Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Informatica

Sviluppo di un Framework DevOps per la Gestione di Applicativi Cloud Serverless

Relatore:
Chiar.mo Prof.
Saverio Giallorenzo

Presentata da:
Christian Preti

Sessione II
2023/24

Vorrei dedicare questo spazio a tutti coloro che
mi hanno supportato ed accompagnato
durante l'intero percorso universitario.
In primis, ringrazio il Prof. Saverio Giallorenzo,
per la costante disponibilità ed assistenza.
Ringrazio Giuseppe, Umberto e Stefano per i consigli
ed il supporto fornitomi durante il tirocinio.
Un ringraziamento speciale alla mia ragazza,
Gloria, che più di tutti mi ha supportato e
spronato a fare sempre meglio durante l'intero
percorso. Ovviamente, un grazie anche alla
mia famiglia e ai miei amici, che hanno contribuito
a rendere possibile tutto ciò.

Indice

1 Introduzione

Il «cloud computing» rappresenta la distribuzione di servizi di diverso tipo, principalmente servizi di calcolo, risorse di archiviazione, rete e software, tramite Internet. Rispetto ad una visione tradizionale delle risorse IT, migrare verso il cloud computing significa eliminare le spese associate all'acquisto di hardware, la loro configurazione, la gestione di data center locali e altre spese annesse. Inoltre, l'hardware messo a disposizione è, spesso, di ultima generazione, con il software aggiornato alle versioni più recenti. Un hardware e software di questo tipo permette di avere un'altissima scalabilità, in modo da fornire supporto a qualsiasi economia di scala. La cybersicurezza è inoltre uno dei punti cardine di queste tecnologie, data la gamma di criteri e controlli che vengono inseriti per garantire la protezione dei dati all'interno [30]. Sebbene la parola «cloud» sia stata utilizzata per la prima volta intorno al 1993, la sua diffusione commerciale in ambito enterprise avviene nel 2006, con l'introduzione, da parte di AWS, dei servizi EC2 [34].

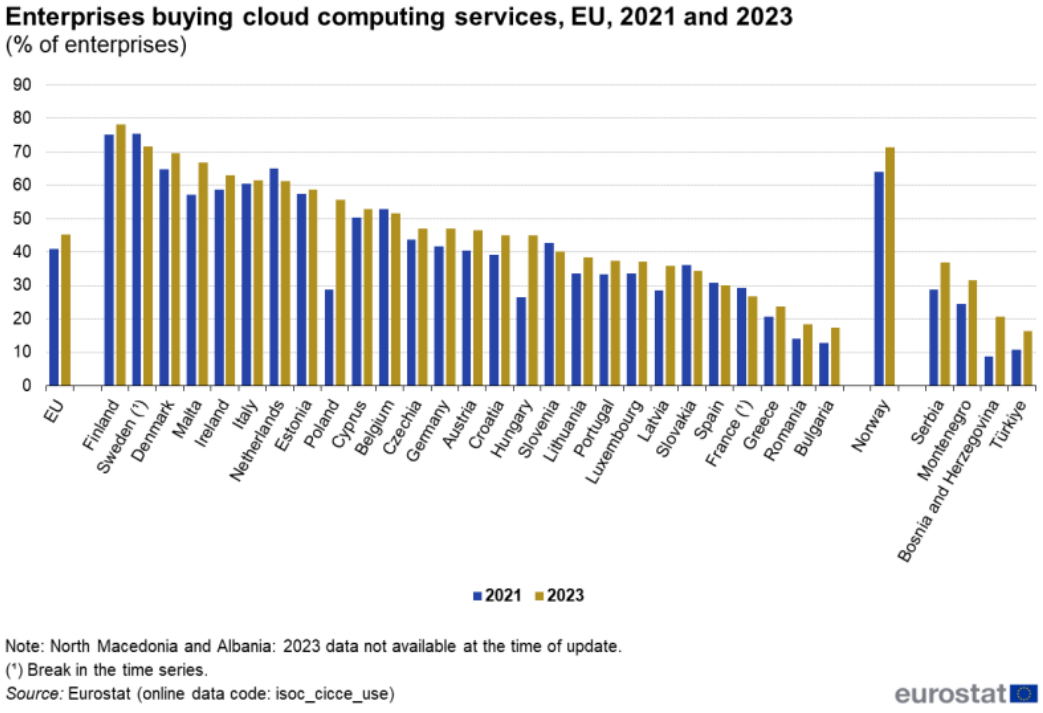


Figure 1: Aziende europee facenti uso di servizi cloud, in percentuale.

L'adozione pervasiva del cloud computing da parte delle organizzazioni è il risultato di una sequenza di fattori economici, tecnologici e strategici.
L'esigenza di archiviare grandi quantità di dati ha portato le aziende a cercare soluzioni che riducessero i costi legati all'acquisto e alla manutenzione di hardware dedicato. La scalabilità e la flessibilità dei modelli di pricing sono due dei principali fattori che spingono le aziende verso soluzioni cloud. La possibilità di gestire la disponibilità delle risorse, pagando con un modello pay-per-use, consente alle organizzazioni di ottimizzare i costi e rispondere in modo più agile alle opportunità e oscillazioni di mercato.
L'Unione Europea ha riconosciuto, nell'ultimo decennio, l'importanza sempre maggiore che sta guadagnando il cloud computing. In un mercato dominato da colossi aventi sede legale in aree extra-europee, si dimostra fondamentale ridurre la dipendenza dai servizi forniti da quest'ultimi. A tal fine, l'UE sta investendo in progetti atti a sviluppare un'infrastruttura all'interno del territorio europeo. Per garantire un mercato libero e non competitivo, l'UE sta lavorando ad un insieme di regole per il cloud, detto «EU Cloud Rulebook». Da non sottovalutare l'aspetto sostenibilità: come affermato dall'UE stessa, i data center che verranno installati dovranno essere climate-neutral, in modo da favorire l'applicazione dell'«European Green Deal» [35].
L'Unione Europea ha disposto alcuni obiettivi da raggiungere entro il 2030, tra i quali:

- Almeno il 75% delle aziende aventi una sede in Europa dovrà fare uso di sistemi di cloud computing nello svolgimento delle proprie attività;
- Dovranno essere costruiti 10000 «edge nodes» climate-neutral e con standard di sicurezza molto elevati, per garantire una connettività sicura e rapidi scambi di dati [35].

Inoltre, come affermato dall'Unione Europea, sebbene attualmente la quasi totalità dei dati venga processata all'interno di data center gestiti dai cloud providers, nei prossimi anni ci si aspetta che queste computazioni avvengano direttamente sui dispositivi degli utenti finali (o quanto più vicini ad essi possibile), passando ad un modello di calcolo noto come «edge computing». Proprio a causa di questa previsione, si rivela fondamentale la proprietà climate-neutral che questi dispositivi devono possedere.

2 Obiettivi progettuali

Dato il cambiamento sempre più radicale che sta subendo lo sviluppo software al giorno d'oggi, si è reso necessario effettuare cambiamenti, alle volte anche drastici, alle metodologie di sviluppo del software stesso.
Questo progetto di tesi nasce con lo scopo di investigare l'applicabilità, e i relativi vantaggi, di metodologie DevOps e strategie di branching su progetti cloud, specificamente in modalità serverless. Per farlo, è stato sviluppato un framework con lo scopo di automatizzare l'inizializzazione e la successiva gestione di un progetto cloud serverless, che fornisca flessibilità e supporto a diverse strategie di branching, dettagliatamente descritte ed approfondite nei successivi capitoli.
Tra le necessità di cui la comunità di sviluppatori ha bisogno, è emerso come uno dei principali problemi dello sviluppo software sia la mancanza, o insufficienza, di automazione nelle varie fasi del ciclo di vita del software. Per questo motivo, sono nati nuovi ruoli all'interno delle aziende, come la figura del DevOps Engineer.
Unendo gli ambiti Cloud e DevOps è stato sviluppato BlueDAC, un framework per l'automazione delle varie fasi del ciclo di vita del software.

3 Tecnologie utilizzate

Verranno di seguito introdotte le principali tecnologie utilizzate per la realizzazione del framework. Esse verranno brevemente descritte, per poi essere dettagliatamente discusse nei relativi capitoli.

3.1 AWS

Grazie alla quantità e la qualità dei servizi offerti, AWS rientra tra i principali provider del panorama cloud. Il framework sviluppato è stato pensato per essere integrato specificamente con questo provider, sebbene si sia valutata la possibilità, come sviluppo futuro, di fornire supporto ad ulteriori cloud provider.

3.2 Microsoft Teams

Al fine di garantire scambi di informazioni puntuali ed efficaci, le comunicazioni sono avvenute sulla piattaforma Microsoft Teams, estremamente diffusa in ambito aziendale grazie all'integrazione con altri strumenti, messi a disposizione da Microsoft (e.g. suite Office, Planner).

3.3 Artillery

Artillery rappresenta uno strumento per la creazione e l'esecuzione di test di carico per applicazioni cloud. Il suo utilizzo è consigliato da AWS stesso [10], grazie alla potenza, facilità di utilizzo e flessibilità che esso mette a disposizione. Inoltre, anche l'alto livello di integrazione con gli altri strumenti utilizzati all'interno del progetto ne ha permesso un pratico utilizzo.

3.4 Poetry

Il framework sviluppato è stato liberamente rilasciato su PyPi, principale piattaforma per la distribuzione di librerie Python. Maggiori informazioni sulla sua reperibilità sono fornite nei capitoli successivi.

3.5 GitLab

GitLab rappresenta una delle principali soluzioni per la gestione di repository Git. A differenza di soluzioni ben più note, essa possiede strumenti di DevOps

molto più avanzati, senza dimenticare il modello open-source adottato dalla piattaforma stessa. L'intero codice del framework è stato caricato su questa piattaforma.

3.6 Jira

Jira è uno dei principali strumenti di project management utilizzati in ambito enterprise, sviluppato da Atlassian. Le sue funzionalità principali sono il tracciamento di bug, tramite l'apertura di appositi ticket, e la possibilità di organizzare i diversi task all'interno del progetto, tramite l'assegnazione di quest'ultimi ai vari membri del team. Col tempo, ha fornito supporto a molte funzionalità, tra cui, ad esempio, la possibilità di definire user stories e calibrarne le stime e supporto agli «Scrum sprints».

4 Branching Strategies

Una branching strategy rappresenta una particolare gestione dei branches all'interno del repository della piattaforma di versionamento in uso.
La scelta della branching strategy è essenziale per garantire un'efficace gestione del codice e comunicazione tra gli sviluppatori. È bene ricordare che ogni strategia possiede pro e contro, che dipenderanno dalle esigenze del team e dell'applicativo stesso.
Nel corso degli ultimi anni, le branching strategies hanno subito significativi adattamenti per rispondere alle mutevoli esigenze del software. Questi cambiamenti sono stati guidati da fattori come l'adozione di architetture a microservizi, lo sviluppo cloud ed altre tendenze emergenti.
Prima di affrontare un approfondimento e confronto tra le diverse branching strategies, è doveroso definire il principale problema che è possibile incontrare a seconda della branching strategy scelta, ossia i conflitti.

4.1 Conflitti

Un conflitto si verifica quando due, o più, sviluppatori modificano le stesse righe dello stesso file o se uno sviluppatore elimina un file che un collega ha modificato. In queste situazioni, Git non è in grado di effettuare automaticamente una scelta che venga incontro alle modifiche di entrambi gli sviluppatori. Per risolvere i conflitti, lo sviluppatore che ha aperto la richiesta di merge avrà il compito di risolvere manualmente ogni singolo conflitto, per procedere con la richiesta.
Sono presenti diversi tipi di conflitto, sia nella fase iniziale del merge, sia in fase avanzata:

- Conflitto in fase iniziale di merge: Si verifica quando uno sviluppatore apre una merge request dal branch «A» verso il branch «B». Se il branch B contiene modifiche in sospeso (modifiche locali alla directory di lavoro che non sono state sincronizzate con il repository remoto), la richiesta verrà interrotta. Questo permette di evitare che le nuove modifiche, ottenute al termine della richiesta di merge, vadano a sovrascrivere modifiche in sospeso presenti sul branch di destinazione. Per procedere, lo sviluppatore dovrà decidere se eliminare le modifiche effettuate, sincronizzarle con il repository remoto (push) oppure effettuare il cosiddetto «stash».

- Conflitto durante la richiesta di merge: Si verifica quando le proprie modifiche sono in contrasto con quelle apportate da un altro sviluppatore. Git proverà a risolvere automaticamente i conflitti, ma lascerà sempre decidere allo sviluppatore se intervenire manualmente. Si noti come, spesso, i conflitti necessitino di intervento umano a causa di impossibilità di decisione automatica da parte di Git.

