità è proporzionale al numero dei container da gestire. Per l'ambiente senza orchestratore ho utilizzato ampiamente il pattern sidecar applicato allo storage. La rappresentazione del pattern segue in figura.

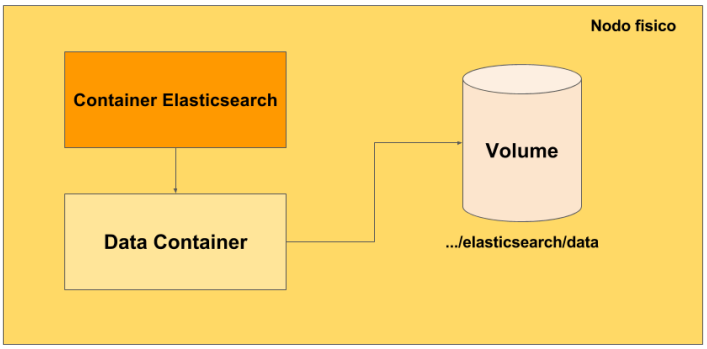


Figura 3.14: Riduzione della dipendenza dei dati dall'infrastruttura fisica in contesto senza orchestratore.

Un alias del sidecar pattern è il *data container*. Ho utilizzato questo pattern per progettare la l'accesso di scrittura e lettura dei dati a livello globale senza la necessità di specifica di indirizzi IP. Infatti, i container applicativi del sistema per accedere ai dati semplicemente utilizzano il nome del data container. Dal punto di vista dei

pattern di progettazione di dettaglio, un data container è un pattern Singleton.

3.5.5 Il *Networking*

La rete è un altro aspetto importante nel contesto della *light virtualization*. Kubernetes richiede come vincolo che ad ogni Pod venga assegnato un indirizzo IP univoco. La necessità di utilizzare un indirizzo IP univoco per container è necessaria per evitare la complessità di gestione delle porte logiche su cui esporre i servizi. Inoltre, Kubernetes gestisce il traffico tra container tramite regole di routing di basso livello e gestito in modo automatico. Il traffico del cluster Kubernetes è gestito dalla componente Kube-Proxy. Per assegnare indirizzi univoci ai Pod, ho utilizzato il plugin WeaveNet. Questo strumento mi ha permesso di creare una rete *full mesh* tra i container.

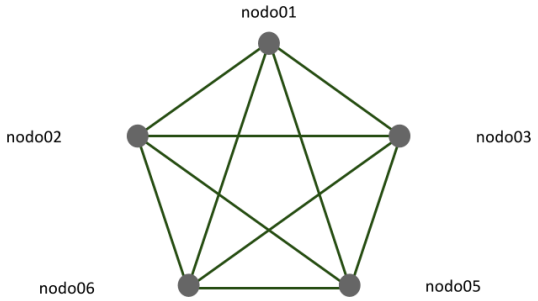


Figura 3.15: Visione di alto livello della topologia di rete.

WeaveNet crea una rete virtuale distribuita su tutte le macchine virtuali. L'obbiettivo principale dello strumento è l'assegnazione degli indirizzi IP alle risorse Kubernetes *on-demand*. La classe degli indirizzi allocabili dalla componente WeaveNet è di tipo A (10.1.0.0/16). Il prodotto WeaveNet è caratterizzato da un insieme di agenti che cooperano per gestire il routing del traffico di rete nel modo più ottimale possibile. Ho gestito il deployment degli agenti WeaveNet tramite l'utilizzo della risorsa DeamonSet. Questa risorsa è una estensione della risorsa Deployment. Tramite la risorsa del DaemonSet, in caso di aggiunta di nodi al cluster Kubernetes, il controller del cluster schedulerà dinamicamente una istanziazione dell'agente sul nuovo nodo senza alcun intervento manuale.
Ho utilizzato la ridondanza a livello della rete virtuale per incrementare l'alta affidabilità della rete stessa. Infatti, in caso di un problema con qualche agente WeaveNet, i pod possono continuare a comunicare con le altre componenti in modo trasparente.

3.6 Test

I test che ho effettuato durante lo stage, per la verifica del sistema, sono i test di carico e i test di durata. Ho valutato diversi strumenti open source idonei per questa tipologia di test. Lo strumento che ho scelto per effettuare i test è JMeter. La peculiarità

di JMeter, che mi ha impressionato, è il variegato insieme di plugin e l'insieme di configurazioni che esso supporta.

