

Integrazione di Ansible e Docker per una pipeline CI/CD utilizzando Jenkins

Daniele Commodaro
daniele.commodaro@studio.unibo.it

Luglio 2022

CI/CD è un metodo per la distribuzione frequente delle app ai clienti, che prevede l'introduzione dell'automazione nelle fasi di sviluppo dell'applicazione. Principalmente, si basa sui concetti di integrazione, distribuzione e deployment continui. L'approccio CI/CD supera le difficoltà legate all'integrazione di nuovo codice, una situazione così problematica per i team operativi e di sviluppo da essere conosciuta "inferno dell'integrazione"[1].

Più specificamente, il metodo CI/CD introduce l'automazione costante e il monitoraggio continuo in tutto il ciclo di vita delle applicazioni, dalle fasi di integrazione e test a quelle di distribuzione e deployment. Nel complesso, questi processi interconnessi vengono definiti "flussi CI/CD" e sono supportati dai team operativi e di sviluppo che collaborano secondo modalità agili[1].

Il proposito di questo progetto è quello di realizzare l'automazione di un flusso CI/CD, utilizzando Ansible, Docker e Jenkins.

Un approccio comune per creare una pipeline automatizzata consiste nel combinare due piattaforme per orchestrare e quindi distribuire, in modo Agile e continuo, applicazioni e infrastruttura. Ansible viene utilizzato per il provisioning di ambienti, configurazioni dell'infrastruttura e distribuzione di app all'interno di un processo CI/CD utilizzando una pipeline Jenkins.

Nel mondo del processo CI/CD, Jenkins è uno strumento popolare per il provisioning di ambienti di sviluppo/produzione e per la distribuzione di applicazioni attraverso il flusso della pipeline. Tuttavia, a volte, mantenere lo stato dell'applicazione diventa opprimente e il riutilizzo degli script diventa più difficile man mano che il progetto cresce.

Per superare questa limitazione, Ansible svolge un ruolo fondamentale come esecutore di script di shell. Lo scopo del suo utilizzo nel flusso della pipeline è di riutilizzare ruoli e playbook per il provisioning, lasciando Jenkins solo come orchestratore di processi anziché come esecutore di script di shell.

L'impiego di Jenkins facilita anche l'utilizzo di Docker, il quale può rendere il processo di software delivery più veloce, prevedibile e gestibile. L'implementazione della Continuous Delivery su applicazioni pacchettizzate in Docker favorisce molteplici vantaggi, tra cui una maggiore automazione ed una migliore gestione dello sviluppo software e dei processi di delivery, conferendo al contempo un livello di sicurezza maggiore alle applicazioni che girano in un ambiente condiviso.

Indice

1	Obiettivi e Risultati Attesi	4
2	Piano di Lavoro	4
3	Integrazione continua	4
3.1	Teoria	5
3.2	Principi	5
3.2.1	Mantenere un repository del codice sorgente	5
3.2.2	Automatizzare la build	5
3.2.3	Rendere la build auto-testante	5
3.2.4	Tutti eseguono commit alla baseline tutti i giorni	6
3.2.5	Ogni commit fa partire una build	6
3.2.6	Fare in modo che la build sia veloce	6
3.2.7	Eseguire i test in un clone dell'ambiente di produzione	6
3.2.8	Fare in modo che sia facile acquisire le ultime versioni dei pacchetti . . .	6
3.2.9	Ognuno può vedere i risultati dell'ultima build	6
3.2.10	Automatizzare i rilasci	6
4	Distribuzione continua e consegna continua	6
4.1	Distribuzione continua	6
4.2	Consegna continua	7
5	Panoramica dell'applicazione	7
5.1	La piattaforma Oracle	7
5.1.1	Contatti	8
5.1.2	Campagne	8
5.1.3	Oggetti personalizzati	8
5.2	Design del sistema	9
5.2.1	Tecnica AJAX	9
5.2.2	Django	9
5.2.3	Redis come message broker per Celery	9
5.2.4	APIs	10
5.2.5	Attività	10
5.2.6	RabbitMQ	11
5.2.7	Test dell'applicazione con Postman e Newman	11
6	Perché usare Docker per il deploy di applicazioni complesse	12
7	Docker	13
7.1	Il Dockerfile	13
7.1.1	Comandi del Dockerfile	14
7.2	Volumi	15
7.3	Docker Compose	15
8	Ansible	16
8.1	Come funziona Ansible	17
8.2	Cosa fa Ansible	17
8.3	Playbook di Ansible	17
8.4	Inventari	18
8.5	Variabili e ruoli	18
8.6	Gestione della configurazione con Ansible	18
8.7	Provisioning con Ansible	18

8.8	Automazione della distribuzione	19
8.9	Orchestrazione con Ansible	19
8.10	Vantaggi e svantaggi di Ansible	19
8.10.1	Vantaggi di Ansible	19
8.10.2	Svantaggi di Ansible	20
9	Jenkins. Spiegazione del server per l'integrazione continua	21
9.1	Hudson e Jenkins	21
9.2	Automazione con Jenkins	22
9.3	Come funziona Jenkins	22
9.4	Jenkins pipelines	22
9.5	Perché usare Jenkins	23
9.6	Jenkins per CI/CD	23
9.7	Vantaggi e svantaggi di Jenkins	23
9.7.1	Vantaggi di Jenkins	23
9.7.2	Svantaggi di Jenkins	25
10	Ansible contro Jenkins	25
11	Parte pratica	27
11.1	Dockerizzazione dell'applicazione	27
11.1.1	Definizione del Dockerfile per il database PostgreSQL	27
11.1.2	Definizione del Dockerfile per la webapp	27
11.1.3	Definizione del Docker Compose	28
11.2	Setup e deploy con Ansible	29
11.2.1	Inventari	29
11.2.2	Playbook	29
11.2.3	Role: deploy_linux.yml	30
11.2.4	Role: linux.yml	32
11.2.5	Role: postgres.yml	35
11.3	Creazione della pipeline Jenkins	36
11.4	Funzionamento del sistema	38
11.5	Test del sistema	39
11.5.1	Limiti delle tecnologie impiegate e del sistema	40
12	Conclusioni	42
12.1	Sviluppi futuri	42

1 Obiettivi e Risultati Attesi

Gli obiettivi del progetto sono:

- Attraverso articoli scientifici e/o documentazione tecnica, porre le basi per la conoscenza di:
 - Metodologia CI/CD
 - Ansible e Jenkins: architettura, vantaggi/svantaggi, confronto
 - Provisioning con Ansible
 - Docker: impiego in questo contesto, pro e contro
 - Configurazione e utilizzo delle componenti
- Realizzare una pipeline CI/CD con Jenkins integrando Ansible e Docker e testarne la correttezza.
- Creare uno scenario di test progettato per rilevare i limiti delle tecnologie utilizzate

I risultati attesi sono:

- Comprendere bene i flussi CI/CD
- Conoscere meglio gli strumenti proposti
- Avere un sistema che faccia quello per cui è stato progettato
- Ottenere dei risultati soddisfacenti attraverso lo scenario di test progettato

2 Piano di Lavoro

Il piano di lavoro è il seguente:

- Parte teorica:
 1. Raccolta delle informazioni
 2. Comprensione e apprendimento
- Parte pratica:
 1. Sviluppo e test del sistema
 2. Creazione dello scenario di test rivolto alle tecnologie
 3. Testing delle tecnologie

