UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea in Advanced Computer Programming

Valutazione di un' architettura 5G container-based in condizioni di guasti software di persistenza

Anno Accademico 2023/2024

Candidato
Hamza En Nakhly
matr. N46005845

Per i miei genitori En Nakhly Hamid , Essoudi Amina

Abstract

Questa tesi esamina come risponde un'architettura 5G container-based in presenza di guasti legati alla persistenza dei dati. Attraverso l'analisi di un'implementazione open-source nota come Open5GS, si valuta il comportamento della rete in un ambiente non produttivo soggetto a tali guasti. Il focus principale si posiziona sulle tecnologie fondamentali per il deployment di una rete 5G Stand-Alone (SA), includendo un approfondimento sul progetto Open5GS. Mediante l'applicazione dell'ingegneria del caos e tecniche di fault injection, la tesi mira a iniettare guasti nel componente MongoDB della rete 5G SA, che gestisce i servizi di persistenza dei dati. Attraverso esperimenti mirati, viene valutato l'impatto dei guasti su vari aspetti della rete, come il fallimento dei pod .L'obiettivo è identificare vulnerabilità e proporre soluzioni per migliorare la robustezza e l'affidabilità delle future architetture container-based di rete 5G

Contents

Chapter 1

Background

1.1 5G e le tecnologie per abilitarla

L'evoluzione delle reti di comunicazione attraverso le generazioni tecnologiche è un fenomeno di grande rilevanza nel settore delle telecomunicazioni. Dalla prima generazione (1G) alla quinta generazione (5G), le reti cellulari hanno subito trasformazioni significative per soddisfare le crescenti esigenze degli utenti e sfruttare le innovazioni tecnologiche emergenti.

Il passaggio dal 1G al 2G ha visto la digitalizzazione delle reti vocali mobili, mentre il salto successi vo al 3G ha permesso l'accesso a servizi dati avanzati e a Internet ad alta velocità tramite dispositivi mobili. Il 4G ha rivoluzionato le reti mobili, trasformandole in piattaforme di banda larga senza fili e aprendo la strada a una vasta gamma di applicazioni e servizi basati sulla connettività mobile ad alta velocità.

Oggi, il 5G rappresenta la prossima frontiera nell'evoluzione delle reti di comunicazione, promettendo maggiore velocità, larghezza di banda, latenza ridotta e maggiore affidabilità. Tuttavia, gli operatori di rete devono affrontare sfide significative nell'espandere e aggiornare le loro infrastrutture, mantenendo sotto controllo i costi e garantendo una redditività sostenibile in un mercato altamente competitivo.

La visione delle reti 5G si concentra sul concetto di *"slicing di rete"*, che rappresenta una componente essenziale. Lo *slicing* consente agli operatori di rete di suddividere la loro infrastruttura in reti logiche, ciascuna dedicata a specifici requisiti di servizio e temporaneamente assegnata ai *tenant*. Questi *tenant*, chiamati anche "verticali", controllano diversi livelli dell'infrastruttura e integrano così verticalmente il sistema 5G.[14]

Lo *slicing* permette di offrire una varietà di servizi su un'unica infrastruttura, migliorando l'efficienza e aprendo nuove opportunità di mercato. La standardizzazione dello *slicing* è al centro dell'attenzione di organismi come il ***3GPP*** e la ***NGMN***. Il concetto coinvolge tre livelli: istanza di servizio, istanza di slice di rete e livello delle risorse.

La gestione delle reti 5G richiede tecnologie come *NFV* e *SDN*, che consentono flessibilità e programmabilità. Tuttavia, ci sono sfide da affrontare per rendere queste tecnologie *carrier-grade*[12], motivo per cui sono in corso sforzi di standardizzazione da parte di organizzazioni come *l'ONF e l'ETSI*.

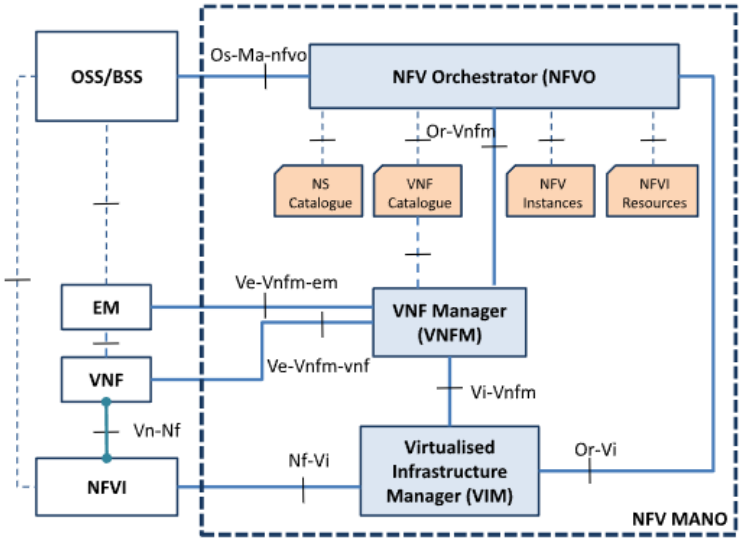
In sintesi, lo *slicing* di rete rappresenta un pilastro fondamentale delle reti 5G, consentendo la distribuzione di servizi diversificati su un'infrastruttura comune, con *NFV e SDN* che giocano ruoli chiave nell'abilitare questa architettura avanzata [14]

1.1.1 Network Functions Virtualization (NFV)

NFV mira a trasformare le funzioni di rete tradizionalmente legate a hardware specifici in applicazioni software che possono essere eseguite su server standard, riducendo i costi e migliorando l'efficienza operativa. **L'ETSI** (European Telecommunications Standards Institute) ha definito un framework di riferimento per **NFV**, composto da tre blocchi funzionali principali: il Virtualized Infrastructure Manager (**VIM**), il Virtual Network Function Manager (**VNFM**) e l'NFV Orchestrator (**NFVO**). **VIM** gestisce le risorse infrastrutturali virtualizzate, come server e storage.1.1.1 **VNFM** si occupa del ciclo di vita delle funzioni di rete virtualizzate (**VNF**), inclusi il deployment, la configurazione e la terminazione. **NFVO** coordina le risorse tra **VIM e VNFM**, assicurando l'integrità del servizio end-to-end. [11]

1.1.2 Software-Defined Networking (SDN)

SDN separa il piano di controllo dal piano di dati nelle apparecchiature di rete, consentendo una gestione centralizzata e programmabile della rete. Questo approccio offre una flessibilità senza precedenti nella



configurazione della rete, ottimizzazione del traffico e gestione della sicurezza. Protocolli come OpenFlow consentono la comunicazione tra il controller SDN e i dispositivi di rete, facilitando l'implementazione di politiche di rete dinamiche. [13]

1.1.3 Sinergia tra NFV e SDN

L'integrazione di NFV e SDN offre una soluzione potente per il design e la gestione delle reti 5G. Mentre NFV fornisce l'agilità necessaria per implementare e scalare rapidamente le funzioni di rete come servizi software, SDN offre il controllo centralizzato e la programmabilità per gestire il traffico di rete in modo dinamico. Questa combinazione permette agli operatori di rete di rispondere in modo più efficace alle esigenze in continua evoluzione dei servizi di rete, ottimizzando le risorse e migliorando l'esperienza dell'utente finale.

