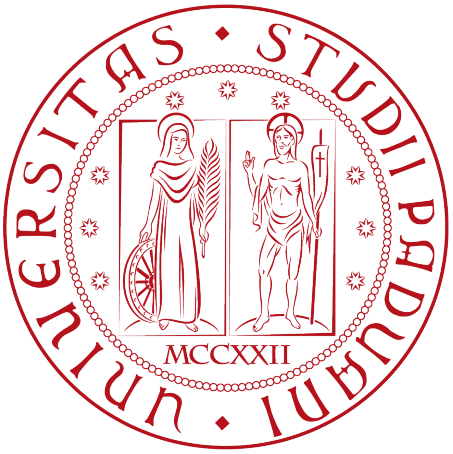
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Test di carico in ambiente enterprise: un effettivo caso di studio

Tesi di laurea triennale

Relatore

Prof.Francesco Ranzato

Laureando

Gianluca Marraffa

Anno Accademico 2017-2018

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata trecentoventi ore, dal laureando Gianluca Marraffa presso l'azienda Infocert S.p.A.
Durante il periodo di tirocinio il laureando è stato inserito all'interno del team Legalmail, responsabile dello sviluppo del prodotto PEC Legalmail dell'azienda.
Lo scopo ultimo dello stage si proponeva come la progettazione e l'installazione di una infrastruttura per test di carico capace di evidenziare e documentare le capacità e i limiti degli applicativi aziendali. In primo luogo si chiedeva un'analisi dello stato dell'arte dei vari strumenti di Load Testing presenti sul mercato volta a sottolineare pro e contro delle varie soluzioni, successivamente si sarebbe scelta la soluzione più adatta alle caratteristiche aziendali e del prodotto in esame.
In secondo luogo in secondo veniva richiesta l'effettiva implementazione dell'infrastruttura concordata, e l'esecuzione di test di carico negli ambienti di test e accettazione.
Per ultimo veniva richiesta la stesura di un documento che esponesse le caratteristiche dell'infrastruttura implementata in modo da estendere il lavoro effettuato, e di un manuale d'utilizzo per il team Legalmail.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Il Contesto Aziendale

1.1 Profilo Aziendale

InfoCert è leader del mercato italiano nei servizi di digitalizzazione e dematerializzazione nonché una delle principali Certification Authority a livello europeo per i servizi di Posta Elettronica Certificata, Firma Digitale e Conservazione digitale dei documenti (Conservatore Accreditato AgID).
Da dicembre 2015 InfoCert è anche gestore accreditato AgID dell'identità digitale di cittadini e imprese, in conformità ai requisiti regolamentari e tecnici dello SPID.
InfoCert si pone sul mercato come un partner altamente specializzato nei servizi di dematerializzazione, capace di garantire ai propri clienti la piena innovazione nei processi di gestione del patrimonio documentale e informativo.
Con sedi a Roma, Milano e Padova, InfoCert rivolge la propria offerta sia alle imprese, pubbliche e private, operanti nel settore Bancario, Assicurativo, Farmaceutico, Manifatturiero, Energy, Utilities, Distribuzione Commerciale, Ambiente, Qualità, Sicurezza, Sanità, Pubblica Amministrazione; sia ad Associazioni di Categoria, Ordini Professionali e Professionisti.

Figura 1.1: Logo di Infocert S.p.A

1.2 Legalmail

La Posta Elettronica Certificata (detta anche posta certificata o PEC) Legalmail di InfoCert è un sistema di comunicazione simile alla posta elettronica standard con in più alcune caratteristiche di sicurezza e di certificazione della trasmissione che rendono i messaggi opponibili a terzi.
La PEC Legalmail consente infatti di inviare/ricevere messaggi di testo e allegati con lo stesso valore legale di una raccomandata con avviso di ricevimento.
Dal punto di vista tecnologico, il prodotto si presenta come un'applicazione web scomposta in diversi microservizi, ognuno responsabile di gestire una particolare funzionalità applicativa. La maggior parte di questi servizi è suddiviso in backend (la logica applicativa) e frontend (l'interfaccia utente) e comunicano con gli altri microservizi attraverso REST API.
Le varie componenti sono tante e non avrebbe senso elencarle tutte, per il progetto di tirocinio le parti da me interessate sono state quelle del backend della dashboard informativa e la gestione delle code di priorità per errori e segnalazioni.

1.3 Tecnologie Utilizzate

Pur essendo molto collaborativi, sviluppatori e operatori utilizzano tecnologie diverse per soddisfare i requisiti richiesti. I primi infatti usano principalmente linguaggi di programmazione per sviluppare le nuove funzionalità, mentre i secondi adoperano anche linguaggi dichiarativi per configurare i vari ambienti di esecuzione.

1.3.1 Dev

Gli sviluppatori seguono la classica divisione in backend developers, responsabili della logica di business dell'appplicativo, e frontend developers, responsabili dell'interfacciamento tra utente e applicazione.

Java 8 Enterprise Edition

Lo standard scelto per lo sviluppo della business logic dei vari applicativi è Java[1], linguaggio di programmazione orientato agli oggetti e spesso presenza indiscussa nella maggior parte degli ambienti enterprise per una moltitudine di obiettivi, tra i principali:

* **Maturità**: Oracle[2] rilascia la prima versione di Java nel 1995 e conquista col tempo la maggior parte dei software sviluppati, rivelandosi fin da subito un componente robusto e sicuro per lo sviluppo di applicazioni. Le aziende di grosso taglio si sentono quindi al sicuro nell'adottare Java, assicurandosi un prodotto ben consolidato;

* **Curva d'apprendimento**: nei suoi principi base, Java è un linguaggio molto semplice e, essendo i team di aziende enterprise composti da molte persone, è molto facile formare nuovi sviluppatori all'utilizzo della tecnologia.
  Inoltre la sua diffusione ha portato negli anni alla creazione di una moltitudine di corsi ben strutturati, conquistando spazi anche nelle università;

[1] *Java*. URL: https://www.java.com/.
[2] *Oracle*. URL: https://www.oracle.com/.

- **Funzionalità**: essendo la community di sviluppatori Java molto diffusa, il mondo open source è pieno di librerie che risolvono problemi comuni a molti casi d'uso, velocizzando i tempi di sviluppo e riducendo l'introduzione di nuovi errori;
- **Multipiattaforma**: un applicativo Java viene scritto una sola volta e, con pochi accorgimenti, può essere distribuito su moltissime piattaforme, permettendo all'azienda di distribuire i propri prodotti senza incappare in "barriere architettoniche";
- **Scalabilità**: i prodotti di stampo enterprise sono utilizzati da parecchie persone, per questo l'infrastruttura che li ospita deve essere flessibile ad aumenti o diminuzioni di traffico. Java, grazie alle numerose integrazioni esistenti, si presenta come un ottimo candidato per queste necessità.

L'azienda ha sviluppato su di esso un framework proprietario chiamato BEcon.

Angular JS

Per la parte frontend dei vari prodotti, Infocert ha sviluppato un framework interno, FEcon, basato su AngularJS[3]: un framework strutturale per la creazione di applicazioni web dinamiche.

1.3.2 Ops

Gli operatori non hanno distinzioni interne e utilizzano Puppet per la configurazione delle macchine e Java e/o Python per la realizzazione dei test d'accettazione e sviluppo delle sonde di monitoraggio.
Le sonde di monitoraggio sono piccoli applicativi che verificano lo stato dei server nei vari ambienti d'esecuzione; in caso di macchine non operative avvisano i relativi responsabili.

Puppet

Puppet è uno strumento di configuration management che permette di definire lo stato delle macchine tramite un linguaggio dichiarativo dedicato, garantendo il versionamento dell'infrastruttura e la replica delle configurazioni su più macchine con il minimo sforzo. Operando le modifiche ai server esclusivamente tramite Puppet è infatti possibile mantenere consistente lo stato dell'infrastruttura, rendendo più rapida la propagazione delle modifiche in flotte composte da molti server.
Un altro punto di forza di Puppet è quello di definire delle classi tramite il linguaggio proprietario dell'applicativo, permettendo ai meno esperti di utilizzarle senza conoscerne l'implementazione interna (proprio come una classe vera e propria, con metodi privati e pubblici) e rendendo più accessibile la gestione della configurazione. Queste classi infatti possono essere configurate tramite file YAML, rendendone possibile l'utilizzo anche a chi non è formato per utilizzare il linguaggio interno di Puppet.

Python

Python è un linguaggio di programmazione non tipizzato che da tempo sta conquistando sempre più aree dello sviluppo software. Grazie alla sua bassa curva d'apprendimento e alle innumerevoli librerie open-source presenti sul mercato, risulta la scelta ideale

[3] *AngularJS*. URL: https://angularjs.org/.

per molte soluzioni software, tra le quali troviamo le utility di scripting, il calcolo distribuito, il data mining e lo sviluppo dei test di sistema.
La forza di python sta nel permettere la realizzazione di compiti complessi tramite la scrittura poche righe di codice, mantenendo alta la leggibilità dello stesso.

1.4 Metodologia Aziendale

1.4.1 Metodologia SCRUM

La metodologia utilizzata dal team Legalmail (e dalla maggior parte dei restanti team in Infocert) è quella Agile, implementata tramite il framework SCRUM. Questa metodologia impone un'approccio semplice ma efficace per la gestione del ciclo di vita del software e fa leva sulla capacità auto organizzativa del team di sviluppo.
In un team SCRUM infatti sono presenti solo 3 categorie di elementi e sono pensate per permettere agli sviluppatori di concentrarsi principalmente sul lavoro da effettuare piuttosto che all'interazione con gli stakeholders.

* **Product Owner**: il ruolo del Product Owner è cruciale in quanto rappresenta il ponte tra team di sviluppo e stakeholders. Il PO hai infatti il compito di convertire i requisiti del cliente in items usabili dal team di sviluppo, oltre ha comunicare eventuali variazioni delle tempistiche agli stakeholders;

* **Scrum Master**: il ruolo dello Scrum Master è essenziale per permettere il corretto svolgimento dello sprint, questo infatti s'impegna a rimuovere ostacoli nello sviluppo (come improvvisa mancanza di personale, assegnamento di maggiori risorse su un particolare compito etc.) e a variare gli obiettivi fissati nello sprint planning in base al feedback offerto dagli sviluppatori;

* **Team di Sviluppo**: Il team di sviluppo rappresenta tutti i membri responsabili delle varie fasi del ciclo di vita del software ed è completamente autonomo: al suo interno ci sono tutte le competenze necessarie per iniziare e finire il progetto. I componenti del gruppo sono in grado di organizzarsi da soli e assegnarsi le attività che ritengono più appropriate in modo completamente indipendente.
  Avendo l'azienda abbracciato la filosofia DevOps i team di sviluppo si dividono internamente in:

  – **Dev**: responsabili di progettazione e sviluppo delle funzionalità applicative;
  – **Ops**: responsabili della fase di testing e distribuzione delle nuove release, monitoraggio della produzione, verifica e rispetto degli SLA contrattualizzati, assolvimento delle procedure normative e automazione dei compiti ripetitivi.

I momenti formali che prevede questa metodologia sono chiamati eventi e sono *Sprint*, *Sprint Planning*, *Daily Scrum*, *Sprint Review* e *Sprint retrospective*.
Gli eventi a cui ho avuto modo di partecipare attivamente sono stati lo Scrum Daily e lo Sprint. Il primo è una breve riunione di massimo 15 minuti effettuata nel primo mattino atta ad esporre ai colleghi le attività lavorative svolte nella precedente giornata e ad organizzare le prossime 24 ore, durante la riunione vengono esposti anche eventuali problemi sorti durante lo sviluppo in modo da aggiornare i colleghi sullo stato del lavoro che si sta svolgendo. Durante questa breve riunione viene aggiornata la SCRUM board con le informazioni rilevate, permettendo allo SCRUM master di adeguare la visione

generale dello stato del progetto. La SCRUM board è una lavagna che tiene traccia dei task che i Dev e gli Ops devono svolgere durante lo sprint e permettono allo Scrum master di avere una buona visione d'insieme del suo andamento. La struttura della SCRUM board è a discrezione delle esigenze del team, per quanto riguarda Legalmail ogni riga rappresenta un componente del team mentre le colonne sono divise in:

* **Todo**: I compiti assegnati al componente ma non ancora presi in carico dallo stesso;
* **Progress**: I compiti assegnati che il componente ha iniziato a svolgere;
* **Impediment**: I compiti che non possono proseguire perché bloccati da impedimenti al di fuori della responsabilità del componente;
* **Developed**: I compiti sviluppati dal componente che devono essere verificati;
* **Testing**: I compiti in fase di verifica;
* **Failed**: I compiti che non hanno soddisfatto i requisiti richiesti;
* **Done**: I compiti che sono terminati ed hanno ricevuto l'approvazione da parte dello SCRUM master.

Dopo poche partecipazioni ho assorbito l'importanza di questo momento: il fatto di esporre a tutto il team il problema riscontrato permette di giungere più facilmente ad una soluzione. Può essere infatti che un membro esterno alla vicenda (il compito da svolgere) si sia imbattuto in un problema simile nel passato e abbia trovato una soluzione, ma il componente colpito dal problema non essendone a conoscenza non può confrontarsi con il diretto interessato. Grazie a questi 15 minuti di condivisione invece, il collega può prendere visione del problema e proporre la sua soluzione, permettendo al team di risparmiare tempo e procedere velocemente verso la conclusione del task assegnato. Lo sprint è invece l'effettiva finestra temporale nella quale si prefissano degli obiettivi che dovranno essere completati alla fine della stessa. All'interno del team Legalmail la durata di uno sprint è stata fissata a 14 giorni (10 giorni lavorativi). Ogni informazione aggiuntiva può essere ricavata visitando il sito ufficiale della metodologia SCRUM.