4.2 GitFlow

GitFlow [7] è una branching strategy «legacy», ossia una strategia che in passato ha rappresentato un'innovazione nel campo della gestione dei branches, ma che vede sempre più la necessità di un rimpiazzo.
Si tratta di un insieme di pratiche con cui organizzare il proprio repository Git, che coinvolge l'utilizzo di alcuni branch primari e diversi branch secondari, detti «feature branches».
Essa è stata creata da Vincent Driessen nel 2010, per indirizzare gli sviluppatori ad una migliore gestione dei branches all'interno del proprio repository, in modo da ottimizzare l'organizzazione e la collaborazione dei diversi membri del team.
I branch primari sono:

- Master (recentemente rinominato in «main»): contenente il codice attualmente rilasciato in produzione;

- Develop: contenente il codice derivato dai feature branches che sono stati completati, e che verrà rilasciato tramite un release branch;

- Release: Un branch di release è un branch contenente codice che è pronto per essere distribuito in produzione, definendo una nuova versione del prodotto. Esso è creato a partire dal «develop» branch. Dopo averlo creato, non è più possibile inserire nuove features all'interno della versione, bensì le uniche modifiche che è lecito effettuare sono «bug-fixes», generazione della documentazione e altre piccole operazioni legate alla nuova versione. Una volta pronti a distribuire la nuova versione, bisognerà aprire una «merge request» sul main branch. Nel caso la procedura andasse a buon fine, dovrà essere applicato un «tag» al commit appena creato per farlo risultare, a tutti gli effetti, come una nuova release. È essenziale ricordarsi di effettuare una merge request anche sul «develop»

branch, dato che il codice contenuto in quest'ultimo potrebbe non essere stato aggiornato, date le modifiche effettuate al release branch.

- Hotfix: Un hotfix branch è un branch utilizzato per risolvere bug presenti in produzione. Esso, a differenza dei release branch, è creato a partire dal main. Dopo aver risolto correttamente i bug, è possibile aprire una merge request dall'hotfix branch al main branch (e develop), taggando il nuovo commit sul main con una versione aggiornata (solitamente si applica una «minor version change»).

I branch secondari citati prima sono i cosiddetti «feature branches», ossia branch creati con lo scopo di ospitare l'implementazione di nuove features.
Ogni feature branch deve corrispondere ad una ed una sola feature. Quando la feature è pronta, può essere effettuata una merge request sul develop branch.
GitFlow è la branching strategy più datata, pensata e, di conseguenza, ottimizzata per software che non hanno necessità di essere rilasciati frequentemente.
Il punto a sfavore principale del GitFlow è, infatti, quello di inibire la «Continuous Delivery», proprietà particolarmente in voga nell'ultimo periodo che indica un rilascio frequente (una o più volte al giorno) del software. Ne consegue che la fase di merge sia particolarmente problematica, essendo solitamente i commit molto grandi e pertanto più a rischio di causare conflitti all'interno della codebase.

4.3 GitHub Flow

Il GitHub Flow [21] è stato sviluppato da GitHub stesso ed è utilizzato da quest'ultimi per gestire alcuni loro repository, quali:

- site-policy
- documentation
- roadmap

La prima differenza che troviamo rispetto a GitFlow è l'assenza di alcuni branch primari, ossia: «develop», «release» e «hotfix».
Per applicare correttamente l'approccio GitHub Flow, quando è necessario inserire una nuova feature all'interno del software, si crea un nuovo branch a partire dal main. Dopo aver completato lo sviluppo della feature, verrà aperta una pull request verso il main. È buona norma che, all'apertura della pull request, gli altri membri del team controllino le modifiche effettuate, fornendo

consigli e possibili ottimizzazioni.
Una volta ottenuto un feedback positivo, si può procedere con il merge sul main branch. Dopo aver concluso il processo, sarà necessario eliminare il feature branch, in modo da evidenziare la conclusione del lavoro. Non è necessario preoccuparsi di perdita di informazioni, dato che tutte le modifiche verranno mostrate nella descrizione del «merge commit».

4.4 GitLab Flow

Allo stesso modo di come GitHub ha creato il proprio workflow, troviamo l'approccio GitLab [22].
Esso si differenzia dal GitFlow evitando di usare un develop branch, ma bensì effettuare tutte le modifiche sul main branch. È poi presente un «production branch», che conterrà il codice attualmente in produzione. Non appena il codice sul main è pronto per essere rilasciato in produzione, verrà aperta una merge request verso un branch di pre-produzione, su cui verranno effettuate ulteriori (piccole) modifiche, per poi procedere con il rilascio sul production branch. Solitamente, GitLab Flow fa uso dei release branch per evidenziare le varie versioni del software.

4.5 Trunk Based Development

Il Trunk Based Development, spesso abbreviato con l'acronimo TBD, è una nuova branching strategy con numerose differenze rispetto a GitFlow. Infatti, se con GitHub Flow e GitLab Flow si è assistito a dei miglioramenti del GitFlow, con TBD si ha un completo cambio di paradigma.
Il main branch, nel TBD, prende il nome di «trunk» in relazione ad un paragone con il tronco di un albero. Quest'ultimo rappresenta la parte principale e più grossa dell'albero, a cui troviamo annessi i rami dell'albero, di ben più piccole dimensioni, ossia i feature branches. [26]
Di seguito, un approfondimento alle principalmente differenze del Trunk Based Development.

4.5.1 Branch di breve durata

Il cambiamento principale risiede nel «lifetime» dei feature branch. Se nel GitFlow si avevano feature branches che potevano rimanere aperti per molto tempo (causa l'implementazione di intere features, spesso molto grosse) e fornire, in fase di merge, grossi cambiamenti alla codebase, con TBD lo scopo è

quello di creare feature branches con una durata di vita breve, che risolvino un task di dimensione inferiore e che apportino meno problemi in fase di merge. Infatti, più saranno le modifiche effettuate nel branch sorgente del merge, più alta sarà la possibilità di incontrare conflitti.
Inoltre, avere feature branches di breve durata permette di avere rilasci molto più frequenti sul trunk, abilitando la «continuous delivery».
Essendo i feature branches di breve durata e frequentemente spostati sul trunk, gli sviluppatori avranno la possibilità di rimanere più aggiornati con la versione più recente del codice, rispetto ad altre branching strategies. Se così non fosse, si verificherebbe una situazione problematica. In fase di sviluppo si assisterebbe a numerosi conflitti in fase di merge request, causate da mancate sincronie tra le varie codebase locali di ogni sviluppatore, che causerebbero grosse perdite di tempo in fase di risoluzione. [29]
In riferimento a quest'ultimo fenomeno, si tenga conto anche dell'alta probabilità di effettuare errori, siccome la maggior parte dei conflitti richiede intervento manuale.

4.5.2 Push diretti sul trunk

In caso di team di dimensione molto piccola, potrebbe essere molto conveniente effettuare il push delle modifiche direttamente sul trunk, senza dover creare un feature branch. Questo è consigliato principalmente per piccoli team dato che, essendo composto da poche persone, i membri sono aggiornati sul lavoro dei colleghi.
Il benificio principale è relativo alla velocità di sviluppo, grazie alla non necessità di dover creare un branch apposito e dover aprire una richiesta di merge, che dovrà successivamente superare una pipeline di approvazione automatica oppure ottenere l'approvazione manuale di altri membri, interrompendo il loro flusso di lavoro.
Il problema, diretta conseguenza di questo approccio, è proprio la distribuzione di codice direttamente sul trunk. Se non opportunamente controllato, potrebbe essere rilasciato codice non propriamente funzionante.
Per risolvere questo problema, la soluzione più efficiente è quella di definire una pipeline che si occupi di verificare la correttezza del codice, eseguendo propriamente una suite di test di vario tipo su ogni nuovo commit. Il codice verrà approvato se e solamente se ogni passo della pipeline verrà superato correttamente [24].

4.5.3 Totale disponibilità al rilascio

Uno dei principi cardine dell'approccio TBD è quello di mantenere il codice in una fase di perenne disponibilità al rilascio. [27]
Ciò significa che, in qualsiasi momento, si deve poter assicurare la disponibilità e fattibilità del rilascio del codice in produzione. Questo, a differenza di altre branching strategies, è facilitato dalla poca distanza tra gli sviluppatori. Con distanza, in questo contesto, intendiamo una distanza logica tra il codice locale dei diversi sviluppatori. La principale fonte di distanza sono i branch, soprattutto se di durata prolungata, per la loro natura di dover implementare grosse features e quindi poter rimanere disallineati dal trunk per molto tempo. Nel TBD, avendo branch di breve durata, è possibile garantire un rilascio nel breve periodo tramite la terminazione del lavoro rimanente presente nei feature branches, apertura di una richiesta di merge sul trunk ed effettivo rilascio.

4.5.4 Feature flags

Definiamo feature flags un approccio allo sviluppo che permette di specificare, tramite costrutti condizionali, un diverso comportamento che verrà esibito dal software. [25] Questo permette di mantenere diverse funzionalità visivamente «nascoste» ma presenti all'interno del codice. I vantaggi di questo approccio sono una maggior velocità nelle fasi di sviluppo, dato dalla non necessità di dover eliminare e riscrivere codice, limitando l'effettivo utilizzo della feature tramite un costrutto condizionale.
La configurazione dei parametri di avvio del software offre un meccanismo flessibile per attivare e disattivare le feature flag. Ciò permette di personalizzare l'esperienza di utilizzo per specifici gruppi di utenti, oltre a fornire la possibilità agli sviluppatori di testare nuove features senza dover creare ulteriori branch.

4.5.5 Branch by Abstraction

Applicare feature flags al codice vuol dire effettuare la cosiddetta «Branch by Abstraction», ossia astrarre porzioni di codice in modo da oscurarne l'utilizzo. Si immagini come scenario lo sviluppo di una funzionalità che richiederà una settimana di lavoro. La tentazione principale sarebbe la creazione di un branch in cui procedere con lo sviluppo della suddetta, allontanandosi momentaneamente dal resto del team. L'idea della branch by abstraction è quella di introdurre sia il codice precedente, sia il nuovo. Tramite feature flags (o altri

approcci per la gestione condizionale), bisognerà mantenere attivo il codice precedente, in modo da fornire piena compatibilità con il resto del codice, che potrebbe star venendo sviluppato da altri membri.
Una volta accertatosi della funzionalità del nuovo codice sarà possibile invertire la condizione (o rimuovere direttamente il costrutto), in modo da sostituire il vecchio codice.
Oltre alla comodità di non dover gestire creazione, richiesta di merge ed eliminazione di un branch, la branch by abstraction risulta efficiente anche per un «rollback» in caso di problemi: difatti, sarà sufficiente ritornare alla condizione originale dell'astrazione per oscurare il codice problematico.

4.5.6 Modalità di rilascio

I team che fanno uso dell'approccio TBD hanno due principali modalità di rilascio, a seconda dei requisiti progettuali e cadenze di rilascio.
I team che hanno cadenze di rilascio regolari ma non frequenti (ad esempio, mensili) fanno solitamente uso di release branch per effettuare rilasci di nuove versioni del codice. Questa soluzione prende il nome di «branch for release» [23].
In caso venisse trovato un bug all'interno del codice rilasciato tramite un release branch, la soluzione migliore (ma non sempre possibile) è quella di riprodurre il bug sul trunk, risolverlo e verificarne la risoluzione tramite una pipeline automatica, e procedere con lo «cherry-pick» del commit verso il release branch. In caso non fosse possibile riprodurre il bug sul trunk, sarà necessario risolvere direttamente sul release branch.
Per team aventi necessità di un'alta frequenza di rilascio, la soluzione più efficiente è quella del «release from a tag» [28]. In questo modo, non sarà necessario creare un release branch ad ogni nuovo rilascio, ma sarà sufficiente applicare un tag al commit sul trunk che dovrà essere rilasciato. In caso emergessero problemi con il codice rilasciato, non essendo stato creato alcun release branch, sarà possibile intervenire direttamente sul trunk ed, eventualmente, procedere con l'applicazione di un nuovo tag.

5 DevOps

Nato intorno al 2007 in seguito alle perplessità della comunità di sviluppatori in merito alle tradizionali metodologie di sviluppo software, con il termine «DevOps» indichiamo un insieme di pratiche e strumenti che permettono di automatizzare e integrare i processi tra i team di sviluppatori e il team IT. L'enfasi principale è sull'automazione, la comunicazione e la collaborazione tra team, spesso di ambiti differenti.
Il termine DevOps rappresenta l'unione delle parole «Development» e «Operations»: tradizionalmente, i team di sviluppo e i team operativi erano completamente separati. L'integrazione tra i due team permette di ottenere un ciclo di sviluppo e gestione successiva del software continuo [5].

5.1 Principi e cultura

Il team DevOps include sviluppatori e addetti alle operazioni IT che collaborano durante l'intero ciclo di vita del prodotto, al fine di aumentare la velocità e la qualità delle distribuzioni del software. I team DevOps utilizzano strumenti per automatizzare ed accelerare i processi, in modo da aumentarne la velocità e l'affidabilità.
Questa toolchain permette di affrontare ed ottenere i principi cardine del DevOps, tra i quali:

- Automazione;
- Collaborazione;
- Continuous Integration;
- Continuous Delivery/Deployment.

5.2 Toolchain

Una toolchain [4] è un insieme di strumenti che si integrano tra di loro per raggiungere un obiettivo comune.
Solitamente appartengono a fornitori differenti, ma sono facilmente integrabili tra di loro. Si differenziano principalmente due tipi di toolchain:

- All-in-one: Si tratta di una soluzione completa, spesso comprendente strumenti dello stesso fornitore. Per questo motivo, potrebbero esserci

problemi in fase di integrazione con strumenti di terze parti. Il vantaggio principale è quello di avere fin da subito una toolchain completa, utile per team che hanno appena iniziato ad approcciarsi alle pratiche DevOps o per team che vogliono avviare rapidamente un nuovo progetto;

- Personalizzata: Si tratta di una soluzione personalizzata, che risolve il problema principale della versione «all-in-one». È principalmente indicata per team esperti, con preferenze verso specifici strumenti o che hanno necessità di integrare tool provenienti da fornitori differenti.

