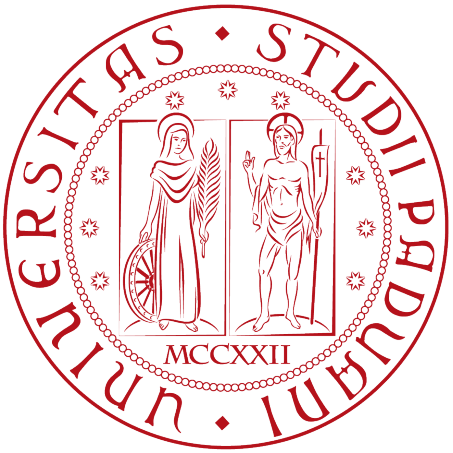
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Studio e applicazione della Chaos Engineering nello sviluppo di un microservizio reactive

Tesi di laurea

Relatore

Prof.ssa Silvia Crafa

Laureando

Alessandro Rizzo

Anno Accademico 2019-2020

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

— Oscar Wilde

Dedicato a ...

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecentoventi ore, dal laureando Alessandro Rizzo presso l'azienda Infocert S.p.A. Gli obiettivi da raggiungere erano molteplici.
In primo luogo era richiesto lo studio e la comprensione dei fondamenti della Chaos Engineering In secondo luogo era richiesta l'implementazione di una versione rivisitata di un software aziendale già esistente, MICO, in seguendo i principi dell'architettura a microservizi e reactive tramite il framework Akka. Tale framework permette di utilizzare il modello ad attori per gestire il completamento di diversi task simultaneamente e in maniera asincrona. In questo sviluppo andava applicato quanto appreso nella fase di studio per progettare e realizzare un'applicazione il più resiliente possibile. Infine, una volta completato lo sviluppo, andavano applicati tutti i principi di Chaos Engineering appresi durante la fase di studio per aumentare la fiducia nell'applicazione e per scoprire eventuali vulnerabilità non ancora considerate con lo scopo di aumentare la resilienza e l'affidabilità del prodotto.

"Life is really simple, but we insist on making it complicated"

— Confucius

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. NomeDelProfessore, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Desidero ringraziare con affetto i miei genitori per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Introduzione al contesto applicativo.

Esempio di utilizzo di un termine nel glossario
Application Program Interface (API).

Esempio di citazione in linea
site:agile-manifesto.

Esempio di citazione nel pie' di pagina
citazione[1]

1.1 L'azienda

Infocert S.p.A. è una delle più importanti Certification Authority a livello europeo e fornisce servizi di firma digitale, Posta Elettronica Certificata (PEC)[g], Sistema Pubblico di Identità Digitale (SPID)[g], e fatturazione elettronica. L'obiettivo principale dell'azienda è quello di rendere disponibili tutti gli strumenti per la creazione di un ufficio digitale, fornendo ai propri clienti soluzioni paperless. Queste vengono fornite ad altre aziende e a professionisti.

1.2 L'idea

Il progetto di stage è nato da un bisongno dell'azienda: analizzare l'utilizzo di un framework nuovo per sviluppare applicazioni moderne ed esplorare i principi della Chaos Engineering per una eventualmente graduale integrazione ne ciclo di sviluppo aziendale con lo scopo di creare prodotti di maggior qualità e ridurre il costo di mantenimento dek software. In particolare il framework coinvolto è Akka ed è stato deciso di realizzare una nuova versione di un prodotto aziendale già esistente. Infine è stato deciso di eseguire dei test per analizzare le performance tra l'applicativo appena sviluppato e la sua versione precedente.

[1] **womak:lean-thinking**.

1.3 Organizzazione del testo

Il secondo capitolo descrive ...

Il terzo capitolo approfondisce ...

Il quarto capitolo approfondisce ...

Il quinto capitolo approfondisce ...

Il sesto capitolo approfondisce ...

Nel settimo capitolo descrive ...

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

* gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
* per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*[g];
* i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Chaos Engineering: cos'è, principi e metodologie

In questo capitolo tratterò della Chaos Engineering, della sua storia, dei suoi principi e dei vantaggi che porta all'interno di un team di sviluppo

2.1 Come nasce la Chaos Engineering

Nel 2017 un'intervista rivolta ad aziende che basano almeno una parte del proprio fatturato sul web ha rivelato che il 98% delle aziende perderebbe almeno 100.000€ di fatturato con un'ora di downtime del sistema.

Questa intervista ha evidenziato un problema che però andava avanti già da diversi anni, nel 2010 infatti hanno iniziato ad affermarsi queste realtà basate sul cloud e il cui fatturato dipendeva direttamente dalla reattività e dalla disponibilità dei loro sistemi. Questo unito all'aumento del numero dei servizi offerti tramite il cloud ha portato alla formazione di software dall'architettura molto complessa e quindi vulnerabile a dei malfunzionamenti software o hardware molto difficili da prevedere.

Per questo nello stesso anno un team all'interno di Netflix creò un software chiamato poi "Chaos Monkey", questo software aveva lo scopo di terminare in maniera randomica alcune istanze dell'applicazione desiderata all'interno del deploy di produzione Netflix, mettendo così in difficoltà in sistema e andando a creare dei malfunzionamenti controllati prima che questi si verificassero autonomamente causando quindi un grosso danno economico all'azienda. Nei due anni successivi Netflix inizia così a misurare e migliorare la resilienza del proprio sistema facendo emergere delle vulnerabilità prima sconosciute.

Nel 2012, Netflix rende noto il codice sorgente di Chaos Monkey iniziando a suscitare l'interesse di altre grandi compagnie come Amazon che si approcciano così alla Chaos Engineering. Per altri due anni Netflix continua questi esperimenti creando nuovi tool per andare a misurare diversi aspetti dei propri sistemi.

Nel 2014 infine Netflix inventa un nuovo ruolo all'interno dei propri team di sviluppo, il Chaos Engineer, andando a porre le basi per la disciplina della Chaos Engineering anche grazie al loro nuovo software FIT. Questo software permette per la prima volta di effettuare esperimenti di Chaos Engineering veri e propri poichè li svolge in un ambiente controllato, include degli strumenti di monitoraggio dello stato del sistema e consente di effettuare "rollbacks" se ci fossero malfunzionamenti imprevisti.

2.2 I Principi della Chaos Engineering

La Chaos Engineering è una pratica relativamente recente ma una cosa su cui molti dei professionisti che la praticano sono d'accordo è questa definizione: Osservazione di un sistema distribuito in un esperimento controllato.

I punti più importanti che emergono da questa definizione e che approfondirò nei capitoli seguenti sono due: l'osservazione del sistema e l'esperimento controllato; contrariamente a quanto si può evincere dal nome infatti nella Chaos Engineering è molto importante eseguire gli esperimenti in un ambiente controllato in cui sia facile recuperare lo stato normale del sistema.

L'obiettivo principale che questa disciplina si pone é aumentare la "confidence" che il team di sviluppo e gli stakeholder di un prodotto hanno nei confronti dello stesso oltre a scoprire eventuali vulnerabilità del sistema prima che esse si verifichino in un ambiente non controllato.

Per quanto riguarda le caratteristiche del prodotto la Chaos Engineering cerca di massimizzarne 4 tramite gli esperimenti:

Performance latenza minima e massima capacità di carico

Availability tempo in cui il server è in grado di rispondere alle richieste in entrata, una metrica molto importante per le architetture a microservizi

Fault tolerance velocità con cui un sistema è in grado di ritornare al suo stato normale da uno indesiderato

Velocità di sviluppo* velocità a cui il team di sviluppo rende disponibili nuove feature ai clienti

L'obiettivo principale però rimane comunque la fiducia del team e degli stakeholder nel prodotto ed è per questo che è molto importante che l'esperimento non produca effetti indesiderati che vanno fuori scala.

2.3 L'esperimento

Un esperimento di Chaos Engineering rappresenta il cuore della disciplina e può essere fondamentalmente riassunto nell'inserimento di eventi in un ambiente controllato per osservare il comportamento del sistema. L'esecuzione vera e propria dell'esperimento invece può essere suddivisa in 5 step fondamentali:

1. Definire uno stato preciso del sistema che ne indica il comportamento normale, questo serve per poter controllare in maniera semplice durante l'esperimento l'effetto sul sistema degli eventi che vengono introdotti.