Nonostante i numerosi vantaggi, l'adozione di NFV e SDN presenta

sfide, inclusa la necessità di standardizzazione, preoccupazioni per la sicurezza e la complessità della gestione di ambienti di rete altamente dinamici e virtualizzati. Tuttavia, con il continuo sviluppo di nuove tecnologie e la collaborazione tra enti di standardizzazione, fornitori di tecnologia e operatori di rete, NFV e SDN sono destinati a svolgere un ruolo cruciale nell'abilitare le reti 5G a soddisfare le promesse di una connettività senza precedenti.[14] [14]

1.2 Container

La containerizzazione è il processo di imballaggio del codice software insieme soltanto alle librerie e alle dipendenze del sistema operativo (OS) necessarie per eseguire il codice, al fine di creare un singolo eseguibile leggero, chiamato "container", che funziona in modo coerente su qualsiasi infrastruttura. Più efficienti ed portabili in termini di risorse rispetto alle virtual machine (VM), i container sono diventati le unità di calcolo più utlizzate delle applicazioni moderne orientate al cloud. La containerizzazione permette agli sviluppatori di creare e distribuire applicazioni più rapidamente e in modo più sicuro. Con i metodi tradizionali, il codice viene sviluppato in un ambiente di calcolo specifico che, quando trasferito in una nuova posizione, spesso risulta in bug ed errori. Ad esempio, quando uno sviluppatore trasferisce il codice scritto da un computer desktop a una VM o da un sistema operativo Linux a Windows. La containerizzazione elimina questo

problema raggruppando insieme il codice dell'applicazione con i relativi file di configurazione, librerie e dipendenze necessarie per la sua esecuzione , non esiste che "non funziona sul mio computer!". Questo pacchetto unico di software, o "container", è astratto dal os host e, quindi, diventa autonomo e portatile, in grado di funzionare su qualsiasi piattaforma o cloud, libero da problemi. [8]

Il concetto di containerizzazione e isolamento dei processi è in realtà vecchio di decenni, ma l'emergere nel 2013 di Docker Engine, un progetto open-source diventato lo standard industriale per i container con strumenti di sviluppo semplici e un approccio universale all'imballaggio, ha accelerato l'adozione di questa tecnologia. Oggi, le organizzazioni utilizzano sempre più la containerizzazione per creare nuove applicazioni e per modernizzare le applicazioni esistenti per il cloud. In un sondaggio IBM, il 61% degli utilizzatori di container ha riferito di usare i container in più del 50% delle nuove applicazioni costruite nei due anni precedenti; il 64% degli adottanti si aspetta di inserire il 50% o più delle loro applicazioni esistenti in container nei prossimi due anni.

I container sono spesso descritti come "leggeri", nel senso che condividono il kernel del sistema operativo della macchina e non richiedono il sovraccarico di associare un sistema operativo con ogni applicazione. I container sono intrinsecamente più piccoli in capacità rispetto a una VM e richiedono meno tempo di avvio, consentendo di eseguire molti

più container sulla stessa capacità di calcolo di una singola VM. Questo porta a un'efficienza server maggiore e, di conseguenza, riduce i costi dei server e delle licenze.

Forse l'aspetto più importante è che la containerizzazione permette alle applicazioni di essere "scritte una volta e eseguite ovunque". Questa portabilità accelera lo sviluppo, previene il blocco da parte dei fornitori di servizi cloud e offre altri benefici notevoli come l'isolamento dei guasti, la facilità di gestione, la semplificazione della sicurezza ed altro . [8]

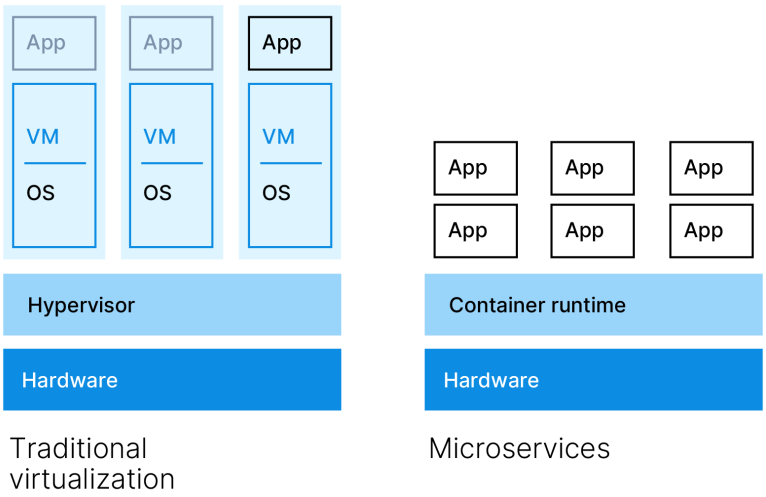


Figure 1.1: Tipologie diverese di virtualizzazione

1.3 Kubernetes

Kubernetes è diventato uno strumento fondamentale nella gestione delle infrastrutture cloud-native, offrendo una piattaforma robusta per l'orchestrazione di container. Nel contesto della Network Function

Virtualization (NFV) e del suo framework di gestione e orchestrazione (MANO), Kubernetes può svolgere un ruolo cruciale nell'automazione e nell'efficienza delle operazioni di rete.

Kubernetes, originariamente sviluppato da Google e ora gestito dalla Cloud Native Computing Foundation (CNCF), fornisce un sistema per l'automazione del deployment, del scaling e delle operazioni dei container di applicazioni su un cluster di host. Questo lo rende particolarmente adatto per gestire le applicazioni containerizzate all'interno dell'ecosistema NFV.

1.3.1 archiettatua di Kubernetes

L'integrazione di Kubernetes nell'architettura NFV MANO si basa sulla sua capacità di gestire container su larga scala, offrendo un'orchestrazione dinamica e flessibile delle risorse. L'architettura di Kubernetes può essere descritta attraverso i seguenti componenti chiave [4]:

- Master Components (Componenti del Nodo Master) I componenti del master controllano il cluster e prendono decisioni globali sul cluster (ad esempio, scheduling), oltre a rilevare e rispondere agli eventi del cluster (come ad esempio il lancio di un nuovo container se il conteggio delle repliche non corrisponde al desiderato).

- – **API Server (kube-apiserver)**: Funge da front-end per il control plane di Kubernetes. Tutte le operazioni e le co-

municazioni tra i componenti del cluster passano attraverso l'API server.

  - **Etcd**: Un database chiave-valore leggero e distribuito che Kubernetes utilizza per la persistenza e la conservazione di tutti i dati del cluster, agendo come la sorgente di verità per lo stato del cluster.
  - **Scheduler (kube-scheduler)**: Seleziona su quale nodo eseguire i container non ancora in esecuzione, basandosi su vari fattori come risorse disponibili e vincoli di affinità/antiaffinità.
  - **Controller Manager (kube-controller-manager)**: Esegue i processi di controllo di background. Ogni controller è un ciclo di controllo che osserva lo stato condiviso del cluster attraverso l'API server e apporta modifiche tentando di spostare lo stato attuale verso lo stato desiderato.
  - **Cloud Controller Manager (cloud-controller-manager)**: Gestisce i controller che interagiscono con i provider di cloud sottostanti per integrare le infrastrutture cloud con il cluster Kubernetes.

- Node Components (Componenti del Nodo worker) I componenti del nodo lavorano su ogni nodo, mantenendo i container in esecuzione e fornendo l'ambiente di runtime Kubernetes.