Qualunque sia il tipo di toolchain utilizzata dal team, l'aspetto principale è l'integrazione. Se i vari strumenti faticano ad integrarsi tra di loro, i membri del team perderanno tempo nel passare da uno strumento all'altro manualmente, inibendo anche una sostanziale parte di automazione.

5.3 Continuous Integration

Il concetto di «Continuous Integration» indica un processo di aggiornamento costante e frequente del repository remoto con la propria codebase locale. [3]
Effettuare push di codice con cadenza alta e regolare permette di risolvere un fenomeno comunemente detto «integration hell», ossia la situazione, in fase di merge, in cui gli sviluppatori si trovano a dover risolvere molteplici conflitti all'interno del codice. Ciò è causato da un elevato disallineamento tra il codice disponibile sul repository remoto e il proprio codice locale. Il modo più semplice per risolvere questo problema è quello di effettuare push frequenti e disallinearsi dalla codebase remota per il minor tempo possibile.
Ciò si riflette anche sul lavoro degli altri sviluppatori che, seguendo la best practice di effettuare frequentemente operazioni di «pull», potranno mantenere quanto più possibile il proprio codice locale aggiornato.

5.4 Continuous Delivery

La Continuous Delivery rappresenta un'estensione della «CI», descritta al capitolo precedente. [1]
Con essa indichiamo la pratica di rilasciare frequentemente il prodotto software già dalle fasi iniziali di sviluppo. Nello sviluppo software classico, il prodotto veniva rilasciato nelle fasi finali del processo di sviluppo, dopo aver inserito tutte le features richieste dal cliente e aver effettuato tutti i controlli di qualità necessari. Questo impediva di avere feedback costanti dal cliente e dagli utenti

finali, utilizzatori del prodotto.
Inoltre, rilasciare frequentemente permette anche di velocizzare la risoluzione di eventuali problematiche, data la dimensione ridotta delle modifiche presenti tra un rilascio e l'altro.

5.5 Continuous Deployment

La Continuous Deployment rappresenta uno step ancora successivo alla Continuous Delivery. [2]
Essa consiste nell'automazione dell'intero processo di rilascio, portando il prodotto ad essere disponibile all'utente finale nel caso in cui la pipeline CI/CD non avesse problemi.
La differenza con la Continuous Delivery risiede nell'ambiente in cui il prodotto viene rilasciato: nella Continuous Delivery abbiamo un rilascio in qualsiasi ambiente, sia esso di testing, staging o produzione. Nella Continuous Deployment indichiamo specificamente un rilascio in produzione.

5.6 Pipeline CI/CD

Come descritto nei capitoli precedenti, per ottenere la Continuous Integration, Continuous Delivery e Continuous Deployment, è necessario che il codice superi una determinata sequenza di passi (solitamente una suite di test di vario tipo). L'insieme di step che devono essere superati compone la cosiddetta «pipeline CI/CD». [32]
Una pipeline CI/CD prende il nome dall'integrazione dei processi di Continuous Integration e Continuous Delivery (o Deployment, a seconda dei casi).
Il suo scopo è quello di automatizzare entrambi i processi, eliminando la necessità di intervento umano che potrebbe portare a errori, o anche semplicemente rallentamenti nel flusso di lavoro.
All'interno di una pipeline CI/CD solitamente troviamo «jobs» che si occupano di:

- Compilazione del codice e verifica dell'integrità;
- Verifica della funzionalità del prodotto tramite suite di test;
- Analisi statica del codice;
- Creazione di binari o eseguibili;
- Eventuale inserimento del codice in un container.

Tramite l'esecuzione di questi passi, sarà possibile garantire che ogni commit inserisca modifiche funzionanti. Ovviamente, quest'ultima garanzia è interamente dipendente dai test sviluppati.

6 Ciclo di vita dello sviluppo software

Il ciclo di vita dello sviluppo software è un insieme di fasi durante il quale si gestisce un particolare aspetto del processo di creazione del software stesso. Si tratta, se ben applicato, di un processo economico ed efficiente in termini temporali, che permette ai team di progettare e realizzare software di altà qualità. Durante la definizione di questo piano, gli stakeholder partecipano e stabiliscono gli obiettivi progettuali e le features che il software deve possedere.
I vantaggi della definizione di questo ciclo di vita sono:

- Maggior trasparenza delle fasi di sviluppo verso gli stakeholder coinvolti;
- Pianificazione efficiente del lavoro;
- Gestione dei costi e dei rischi.

6.1 Pianificazione

Durante la fase di pianificazione [20] vengono effettuate l'analisi costi-benefici, la stima e l'allocazione delle risorse. Il team di sviluppo raccoglie informazioni dai clienti e da figure esperte interne all'azienda e stila il documento di specifiche sui requisiti del software. Quest'ultimo stabilisce le aspettative progettuali e definisce gli obiettivi comuni utili alla pianificazione del progetto.

6.2 Scrittura codice

Durante questa fase [20] avviene l'analisi dei requisiti e si stimano le migliori strategie per l'implementazione di nuove features all'interno del software. Oltre alla scrittura del codice, vengono definite anche le scelte tecnologiche e gli strumenti di sviluppo.

6.3 Testing

In questa fase vengono combinate l'automazione e i test manuali per verificare la presenza di bug all'interno del codice [20]. Una buona fase di test dovrebbe essere effettuata parallelamente alla fase di sviluppo, in modo da riconoscere e risolvere gli errori quanto più velocemente possibile. [6]
Di seguito, un approfondimento sulle principali tipologie di test effettuati durante lo sviluppo software.

6.3.1 Test unitario

Un test si dice unitario quando si occupa di verificare l'effettiva funzionalità di una singola unità di software. Definiamo «unità software» il minimo componente di un programma avente un funzionamento autonomo, ossia indipendente da altre unità. Essendo test relativamente leggeri e veloci da eseguire, essi vengono facilmente automatizzati all'interno della pipeline CI/CD, in modo da proseguire con le fasi successive solamente se le singole unità risultano tutte funzionanti.

6.3.2 Test di integrazione

Un test di integrazione è un tipo di test atto a verificare l'integrazione tra due componenti software e il loro effettivo funzionamento. Il loro scopo è quello di verificare che, assunto il funzionamento delle singole, le unità software si integrino bene tra loro, ritornando il risultato atteso.

6.3.3 Test End-to-End

Il testing end-to-end consiste nel controllare il funzionamento del software nella sua interezza. Consiste nel simulare il comportamento di un utente reale, verificando sia la funzionalità dei componenti dell'interfaccia grafica, sia dei componenti tecnici. Il loro scopo è verificare l'assenza di bug una volta che tutti i componenti sono integrati tra loro.
Si tratta di una modalità di testing particolarmente importante dato che, un po' come avviene nei test di integrazione, una singola unità di software potrebbe comportarsi correttamente se utilizzata singolarmente, ma integrarsi male con gli altri componenti.

6.3.4 Test di carico

Il termine test di carico, o «load testing», indica la pratica di stressare il software in modo da verificarne il comportamento in presenza di altissime richieste. Le richieste appaiono come utenti virtuali che utilizzano il software testato. Dato il tipo di test, è particolarmente utile in software che prevedono un utilizzo multi-utente, come ad esempio software distribuito su cloud o, comunque, con architettura client/server. I principali componenti, nonchè le principali cause di rallentamento di un software, che si vogliono testare sono:

- Server di appoggio;

- Load balancing tra i diversi server della rete;
- Database;
- Rete.

6.3.5 Smoke test

Si tratta di una particolare sotto-suite di test che verificano il funzionamento delle principali funzionalità del software. Tramite il loro utilizzo è possibile assicurare che, nonostante possano sorgere altre problematiche software, ad esempio il fallimento di altre suite di test, le principali funzionalità del software sono comunque funzionanti.

6.4 Distribuzione

Una volta che si è assicurata l'assenza di bug (o assenza di bug considerati *gravi*), è possibile procedere con la distribuzione del software. Distribuire un software vuol dire renderlo accessibile agli utilizzatori, con lo scopo principale di raccogliere feedback e pareri per migliorare iterativamente l'esperienza. A seconda della tipologia di software, possono essere presenti diversi ambienti di distribuzione, anche detti ambienti di rilascio, su cui distribuire il prodotto. Avere più ambienti a disposizione può essere utile per vari motivi:

- Isolamento dei dati e delle features: In questo modo si avranno spazi differenti su cui effettuare e implementare le modifiche. Inoltre, può essere utile per avere a disposizione un ambiente su cui eseguire i test, solitamente questo ambiente prende il nome di «testing environment»;
- Feedback dei tester: Nello sviluppo di software molto grandi troviamo la presenza di personale adibito allo svolgimento di test, raccolta degli esiti e comunicazione dei risultati agli sviluppatori. È particolarmente utile avere un ambiente complementamente dedicato ai test, in modo da non rischiare di effettuare modifiche ai dati degli utenti reali;
- Sicurezza in fase di rilascio: Nel caso ci fossero problematiche nel codice che deve essere rilasciato, distribuire il codice negli ambienti intermedi può essere utile per evitare di rilasciare codice non funzionamente direttamente in produzione;

Gli ambienti di rilascio più diffusi sono solitamente tre, e sono:

- Ambiente di testing: Solitamente utilizzato dagli sviluppatori stessi per controllare il funzionamento del codice da essi inserito;
- Ambiente di staging: Ambiente spesso utilizzato come pre-produzione. Il suo scopo è quello di ricevere codice pronto per essere distribuito in produzione, avendo superato l'ostacolo dell'ambiente di testing, ma che necessita dell'approvazione del team di tester;
- Ambiente di produzione: Ambiente contenente il codice disponibile agli utenti, o utilizzatori del servizio. Il codice, e relativo software, contenuto al suo interno è da intendersi come stabile.

A seconda del software e dell'azienda produttrice, possono essere presenti diverse politiche di rilascio, che definiscono le modalità con cui il software verrà distribuito nei vari ambienti di rilascio e con cui gli utenti riceveranno le nuove modifiche.

6.4.1 Canary Deployment

Questa strategia di rilascio consente di migrare una piccola parte di utenti verso la nuova versione, che sta attualmente venendo rilasciata, mantenendo la maggior parte dell'utenza sulla vecchia versione.
Si tratta di un'ottima strategia in quanto permette di provare su scala ridotta le funzionalità della nuova versione evitando, in presenza di errori, che tutti risentano dei problemi.
Una volta aver ottenuto risultati soddisfacenti dalla canary, possiamo completare il rilascio spostando tutta l'utenza verso la nuova versione. In caso, invece, ci fossero problemi, effettuare il «rollback» sarà un'operazione semplice: basterà, infatti, spostare l'intero traffico verso la vecchia versione.

6.4.2 Blue/Green Deployment

Si tratta di una strategia di rilascio in cui vengono definiti due ambienti inizialmente identici: l'ambiente «blue» e l'ambiente «green».
Al momento del rilascio, esso verrà distribuito sull'ambiente green. I tester potranno effettuare controlli e verifiche direttamente su quest'ultimo, evitando di interferire con l'ambiente principale. Una volta soddisfatti dei risultati presenti sull'ambiente green, l'intero traffico verrà re-direzionato verso l'ambiente green.
Si tratta di una strategia di rilascio ideale per progetti in cui si vuole evitare

che gli utenti, anche solamente una piccola parte di essi, possano accedere alla nuova versione del software senza i dovuti accertamenti del team di testing.

6.4.3 Rolling Deployment

Si tratta di una strategia in cui le varie istanze di un'applicazione vengono aggiornate alla nuova versione.
Il vantaggio nell'uso di questa strategia risiede nella semplicità di implementazione e comodità di rollback, non dovendo gestire ambienti differenti o traffic routing.
A causa della moltitudine di istanze, il loro aggiornamento non avverrà contemporaneamente. Data questa situazione, i servizi dell'applicativo dovranno fornire supporto sia alla vecchia sia alla nuova versione, in modo da non causare problemi agli utenti, indipendentemente dall'istanza a cui appartengono.

6.5 Monitoraggio

Dopo aver rilasciato una nuova versione, l'operazione successiva prende il nome di monitoraggio. Si tratta di un'operazione fondamentale per assicurare il corretto funzionamento dell'applicativo e garantire un grado di prestazioni adeguate.
Si noti come la fase di monitoraggio presenti diverse sfaccettature, a seconda del tipo di processo che si vuole monitorare.
Tra le principali, troviamo:

- Monitoraggio dei costi: Utile soprattutto in ambito cloud serverless o servizi «pay as you go», per ottenere una stima dei costi che dovranno essere sostenuti;

- Monitoraggio delle risorse: Tenere traccia dell'utilizzo di risorse delle macchine all'interno dell'architettura è utile per identificare e prevenire bottleneck, nel caso in cui il numero di richieste aumentasse improvvisamente. Anche la rete rientra tra i parametri monitorati all'interno di questa tipologia, in modo da evitare network failures o bottleneck di rete.

- Monitoraggio degli errori: Si tratta dell'osservazione costante di errori a livello applicativo tramite allarmi (o altri sistemi di controllo) presenti all'interno dell'architettura. Esse permettono, in modo automatico, di identificare problemi emersi in fase di utilizzo del prodotto da parte di

un utente. In questo modo si avrà una segnalazione con relativo codice di errore e descrizione, utile per una successiva investigazione manuale da parte di team specializzati.