3 Integrazione continua

Nell'ingegneria del software, l'integrazione continua (continuous integration in inglese, spesso abbreviato in CI) è una pratica che si applica in contesti in cui lo sviluppo del software avviene attraverso un sistema di controllo versione. Consiste nell'allineamento frequente (ovvero "molte volte al giorno") dagli ambienti di lavoro degli sviluppatori verso l'ambiente condiviso. Il concetto è stato originariamente proposto nel contesto dell'extreme programming(XP), come contromisura preventiva per il problema dell'"integration hell" (le difficoltà dell'integrazione di porzioni di software sviluppati in modo indipendente su lunghi periodi di tempo e che di conseguenza potrebbero essere significativamente divergenti). Il CI può essere considerato come una estremizzazione di idee già presenti in altri metodi precedenti all'XP, per esempio il metodo Booch. Il CI è stato originariamente concepito per essere complementare rispetto ad altre

pratiche, in particolare legate al Test Driven Development(sviluppo guidato dai test, TDD). In particolare, si suppone generalmente che siano stati predisposti test automatici che gli sviluppatori possono eseguire immediatamente prima di rilasciare i loro contributi verso l'ambiente condiviso, in modo da garantire che le modifiche non introducano errori nel software esistente. Per questo motivo, il CI viene spesso applicato in ambienti in cui siano presenti sistemi di build automatici e/o esecuzione automatica di test, come Jenkins. Tuttavia, l'identificazione del CI con l'uso di strumenti di questo tipo, frequente in letteratura, è scorretta: di per sé, la pratica del CI può essere applicata anche a prescindere da sistemi di testing automatico[2].

3.1 Teoria

Prima di effettuare una modifica sul codice sorgente, uno sviluppatore prende una copia del codice corrente sulla quale lavorare. Man mano che gli altri sviluppatori fanno delle modifiche sul repository, la copia smette di rappresentare il codice del repository. Le modifiche non si riflettono unicamente sul codice, ma anche sulle librerie ed altre risorse che possono creare dipendenze e potenziali conflitti. Più è lungo il tempo durante il quale un branch di codice rimane scaricato, maggiore è il rischio di conflitti di integrazione quando il branch viene reintegrato nel repository. Quando gli sviluppatori vogliono sottomettere il codice, devono prima aggiornare le proprie copie locali del codice con i cambiamenti che sono stati fatti al repository dal momento in cui loro hanno aggiornato la copia locale. Più modifiche sono state fatte e più lavoro è necessario fare prima di sottomettere le proprie modifiche. È possibile che il repository sia diventato così diverso dalla copia locale dello sviluppatore che il tempo necessario per integrare le proprie modifiche con quelle fatte dagli altri superi il tempo necessario a fare le modifiche. Questo fenomeno viene descritto come inferno dell'integrazione(integration hell in inglese) o inferno dei merge(merge hell). In uno scenario estremamente negativo, gli sviluppatori potrebbero dover annullare le proprie modifiche e rifarle da capo. L'integrazione continua è un processo che consiste nell'integrare presto e spesso, limitando le possibilità dell'inferno dell'integrazione. La pratica cerca di minimizzare il lavoro inutile e risparmiare tempo[2].

3.2 Principi

Di seguito vengono riportati i principi introdotti da Martin Fowler, che permettono di svolgere l'integrazione continua.

3.2.1 Mantenere un repository del codice sorgente

Questo elemento è propedeutico a tutti gli altri principi, perché senza avere un repository del codice è impossibile automatizzare la build ed i test.

3.2.2 Automatizzare la build

Per build si intende il processo che trasforma il codice sorgente in un artefatto eseguibile. Deve essere possibile eseguire la compilazione e la creazione dei pacchetti da mettere sui server in modo completamente automatizzato, senza alcun intervento umano.

3.2.3 Rendere la build auto-testante

Ogni volta che il codice sorgente viene compilato ed impacchettato, è importante che vengano eseguiti dei test sul sorgente affinché la qualità del codice venga tenuta sotto controllo ed eventuali bug vengano scoperti il prima possibile.

3.2.4 Tutti eseguono commit alla baseline tutti i giorni

É inutile compilare e testare il sorgente se gli sviluppatori sviluppano codice sui propri computer senza sincronizzarsi spesso col codice scritto da altri. É importante però che il codice sia compilabile ed abbia superato tutti i test.

3.2.5 Ogni commit fa partire una build

Ogni modifica al codice sorgente condiviso potrebbe generare dei bug e quindi compilare e testare subito dá la possibilità di intervenire immediatamente su eventuali bug. Questo principio é importante a seguito di un altro principio, detto in inglese fail fast, ovvero se un commit di uno sviluppatore é tale da far fallire un test, bisogna intervenire immediatamente prima che si crei una sequenza di errori successivi a valanga.

3.2.6 Fare in modo che la build sia veloce

Un build troppo lenta può rendere poco agevole il far partire la build automatica dopo i commit, e potrebbe costringere gli sviluppatori ad aspettare la fine della build prima di effettuare un commit o di verificarne l'esito.

3.2.7 Eseguire i test in un clone dell'ambiente di produzione

Se l'ambiente su cui vengono eseguiti i test non é assolutamente identico all'ambiente di produzione, c'è il rischio che qualcosa che é stato testato generi dei bug inspiegabili una volta rilasciato in produzione, con conseguenti danni economici.

3.2.8 Fare in modo che sia facile acquisire le ultime versioni dei pacchetti

Questo principio serve quando si hanno tanti ambienti che devono essere tenuti allineati.

3.2.9 Ognuno può vedere i risultati dell'ultima build

La trasparenza permette di stabilire il livello di qualità raggiunto da ogni modulo e capire quali sono i moduli che hanno bisogno di maggiore attenzione.

3.2.10 Automatizzare i rilasci

Anche il rilascio del software sugli ambienti di test o sull'ambiente di produzione deve essere automatizzato.

4 Distribuzione continua e consegna continua

4.1 Distribuzione continua

La distribuzione continua si concentra, proprio come suggerisce il nome, sulla distribuzione; l'effettiva installazione e distribuzione dei bit. Durante una distribuzione, il pacchetto binario/dell'applicazione può attraversare la topologia in cui l'applicazione o l'infrastruttura dell'applicazione deve servire il traffico. In senso tradizionale, la distribuzione continua si concentra sull'automazione da distribuire tra ambienti o cluster. Man mano che si attraversano gli ambienti dall'ambiente non di produzione all'ambiente di staging e infine alla produzione, il numero di endpoint in cui si distribuisce aumenta. La distribuzione continua si concentra sul percorso di minor resistenza per portare il software negli ambienti necessari. Con piattaforme moderne come Kubernetes, la separazione degli ambienti potrebbe non essere fisica rispetto alle piattaforme legacy o tradizionali basate su macchine. Uno spazio dei nomi (separazione del software) potrebbe essere tutto ciò che separa lo sviluppo dalla produzione, sebbene i buoni principi dei

sistemi distribuiti si applichino comunque indipendentemente dalla piattaforma scelta. Con i sistemi distribuiti, le topologie in cui le modifiche devono propagarsi possono essere di grandi dimensioni, anche negli ambienti di pre-produzione[3].

Poiché l'integrazione continua fornisce un artefatto distribuibile, la distribuzione continua può portare tale artefatto in altri ambienti. Non appena viene creato un nuovo artefatto, è possibile portarlo immediatamente in un ambiente di integrazione di sviluppo e/o garanzia di qualità per avviare i test di integrazione.