- **Kubelet**: Un agente che si esegue su ogni nodo nel cluster. Assicura che i container siano in esecuzione in un Pod.
- **Kube-proxy**: Mantiene le regole di networking sui nodi. Queste regole permettono la comunicazione ai tuoi container all'interno del cluster o dall'esterno.
- **Container Runtime**: Il software responsabile dell'esecuzione dei container. Kubernetes supporta vari runtime come Docker, containerd, CRI-O, e altri compatibili con l'interfaccia di runtime dei container (CRI) Ex: Docker Engine [4].

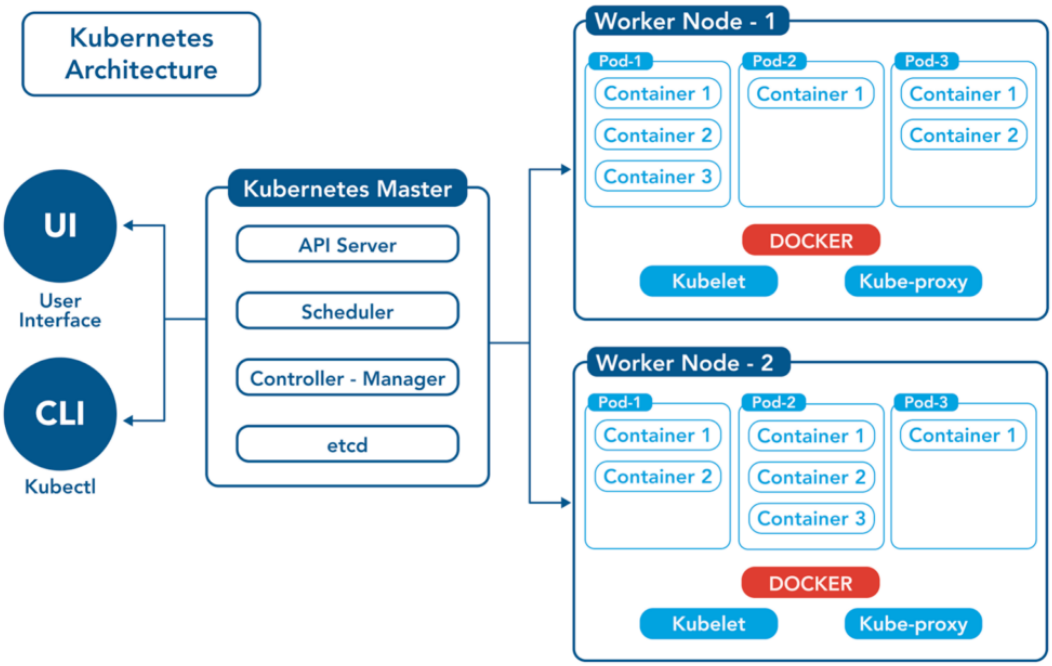


Figure 1.2: Archiettatura di Kubernetes

1.3.2 Vantaggi dell'uso di Kubernetes in NFV MANO

L'integrazione di Kubernetes nell'ecosistema NFV MANO offre numerosi vantaggi:

- **Scalabilità**: Kubernetes facilita il scaling orizzontale delle applicazioni, consentendo alle VNF di rispondere dinamicamente alle variazioni del carico di lavoro.
- **Affidabilità**: La gestione dei failover, la riparazione automatica e il deployment senza interruzioni delle applicazioni migliorano l'affidabilità delle VNF.
- **Automazione**: L'automazione del deployment, del scaling e delle operazioni riduce la necessità di intervento manuale, aumentando l'efficienza operativa.
- **Portabilità**: La containerizzazione consente alle VNF di essere eseguite in modo uniforme su diverse infrastrutture cloud, sia pubbliche che private.[3]

1.3.3 Considerazioni

Nonostante i suoi vantaggi, l'integrazione di Kubernetes in NFV MANO presenta alcune sfide:

- Complessità: L'architettura di Kubernetes può essere complessa da comprendere e gestire, richiedendo competenze specifiche.

- Sicurezza: La gestione della sicurezza in un ambiente Kubernetes distribuito richiede attenzione particolare, specialmente in contesti di rete sensibili.

- Risorse: Kubernetes può essere esigente in termini di risorse, richiedendo una pianificazione attenta per garantire che le risorse di calcolo, rete e storage siano ottimizzate.[3]

1.4 Migrazione verso sistemi Container Based

Nell'ambito dei servizi digitali, l'adozione di strategie basate sul cloud nativo, che impiegano infrastrutture sia centralizzate che distribuite, si traduce in un netto aumento della flessibilità, della scalabilità, dell'affidabilità e della portabilità delle applicazioni. La transizione dalla virtualizzazione tradizionale verso un'infrastruttura completamente orientata al cloud nativo rappresenta un miglioramento significativo, permettendo una maggiore efficienza e agilità. Questo, a sua volta, facilita una rapida implementazione di prodotti innovativi, in risposta alle esigenze di mercato e alle aspettative dei consumatori.

1.4.1 CNF

Uno degli aspetti fondamentali dell'approccio cloud nativo è l'adozione dei container in luogo delle macchine virtuali (VM). Le funzioni di rete cloud native (CNF) rappresentano un'evoluzione delle funzioni di

virtualizzazione di rete (VNF) e sono progettate per operare all'interno dei container. La containerizzazione dei componenti dell'infrastruttura di rete favorisce l'esecuzione di un'ampia varietà di servizi all'interno di un unico cluster, semplificando l'integrazione di applicazioni modulari e permettendo una gestione dinamica del traffico di rete verso i pod corretti. [10]

Tuttavia, l'impiego delle CNF non si ferma semplicemente alla containerizzazione. Per massimizzare i benefici dell'architettura cloud native, è necessaria una ristrutturazione profonda del software di rete, che può includere la sua decomposizione in microservizi. Questo permette non solo l'utilizzo parallelo di diverse versioni del software durante gli aggiornamenti, ma anche l'adozione di servizi di piattaforma integrati, come soluzioni di bilanciamento del carico o sistemi di gestione dati universali.

Con l'incremento dell'utilizzo degli ambienti cloud native, è fondamentale che le CNF possano operare in concomitanza con le preesistenti VNF durante il periodo di transizione. Per rispondere efficacemente a una domanda sempre più esigente, velocizzare i processi di deployment e minimizzare la complessità, i fornitori di servizi digitali necessitano di un'automazione completa. Questo comprende lo sviluppo, il deployment, la manutenzione e la gestione della rete, sottolineando l'importanza di adottare configurazioni standardizzate, strumenti sviluppati all'interno delle comunità open source, procedure di test rig-

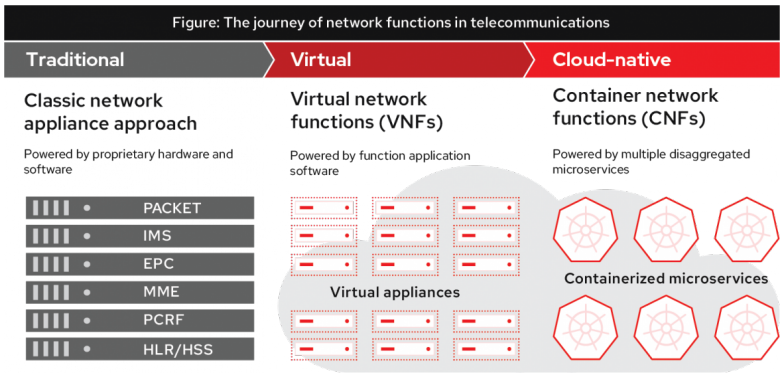


Figure 1.3: Evoluzione delle funzioni di rete nel tempo

orose e processi di certificazione.[10]

Chapter 2

Open5Gs

Open5GS è un progetto open source che fornisce un'implementazione software di una core network 5G e EPC (Evolved Packet Core) completamente funzionale.[5]

L'obiettivo del progetto è di supportare le funzionalità della rete 5G per consentire la comunicazione e la gestione dei dati in reti mobili di quinta generazione. Open5GS segue gli standard definiti dal 3GPP (3rd Generation Partnership Project), l'organizzazione che sviluppa protocolli per la telefonia mobile.