Configurazione VM	Valore
Sistema Operativo	CentOS7
Kernel	3.10.x86-64
CPU(s)	4
RAM	4GB

Tabella 3.4: Parametri di configurazione comuni delle macchiane vituali utilizzate durante i test.

Il taglio di macchine virtuali che ho utilizzato è medio-piccola. Ho scelto una simile configurazione perché è una categoria molto frequente in cloud. Questo taglio di macchine sono caratterizzate da un prezzo conveniente. Effettuare i test su questo taglio di macchine permette di paragonare i test fatti da altri con i risultati da me ottenuti.

3.6.1 Obiettivi dei test

Analizzare, progettare ed implementare i test è un forma d'arte. L'obiettivo principale dei test è lo studio del comportamento del sistema nel complesso e nell'ambiente containerizzato con orchestratore Kubernetes. Per effettuare i test ho fissato i limiti di memoria e CPU per ciascuna componente containerizzata.
Gli obiettivi che ho individuato per i test sono i seguenti:

* Studiare il comportamento di Elasticsearch a regime e sotto carico di lavoro con le configurazioni da me individuate;
* Individuare il punto di rottura del sistema, oltre al quale il livello di servizio degrada;
* Confrontare, a pari di configurazione, il sistema containerizzato con un sistema containerizzato.

3.6.2 Metodologia

In seguito alla preparazione dell'ambiente di test, ho individuato degli scenari possibili di test. Tramite uno scenario cerco di simulare l'attività dell'utente con il sistema. L'utente virtuale deve effettuare una serie di richieste di complessità sempre crescente. Le richieste riguardano la visualizzazione delle dashboard. Durante le prime visualizzazioni l'utente virtuale visualizza i singoli elementi alla base di una dashboard complessa. E successivamente tramite aggregazione l'utente effettua visualizzazioni sempre più complesse e numerose. Con JMeter ho codificato degli script per i scenari di test. Un esempio di script, che allego in figura per il test di durata, ha lo scopo di simulare 100 utenti che interagiscono con il sistema in un intervallo temporale di un'ora.
La configurazione utilizzata per lo scenario del test è come nella tabella seguente.

Parametro di configurazione	Valore)
Utenti	100
Tempo d'inizializzazione	20 minuti
Fasi di inizializzazione	10
Durata test	60 minuti

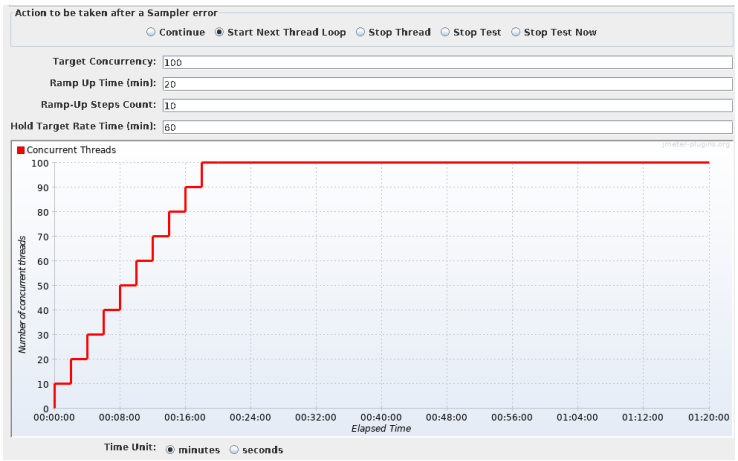


Figura 3.16: Script JMeter per i test di durata con 100 utenti, 1h di durata.

Tabella 3.5: Paramentri di configurazione di uno scenario di esempio del test di durata.

Come output, per ciascun test in parte ho generato un resoconto finale. Per la creazione dei report, ho progettato ed implementato un processo automatico. Ho scelto le ore notturne per effettuare i test e generare i report per i seguenti motivi:

* Utilizzare le ore di lavoro regolari per fare analisi e studio dei dati;
* Impiegare i server fisici di IKS al massimo durante le ore notturne;
* Ridurre al minimo gli impatti delle eventuali interferenze sulle macchine virtuali da me utilizzate.