1.4.2 Infrastruttura e Applicazioni

Infocert è un'azienda che sviluppa prodotti di stampo Enterprise: soluzioni dalla grande distribuzione rivolte ad aziende molto grandi e importanti. Per garantire qualità in termini di prodotto ai suoi clienti, l'azienda ha negli anni maturato diverse certificazioni necessarie per lo sviluppo di certi tipi di applicativi, queste certificazioni si portano l'onere di rispettare imposizioni particolari, imponendo una determinata way of working aziendale. I server utilizzati per conservare determinati tipi di dati, ad esempio, devono rispettare caratteristiche di sicurezza non banali, impedendo ai team dei vari prodotti di gestire in totale libertà la loro infrastruttura.
L'azienda ha quindi predisposto un team, *Infrastructure* per l'appunto, dedito alla definizione e implementazione di standard per la creazione delle macchine virtuali utilizzate per ospitare le applicazioni, lasciando come unica libertà agli *ops* quella di configurare, nei limiti definiti, le macchine richieste.
Questa specificazione non è atta in alcun modo a screditare le modalità operative aziendali, ma è stata introdotta per giustificare alcune scelte adoperate per portare

a termine il progetto di stage che, come descritto nei capitoli successivi, ha dovuto accettare dei compromessi per essere integrato nell'infrastruttura aziendale.

1.5 Strumenti Utilizzati

1.5.1 Jira

Jira[4] è uno strumento rilasciato da Atlassian per gestire tutta la componente organizzativa del ciclo di vita del software, è ideato per supportare i modelli agili ed è lo strumento più utilizzato dalle aziende che ne adottano le metodologie.
Offre funzionalità di *Project Managment*, *Issue Tracking*, *Agile Reporting* e *Roadmap Planning*. All'interno dell'azienda viene ad esempio utilizzato per la gestione delle SCRUM board e del release planning, oltre che a tutti gli aspetti della gestione di progetto.

1.5.2 Confluence

Confluence[5] è uno strumento rilasciato sempre da Atlassian per la gestione della parte documentale non per forza legata alla documentazione del software: descrizioni di procedure, guide per installazioni, inventario indirizzi macchine virtuali e tutte le informazioni necessarie per distribuire la conoscenza aziendale a tutti i dipendenti.
L'azienda ha adottato Confluence per racchiudere in un punto centrale tutte queste informazioni, permettendo anche di commentare i documenti scritti da altre persone in modo da migliorare attivamente la qualità della documentazione.
Una parte del mio progetto consisteva nello scrivere una guida su Confluence per l'instanziazione dell'infrastruttura di test ed il lancio dello stesso.

1.5.3 Gitlab

Il versionamento degli applicativi viene fatto tramite Git[6] e i repository sono ospitati su Gitlab[7]: una piattaforma che permette la gestione centralizzate dei repository Git, permettendo l'amministrazione dei permessi d'accesso tramite una semplice interfaccia grafica. Tutti gli applicativi e le configurazioni dell'infrastruttura vengono ospitati sulla piattaforma e il versionamento segue le regole del Git-Flow: le funzionalità nuove vengono sviluppate su un branch dedicato, per poi essere spostate successivamente nei branch di sviluppo, accettazione e infine produzione.
Il rispetto del Git-Flow non ha solo scopo formale: la definizione corretta dei branch scatena tutto un processo che dall'esecuzione dei test effettua il deploy dell'applicazione su Artifactory.
Questo procedimento prende il nome di Integrazione Continua e viene gestito da Jenkins.

1.5.4 Jenkins

Jenkins[8] è uno strumento open source sviluppato dalla Jenkins CI community che permette l'implementazione di sistemi di Integrazione Continua su web-server Java.

[4] *Jira*. URL: https://it.atlassian.com/software/jira.
[5] *Confluence*. URL: https://it.atlassian.com/software/confluence.
[6] *Git*. URL: https://git-scm.com/.
[7] *Gitlab*. URL: https://about.gitlab.com/.
[8] *Jenkins*. URL: https://jenkins.io/.

Jenkins è molto flessibile, permette la sua esecuzione in modo automatizzato e/o manuale ed è predisposto per essere esteso tramite plugin scritti dalla community.
All'interno dell'azienda la sua esecuzione è innescata dai commit su gitlab e, in base al branch di appartenenza del commit, scatena una serie di operazioni volete ad automatizzare la verifica, validazione e distribuzione su Artifactory del software.

1.5.5 Artifactory

Artifactory[9] è uno strumento di repository per pacchetti applicativi (binaries) sviluppato da JFrog, questo permette il versionamento delle build e una conseguente diminuzione dei tempi di deploy dell'applicativo. All'interno dell'azienda, Jenkins effettua la build dei codici sorgenti e li carica su Artifactory; questi pacchetti, stabili e versionati, vengono poi indirizzati dai sistemi di configuration management per essere installati sui server operativi.

1.5.6 Puppet Enterprise

Puppet Enterprise[10] è la versione Enterprise dello strumento di configuration management Puppet, questa piattaforma offre una semplice gestione della configurazione per grandi flotte di server in modo completamente automatizzato e parallelo, con la possibilità di gestire la propagazione delle modifiche tramite un'intuitiva interfaccia grafica.
All'interno dell'azienda lo strumento viene utilizzato per applicare le modifiche ai server in modo rapido e affidabile.

1.5.7 Foreman

Foreman[11] è uno strumento per la completa gestione del ciclo di vita delle macchine fisiche e/o virtuali. Si presente come un'interfaccia grafica e permette l'approvvigionamento dei server sia su datacenter privati che soluzioni nel cloud.
All'interno dell'azienda il team infrastrutture crea dei modelli di macchine interfacciandosi al datacenter privato, mentre i team di sviluppo personalizzano questi modelli per creare le macchine necessarie al funzionamento dell'applicativo desiderato.

1.5.8 Oracle Secure Global Desktop

Oracle secure global desktop[12] è uno strumento sviluppato da Oracle per l'accesso sicuro alle macchine virtuali da remoto.
Nel contesto aziendale viene utilizzato per centralizzare la definizione dei permessi di accesso alla flotta dei server del datacenter privato.

1.5.9 Ambienti di sviluppo

L'azienda non pone limitazioni per quanto riguarda gli IDE, tuttavia mostra una forte preferenza per i software sviluppati da JetBrains[13], in particolare *Intellij IDEA*, per Java, *PyCharm* per Python e *Webstorm* per Javascript.

[9] *Artifactory*. URL: https://jfrog.com/artifactory/.
[10] *Puppet*. URL: https://puppet.com/.
[11] *Foreman*. URL: https://www.theforeman.org/.
[12] *Oracle Secure Global Desktop*. URL: https://www.oracle.com/it/secure-global-desktop/.
[13] *Jetbrains*. URL: https://www.jetbrains.com/.

Necessitando il mio progetto di diverse piattaforme d'esecuzione, ho optato per *Visual Studio Code*[14], IDE sviluppato da Microsoft[15] e consigliato per lavorare per progetti multi linguaggio, in quanto estensibile tramite plugin per supportare la maggior parte dei linguaggi di programmazione.

1.6 Propensione all'innovazione

Infocert pone l'innovazione alla base della crescita dell'azienda, non solo infatti la compagnia ha già ottenuto diverse certificazioni di qualità, ma una buona parte degli investimenti viene riposta in ricerca e sviluppo, partendo dall'acquisizione di società innovative fino alla collaborazione con le università per gli stage accademici, che consentono di concentrare risorse unicamente sulla ricerca e lo sviluppo di nuove ed efficaci soluzioni.
Il personale lavorativo inoltre è spesso coinvolto in processi di formazione aziendale, iniziando dai corsi individuali per arrivare alla partecipazione a workshop e conferenze per rimanere aggiornati sulle ultime novità in ambito tecnologico.
Grazie ad una concreta Corporate Social Responsability e ad un robusto codice etico, Infocert sostiene in modo fervido l'azienda ecosostenibile, dalla formazione del singolo fino all'organizzazione dei processi lavorativi.

[14] *Visual Studio Code*. URL: https://code.visualstudio.com/.
[15] *Microsoft*. URL: https://www.microsoft.com/.

Capitolo 2

Interessi e Aspettative

2.1 Progetto Test di Carico

2.1.1 Descrizione

Il progetto Test di Carico, nella sua accezione più semplice, si traduce in un effettivo caso di studio dello stato dell'arte del mondo del **Performance Testing**.
Questo processo è una categoria di test di sistema essenziale per applicazioni esposte a grandi flussi di utenza e con vincoli di qualità stringenti riguardanti le performance del sistema. Lo scopo finale di questi test è infatti quello di misurare le prestazioni di un applicativo a svariati livelli di utilizzo permettendo di determinarne eventuali pregi e difetti, e attestare la qualità del software prima del rilascio.
Applicativi rilasciati senza includere i Performance Test nella fase di validazione spesso presentano problemi di reattività, peggiorando l'esperienza dell'utente e causando cattiva reputazione del servizio.
Esistono svariate declinazioni dei performance test, tutte concentrate sui vari aspetti dei possibili problemi di prestazioni e incentrate su determinati tipi di metrica.
Lo stage si è concentrato su due particolari categorie di questi test, i **Test di carico** e gli **Stress Test**.

Test di carico

I test di carico si concentrano sul verificare le abilità del sistema a rispondere a flussi di carico predeterminati, ricavati da aspettative di popolarità che l'applicativo dovrà sostenere. In questo particolare tipo di test infatti si applica il traffico desiderato al sistema, verificando che i tempi di risposta (e quindi l'esperienza utente) siano coerenti con i valori indicati negli SLA.
Dai risultati prodotti da questo test si deduce se l'infrastruttura a supporto dell'applicativo sia adeguata a sostenere il flusso d'utenza che gli studi di mercato predicono.

Stress Test

Gli stress test tendono invece ad individuare il limite superiore delle performance di un sistema, verificando la robustezza di un applicativo all'insorgere di flussi d'utenza non previsti inizialmente.
Questo processo quindi viene fatto aumentando gradualmente le richieste al sistema

fino all'insorgere di risposte non previste per arrivare eventualmente alla rottura del sistema, ovvero quando il software smette completamente di rispondere.

La prima parte dello studio si è quindi concentrata sulla ricerca delle soluzioni software presenti sul mercato che permettano di simulare il comportamento di un grande flusso di utenti, software racchiusi nella categoria dei **Load testing tools**.
Essendo Legalmail un prodotto enterprise il traffico d'utenza usuale è molto alto, richiedendo un grande sforzo computazionale per simularne il volume; per questo la seconda parte dello studio è stata rivolta all'analisi delle soluzioni nel cloud che permettano la simulazione di questi carichi in caso l'infrastruttura interna dell'azienda non ne fosse in grado.
Una volta ottenute le basi teoriche sopraelencate si sarebbe progettata ed implementata un'infrastruttura che permettesse l'esecuzione di questi test in modo *semplice*, *flessibile*, *ripetibile*, ed *automatizzabile*.
Per un'efficacie realizzazione dei test di performance non è sufficiente l'implementazione dell'infrastruttura per l'esecuzione dei test, ma si necessita di strumenti di monitoraggio delle macchine target e ingegnerizzazione della reportistica. Tuttavia, essendo il tempo di tirocinio limitato, lo scopo ultimo dello stage è quello di creare una solida base per l'esecuzione di questi test, in modo da essere estensibile per eventuali sviluppi futuri.

2.1.2 Obiettivi

A seguito delle necessità descritte nella sezione precedente, assieme al tutor aziendale sono stati definiti gli obiettivi da portare a termine per considerare il tirocinio riuscito. Questi sono stati divisi in due categorie così descritte:

* *MIN*: requisiti minimi, vincolanti in quanto obiettivo primario richiesto dal committente;
* *MAX*: requisiti massimi (comprendenti quelli desiderabili e opzionali), non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;
* *FOR*: obiettivi formativi, rappresentanti valore aggiunto non strettamente competitivo.

Le sigle precedentemente indicate saranno seguite da una coppia sequenziale di numeri, identificativo del requisito.

Gli obiettivi fissati sono dunque:

* **Minimi**
  - *MIN01*: Individuazione degli strumenti idonei agli obiettivi dello stage (strumenti di load test, soluzioni cloud);
  - *MIN02*: Realizzazione infrastruttura di sviluppo per test di carico;
  - *MIN03*: Installazione suite test di carico con relativi casi di test;
  - *MIN04*: Documentazione di Progettazione e Sviluppo dei componenti menzionati nei punti precedenti;
* **Massimi**

  - *MAX01*: Realizzazione infrastruttura di test e accettazione;
  - *MAX02*: Esecuzione test di carico su infrastruttura di accettazione;
  - *MAX03*: Risultati dei test di carico in ambiente di accettazione;

* **Formativi**

  - *FOR01*: Collegato a *MIN01*. Acquisizione teorica dei principi dei test di carico e gestione delle infrastrutture;
  - *FOR02*: Collegato a *MIN02*. Acquisizione pratica di strumenti per la gestione delle infrastrutture;
  - *FOR03*: Collegato a *MIN03*. Acquisizione pratica sull'utilizzo di uno strumento dei test di carico;
  - *FOR04*: Utilizzo della metodologia agile in ambito aziendale;

2.1.3 Pianificazione Temporale

La pianificazione temporale segue la durata degli Sprint aziendali, ovvero due settimane. Alla fine di ognuno viene collegata una *milestone*, alla quale sono associati quelli che si ritiene saranno i traguardi raggiunti entro ogni corrispondente scadenza. Le milestone fissate sono le seguenti:

* *M1*: Fine primo sprint, analisi di strumenti per test di carico e reportistica, selezione infrastruttura;
* *M2*: Fine secondo sprint, predisposizione ambiente di sviluppo con relativa installazione dei tools oggetto di stage;
* *M3*: Fine terzo sprint, predisposizione e dimensionamento ambienti di test, accettazione;
* *M4*: Fine quarto sprint, predisposizione e dimensionamento ambiente di produzione, primo test di carico in ambiente di accettazione;
* *M5*: Fine quarto sprint, documentazione infrastruttura, test di carico, analisi risultati e report finale con presentazione della piattaforma al team.