Uno dei principali vantaggi di Open5GS è la sua natura open-source, che favorisce la collaborazione e l'innovazione all'interno della comunità delle telecomunicazioni. Gli utenti possono contribuire al progetto, personalizzare il software secondo le proprie necessità e condividere le proprie modifiche con altri.

Questo approccio collaborativo accelera lo sviluppo di nuove tecnolo-

gie e applicazioni basate sulla rete 5G, promuovendo allo stesso tempo l'interoperabilità e la standardizzazione.

2.1 4G/ 5G NSA Core

Il Core NSA di Open5GS utilizza **Evolved Packet Core (EPC)**, l'architettura di core network utilizzata nelle reti 4G LTE abilitata per supportare accessi da parte di gNodeB , senza però la possibilità di utilizzare nuove funzionalità esclusive di un 5G Core SA ,come ad esempio *il network slicing* .

2.1.1 Componenti del Core NSA

Il Core Open5GS è strutturato in due piani principali, il piano di controllo e il piano utente, separati fisicamente per ottimizzare le prestazioni e la gestione della rete.

2.1.2 Piano di Controllo

Il piano di controllo include i seguenti componenti:

- **MME (Mobility Management Entity):** Gestisce la mobilità degli utenti, le sessioni, il paging e i bearers.
- **HSS (Home Subscriber Server):** Conserva i dati degli abbonati, inclusi il profilo utente e i vettori di autenticazione SIM.

- **PCRF (Policy and Charging Rules Function):** Stabilisce le regole di policy e di tariffazione.
- **SGWC (Serving Gateway Control Plane):** Coordina il traffico dati e le informazioni di controllo.
- **PGWC/SMF (Packet Gateway Control Plane):** Gestisce le sessioni di dati e il loro routing.

2.1.3 Piano Utente

Il piano utente gestisce il trasporto effettivo dei dati utente e include:

- **SGWU (Serving Gateway User Plane):** Trasporta i dati utente.
- **PGWU/UPF (Packet Gateway User Plane):** Forwarding dei pacchetti dati verso la rete esterna.

2.2 Separazione dei Piani

La separazione dei piani di controllo e utente (CUPS) in Open5GS migliora la scalabilità e la flessibilità della rete, permettendo distribuzioni ottimizzate per scenari di edge computing mobile (MEC)(si possono avere più piani utente).

Ogni componente di Open5GS è configurato tramite file di configurazione che includono gli indirizzi IP, i nomi delle interfacce locali e le

informazioni di connessione necessarie per l'integrazione nell'ecosistema di rete. [5]

2.3 5G SA Core

Il 5G Standalone (SA) rappresenta l'architettura di rete completamente nuova e indipendente del 5G, che non si basa sulle infrastrutture di rete 4G esistenti, come fa invece il 5G Non-Standalone (NSA). Il 5G SA è progettato per sfruttare appieno le capacità avanzate del 5G, inclusa l'alta velocità, la bassa latenza, e l'ampia connettività per supportare una vasta gamma di applicazioni e servizi, dall'Internet delle cose (IoT) alle comunicazioni ultra-affidabili a bassa latenza (URLLC), fino al broadband mobile esteso (eMBB) e al network slicing

Il Core Open5GS 5G SA introduce un'architettura di rete basata sui servizi (SBA), diversa dal core 4G, ottimizzata per le reti di quinta generazione con funzioni specifiche per gestire connessioni, mobilità, autenticazione, e molto altro. L'architettura SBA del Core 5G SA facilita una maggiore modularità e flessibilità. Questo approccio semplifica la gestione della rete e supporta l'innovazione e l'integrazione dei servizi.

2.3.1 Funzioni del Core 5G SA

Il Core 5G SA contiene le seguenti funzioni principali:

- **AMF (Funzione di Gestione dell'Accesso e della Mobilità)**: Gestisce l'accesso alla rete, autenticando in modo sicuro i clienti che desiderano accedere alla rete 5G, e la mobilità dell'utente, essendo responsabile della procedura di trasferimento del traffico per mantenere la continuità della connessione. L'AMF si occupa anche di mantenere lo stato di registrazione dell'abbonato durante il suo spostamento. Nella rete 4G, l'equivalente dell'AMF è l'MME (Entità di Gestione della Mobilità).

- **SMF (Funzione di Gestione della Sessione)**: Gestisce la sessione tra l'utente e la rete 5G. Controlla il traffico nella rete 5G secondo la politica di QoS (Qualità del Servizio) implementata, dando priorità al traffico di rete, controllando la larghezza di banda e gestendo il carico di rete.

- **UPF (Funzione del Piano Utente)**: Facilita la trasmissione dei pacchetti dati dell'utente tra gNodeB e la rete. Assicura un trasferimento dei dati efficace in conformità con i requisiti di qualità del servizio (QoS) e le politiche di rete, la gestione del traffico e la sicurezza. Interagisce con l'SMF per tracciare e mantenere la sessione utente e può essere posizionato geograficamente in qualsiasi luogo per migliorare le prestazioni della rete e la latenza.

- **AUSF (Funzione del Server di Autenticazione)**: Autentica gli utenti nella rete 5G, verifica l'identità degli utenti e controlla la loro autorizzazione all'uso della rete, comunicando con l'UDM dove sono memorizzate le informazioni sugli abbonati.

- **NRF (Funzione di Repository delle Funzioni di Rete)**: Memorizza informazioni su tutte le Funzioni di Rete (NF) disponibili, consentendo una gestione dinamica e la scoperta di elementi di rete e funzioni.

- **SCP (Proxy di Comunicazione del Servizio)**: Gestisce la comunicazione tra le Funzioni di Rete (NF) e le applicazioni o i servizi esterni, facilitando l'integrazione di vari componenti del sistema 5G.

- **NSSF (Funzione di Selezione della Fetta di Rete)**: Consente di selezionare la fetta di rete appropriata per una data richiesta di servizio, lavorando con altre funzioni di rete per garantire coerenza e conformità nella gestione delle fette di rete.

- **UDM (Gestione Unificata dei Dati)**: Gestisce i dati degli abbonati nella rete 5G, fornendo all'SMF e all'AMF le informazioni necessarie per stabilire, gestire e controllare le sessioni degli abbonati e per l'autenticazione degli utenti.

- **UDR (Repository Unificato dei Dati)**: Funge da database di backend utilizzato da nodi come l'UDM o il PCF, fornendo la

centralizzazione dello storage dei dati per tali nodi.

- **PCF (Funzione di Politica e Tariffazione)**: Mantiene la politica di sessione e prende decisioni riguardo all'allocazione delle risorse di rete per garantire una qualità ottimale del servizio. Opera in base a politiche che possono includere la priorizzazione del traffico e la gestione della larghezza di banda.