6.6 DevOps nel ciclo di sviluppo software

Le fasi appena descritte compongono un modello di sviluppo sequenziale, detto «waterfall», o «a cascata». Il principale aspetto negativo di questo modello è proprio la sequenzialità: infatti, per passare alla fase successiva è necessario terminare quella attuale. Questo crea gravi problemi in caso si verificasse la necessità di intervenire sui processi di sviluppo (si pensi, ad esempio, all'inserimento di nuove feature nel documento dei requisiti). Si è dunque resa necessaria l'applicazione di nuovi metodi di sviluppo iterativi, che permettano di ritornare sui propri passi e favoriscano la modifica di alcuni aspetti dello sviluppo, se necessario. Ciò ha favorito una costante diffusione delle pratiche DevOps all'interno dei processi di sviluppo.
Un esempio di applicazione di un modello di sviluppo iterativo è ben visibile pensando alla fase post-deployment: a differenza delle procedure di sviluppo passate, il rilascio viene eseguito fin dalle fasi embrionali del progetto, seguito da fasi di sviluppo atte al perfezionamento del codice dell'applicativo, in modo da apportare un continuo e costante miglioramento al prodotto.

7 Cloud

Sebbene la definizione di cloud possa sembrare astratta, esso può essere definito come una rete di server, collegati tra loro, in grado di archiviare dati, eseguire applicazioni o distribuire servizi, che possono essere acceduti tramite qualsiasi dispositivo e da qualsiasi locazione [18].
La sua adozione è in continua crescita e ciò è dovuto alla comodità per i clienti di non dover installare alcun software aggiuntivo, mentre le imprese non devono occuparsi della gestione dei server fisici e dell'infrastruttura sottostante.
Il cloud computing è la pratica di eseguire un servizio in maniera «virtuale» su un server, fornito dal cloud provider. Come detto, il servizio viene eseguito virtualmente, ossia in un ambiente simulato e riservato esclusivamente al cliente che ne sta usufruendo, che verrà trattato come un computer fisico con hardware dedicato.
Le aziende usano diversi metodi per distribuire le risorse cloud [31]:

- Cloud pubblico, che condivide le risorse e offre servizi al pubblico tramite Internet;
- Cloud privato, che non è condiviso e offre servizi tramite una rete interna privata, in genere ospitata in locale;
- Cloud ibrido, che condivide servizi tra cloud pubblici e privati a seconda dello scopo.

7.1 Modelli di servizi cloud

A seconda della tipologia di servizio fornito, è possibile definire diverse categorie, o modelli, di servizi cloud, che verranno presentati di seguito. Sebbene le definizioni cambino da azienda ad azienda, tradizionalmente ci si riferisce a tre modelli principali per il cloud computing.
Ogni modello rappresenta un aspetto diverso del cloud computing.

7.1.1 Infrastructure as a Service

Questo tipo di servizio di cloud computing include principalmente la fornitura di server e data center, utili nella memorizzazione di grandissime quantità di dati. Esempi di IaaS sono soluzioni di backup centralizzate, sistemi per il «disaster recovery» e piattaforme di hosting per siti web.
Questa soluzione è pensata appositamente per amministratori di rete, aventi il

compito di gestire l'infrastruttura sottostante e i dati che devono essere ospitati all'interno.

7.1.2 Platform as a Service

Si tratta di un modello di cloud computing il cui scopo è quello di fornire agli sviluppatori tutti gli strumenti necessari per lo sviluppo di nuove applicazioni, che potranno (o meno) essere distribuite su piattaforme cloud. Esempi di strumenti forniti all'interno di questa soluzione sono:

- Ambienti di sviluppo;
- Database;
- Web server;
- Sistemi operativi, completi di librerie utili allo sviluppo degli applicativi.

I vantaggi di questo approccio sono una maggior velocità nello sviluppo e tempi ridotti in fase di deployment. Tuttavia, il problema principale di questa soluzione è la poca flessibilità e personalizzazione che gli sviluppatori possiedono in fase di selezione degli strumenti, che ricadrà unicamente sulle scelte del fornitore.

7.1.3 Software as a Service

Con Software as a Service (SaaS) si fa riferimento alle applicazioni web che sempre più spesso vengono distribuite dai relativi fornitori, accessibili da browser e da qualsiasi dispositivo, senza bisogno di dover installare alcun software aggiuntivo.
Un esempio di SaaS è la posta elettronica web, che permette la comunicazione senza la necessità di dover configurare particolari impostazioni di rete all'interno del sistema locale.
Questo modello di cloud computing è pensato appositamente per gli utenti finali.

7.2 Vantaggi e svantaggi del cloud computing

Tra i vantaggi del cloud computing troviamo sicuramente la comodità di non dover gestire aspetti esterni all'applicativo stesso, riducendo sensibilmente i tempi di sviluppo, nel caso di sviluppatori, o di utilizzo, nel caso di utenti

finali.
Ciò porta con sé anche degli svantaggi, tra cui un maggior costo relativo all'utilizzo delle risorse cloud, necessità di prestare particolare attenzione agli aspetti di cybersecurity e trattamento dei dati, e garantire un'adeguata continuità del servizio.

7.3 Privacy e Cybersecurity nel cloud

Quando si utilizza un servizio di cloud computing per memorizzare dati personali o sensibili, l'utente è esposto a potenziali problemi di violazione della privacy. I dati personali vengono memorizzati nelle cosiddette «server farms», ossia server con il principale scopo di immagazzinare al loro interno un'enorme quantità di dati proveniente dagli utilizzatori del servizio cloud, appartenenti ad aziende che spesso risiedono in uno stato differente da quello dell'utente. Il cloud provider, in caso di comportamento scorretto o malevolo, potrebbe accedere ai dati personali per eseguire ricerche di mercato e profilazione degli utenti. È bene però ricordare che il cloud provider, ossia il fornitore dei servizi cloud e colui che raccoglie i dati, solitamente non li immagazzina in server di sua proprietà, ma si affida bensì a provider di terze parti.
Nonostante ciò, il compito principale in materia di trattamento dei dati da parte del cloud provider sarà quello di trasferire al provider di terze parti le modalità di trattamento dei dati che l'utente finale gli ha comunicato, in modo che (l'utente) abbia sempre il controllo sulla gestione dei propri dati.
La situazione si fa ancora più complessa a causa delle comunicazioni wireless tra i vari dispositivi. Infatti, nonostante i numerosi tentativi dei provider di fornire servizi di crittografia dei dati, le truffe informatiche e la pirateria sono fenomeni sempre più diffusi.

7.4 Serverless

Il serverless computing [33] è un modello di sviluppo cloud che consente agli sviluppatori di creare ed eseguire applicazioni senza gestire i server. Nonostante il nome «serverless» possa far pensare alla non-presenza di alcun server all'interno dell'infrastruttura, in realtà esso indica la non necessità di dover gestire alcun server su cui verrà eseguito l'applicativo. La manutenzione, la scalabilità e il «provisioning» delle risorse è completamente gestito dal cloud provider.
Servizi con un business model simile a quello serverless sono stati distribuiti

intorno al 2008, da Google, con «Google App Engine», nonostante sia necessario attendere il 2014, con «AWS Lambda», per avere un modello di sviluppo decisamente più popolare tra la comunità.
L'aspetto principale dell'architettura serverless è la sua scalabilità completamente automatica e gestita dal provider stesso, che addebiterà un costo in base al numero di richieste ricevute dall'applicativo. Questo tipo di soluzione prende il nome di «pay as you go».

7.5 Infrastructure as Code

Come suggerito dal nome, con «Infrastructure as Code» si indica il processo di definizione dell'architettura cloud tramite codice. [19]
Rispetto ad un approccio classico, usare il codice ed i relativi costrutti di programmazione permette di avere maggior controllo e flessibilità, consentendo inoltre di applicare le best practices DevOps. Inoltre, permette di avere una struttura facilmente modificabile e «versionabile».
Poter utilizzare un linguaggio di programmazione per gestire l'infrastruttura è un grosso vantaggio in fase di sviluppo: infatti, permetterà agli sviluppatori di ri-utilizzare un'infrastruttura precedentemente definita ed usarla come base per progetti futuri. La possibilità di versionare l'infrastruttura permette inoltre di poter tornare ad una versione precedente, nel caso sorgessero problemi a seguito di nuove modifiche.

8 AWS

AWS, acronimo di Amazon Web Services, è un servizio di cloud computing fornito da Amazon. È stato fondato nel 2006 e al momento fornisce servizi a circa 26 regioni geografiche.
Il quantitativo di servizi offerti dalla piattaforma è nell'ordine delle centinaia e permettono al cliente di avere a disposizione qualsiasi tipo di risorsa di cui esso abbia bisogno.

8.1 Risorse AWS

Di seguito, una breve introduzione alle principali risorse messe a disposizione da AWS, utili per la creazione dell'infrastruttura.

8.1.1 Lambda Function

Una Lambda Function [8] rappresenta l'elemento centrale del servizio Lambda di AWS.
Si tratta di una funzione che, al momento della chiamata, effettua computazioni e ritorna una «response», ossia il risultato dell'esecuzione.
Da un punto di vista strutturale altro non è che una classica funzione scritta tramite il linguaggio di programmazione utilizzato dallo sviluppatore.
Sono presenti diverse modalità con cui chiamare una lambda function, di seguito:

- Chiamata da interfaccia web AWS: Visitando l'apposita sezione dal sito web di AWS, sarà possibile chiamare la lambda function specificando il metodo HTTP attraverso cui si vuole contattare la funzione e il «payload», ossia il corpo della richiesta;

- Chiamata da un'altra lambda function: All'interno del codice della lambda function è possibile chiamare un'altra lambda function similmente a come si chiamerebbe una classica funzione nel linguaggio di programmazione in uso;

- Binding con API Gateway: Il metodo principale con cui chiamare una lambda function è facendo il binding con un percorso nell'URL dell'API Gateway. Al momento del deploy dell'infrastruttura, l'API Gateway

verrà rilasciato e verrà reso disponibile un URL per contattarlo. Specificando un percorso aggiuntivo, sarà possibile collegare una lambda function, esponendola al traffico esterno. Ulteriori informazioni relative ad API Gateway nell'apposita sezione.

8.1.2 S3 Bucket

Un S3 Bucket [11], abbreviazione di Simple Storage Service, è una risorsa utile alla memorizzazione di dati. Non si tratta dell'unica risorsa atta a questo scopo: il suo focus, infatti, è sull'archiviazione di dati di medio-grandi dimensioni (come ad esempio foto, video, binari, documenti) e la facilità di accesso a quest'ultimi.

8.1.3 DynamoDB

DynamoDB [17] è un servizio di database NoSQL incentrato sulla rapidità e scalabilità. È un servizio completamente gestito da AWS: ciò significa che la scalabilità è automaticamente effettuata da AWS e basata sul numero di richieste ricevute.
Si tratta di una scelta ideale nel caso si volessero memorizzare dati strutturati o semi-strutturati.

8.1.4 API Gateway

API Gateway [12] è un servizio completamente gestito da AWS che consente e semplifica la creazione, manutenzione e pubblicazione delle API. Con API facciamo riferimento a servizi eseguiti da un applicativo che possono essere raggiunti, in questo caso, tramite API Gateway.
Quest'ultimo, inoltre, si occupa anche di gestire:

- Traffic Routing;
- Permessi di accesso alle risorse;
- Monitoraggio e versionamento delle API.

8.2 Servizi AWS

Verranno ora discussi alcuni tra i principali servizi forniti da AWS, nonchè quelli utilizzati per la realizzazione del framework.

8.2.1 AWS Cloud Development Kit

AWS Cloud Development Kit [13], anche noto come AWS CDK, è un framework open source che consente di definire l'infrastruttura cloud tramite linguaggi di programmazione. Esso rappresenta un'evoluzione del concetto di «Infrastructure as Code», dal momento che il codice non è YAML, JSON o XML, ma bensì vero e proprio codice che fornisce supporto ai costrutti iterativi e condizionali propri della programmazione.
AWS CDK è formato da due componenti principali:

- AWS CDK Construct Library: Insieme di codici modulari e riutilizzabili, chiamati costrutti, modificabili ad-hoc ed integrabili all'interno di un progetto per sviluppare rapidamente l'infrastruttura. L'obiettivo di AWS CDK Construct Library è quello di ridurre la complessità necessaria a definire l'infrastruttura cloud di un progetto;

- AWS CDK Toolkit: Strumento CLI per l'interazione con le applicazioni CDK. Un'applicazione CDK è un progetto creato con l'apposito comando da terminale. Con l'uso di AWS CDK Toolkit sarà possibile interrogare l'infrastruttura e ottenere informazioni su quest'ultima.

8.2.2 AWS Lambda

AWS Lambda [9] è il servizio di calcolo basato su serverless che permette agli sviluppatori di gestire la propria applicazione su cloud senza la necessità di dover gestire i server sottostanti.
AWS Lambda esegue codice su un'infrastruttura di calcolo ad altissima disponibilità in risposta ad eventi, ossia richieste HTTP ai relativi endpoint, gestendo autonomamente:

- Il provisioning delle risorse;

- Manutenzione del server e del sistema operativo;

- Scalabilità delle risorse.