2.1.4 Prodotti Attesi

A seguito degli obiettivi fissati sono stati definiti dei prodotti che potessero attestare il soddisfacimento dei requisiti:

1. Definizione infrastruttura per gli ambienti di sviluppo, test, accettazione e produzione.
   Ciascun ambiente sarà composto da:
   * Strumenti per l'esecuzione di test di carico;
   * Suite di test.
2. Documentazione del lavoro svolto per per permetterne la replica e l'estensione.

2.1.5 Vincoli Metodologici

La realizzazione dello studio e la progettazione dell'infrastruttura non hanno subito limitazioni durante il loro svolgimento, ogni obiettivo è stato perseguito in piena libertà, ponendo come unico obbligo la documentazione adeguata di ogni scelta progettuale e metodologica effettuata.
L'azienda ha inoltre posto forte attenzione sulla mia partecipazione agli eventi SCRUM affrontati nel periodo di stage, in modo da ampliare la mia formazione sulle metodologie Agile.

2.1.6 Vincoli Tecnologici

L'azienda non ha imposto veri e propri vincoli sulle tecnologie da utilizzare per portare a termine gli obiettivi. Tuttavia, sia per la parte infrastrutturale che per quella applicativa, sarebbe stato preferibile proporre una soluzione che utilizzasse linguaggi di programmazione già adottati all'interno del team Legalmail, in modo da abbattere l'eventuale debito tecnico d'ingresso.
Eventuali prodotti che interagissero con l'infrastruttura esistente avrebbero comunque dovuto rispettare i vincoli tecnologici della stessa.

2.2 Interessi Aziendali

2.2.1 Stage nella strategia aziendale

Come sottolineato nella sezione Propensione all'innovazione, la collaborazione con le università svolge un ruolo cardine per la ricerca e lo sviluppo di soluzioni software all'avanguardia.
Grazie alla collaborazione con Università e Centri di Ricerca (Università degli studi di Padova, Politecnico di Milano, EXO Organismo di ricerca, Università di Tor Vergata, Università di Salerno e SDA Bocconi) Infocert riesce a proporre soluzioni sempre migliori a propri clienti, permettendole di sperimentare tecnologie nuove tramite le eccellenze prodotte dal sistema accademico italiano.
Tramite questi stage inoltre l'azienda è in grado di avvicinare e formare nuovi possibili lavoratori, offrendo in cambio esperienza diretta in un ambiente produttivo e professionale.
Gli stage offerti agli studenti dell'Università degli studi di Padova, ad esempio, sono finalizzati all'assunzione nel prossimo periodo ma senza obblighi contrattuali, permettendo al tirocinante di ricavare il meglio dall'esperienza offerta e all'azienda di ampliare il proprio know how tecnologico.

2.2.2 I test di carico nel ciclo di vita del software

Infocert tramite questo progetto formativo mira ad impostare le basi per l'inserimento in modo costante dei test di carico nel ciclo di vita del software da loro rilasciato.
Il team Legalmail già in passato ha effettuato test di questo tipo ma, a causa di stretti vincoli temporali e dall'impossibilità di posizionare risorse esclusivamente su questo progetto, non ha avuto modo di creare un'infrastruttura che rendesse questo processo ripetibile e semplice da implementare. Il gruppo infatti ha da tempo abbracciato la filosofia DevOps, sviluppando framework interni ottimizzati per aderire alla pratica dell'Integrazione Continua, proprio per questo motivo il team ha estremo interesse

nello sviluppare un framework facilmente integrabile nella loro piattaforma anche per l'esecuzione dei test di carico.
Per l'azienda quindi, l'inserimento di una risorsa che si occupi di questo progetto a tempo pieno, risulta una scelta molto efficace.
L'esecuzione dei test di carico, inoltre, presenta vantaggi non indifferenti e ha valenza informativa non solo per l'area tecnica ma anche per gli stakeholders: la prima infatti può analizzare i dati ottenuti per pianificare l'ampliamento o la riduzione della flotta dei server, mentre i secondi possono utilizzare i risultati dei test come certificazione della qualità del prodotto.

2.3 Aspettative Personali

Fin dalla pubblicazione dei progetti proposti dalle aziende ospitanti, il mio entusiasmo per l'evento *Stage-It* è cresciuto esponenzialmente. La varietà delle aziende coinvolte e la moltitudine di progetti presentati dalle stesse fanno dell'evento una grandissima opportunità offerta dal nostro corso di studi. Non capita spesso che ad un laureando venga data la possibilità di **scegliere** dove confluire all'interno del mercato del lavoro permettendogli di perseguire le proprie ambizioni professionali.
Avendo lavorato come sviluppatore prima e durante il percorso universitario, e avendo colto le varie sfaccettature del mondo dello sviluppo software grazie al corso di Ingegneria del Software, avevo ben chiare le prospettive che questo tirocinio avrebbe dovuto offrirmi e gli ambiti, invece, che avrei preferito evitare.
In primo luogo c'era la volontà di lavorare in un'azienda fortemente strutturata, con metodologie Agile ben consolidate e approcci moderni all'ingegnerizzazione del software, come l'Integrazione Continua e la filosofia DevOps.
In secondo luogo c'era l'idea di allontanarsi un po' dalla figura dello sviluppatore, a mio parere a volte un po' troppo schiava delle esigenze, legittime e non, del cliente. L'intenzione era quella di avvicinarsi alla figura del DevOps per automatizzare tutti i compiti ripetitivi e creare l'involucro che avrebbe ospitato l'applicazione vera e propria, garantendo qualità e rapidità.
Un'altra figura che mi sarebbe piaciuto affrontare era quella del Quality Assurance, ponendomi dalla parte opposta dello sviluppatore per scovare errori e migliorare la qualità del prodotto.
Dopo la partecipazione a *Stage-It* tre aziende hanno catturato maggiormente la mia attenzione:

* **Infocert**: il cui progetto è ben spiegato nelle sezioni precedenti;
* **Finantix**[1]: che proponeva una sensazionale filosofia DevOps ben rodata e integrata nella metodologia aziendale, offrendo la possibilità di ampliarne le procedure;
* **Thron**[2]: che offriva l'inserimento in un team di Quality Assurance per migliorarne i processi.

Sebbene tutte e tre le aziende mi avessero lasciato una grande impressione, l'unicità del progetto di Infocert (non sono tante le aziende che permettono di concentrarti sui test di carico) permetteva di esplorare sia l'approccio DevOps, tramite l'automazione dell'infrastruttura, e il mondo della Quality Assurance, tramite l'esecuzione dei test.

[1] **site:finantix**.
[2] **site:thron**.

Capitolo 3

Load Test Framework

In questo capitolo viene spiegato lo svolgimento del progetto assegnato: essendo quest'ultimo un caso di studio, non ha necessitato della stesura di analisi di requisiti o particolari documenti progettuali da riportare nella stesura della tesi.
Verranno descritte in ordine cronologico le scelte effettuate per portare a termine il compito: i primi due capitoli descrivono lo studio effettuato sullo stato d'arte delle soluzioni presenti sul mercato e le caratteristiche che questi avrebbero dovuto avere.
Il terzo capitolo spiega i compromessi adottati per adattare il software scelto all'infrastruttura aziendale.
Il quarto capitolo entra nel dettaglio della progettazione, narrando da un punto di vista più tecnico le scelte adoperate.
Gli ultimi tre capitoli sono invece dedicati all'utilizzo del software sviluppato e alla stesura della documentazione.

3.1 Analisi dei tool di test di carico

3.1.1 Obiettivi

In questa sezione viene analizzato il cuore dell'intero studio: lo strumento dei test di carico. In particolare, dopo diverse discussioni con il tutor aziendale sono emerse le caratteristiche principali che il software avrebbe dovuto dimostrare (in ordine di importanza):

1. **Vitalità**: un requisito fondamentale era che il software fosse vivo: ampliato e manutenuto costantemente.
   Potenzialmente i test di carico diventeranno un elemento cruciale della certificazione della qualità dei prodotti aziendali, la scoperta di bug molto gravi all'interno dello strumento di test di carico non può essere ignorata, un team di sviluppo attivo si opererebbe al più presto per correggere il difetto;

2. **Open Source**: un requisito che si collega al precedente: eventuali bug potrebbero essere risolti dai team di sviluppo aziendali se il software utilizzato fosse condiviso, inoltre eventuali personalizzazioni interne potrebbero migliorarne l'uso;

3. **Maturità**: anche questa caratteristica si collega alle precedenti: avviare progetti usando software giovani o non ancora consolidati potrebbe rivelarsi controproducente in caso di abbandono del progetto o della presenza di bug strutturali;

4. **Potenza**: l'applicativo Legalmail regge picchi di 500/600 mila visitatori giornalieri, lo strumento adottato dovrà poter simularne la quantità: performance adeguate e predisposizione ad essere eseguito in ambiente distribuito sono caratteristiche essenziali;

5. **Versatilità**: per eseguire dei validi test di carico è necessario simulare il comportamento dell'utente in fase di navigazione, inoltre Legalmail è composto da più servizi che non sono esposti solo tramite HTTP: la possibilità di testare anche protocolli diversi e la simulazione di complessi scenari di navigazione diventano necessari.

6. **Esaustività**: i report prodotti dal software devono essere validi: i risultati dei testi di carico possono avere valenza di certificazione per i clienti. Inoltre i report devono poter essere esportabili in modo da essere elaborati come un documento ufficiale;

7. **Versionamento**: gli scenari di test elaborati dal software dovrebbero poter essere scritti con linguaggi di scripting, preferibilmente già adottati in azienda, in modo da favorirne il riuso e permetterne il versionamento tramite Version Control System;

8. **Integrazione Continua**: caratteristica desiderabile: se il software è già predisposto per l'integrazione con strumenti di Continuous Integration sarà più facile introdurlo nel sistema aziendale;

9. **Prezzo**: più un requisito preferenziale: l'adozione del software non deve avere costi proibitivi.

3.1.2 Strumenti Visionati

Sebbene questi strumenti sono sempre più diffusi, non c'è ancora molta letteratura a riguardo si cui studiare sopra. Molti articoli inoltre sono scritti dagli stessi produttori del software, necessitando una certa cautela durante la lettura per evitare di "farsi vendere il vino dall'oste".
Tutti gli articoli visionati per l'analisi degli strumenti sono citati all'interno della bibliografia, nella sezione *Analisi degli strumenti di test di carico*.
Tutti gli strumenti visionati sono *Open Source* e, salvo diversamente indicato, sono manutenuti in modo frequente.

Apache JMeter

Apache JMeter è stato sviluppato da Apache e presenta la sua prima versione nel 1998. È considerato il leader degli strumenti di test di carico nel mercato open source: La sua longevità vanta una vastissima community attiva e il prodotto offre un esteso parco di funzionalità.
E' scritto in Java richiedendo una certa familiarità con la JVM e ha una curva di apprendimento piuttosto ripida: l'interfaccia risulta molto complessa e ci vuole del tempo per prendere familiarità con tutte le funzionalità offerte dal prodotto.
Utilizza un DSL di XML ed è sconsigliato scrivere i test a mano, rendendo molto difficile la modularità e il versionamento.
I test vengono eseguiti via CLI e sono molto performanti, il software riesce a generare molti utenti concorrenti e supporta l'esecuzione distribuita tramite architettura

master/slave sfruttando la tecnologia Java RMI e rendendo possibile la generazione di carichi molto elevati.
Permette di generare gli scnari di test registrando la navigazione utente sul browser ed esiste il plugin ufficiale per Jenkins.
Il report generato (in formato HTML) è molto completo ed è possibile esportare i dati grezzi in formato XML o CSV, permettendo di creare documenti in modo personalizzato.
È IL prodotto per l'enterprise: maturo, vivo, versatile e performante, oltre ad essere completamente gratuito. L'unica pecca si riscontra nella stesura dei test: l'interfaccia grafica può rendere macchinosa la fase di sviluppo, rendendo molto difficile il loro versionamento e/o la loro modifica al di fuori dell'ambiente integrato.

Gatling

Attivo da circa 5 anni, Gatling può essere considerato come una versione più moderna di Jmeter: entrambi girano sulla JVM (nonostante Gatling sfrutti Scala) e offrono prestazioni molto simili. Tuttavia Gatling converge più verso il mondo degli sviluppatori che a quello dei tester: l'espressivo e semplice DSL di Scala con cui si scrivono gli scenari di test ne permette il versionamento e la modularità, favorendone l'inserimento in un Version Control System.
Come JMeter, permette la generazione dei test registrando la navigazione del browser ma non offre lo stesso parco di protocolli del software di Apache, limitandosi a HTTP, JDBC e JMS.
Nasce come versione free ma permette l'upgrade alla versione enterprise che offre un'interfaccia grafica per controllare l'esecuzione dei test e migliora le metriche rilevate, oltre a facilitare l'integrazione con le piattaforme di Cloud. Il punto forte della versione enterprise è nell'esecuzione dei test su infrastruttura distribuita, che nella versione free è disponibile solo tramite script personalizzati.
Il report generato viene fornito in formato HTML permettendo di personalizzarne i grafico, i dati grezzi purtroppo sono salvati in un formato non documentato e l'azienda ne sconsiglia l'utilizzo per generare documenti personalizzati. Tuttavia esistono librerie che ne permettono la conversione in CSV in modo da facilitarne l'utilizzo.
Esiste il plugin ufficiale per Jenkins e tramite il DSL è possibile progettare dei criteri che marchino il test come riuscito o fallito, rendendolo ideale per i sistemi di Integrazione Continua.
Anche Gatling è un prodotto enterprise ready e compensa la minore completezza rispetto a JMeter con la modernità del suo utilizzo.