- **BSF (Funzione di Supporto al Binding)**: Coopera con il PCF con cui scambia informazioni riguardo a quale abbonato sta utilizzando (poiché sono normalmente situati in pool). [6]

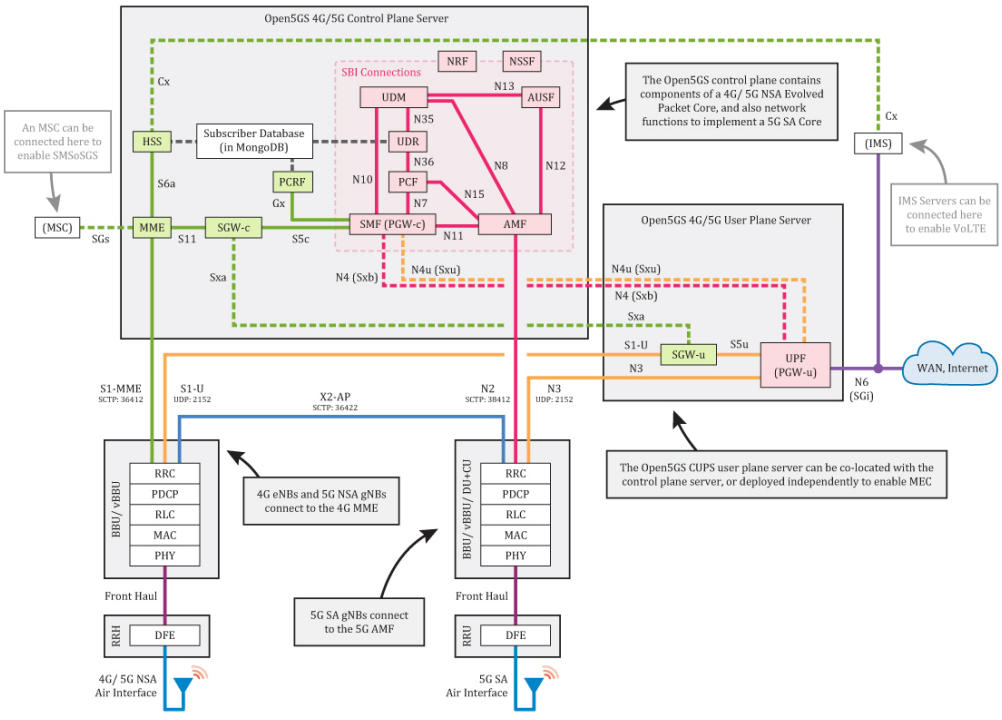


Figure 2.1: Architettatura di Open5GS

Chapter 3

Chaos Engineering

3.1 Introduzione all'Ingegneria del Caos

Con l'avanzamento delle *microservizi* e delle architetture *cloud* distribuite, il web è diventato sempre più complesso. Dipendiamo da questi sistemi più che mai, tuttavia, prevedere i fallimenti è diventato molto più difficile.

Questi fallimenti provocano interruzioni costose per le aziende, danneggiando i clienti che cercano di fare acquisti, concludere affari e svolgere il proprio lavoro. Anche brevi interruzioni possono influire sui risultati finanziari di un'azienda, quindi il costo del tempo di inattività sta diventando un indicatore chiave di prestazione (**KPI**) per molti team di ingegneria. Ad esempio, nel 2017, il 98% delle organizzazioni ha dichiarato che un'ora di inattività avrebbe costato alla loro azienda oltre 100.000 dollari. Un'unica interruzione può costare a un'azienda

milioni di dollari. Il CEO di British Airways ha recentemente spiegato come un fallimento che ha bloccato decine di migliaia di passeggeri della BA a maggio 2017 è costato alla compagnia 80 milioni di sterline (102,19 milioni di dollari USA).

Le aziende necessitano di una soluzione a questa sfida: attendere il prossimo costoso fermo non è un'opzione. Per affrontare direttamente la sfida, sempre più aziende si stanno rivolgendo all'Ingegneria del Caos. [7]

3.2 Breve storia sull ' Ingegneria del Caos

La pratica dell'ingegneria del caos è stata originata da *Netflix* intorno al 2008, dopo che avevano formalmente lanciato il loro servizio di streaming. A seguito di un problema di corruzione del database intorno al 2011, Netflix pianificò di trasferire il proprio datacenter sul cloud tramite **AWS** (Amazon Web Services). Infatti, impiegarono otto anni per completare definitivamente la migrazione. Nel 2015, **AWS** subì un'interruzione di servizio, che causò l'inoperatività di Netflix per diverse ore. Questi erano i primi giorni del **cloud computing**, quindi non era robusto, stabile e sicuro come lo è oggi. Quando scoprirono che il passaggio al *cloud* non aveva creato alcuni dei benefici che si aspettavano, come la scalabilità, *l'uptime*, l'evitare punti singoli di fallimento, *l'autoscaling*, ecc., decisero che avevano bisogno di un modo per testare queste problematiche inaspettate per garantire che i loro

servizi fossero operativi e, in ultima analisi, evitare impatti negativi sugli utenti e frustrazioni. Da questa esperienza nacque **l'ingegneria del caos**.[9]

3.3 Differenza tra Ingegnera del Caos e Performance Testing

L'*ingegneria del caos* e il *testing* delle prestazioni, come i test di stress o di carico, assistono i team nell'individuazione di punti di rottura o malfunzionamenti creando ambienti anormali o instabili. Tuttavia, una delle principali distinzioni tra l'*ingegneria del caos* e il *testing* delle prestazioni è che l'ingegneria del caos non si concentra esclusivamente su alcuni componenti chiave, ma può includere un numero apparentemente illimitato di fattori, oltre l'ambito delle considerazioni di test normali e ovvie. In ambienti di rete distribuiti su larga scala, i sistemi possono fallire per una varietà di motivi che non sono facilmente individuabili rispetto ad altri contesti. Rilevare queste vulnerabilità aiuta i team a comprendere dove si trovano le debolezze per prevenire la possibilità che tali fallimenti si verifichino [9]

3.4 Fault injection

Le *fault injection* sono gli strumenti con quali condurre gli esperimenti del caos . Le *fault injection* rappresenta una tecnica fondamentale

per valutare la resilienza dei sistemi a guasti, che spaziano dai sistemi embedded e mobili a quelli distribuiti. Al fine di attuare una campagna di *fault injection* , è cruciale definire un modello di guasto, il quale delinei i guasti da emulare negli esperimenti. Tale modello comprende la definizione di tre aspetti principali: cosa iniettare (ovvero, il tipo di guasto), quando iniettare (ossia, il momento dell'iniezione) e dove iniettare (cioè, la parte del sistema bersaglio dell'iniezione). Il "cosa" può essere rappresentato da inversioni di bit, eccezioni di programma per amplificare test di unità e di integrazione, crash dei nodi, partizioni di rete e latenze per sistemi di rete e distribuiti. Il "quando" e il "dove" iniettare sono selezionati da un ampio spettro di possibilità attraverso il tempo e le posizioni nel programma.