2. Ipotizzare in che modo lo stato del sistema potrebbe variare e cosa potrebbe causare questa variazione, meglio un'ipotesi alla volta per rendere l'esperimento più facile da gestire.

3. Introdurre nuovi eventi *realistici* che mettano in difficoltà il nostro sistema (crash dei server, malfunzionamenti hardware, esaurimento delle risorse, aumento della latenza nella comunicazione tra i microservizi, fallimento bizantino, concorrenza ecc.), questi eventi devono necessariamente partire come problemi contenuti e poi eventualmente aumentare di intensità nel caso in cui il sistema si rivelasse essere resiliente.

4. Controllare frequentemente se lo stato del sistema è stato modificato dagli eventi introdotti prima,se così fosse infatti avremmo individuato un problema. In questo passaggio è molto importante valutare metriche semplici e che mostrino in maniera evidente l'insorgere di problemi, più sono meglio è.

5. Se lo stato dovesse rimanere invariato, il sistema si è rivelato essere valido e l'esperimento si può ripetere aumentando il Blast radius.

Un esperimento tuttavia ha bisogno sia di una solida preparazione prima che di una adeguata riflessione dopo per poter essere effettivamente utile al team.

2.4 Best practices per l'esperimento

Di seguito andrò ad elencare alcune best practices estrapolate da diversi articoli o conferenze di esperti di Chaos Engineering:

1. Nel punto 1 della sezione sopra è meglio concentrarsi sull'output del sistema piuttosto che sulle sue caratteristiche interne per definire lo stato normale, questo perchè l'obiettivo ultimo è misurare come cambia l'esperienza degli utenti a fronte degli eventi inseriti (Amazon per esempio utilizza il numero di ordini ricevuti in un determinato lasso di tempo)

2. Osservare le metriche per determinare lo stato del sistema dovrebbe costare il minimo sforzo sia per una persona che per un tool automatizzato

3. le metriche scelte devono riflette in tempo reale lo stato del sistema

4. Nel punto 3 è importante scegliere eventi che riflettano problemi realistici e che possano potenzialmente disturbare il normale funzionamento del sistema

5. Si consiglia di eseguire gli esperimenti in ambiente di produzione proprio per simulare con il massimo realismo le condizioni in cui si verificherebbero i problemi che vogliamo evitare

6. Secondo l'esperienza degli ingegneri Netflix i risultati migliori si ottengono inserendo più fallimenti insieme, così si riescono a scoprire delle vulnerabilità molto difficili da prevedere per il team di sviluppo; questo tuttavia deriva anche dal fatto che Netflix ha diversi anni di esperienza alle spalle e sa come rendere un esperimento complesso gestibile.

7. Se possibile bisognerebbe puntare ad avere un sistema in grado di riprendersi da solo da uno stato indesiderato, la necessità di un intervento umano amenta infinitamente il tempo medio di downtime del sistema in caso di malfunzionamenti, una cosa inaccettabile per un'applicazione cloud.

8. Per le stesse motivazioni del punto precedente non bisogna puntare a creare un sistema immune ai fallimenti, perchè questi accadono, ma piuttosto ad uno che sia in grado di riprendersi velocemente e autonomamente.

2.5 analisi dopo l'esperimento

Per rendere utili i risultati ottenuti da un esperimento è importante ragionarci a posteriori con tutte le persone coinvolte, sia per avere diverse opinioni sull'esperimento

al fine di migliorare quelli sucessivi (specialmente nelle fasi iniziali) sia per analizzare e comprendere quanto è successo durante l'esperimenti affinchè si possa convertire in miglioramenti futuri per il prodotto. Bisogna redarre un documento scritto per tenere traccia di quanto svolto durante l'esperimento e di quanto discusso durante quest'analisi a posteriori, in questo caso il documento viene chiamato generalmente Correction-of-Errors e si compone di 5 sezioni principali:

1. Cos'è successo
2. Qual'è stato l'impatto
3. Perchè si è verificato questo errore
4. Cosa abbiamo imparato
5. Come possiamo evitare che ricapiti in futuro

Questo documento scritto ha una duplice funzione a sua volta, serve sia come registro dell'esperimento e delle decisioni prese ma anche da pubblicità all'interno dell'azienda di ciò che il team di sviluppo sta portando avanti specialmente nelle prime fasi di adozione della Chaos Engineering.

2.6 preparazione all'esperimento

Un esperimento di Chaos Engineering, specialmente nelle prime fasi di adozione, è sempre un rischio perchè non si è mai veramente sicuri di come si comporta il sistema che andremo ad esplorare. Per questo dunque la preparazione di un esperimento è molto importante, come prima cosa bisogna avere ben chiara la struttura del sistema e quali sono le sue possibili vulnerabilità.

È consigliabile già dalla progettazione di un prodotto avere uno schema del suddetto per poterlo meglio visualizzare, questo schema andrà poi analizzato con tutto il team secondo le tecniche di FMEA* (che spiegherò nei capitoli successivi) al fine di avere un'idea ben chiara dell'architettura del sistema. Questo schema è utile soprattutto in sistemi molto complessi anche per mappare le dipendenze critiche e non critiche del sistema, quelle non critiche sono un ottimo punto di partenza per gli esperimenti in quanto il sistema dovrebbe comunque garantire il servizio anche senza i servizi verso cui ha una dipendenza non critica. Alcuni strumenti possono aiutare nell'individuazione delle dipendenze: VPC flow logs e AWS X-ray ne sono un esempio.

In secondo luogo conviene guardare agli esperimenti passati, se ne esistono, per scoprire quali sono i pattern che spesso generano problemi e che possono essere eventualmente meglio esplorati con un nuovo esperimento. Bisogna anche porre molta attenzione all'overconfidence effect: non bisogna avere il bias per cui se si è speso molto tempo e molte risorse su una determinata tecnologia essa certamente funzionerà, maggiore è questa sicurezza maggiore saranno i danni di un esito inaspettato dell'esperimento.

2.7 Blast radius

Vale la pena spendere una sezione per parlare nel dettaglio del blast radius poichè si può dire che ogni parametro del test lo influenza ed è quindi essenziale che sia ben pensato per la buona riuscita di un esperimento, inoltre svolgere gli esperimenti nell'ambiente di

produzione ha lo svantaggio di poter causare dei disagi agli utilizzatori del prodotto, cosa che vogliamo assolutamente evitare. In generale quando si effettua un esperimento di Chaos Engineering in produzione si prende un sottoinsieme dell'ambiente di produzione targettizando uno specifico gruppo di dispositivi, questo gruppo inizialmente dovrà essere composto da un numero molto ristretto di dispositivi visto che se il sistema avesse problemi anche con un così piccolo campione di utenti sarebbe insensato aumentare ulteriormente il numero dei dispositivi. In questo modo andiamo a fornire un disservizio ad un numero minimo di utenti se ci dovessero essere dei problemi ed è molto più facile seguire i rollback ai singoli errori. Se il sistema invece dovesse rivelarsi affidabile con piccoli gruppi di utenti allora il campione preso in esame durante l'esperimento può essere allargato aumentando il numero di cluster in cui si svolge l'esperimento. Infine il passaggio finale sarebbe estendere il test a tutto l'ambiente di produzione per vedere come funziona la redistribuzione del traffico, il circiut breaker e la gestione condivisa delle risorse nel deploy. Bisogna fare attenzione tuttavia a quest'ultimo passaggio, richiede davvero molta esperienza negli esperimenti di Chaos Engineering e una grande fiducia nel proprio sistema. Netflix e Amazon ora effettuano esperimenti che riguardano anche più zone AWS contemporaneamente ma hanno anni di esperienza nella Chaos Engineering.