Locust

Locust è un progetto nato nel 2011, scritto da sviluppatori per sviluppatori: utilizza il linguaggio di programmazione Python e tramite l'ottima documentazione è possibile estenderne le funzionalità con del codice personalizzato.
Questa caratteristica si rivela però un arma a doppio taglio: Out of the box il software non offre molte funzionalità, supportando il solo protocollo HTTP e generando report piuttosto scarni.
Lo strumento permette due modalità di utilizzo: via CLI e via browser. La prima è ideale per i sistemi di Integrazione Continua mentre la seconda è ottima per monitorare lo svolgimento del test grazie a dei grafici che vengono generati in tempo reale.
I risultati finali possono essere esportati in formato CSV o essere raccolti direttamente dalla console di esecuzione in caso di utilizzo via CLI. Gli scenari di test vengono scritti

completamente in Python garantendone la modularità e il versionamento, oltre ad una discreta flessibilità. Purtroppo il prodotto non presenta funzionalità di browser recording, rendendo oneroso lo sviluppo di scenari complessi.
Il software consuma poca memoria durante la sua esecuzione ma per sfruttare al 100% la sua potenza è necessario superare una ripida curva d'apprendimento: pur supportando l'esecuzione dei test su infrastruttura distribuita, errate configurazioni possono fare da collo di bottiglia alle prestazioni, rendendo poco utile lo strumento. Per capire quest'ultima constatazione è sufficiente guardare i benchmark di Blazemeter e Loadimpact citati nella bibliografia: la differenza di prestazioni è abissale e nella sezione commenti di Loadimpact un utente spiega il perché. Non esistono plugin ufficiali per jenkins ma grazie all'architettura del prodotto non è difficile svilupparne uno tramite Python.
Sostanzialmente Locust è un prodotto focalizzato molto per gli sviluppatori e a causa di questo si porta con sé l'onere di sviluppare: con tempo e volontà è sicuramente possibile implementare il test di carico perfetto per la propria azienda, ma sono in poche a poter fare questo sacrificio. Allo stato attuale Locust è adeguato per testare sistemi poco complessi ma magari, con l'espansione della community, in un futuro sarà possibile utilizzarlo anche per applicazioni enterprise con più sicurezza e meno personalizzazioni.

The Grinder

The grinder è un tool di load testing scritto in Java nel 2000. Pare essere un tool completo e performante ma viola il primo requisito della ricerca: l'ultima release ufficiale risale al 2012 mentre l'ultimo commit è del 2015 (la repository si trova su Sourceforge). Evenutalmente esiste un repository anche su Github che è stato aggiornato di recente ma, osservando le statistiche, è poco diffuso a non ha molti manutentori.

K6

K6 è un tool di recentissima fattura pubblicato da Loadimpact per la prima volta nel 2017, autodefinendosi "come dovrebbe essere un tool di load testing nel 2017".
Gli scenari di test sono scritti in Javascript ed è possibile utilizzare tutte le librerie offerte dal linguaggio, oltre a quella specifica del software K6, rendendo il migliore per il lato modularità e versionamento. Il prodotto è rivolto principalmente agli sviluppatori e l'esecuzione dei test via CLI unito all'espressività di Javascript lo rende facilmente integrabile in ambienti di Integrazione Continua come Jenkins.
Il report vengono generati direttamente sulla CLI ma è minimale e poco esaustivo, questo è voluto in quanto il prodotto sembra essere parte di una strategia che invogli l'utente alla sottoscrizione di abbonamenti con Loadimpact che offre un più completo sistema di analisi (è presente infatti una modalità esecuzione via CLI direttamente sui server di Loadimpact).
Nonostante ciò è possibile esportare i dati in formato JSON che contiene informazioni molto più dettagliate rispetto al report standard, inoltre è presente in modo nativo l'integrazione con diversi software di reportistica, come InfluxDB e Grafana, permettendo di generare report molto più completi.
Le performance su un singolo computer sono simili a quelle di Gatling ma al momento non è possibile eseguire i test in modalità distribuita (al netto di orchestrazioni personalizzate), tuttavia la funzionalità è prevista nella roadmap per fine anno (anche se non ci sono certezze da questo punto di vista).
Piccola nota: molta della letteratura trovata sul tema è fornita da Loadimpact, quindi

le valutazioni su questo strumento potrebbero essere un po' di parte. Tuttavia anche altre fonti sembrano suggerire la validità di questo software.

Taurus

Taurus è un software prodotto da Blazemeter nel 2015 ed è da considerarsi più come un'orchestratore di strumenti di test di carico che un load testing tool stesso. È stato pensato per risolvere i problemi di automazione e/o Integrazione Continua di molti dei tool di test di carico presenti sul mercato offrendo la libertà di scelta su quale di questi usare.
Taurus è infatti capace di eseguire file di JMeter, Gatling, Tsung, The Grinder e altri tool di test di carico fino ad arrivare alla simulazione effettiva di un browser tramite Selenium (oltre a supportare tool di test funzionale come Robot Framework), lasciando come unico onere la configurazione tipo di carico da generare tramite comodi file YAML.
Non è presente letteratura sulle sue performance, tuttavia è facile immaginare come queste non siano prevedibili: dipendono dallo strumento utilizzato a cui vanno aggiunti i costi di orchestrazione.
Come funzionalità non ha eguali ma essendo sviluppato dal team di Blazemeter per il loro servizio di Cloud non può offrire tutte queste funzionalità gratuitamente: la reportistica infatti è minimale, nonostante quella in tempo reale della console sia molto valida, e per ottenere risultati persistenti è necessario abbonarsi ai servizi di Blazemeter, che nella versione free permettono di salvare i report, visualizzandoli tramite il servizio, per un massimo 7 giorni. Eventualmente Taurus produce una cartella con tutti i log generati durante l'esecuzione, questi però non sono a formato fisso in quanto dipendono dai tool utilizzati per il testing (JMeter, Gatling, Selenium, etc.). Volendo è quindi possibile generare reportistica esaustiva e persistete ma necessita di orchestrazioni personalizzate.
L'esecuzione su infrastruttura distribuita dipende dal tool che si usa tramite Taurus oppure è garantita con tutti i tipi di test utilizzando i servizi di Blazemeter a pagamento.
Taurus si presenta come un software molto competente e valido, purtroppo le limitazioni della versione free non ne permettono l'utilizzo in ambiente enterprise e i costi del servizio a pagamento non sono trascurabili.

Vegeta

Vegeta è un tool di test di carico presentato nel 2014 da un singolo sviluppatore, è un tool eseguito via CLI che nonostante sia semplice da apprendere risulta un po' intricata come usabilità.
Le performance sono buone, leggermente superiori a quelle di gatling, ed è possibile creare tanti carichi tramite infrastruttura distribuita anche se la funzionalità non è inclusa nativamente nel software ma necessita di orchestrazioni esterne per essere funzionante.
Vegeta offre anche la possibilità di scrivere ed eseguire i test tramite una libreria scritta in GO, permettendone la modularità e il versionamento.
Tuttavia la semplicità d'utilizzo di Vegeta giunge ad un compromesso con le poche funzionalità messe a disposizione: non è possibile infatti descrivere scenari complessi: la definizione degli stessi si limita a selezionare in round-robin gli url indicati senza poter simulare un vero e proprio comportamento dell'utente.
Inoltre la configurazione di vegeta permette poco controllo sulla distribuzione del carico: il tool infatti permette unicamente di indicare la quantità di richieste per

secondo desiderate (caratteristica che in pochi altri tool riescono ad offrire) lasciando libertà al tool di gestire le risorse della macchina per raggiungere il ratio indicato. Questa caratteristica non sopperisce però alla mancanza di ramp-up (anche se possibile realizzarla via bash scripting) e alla possibilità di controllare attivamente le risorse del sistema (come il numero massimo di thread).
In definitiva, Vegeta è un buon per testare singoli endpoint in modo controllato e imbastire sistemi di CI più complessi grazie alla versatilità della libreria in Go e la facilità d'integrazione con tool di terze parti, tuttavia per scenari più complessi e configurazioni più avanzate guardare altrove.

Wrk

Wrk è un tool amatoriale pubblicato per la prima volta nel 2012. Onestamente non c'è troppo da dire su Wrk, partendo dalla documentazione ufficiale praticamente inesistente, le funzionalità offerte da Wrk non offrono tanto spazio di manovra. E' il tool più potente per quanto riguarda il throughput grezzo su una singola macchina ed è l'unico strumento in grado di utilizzare al 100% le risorse della CPU in condizioni ottimali: target server immediato nella risposta e basso peso, in byte, della richiesta invat. Con scenari di test più complessi inoltre il tool perde accuratezza nelle misurazioni dei tempi di riposta, diventando poco utile.
Non offre la possibilità di distribuire l'esecuzione dei test su più macchine e in rete non si trovano argomenti per farlo a mano, suggerendo che questo non è il suo caso d'uso.
La CLI è comoda e permette di integrare scenari più complessi utilizzando il linguaggio di scripting Lua che permette anche di convertire i risultati nel formato di output desiderato. I report sono poco esaustivi, Out of the box è offerta solo la visualizzazione in CLI ma le metriche raccolte sono generiche e non permettono un'analisi dettagliata.
Non esiste letteratura per quanto riguarda eventuali integrazioni in Integrazione Continua, suggerendo anche qui che per l'automazione non è il miglior tool, a meno di personalizzazioni in Lua.

Apachebench

Come per Wrk, anche Apache Bench (ab) non ha molto di raccontare, sviluppato per testare le performance del webserver apache, svolge un ottimo lavoro nel colpire un singolo URL e riportarne i risultati.
Data la sua maturità e ottimizzazione è in grado di generare un grosso carico sfruttando unicamente un singolo core della CPU ma oltre non va: non supporta l'esecuzione distribuita e non sfrutta CPU multicore.
Genera un carico dimezzato rispetto a Wrk ma offre un report ed un'accuratezza migliore, nonostante non sia possibile esportare il report in formati universali di default, richiedendo un'elaborazione ad hoc in caso si voglia esportare i dati dalla CLI.
L'esecuzione avviene via CLI e permette di configurare comodamente diversi parametri.
Ottimo per test semplici e feedback immediati ma non adeguato per integrazione in Integrazione Continua e stesura di scenari complessi.

Tsung

Nato nel 2001 come successore di Tsunami IDX,Tsung è uno strumento di test di carico multiprotocollo scritto in Erlang, robusto e maturo: tra le sue funzionalità prevede la registrazione del browser e il supporto Out of the box per l'esecuzione dei test in modalità distribuita.

Gli scenari di test vengono scritti in XML ma a differenza di JMeter non offre un'interfaccia grafica per aiutare la scrittura degli scenari, nonostante il DSL di Tsung sia molto più chiaro e facile da apprendere rispetto a quello di JMeter. Eventualmente possono essere scritti scenari più complessi con l'ausilio di Erlang.
Il report generato è molto esaustivo ed oltre ai risultati statistici finali offre la possibilità di monitorare le statistiche d'esecuzione in tempo reale tramite un'intuitiva interfaccia web. I dati finali possono essere esportati in JSON e la loro struttura è ben documentata in modo da poter esser facilmente consumati da altre piattaforme di reportistica.
Tsung è sicuramente un prodotto pronto per l'enterprise ma è accompagnato da due maggiori inconvenienti: l'usabilità e la predisposizione all'integrazione continua.
Gli sviluppatori infatti sono avvezzi a "programmare" in XML che, pur presentando un DSL chiaro ed intuitivo, non offre la stessa espressività e flessibilità dei linguaggi di scripting. Inoltre, pur essendo disposto al versionamento, XML è più difficile da modularizzare rispetto ad un normale linguaggio di programmazione.
Per quanto riguarda l'Integrazione Continua, Tsung non presenta plugin ufficiali e la community non offre soluzioni a riguardo, rendendo eventuali integrazioni completamente a carico dell'utente.
In definitiva, Tsung è un ottimo prodotto: robusto, maturo e versatile, avente l'unica pecca di presentare un design non al passo con le metodologie di sviluppo software moderne .

Artillery

Artillery è un tool di test di carico nato da sviluppatori per sviluppatori, è sviluppato in Javascript ed è orientato verso le infrastrutture di Integrazione Continua proponendo un sistema per bollare il test come riuscito o fallito al termine della sua esecuzione.
È possibile configurare il carico e gli scenari di test tramite YAML (o JSON) che offrono la possibilità di integrare codice Javascript. E' estensibile tramite plugin (la community ne offre già diversi) e offre la possibilità di testare non solo HTTP ma anche WebSocket e Socket.io, suggerendo una propensione per l'ambiente Javascript da parte degli sviluppatori.
I report prodotti sono relativi all'analisi statistica finale e vengono forniti in formato HTML ma è possibile esportare i report in CSV o integrarsi database tramite integrazioni via plugin.
Le prestazioni non sono ottimali: il tool non riesce a generare un grande carico e non è prevista una modalità di esecuzione distribuita se non acquistando la versione Enterprise che grazie all'integrazione con AWS riesce a distribuire l'esecuzione su più macchine migliorandone le performance.
Inoltre il software non riesce a sfruttare le CPU multicore anche se questa funzionalità è prevista nella rilascio della seconda versione del prodotto.