Definire un modello di guasto si rivela più arduo quando si iniettano difetti software (ad esempio, errori di progettazione e/o di programmazione), dato che dipendono da una varietà di fattori tecnici e organizzativi, inclusi il linguaggio di programmazione, il processo di sviluppo software, la maturità del sistema, l'esperienza degli sviluppatori e il dominio applicativo. Nonostante la variabilità dei difetti software tra i sistemi, gli strumenti esistenti per l'iniezione di guasti software si basano su un modello di guasto software predefinito e fisso, che non può essere facilmente personalizzato dagli utenti. [2] In primo luogo, un'esigenza tipica nell'industria, che emerge quando si verifica un fallimento critico, è introdurre test di regressione contro il guasto

che ha causato il fallimento, per assicurarsi che lo stesso non possa verificarsi nuovamente. In secondo luogo, per preservare l'efficienza della campagna di iniezione di guasti, è importante evitare di iniettare bug che sono improbabili ad impattare un sistema; ad esempio, alcune classi di guasti possono essere prevenute tramite politiche di test e analisi statica adottate dall'azienda. In terzo luogo, man mano che la scala e la complessità dei sistemi aumentano, cresce la necessità di un modello di guasto più sofisticato. Ad esempio, i moderni sistemi distribuiti, come le applicazioni cloud, devono integrare una varietà di componenti, inclusi quelli di terze parti e open-source, e devono gestire alti volumi di traffico. Per questi sistemi, l'utente ha bisogno di iniettare varianti di difetti di progettazione/programmazione più numerose rispetto a quelle riportate in letteratura, includendo colli di bottiglia delle prestazioni, problemi di gestione delle risorse, mancanza di interoperabilità tra componenti, questioni di sicurezza, aggiornamenti falliti, ecc., e di adattare questi guasti ai propri progetti. In generale, gli utenti potenziali dell'iniezione di guasti software vogliono sintonizzare il modello di guasto in modo che rifletta la loro esperienza e aspettative riguardo ai fallimenti. Tutti questi casi d'uso richiedono un maggiore grado di controllo sul modello di guasto rispetto a quello fornito dagli strumenti di iniezione di guasti esistenti.[2]

3.5 Principi e Steps

L ' ingegneria del caos non ha come obbiettivo di fare solo terra bruciata , anzi il *caos* deve essere pianificato deve seguire una scaletta che ci permette di mitigare i rischi in un sistema distribuito . Gli step che ci permettono di creare una linea-guida sul come precedere con esperimenti del caos sono :

- **Step 1: Ipotesi**

  In questo step facciamo un assunzione sul come il sistema possa rispondere a condizioni di anomalia e guasti , in questa fase decidiamo anche quali metriche utilizzare come latenza , throughput per essere valutate .

- **Step 2 : Identificare Variabili e Anticipare gli Effetti**

  Immaginare scenari che posso accadere nella realtà che possono inficiare su tutto il sistema come ad esempio il crash di un server che vede un aumentare notevole di traffico

- **Step 3:Inizio esperimento**

  Gli *esperimenti del caos* possono essere effettuati in ambiente isolato di testing , che non possono comportare rischi , ma potrebbe non riflette scenari reali. Il posto migliore dove effettuare *esperimenti del caos* è proprio in produzione ,il che pero comporta dei rischi che vanno gestiti in modo tale da avere ancora il controllo del sistema nel *worst-case* , ciò viene chiamato controllo del *blast*

radius

- **Step 4 : Misurazione dell ' impatto**

  Valutare se gli i risultati corrispondo effettivamente le ipotesi poste , capire se effettivamente se l ' esperimento va ripetuto scalandolo di più , quali metriche sono state utilizzante e valutare numericamente i risultati ottenuti . [9]

Strumenti open-source e commerciali , come *Gremlin* , *Litmus Chaos* e *Chaos mesh* vengono adottati sempre di più nell' aziende e nella ricerca per sviluppare sistemi sempre più robusti e che in vari campi devono necessariamente dimostrare di possedere performance accettabili . Vederemo come nel capitolo successivo applicheremo i principi di questa disciplina per andare a valutare, e possibilmente scoprire nuove vulnerabilità ,il sistema oggetto di questa tesi

z

Chapter 4

Analisi Sperimentale

Prima di effettuare gli esperimenti e analizzare i risultati ottenuti facciamo una panoramica sull'ambiente in cui verrà effettuato il depolyment , orchestrato da Kubernetes, del 5C-Core . Il cluster di utilizzato è raggiunto tramite ssh ed è composto da due macchine virtuali :

- **Nodo master :** 8vCPU, 4GB di RAM , 200GB di storage , Ubuntu 22.04.3 LTS
- **Nodo worker :** 8vCPU, 16GB di RAM , 200GB di storage , Ubuntu 22.04.3 LTS

Entrambi utilizzano come runtime per i container *CRI-O*. Per effettuare gli esperimenti abbiamo utilizzato come strumento *Chaos Mesh* che tramite la dashboard rende la pianificazione di *fault injection* semplificata 4.1. Invece come strumento per la misura le metriche osservate abbiamo utilizzato *kiali* che analizza i dati e ci da a disposizione i

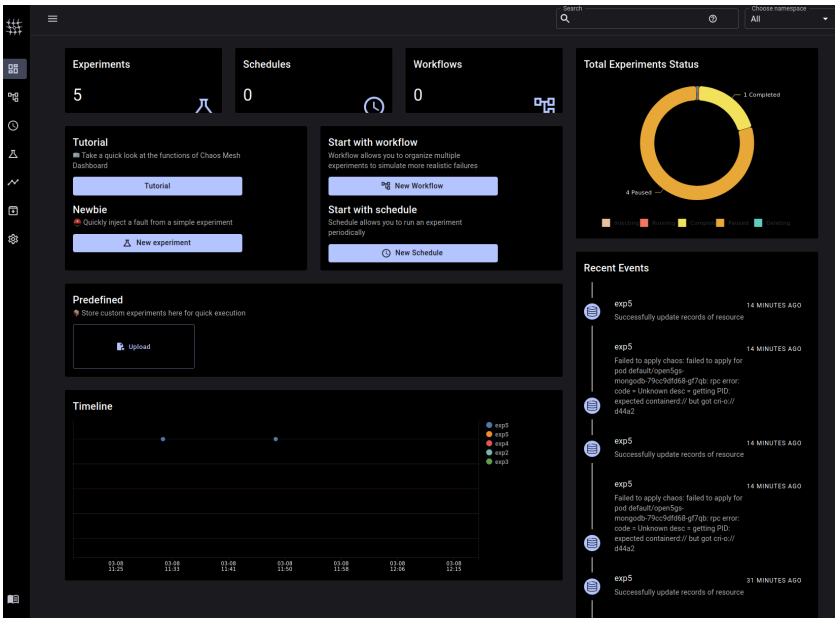


Figure 4.1: Chaos Mesh Dashboard

grafici necessari per l' analisi dei dati 4.2 Il deploy che è stato utilizzato inoltre utilizza UERANSIM per simulare un carico nel 5G-Core . [1]

4.1 Esperimento n°1 : Pod Failure

4.1.1 Ipotesi

Ipotizziamo che un esperimento condotto tramite una fault injection di tipo Pod failure nel depoloyment di mongodb della durata di 2 minuti provochi la caduta temporanea di 2 minuti di mongodb e quindi l' impossibilita di usare il servizio DB , il che implica che tutte le operazioni di registrazioni di UE verranno rigettate . Possiamo utilizzare come metriche il tempo di risposta del AMF ad una richiesta di registrazione .