3.6.3 Risultati

Durante i test, ho appreso che con specifiche configurazioni di memoria, CPU e topologia del cluster Elasticsearch il sistema è più responsivo. Ottenere una configurazione del sistema con specifici limiti sulle non è stato facile. Questa ricerca ha rafforzato la mia confidenza con l'amministrazione del cluster Kubernetes.
In figura a seguire, presento il comportamento dei tempi di risposta in relazione con il numero di utenti attivi nel sistema.
Qui, il numero dei thread rappresentano il numero di utenti attivi nel sistema. I dati mostrano che all'aumentare del numero degli utenti, per le tre attività di visualizzazione di dashboard, i tempi di risposta sono sempre al di sotto dei 9 secondi. Questo è un buon risultato, assumendo come limiti di attesa i 10 secondi per ottenere la visualizzazione di una pagina, durante l'attività di carico del sistema. Ho recuperato questa immagine da uno dei report che il processo di test ha generato.

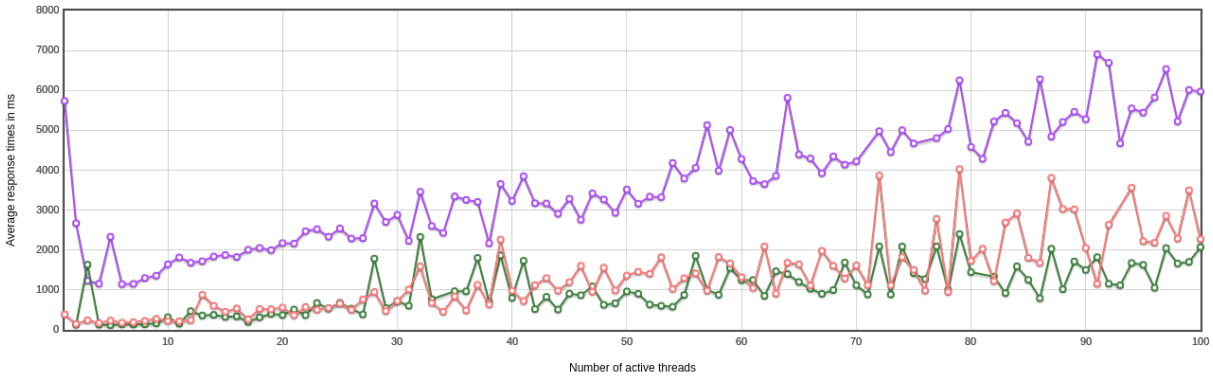


Figura 3.17: Andamento dei tempi di risposta all'aumentare del numero degli utenti.

3.7 Monitoraggio

Il monitoraggio delle applicazioni e dell'infrastruttura è un aspetto molto importante. Sia Docker sia Kubernetes non offrono soluzioni native di monitoraggio. I comuni strumenti di monitoraggio, orientati a macchine virtuali, non sono più idonei. Le macchine virtuali hanno un tempo di accensione di un ordine di grandezza superiore rispetto ai container. Se le macchine virtuali operano nell'ordine dei minuti, allora i container operano nell'arco dei secondi.
Visto che non esiste una soluzione nativa per l'esportazione delle metriche in Docker, ho utilizzato la componente cAdvisor che esporta i dettagli da monitorare nel comodissimo formatto JSON. Definito il modo per l'estrapolazione delle informazioni sui container, ho integrato cAdvisor con InfluxDB, un *timeseries database*, e Graphana, una componente di visualizzazione. Quest'ultima offre un familiare DSL (*Domain Specific Language*) in stile SQL (*Search Query Language*).
Un'ulteriore componente che ho utilizzato per il monitoraggio dell'infrastruttura è la *dashboard* di Kubernetes. Oltre al monitoraggio, la dashboard nativa di Kubernetes offre la possibilità di scrivere nuovi manifest, editare e cancellare quelli esistenti. Oltre alla creazione di risorse è possibile visionare i log per ciascun container da un unico punto di accesso. La dashboard centralizza la gestione del cluster.
In conclusione di stage, mi sono reso conto che la visione d'insieme del cluster e del sistema realizzato è poco chiara. Uno svantaggio di un sistema containerizzato è la mancanza della rappresentazione grafica delle relazioni tra le componenti. Questo è dovuto alla natura distribuita dei microservizi. Con questo fatto in mente, ho ridotto la difficoltà di gestione tramite uno strumento open source. Il nome di questo strumento è Weave Scope.