Siege

Siege è un software di test di carico scritto prevalentemente in C che si concentra sul protocollo HTTP.
Dopo diverse ricerche ho notato come Siege non offra grosse prestazioni con test complessi e che non riesca a simulare grossi carichi senza perdere in accuratezza di misurazione.
Considerando l'obiettivo di **Maturità** della ricerca, la poca accuratezza nella mis-

urazione, valutata come bug strutturale, ha portato all'abbandono dello studio di questo prodotto.

Bees with machine guns

Una menzione d'onore è assegnata a Bees with machine guns, pur non essendo aggiornato da più di un anno, questo tool permette di generare grossi carichi in maniera distribuita tramite poche righe di CLI.
Sostanzialmente richiede le credenziali di un account AWS e, tramite una piccola configurazione, imbastisce le macchine EC2 richieste (le crea e installa il software necessario), le utilizza per generare il carico richiesto e poi le spegne, tutto in maniera automatica (è possibile spezzare il processo in 3 step, l'orchestrazione, l'esecuzione e lo spegnimento).
Non è un software molto sofisticato, ma per test senza grosse pretese o per l'esecuzione di stress test può risultare una scelta molto efficace, grazie alla facilità con cui si può generare un grosso traffico d'utenti.

3.1.3 Valutazioni intermedie

Gli strumenti descritti in precedenza, sono solo una parte di quelli presenti sul mercato, tuttavia dati i limiti di tempo non era possibile analizzarli tutti.
Di quelli analizzati, tuttavia, i candidati più vicini alle caratteristiche richieste sono risultati essere: **JMeter**, **Tsung**, **Gatling** e **K6**: i primi due per la maturità, versatilità e robustezza dimostrati negli anni; i secondi, di più recente fattura, per il loro design più moderno e per maggiore vicinanza alle pratiche DevOps. La soluzione ideale sarebbe rappresentata dalla fusione delle caratteristiche principali degli strumenti citati, idea che trova modo di esprimersi attraverso **Taurus**, che, esponendo JMeter tramite un interfaccia molto semplice avvicinandolo alle pratiche DevOps, ingloba tutte le caratteristiche ricercate da questo studio. Tuttavia, l'elevato prezzo di sottoscrizione ne rende proibitiva l'adozione.
Wrk, **Apachebench** e **Bees with machine guns** potrebbero essere integrati per realizzare gli stress test, considerata la facilita con la quale permettono di generare un grosso carico, tuttavia, una buona conoscenza dei tool citati in precedenza può portare agli stessi risultati evitando l'apprendimento di altri strumenti.
The Grinder è un prodotto solido, ma la scarsa vitalità lo rende inadatto agli scopi aziendali. **Locust** è un altro prodotto degno di nota ma le opinioni contrastanti trovate sul suo conto non hanno convinto i referenti aziendali ad approfondirne le caratteristiche.
In ultimo troviamo **Artillery**, **Vegeta** e **Siege** che, sebbene con caratteristiche differenti, sono accomunati dalla scarsa maturità, rendendoli inadeguati alle aspettative aziendali.

3.1.4 Prova dei possibili candidati

I quattro candidati più promettenti sono poi stati sottoposti ad un ulteriore step di valutazione: installazione e lancio di un test in modalità distribuita. Questa fase ha permesso di capire meglio i pro e i contro (dal punto di vista aziendale) dei vari strumenti, suddividendo le caratteristiche analizzate in 4 categorie:

* **Configurazione**: la modalità di stesura degli scenari di test e la loro successiva manutenzione;
* **Load Generation**[1]: la modalità in cui viene generato il carico: aperta (open), quando si fissa un numero di richieste per secondo (utenti) e il software le genera indipendentemente dal termine della richiesta; chiusa (closed) quando il software simula un numero fissato di utenti che effettua periodicamente nuove richieste unicamente al termine delle richieste precedenti;
* **Cluster Mode**: la facilità dell'impostazione della modalità di esecuzione distribuita;
* **Documentazione**: la bontà della documentazione trovata, inclusa l'attività della community, per scrivere ed eseguire il test.

Lo scenario di test rappresentato simula una semplice navigazione di quattro pagine di un sito web.

[1] *Open-Closed Model Load Generators, Adam Wierman.* URL: http://users.cms.caltech.edu/~adamw/publications-openclosed.html.

Strumento		Configurazione	Load Generation	Cluster Mode	Documentazio
JMeter	PRO	-Recording del browser semplice	-Open/Closed -Facile da configurare	-Out of the box	-Community Att -Estesa docum tazione -Abbondanza tutorial
	CON	-Interfaccia grafica a primo impatto complessa -XML oneroso da modificare a mano		-Eventuali file di input devono essere caricati sulle macchine del cluster a mano	
Tsung	PRO	-DSL Espressivo -Recording del browser semplice	-Facile da configurare	-Out of the box	-Documentazion chiara e complet
	CON	-Scenari particolari richiedono l'uso di Erlang -XML oneroso da modificare a mano	-Solo Open	-Definzione del cluster da XML e non da parametri	-Community molto attiva
Gatling	PRO	-DSL Facile ed espressivo -Recording del browser semplice -Facile da modularizzare	-Open/Closed -Facile da configurare -Molto personalizzabile		-Buona docum tazione -Community Atti -Molti tutorial
	CON	-Scenari particolari potrebbero richiedere la conoscenza di Scala		-Non Out of the box: necessita di script ad hoc per funzonare	
K6	PRO	-Supporta tutta l'interprete Javascript -Recording del browser semplice -Facile da modularizzare	-Facile da configurare -Permette la configurazione tramite files JSON		-Documentazion chiara e complet
	CON	-Scenari lunghi possono generare file scarsamente leggibili se non strutturati con criterio	-Solo Open	-Non Out of the box: necessita di script ad hoc per funzionare	-Community anc di nicchia

Tabella 3.1: Pro e Contro strumenti installati

3.1.5 Valutazioni Finali

La prova dei candidati ha mostrato due fazioni ben distinte: da un lato K6 e Gatling: tool moderni e funzionali con il design progettato per soddisfare le esigenze degli sviluppatori; dall'altra JMeter e Tsung: strumenti più robusti e versatili ma con un'usabilità più vicina al mondo del Quality Assurance.
Tuttavia, la mancanza di un sistema d'esecuzione distribuita Out of the box dei primi due si è rivelata una pecca piuttosto grave, ricordando come la *potenza*, intesa come capacità di carico generato, sia un'elemento chiave di questa ricerca e una caratteristica necessaria per gli strumenti in esame.
La scelta quindi si è spostata sul secondo blocco di strumenti e, pur constatando l'ottima fattura di Tsung, la familiarità con l'ambiente Java da parte dell'azienda e l'ottima reputazione del software di Apache hanno decretato **JMeter** come lo strumento ideale per l'esecuzione dei test di carico nell'ambiente Enterprise di Infocert S.p.A.

3.2 Analisi delle soluzioni cloud

Dopo aver selezionato lo strumento di test di carico adatto per la realizzazione dei test è stato svolto uno studio per capire come rendere semplice ed immediata l'istanziazione dell'infrastruttura per eseguirlo in modalità distribuita. In pratica servivano strumenti e procedure per automatizzare l'approvvigionamento dei server sulla quale eseguire i test di carico.

3.2.1 Obiettivi

Gli obiettivi dello studio possono riassumersi nel trovare una procedura di approvvigionamento dell'infrastruttura che fosse:

* **Funzionale**: pronta ad eseguire i test di carico tramite JMeter;
* **Automatica**: la configurazione dei server e l'esecuzione dei test doveva ridurre al minimo le iterazioni dell'utente;
* **Flessibile**: la modifica della configurazione doveva avvenire tramite parametri;
* **Economica**: i costi finanziari di approvvigionamento dovevano essere ridotti al minimo;
* **Adattabile**: predisposta ad essere, eventualmente, integrata con i tool aziendali come Jenkins e Puppet.

3.2.2 Infrastructure as Code

Prima di vagliare le soluzioni presenti sul mercato è stato svolto uno studio teorico sul mondo del Cloud, per permetterne una migliore comprensione dei concetti e della terminologia che lo definiscono.
Questo studio, adattato agli obiettivi sopraelencati, è confluito nell'Infrastructure as Code (IaC)[2]: un'approccio che sta alla base della filosofia DevOps e che consiste nel gestire l'infrastruttura aziendale come se fosse un prodotto software: definita tramite codice versionato, testato e aderente a tutte le altre pratiche comuni allo sviluppo

[2] *What is infrastructure as code?, Margaret Rouse.* URL: https://searchitoperations.techtarget.com/definition/Infrastructure-as-Code-IAC.

software.
L'IaC, nel contesto della gestione di server atti a ospitare applicativi, si compone di quattro grandi macro fasi/componenti così definite[3]:

1. **Provisioning**: atta a creare ed avviare le macchine fisiche e/o virtuali, conferendogli le risorse necessarie (CPU, RAM, Hard-disk, etc.) e installando il software per il configuration management;

2. **Configuration Management**: atta gestire e mantenere le configurazioni necessarie per ospitare l'applicativo: software, credenziali, impostazioni di sistema, variabili d'ambiente etc.;

3. **Deployment**: atta a distribuire l'applicativo sulle macchine istanziate, gestendone le versioni e i rollback: ripristino della versione precedente in caso di errori;

4. **Orchestration**: atta a orchestrare tutte le varie fasi elencate in precedenza, definendo ordine, modalità d'esecuzione e tutte le configurazioni necessarie in modo da rendere più agile la gestione di infrastrutture molto grandi.

Va specificato che queste fasi non sono fisse ma possono mischiarsi tra loro: ad esempio nella fase di provisioning possono essere impostate variabili d'ambiente o la fase di configuration management può occuparsi dell'aggiornamento di versione dell'applicativo. Questo accade perchè molti strumenti presenti sul mercato non si limitano a gestire una sola fase, ma ne inglobano diverse (come Ansible[4] ad esempio), centralizzando la gestione dell'infrastruttura.
Queste componenti dovevano poi rientrare nel contesto dei *disposable servers*: server usa e getta creati per svolgere un determinato compito ed essere distrutti immediatamente dopo. L'esecuzione dei test di carico infatti non necessita di un servizio attivo perennemente, quindi l'istanziazione di server dedicati allo scopo e poi cancellati al termine avrebbe permesso un abbattimento dei costi.
L'ultimo componente della ricerca era quindi una piattaforma di Cloud che permettesse la creazione di questi disposable servers.

3.2.3 Cloud Platform

Riprendendo il discorso dei disposable servers, la piattaforma ricercata doveva prevedere un sistema di sottoscrizione aderente al modello *pay as you go (PAYG)*[5]: il calcolo della fattura in base all'effettivo tempo nel quale le macchine sarebbero state operative. Per evitare di imbattersi in servizi non pienamente affidabili/maturi è stato scelto di analizzare esclusivamente le tre grandi piattaforme di riferimento per quanto riguarda il Cloud Computing: **Amazon Web Services (AWS)**, **Google Cloud Platform (GCP)** e **Microsoft Azure (MA)**; tutte aderenti al modello PAYG. In particolare sono state vagliate le proposte AWS EC2[6], GCP Compute Engine[7] e MA Virtual Machines[8].

[3] *Deployment vs Provisioning vs Orchestration vs Configuration Management, Peter Souter.* URL: https://archive.fosdem.org/2018/schedule/event/deployment_provisioning_orchestration/.
[4] **site:ansible**.
[5] *What is pay-as-you-go cloud computing?, Margaret Rouse.* URL: https://searchstorage.techtarget.com/definition/pay-as-you-go-cloud-computing-PAYG-cloud-computing.
[6] *Amazon Web Services Elastic Compute.* URL: https://aws.amazon.com/it/ec2/.
[7] *Google Cloud Platform Computer Engines.* URL: https://cloud.google.com/compute/.
[8] *Microsoft Azure Virtual Machines.* URL: https://azure.microsoft.com/it-it/pricing/details/virtual-machines/linux/.

Tutte e tre le soluzioni sono molto simili, sia come prezzo che come funzionalità: oltre che al modello PAYG, tutte e 3 propongono configurazioni di istanze dedicate al calcolo, con una maggiore dotazione di CPU rispetto alla RAM. Quest'ultima classificazione è importante: l'esecuzione dei test di carico tramite JMeter tende a saturare prima la CPU che la memoria.
AWS[9] e GCP[10] offrivano in più particolari tipi di istanze chiamate prerilasciabili: queste hanno costi nettamente inferiori rispetto a quelle standard, ma possono essere fermate dal provider e assegnate ad altri clienti in caso di operazioni con priorità più alta. Questo modello non è adatto per l'esecuzione dei test di carico: questi ultimi sono entità con inizio e fine senza la possibilità di essere stoppati senza dover ricominciare da capo tutta la procedura.
Un'altra categorizzazione importante è quella che fa AWS sulle istanze a prestazioni fisse ed espandibili[11], le prime possono utilizzare il 100% della CPU in qualsiasi momento, mentre le seconde, più economiche, possiedono dei crediti CPU che possono accumulare quando la macchina ne sta usando meno. Nel caso d'uso dei test di carico, le seconde non sono utilizzabili: JMeter fa un utilizzo intensivo della CPU fin dall'inizio del test.
Considerata la mia personale esperienza pregressa e la migrazione di alcuni servizi aziendali verso questa piattaforma, AWS è stata etichettata come la soluzione più conforme. In particolare è stata concordata la scelta delle istanze EC2 tipo C5: ottimizzate per il calcolo e a prestazione fissa.