8.2.3 AWS CloudFormation

AWS CloudFormation è un servizio di Infrastructure as Code che consente di rappresentare e configurare tramite codice le risorse all'interno della nostra architettura. In questo modo, sarà più facile interrogarla e ottenere informazioni

sullo stato e disponibilità delle risorse stesse. [14]
AWS CDK e CloudFormation lavorano a stretto contatto nella creazione di progetti tramite CDK. Infatti, al momento del deployment di un'infrastruttura CDK, il codice viene convertito in sintassi CloudFormation, per una maggior integrazione con gli altri servizi AWS.
I componenti principali di AWS CloudFormation sono:

- Modelli: File testuali in formato YAML o JSON che permettono la definizione e rappresentazione di risorse AWS. Ad esempio, permettono di definire un DynamoDB, che useremo all'interno della nostra architettura;

- Stacks: Insiemi di risorse che, unite tra loro, comporranno la nostra architettura.

8.2.4 AWS CloudWatch

AWS CloudWatch [15] è un servizio che monitora le risorse AWS. Nel ciclo di vita del software, AWS CloudWatch gestisce la fase di «monitoring», permettendo all'utente di avere informazioni aggiuntive sullo stato delle risorse. Occupandosi di monitoraggio, all'interno di AWS CloudWatch è possibile creare:

- Dashboard: Insieme di widget di due differenti tipi: testuali e a grafi. Un esempio di widget testuale può essere un header per riconoscere una determinata dashboard dalle altre, mentre un widget a grafo è un grafo che rappresenta una determinata metrica, con una determinata statistica in un determinato lasso temporale. Ad esempio, si potrebbe essere interessati a monitorare il numero di errori provocati da una «Lambda function» o la sua durata media (per verificare eventuali ottimizzazioni). Nel primo caso, la metrica sarà «Errors» e la statistica sarà «Sum», nel secondo caso la metrica sarà «Duration» e la statistica sarà «Average». Le dashboard sono dunque utili per raggruppare sotto un'unica schermata tutte le metriche, e relative statistiche, che vogliamo monitorare dopo il deployment di un progetto.

- Allarmi: Risorse in ascolto su una determinata metrica e statistica che, al superamento della soglia indicata, entrano in stato di allarme. È possibile configurare secondo le proprie necessità le azioni da compiere una volta che l'allarme si sarà attivata: per esempio, è possibile impostare l'invio di una notifica su diversi canali (quali email, server Slack, SMS, ...)

8.2.5 AWS CodeDeploy

AWS CodeDeploy [16] è un servizio di distribuzione che permette di automatizzare il rilascio di applicazioni.
Il vantaggio principale dell'utilizzo di AWS CodeDeploy, che si occupa della fase di deployment nel ciclo di sviluppo software, è l'automazione. Infatti, con uno specifico focus sull'ambito serverless e la piattaforma di calcolo Lambda, CodeDeploy può automatizzare il rilascio di nuove versioni di Lambda functions che implementano un «DeploymentGroup», ossia un particolare costrutto che definisce con quale strategia di rilascio gestire la funzione stessa. Ad esempio, si immagini un'architettura composta da due Lambda functions: «A» e «B». Si vorrebbe gestire il rilascio di A usando «Canary» come release strategy, mentre si vorrebbe usare «Linear» per la funzione B. Gestendo i parametri del DeploymentGroup di ogni funzione, sarà possibile gestire in maniera granulare la durata temporale e il bacino di utenza che verrà influenzato dalla specifica tipologia di rilascio.
AWS CodeDeploy e AWS CloudWatch presentano un fattore di integrazione molto alto. Difatti, sarà possibile definire un'allarme CloudWatch all'interno del DeploymentGroup di AWS CodeDeploy, in modo tale che il deployment venga annullato se durante la fase di transizione da una versione ad un'altra l'allarme venisse attivata. Anche in questa fase l'utente avrà piena libertà nella gestione del comportamento di AWS CodeDeploy, nel caso il deployment venisse annullato. Il comportamento di default sarà un rollback alla versione precedente, che riporterà tutti gli utenti alla versione stabile. L'obiettivo di questa pratica è quello di garantire un «downtime», ossia un periodo di irraggiungibilità del servizio, quanto più basso possibile.

9 BlueDAC

Nel panorama odierno dello sviluppo software, l'automazione è diventato un elemento imprescindibile per ottimizzare i processi e massimizzare l'efficienza. È proprio in risposta alla costante diffusione del cloud computing e relativi servizi che si è reso necessario lo sviluppo di BlueDAC, un framework pensato per automatizzare le diverse fasi DevOps relative allo sviluppo di progetti cloud serverless su AWS. BlueDAC si propone come strumento per i team DevOps che desiderano semplificare e velocizzare il processo di inizializzazione e successiva gestione di progetti cloud serverless.
Prima di procedere con lo sviluppo, è stata condotta un'analisi dello stato dell'arte, per individuare soluzioni simili sul mercato. Le principali soluzioni emerse sono state AWS Cloud Development Kit, Serverless Framework e Pulumi, quest'ultimi due noti per semplificare lo sviluppo e il rilascio di architetture cloud: Serverless, con maggior esperienza e maturità sul mercato, offre supporto al solo AWS, mentre Pulumi è una soluzione più recente che offre un supporto multi-cloud.
Ad ogni modo, la scelta finale è ricaduta su AWS Cloud Development Kit, garantendo così un supporto ufficiale per l'ambiente AWS, per il quale BlueDAC è stato specificamente progettato.

9.1 Installazione

BlueDAC è stato scritto interamente in Python ed è liberamente accessibile da PyPI, la principale piattaforma per la distribuzione di librerie Python. Per installarlo, è possibile utilizzare «pip», il package manager predefinito di Python, eseguendo il seguente comando:

```
pip install bluedac
```

9.2 Ciclo di vita di un progetto BlueDAC

Verrà di seguito descritto il ciclo di vita di un progetto gestito tramite BlueDAC. Verranno approfondite le fasi che compongono questo processo, definendo l'ordine con cui esse si susseguono.

9.2.1 Inizializzazione del progetto

L'inizializzazione del progetto avviene tramite il comando:

`dac init`

Il comando mostrerà un'interfaccia interattiva da terminale che guiderà l'utente attraverso una serie di passaggi per la raccolta di informazioni necessarie alla configurazione del progetto. L'interfaccia porrà domande relative a:

- Nome del progetto;
- Linguaggio di programmazione;
- Branching strategy;
- Ambienti di rilascio.

Al termine, verrà creata una directory omonima al nome del progetto, al cui interno verrà inizializzato un repository git locale e un'infrastruttura CDK di partenza. Quest'ultima farà uso del linguaggio di programmazione selezionato dall'utente.
A distinguere un progetto BlueDAC è la presenza del file «bluedac_config.json», creato anch'esso in fase di inizializzazione, che conterrà la configurazione BlueDAC del progetto creato.
Al suo interno saranno presenti:

- Nome del progetto;
- Linguaggio di programmazione utilizzato da CDK per la definizione dell'architettura cloud;
- Branching strategy in uso;
- Valore minimo di coverage che gli unit test dovranno rispettare per essere considerati superati;
- Ambienti di rilascio;
- Ambienti di rilascio «manuali», nel caso non si volesse rilasciare automaticamente (di default, il rilascio su un ambiente è automatico);
- Release strategy di ogni ambiente di rilascio, per una maggior flessibilità e personalizzazione;
- Pipeline CI/CD in formato JSON: sarà fornita una versione iniziale della pipeline che rispecchi la branching strategy nella sua purezza. A seconda dei requisiti aziendali e/o progettuali, la pipeline potrà essere modificata andando a gestire i jobs presenti all'interno di ogni evento, ottenendo un'elevata flessibilità per la gestione del progetto.

9.2.2 Creazione della pipeline GitLab

A seguito dell'inizializzazione del progetto, si rende necessaria la verifica della correttezza delle informazioni contenute all'interno del file «bluedac_config.json». Sulla base della branching strategy selezionata, BlueDAC fornirà, in formato JSON, una pipeline CI/CD rappresentante la strategia stessa.
La pipeline JSON sarà contenuta all'interno dell'attributo «pipeline» e conterrà, a seconda della branching strategy selezionata, opzioni differenti. Verrà di seguito fornita una breve descrizione dei possibili stage della pipeline JSON:

- release_mode: Permette di specificare la modalità di rilascio. I possibili valori sono:
  - tag: Applicando un tag ad un commit, verranno eseguiti i jobs contenuti all'interno del blocco «release».
  - branch: Indicando questa opzione, un rilascio verrà definito tale quando un commit verrà utilizzato per aprire un release branch. Se si dovesse scegliere di rilasciare tramite branch, sarà necessario indicare un parametro aggiuntivo: «branch_regex». I branch che rispetteranno quest'espressione regolare saranno identificati come release branch, e pertanto su di essi verranno eseguiti i jobs contenuti all'interno del blocco «release».

- merge_request: I jobs contenuti all'interno di questo blocco verranno eseguiti all'apertura di una merge request. Il codice che verrà sottoposto alla pipeline sarà quello proveniente dal branch sorgente. Da sottolineare come questi jobs vengano eseguiti all'apertura della merge request, indipendentemente dal suo esito;

- commit_main: Questo blocco di jobs verrà eseguito al push di un nuovo commit sul main branch (indipendentemente dal nome, verrà considerato «main branch» il branch specificato come default nell'interfaccia web di GitLab). Da sottolineare come l'esecuzione di questo blocco potrebbe avvenire sia per mezzo di un push «diretto» sul main branch, sia per una merge request approvata, avente il main branch come branch di destinazione;

- release: Questo insieme di jobs verrà eseguito in caso di rilacio, seguendo le modalità specificate dal parametro «release_mode».

Verrà ora fornita una breve descrizione dei possibili job utilizzabili per la composizione degli stage della pipeline:

- launch_integration_tests: Questo job si occupa dell'esecuzione dei test di integrazione, presenti nella directory «tests/integration»;
- launch_unit_tests: Questo job si occupa dell'esecuzione dei test unitari, presenti nella directory «tests/unit». È possibile specificare un valore minimo di coverage, che definirà l'esito dell'esecuzione dei test unitari;
- launch_load_tests: Questo job è responsabile dell'esecuzione dei test di carico, tramite l'utilizzo di Artillery;
- launch_e2e_tests: Questo job si occupa dell'esecuzione dei test end-to-end, presenti nella directory «tests/e2e».
- deploy_testing: Questo job si occupa del rilascio dello stack sull'ambiente di staging.
- deploy_staging: Questo job è responsabile del deployment dello stack sull'ambiente di staging.
- deploy_production: Questo job si occupa del rilascio dello stack sull'ambiente di produzione.

Una volta accertatosi della correttezza del file di configurazione del framework, si potrà procedere con la conversione della pipeline, in formato YAML. Essa verrà inserita all'interno del file «.gitlab_ci.yml», fornendo diretta integrazione con la piattaforma GitLab.
Per procedere con la conversione, sarà necessario eseguire il seguente comando:

```
dac confirm
```

Sarà dunque creato il file «.gitlab_ci.yml», che conterrà la pipeline CI/CD in sintassi GitLab.

9.2.3 Generazione dei test

BlueDAC supporta la creazione di test unitari, test di integrazione, test end-to-end e load test. I test creati saranno dei «boilerplate», ossia conterranno una struttura generica che necessiterà dell'intervento manuale dello sviluppatore per adattarlo alle proprie necessità. L'uso di questo tipo di test permette di incrementare significativamente la velocità di scrittura di test.
Per procedere con la creazione di una specifica suite di test, il comando sarà:

```
dac test generate --suite
```

Attualmente, «suite» dovrà rientrare tra le seguenti opzioni:

- unit;
- integration;
- e2e;
- load;

Omettendo il parametro «–suite», verranno generate tutte le suite di test sopra elencate.
I test saranno inseriti all'interno della directory «tests/», presente nella root del progetto, all'interno di una directory omonima alla suite di test.
La generazione di test unitari necessita di maggior approfondimento: BlueDAC farà uso del comando «cdk synth», fornito da AWS CDK, per ottenere l'architettura CloudFormation del progetto, in formato YAML. Tramite il parsing di quest'ultima, BlueDAC verificherà quali risorse AWS sono effettivamente utilizzate, e creerà i relativi test unitari. Quest'operazione consente di evitare la creazione di test unitari per componenti non utilizzati, che porterebbero ad errori in fase di esecuzione.
Durante la creazione di un load test verrà richiesto all'utente di selezionare lo stack di riferimento: l'operazione è necessaria in quanto ogni stack rappresenta un differente ambiente di rilascio, e possiede pertanto un differente API Gateway (e relativo URL). Avendo noto lo stack di interesse, verrà recuperato l'URL dell'API Gateway di riferimento.
Il nome del file generato sarà «artillery-env.yaml», in cui «env» indica l'ambiente di rilascio al quale il test fa riferimento. Il file sarà in sintassi YAML ed utilizzabile tramite Artillery, sul quale sarà possibile intervenire al fine di gestire parametri aggiuntivi, come il numero di utenti virtuali simulati, il quantitativo di richieste da effettuare e l'eventuale sequenza di URL da interrogare. Quest'ultima modalità si rivela particolarmente efficace per verificare le performance dell'applicativo all'interno di scenari realistici.