2.8 Osservare il sistema

Osservare il sistema è uno dei passaggi più importanti dell'esperimento, è ciò che ci permette di verificare lo stato del sistema e di capirne meglio il funzionamento. Ma cosa conviene osservare? Sicuramente dobbiamo concentrarci sugli output del sistema piuttosto che sulle sue caratteristiche interne, ad esempio la latenza nell'invio delle risposte, lo stato di salute dei vari microservizi, la correttezza delle risposte. Queste unità di misura sono più immediate e ci riferiscono con certezza e in tempo reale se qualcosa nel sistema non sta funzionando nel modo corretto. Osservare in maniera costante il sistema tramite degli strumenti automatici durante gli esperimenti è una parte molto importante per raggiungere la maturità nella Chaos Engineering, strumenti diversi si adattano bene a sistemi diversi e ci permettono di monitorare ciascuno un aspetto specifico del sistema. Una suite abbastanza ampia di strumenti quindi ci permette di osservare in maniera completa il nostro prodotto per trarre delle conclusioni migliori dai nostri esperimenti.

2.9 GameDay

Amazon, quando nel 2014 iniziò a praticare la Chaos Engineering, diede molta importanza alla partecipazione di tutto il team di sviluppo, compresi responsabili di progetto e ogni figura in qualche modo coinvolta con il prodotto in question, a tal punto da istituire un evento all'interno della propria azienda chiamato GameDay. Il GameDay è un periodo di circa 4-6 ore collocabile in qualsiasi punto della giornata, anche fuori dal normale orario d'ufficio, e viene organizzato e seguito dai Chaos Engineers in collaborazione appunto con il team di sviluppo. In un GameDay vengono specificati determinati obiettivi da raggiungere tramite uno o più esperimenti di Chaos Engineering rivolti a certi specifici aspetti del prodotto, la data solitamente viene annunciata preventivamente al team ma l'argomento può essere omesso fino alla data stabilita così da testare anche il team di sviluppo in una situazione di stress. Due novità importanti introdotte dal GameDay sono la partecipazione globale di tutte le

persone coinvolte nel progetto e che in questi eventi non è solo il prodotto ad essere testato per il miglioramento ma anche il team di sviluppo. Il team che viene convocato, se dovessero verificarsi situazioni impreviste, deve essere pronto a risolvere entro la durata del Gameday ed è proprio per questo che un fallimento disastroso avrebbe un impatto doppio su tutto il team e anche una risonanza a livello aziendale. La struttura di un Gameday in fondo è simile a quella di un esperimento e si compone di 3 fasi:

Progettazione in questa fase è importante identificare il rischio di fallimento dei vari esperimenti e mitigarlo

Esecuzione l'obiettivo in questa fase è aumentare la sicurezza con cui il team gestisce le situazioni critiche e aumentare la fiducia nel prodotto.

Report come negli esperimenti alla fine è importante scrivere un report, questo migliora la percezione del Gameday all'interno dell'azienda (specialmente se ha avuto un esito positivo) e aumenta la partecipazione ai prossimi Gameday; inoltre aiuta il team a fare il punto su cosa sia andato bene o male durante il Gameday.

2.10 adozione della Chaos Engineering

Essendo la Chaos Engineering una pratica relativamente recente nell'ambiente della Software Engineering uno dei temi più discussi è l'adozione della Chaos Engineering all'interno di un'azienda e soprattutto da dove cominciare. Per iniziare è improbabile che sia direttamente l'amministrazione del proprio ente a proporre di adottare questa nuova metodologia di sviluppo, generalmente quindi sarai tu ingegnere che si interessa a questa pratica e che vuole proporre di adottarla per il proprio team di sviluppo. Inoltre probabilmente 4 colleghi su 5 penseranno che sia una pessima idea introdurre dei malfunzionamenti mirati in produzione, l'adozione in azienda all'inizio è quindi quasi sempre il tentativo di poche persone di convincere tutti gli altri che quello che fanno abbia un senso, ma come fare? Nella prima parte la tua attività principale sarà informare tutti i membri del team di sviluppo di quello che stai per fare, mandare loro articoli, conferenze e qualsiasi altra cosa che possa aiutarli ad approcciarsi alla Chaos Engineering. Poi una volta scelti gli strumenti da utilizzare è ora di progettare il primo Gameday, dev'essere sicuramente qualcosa di molto semplice e che quasi sicuramente produrrà un esito positivo per il prodotto.

I primi esperimenti sono sempre i più importanti, cerca di non traumatizzare il tuo team al loro primo Gameday oppure la loro adozione della Chaos Engineering finirà quello stesso giorno. Cerca invece di coinvolgere quante più persone possibili ai tuoi Gameday, anche persone che non c'entrano nulla con il prodotto testato; più spettatori ci saranno più il tuo lavoro avrà una risonanza all'interno dell'azienda e altre persone potrebbero convincersi a provare la Chaos Engineering. Alcuni Gameday andranno male ma nel report finale cerca sempre di evidenziare l'aspetto positivo di quanto scoperto e di come il prodotto e il ciò che è stato fatto dal team di sviluppo può essere migliorato. Con il team di sviluppo che acquisisce sempre più familiarità con queste pratiche sarebbe il caso di aumentare il blast radius per mettere più a dura prova il prodotto e il team e aumentare la frequenza degli esperimenti: a pieno regime ci dovrebbe essere circa un piccolo esperimento ogni settimana o due e un gameday che riguardi l'intero ambiente di produzione una volta al mese.

2.11 ordine dei Test

Quando si deve scegliere l'argomento di un esperimento o di un Gameday bisogna ragionare sugli esperimenti fatti in precedenza. In ogni caso la linea guida su cui gli esperti concordano è iniziare sempre da ciò che già si conosce sia per prendere confidenza con gli esperimenti sia per aumentare la fiducia nel prodotto. In particolare dovremmo partire da ciò che conosciamo e che sappiamo bene come funziona, questo ci aiuterà durante gli esperimenti a recuperare informazioni su ciò che conosciamo e che non sappiamo ancora come funziona. Date le basi che ora abbiamo acquisito su ciò che conosciamo, nei successivi esperimenti ciò che ci apparirà nuovo e non conosciuto sarà il nostro prossimo target.

Bisogna insomma mettere prima delle basi di conoscenza del funzionamento del proprio sistema, particolarmente importante se la sua architettura è molto complessa, per poi osservare e sperimentare gli aspetti meno chiari del sistema. Difficilmente si arriverà in un punto in cui una sola persona potrà aver chiaro il funzionamento dell'intero sistema, dunque è una pratica destinata a continuare e ad evolversi nel tempo.

Capitolo 3

Approccio pratico alla Chaos Engineering

In questo capitolo discuterò di come approcciarsi in pratica allo sviluppo di un prodotto seguendo i principi della Chaos Engineering.

3.1 Creare una raccolta di ipotesi

La Chaos Engineering, come abbiamo visto, è una pratica con delle regole ben specifiche che ha lo scopo di aumentare la fiducia del prodotto. La prima cosa da fare quando ci si approccia a questa pratica è chiaramente fare un esperimento, iniziare subito a rilasciare il chaos nel proprio ambiente di produzione però non è un metodo propriamente ortodosso. Conviene invece fare un passo indietro e capire che esperimenti conviene fare e come conviene farli, bisogna costruire quindi una raccolta di ipotesi sul nostro sistema che ci aiuteranno poi a scegliere con criterio quali sono gli esperimenti che meglio posso contribuire alla fiducia del prodotto. Ci sono due fonti principali di ipotesi:

1. Malfunzionamenti passati analizzando le cause
2. Analisi dell'architettura del sistema chiedendosi cosa potrebbe andare storto oppure cosa ci preoccupa di più

Se sei alle prese con un nuovo prodotto l'unica opzione che ti resta e la seconda, per metterla in pratica serve avere uno "schizzo" del sistema su cui ragionare e il momento migliore in cui cominciare è la progettazione.

3.2 Progettare pensando al Chaos

Iniziare a pensare ai possibili rischi nascosti di un'architettura già dalla sua progettazione porta con se numerosi vantaggi, cominciando dal fatto che permette di individuare già dal principio alcune problematiche dell'architettura e eventualmente di correggerle subito. Per cominciare è necessario avere uno schizzo del sistema per ragionarci sopra insieme al team di sviluppo individuando secondo un metodo chiamato FMEA che andremo a descrivere nella sezione successiva. Lo schizzo del sistema che andremo ad

usare dev'essere abbastanza dettagliato da permettere al team di porsi domande su specifiche componenti dell'architettura e sul loro funzionamento, più persone collaborano e più lo schema sarà completo, probabilmente si arriverà ad ottenere più di uno schizzo ma va bene così. A questo punto analizzando questi schizzi si inizia a cercare possibili errori, malfunzionamenti, imperfezioni che andranno annotati con il componente a cui fanno riferimento e una breve descrizione; quando si arriverà all'esaurimento di queste idee bisognerà iniziare a catalogare questi rischi del sistema.