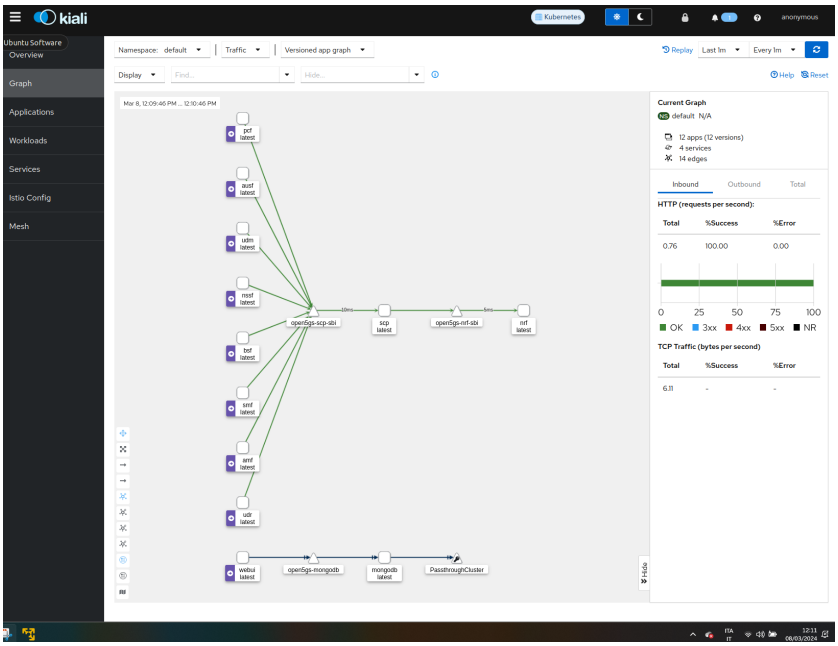


Figure 4.2: kiali dashboard

4.1.2 Inizio Esperimento

Tramite il deployment di ueransim-gnb-ues ho inviato 4 richieste di registrazioni tramite il comando *nr-ue -c ue.yaml*, 2 durante l ' intervallo di Failure injection del deployment di mongodb, 2 successivi alla ripresa . Tramite il deployment open5gs-populate ho inviato 2 richieste di iscrizioni:

- open5gs-dbctl add_ue_with_slice 999700000000004
  465B5CE8B199B49FAA5F0A2EE238A6BC
  E8ED289DEBA952E4283B54E88E6183CA
  internet 1 111111

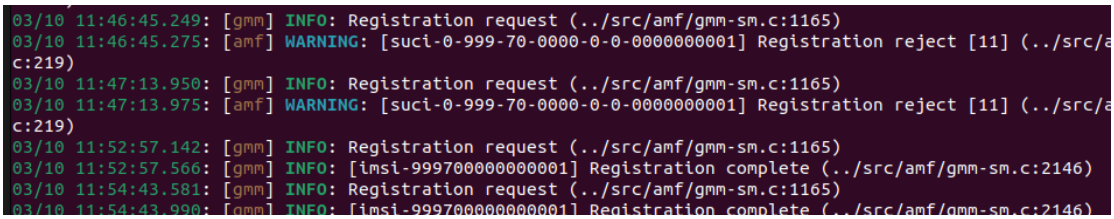
- open5gs-dbctl add_ue_with_slice 999700000000005
  465B5CE8B199B49FAA5F0A2EE238A6BC

E8ED289DEBA952E4283B54E88E6183CA

internet 1 111111

Possiamo procedere con l' esperimento utilizzando questo yaml :

```
kind: PodChaos
apiVersion: chaos-mesh.org/v1alpha1
metadata:
  namespace: default
  name: ExpPodFailureMongo
spec:
  selector:
    namespaces:
      - default
    labelSelectors:
      app.kubernetes.io/name: mongodb
  mode: all
  action: pod-failure
  duration: 2m
```



4.1.3 Misurazione impatto

Da come possiamo notare da 4.1.3 , le prime due richieste inviate all' AMF vengono rigettate con un tempo di risposta nell' ordine di decine di ms[26 , 25], mentre le successive 2 richieste vengono completate con un tempo di riposta nell ' ordine di centinaia di ms [420 , 409]. Come previsto la richiesta di iscrizione inviata da open5gs-populate prima del Failure è andata a buon fine , mentre quella successiva non viene portata a termine , possiamo pero notare che la richiesta non viene immediatamente rigettata anzi persiste finché non scade il timer per i tentativi di connessione a mongodb

4.1.4 Considerazioni

Utilizzare un Deployment per MongoDB in un ambiente come Open5Gs può non essere l'approccio ottimale, principalmente a causa della natura stateful di MongoDB. Ecco alcuni motivi specifici per cui l'uso di un StatefulSet è generalmente preferito per MongoDB (e database simili) in Kubernetes:

- **Persistenza dei Dati**: MongoDB, essendo un database, memorizza dati che devono persistere oltre il ciclo di vita di un singolo

pod. I Deployment non gestiscono in modo nativo la persistenza dello stato attraverso i riavvii dei pod o i ricollocamenti. I StatefulSet, invece, sono progettati per garantire che ogni pod abbia un'identità persistente e possa essere associato a volumi persistenti che conservano i dati anche quando i pod vengono riavviati o ricollocati.

- **Identità Fissa**: Ogni istanza di MongoDB necessita di un'identità stabile (per esempio, per la configurazione del replica set). I StatefulSet garantiscono che ogni pod abbia un nome persistente e prevedibile (ad esempio, mongodb-0, mongodb-1, ecc.), che rimane costante attraverso riavvii o aggiornamenti. Questo è cruciale per mantenere la configurazione del cluster di MongoDB coerente e funzionante.[4]

- **Scalabilità Ordinata e Aggiornamenti**: Nel caso dei database come MongoDB, potrebbe essere necessario scalare o aggiornare i pod in un ordine specifico per mantenere l'integrità e la disponibilità del cluster. I StatefulSet supportano scalabilità e aggiornamenti ordinati, assicurando che le operazioni siano eseguite in modo sequenziale, rispettando le dipendenze tra i pod.

- **Gestione dei Volumi Persistente**: MongoDB richiede un sistema di storage persistente per salvare i dati del database. I StatefulSet facilitano l'assegnazione di PersistentVolume a cias-

cun pod in modo che i dati persistano oltre il ciclo di vita del pod. Questo è essenziale per garantire che non ci sia perdita di dati durante la ricreazione o l'aggiornamento dei pod.

- **Riavvio e Ripristino**: In caso di fallimento, i StatefulSet facilitano il processo di riavvio e ripristino dei pod del database, mantenendo le stesse identità e collegamenti ai volumi persistenti. Ciò riduce la complessità della gestione dei fallimenti e del ripristino dei dati[4]

In sintesi, sebbene sia tecnicamente possibile eseguire MongoDB utilizzando un Deployment in Kubernetes, tale approccio non sfrutta le caratteristiche di Kubernetes progettate per gestire efficacemente le applicazioni stateful come MongoDB. I StatefulSet offrono vantaggi significativi in termini di gestione dello stato, persistenza dei dati, identità dei pod, e scalabilità ordinata, rendendoli la scelta preferibile per la maggior parte delle implementazioni di database in Kubernetes.