9.2.4 Esecuzione dei test

L'esecuzione dei test è supportata dalla libreria «pytest». Tramite l'uso dell'apposito comando, BlueDAC fornirà un'interazione più rapida e comoda rispetto quella di libreria.

Per procedere con l'esecuzione di una suite di test, sarà necessario eseguire il seguente comando:

```
dac test suite
```

Dove «suite» sarà una tra le seguenti opzioni:

- unit;
- integration;
- e2e;

Allo stesso modo della generazione dei test, omettendo il parametro suite verranno eseguiti tutti i test presenti all'interno della directory «tests/», ricorsivamente.
Per l'esecuzione di un test di carico si dovrà interagire direttamente con Artillery, tramite il seguente comando:

```
artillery run test_file
```

Dove «test_file» rappresenta la posizione in cui è memorizzato il load test che si intende eseguire.

9.3 Risorse aggiuntive

Oltre alle risorse base fornite dalla libreria «aws_cdk», BlueDAC introduce un insieme di risorse aggiuntive. Lo scopo di queste risorse è quello di ampliare le funzionalità di base, fornendo un livello superiore di automazione e velocità di applicazione delle risorse stesse agli stack CDK.

9.3.1 Bluedac_Lambda

Estensione della classe «aws_cdk.aws_lambda.Function», rappresentante una Lambda Function.
Essa presenta l'implementazione del metodo «apply_deployment_strategy», che applica la deployment strategy indicata nel file di configurazione del framework alla Lambda Function chiamante. Ciò è possibile tramite il parsing del file di configurazione di BlueDAC: il metodo riceverà in input il parametro «environment», che conterrà l'ambiente di riferimento dello stack attuale, e lo utilizzerà come chiave dell'attributo «release_strategy», definito nel file di configurazione. L'oggetto recuperato sarà un dizionario contenente i seguenti parametri:

- name: Nome della release strategy;
- interval: Durata dello shift;
- percentage: Percentuale dell'utenza spostata durante il deployment.

Il valore del parametro «name» identificherà la strategia che verrà applicata alla Lambda function. Le strategie attualmente supportate, identificate dai possibili valori di «name», sono le seguenti:

- canary: Essa prevede lo spostamento di una percentuale di utenza, specificata dal parametro «percentage», sulla nuova versione. Al termine dello shift, avente una durata pari al parametro «interval», tutta l'utenza verrà spostata sulla nuova versione, che diventerà quella attuale. La durata dello shift, fornita dal parametro «interval», è da intendersi in minuti;
- linear: Essa prevede lo spostamento di una percentuale di utenza, specificata dal parametro «percentage», sulla nuova versione in fase di distribuzione. Saranno presenti multipli shift, che aumenteranno il valore precedente con uno «step-rate» pari al valore del parametro «percentage». Ogni singolo shift avrà una durata pari al valore di «interval», espresso in minuti. Il deployment si considererà completato quando la percentuale di utenza spostata sulla nuova versione sarà pari, o superiore, a 100;
- all-at-once: L'intera utenza viene spostata sulla nuova versione, senza shift intermedi.

La strategia «all-at-once» sarà applicata di default, come previsto da AWS stesso, nel caso in cui il parametro «name» non possedesse un valore o avesse un valore non supportato.
Una volta recuperati i parametri relativi alla deployment strategy verrà creata, appositamente per la Lambda function in analisi, una configurazione di rilascio necessaria al servizio AWS CodeDeploy per eseguire la distribuzione.
La configurazione sarà rappresentata da un «LambdaDeploymentGroup», ossia un costrutto composto dai seguenti componenti:

- Lo stack di riferimento;
- L'alias con cui si identificherà la Lambda function, necessario per tenere traccia del versionamento di quest'ultima;

- DeploymentConfig: Costrutto avente lo scopo di applicare la deployment strategy;
- Allarme: L'allarme sarà configurata tramite una serie di parametri che dovranno essere passati come input al metodo. I parametri attesi sono:
  - La metrica da monitorare;
  - La statistica su cui monitorare la metrica;
  - La durata del periodo di monitoraggio.

In questo modo, lo sviluppatore avrà piena libertà in fase di configurazione dell'allarme, operazione fondamentale in quanto consente l'annullamento del rilascio, nel caso sorgessero errori sulla nuova versione.

9.3.2 Bluedac_APIGW

Estensione della classe «aws_cdk.aws_apigateway.RestApi», rappresentante un API Gateway.
Esso implementa il metodo «bind_lambda», che consente di velocizzare il processo di binding di una Lambda Function ad un path dell'API Gateway. Sarà necessario fornire al metodo i seguenti parametri:

- lambda_fun: La Lambda function in esame;
- method_type: Il metodo HTTP tramite cui si intende fornire l'API (ossia, la Lambda Function);
- path: Il path dell'URL tramite cui si intende raggiungere l'API.

9.3.3 Bluedac_Dashboard

Estensione della classe «aws_cdk.aws_cloudwatch.Dashboard».
Esso implementa il metodo «build_with», che consente di costruire rapidamente una dashboard ed evitare la scrittura di codice verboso e ripetitivo.
L'unico parametro necessario sarà una lista di dizionari, ognuno di essi rappresentante un «widget».
Un widget CloudWatch è una risorsa utilizzata per personalizzare le dashboard. BlueDAC supporta attualmente i seguenti widget:

- TextWidget: ossia un blocco di testo in formato Markdown. L'unica chiave contenuta all'interno del dizionario sarà «text», contenente il corpo del messaggio;

- GraphWidget: ossia un grafico illustrante una metrica. In questo caso, il dizionario conterrà le seguenti chiavi:
  - metric: La metrica che verrà rappresentata (e.g. numero di errori, numero di invocazioni, durata dell'invocazione);
  - statistic: La statistica con cui visualizzare la metrica (e.g. somma, media);
  - resource: La risorsa da cui recuperare i dati (e.g. Lambda Function, DynamoDB);
  - duration: Il range temporale in cui analizzare la metrica. L'utilizzo della classe «aws_cdk.Duration» permette di specificare l'unità temporale di riferimento (e.g. secondi, minuti, ore).

9.3.4 StackUtils

StackUtils rappresenta una classe contenente metodi statici, utilizzati per una comoda interazione con lo stack CDK di riferimento. I metodi da essa forniti sono:

- get_rs_info: Consente di recuperare i parametri della release strategy impostata per l'ambiente attuale. Quest'ultimo dovrà essere passato come parametro al metodo;
- retrieve_apigw_endpoint: Permette di recuperare l'URL dell' API Gateway dello stack attuale. Il recupero avviene tramite una query allo stack CloudFormation che fornirà i valori impostati da «CfnOutput», ossia un costrutto utilizzato per il salvataggio (e recupero) di informazioni relative allo stack ed ai suoi componenti. Sarà dunque necessario, in fase di creazione dello stack, fornire a CfnOutput il valore dell'URL dell'API Gateway. L'unico parametro atteso dal metodo è lo stack, necessario per il recupero del relativo CloudFormation.

9.4 Esempio file di configurazione BlueDAC

Verrà riportato di seguito il contenuto del file «bluedac_config.json», rappresentante la configurazione di BlueDAC per un progetto di prova.

```
{
    "project_name": "bluedac-test",
    "programming_language": "python",
```

```
"branching_strategy": "tbd",
"envs": [
    "production",
    "staging",
    "testing"
],
"manual_release_envs": ["production"],
"min_test_coverage": 75,
"release_strategy": {
    "production": {
        "name": "",
        "interval": 0,
        "percentage": 0
    },
    "staging": {
        "name": "canary",
        "interval": 2,
        "percentage": 30
    },
    "testing": {
        "name": "linear",
        "interval": 1,
        "percentage": 25
    }
},
"pipeline": {
    "release_mode": "tag",
    "commit_fb": [
        "launch_unit_tests",
        "deploy_testing",
        "launch_load_tests"
    ],
    "merge_request": [
        "launch_integration_tests"
    ],
    "commit_main": [
        "launch_unit_tests",
        "deploy_testing",
        "launch_integration_tests",
        "deploy_staging",
        "launch_e2e_tests"
    ],
    "release": [
        "launch_unit_tests",
        "launch_integration_tests",
```

```
            "launch_e2e_tests",
            "deploy_production"
        ]
    }
}
```

La pipeline CI/CD, per la piattaforma GitLab, che verrà prodotta dalla precedente configurazione, sarà la seguente:

```
launch_unit_tests_commit_fb:
    stage: test
    image: alpine:latest
    script:
        - apk add --update python3 py3-pip jq
        - python -m venv .venv
        - source .venv/bin/activate
        - python -m pip install -r requirements.txt
        - export TEST_COVERAGE=$(jq .min_test_coverage bluedac_config.json)
        - python -m pytest --cov-fail-under=$TEST_COVERAGE --cov=resources
        ↪  tests/unit/*/*
    rules:
        - if: $CI_PIPELINE_SOURCE == "push" && $CI_COMMIT_BRANCH != "main"
deploy_testing_commit_fb:
    stage: deploy
    image: alpine:latest
    environment:
        name: testing
    script:
        - apk add --update nodejs npm
        - npm install -g aws-cdk
        - apk add --update python3 py3-pip
        - python -m venv .venv
        - source .venv/bin/activate
        - python -m pip install -r requirements.txt
        - export CDK_DEPLOY_ENV=testing
        - echo "CDK_DEPLOY_ENV=testing" >> deploy.env
        - cdk deploy "*-$CDK_DEPLOY_ENV" --require-approval never
    artifacts:
        reports:
          dotenv: deploy.env
    needs:
        - launch_unit_tests_commit_fb
    rules:
        - if: $CI_PIPELINE_SOURCE == "push" && $CI_COMMIT_BRANCH != "main"
launch_load_tests_commit_fb:
```

```
    stage: deploy
    image: alpine:latest
    script:
        - apk add --update python3 py3-pip nodejs npm
        - npm install -g artillery@latest
        - python -m venv .venv
        - source .venv/bin/activate
        - python -m pip install -r requirements.txt
        - artillery run "artillery-$CDK_DEPLOY_ENV.yml"
    needs:
        - launch_unit_tests_commit_fb
        - deploy_testing_commit_fb
    rules:
        - if: $CI_PIPELINE_SOURCE == "push" && $CI_COMMIT_BRANCH != "main"
launch_integration_tests_merge_request:
    stage: deploy
    image: alpine:latest
    script:
        - apk add --update python3 py3-pip
        - python -m venv .venv
        - source .venv/bin/activate
        - python -m pip install -r requirements.txt
        - python -m pytest --cov=resources tests/integration/*
    rules:
        - if: $CI_PIPELINE_SOURCE == 'merge_request_event' &&
        ↪  ($CI_MERGE_REQUEST_TARGET_BRANCH_NAME == 'main' ||
        ↪  $CI_MERGE_REQUEST_TARGET_BRANCH_NAME == 'develop')
launch_unit_tests_commit_main:
    stage: test
    image: alpine:latest
    script:
        - apk add --update python3 py3-pip jq
        - python -m venv .venv
        - source .venv/bin/activate
        - python -m pip install -r requirements.txt
        - export TEST_COVERAGE=$(jq .min_test_coverage bluedac_config.json)
        - python -m pytest --cov-fail-under=$TEST_COVERAGE --cov=resources
        ↪  tests/unit/*/*
    rules:
        - if: $CI_PIPELINE_SOURCE == "push" && $CI_COMMIT_BRANCH == "main"
deploy_testing_commit_main:
    stage: deploy
    image: alpine:latest
    environment:
        name: testing
```

```
    script:
        - apk add --update nodejs npm
        - npm install -g aws-cdk
        - apk add --update python3 py3-pip
        - python -m venv .venv
        - source .venv/bin/activate
        - python -m pip install -r requirements.txt
        - export CDK_DEPLOY_ENV=testing
        - echo "CDK_DEPLOY_ENV=testing" >> deploy.env
        - cdk deploy "*-$CDK_DEPLOY_ENV" --require-approval never
    artifacts:
        reports:
          dotenv: deploy.env
    needs:
        - launch_unit_tests_commit_main
    rules:
        - if: $CI_PIPELINE_SOURCE == "push" && $CI_COMMIT_BRANCH == "main"
launch_integration_tests_commit_main:
    stage: deploy
    image: alpine:latest
    script:
        - apk add --update python3 py3-pip
        - python -m venv .venv
        - source .venv/bin/activate
        - python -m pip install -r requirements.txt
        - python -m pytest --cov=resources tests/integration/*
    needs:
        - launch_unit_tests_commit_main
        - deploy_testing_commit_main
    rules:
        - if: $CI_PIPELINE_SOURCE == "push" && $CI_COMMIT_BRANCH == "main"
deploy_staging_commit_main:
    stage: deploy
    image: alpine:latest
    environment:
        name: staging
    script:
        - apk add --update nodejs npm
        - npm install -g aws-cdk
        - apk add --update python3 py3-pip
        - python -m venv .venv
        - source .venv/bin/activate
        - python -m pip install -r requirements.txt
        - export CDK_DEPLOY_ENV=staging
        - echo "CDK_DEPLOY_ENV=staging" >> deploy.env
```

```
        - cdk deploy "*-$CDK_DEPLOY_ENV" --require-approval never
    artifacts:
        reports:
          dotenv: deploy.env
    needs:
        - launch_unit_tests_commit_main
        - deploy_testing_commit_main
        - launch_integration_tests_commit_main
    rules:
        - if: $CI_PIPELINE_SOURCE == "push" && $CI_COMMIT_BRANCH == "main"
launch_e2e_tests_commit_main:
    stage: deploy
    image: alpine:latest
    script:
        - apk add --update python3 py3-pip
        - python -m venv .venv
        - source .venv/bin/activate
        - python -m pip install -r requirements.txt
        - python -m pytest --cov=resources tests/e2e/*
    needs:
        - launch_unit_tests_commit_main
        - deploy_testing_commit_main
        - launch_integration_tests_commit_main
        - deploy_staging_commit_main
    rules:
        - if: $CI_PIPELINE_SOURCE == "push" && $CI_COMMIT_BRANCH == "main"
launch_unit_tests_release:
    stage: test
    image: alpine:latest
    script:
        - apk add --update python3 py3-pip jq
        - python -m venv .venv
        - source .venv/bin/activate
        - python -m pip install -r requirements.txt
        - export TEST_COVERAGE=$(jq .min_test_coverage bluedac_config.json)
        - python -m pytest --cov-fail-under=$TEST_COVERAGE --cov=resources
        ↪  tests/unit/*/*
    rules:
        - if: $CI_COMMIT_TAG
launch_integration_tests_release:
    stage: deploy
    image: alpine:latest
    script:
        - apk add --update python3 py3-pip
        - python -m venv .venv
```

```
        - source .venv/bin/activate
        - python -m pip install -r requirements.txt
        - python -m pytest --cov=resources tests/integration/*
    needs:
        - launch_unit_tests_release
    rules:
        - if: $CI_COMMIT_TAG
launch_e2e_tests_release:
    stage: deploy
    image: alpine:latest
    script:
        - apk add --update python3 py3-pip
        - python -m venv .venv
        - source .venv/bin/activate
        - python -m pip install -r requirements.txt
        - python -m pytest --cov=resources tests/e2e/*
    needs:
        - launch_unit_tests_release
        - launch_integration_tests_release
    rules:
        - if: $CI_COMMIT_TAG
deploy_production_release:
    stage: deploy
    image: registry.gitlab.com/gitlab-org/release-cli:latest
    environment:
        name: production
    script:
        - apk add --update nodejs npm
        - npm install -g aws-cdk
        - apk add --update python3 py3-pip
        - python -m venv .venv
        - source .venv/bin/activate
        - python -m pip install -r requirements.txt
        - export CDK_DEPLOY_ENV=production
        - echo "CDK_DEPLOY_ENV=production" >> deploy.env
        - cdk deploy "*-$CDK_DEPLOY_ENV" --require-approval never
    artifacts:
        reports:
          dotenv: deploy.env
    needs:
        - launch_unit_tests_release
        - launch_integration_tests_release
        - launch_e2e_tests_release
    rules:
        - if: $CI_COMMIT_TAG
```

```
    when: manual
    allow_failure: False
```