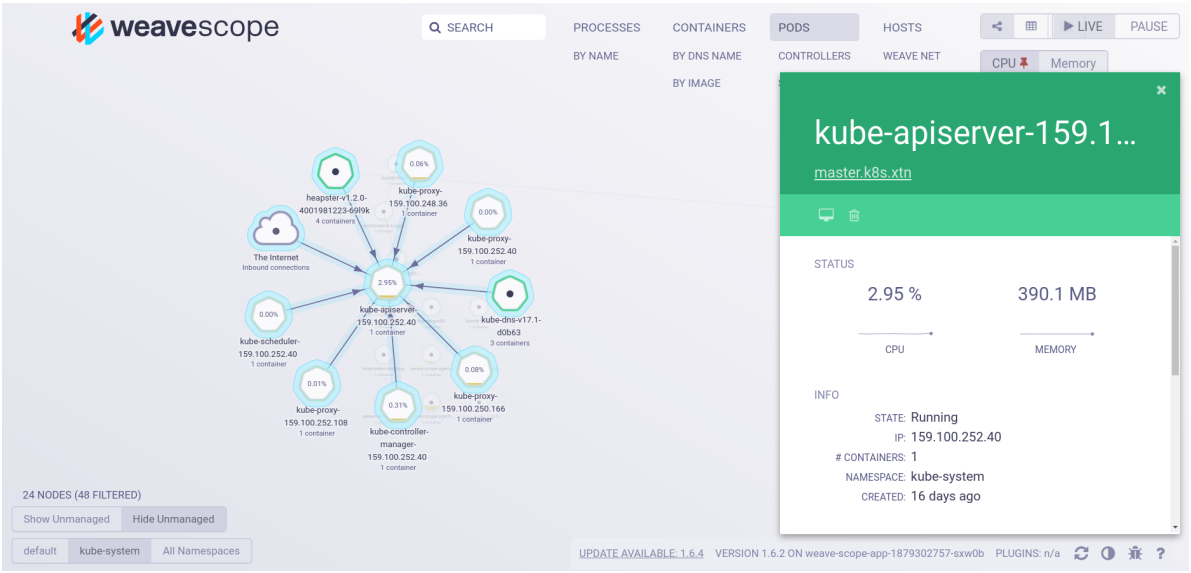


Figura 3.18: Grafo dei POD a supporto del cluster Kubernetes.

Per lo studio del consumo delle risorse, il prodotto di Weaveworks offre una vista dello stato delle risorse sul server. In figura illustro qual è lo stato delle risorse del nodo master del cluster K8s.

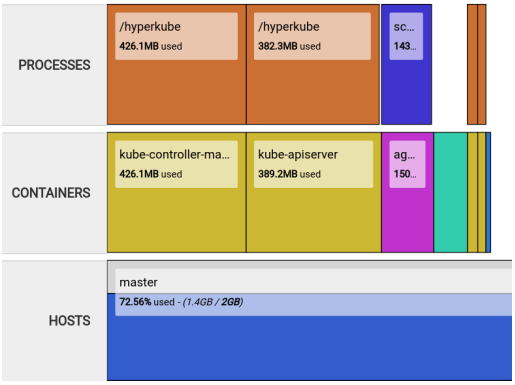


Figura 3.19: Vista del consumo di risorse su un nodo del cluster Kubernetes.

In conclusione, illustro la visione completa del sistema dal punto di vista delle componenti. Ogni componente è presentata in versione ridondata.