4.2 Consegna continua

La distribuzione continua e la consegna continua hanno obiettivi leggermente diversi. La distribuzione continua si sovrappone alla consegna continua sul fronte della distribuzione, ma occorre essere attenti a non sovraccaricare i sistemi di distribuzione, al fine di mantenere fiducia e sicurezza che sono fondamentali per la distribuzione continua.

La Continuous Delivery è l'automazione dei passaggi per portare in sicurezza le modifiche alla produzione. Laddove la distribuzione continua si concentra sulla distribuzione effettiva, la consegna continua si concentra sul rilascio e sulla strategia di rilascio.

Il percorso verso un rilascio può essere delicato. Poiché il software è il culmine delle scelte di diversi team, attraversare tutti i passaggi e le modifiche necessarie per propagare/supportare le modifiche o le nuove funzionalità può essere tortuoso. La consegna continua include tutte le fasi di test, implementazioni incrementali/ambientali e fasi di convalida per portare in sicurezza le modifiche alla produzione.

5 Panoramica dell'applicazione

Kaleyra S.p.a. è la società presso cui lavorano i membri del team che partecipano allo sviluppo di questo progetto, essa si occupa principalmente di offrire servizi di comunicazione nel settore bancario e finanziario. L'applicazione per la quale si vuole realizzare una pipeline automatizzata è un plug-in usato per l'invio di notifiche push disponibile nel Marketplace di Oracle. Il plug-in di Kaleyra consente agli esperti di marketing di fornire campagne per l'invio di notifiche push, in grado di incorporare le personalizzazioni e le opzioni impostate all'interno di Oracle Marketing Cloud. L'app è progettata per consentire ai clienti della piattaforma di gestire facilmente le campagne multicanale aggiungendo la comunicazione push ai propri strumenti. Il metodo di personalizzazione dell'applicazione è anche in grado di migliorare la capacità degli esperti di marketing di fornire ai clienti contenuti personalizzati, che a loro volta consentono un aumento delle conversioni e la fidelizzazione dei clienti.

5.1 La piattaforma Oracle

Il plugin di Kaleyra è ospitato all'interno di una piattaforma software as a service(SaaS) per l'automazione del marketing offerta da Oracle, che mira ad aiutare i marketer e le organizzazioni b2b a gestire le campagne di marketing. La piattaforma invia annunci di marketing a dispositivi mobili, e-mail, video e pagine dei risultati di ricerca. La soluzione marketing di Oracle viene spesso utilizzata nelle grandi aziende che commercializzano principalmente ad altre aziende. Essa offre pacchetti per il marketing di base, standard e aziendale. Numerose funzionalità differenziano i livelli di account, come l'accesso a vari servizi di vendita e marketing. Nel supportare le funzionalità fornite dal plugin di Kaleyra, la piattaforma offre i seguenti servizi:

- fornisce i contatti, che sono i soggetti ai quali verrà sottomessa la campagna di marketing
- fornisce agli utenti gli elementi necessari per poter in autonomia creare, gestire e monitorare le campagne

- fornisce supporto per la memorizzazione dei dati relativi agli elementi di una campagna in delle tabelle all'interno del cloud Oracle che sono chiamate oggetti personalizzati. Questi dati vengono alimentati dalle applicazioni che compongono la campagna e con queste vengono sincronizzati per essere aggiornati

É possibile anche monitorare il rendimento delle proprie campagne in quanto la piattaforma consente di accedere facilmente a report e metriche di performance direttamente dall'area di disegno della campagna.

5.1.1 Contatti

Nella piattaforma un contatto é un'entità di dati che contiene le informazioni esplicite su una singola persona. I contatti(e gli account associati) vengono utilizzati come elementi costitutivi principali per i segmenti. Un segmento é un elemento fondamentale di una campagna, esso rappresenta un insieme di contatti che viene usato per alimentare le campagne.

5.1.2 Campagne

Nella piattaforma le campagne sono il cuore del processo di automazione del marketing. Le campagne consentono di creare viaggi dinamici, personalizzati e unici per i contatti. Il canvass della campagna della piattaforma fornisce un'interfaccia grafica di facile utilizzo, drag-and-drop, che consente di progettare visivamente i flussi di lead(flussi di vendita) attraverso campagne multicanale, multifase e basate su trigger. L'area di disegno della campagna include un'ampia varietà di elementi che é possibile utilizzare per creare campagne ricche e multifase che soddisfano gli obiettivi del marketing. Il plugin di Kaleyra é uno di questi elementi, le funzionalità che introduce in una campagna sono:

- la possibilità di generare delle notifiche push da inviare ai contatti presenti nella campagna. Le push notification possono essere configurate sulla base di diversi servizi usati dal cliente. Un servizio definisce le etichette usate nei campi di invio della notifica, gli oggetti personalizzati di riferimento e le code da usare per inviare le push al Gateway
- la sincronizzazione delle informazioni relative l'invio delle notifiche push con gli oggetti personalizzati della piattaforma Oracle: i dati di maggiore interesse possono essere immagazzinati sugli oggetti personalizzati della piattaforma per poter essere poi consultati dai clienti
- delle operazioni per il controllo e la gestione delle notifiche push durante l'esecuzione di una campagna: il plugin fornisce operazioni per eseguire dei controlli sui campi di una notifica push, e sulla base dei controlli permette di gestire la notifica all'interno del flusso della campagna. Questo permette ad esempio di risottomettere una notifica nella campagna qualora un primo invio non fosse andato a buon fine. Per avere le informazioni relative l'invio delle push il plugin riceve dei report di consegna dal Gateway attraverso un'apposita coda. I dati dei report vengono poi aggiornati negli oggetti personalizzati di Oracle

5.1.3 Oggetti personalizzati

Sulla piattaforma Oracle un oggetto personalizzato é un insieme di record personalizzati relativi a un contatto. Gli oggetti personalizzati consentono di archiviare dati aggiuntivi in modo scalabile e di collegare tali dati a un contatto. Di solito vengono usati per dati temporanei con durata di conservazione limitata e contengono informazioni relative alle operazioni di marketing. Nel caso del plugin l'oggetto personalizzato contiene record relativi all'invio della notifica push generata.

5.2 Design del sistema

5.2.1 Tecnica AJAX

Lo scambio di dati in background tra browser web e l'applicazione é gestito con tecnica di sviluppo AJAX, consentendo così l'aggiornamento dinamico delle view senza esplicito ricaricamento. Le chiamate HTTP vengono quindi gestite in modo asincrono da Python e le pagine vengono renderizzate tramite il motore Django che elabora i template. In questo modo vengono richiesti i dati dal server e caricati in background senza interferire con il comportamento della view esistente.

5.2.2 Django

L'applicazione utilizza il framework web Django, é un framework web Python di alto livello gratuito e open source che consente lo sviluppo di applicazioni web sicure e gestibili. Django fornisce:

- un URL mapper per reindirizzare le richieste HTTP alla vista appropriata in base all'URL ricevuto. Il mapper definisce un elenco di mappature tra rotte(modelli URL specifici) e funzioni di visualizzazione corrispondenti. Se viene ricevuta una richiesta HTTP con un URL che corrisponde a un modello specificato, la funzione di visualizzazione associata viene chiamata e passata la richiesta
- le viste che sono delle funzioni impiegate nella gestione delle richieste, che ricevono richieste HTTP e restituiscono risposte HTTP. Le viste accedono ai dati necessari per soddisfare le richieste tramite i modelli e delegano la formattazione della risposta ai template
- i modelli: sono oggetti Python che definiscono la struttura dei dati dell'applicazione e forniscono meccanismi per gestire (aggiungere, modificare, eliminare) e interrogare i record nel database
- i template che definiscono la struttura o il layout delle pagine HTML
- dei middleware per il supporto di meccanismi di sicurezza e autenticazione, gestione dei dati attraverso una console di amministrazione, gestione dei cookie
- la libreria Celery che permette di distribuire il carico computazionale tra thread e macchine. Celery comunica tramite messaggi attraverso Redis e si concentra sull'elaborazione in tempo reale, supportando allo stesso tempo la pianificazione delle attività. Le attività vengono eseguite in modo asincrono

Una visione olistica dell'architettura di Django può essere rappresentata come nell'immagine seguente

5.2.3 Redis come message broker per Celery

Redis é un archivio di strutture dati in memoria che mantiene i dati in RAM e permette di consegnarli più rapidamente. Viene utilizzato come broker di messaggi per Celery che é una libreria di Django usata per gestire le attività dell'applicazione.