3.2.4 Provisioner

Per lo studio degli strumenti di provisioning orientati all'IaC sono state analizzate in primis le soluzioni proposte direttamente dai Cloud Provider: **AWS Cloud Formation**, **GCP Deployment Manager** e **MA Automation**. L'ultima è stata immediatamente accantonata in quanto valutata un po' acerba e complessa da utilizzare.
Successivamente sono state analizzate due soluzioni a se stanti: **Terraform** e **Ansible**. Il primo è uno strumento sviluppato da HashiCorp che sta emergendo sempre più rapidamente grazie alla sua facilità d'utilizzo e le innumerevoli integrazioni con servizi esterni, il secondo invece è un tool ben maturo che, presentato inizialmente come Configuration Manager, si è trasformato negli anni come strumento di gestione dell'IaC a tutto tondo.
Questi strumenti sono stati valutati su quattro diversi aspetti:

* **Piattaforme Supportate**: le piattaforme di cloud sul quale può lavorare lo strumento;
* **Configurazione**: la modalità di configurazione: dichiarativa, dove l'utente specifica *cosa* gli serve e il software si occupa di *come* implementare; imperativa dove l'utente è responsabile del *cosa* e del *come*;
* **Modifiche**: la gestione delle modifiche all'infrastruttura: indipendente, dove il software capisce se creare o eliminare le macchine in base ai parametri; manuale dove l'utente si occupa di questo aspetto;
* **Esecuzione**: la modalità di esecuzione del software: CLI, ideale per l'automazione; GUI, ideale per l'utilizzo da parte dell'utente.

[9] *AWS EC2 Spot instances, Amazon.* URL: https://aws.amazon.com/ec2/spot/pricing.
[10] *GCP Preemptible instances, Google.* URL: https://cloud.google.com/compute/docs/instances/preemptible.
[11] *AWS CPU Credits, Amazon.* URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-credits-baseline-concepts.html.

Strumento	Piattaforme Supportate	Configurazione	Modifiche	Esecuzione
AWS Cloud Formation	AWS	Dichiarativa	Automatiche	GUI/CLI
GCP Deployment Manager	GCP	Dichiarativa	Manuali	CLI
Terraform	AWS/GCP/MA e molti altri	Dichiarativa	Automatiche	CLI
Ansible	AWS/GCP/MA e molti altri	Dichiarativa/ Imperativa	Manuali	GUI/CLI

Tabella 3.2: Tabella comparativa strumenti di provisioning

L'analisi ha decretato **Terraform** come soluzione più adeguata per i seguenti motivi:

1. **Semplicità**: questo software è molto facile da utilizzare, con poche righe del DSL (chiaro ed espressivo) è possibile approvvigionare una flotta di server di dimensione elevate. Inoltre la modalità di configurazione dichiarativa e la gestione automatica delle modifiche permette all'utente di non arrovellarsi troppo sul come implementare l'infrastruttura. In più la distruzione delle macchine avviene tramite un semplice comando da CLI, rendendo facile l'implementazione dei disposable servers;

2. **Flessibilità**: Terraform non è legato ad una sola piattaforma di Cloud ma stimola i provider[12] a sviluppare il proprio plugin per integrarsi con il programma. Questo permette di avere flessibilità nell'infrastruttura e nel futuro potranno essere scritte soluzioni ad hoc per l'infrastruttura aziendale interna;

3. **Documentazione**: la documentazione è chiara e completa e viene aggiornata molto spesso. La community inoltre è molto attiva ed è molto facile ricavare soluzioni;

4. **Coerenza**: Terraform si concentra su un solo aspetto dell'IaC e lo fa molto bene: il provisioning. Questo permette di relegare al software questo unico aspetto, rendendo più strutturato ed immediato il codice che verrà eseguito;

5. **Verificabilità**: Terraform è predisposto all'analisi dinamica delle sue configurazioni[13].

[12] *Writing Custom Providers, Terraform.* URL: https://www.terraform.io/docs/extend/writing-custom-providers.html.

[13] *Top 3 Terraform Testing Strategies for Ultra-Reliable Infrastructure-as-Code, Carlos Nunez.* URL: https://www.contino.io/insights/top-3-terraform-testing-strategies-for-ultra-reliable-infrastructure-as-code.

3.2.5 Configuration Management

La gestione della configurazione all'interno dell'azienda è effettuata tramite Puppet, rendendo non necessaria la ricerca di un altro strumento.

3.2.6 Deployment

All'interno dell'azienda la fase di deploy viene gestita direttamente da Puppet.

3.2.7 Orchestrator

Per la scelta dell'orchestratore sono stati accantonati preventivamente prodotti come Kubernetes[14], Nomad[15] e simili, questi infatti, oltre a richiedere l'utilizzo di Container System come Docker[16] o Packer[17], sono orientati per architetture a microservizi, dove più componenti differenti comunicano tra di loro. L'esecuzione dei test di carico avviene invece tramite un solo servizio, JMeter, che viene replicato su più server.
La scelta quindi si è spostata verso una soluzione ad hoc, sviluppata internamente, integrabile con diversi approcci:

* Scrittura di uno script bash parametrizzabile;
* Sviluppo di un plugin per Jenkins;
* Stesura di una guida su Confluence per eseguire le varie fasi in modo manuale, violando l'obiettivo di ridurre al minimo le interazioni dell'utente;
* Progettazione di una utility in Python focalizzata sull'estensibilità, in modo da poter adattare l'esecuzione dei test di carico in caso di necessità future.

3.2.8 Soluzione proposta

Al termine dell'analisi la soluzione proposta è stata la progettazione di un orchestratore in Python, che sfruttasse Terraform e Puppet per l'approvvigionamento e configurazione dell'infrastruttura (Cluster), eseguisse i test di carico, consumasse i dati in una maniera concordata e infine sfruttasse Terrafom per distruggere l'infrastruttura appena creata. Eventualmente, questo orchestratore in Python potrebbe essere integrato con Jenkins.

[14] *Kubernetes*. URL: https://kubernetes.io/.
[15] *Nomad*. URL: https://www.nomadproject.io/.
[16] *Docker*. URL: https://www.docker.com/.
[17] *Packer*. URL: https://www.packer.io/.

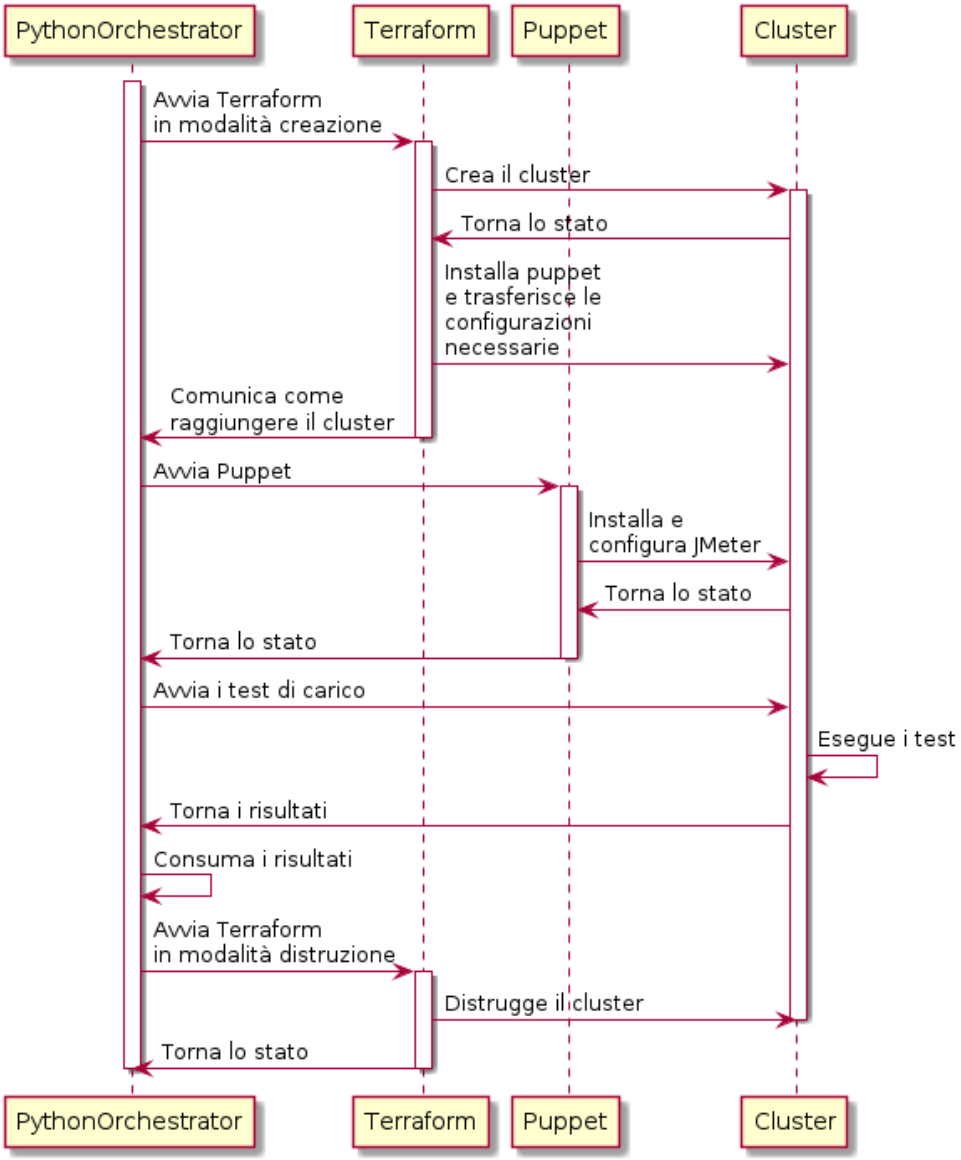


Figura 3.1: Visione ad alto livello della soluzione proposta

3.3 Lo scontro con l'organizzazione aziendale

3.3.1 Provisioning nell'intranet

La soluzione proposta discussa nella sezione precedente, seppur condivisa dal tutor aziendale, ha dovuto subire degli accorgimenti per aderire agli standard aziendali. Come spiegato nella sezione Infrastruttura e Applicazioni, il provisioning delle macchine virtuali all'interno dell'Intranet deve rispettare gli standard definiti dal team *Infrastructure*, che attualmente prevede l'instanziazione dei server solo tramite Foreman. Inoltre le policy aziendali, allo status attuale, non permettono l'integrazione con le API

di Foreman, rendendo troppo oneroso lo sviluppo di un automatismo di provisioning. L'integrazione con i servizi di AWS inoltre, seppure già adottati in piccola parte per alcuni servizi aziendali, richiede la collaborazione sia con il team *Infrastructure* che con il team *Network*, responsabile della gestione dell'instrastruttura di rete.

3.3.2 Piano di lavoro

A seguito di queste considerazioni è stato approvato il progetto dell'orchestratore in python, denominato **JMeterOrchestrator**, che, oltre alla gestione delle meccaniche di JMeter, avrebbe dovuto permettere due modalità d'uso:

1. **Provisioning Interno (Obbligatorio)**: utilizzando l'infrastruttura aziendale, ovvero creando a priori le istanze tramite Foreman e utilizzandole tramite l'orchestratore;

2. **Provisioning Esterno (Opzionale)**: sfruttando terraform per creare i disposable servers su AWS direttamente dall'orchestratore.

Il consumo dei risultati, inoltre, si sarebbe tradotto con l'invio di una E-Mail, contenente i dati prodotti, al personale del team Legalmail.
L'installazione di JMeterOrchestrator sarebbe comunque dovuta avvenire all'interno della rete aziendale.

3.4 Progettazione dell'orchestratore

3.4.1 JMeter - Modalità Distribuita

Prima di descrivere la progettazione di JMeterOrchestrator, vorrei inizare questa sezione con una panoramica del funzionamento di JMeter in modalità distribuita (cluster mode), in modo da chiarire quali azioni fossero direttamente gestite da JMeter e specificare quali procedure invece sarebbero state prese in carico dall'orchestratore.
JMeter quando viene eseguito in cluster mode si divide in due entità:

* **Master**: la macchina, fisica o virtuale, responsabile di coordinare le macchine slave e collezionare i dati da queste prodotti.

* **Slaves**: le macchine, fisiche o virtuali, responsabili di generare le richieste al server di destinazione, chiamato target.

L'esecuzione di JMeter in modalità distribuita avviene con l'immissione, tramite CLI sulla macchina master, di un comando simile:

```
jmeter -n -t file.jmx -R 127.0.0.1 -l risultati.csv -e -o report
```

Dove i parametri possiedono il seguente significato:

* **-n**: non aprire la GUI, in modo da risparmiare RAM;

* **-t**: indica il file da utilizzare come test;

* **-R**: specifica gli indirizzi ip degli slaves da usare per l'esecuzione del test;

* **-l**: indica dove salvare i risultati;

* **-e -o**: indica dove generare il report HTML.