9.4.1 Esempi test generati

Di seguito, un esempio di unit test. Il suo scopo è quello di verificare il corretto funzionamento di una singola Lambda Function.

```
import sys

# By default, resources can be found in project_root/resources.
sys.path.append("./resources")

# Edit this according to your files.
from lambda1 import handler

def test_lambda_alone():
    # Fill these, if needed.
    event = {}
    context = {}

    lambda_response = handler(event, context)

    assert lambda_response["body"].startswith("Actual version:")
```

Di seguito, un esempio di integration test. In questo esempio, verrà verificata la corretta integrazione di una Lambda Function facente uso, internamente, di un'altra Lambda Function.

```
import sys
from moto import mock_aws

# By default, resources can be found in project_root/resources.
sys.path.append("./resources")

# Edit this according to your files.
from lambda1 import handler

# Decorator needed since other AWS components could be used.
@mock_aws
def test_integration():
    # Fill these, if needed.
    event = {}
```

```
    context = {}

    # Since it's an integration test, handler will use internally
    # another resource (e.g. Lambda function).
    integration_response = handler(event, context)

    assert integration_response["body"].startswith("Actual version:")
```

Di seguito, un esempio di test end-to-end. Il suo scopo è quello di simulare un'interazione completa con l'applicativo, verificando il corretto funzionamento di ogni componente coinvolto nell'applicazione. A seconda dell'applicativo in esame, la complessità di un test end-to-end può diventare molto alta ed è pertanto stato fornito un boilerplate avente lo scopo di mostrare allo sviluppatore come recuperare tutti i dati necessari ai fini del test.

```
import boto3
import requests

def test_e2e():
    # Before running the test, stack_name must be set (use 'cdk ls' to list
    ↪  stacks).
    stack_name = "BluedacTestTbd1Stack-staging"

    # Change the lambda function API path accordingly.
    lambda_api_path = "/test"

    # CloudFormation stack resources retrieval.
    client = boto3.client("cloudformation")

    try:
        response = client.describe_stacks(StackName=stack_name)
    except Exception:
        print(f'No stack found with that name. Are you sure {stack_name} is
        ↪  deployed?')
        exit()

    stack_outputs = response["Stacks"][0]["Outputs"]

    # OutputValue parameter contains API Gateway endpoint URL.
    api_endpoint = [output['OutputValue'] for output in stack_outputs if
    ↪  output['OutputValue'].startswith('https://')][0]

    response = requests.get(api_endpoint + lambda_api_path)
```

```
    assert response.status_code == 200
```

Di seguito, la configurazione YAML di un test di carico eseguito tramite Artillery. Per motivi di sicurezza, il valore di «target», nel seguente test, è stato oscurato. Creando il suddetto test tramite BlueDAC, con l'apposito comando, il valore di «target» verrà automaticamente recuperato dallo stack.

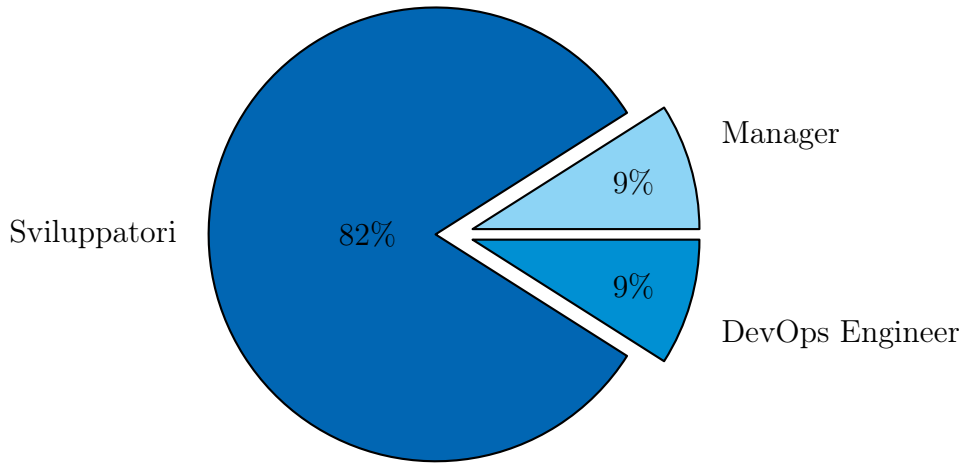
```
config:
    ensure:
        thresholds:
        conditions:
        - expression: aggregate.apdex.satisfied > aggregate.apdex tolerated
        ↪  and aggregate.apdex.frustrated < 0.1 *
        ↪  aggregate.apdex.satisfied
    phases:
        - arrivalRate: 3
            duration: 3
            name: 'Phase 1'
            rampTo: 5
    plugins:
        - apdex: {}
        - ensure: {}
    target: 'STACK_APIGW_URL'
scenarios:
    - flow:
    - count: 1
        loop:
        - get:
            url: "/test"
```

10 Caso di studio

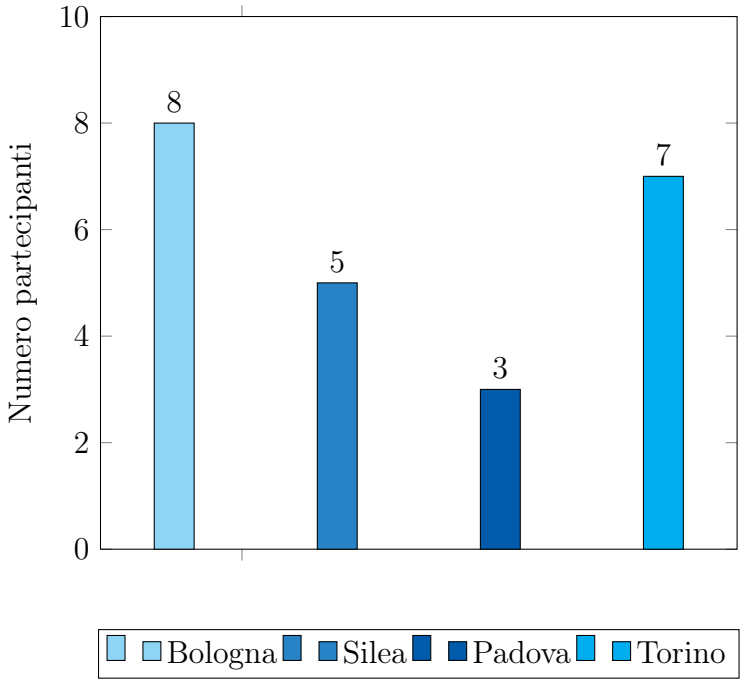
Al fine di raccogliere opinioni personali, motivazioni ed effettive conoscenze da professionisti del settore, è stato condotto un caso di studio sulle branching strategies utilizzate in ambito aziendale. Quest'ultimo si è basato su un'indagine campionaria, effettuata tramite la somministrazione di un questionario online.

10.1 Profilo dei partecipanti

I dati raccolti provengono da 23 professionisti del settore. Il tasso di risposta al questionario è stato pari al 23%, considerando un bacino di 100 partecipanti. Sebbene non sia stato previsto alcun vincolo sul ruolo professionale dei partecipanti, è emersa la seguente distribuzione:

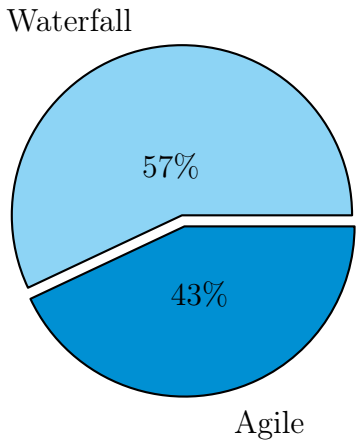


Ai fini dello studio, sono stati contattati professionisti provenienti da tutte le sedi dell'azienda. Solamente quattro di esse hanno avuto almeno un partecipante. Di seguito, un grafico rappresentante la distribuzione dei partecipanti all'interno delle sedi aziendali.

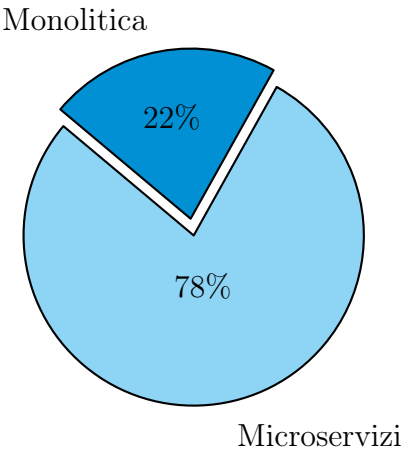


10.2 Struttura del questionario

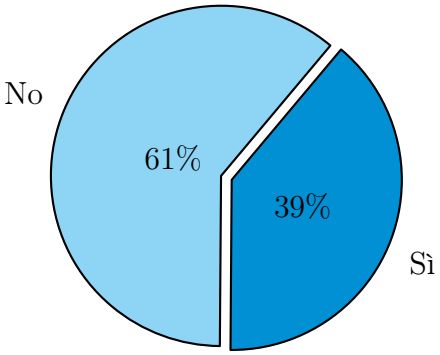
Il questionario proposto ha previsto sia domande a risposta aperta, utili per la raccolta di pareri personali sulle strategie in esame, sia domande a risposta multipla, volte a verificare l'allineamento tra le risposte dei partecipanti e le aspettative emerse in letteratura.



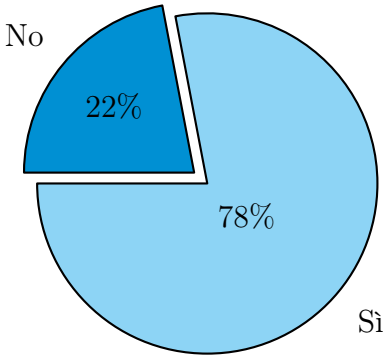
(a) Attuale metodologia di riferimento.



(b) Attuale architettura di riferimento.



(c) Conoscenza dell'approccio Trunk Based.



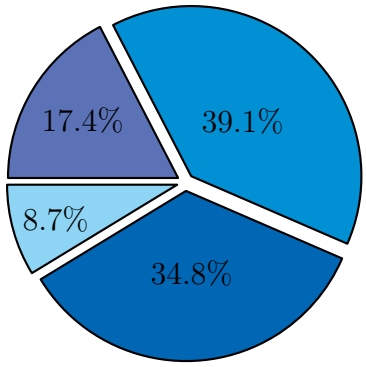
(d) Attuale, o precedente, esperienza pratica con l'approccio Trunk Based.