Celery consente l'esecuzione di carichi di lavoro computazionali asincroni guidati dalle informazioni contenute nei messaggi prodotti nel codice dell'applicazione destinati a una coda di attività Celery. Viene anche usato per eseguire attività ripetibili, periodiche(cioé programmate). Celery migliora le sue prestazioni quando sfrutta a una soluzione di archiviazione definita come broker di messaggi. Il broker di messaggi utilizzato con Celery é Redis, funge da archivio dati chiave-valore performante, in memoria. In particolare, Redis viene utilizzato per memorizzare nella coda delle attività di Celery i messaggi prodotti dal codice dell'applicazione che descrivono il lavoro da svolgere.

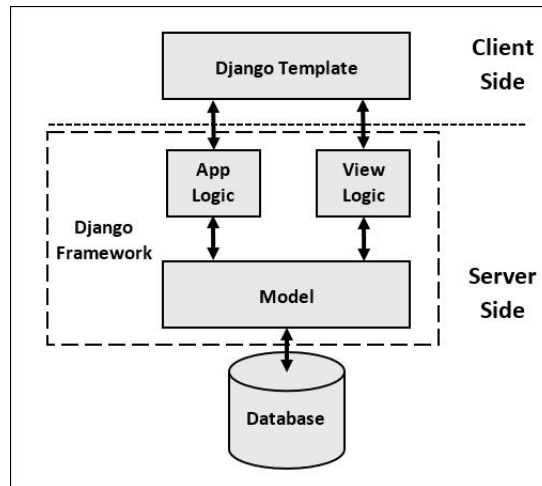


Figura 1: Architettura di Django

5.2.4 APIs

Attraverso delle API REST esposte il plugin viene contattato dalla piattaforma Oracle, queste API permettono di:

- creare un'istanza del plugin nell'area di disegno della piattaforma. Questa chiamata avviene quando l'utente seleziona il plugin nell'elenco delle possibili scelte ed esegue un drag and drop nel canvass
- fornire in risposta alle chiamate i template e i dati necessari per poter rappresentare correttamente gli elementi grafici del plugin
- configurare il plugin per impostare:
 - il tipo di servizio che serve a determinare le label delle notifiche e le code usate per lo scambio di messaggi col Gateway
 - i campi dei contatti da considerare per l'invio delle notifiche
 - il corpo del messaggio
 - contattare un servizio esterno(shortlink) adibito a ridurre la lunghezza dei link
- cancellare l'elemento dall'area di disegno
- creare una copia del plugin nell'area di disegno
- ricevere chiamate di notifica. Quando una campagna viene lanciata in esecuzione il plugin riceve delle chiamate di notifica dalla piattaforma Oracle, queste chiamate trasportano i contatti per i quali vengono generate le notifiche push
- gestire la sicurezza, i token, l'autenticazione

Attraverso dei processi batch asincroni il plugin contatta la piattaforma periodicamente per eseguire delle operazioni di aggiornamento sui contatti e gli oggetti personalizzati

5.2.5 Attività

La logica applicativa é separata in attività che vengono distribuite su dei worker thread gestiti dalla libreria Celery. Le attività possono essere generate in seguito alle chiamate di API oppure sono programmate per avere un'esecuzione periodica. Le principali attività dell'applicazione riguardano:

- l'aggiornamento dei dati con la piattaforma Oracle
- il recupero di informazioni dal database
- l'invio di notifiche push
- controllo delle notifiche push
- la gestione dei token
- la gestione della data retention per il GDPR
- un meccanismo di retry per riprovare ad inviare le notifiche push che sono andate in errore durante l'accodamento con Rabbit

5.2.6 RabbitMQ

RabbitMQ é un message-oriented middleware(detto anche broker di messaggistica) che implementa il protocollo Advanced Message Queuing Protocol(AMQP). Un broker di messaggi é un modello architetturale per la convalida, la trasformazione e il routing dei messaggi. Il broker media la comunicazione tra le applicazioni, riducendo al minimo la consapevolezza reciproca, ovvero quella che le applicazioni dovrebbero avere l'una rispetto all'altra al fine di poter scambiare messaggi, implementando in modo efficace il disaccoppiamento. Lo scopo principale di un broker é di prendere i messaggi in arrivo dalle applicazioni ed eseguire alcune azioni su di essi. Il broker di messaggi in questo scenario viene utilizzato per scambiare dei messaggi in formato JSON tra l'applicazione Django e il Gateway dell'azienda, e viceversa. Questi messaggi sono le notifiche push da inviare quando vanno verso GW mentre sono dei report sugli invii quando l'applicazione li riceve.

5.2.7 Test dell'applicazione con Postman e Newman

Per poter testare il corretto funzionamento dell'applicazione durante le fasi di sviluppo e deploy é stata realizzata una collezione Postman che verifica il corretto funzionamento delle varie API dell'applicazione. Una collezione é un gruppo di richieste salvate in formato JSON attraverso cui il client Postman contatta l'applicazione nei vari punti da testare. La collezione fa uso delle variabili di ambiente previste in Postman definite con una notazione mustache che permettono di parametrizzare le richieste in maniera tale da poterle eseguire con diverse configurazioni a seconda dell'ambiente: locale, sviluppo, collaudo, etc...

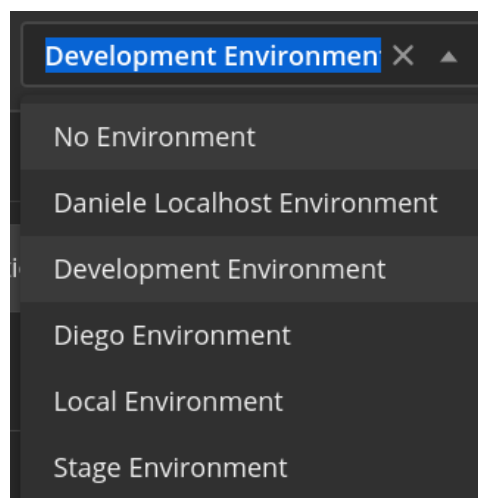


Figura 2: Menu a tendina delle variabili di ambiente

Quando si effettua una chiamata si possono valutare il corpo e lo status della risposta al fine di effettuare i controlli desiderati. In linea di massima per le varie API con i test implementati si verifica sempre che lo status di risposta sia corretto, e per le API che restituiscono oggetti JSON si verifica la coerenza dei dati in essi contenuti. La seguente chiamata di API generica restituisce l'id appartenente a un contatto e il suo indirizzo mail. Il test di questa API implementato nella sezione Test di Postman verifica se lo status della risposta é 200 e che il contact id sia presente:

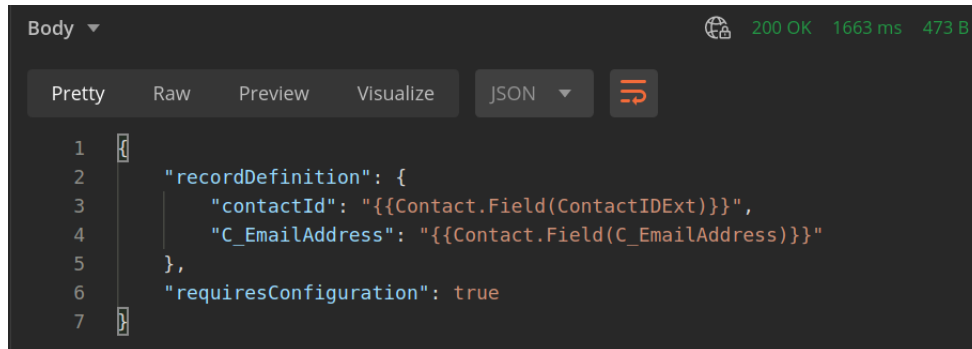


Figura 3: Risposta a una chiamata di API

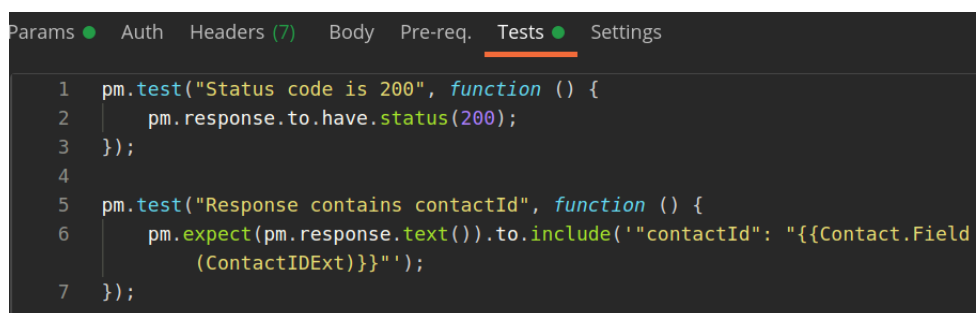


Figura 4: Test in Postman

Per coprire la sezione di test sono state create 19 richieste e un totale di 23 asserzioni. La collezione, cosí come le variabili di ambiente e i test possono essere esportati e immagazzinati sotto forma di oggetti JSON. Fare questo consente di affidarsi a uno strumento Newman che é un collection runner usato per eseguire e testare la collezione di API da riga di comando. In questo modo é possibile integrare quanto esportato con altri strumenti, come per esempio un sistema per la CI.

6 Perché usare Docker per il deploy di applicazioni complesse

Il mercato IT é da sempre caratterizzato dalla continua ricerca di tecnologie e paradigmi in grado di ottimizzare l'intero processo di gestione dei progetti software, in tutti le fasi del suo ciclo di vita. L'incredibile pervasività delle tecnologie IT ha contribuito a rendere questa esigenza ancora piú stringente: sono innumerevoli i mercati in cui le tecnologie IT vengono applicate, ciascuno con esigenze differenti in termini di tempi, strategie di comunicazione e risorse disponibili.

Questo scenario ha portato alla nascita di paradigmi come CI/CD (Continuous Integration e Continuous Delivery), Agile e piú in generale alla formalizzazione del modello DevOps. Questi paradigmi si fondano, seppur non direttamente, su una serie di tecnologie abilitanti, tra le quali spicca per l'appunto Docker.

Prima dell'avvento di Docker, il compito di semplificare la gestione del ciclo di vita del software (ed in particolare di quelli basati su architetture SOA) é stato affidato alle tecnologie di virtualizzazione. Tuttavia, la pratica ha dimostrato come la virtualizzazione abbia in sé una

serie di limitazioni che mal si coniugano con le esigenze delle applicazioni moderne. Se da una parte esse offrono un elevato grado di isolamento (ogni VM é di fatto un'unità a sé) dall'altra comportano un significativo spreco di risorse di calcolo, dato che ogni VM impegna l'hardware ospitante ad eseguire il codice del sistema operativo virtualizzato oltre che delle applicazioni che esegue. Ciò limita fortemente il numero di VM che uno stesso host può ospitare senza degradare le prestazioni generali del sistema. Inoltre, sebbene esistano degli standard aperti quali OVF (Open Virtualization Format), le tecnologie di virtualizzazione oggi presenti sul mercato non sono del tutto interoperabili, rendendo così le VM poco portabili: ad esempio, migrare una VM da un host VMware vSphere ad un host KVM comporta la disinstallazione dell'agent VMware tools dalla VM, la conversione delle immagini disco dal formato VMDK a RAW o QCOW, la creazione di una configurazione QEMU con hardware analogo a quello esposto da VMware (laddove ciò sia possibile) ed infine la riconfigurazione del sistema operativo della VM (installazione di driver ed agent per i nuovi device virtuali). Tutte queste operazioni richiedono degli interventi manuali, anche solo di supervisione, e non sono esenti da rischi[4].

Dalla sua introduzione, Docker ha goduto di un tasso di adozione pressoché costante, divenendo di fatto una tecnologia fondamentale per lo sviluppo di applicazioni moderne. Vengono di seguito illustrate le sue caratteristiche principali e le tecnologie sulle quali si fonda.

7 Docker

Docker é un progetto open source nato con lo scopo di automatizzare la distribuzione di applicazioni sotto forma di "contenitori" leggeri, portabili e autosufficienti che possono essere eseguiti su cloud(pubblici o privati) o in locale.

I contenitori di Docker sono quindi l'insieme dei dati di cui necessita un'applicazione per essere eseguita: librerie, altri eseguibili, rami del file system, file di configurazione, script, ecc.

Il processo di distribuzione di un'applicazione si riduce quindi alla semplice creazione di una immagine Docker, ovvero di un file contenente tutti i dati di cui sopra. L'immagine viene utilizzata da Docker per creare un Container, un'istanza dell'immagine che eseguirá l'applicazione in essa contenuta.

I container sono quindi autosufficienti perché contengono già tutte le dipendenze dell'applicazione e non richiedono quindi particolari configurazioni sull'host, ma sono soprattutto portabili perché essi sono distribuiti in un formato standard(le immagini appunto), che può essere letto ed eseguito da qualunque server Docker.

Inoltre, i container sono leggeri perché sfruttano i servizi offerti dal kernel del sistema operativo ospitante, invece di richiedere l'esecuzione di un kernel virtualizzato come avviene nel caso delle VM. Ciò permette di ospitare un gran numero di container nella stessa macchina fisica.

Data la sua stretta relazione con il kernel Linux, Docker é nativamente disponibile in ambiente GNU/Linux. Tuttavia, essendo uno strumento concepito per gli sviluppatori, si é resa necessaria la possibilità di eseguirlo direttamente sulle workstation, senza imporre limiti sul sistema operativo in uso. Docker é pertanto disponibile anche in ambiente macOS e Windows. In entrambi i casi, si ricorre ad una tecnologia di virtualizzazione per eseguire un kernel Linux al quale si appoggia l'engine Docker[4].

7.1 Il Dockerfile

Un Dockerfile é un semplice file di testo che, con una sintassi semplice e concisa, permette di esprimere le personalizzazioni che si vuole apportare ai vari template affinché questi possano diventare delle immagini su misura. Docker crea immagini automaticamente leggendo le istruzioni da un Dockerfile, esso é un file di testo senza estensione .txt che contiene tutti i comandi in ordine necessari per costruire una determinata immagine.