3.3 FMEA

Failure Mode and Effect Analysis è una tecnica per l'analisi degli errori in un'architettura e consiste essenzialmente nel controllare quanti più componenti e sottosistemi possibili dell'architettura al fine di individuare potenziali errori, le loro cause e le conseguenze che si verificherebbero sul sistema. Il risultato di questi controlli viene poi riportato nella tabella FMEA, di questa ne esistono numerose varianti a seconda di quale aspetto dei problemi individuati ci interessa per catalogarli. Potremmo per esempio avere interesse a catalogare questi rischi in base alla probabilità che accadano o alla gravità dell'evento e conseguentemente avremo tabelle diverse. Alcune aziende invece trovano più efficace adibire una parete a questo scopo attaccandoci diversi post-it ciascuno con la descrizione e il livello di probabilità del rischio. Nel caso specifico della Chaos Engineering i due parametri fondamentali sono la probabilità e l'impatto dei rischi, in ogni voce della nostra tabella o in ogni post-it dunque dovrà comparire il componente a cui il rischio fa riferimento, una breve descrizione di quest'ultimo e delle possibili cause, la probabilità e l'impatto del rischio in una scala a piacere, quali caratteristiche del sistema questo rischio andrebbe a compromettere e il contributo che darebbe al sistema se questo problema fosse risolto (ad esempio aumenta la resilienza, aumenta l'availability, riduce il tempo di risposta). Una volta esauriti i rischi vale la pena spendere ancora dei momenti di riflessione con il team per pensare a come risolvere i rischi che sono stati elencati eventualmente aggiungendo una voce alla tabella, in questo modo i rischi possono essere già mitigati in fase di sviluppo dell'applicazione o eventualmente può esser facilitato il percorso della Chaos Engineering in seguito. Mettiamo in chiaro però che quello che ho appena descritto e come le cose andrebbero fatte ma ci sono infinite soluzione intermedie o anche più avanzate che possono adattarsi alle esigenze del team, nella mia esperienza di stage infatti il nostro team di sviluppo ha adottato una versione semplicata di questa tecnica più adatta ai tempi che avevamo a disposizione.

3.4 Lo sviluppo

Durante lo sviluppo è buona pratica aggiornare regolarmente la tabella del capitolo precedente con nuovi rischi o rimuovendo quelli già risolti in modo da arrivare alla fine della fase di sviluppo con una lista dei rischi valida per essere di aiuto nella creazione di esperimenti di Chaos Engineering.

3.5 Strumenti analizzati

Qui sotto riporterò i principali strumenti analizzati per praticare la Chaos Engineering, alcuni di loro hanno lo scopo di inserire errori nel sistema, altri di osservarne i parametri

e altri ancora si pongono come orchestratori degli strumenti citati prima.

3.5.1 Chaos Monkey

Chaos Monkey è uno strumento storico nella disciplina in quanto è stato il primo creato da Netflix, il suo scopo principale è terminare randomicamente istanze dell'applicazione desiderata nel deploy, è uno strumento abbastanza basilare con poche possibilità di personalizzazione ma è un buon punto di partenza.

Tabella 3.1: Pro e Contro di Chaos Monkey

Pro	Contro
Semplice da imparare e configurare	Poche possibilità di personalizzazione degli esperimenti Richiede che l'applicazione sia sviluppata con Spinnaker

3.5.2 Simian army

Questo è il team di Netflix ha dato alla sua suite di strumenti per la Chaos Engineering, ognuno ha un compito preciso e insieme permettono di inserire eventi di vario tipo all'interno del sistema. Di seguito riporto una lista degli strumenti principali:

Chaos Monkey

Latency Monkey : inserisce dei ritardi nelle comunicazioni tra client e server o tra i vari microservizi simulandone un cattivo stato di salute

Doctor Monkey : controlla gli stati di salute delle varie istanze per rimuovere quelle malate

Conformity Monkey : trova istanza che non aderiscono alle best-practice fissate

Chaos Kong

3.5.3 Chaos Kong

Chaos Kong, a differenza di Chaos Monkey, è progettato per spegnere intere zone AWS durante l'esperimento, inoltre ridireziona il traffico in una o più zone dove l'applicazione è ancora deployata per controllare se il loro sistema è resiliente persino al malfunzionamento di un'intera regione AWS. Un esperimento che Netflix effettua con Chaos Kong ad esempio consiste nel terminare la zona us-east e ridirigere tutto il traffico verso la zona us-west per vedere se è da sola in grado di reggere tutto il traffico proveniente dagli stati uniti.

3.5.4 FIT

Failure Injection Testing, o FIT è una delle ultime soluzioni di Netflix alla ricerca di un software che consenta di inserire dei malfunzionamenti in maniera precisa e controllata. FIT è una piattaforma che semplifica l'inserimento di eventi rispetto ai software precedenti, o meglio, lo rende più sicuro; Alcune "scimmie" infatti hanno il difetto di essere difficilmente controllabili e FIT offre un modo per limitare l'impatto dei fallimenti ed essere in grado di controllare meglio il risultato di un esperimento. L'inserimento