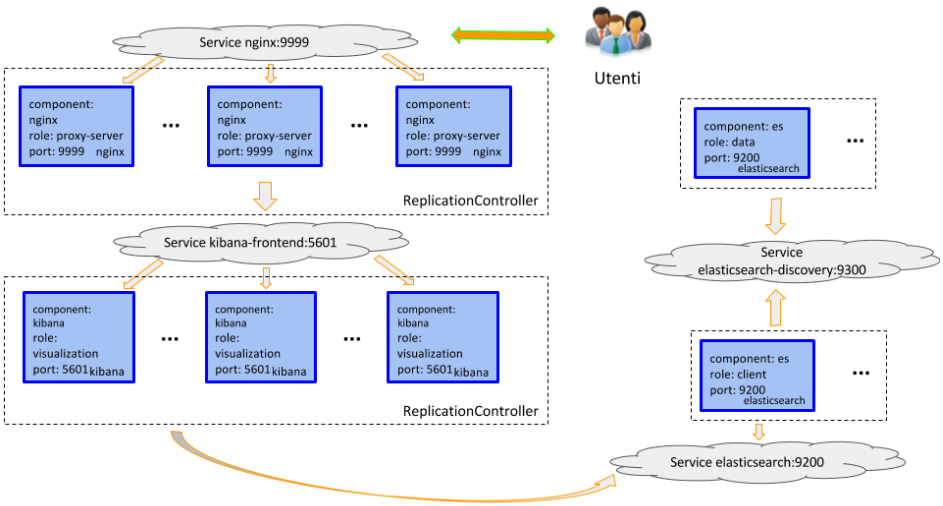


Figura 3.20: Visione completa delle componenti del sistema e delle loro relazioni.

Capitolo 4

Valutazioni retrospettive

4.1 Obiettivi raggiunti

4.2 Problematiche riscontrate

4.3 Bilancio formativo

4.4 Valutazione critica del Corso di Laurea

Glossario

Agile Metodo per lo sviluppo del software che coinvolge quanto pi‘u possibile il committente, ottenendo in tal modo una elevata reattivit‘a alle sue richieste. . 1, 49

Cloud Paradigma di erogazione di risorse informatiche, come l'archiviazione, l'elaborazione o la trasmissione di dati, caratterizzato dalla disponibilità on demand attraverso Internet a partire da un insieme di risorse preesistenti e configurabili. . 11, 17, 49

Framework Architettura logica di supporto su cui un software può essere progettato e realizzato, spesso facilitandone lo sviluppo da parte del programmatore. La sua funzione è quella di creare una infrastruttura generale, lasciando al programmatore il contenuto vero e proprio dell'applicazione. . vii, 4, 49

ICT insieme di metodi e tecnologie che implementano i sistemi di trasmissione, ricezione e elaborazione di informazioni.. 51

IT utilizzo di qualsiasi tecnologia di calcolo per offrire servizio di memorizzazione, reti per creare, processare, memorizzare e mettere in sicurezza ogni forma immaginabile di dato elettronico. . 51

Patching Applicare una patch, porzione di software progettata per aggiornare o migliorare un programma. Una patch permette di risolvere vulnerabilità di sicurezza e altri BugFix di un applicativo sviluppato. . 12, 49

Acronimi

ICT Information and Communication Technology. 1

IT Information Technology. 1

Riferimenti

Bibliografia

Baier, Jonathan. *Getting Started With Kubernetes*. PACKT, 2015.

Clinton Gormley, Zachary Tong. *Elasticsearch: The Definitive Guide*. O'Reilly Media, 2015.

Rafal Kuć, Marek Rogoziński. *Elasticsearch Server*. PACKT, 2016.

Turnbull, James. *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2016.

Vohra, Deepak. *Kubernetes Microservices With Docker*. Apress, 2016.

Sitografia

Documentazione Docker. URL: https://docs.docker.com/.

Documentazione Elasticsearch. URL: https://www.elastic.co/guide/en/elasticsearch/reference/2.4/index.html.

Documentazione Kibana. URL: https://www.elastic.co/guide/en/kibana/4.6/index.html.

Documentazione Kubernetes. URL: https://kubernetes.io/docs/home/.

Documentazione Linux. URL: http://www.tldp.org/.

Documentazione Logstash. URL: https://www.elastic.co/guide/en/logstash/2.3/index.html.

Documentazione Nginx. URL: https://nginx.org/en/docs/.

Martin Fawler: Micorservices. URL: https://martinfowler.com/articles/microservices.html.

Pattern architetturale a microservizi. URL: http://microservices.io/index.html.

Wikipedia: Cloud computing. URL: https://en.wikipedia.org/wiki/Cloud_computing.

Wikipedia: Microservices. URL: https://en.wikipedia.org/wiki/Microservices.