L'immagine Docker é costituita da livelli di sola lettura, ciascuno dei quali rappresenta un'istruzione Dockerfile. I livelli vengono impilati e ognuno viene creato dalla modifica rispetto

al livello precedente. Ad esempio, se si crea un livello base di Ubuntu e quindi nella seconda istruzione si installa Python, verrà creato un secondo livello. Allo stesso modo, se vengono eseguite delle modifiche con le istruzioni (RUN, COPY, ADD) si creerà un nuovo livello in quell'immagine.

I container sono livelli di lettura e scrittura creati dalle immagini Docker. In parole semplici, un Dockerfile è un insieme di istruzioni che crea un livello impilato per ogni istruzione che crea collettivamente un'immagine (che è un prototipo o un modello per i contenitori)[5].

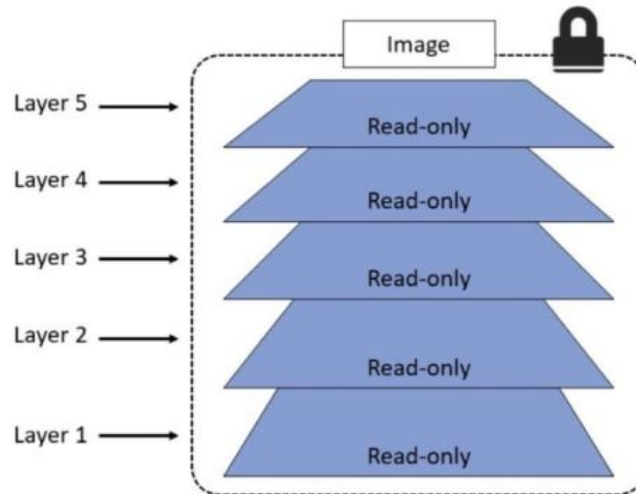


Figura 5: Costruzione di un'immagine Docker



Figura 6: Processo di creazione dei container

7.1.1 Comandi del Dockerfile

- **FROM** - Definisce un'immagine di base, inizializza quindi una nuova fase di compilazione e imposta l'immagine di base per le istruzioni successive. Pertanto, un Dockerfile valido deve iniziare con un'istruzione FROM
- **PULL** - È un comando per scaricare un'immagine Docker o un repository localmente sull'host da un registro pubblico o privato. Quando si esegue un qualsiasi contenitore e l'immagine Docker specificata non è presente localmente, prima la estrae dal registro. La maggior parte delle volte le immagini vengono scaricate da un registro pubblico che è "hub.docker.com" quando si creano le proprie immagini personalizzate poiché si utilizza l'immagine Docker ufficiale come immagine di base
- **RUN** - Consente di eseguire comandi all'interno dell'immagine Docker. Questi comandi vengono eseguiti una volta in fase di compilazione e vengono scritti nell'immagine Docker come un nuovo livello. Ad esempio, se si desidera installare un pacchetto o creare una directory all'interno della propria immagine Docker, RUN è il comando da usare

- **CMD** - Specifica l'istruzione che deve essere eseguita all'avvio di un container Docker. Questo comando non é realmente necessario per il funzionamento del contenitore, poiché il comando `echo` può essere chiamato anche in un'istruzione `RUN`. Lo scopo principale del comando `CMD` é avviare il software richiesto in un container
- **EXPOSE** - Espone la porta specificata e la rende disponibile solo per la comunicazione tra container
- **ENV** - Imposta le variabili di ambiente all'interno dell'immagine
- **COPY** - Viene utilizzato per copiare i file/directory locali sui container Docker
- **ADD** - É una versione più ricca di funzionalità dell'istruzione `COPY`. La principale differenza tra `ADD` e `COPY` é che `ADD` consente di scaricare file remoti attraverso l'URL. Specificando un URL pubblicamente accessibile come percorso di origine, questo file verrà scaricato e aggiunto all'immagine del contenitore
- **ENTRYPOINT** - Viene utilizzato per impostare eseguibili che verranno sempre eseguiti all'avvio del contenitore. A differenza dei comandi `CMD`, i comandi `ENTRYPOINT` non possono essere ignorati o sovrascritti, anche quando il contenitore viene eseguito con gli argomenti della riga di comando dichiarati
- **VOLUME** - Definisce quale directory in un'immagine deve essere trattata come un volume.
- **WORKDIR** - Definisce la directory di lavoro per le istruzioni successive nel Dockerfile (non crea un nuovo livello intermedio nell'immagine)

Ogni istruzione crea un livello. Se é necessario impedire la copia di alcuni file nell'immagine Docker (possono essere informazioni sensibili come file `.env` che contengono chiavi API o qualsiasi altro file non molto importante), é possibile aggiungere un file `.dockerignore` allo stesso livello del Dockerfile in cui é possibile specificare i file che non devono essere copiati nell'immagine Docker. In questo modo, se vengono utilizzate delle istruzioni `COPY` o `ADD` in un Dockerfile per specificare i file da aggiungere a un'immagine Docker, qualsiasi file specificato nel file `.dockerignore` verrà ignorato e non aggiunto all'immagine Docker.

7.2 Volumi

Un volume di un container “vive” al di fuori del container in questione sul computer host. Dal container il volume si comporta come una cartella in cui salvare file o da cui recuperare dati. I volumi che sono nativi di Docker hanno un ciclo di vita più lungo rispetto a quello del contenitore, e consentono ai dati di persistere fino a quando non è più necessario. I volumi possono essere condivisi tra i contenitori. Docker consente di gestire i volumi tramite il set di comandi del volume Docker. É possibile assegnare a un volume un nome esplicito o consentire a Docker di generarne uno casuale.

7.3 Docker Compose

Compose é uno strumento per definire ed eseguire applicazioni Docker multi-container. Con Compose, si utilizza un file YAML per configurare i servizi della propria applicazione. Quindi, con un solo comando, si creano e avviano tutti i servizi dalla propria configurazione. Compose funziona in tutti gli ambienti: produzione, staging, sviluppo, test e flussi di lavoro CI. L'uso di Compose é fondamentalmente un processo in tre fasi[6]:

1. Definizione dell'ambiente dell'app nel Dockerfile in modo che possa essere riprodotta ovunque

2. Definizione dei servizi che compongono l'app in `docker-compose.yml` in modo che possano essere eseguiti insieme in un ambiente isolato
3. Esecuzione del Docker Compose per avviare l'intera app

Compose dispone di una serie di comandi per la gestione dell'intero ciclo di vita della propria applicazione[6]:

- Avvia, arresta e ricostruisci i servizi
- Visualizza lo stato dei servizi in esecuzione
- Trasmette in output i log dei servizi in esecuzione
- Esegue un comando una sola volta su un servizio

Le caratteristiche di Compose che lo rendono efficace sono:

- Più ambienti isolati su un unico host: Compose utilizza un nome di progetto per isolare gli ambienti l'uno dall'altro
- Conserva i dati sul volume quando vengono creati i contenitori: conserva tutti i volumi utilizzati dai servizi. Quando la composizione viene eseguita, se trova contenitori dalle esecuzioni precedenti, copia i volumi dal vecchio contenitore al nuovo contenitore. Questo processo garantisce che tutti i dati che sono stati creati nei volumi non vadano persi
- Ricrea solo i contenitori che sono stati modificati: memorizza nella cache la configurazione utilizzata per creare un contenitore. Quando si riavvia un servizio che non è stato modificato, Compose riutilizza i contenitori esistenti. Riutilizzare i container significa che si possono apportare modifiche all'ambiente molto rapidamente
- Variabili e spostamento di una composizione tra ambienti: supporta le variabili nel file Compose. Queste variabili possono essere utilizzate al fine di personalizzare la composizione per ambienti diversi o utenti diversi

8 Ansible

Ansible è uno strumento software che fornisce un'automazione semplice ma potente per il supporto di computer multiplatforma. È destinato principalmente ai professionisti IT, che lo utilizzano per l'implementazione di applicazioni, aggiornamenti su workstation e server, provisioning cloud, gestione della configurazione, orchestrazione intra-servizio e quasi tutto ciò che un amministratore di sistema fa su base settimanale o giornaliera. Ansible non dipende dal software dell'agente e non dispone di un'infrastruttura di sicurezza aggiuntiva, quindi è facile da implementare.