La domanda «a» ha riguardato la metodologia di riferimento del progetto attualmente seguito, in modo da ottenere informazioni sul tipo di progetto e modello di sviluppo di quest'ultimo. Similmente, tramite la domanda «b» sono state chieste informazioni relative all'architettura del progetto seguito attualmente.

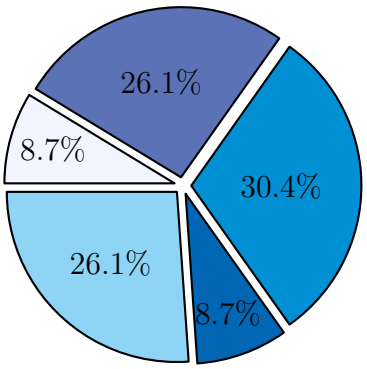
La domanda «c» è stata formulata con l'obiettivo di segmentare i partecipanti sulla base della conoscenza pregressa verso la metodologia Trunk Based Development, al fine di poter correlare le risposte emerse con il livello di familiarità verso tale approccio. Al fine di ottenere informazioni aggiuntive è stato chiesto, ai soli partecipanti conoscitori della metodologia Trunk Based, una precedente (o attuale) applicazione della metodologia in progetti aziendali. Tale distribuzione è stata ottenuta tramite la domanda «d».

La sezione successiva del questionario ha riguardato una serie di domande a

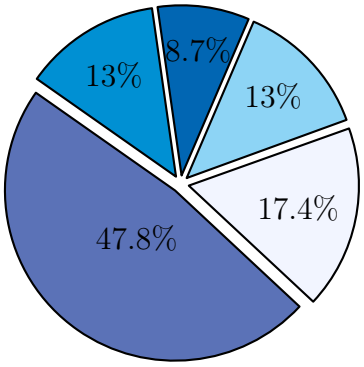
risposta multipla, aventi lo scopo di far emergere i problemi derivati dalla metodologia di sviluppo attualmente in uso.
Di seguito, un resoconto delle risposte:



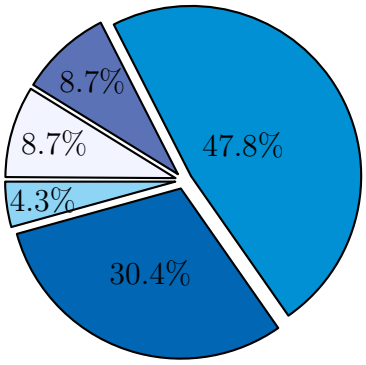
(a) Consiglierei al mio team di cambiare strategia di sviluppo.



(b) Consiglierei al mio team di applicare l'approccio TBD in progetti futuri.



(c) Consiglierei al mio team di applicare l'approccio GitFlow in progetti futuri.



(d) Attualmente, ho spesso conflitti in fase di merge.

- Decisamente no
- Più no che sì
- Più sì che no
- Decisamente sì
- Non so rispondere

La domanda «a» è stata utilizzata per valutare la soddisfazione verso l'attuale metodologia di sviluppo e identificare possibili problematiche nei processi at-

tuali. Le domande «b» e «c» sono state poste con l'obiettivo di valutare l'inclinazione e le preferenze dei partecipanti verso una particolare strategia di branching. La domanda «d» è stata posta per verificare la presenza, più o meno frequente, di conflitti in fase di merge. Quest'ultima domanda è particolarmente rilevante se affiancata alla strategia di branching utilizzata dal team. Nella sezione finale del questionario sono state proposte alcune domande aperte, con lo scopo di confrontare le opinioni dei partecipanti con la letteratura scientifica esistente, chiedendo loro di individuare i principali pregi e difetti degli approcci GitFlow e Trunk Based.

10.3 Analisi dei risultati

Dalle risposte degli intervistati emerge come poco più di un terzo (39%) di essi fosse già a conoscenza della strategia Trunk Based Development, ciò ha inciso particolarmente sulla qualità delle risposte fornite alle altre domande.
Emerge come coloro che abbiano affermato di non conoscere la strategia sopracitata abbiano confuso l'approccio GitFlow con lo strumento Git, portando a risposte completamente incoerenti. D'altra parte, coloro che hanno affermato di esserne a conoscenza hanno citato aspetti coerenti con le aspettative.
Gli aspetti maggiormente citati, relativi all'approccio Trunk Based, sono stati:

- Maggiore propensione alla continuous integration e continuous delivery;
- Minore probabilità di incontrare conflitti in fase di merge;
- Richiesta di un'alta maturità, spesso non posseduta, per raggiungere la continuous deployment;
- Maggiore stabilità e facilità di gestione delle release.

La valutazione media assegnata all'approccio Trunk Based è stato 4,4, mentre il voto medio assegnato all'approccio GitFlow è stato 4,1.

10.4 Considerazioni emerse dallo studio

L'indagine ha messo in luce come una porzione significativa dei professionisti intervistati manifesti incertezze e lacune nella conoscenza e applicazione delle strategie di branching. Tale evidenza suggerisce l'opportunità e la necessità per le aziende di investire in iniziative di formazione continua, finalizzate alla diffusione di una cultura DevOps più solida.

11 Conclusione

Per riassumere il contributo apportato da questa tesi, è stato sviluppato un framework avente lo scopo di automatizzare tutte le fasi DevOps di un progetto cloud serverless, gestendo sia la fase di inizializzazione di un nuovo progetto, sia la fase di gestione inoltrata del progetto stesso.
Con lo specifico focus sul cloud provider AWS, il framework è in grado di inizializzare il progetto come un repository git locale, inserendo al suo interno un'iniziale architettura AWS. Tramite un'interazione da terminale, i comandi messi a disposizione da BlueDAC consentono la creazione di una pipeline CI/CD in formato GitLab, a partire dal file di configurazione «bluedac_config.json». Sarà poi possibile gestire la creazione e l'esecuzione di diversi tipi di test, tra cui:

- Unit test;
- Integration test;
- Load test;
- End-to-End test.

Un ulteriore focus del framework è la scelta della branching strategy, che fornisce una maggiore flessibilità in fase di organizzazione e gestione dei repository, a seconda delle modalità di sviluppo che si intendono utilizzare all'interno del progetto. In un'ottica di sviluppo di nuovi progetti in un ambito emergente come quello cloud, il framework si rende particolarmente adatto al supporto di strategie emergenti come la Trunk Based Development, in grado di abilitare a pratiche sempre più necessarie, come la continuous integration e la continuous delivery/deployment.

11.1 Risultati ottenuti

Il risultato è un framework in grado di accelerare le fasi iniziali di un progetto cloud serverless, consentendo di risolvere la «sindrome da foglio bianco». La gestione del progetto, anche in fasi inoltrate, risulta fluida e flessibile, dando la possibilità di modificare in qualsiasi momento la strategia di branching, il linguaggio di programmazione, le modalità di rilascio e numerosi altri parametri. Inoltre, la pratica gestione dei test tramite il framework stesso consente di avere maggiore comodità nella definizione di script, essendo i comandi meno verbosi rispetto all'implementazione di libreria.

11.2 Sviluppi futuri

Un'attuale limitazione del framework è rappresentata dalla sua natura single-cloud, che lo rende al momento compatibile con il solo AWS. In un'ottica futura, in cui le organizzazioni stanno migrando verso soluzioni multi-cloud, il framework dovrà ampliare il proprio supporto ad altri fornitori.
Inoltre, il quantitativo di job utilizzabili ai fini della composizione della pipeline CI/CD potrebbe risultare limitante dal punto di vista della flessibilità in caso di progetti particolarmente complessi. Dato l'alto coefficiente di interesse su questo fattore, si prevede di intervenire con l'inserimento di nuovi job, al fine di fornire supporto a quante più richieste progettuali possibili.
Un altro punto di interesse, come da attuali tendenze nel mondo dello sviluppo, è quello di inserire una componente di machine learning. Sarà in questo modo possibile evitare di dover interagire direttamente con il terminale, facendo eseguire al modello linguistico le opportune configurazioni. Questo eviterà di richiedere al gestore del progetto particolari competenze tecnico-informatiche.

Bibliografia

[1] Atlassian. *Continuous Delivery - What is continuous delivery?* URL: https://www.atlassian.com/it/continuous-delivery. (accessed: 14.07.2024).

[2] Atlassian. *Continuous Deployment - What is continuous deployment?* URL: https://www.atlassian.com/continuous-delivery/software-testing/continuous-deployment. (accessed: 14.07.2024).

[3] Atlassian. *Continuous Integration - What is continuous integration?* URL: https://www.atlassian.com/continuous-delivery/continuous-integration. (accessed: 14.07.2024).

[4] Atlassian. *DevOps - DevOps Toolchain: key considerations.* URL: https://www.atlassian.com/devops/devops-tools/choose-devops-tools. (accessed: 14.07.2024).

[5] Atlassian. *DevOps - What is DevOps?* URL: https://www.atlassian.com/devops. (accessed: 14.07.2024).

[6] Atlassian. *Different types of software testing.* URL: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing. (accessed: 18.07.2024).

[7] Atlassian. *Gitflow workflow.* URL: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow. (accessed: 14.07.2024).

[8] AWS. *Create your first Lambda Function.* URL: https://docs.aws.amazon.com/lambda/latest/dg/getting-started.html. (accessed: 18.07.2024).

[9] AWS. *Create your first Lambda Function.* URL: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html. (accessed: 18.07.2024).

[10] AWS. *Load testing a web application's serverless backend.* URL: https://aws.amazon.com/blogs/compute/load-testing-a-web-applications-serverless-backend/. (accessed: 24.07.2024).

[11] AWS. *What is Amazon S3?* URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html. (accessed: 18.07.2024).

[12] AWS. *What is API Gateway?* URL: https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html. (accessed: 18.07.2024).

[13] AWS. *What is AWS CDK?* URL: https://docs.aws.amazon.com/cdk/v2/guide/home.html. (accessed: 18.07.2024).

[14] AWS. *What is AWS CloudFormation?* URL: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html. (accessed: 18.07.2024).

[15] AWS. *What is AWS CloudWatch?* URL: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html. (accessed: 18.07.2024).

[16] AWS. *What is AWS CodeDeploy?* URL: https://docs.aws.amazon.com/codedeploy/latest/userguide/welcome.html. (accessed: 18.07.2024).

[17] AWS. *What is DynamoDB?* URL: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html. (accessed: 18.07.2024).

[18] AWS. *What is the Cloud?* URL: https://azure.microsoft.com/it-it/resources/cloud-computing-dictionary/what-is-the-cloud/. (accessed: 18.07.2024).

[19] AWS. *What is the Infrastructure as Code?* URL: https://aws.amazon.com/what-is/iac/. (accessed: 18.07.2024).

[20] AWS. *What is the SDLC?* URL: https://aws.amazon.com/what-is/sdlc/. (accessed: 18.07.2024).

[21] GitHub. *GitHub Flow.* URL: https://docs.github.com/en/get-started/using-github/github-flow. (accessed: 14.07.2024).

[22] GitLab. *GitLab Flow.* URL: https://about.gitlab.com/topics/version-control/what-is-gitlab-flow/. (accessed: 14.07.2024).

[23] Paul Hammant. *Trunk Based Development - Branch for Release.* URL: https://trunkbaseddevelopment.com/branch-for-release/. (accessed: 14.07.2024).

[24] Paul Hammant. *Trunk Based Development - Committing straight to Trunk.* URL: https://trunkbaseddevelopment.com/committing-straight-to-the-trunk/. (accessed: 14.07.2024).

[25] Paul Hammant. *Trunk Based Development - Feature Flags.* URL: https://trunkbaseddevelopment.com/feature-flags/. (accessed: 14.07.2024).

[26] Paul Hammant. *Trunk Based Development - history.* URL: https://trunkbaseddevelopment.com/#history. (accessed: 14.07.2024).

[27] Paul Hammant. *Trunk Based Development - Releasability of work in progress*. URL: https://trunkbaseddevelopment.com/5-min-overview/#developer-team-commitments. (accessed: 14.07.2024).

[28] Paul Hammant. *Trunk Based Development - Release from Trunk*. URL: https://trunkbaseddevelopment.com/release-from-trunk/. (accessed: 14.07.2024).

[29] Paul Hammant. *Trunk Based Development - Short-lived Branches*. URL: https://trunkbaseddevelopment.com/short-lived-feature-branches/. (accessed: 14.07.2024).

[30] Microsoft. *What is Cloud Computing?* URL: https://azure.microsoft.com/resources/cloud-computing-dictionary/what-is-cloud-computing/. (accessed: 11.08.2024).

[31] Microsoft. *What is the Cloud?* URL: https://azure.microsoft.com/resources/cloud-computing-dictionary/what-is-the-cloud/. (accessed: 18.07.2024).

[32] RedHat. *What is a CI/CD Pipeline?* URL: https://www.redhat.com/it/topics/devops/what-cicd-pipeline. (accessed: 14.07.2024).

[33] RedHat. *What is Serverless?* URL: https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless. (accessed: 18.07.2024).

[34] TechTarget. *The history of cloud computing explained*. URL: https://www.techtarget.com/whatis/feature/The-history-of-cloud-computing-explained. (accessed: 11.08.2024).

[35] European Union. *Shaping Europe's digital future*. URL: https://digital-strategy.ec.europa.eu/en/policies/cloud-computing. (accessed: 17.08.2024).