4.2 Esperimento n°2 : Pod failure su Statefulset Mongodb

Spostiamoci dall'architettura container-based Open5gs di per andare a studiare come reagisce ad una fault injection di tipo Pod Failure mongodb configurato come Statefulset con l'obbiettivo di mostrare i

vantaggi di questo approccio . I file yaml utilizzati per questo esperimento sono i seguenti :

- **Headless service** che ci permette di comunicare con le repliche di mongodb:

```
apiVersion: v1
kind: Service
metadata:
  name: mongodb
  labels:
    app: mongodb
spec:
  clusterIP: None
  selector:
    app: mongodb
  ports:
  - port: 27017
    targetPort: 27017
```

- **Il file yaml di mongodb stateful set :**

```

apiVersion: apps/v1
kind: StatefulSet
metadata:
```

```
  name: mongodb
spec:
  selector:
    matchLabels:
      app: mongodb
  serviceName: mongodb
  replicas: 3
  template:
    metadata:
      labels:
        app: mongodb
        selector: mongodb
    spec:
      containers:
      - name: mongodb
        image: mongo:4.0.17
        ports:
        - containerPort: 27017


        volumeMounts:
        - name: pvc
          mountPath: /data/db
  volumeClaimTemplates:
```

```
  - metadata:
      name: pvc
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Gi
```

- **Il file yaml del client di mongodb**:

```
apiVersion: v1
kind: Pod
metadata:
  name: mongodb-data-inserter
spec:
  restartPolicy: Never
  containers:
  - name: mongodb-client
    image: mongo:4.0.17
    command: ["/bin/sh"]
    args:
      - "-c"
      -
        mongo mongodb://mongodb:27017/
```

```
            --eval 'db.test.
                insertOne({"nome": "Mario",
                    "cognome": "Rossi"});'
```

Tramite chaos-mesh appicco una fault-injection di tipo di Pod Failure diretto ad una repliche nel particolare mongodb-0.

durante l' intervallo dell' esperimento (in questo caso indefinito) creo un pod utilizzando il file yaml per il client mongodb che invierà i dati verso mongodb.

4.2.1 Ipotesi

In quanto mongodb è configurato come Statefulset la caduta di mongodb-0 porterà l ' elezione del leader successivo che per ordine crescente sarà mongodb-1 , il che risulterà che i dati inviati tramite il cliente verranno salvati in quest'ultimo.

4.2.2 Inizio Esperimento

Procediamo con l ' esperimento applicando la fault injection che presenta questo file yaml.

```
kind: PodChaos
apiVersion: chaos-mesh.org/v1alpha1
metadata:
  namespace: default
```

```
> db.test.find()
> db.test.find()
> db.test.find()
{ "_id" : ObjectId("65f306dc60c787b4baab8e64"), "nome" : "Mario", "cognome" : "Rossi" }
>
```

```
  name: EsperimentoMongoDBStatefulSet
spec:
  selector:
    namespaces:
      - default
    labelSelectors:
      apps.kubernetes.io/pod-index: '0'
  mode: all
  action: pod-failure
```

4.2.3 Misurazione impatto

Come previsto dopo la caduta di mongodb-0 , i dati inviati tramite client si sono salvati in mongodb-1.4.2.4

4.2.4 Considerazioni

Da questo esperimento possiamo quindi notare l'importante vantaggio di utilizzare Statefulset per servizi che si occupano della persistenza dei dati .

Chapter 5

Conclusione

La presente tesi ha avuto come obiettivo principale la valutazione delle prestazioni e della resilienza di un'architettura 5G containerizzata, con un focus specifico sui guasti di persistenza dei dati utilizzando MongoDB all'interno di un ambiente gestito da Kubernetes. Attraverso una serie di esperimenti sperimentali, incentrati sull'applicazione dei principi dell'ingegneria del caos, abbiamo potuto osservare e quantificare l'impatto dei guasti software sulla disponibilità e sul comportamento complessivo del sistema.

I risultati ottenuti evidenziano l'importanza critica di una gestione adeguata dei componenti stateful, come i database, nelle architetture containerizzate. In particolare, è emersa la superiorità dell'uso di StatefulSets su Deployment in Kubernetes per gestire applicazioni con esigenze di persistenza dei dati, come MongoDB. Questo approccio migliora significativamente la gestione dello stato, garantendo l'identità

persistente dei pod e una gestione più efficace dei volumi persistenti.

Gli esperimenti hanno inoltre sottolineato come la resilienza di un'architettura 5G possa essere compromessa da guasti relativi alla persistenza dei dati, sottolineando l'importanza di implementare strategie di gestione dei guasti e di recupero che possano minimizzare l'impatto di tali eventi. La capacità di un sistema di recuperare rapidamente da un guasto, mantenendo l'integrità dei dati e garantendo la continuità del servizio, rappresenta un fattore critico per il successo delle reti 5G in ambienti di produzione reali.

In conclusione, questa tesi ha fornito importanti intuizioni sulla gestione dei guasti in architetture 5G containerizzate e ha evidenziato l'efficacia dell'ingegneria del caos come strumento per valutare e migliorare la resilienza di tali sistemi. Le conoscenze acquisite contribuiranno alla progettazione di architetture 5G più robuste e affidabili, capaci di fronteggiare efficacemente le sfide poste dall'evoluzione delle esigenze di comunicazione nel panorama tecnologico contemporaneo.

Bibliography

[1] https://gradiant.github.io/5g-charts/open5gs-ueransim-gnb.html.

[2] Domenico Cotroneo, Luigi De Simone, Pietro Liguori, and Roberto Natella. Profipy: Programmable software fault injection as-a-service. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 364–372, 2020.

[3] Maciej Gawel and Krzysztof Zielinski. Analysis and evaluation of kubernetes based nfv management and orchestration. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 511–513, 2019.

[4] https://kubernetes.io/docs/home/. Kubernetes documentation.

[5] https://open5gs.org/open5gs/docs/. Open5g documentaion.

[6] https://softwaremind.com/blog/why-open5gs-plays-an-important-role-in-open-source-private-5g-network development/.

Why open5gs plays an important role in open source private 5g network development.

[7] https://www.gremlin.com/community/tutorials/chaos-engineering-the-history-principles-and practice. Chaos engineering: the history, principles, and practice.

[8] https://www.ibm.com/topics/containerization. What is containerization?

[9] https://www.loadview-testing.com/blog/chaos-engineering-principles-examples tools/. Chaos engineering: Principles, examples and tools.

[10] https://www.redhat.com/en/topics/cloud-native-apps/vnf-and-cnf-whats-the difference/. Quali sono le differenze tra vnf e cnf?

[11] Nathan F. Saraiva de Sousa, Danny A. Lachos Perez, Raphael V. Rosa, Mateus A.S. Santos, and Christian Esteve Rothenberg. Network service orchestration: A survey. *Computer Communications*, 142-143:69–94, 2019.

[12] Tarik Taleb. Toward carrier cloud: Potential, challenges, and solutions. *IEEE Wireless Communications*, 21(3):80–91, 2014.

[13] Qiao Yan, F. Richard Yu, Qingxiang Gong, and Jianqiang Li. Software-defined networking (sdn) and distributed denial of ser-

vice (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges. *IEEE Communications Surveys and Tutorials*, 18(1):602–622, 2016.

[14] Faqir Zarrar Yousaf, Michael Bredel, Sibylle Schaller, and Fabian Schneider. Nfv and sdn—key technology enablers for 5g networks. *IEEE Journal on Selected Areas in Communications*, 35(11):2468–2478, 2017.