Poiché Ansible è incentrato sull'automazione, richiede istruzioni per eseguire ogni compito. Con tutto scritto in una semplice forma di script, è facile eseguire il controllo della versione. Il risultato pratico di ciò è un importante contributo al movimento "infrastructure as code" nell'IT: l'idea è che la manutenzione dell'infrastruttura del server e del client possa e debba essere trattata come lo sviluppo del software, con repository di autodocumentazione, comprovata, e soluzioni eseguibili in grado di gestire un'organizzazione indipendentemente dai cambiamenti del personale[7].

Sebbene Ansible possa essere all'avanguardia nell'automazione, nell'amministrazione dei sistemi e nel DevOps, è anche utile per gli utenti di tutti i giorni. Ansible consente di configurare non solo un computer, ma potenzialmente un'intera rete di computer contemporaneamente e il suo utilizzo non richiede competenze di programmazione. Le istruzioni scritte per Ansible sono leggibili dall'uomo. Sia per utenti esperti che non, i file Ansible sono facili da capire.

8.1 Come funziona Ansible

In Ansible esistono due categorie di computer: il nodo di controllo e i nodi gestiti. Il nodo di controllo è un computer che esegue Ansible. Deve essere presente almeno un nodo di controllo, sebbene possa esistere anche un nodo di controllo di backup. Un nodo gestito è qualsiasi dispositivo gestito dal nodo di controllo.

Ansible funziona connettendosi ai nodi (client, server o qualsiasi cosa si stia configurando) su una rete, quindi inviando un piccolo programma chiamato modulo Ansible a quel nodo. Ansible esegue questi moduli su SSH e li rimuove al termine. L'unico requisito per questa interazione è che il nodo di controllo Ansible abbia diritto di accesso ai nodi gestiti. Le chiavi SSH sono il modo più comune per fornire l'accesso, ma sono supportate anche altre forme di autenticazione[7].

8.2 Cosa fa Ansible

Il termine moduli Ansible sembra complesso, ma la maggior parte della complessità è gestita da Ansible e non dall'utente. Un modulo Ansible è scritto per essere un modello dello stato desiderato di un sistema, il che significa che ogni modulo definisce cosa dovrebbe essere vero su un dato nodo gestito. Ad esempio, se un amministratore di sistema decide che su tutte le workstation di un'organizzazione dovrebbe essere installato LibreOffice versione XZ, dipende dal packaging del modulo Ansible determinare se ogni nodo ha LibreOffice XZ su di esso. Se Ansible trova un nodo gestito con LibreOffice XY installato, rileva il sistema operativo ed esegue la routine necessaria per aggiornarlo a LibreOffice XZ. In questo modo, ogni workstation di un'organizzazione può essere aggiornata durante la notte con il software supportato dal reparto IT.

Tuttavia, la manutenzione dell'infrastruttura è qualcosa di più del semplice controllo delle versioni del software. Quando le persone parlano dell'utilizzo di Ansible, ciò a cui si riferiscono in realtà è l'utilizzo dei moduli Ansible, perché quelle sono le parti di Ansible che svolgono attività specifiche[7].

8.3 Playbook di Ansible

Mentre i moduli forniscono i mezzi per portare a termine un'attività, il modo in cui vengono usati è attraverso un playbook Ansible. Un playbook è un file di configurazione scritto in YAML che fornisce istruzioni su ciò che deve essere fatto per portare un nodo gestito nello stato desiderato. I playbook sono pensati per essere semplici, leggibili dall'uomo e autodocumentanti. Sono anche idempotenti, il che significa che un playbook può essere eseguito su un sistema in qualsiasi momento senza avere un effetto negativo su di esso. Se un playbook viene eseguito su un sistema che è già configurato correttamente e nello stato desiderato, il sistema dovrebbe essere ancora configurato correttamente dopo l'esecuzione di un playbook.

Un playbook può essere molto semplice, come questo che installa, come utente privilegiato, il server HTTP Apache su qualsiasi nodo nel gruppo di server web di un reparto IT[7]:

```
---
- name: Apache server installed
  hosts: webservers
  become: yes
```

Figura 7: Esempio di playbook Ansible

I playbook possono anche essere molto complessi, con condizioni e variabili. Tuttavia, poiché la maggior parte del lavoro reale viene svolto dai moduli Ansible, i playbook rimangono brevi, leggibili e chiari anche se possono orchestrare intere reti di nodi gestiti.

8.4 Inventari

Un file di inventario in Ansible definisce gli host e i gruppi di host su cui operano comandi, moduli e attività di un playbook. Il file può essere in uno dei tanti formati a seconda dell'ambiente e dei plug-in Ansible. I formati comuni includono INI e YAML. Il percorso predefinito per il file di inventario è `/etc/ansible/hosts`. È possibile anche creare file di inventario specifici del progetto in posizioni alternative.

Il file di inventario può elencare singoli host o gruppi di host definiti dall'utente. Ciò consente di definire gruppi di dispositivi con ruoli simili su cui eseguire le stesse attività operative e di configurazione.

8.5 Variabili e ruoli

Le variabili sono un concetto in Ansible che consente di modificare il modo in cui funzionano i playbook. Le variabili vengono utilizzate per tenere conto delle differenze tra i sistemi, come le versioni dei pacchetti o i percorsi dei file. Con Ansible, si possono eseguire playbook su diversi sistemi. Le variabili Ansible dovrebbero essere definite in relazione a ciò che sta effettivamente facendo il proprio playbook. Le variabili seguono un ordine di precedenza procedurale, che definisce il modo in cui le variabili si sovrascriveranno a vicenda. È importante capirlo quando si includono variabili nel proprio playbook.

I ruoli Ansible sono un tipo speciale di playbook completamente autonomo e portatile con le attività, le variabili, i modelli di configurazione e altri file di supporto necessari per completare un'orchestrazione complessa. Possono esistere più ruoli all'interno di una raccolta consentendo una facile condivisione dei contenuti.

Quando si lavora con Ansible, è necessario comprendere anche le raccolte. Le raccolte sono un formato di distribuzione per i contenuti Ansible che può includere playbook, ruoli, moduli e plug-in.

8.6 Gestione della configurazione con Ansible

La gestione della configurazione è un processo per mantenere i sistemi informatici, i server e il software in uno stato desiderato e coerente. È un modo per assicurarsi che un sistema funzioni come previsto man mano che le modifiche vengono apportate nel tempo. Tradizionalmente, questo veniva gestito manualmente o con script personalizzati dagli amministratori di sistema.

Gli strumenti di gestione della configurazione velocizzano le modifiche e le implementazioni, eliminano il rischio di errori umani e rendono la gestione del sistema prevedibile e scalabile. Aiutano anche a tenere traccia dello stato delle risorse e impediscono di ripetere attività, come installare lo stesso pacchetto due volte.