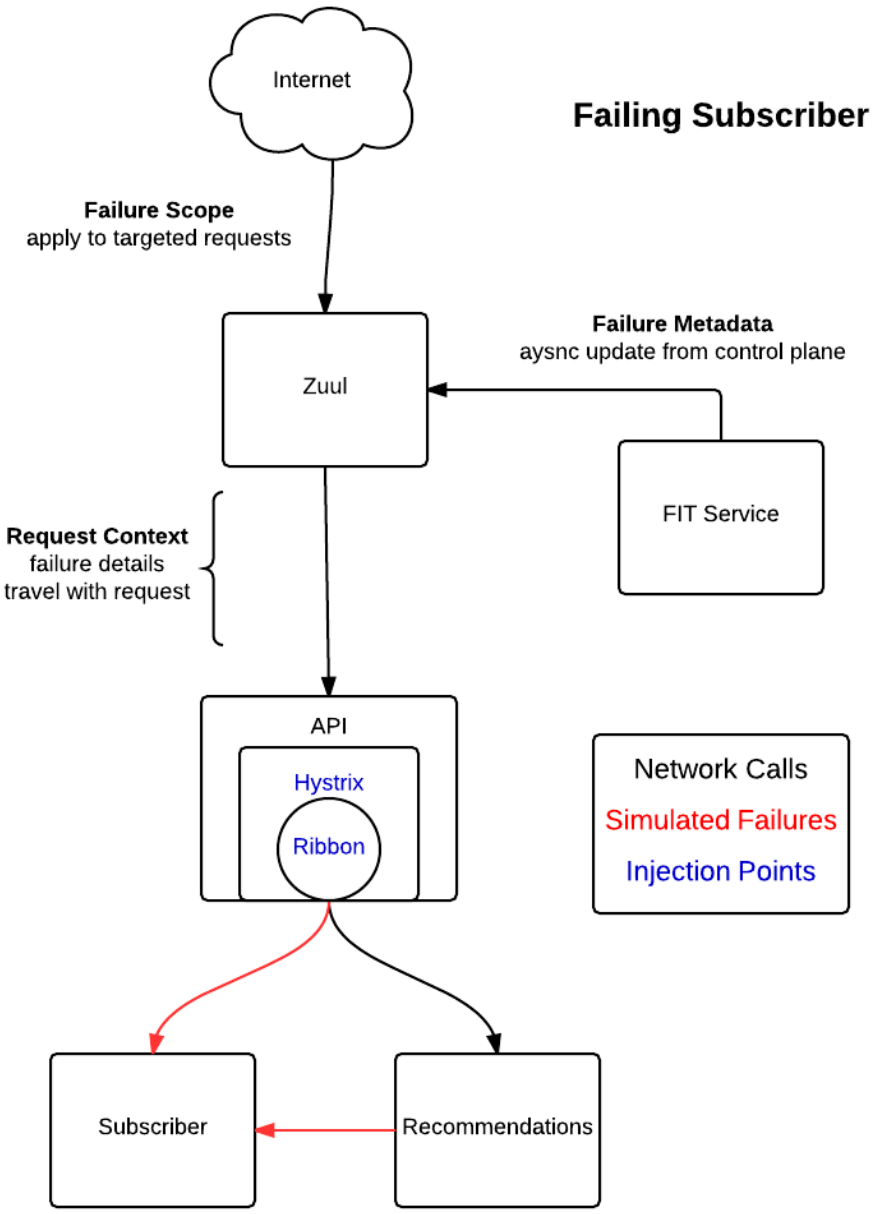


Figura 3.1: schema di FIT

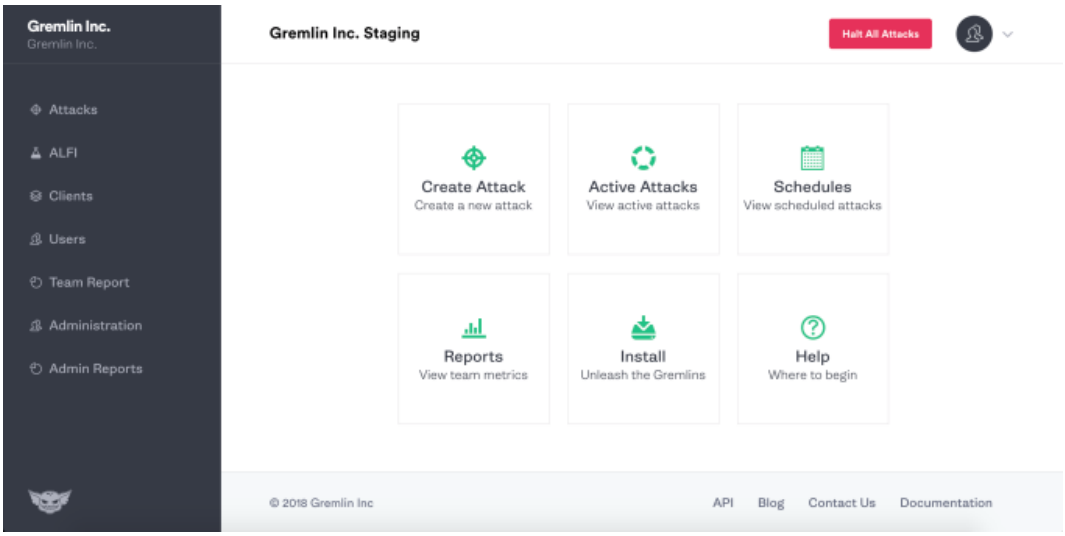


Figura 3.2: Gremlin

dei malfunzionamenti in FIT comincia con il servizi FIT che invia i dati riguardanti il malfunzionamento da simulare a Zuul. Zuul è un tool che permette di controllare e gestire il traffico in entrata, dopo aver ricevuto le istruzioni da FIT Zuul intrecetta le richieste in entrata e se queste rispettano i canoni ricevuti viene aggiunto l'errore. In seguito la richiesta corrotta viaggia fino all'applicazione, dove altri due servizi, Hystrix e Ribbon, aiutano a contenere l'errore e preparano delle azioni di emergenza da effettuare se le cose andassero male. Sempre questi ultimi due servizi si occupano infine di inserire l'errore desiderato nell'applicazione e le fanno processare la richiesta osservandone i risultati. Tutto questo complesso sistema rende più sicuro e automatizzabile effettuare esperimenti di Chaos Engineering, giacchè tutte le operazioni sono gestite da servizi autonomi tenendo conto di possibili situazioni impreviste. Tramite FIT infatti il team di sviluppo di Netfli ha sviluppato un sistema automatico che inserisce un certo tipo di malfunzionamenti e in segutio esegue l'applicazione Netflix per controllare lo stato del sistema e le operazioni che i clienti dovrebbero eseguire.

3.5.5 Gremlin

Gremlin è un tool che permette di effettuare esperimenti di Chaos Engineering in maniera semplice e veloce tramite un'interfaccia grafica intuititva, essendo un prodotto relativamente recente non ci sono ancora molti possibili eventi da introdurre. Gli eventi sono divisi in quattro diversi tipi:

Risorse

Stato modifica lo stato dell'ambiente in cui sta venendo eseguita l'applicazione

Rete simula errori casuali nella rete

Richieste modifica le singole richieste in arrivo

È compatibile con diverse piattaforme di deploy come AWS e Kubernetes, permette di schedulare esperimenti in determinati giorni e fasce orarie. Tuttavia è un prodotto

pensato per le imprese e le versioni di prova offrono pochi contenuti da esplorare, per questo dunque è stato scartato per l'utilizzo durante lo stage.

3.5.6 Chaos Blade

Chaos Blade è un tool piuttosto semplice da linea di comando per inserire in un sistema dei malfunzionamenti secondo i principi della Chaos Engineering, offre una serie di comandi CLI per preparare, creare ed eseguire esperimenti di Chaos Engineering di vario tipo, dalla riduzione della CPU al delay nelle richieste in entrata. $chaoscreatedubbodelay --time3000 --Servicexxx.xxx.Service$ Questo ad esempio inserisce un ritardo di 3 secondi nel servizio xxx.xxx.Service. Consente anche di salvare questi esperimenti creati per utilizzarli nuovamente senza reinserire il comando esatto, purtroppo però non permette di definire dei controlli iniziali e dei rollback da effettuare in caso di errori, per questo motivo questo tool non è stato utilizzato durante lo stage.

3.5.7 ChaosToolkit

ChaosToolkit si pone come uno strumento che standardizza i principi della Chaos Enginnering e si fregia di avere quattro caratteristiche principali:

Dichiarativo : permette di scrivere i propri esperimenti in maniera facile secondo una Open API

Estensibile : Chaos toolkit può essere esteso con un vasto numero di driver che consentono di utilizzare ChaosToolkit con tutte le piattaforme di deploy maggiormente diffuse

Automatizzabile : è facile con questo strumento automatizzare gli esperimenti e inserirli nel processo di CI/CD

Open Source : questo software è Open Source e viene continuamente aggiornato da una comunity molto viva di Chaos Engineers

Questo tool è stato scritto in Python è può essere installato tramite pip, nello stesso modo possono essere installate tutte le estensioni per le varie piattaforme di deploy. Gli esperimenti con questo tool vengono scritti in un file json secondo una specifica OpenAPI che illustrerò più avanti, poi è possibile eseguire questi esperimenti tramite linea di comando scrivendo `chaos run experiment.json` Un esperimento è costituito da delle ipotesi che definiscono se il sistema si trova nel suo stato normale prima dell'esecuzione dell'esperimento, a queste ipotesi si accompagnano delle prove per verificare che lo stato sia quello desirato e infine un metodo, costituito da una sequeza di prove o azioni che costituisce il corpo vero e prorio dell'esperimento, opzionalmente ci possono essere delle strategie di rollback. Un esperimento in ChaosToolkit è rappresentano da un oggetto json e deve necessariamente contenere:

* proprietà version
* proprietà title
* proprietà descrizione
* proprietà metodo

```
{
    "version": "1.0.0",
    "title": "System is resilient to pods failures",
    "description": "Can we still be available after a pod failure?",
```

Figura 3.3: Esperimento ChaosToolkit

```
"steady-state-hypothesis": {
    "title": "Services are all available and healthy",
    "probes": [
        {
            "type": "probe",
            "name": "all-services-are-healthy",
            "tolerance": true,
            "provider": {
                "type": "python",
                "module": "chaosk8s.probes",
                "func": "all_microservices_healthy",
                "arguments": {
                    "ns": "identity"
                }
            }
        },
```

Figura 3.4: Ipotesi dell'esperimento ChaosToolkit

I primi tre servono essenzialmente alle persone per riconoscere un esperimento dall'altro, l'ultimo invece è il corpo del esperimento e deve contenere almeno una azione o una prova. Un esperimento può e dovrebbe dichiarare per essere ben formato un'ipotesi dello stato normale del sistema e dei rollbacks da effettuare in caso di problemi durante lo svolgimento. L'ipotesi dello stato normale al suo interno è costituita da un insieme di probes che sono dei controlli da effettuare e che devono necessariamente essere rispettati per iniziare l'esperimento. Invece la proprietà method è costituita da una serie di azioni, pause e prove che determinano il contenuto vero e proprio dell'esperimento, un esempio potrebbe essere come in questo caso un'azione che termina un pod del deploy e una pausa subito dopo di 15 secondi per poter osservare come il sistema si comporta. In questo caso l'azione proviene da un'estensione di ChaosToolkit per Kubernetes che aggiunge nuove operazioni specifiche come appunto la terminazione dei pod.