Il master assume quindi che gli slaves esistano e il server RMI, responsabile di ricevere le indicazione dal master, sia attivo e in stato di ricezione. Il master si assume poi la responsabilità di distribuire il file di test agli slaves, ma, eventuali file di dati (utili per specificare le credenziali degli utenti da simulare o altre configurazioni) devono essere caricati sulle macchine slaves a mano.
I file di test, suffissi *jmx*, non solo racchiudono le richieste da effettuare sulla macchina target ma specificano anche la configurazione del carico da applicare, richiedendo una modifica del file di test in caso si voglia aggiustare qualche parametro della Load Generation.
In ultimo i file generati da JMeter vengono semplicemente scritti su disco ma non consumanti in qualsivoglia modo, questa caratteristica non è adatta per la realizzazione dei disposable servers, questi infatti vengono eliminati alla fine dei test, cancellando a loro volta i dati prodotti.
In sostanza quindi JMeter gestisce solo il coordinamento delle macchine slave, mentre sarà compito dell'orchestratore:

* Assicurarsi che le macchine slave esistano (creandole eventualmente) e che abbiano il server RMI attivo;
* Distribuire eventuali file di dati alle macchine slaves;
* Prevedere una modalità di modifica della configurazione di carico senza richiedere la riscrittura del file *jmx*;
* Notificare l'avvenuto termine dei test, consegnando i file di risultati prodotti al team Legalmail via E-Mail.

3.4.2 Principi

Per la progettazione e successiva implementazione dell'orchestratore sono stati seguiti dei principi, o best practises, per garantire una solida codebase e una non troppo onerosa manutenzione.

SOLID

L'acrostico SOLID, descritto da Robert C. "Uncle Bob" Martin[18] all'inizio degli anni 2000, si riferisce a cinque principi fondamentali dello sviluppo software orientato agli oggetti e sono:

1. **Single Responsability Principle (SRP)**: afferma che ogni classe dovrebbe avere una ed una sola responsabilità, interamente incapsulata al suo interno. Questa caratteristica troverà riscontro tramite la realizzazione dei **components**, come verrà specificato nella sezione successiva;
2. **Open/Closed Principle (OCP)**: afferma che un'entità software dovrebbe essere aperta alle estensioni, ma chiusa alle modifiche. Su questo principio è stata riposta molta attenzione in quanto, come spiegato nella sezione precedente, le possibili evoluzioni del prodotto sarebbero dovute risultare poco onerose da implementare;

[18] Robert C. "Uncle Bob" Martin. *Design Principles and Design Patterns.* objectmentor.com, 2000.

3. **Liskov Substitution Principle (LSP)**: afferma che gli oggetti dovrebbero poter essere sostituiti con dei loro sottotipi, senza alterare il comportamento del programma che li utilizza. Questa caratteristica è implementata di default in python, in quanto linguaggio aderente al Duck Typing;

4. **Interface Segregation Principle (ISP)**: afferma che sarebbero preferibili più interfacce specifiche, che una singola generica. Questa principio viene rispettato con la definizione di più interfacce **actions** seppur strutturalmente simili, come verrà spiegato nelle sezioni successive;

5. **Depedency Inversion Principle (DIP)**: afferma che una classe dovrebbe dipendere dalle astrazioni, non da classi concrete. Come indicato nella struttura del **Core** dell'orchestratore.

KISS

KISS, acronimo per *Keep It Simple, Stupid*[19], è un principio che impone di mantenere il codice più semplice possibile. Lo scopo ultimo del codice è infatti quello di essere letto da altri umani: le aziende spendono di più per la manutenzione piuttosto che per sviluppo del software[20], quindi gli sviluppatori passano la maggior parte del tempo a sistemare il codice di altri piuttosto che crearne di nuovo.
Durante lo sviluppo dell'orchestratore quindi è stata posta molta attenzione sul refactor, cercando di semplificare al minimo la logica di ogni parte del programma.

DRY

DRY, acronimo di *Don't Repeat Yourself*[21], è un principio che impone di incapsulare le unità logiche applicative in modo da favorirne il riuso, evitando la ripetizione della stesura di linee di codice.
Ripetere in più parti della codebase la stessa funzionalità logica rende onerosa e complicata la manutenzione: la modifica di un concetto logico richiede la riscrittura di più parti dell'applicativo, rendendo difficile tracciare le modifiche effettuate.
Anche per questo principio è stata quindi posta molta attenzione sul refactor in modo da evitare di incappare nell'antipattern WET, *Wasting Everyone's Time*, esatto contrario del principio DRY.

Immediately Repair Your Broken Windows

Per ultimo, ma non meno importante, troviamo il *Immediately Repair Your Broken Windows*[22], principio che unisce teorie dell'ambito della criminologia all'ingegneria del software. In sostanza questa teoria impone che bug, refusi di cattiva progettazione, violazione di principi di programmazione e problemi di questo tipo vadano corretti nell'immediato momento in cui vengono scoperti, in modo da ridurre al minimo il debito tecnico.
L'adozione di questo principio, oltre ad essere buona pratica in ogni situazione, assume

[19] *Software Design Principles DRY and KISS, Arvind Singh Baghel.* URL: https://dzone.com/articles/software-design-principles-dry-and-kiss.
[20] *A software maintenance survey, S.W.L. Yip, T. Lam.* URL: https://ieeexplore.ieee.org/document/465272.
[21] *Software Design Principles DRY and KISS, Arvind Singh Baghel.*
[22] *Don't Live with Broken Windows, Bill Venners.* URL: https://www.artima.com/intv/fixit2.html.

maggiore rilevanza per il progetto di stage: essendo il tempo a disposizione molto limitato, consegnare un prodotto con bug conosciuti e magari non documentati avrebbe incattivito l'opinione sul software sviluppato, diminuendo le possibilità di adozione da parte dell'azienda e mettendo in cattiva luce la mia reputazione professionale.
L'obiettivo del progetto era quindi quello di avere codice, si strutturato e funzionale, ma soprattutto funzionante.

3.4.3 Core

Per la stesura dei diagrammi mostrati qui di seguito è stato utilizzato il dizionario del linguaggio di programmazione Python.

Il cuore di JMeterOrchestrator è stato progettato in modo da rispettare i principi elencati in precedenza e di limitarsi a ricoprire il ruolo base dell'orchestratore: il coordinamento di elementi specializzati.
Questi elementi specializzati prendono il nome di **actions** e sono descritti da sette interfacce che, una volta implementate, assumono le seguenti responsabilità:

* **Wizard**: responsabile di normalizzare le configurazioni e/o i parametri ricevuti dall'utente in modo da essere utilizzabili dall'orchestratore. Eventualmente il Wizard si occupa anche di creare e iniettare la actions (dipendenze);

* **Provisioner**: responsabile di configurare le macchine slaves ed eventualmente master, tramite una potenziale creazione, per l'esecuzione dei test e, una volta terminato, di riportarle allo stato originario, con eventuale distruzione;

* **Dealer**: responsabile di distribuire alle macchine slaves i file di input necessari per l'esecuzione dei test;

* **Parser**: responsabile di modificare la configurazione del file `jmx` secondo i parametri immessi dall'utente;

* **Runner**: responsabile di avviare la macchina master per l'esecuzione dei test;

* **Consumer**: responsabile di consumare i dati prodotti dall'esecuzione dei test;

* **Catcher**: responsabile di catturare eventuali errori (eccezioni) e notificarle all'utilizzatore del software.

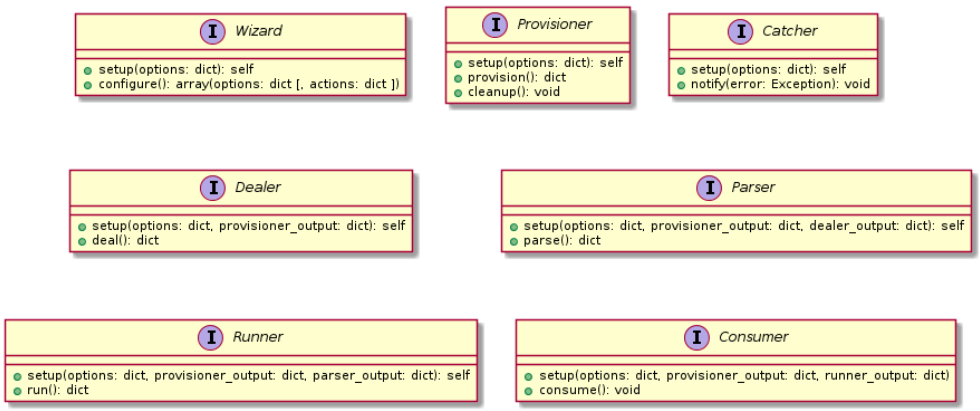


Figura 3.2: Diagramma delle interfacce

Come si può vedere dal diagramma delle classi, queste interfacce sono strutturalmente molto simili: espongono due metodi (al più tre per il *Provisioner*) con firme quasi identiche. Questa differenziazione è stata pensata in modo da rispettare il principio di segregazione delle interfacce (ISP), in modo da rendere immediato al lettore del codice con quale unità logica si stesse confrontando.
L'orchestratore, rappresentato dalla classe *JmeterOrchestrator* presenta un'interfaccia molto semplice e propone il pattern Dependency Injection:

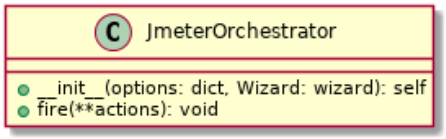


Figura 3.3: Classe JmeterOrchestrator

Le varie actions possono essere infatti "iniettate" dall'utente, sia esso fisico o un'altra componente software, tramite il passaggio di parametri al metodo `fire` o direttamente dal *Wizard* che, in base alla sua implementazione, può essere in grado di crearle. L'utilizzo del pattern Dependency Injection, oltre a rendere la classe funzionale ed estensibile, permette di rispettare al meglio i concetti OCP e DIP.
Il coordinamento effettuato dall'orchestratore, compreso lo scambio di messaggi (parametri) tra le varie *actions* è descritto dal seguente diagramma di sequenza:

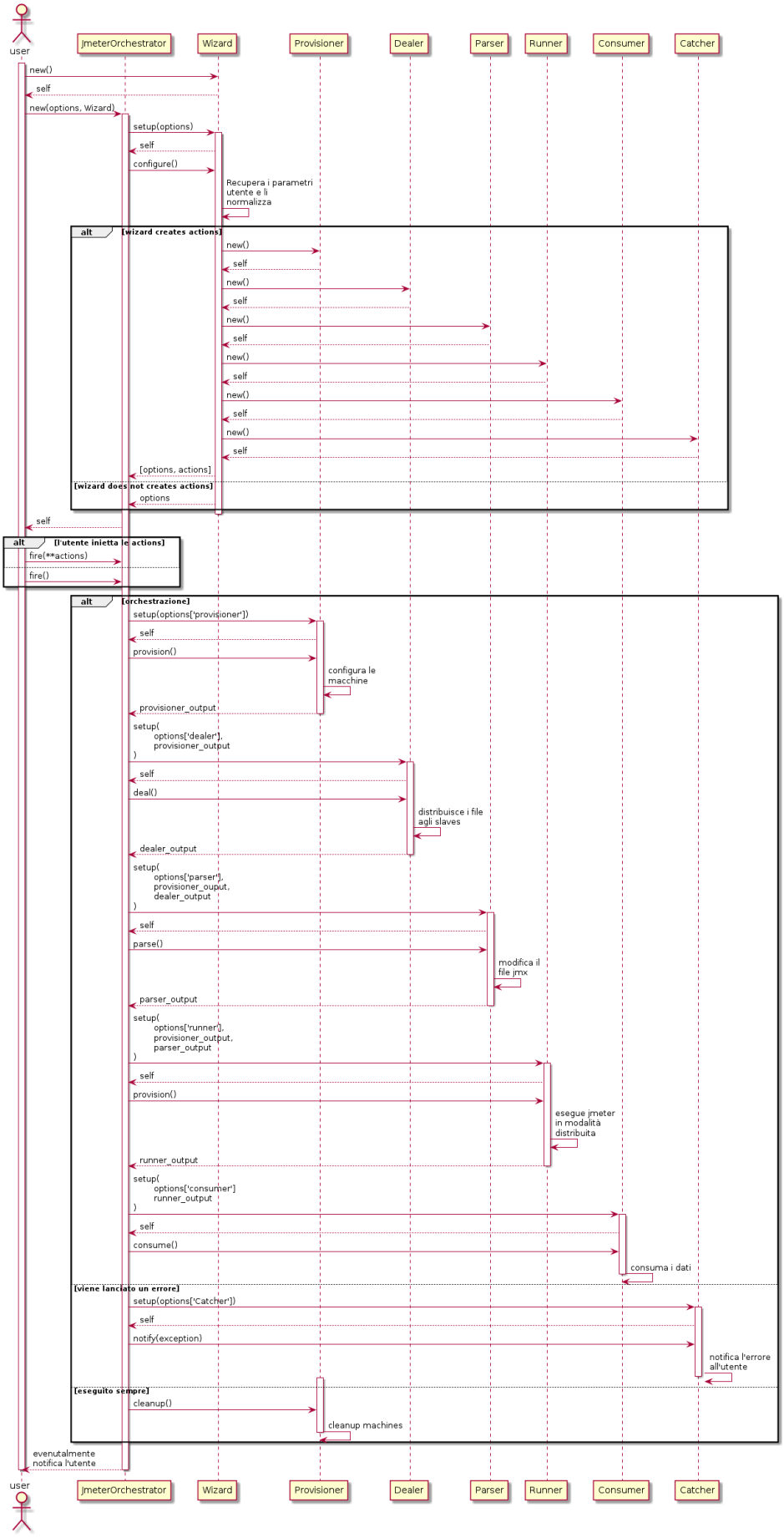


Figura 3.4: Diagramma di sequenza del flusso dell'orchestratore

Gli ultimi attori della progettazione sono rappresentati dai **components** unità logiche utilizzate dalle *actions*, ma indipendenti da queste ultime, realizzati in modo da avvicinarsi ai principi SRP e DRY.
Nelle sezioni seguenti verrano presentate, in forma descrittiva, le *actions* e i *components* implementati per dare significato al prodotto sviluppato. La forma descrittiva è usata in modo da non rivelare troppo i dettagli implementativi delle varie componenti.

3.4.4 Actions

Le *actions* sono raggruppate secondo la loro interfaccia.

Catchers

Consumers

Dealers

Parsers

Provisioners

Runners

Wizards

3.4.5 Components

3.4.6 Diagramma dipendenze actions-components

3.4.7 Command Line Interface

3.4.8 Enterprise Edition

3.5 Installazione nell'infrastruttura aziendale

3.5.1 Configurazione Puppet

Foreman + Gitlab + Artifactory + Puppet + Disegnetto

3.5.2 Analisi delle capacità dei server

3.6 Esecuzione dei test

Sia usando Intranet che AWS dopo ottima collaborazione con gli altri team

3.7 Documentazione prodotta

Capitolo 4

Valutazioni finali

4.1 Soddisfacimento Obiettivi

4.2 Maturazione Professionale

4.3 Considerazioni Personali

Glossario

AgID L'Agenzia per l'Italia digitale (abbreviato AgID) è una agenzia pubblica italiana istituita dal governo Monti. L'Agenzia è sottoposta ai poteri di indirizzo e vigilanza del presidente del Consiglio dei ministri o del ministro da lui delegato. Svolge le funzioni ed i compiti ad essa attribuiti dalla legge al fine di perseguire il massimo livello di innovazione tecnologica nell'organizzazione e nello sviluppo della pubblica amministrazione e al servizio dei cittadini e delle imprese, nel rispetto dei principi di legalità, imparzialità e trasparenza e secondo criteri di efficienza, economicità ed efficacia. 1, 29

Architettura a microservizi L'architettura a microservizi è uno stile architetturale per lo sviluppo di una singola applicazione come un insieme di microservizi, questi sono dei servizi piccoli e autonomi, eseguiti come processi distinti, che lavorano insieme comunicando mediante meccanismi leggeri. Ogni microservizio si occupa di una sola specifica unità applicativa. 2, 29

Branch Nell'ambito del controllo di versione, un branch è una duplicazione di un oggetto (codice sorgente, insieme di file, etc.) che permette di apportare modifiche allo stesso senza impattare la copia originale. I branch vengono solitamente effettuati per creare una nuova funzionalità o apporre una modifica per poi integrarla nel codice principale senza causare problemi nel frattempo, permettendo il lavoro di più persone sulla stessa codebase. 6, 7, 29

Build Nell'ingegneria del software una build è l'output ricavato dalla conversione dell'applicativo da codice sorgente a codice eseguibile. 7, 29

CNA La CNA, Confederazione Nazionale dell'Artigianato e della Piccola e Media Impresa, dal 1946 rappresenta e tutela gli interessi delle micro, piccole e medie imprese, operanti nei settori della manifattura, costruzioni, servizi, trasporto, commercio e turismo, delle piccole e medie industrie, ed in generale del mondo dell'impresa e delle relative forme associate, con particolare riferimento al settore dell'artigianato; degli artigiani, del lavoro autonomo, dei professionisti nelle sue diverse espressioni, delle imprenditrici e degli imprenditori e dei pensionati. 29

Command Line Interface Nell'ingegneria del software, la Quality Assurance è il processo che si occupa di verificare e validare il prodotto sviluppato. 16–20, 22, 29

Configuration Management La gestione della configurazione è una pratica che si occupa di preparare il server ad accogliere l'applicativo desiderato, installando dipendenze, moduli e configurando l'ambiente d'esecuzione. 3, 7, 29

Corporate Social Responsability La CSR (in italiano: Responsabilità sociale d'impresa) è, nel gergo economico e finanziario, l'ambito riguardante le implicazioni di natura etica all'interno della visione strategica d'impresa: è una manifestazione della volontà delle grandi, piccole e medie imprese di gestire efficacemente le problematiche d'impatto sociale ed etico al loro interno e nelle zone di attività. 8, 30

Datacenter In parole semplici un datacenter è la sala macchine che ospita server, storage, gruppi di continuità e tutte le apparecchiature che consentono di governare i processi, le comunicazioni così come i servizi che supportano qualsiasi attività aziendale. Spesso alle aziende non conviene sostenere i costi di un datacenter privato, accedendo a datacenter di terzi tramite servizi di cloud. 7, 30

Deploy Il deploy è l'ultimo step nel rilascio di una nuova versione del software e consiste nell'applicare le modifiche apportate al programma all'interno delle infrastrutture desiderate. 6, 7, 30

DevOps DevOps (contrazione dei termini "development" e "operations") è una cultura/pratica dell'ingegneria del software che mira a unificare lo sviluppo del software e le operazioni effettuate per gestirlo. La caratteristica principale del movimento è il forte orientamento verso l'automazione e il monitoraggio di tutti gli step della costruzione del software, partendo dalla stesura della prima riga di codice fino alla gestione dell'infrastruttura. 4, 12, 13, 22, 30

Domain Specific Language Nell'ingegneria del software, la Quality Assurance è il processo che si occupa di verificare e validare il prodotto sviluppato. 16, 17, 21, 24, 30

Framework Un framework, in informatica e specificatamente nello sviluppo software, è un'architettura logica di supporto (spesso un'implementazione logica di un particolare design pattern) su cui un software può essere progettato e realizzato, spesso facilitandone lo sviluppo da parte del programmatore. 3, 4, 12, 30

Integrated Development Environment (IDE) Gli IDE (in italiano: ambienti di sviluppo integrato) sono strumenti software che supportano il programmatore nello sviluppo del codice sorgente. Questi strumenti solitamente offrono il verificatore di sintassi, l'auto completamento delle sentenze e gli ambienti di debug. 7, 8, 30

CI L'integrazione continua è una pratica dell'ignegneria del software atta a risolvere il problema dell'integration hell: l'insieme di problematiche collegate all'upgrade delle applicazioni in produzione. Facendo un riassunto la CI prevede che ogni commit inneschi un processo che configuri l'applicativo e lo verifichi, in modo da essere facilmente integrato negli ambienti di produzione. Questo processo è completamente automatizzato e prevede due varianti, la continuous delivery, che crea la versione rilasciabile dell'applicativo, e il continuous deploy, che rilascia automaticamente la release negli ambienti specificati. 6, 12, 13, 17–21, 30

Out of the Box Nell'ingegneria del software, la Quality Assurance è il processo che si occupa di verificare e validare il prodotto sviluppato. 17, 20, 24, 25, 30

plugin Il plugin in campo informatico è un programma non autonomo che interagisce con un altro programma per ampliarne o estenderne le funzionalità originarie. 8, 21, 31

Quality Assurance Nell'ingegneria del software, la Quality Assurance è il processo che si occupa di verificare e validare il prodotto sviluppato. 13, 25, 31

Release Una release è l'output ottenuto dall'unione di build e configurazione. Una build infatti spesso, pur essendo eseguibile, non è pronta per essere utilizzata, in quanto priva della configurazione necessaria (url a database, credenziali, variabili d'ambiente, etc.) per fornire valore significativo. La release quindi è il software che viene effettivamente utilizzato negli ambienti esecutivi. 4, 31

Service Level Agreement (SLA) I service level agreement (in italiano: accordo sul livello del servizio) sono strumenti contrattuali attraverso i quali si definiscono le metriche di servizio (es. qualità di servizio) che devono essere rispettate da un fornitore di servizi (provider) nei confronti dei propri clienti/utenti. Di fatto, una volta stipulato il contratto, assumono il significato di obblighi contrattuali. 4, 9, 31

SPID SPID è il sistema di autenticazione che permette a cittadini ed imprese di accedere ai servizi online della pubblica amministrazione e dei privati aderenti con un'identità digitale unica. L'identità SPID è costituita da credenziali (nome utente e password) che vengono rilasciate all'utente e che permettono l'accesso a tutti i servizi online. 1, 31

Stakeholders Tutti i soggetti, individui od organizzazioni, attivamente coinvolti in un'iniziativa economica (progetto, azienda), il cui interesse è negativamente o positivamente influenzato dal risultato dell'esecuzione, o dall'andamento, dell'iniziativa e la cui azione o reazione a sua volta influenza le fasi o il completamento di un progetto o il destino di un'organizzazione. 4, 13, 31

Technical debt Il debito tecnico, nell'ambito dello sviluppo software, è un concetto che spiega il grosso costo da sostenere per modificare software figlio di scelte di design non appropriate. Capita spesso infatti che nuove funzionalità, vuoi per mancanza di tempo e/o di capacità, vengano sviluppate seguendo un approccio rapido piuttosto che ben congegnato, rendendo molto difficile la sua modifica in periodi successivi. Questo debito, se non ripagato subito, applica gli "interessi" nel tempo: ogni estensione del software fara affidamento sulle funzionalità "indebitate" rendendo molto costoso effettuare una modifica. Un esempio di debito tecnico si trova quando nel modificare una piccola parte del software, diventa necessario modificare tutte le parti che interagiscono con questa. 12, 31

Version Control System Nell'ingegneria del software, la Quality Assurance è il processo che si occupa di verificare e validare il prodotto sviluppato. 16, 17, 31

Way Of Working Il way of working è l'insieme di metodi, strumenti e procedure atte a guidare e supportare il lavoro dei team aziendali. Questo modello per funzionare al meglio viene deciso a priori e viene adottato da tutto il team senza obiezioni. Questo non vuol dire che il WoW sia un entità costante, un buon Way of Working infatti deve essere in grado di evolversi e migliorare se stesso. 5, 31

Bibliografia

Riferimenti bibliografici

Martin, Robert C. "Uncle Bob". *Design Principles and Design Patterns*. objectmentor.com, 2000 (cit. a p. 32).

Siti web ufficiali

Amazon Web Services Elastic Compute. URL: https://aws.amazon.com/it/ec2/ (cit. a p. 26).

AngularJS. URL: https://angularjs.org/ (cit. a p. 3).

Artifactory. URL: https://jfrog.com/artifactory/ (cit. a p. 7).

Confluence. URL: https://it.atlassian.com/software/confluence (cit. a p. 6).

Docker. URL: https://www.docker.com/ (cit. a p. 29).

Foreman. URL: https://www.theforeman.org/ (cit. a p. 7).

Git. URL: https://git-scm.com/ (cit. a p. 6).

Gitlab. URL: https://about.gitlab.com/ (cit. a p. 6).

Google Cloud Platform Computer Engines. URL: https://cloud.google.com/compute/ (cit. a p. 26).

Java. URL: https://www.java.com/ (cit. a p. 2).

Jenkins. URL: https://jenkins.io/ (cit. a p. 6).

Jetbrains. URL: https://www.jetbrains.com/ (cit. a p. 7).

Jira. URL: https://it.atlassian.com/software/jira (cit. a p. 6).

Kubernetes. URL: https://kubernetes.io/ (cit. a p. 29).

Microsoft. URL: https://www.microsoft.com/ (cit. a p. 8).

Microsoft Azure Virtual Machines. URL: https://azure.microsoft.com/it-it/pricing/details/virtual-machines/linux/ (cit. a p. 26).

Nomad. URL: https://www.nomadproject.io/ (cit. a p. 29).

Oracle. URL: https://www.oracle.com/ (cit. a p. 2).

Oracle Secure Global Desktop. URL: https://www.oracle.com/it/secure-global-desktop/ (cit. a p. 7).

Packer. URL: https://www.packer.io/ (cit. a p. 29).

Puppet. URL: https://puppet.com/ (cit. a p. 7).

Visual Studio Code. URL: https://code.visualstudio.com/ (cit. a p. 8).

Articoli consultati

A software maintenance survey, S.W.L. Yip, T. Lam. URL: https://ieeexplore.ieee.org/document/465272 (cit. a p. 33).

AWS CPU Credits, Amazon. URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-credits-baseline-concepts.html (cit. a p. 27).

AWS EC2 Spot instances, Amazon. URL: https://aws.amazon.com/ec2/spot/pricing (cit. a p. 27).

Deployment vs Provisioning vs Orchestration vs Configuration Management, Peter Souter. URL: https://archive.fosdem.org/2018/schedule/event/deployment_provisioning_orchestration/ (cit. a p. 26).

Don't Live with Broken Windows, Bill Venners. URL: https://www.artima.com/intv/fixit2.html (cit. a p. 33).

GCP Preemptible instances, Google. URL: https://cloud.google.com/compute/docs/instances/preemptible (cit. a p. 27).

Metodoloigia SCRUM. URL: https://www.scrum.org/resources/what-is-scrum.

Open-Closed Model Load Generators, Adam Wierman. URL: http://users.cms.caltech.edu/~adamw/publications-openclosed.html (cit. a p. 23).

Software Design Principles DRY and KISS, Arvind Singh Baghel. URL: https://dzone.com/articles/software-design-principles-dry-and-kiss (cit. a p. 33).

Top 3 Terraform Testing Strategies for Ultra-Reliable Infrastructure-as-Code, Carlos Nunez. URL: https://www.contino.io/insights/top-3-terraform-testing-strategies-for-ultra-reliable-infrastructure-as-code (cit. a p. 28).

What is infrastructure as code?, Margaret Rouse. URL: https://searchitoperations.techtarget.com/definition/Infrastructure-as-Code-IAC (cit. a p. 25).

What is pay-as-you-go cloud computing?, Margaret Rouse. URL: https://searchstorage.techtarget.com/definition/pay-as-you-go-cloud-computing-PAYG-cloud-computing (cit. a p. 26).

Writing Custom Providers, Terraform. URL: https://www.terraform.io/docs/extend/writing-custom-providers.html (cit. a p. 28).