Il file json contenente l'esperimento è suddiviso in varie sezioni ciascuna contenente elementi specifici: ChaosToolkit mette a disposizione diverse estensioni che aggiungono azioni e prove specifiche per una determinata piattaforma come AWS, Google Cloud e

```
"method": [
{
    "type": "action",
    "name": "terminate-pod",
    "provider": {
        "type": "python",
        "module": "chaosk8s.pod.actions",
        "func": "terminate_pods",
        "arguments": {
            "label_selector": "app=mico,tier=frontend",
            "rand": true,
            "ns":"identity"
        }
    },
    "pauses": {
        "after": 15
    }
```

Figura 3.5: Method dell'esperimento ChaosToolkit

Kubernetes, di seguito elenco le operazioni principali che l'estensione per Kubernetes mette a disposizione:

create_node aggiunge un nuovo nodo al cluster

delete_nodes termina i nodi lasciando un periodo di grazia

get_nodes fornisce la lisa di tutti i nodi attivi nel cluster

all_microservices_healthy controlla che tutti i microservizi del cluster siano attivi

microservice_is_not_available controlla che il microservizio selezionato non sia attivo

kill_microservice termina un microservizio

scale_microservice scala il deployment di un microservizio

count_pods conta il numero di pod secondo i parametri forniti

exec_in_pods esegue un comando nei pod selezionati

terminate_pods termina i pod selezionati con un periodo di grazia

ChaosToolkit inoltre offre la disponibilità all'integrazione con diversi strumenti di osservazione che permettono di osservare in maniera completa il sistema durante l'esperimento, di seguito elenco alcuni degli strumenti più interessanti:

* Humio
* Prometheus
* Open Tracing

Personalmente ho trovato Prometheus il più completo e facile da configurare.

Concludendo ChaosToolkit si pone come uno strumento per uniformare il modo di fare esperimenti di Chaos Engineering seguendo una struttura ben definita, rendendo gli esperimenti molto semplici da scrivere e garantendo compatibilità con diversi altri strumenti per estederne le sua funzionalità. Tutte queste caratteristiche uniche tra gli altri strumenti di Chaos Engineering lo rendono un irrinunciabile strumenti di base che funge da orchestratore di altri sistemi che ne estendono le funzionalità permettendo così da poter effettuare degli esperimenti completi.

Capitolo 4

Chaos Engineering nel progetto aziendale

Breve introduzione al capitolo
In questo capitolo tratto di come la teoria esposta nei capitoli precedenti sia stata applicata durante lo stage e dei compromessi che sono stati fatti tra la teoria e l'utilizzo pratico.

4.1 Progettazione di MICO2

Abbiamo deciso di analizzare il problema attraverso il Domain Driven Development per assicurarci di averlo compreso bene e al fine di ragionare sul come applicare la teoria studiata in queste prime settimane. Una suddivisione ad alto livello del dominio di MICO2 ha portato a definire 3 sotto-domini: Client Interaction Domain: si occupa di ricevere le richieste dal client e di comunicare col Database Interaction Domain e con l'External System Domain per processare la richiesta e ottenere la risposta da inviare al client che ha effettuato la richiesta; Database Interaction Domain: si occupa di ricevere la richiesta di lettura o scrittura di un dato complesso (per esempio un flow) e orchestrare tutte le richieste al DB e comporre il dato finale da restituire al chiamante, ovvero il Client Interaction Domain; External System Domain: si occupa di gestire richieste ai servizi esterni per verificare lo stato dei vari step di un flusso che poi restituirà al chiamante, ovvero il Client Interaction Domain. L'idea finale prevede quindi un attore che riceve la richiesta dal client e si occupa di generare un attore che prende in carico la richiesta, quest'ultimo interroga dunque il database e costruisce l'oggetto (ad esempio flow) da restituire all'attore padre che risponderà al chiamante.

La costruzione del flow necessita di informazioni contenute negli external system che vengono richieste a un attore che si occupa di ottenerle; dunque l'attore che costruisce il flow chiamerà l'attore degli external system.

Dopo aver ottenuto uno schema ad alto livello e uno più specifico del sistema abbiamo ragionato sui possibili rischi legati alla nostra architettura e li abbiamo inseriti in una tabella ottenendo il seguente risultato: Il punto che sicuramente ci è apparso più critico in primo luogo era la comunicazione con i servizi esterni e con il database, questi sistemi infatti essendo indipendenti da MICO2 potrebbero avere dei malfunzionamenti e causare un disservizio all'utente finale. Come soluzione a questi primi due problemi abbiamo pensato alla creazione di risposte positive del sistema che

COMPONENTE	PROBLEMA	PROBABILITÀ	CONTROMISURE
EXTERNAL SYSTEM	Il sistema potrebbe restituire un errore se gli external system non rispondono alle richieste in tempo	Alta	Implementare una risposta positiva del sistema anche i caso di problemi nella comunicazione con gli external system
DATABASE	Il sistema potrebbe restituire un errore se il database non risponde alle query in tempo	Alta	Implementare una risposta positiva del sistema anche i caso di problemi nella comunicazione con i database
MICO	Il sistema potrebbe non rispondere correttamente in caso di un aumento del traffico in condizioni di scarsa CPU disponibile	Bassa	
MICO	Il sistema potrebbe non rispondere correttamente in caso di un aumento del traffico	Bassa	
MICO	Il tempo di risposta del sistema potrebbe aumentare in caso di un aumento del traffico in condizioni di scarsa CPU disponibile	Alta	
MICO	Il tempo di risposta del sistema potrebbe aumentare in caso di un aumento del traffico	Alta	
Kubernetes	Il sistema potrebbe non rispondere correttamente alle richieste se venissero a mancare delle risorse come CPU o RAM da un'instanza dell'applicazione su Kubernetes	Alta	Implementare la scalabilità orizzontale per consentire di redistribuire il carico tra i pod del nodo
Kubernetes	Il sistema potrebbe non rispondere correttamente alle richieste se alcuni pod di Kubernetes dovessero avere dei malfunzionamenti	Media	Implementare un meccanismo per gestire il numero di richieste in entrata e eventualmente non rispondere ad alcune o ritardarne la risposta
Kubernetes	Il sistema potrebbe non rispondere correttamente alle richieste se un nodo di Kubernetes dovessero avere dei malfunzionamenti	Bassa	Implementare un meccanismo per gestire il numero di richieste in entrata e eventualmente non rispondere ad alcune o ritardarne la risposta
AWS	La zona di AWS in cui è stata deployata l'applicazione potrebbe non essere operativa causando un aumento di traffico nelle altre zone di deploy	Molto Bassa	

Figura 4.1: schema di FIT

notifichino la presenza di problemi ai sistemi esterni o al database se ve ne fossero. In secondo luogo abbiamo pensato ai problemi interni che MICO2 poteva avere, in particolare come l'applicazione si comporta in scarsità di risorse come CPU o con un aumento del traffico in entrata, in particolare temevamo che questo potesse causare l'ammanco di alcune risposte o un ritardo nella consegna delle stesse. In particolare l'aumento di latenza nelle risposte è il rischio che più facilmente si può verificare e dunque uno in cui abbiamo posto molta attenzione durante gli esperimenti. Per finire abbiamo pensato a che problemi si potessero verificare nel deploy dell'applicazione anticipando un pò i tempi, sia su come vengano gestite le risorse fisiche che i pod e i nodi di Kubernetes, aspetti che poi siamo andati a controllare con gli esperimenti. In conclusione come team di sviluppo abbiamo ritenuto molto utile questa pratica, ci ha permesso di individuare problemi a cui non avevamo pensato e in alcuni casi anche di trovare delle soluzioni applicabili durante lo sviluppo, risparmiando così tempo prezioso.

4.2 Compromessi della progettazione

Purtroppo durante l'esperienza di stage avevamo a che fare con un software di dimensioni ridotte per cui la lista di problemi individuati non si è rivelata essere molto lunga. Inoltre rispetto alla FMEA worksheet citata nel capitolo precedente quella da noi prodotta è una versione ridotta che tiene conto solamente della probabilità, tuttavia abbiamo ritenuto che questa versione fosse sufficiente per lo scopo dello stage. Concludendo la progettazione secondo la Chaos Engineering porta indubbiamente dei vantaggi per il team anche se non viene adottata in maniera completa.

4.3 Deploy dell'applicazione

Una volta terminato lo sviluppo di MICO2 ne abbiamo effettuato il deploy in un cluster Kubernetes all'interno di una repository AWS. Abbiamo effettuato il deploy di due applicazioni separate, una contentente l'applicazione MICO2 e un'altra con il database postgreSQL di MICO2; i deploy sono molto semplici e prevedono un servizio che esponga l'endpoint per ciascuna delle due applicazioni e, nel caso di MICO2 tre replicas dell'applicazione e invece una sola instanza del database. Con le replicas dell'applicazione puntiamo ad avere sempre il servizio disponibile anche nel caso in cui alcuni pod abbiano un malfunzionamento tramite il LoadBalancer che redirige il traffico tra i pod.

4.4 Test di Chaos Engineering

Con il deploy pronto abbiamo iniziato ad approcciarci agli esperimenti di Chaos Engineering, intanto abbiamo preso in mano la tabella stilata durante la progettazione e abbiamo iniziato a preparare l'ambiente e gli strumenti per effettuare i test. Tra gli strumenti descritti nel capitolo precedente abbiamo deciso di appoggiarci a ChaosToolkit per eseguire i nostri esperimenti, in particolare sfruttando le azioni comprese nel pacchetto ChaosToolkitK8s per inserire eventi nel cluster. Per simulare il traffico di produzione ci siamo avvalsi di uno strumento opensource chiamato go-wrk che è capace di generare un carico molto pesante di richieste anche se eseguito su una sola CPU, lo abbiamo usato per simulare un traffico superiore a quello di produzione durante gli esperimenti; il traffico di produzione massimo previsto dai tutor infatti era di circa 20 richieste al secondo, per i nostri test invece abbiamo utilizzato in media 40 richieste al secondo. Abbiamo interrogato sempre l'endpoint più oneroso per il sistema utilizzando il comando: `go-wrk -T 5000 -c 10 [mico2-address]/flows` che ci ha permesso di simulare con 10 goroutines che mandavano richieste in contemporanea per 5 secondi. Tenendo a mente che lo scopo principale della Chaos Engineering è aumentare la fiducia del team nel prodotto abbiamo deciso di iniziare con un esperimento semplice che prevedeva la terminazione di un pod dell'applicazione. Lo scopo era osservare come il tempo medio di risposta veniva influenzato da questo evento; l'esperimento ha prodotto un esito positivo in quanto il servizio è rimasto attivo e i tempi medi di risposta non sono cambiati in modo significativo. Dopo il primo esperimento con un blast radius molto basso abbiamo deciso di aumentarlo nei prossimi due esperimenti andando ad aumentare il numero di replicas a 10 e terminando rispettivamente 9 e 10 pod. Nel secondo esperimento appunto abbiamo terminato in modo molto analogo al primo 9 pod dal deploy simulando sempre un traffico superiore a quello di produzione, anche in questo caso il sistema è riuscito comunque a gestire con successo tutte le richieste inviate anche se il tempo di risposta è aumentato in modo significativo passando da circa 200 millisecondi a più di 1 secondo. Dopo aver discusso il risultato del secondo esperimento ci è parso chiaro che fosse necessario definire una policy per il riavvio dei pod in caso di errori o terminazioni, dopo averla definita se il deploy aveva un numero di pod attivi inferiore a quello definito questi venivano creati appena possibile; Con questo nuovo strumento abbiamo provato ad effettuare nuovamente il test e abbiamo osservato come questa volta il tempo medio di risposta aumetasse solo di 300 millisecondi dandoci maggiore fiducia nelle capacità del sistema. Con il terzo esperimento invece abbiamo voluto osare e provare a terminare in contemporanea tutti i pod del deploy per osservare quante delle richieste in arrivo avrebbero avuto problemi

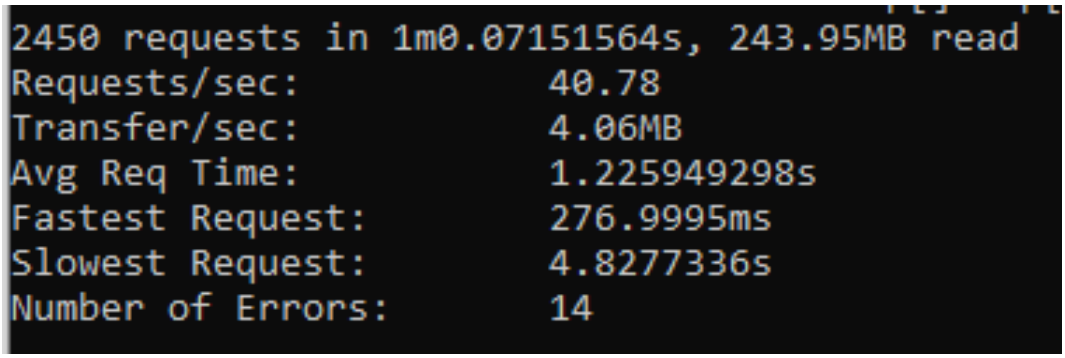


Figura 4.2: Report del tool go-wrk per il terzo esperimento

Probe - service-must-still-respond

Status	succeeded
Background	False
Started	Thu, 13 Aug 2020 07:35:20 GMT
Ended	Thu, 13 Aug 2020 07:35:20 GMT
Duration	0 seconds

Figura 4.3: report dell'esperimento sul database

e per quanto tempo il sistema non sarebbe stato in grado di rispondere alle richieste. Purtroppo stavolta il sistema non è rimasto attivo per tutta la durata dell'esperimento, il report finale di go-wrk infatti ha segnalato come 14 richieste non siano andate a buon fine: Dopo una breve discussione abbiamo comunque considerato il risultato dell'esperimento buono perchè comunque un malfunzionamento contemporaneo di tutti i pod ha to solo alla perdita di 14 richieste su 2450 per il minuto dell'esperimento. Con gli esperimenti successivi abbiamo deciso di cambiare focus e di spostarci sulla comunicazione col database, volevamo essere sicuri che il sistema reagisse correttamente ad un malfunzionamento al database o che restituisse un codice di errore appropriato. Per farlo abbiamo simulato un malfunzionamento al database andando a disattivare il servizio che esponeva l'endpoint del database, non riuscendo più a contattare il database il sistema avrebbe dovuto risponderci in maniera positiva facendoci notare l'errore o con un codice di errore specifico, per un malfunzionamento al database in particolare 503. L'esperimento in questo caso è fallito, poichè la prova che avevamo definito, ossia che il codice della risposta del server fosse 200 o 503 non è stata rispettata, è stato restituito invece un errore interno del server. Abbiamo dunque documentato l'esito dell'esperimento e discusso sulla necessità di predisporre una risposta per gestire in maniera resiliente il malfunzionamento del database e che per motivi di tempo abbiamo lasciato come to-do per chi prenderà in mano il progetto. Infine abbiamo deciso di esplorare il comportmento del sistema in scarsità di risorse, in particolare CPU e memoria RAM. Per simulare un utilizzo intenso di CPU abbiamo utilizzato uno

```
{
    "type": "probe",
    "name": "service-must-still-respond",
    "tolerance": [200,503],
    "provider": {
        "type": "http",
        "url":
    }
}
```

Figura 4.4: prova per controllare lo stato del database

```
9284 requests in 5m54.229827222s, 924.44MB read
Requests/sec:           26.21
Transfer/sec:           2.61MB
Avg Req Time:           1.907743576s
Fastest Request:        174.997ms
Slowest Request:        44.0517007s
Number of Errors:       446
```

Figura 4.5: Report del tool go-wrk per l'esperimento in cui simuliamo scarsità di RAM

strumento chiamato stress-ng che permette di testare un sistema nei suoi sottosistemi fisici in molti modi diversi, in particolare mette a disposizione 78 possibili test per la CPU e 20 per la memoria. Abbiamo scritto un esperimento che oltre a prevedere le prove iniziali includeva nel metodo due azioni, una che installava all'interno di ciascun pod il software e il secondo che eseguiva su bash il seguente comando: `stress-ng {matrix 0 {matrix-size 64 {tz -t 400{metrics-brief` che stressa la CPU con delle matrici di lato 64 da risolvere per 400 secondi, questo richiedeva più dell'80% della CPU dei pod. Per l'esperimento della RAM invece abbiamo utilizzato il seguente comando: `stress-ng --vm 1 --vm-bytes 75% --vm-method flip --verify -t 6m`

In entrambi questi esperimenti abbiamo constatato che il sistema non si è rivelato in grado di gestire tutte le richieste in entrata in quanto la CPU e la RAM in certi momenti raggiungevano livelli critici. A posteriori abbiamo quindi deciso di dichiarare un meccanismo di autoscaling variabile da 3 a 10 replicas e che per scalare richiede un consumo di CPU o RAM pari al 75% di quella massima.

4.5 Compromessi durante gli esperimenti

Il compromesso più grande che come team abbiamo dovuto fare rispetto alla teoria studiata è stato sicuramente il numero di esperimenti che abbiamo potuto fare, un pò per il tempo ristretto e un pò perchè il sistema che noi andavamo a testare non era particolarmente complesso e dunque aveva poche di quelle zone d'ombra che costituiscono il terreno d'indagine per la Chaos Engineering. A limitare leggermente i possibili esperimenti da effettuare sono stati anche i permessi che avevamo a disposizone, per motivi legati all'azienda non avevamo i permessi necessari per inserire eventi a livello dei nodi di Kubernetes o nel deploy di AWS. Nel complesso tuttavia siamo riusciti a rappresentare in una versione ridotta quello che ci aspetta da degli esperimenti

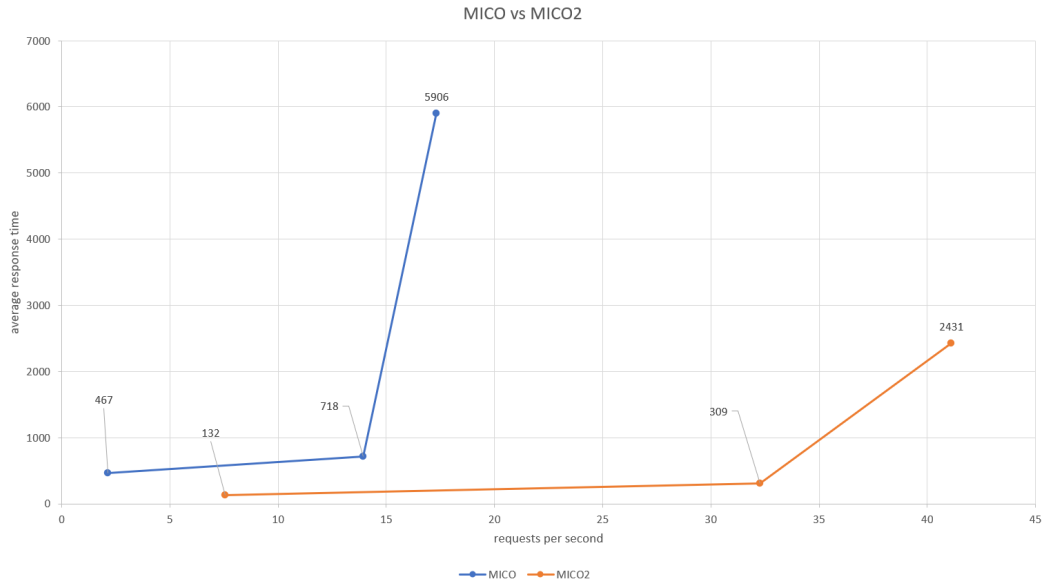


Figura 4.6: Grafico rappresentante il tempo medio di rispota rispetto alle richieste al secondo

di Chaos Engineering: la preparazione, la stesura di esperimenti con step ben definiti, l'esecuzione e la discussione dei risultati insieme al resto del team.

4.6 Valutazioni sul prodotto finito e spunti per il futuro

A conclusione degli esperimenti di Chaos Engineering e sempre nell'ottica di aumentare la fiducia nel prodotto abbiamo deciso di confrontare le performance sotto carico della nostra versione del software con quella precedente. Per simulare il carico abbiamo utilizzato lo stesso strumento usato negli esperimenti di Chaos Engineering variando il numero di goroutines che inviavano richieste per osservare le perfomarce all'aumentare di queste ultime. Dopo aver effettuato le prove abbiamo raggruppato i dati in dei grafici secondo il tempo medio di risposta all'aumentare delle richieste al secondo e all'aumentare delle goroutines che inviavano le richieste. Da entrambi questi grafici possiamo vincere come l'applicazione e il deploy realizzati abbiano delle performance superiori rispetto alla precedente versione sia con poche richieste e questo divario aumenta ulteriormente quando aumentano le richieste e le goroutines. Come detto nel capitolo precedente abbiamo redatto un documento contenente tutti i report ottenuti da ChaosToolkit e le riflessioni del team su ogni esperimento, molti degli esperimenti hanno evidenziato la forza e la resilienza del sistema a situazioni impreviste mentre altri hanno identificato delle criticità, alcune delle quali per motivi di tempo abbiamo lasciato documentate ma non risolte.

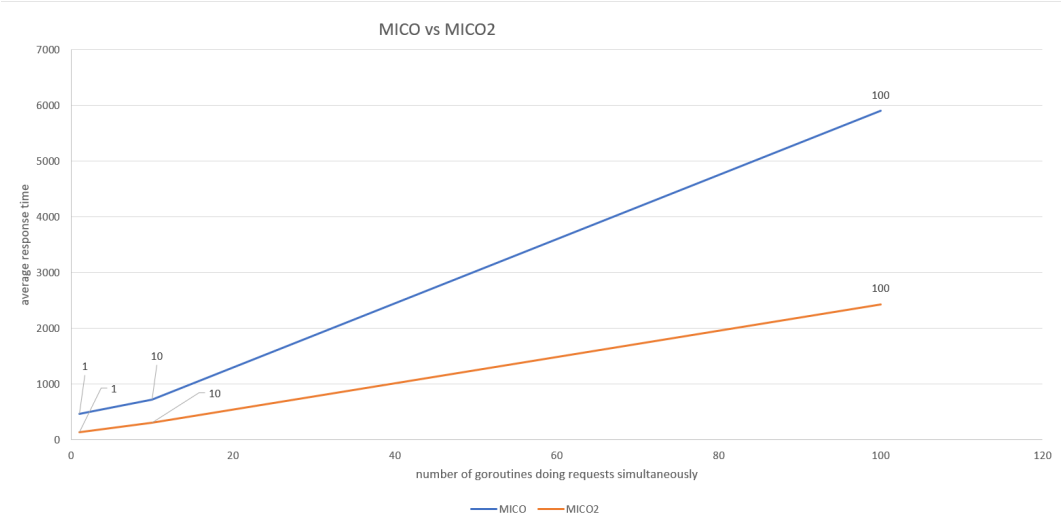


Figura 4.7: Grafico rappresentante il tempo medio di rispota rispetto al numero di goroutines

Capitolo 5

Progettazione e codifica

Breve introduzione al capitolo

5.1 Tecnologie e strumenti

Di seguito viene data una panoramica delle tecnologie e strumenti utilizzati.

Tecnologia 1

Descrizione Tecnologia 1.

Tecnologia 2

Descrizione Tecnologia 2

5.2 Ciclo di vita del software

5.3 Progettazione

Namespace 1

Descrizione namespace 1.

Classe 1: Descrizione classe 1

Classe 2: Descrizione classe 2

5.4 Design Pattern utilizzati

5.5 Codifica

Capitolo 6

Verifica e validazione

Capitolo 7

Conclusioni

7.1 Consuntivo finale

7.2 Raggiungimento degli obiettivi

7.3 Conoscenze acquisite

7.4 Valutazione personale

Appendice A

Appendice A

Citazione

Autore della citazione

Bibliografia