L'automazione della gestione della configurazione con Ansible può aiutare a migliorare il ripristino del sistema dopo un evento critico. Se un server si interrompe per un motivo sconosciuto, è possibile distribuirne uno nuovo rapidamente e avere un registro di tutte le modifiche o aggiornamenti che si sono verificati in modo da poter identificare l'origine del problema[8].

8.7 Provisioning con Ansible

È difficile tenere traccia delle modifiche e controllare le versioni ed evitare errori e incoerenze quando il provisioning viene gestito manualmente per ogni distribuzione. Automatizzare il provisioning di qualsiasi infrastruttura può renderlo molto più semplice; dovrebbe essere il primo passo per automatizzare il ciclo di vita operativo delle proprie applicazioni.

Ansible può essere utilizzato per eseguire il provisioning dell'infrastruttura sottostante dell'ambiente, host e hypervisor virtualizzati, dispositivi di rete e server bare metal. Può anche installare servizi, aggiungere host di calcolo ed eseguire il provisioning di risorse, servizi e applicazioni all'interno del cloud.

Si può utilizzare un playbook Ansible per descrivere lo stato desiderato della propria infrastruttura e quindi fare in modo che Ansible ne effettui il provisioning. La codifica dell'infrastruttura tramite l'uso di un playbook assicura di eseguire sempre il provisioning dello stesso ambiente[8].

8.8 Automazione della distribuzione

L'automazione del deployment offre la possibilità di spostare il software tra ambienti di test e produzione utilizzando processi automatizzati. Ciò porta a implementazioni ripetibili e affidabili durante tutto il ciclo di distribuzione del software. L'automazione della distribuzione è un elemento importante per abilitare le pratiche DevOps e gestire una pipeline CI/CD.

L'automazione della distribuzione non funziona quando il team di sviluppo distribuisce applicazioni o configura gli ambienti in un modo e i team operativi distribuiscono e configurano in un altro modo. Affinché l'ambiente possa essere automatizzato, deve essere coerente. Lo stesso processo di distribuzione deve essere utilizzato per ogni ambiente, incluso l'ambiente di produzione.

Una pipeline di distribuzione in genere segue 3 passaggi principali(sebbene se ne potrebbero averne anche di più): compilazione, test e distribuzione. Questa è la pipeline che supporta la capacità di automatizzare il processo di distribuzione e garantisce che il codice passi rapidamente dall'esecuzione del commit alla distribuzione. Ansible consente di distribuire applicazioni multilivello in modo affidabile e coerente, si possono configurare i servizi necessari e inviare gli artefatti delle applicazioni utilizzando i playbook[8].

8.9 Orchestrazione con Ansible

In generale, l'automazione si riferisce all'automazione di una singola attività. Questo è diverso dall'orchestrazione, che è il modo in cui si automatizza un processo o un flusso di lavoro che prevede molti passaggi su più sistemi disparati.

L'orchestrazione del cloud può essere utilizzata per fornire o distribuire server, assegnare capacità di archiviazione, creare macchine virtuali e gestire il networking, tra le altre attività. Esistono molti strumenti di orchestrazione diversi che possono aiutare con l'orchestrazione cloud. Ansible è un'opzione.

Le implementazioni delle applicazioni possono includere servizi front-end e back-end, database, monitoraggio, reti e storage, ciascuno con il proprio ruolo da svolgere e la propria configurazione e distribuzione. L'orchestrazione garantisce che ogni passaggio avvenga nel modo in cui se ne ha bisogno. Ansible abilita l'orchestrazione eseguendo le attività nel proprio playbook nell'ordine in cui sono scritte, in modo da sapere che i processi di distribuzione delle applicazioni avverranno nell'ordine corretto.

8.10 Vantaggi e svantaggi di Ansible

8.10.1 Vantaggi di Ansible

- Semplice da imparare: la prima menzione tra i vantaggi di Ansible si riferisce alla sua semplicità. La semplicità non è pensata solo per i professionisti ma anche per i principianti. È facile da imparare e quindi gli utenti possono imparare a utilizzare Ansible rapidamente insieme a una migliore produttività. Ansible possiede inoltre il supporto di una documentazione completa e facilmente interpretabile
- Linguaggio Python facilmente comprensibile: uno dei principali vantaggi di Ansible si riferisce anche alla lingua in cui è scritto. Python è un linguaggio leggibile dall'uomo e funge da base per Ansible. Fornisce strutture migliori per avviare Ansible ed eseguirlo grazie alla presenza di librerie Python sulla maggior parte delle distribuzioni Linux per impostazione predefinita

- Python é un'alternativa altamente ideale per attività di amministrazione e scripting che implicano una maggiore popolarità tra ingegneri e amministratori di sistema. Un altro aspetto interessante di Ansible é la facilità dei moduli Ansible che possono migliorarne la funzionalità. I moduli Ansible possono essere scritti in qualsiasi lingua. Tuttavia, la preoccupazione importante, in questo caso, é che il modulo debba restituire i dati in formato JSON
- Nessuna dipendenza dagli agenti: un'aggiunta importante tra i vantaggi di Ansible si riferisce alla sua natura senza agenti. Ansible gestisce tutte le comunicazioni master-agent tramite il modulo Standard SSH o Paramiko. Il modulo Paramiko é un'implementazione Python di SSH2 ed é fondamentale per la gestione dei nodi. Pertanto, Ansible non richiede alcuna forma di agenti installati su sistemi remoti per garantire la gestione. Di conseguenza, le spese generali di manutenzione e il degrado delle prestazioni si riducono considerevolmente con enormi margini con Ansible
- I playbook sono scritti in YAML: l'uso dei Playbook in Ansible é anche un altro motivo dei principali vantaggi di Ansible. I playbook sono file di configurazione Ansible e la lingua per scriverli é YAML. Il fattore interessante, in questo caso, é che YAML é un'alternativa migliore per la gestione della configurazione e l'automazione
- La superiorità di YAML rispetto ad altri formati come JSON rende Ansible uno strumento di automazione e gestione della configurazione migliore. Ansible semplifica la lettura e supporta i commenti. Più importante di tutti, include anche l'uso di ganci(hooks) per fare riferimento ad altri elementi
- Ansible Galaxy: un'altra voce degna di nota nelle migliori pratiche di Ansible si riferisce alla Ansible Galaxy. Ansible Galaxy é un portale che funge da repository centrale per l'individuazione, il riutilizzo e la condivisione di contenuti relativi ad Ansible. Il miglior vantaggio di Ansible Galaxy é nell'esempio del download di ruoli riutilizzabili per l'installazione della configurazione dell'applicazione o del server. I download sono ideali per l'uso nei playbook di un determinato utente e possono contribuire in modo sostanziale a un aumento della velocità di distribuzione[9]

8.10.2 Svantaggi di Ansible

Dopo una chiara riflessione sui diversi vantaggi di Ansible, si pone attenzione sulle battute d'arresto che si possono incontrare con esso:

- Interfaccia utente insufficiente: la prima voce degli svantaggi di Ansible é l'interfaccia utente grezza. Ansible inizialmente era uno strumento solo da riga di comando. Il primo sforzo di Ansible nel creare un'interfaccia utente é stato con l'interfaccia utente grafica AWX. L'altro componente nell'interfaccia utente era l'endpoint REST pensato per semplificare la gestione dell'infrastruttura.

Successivamente, l'AWX si é trasformato in Ansible Tower, che é un'interfaccia utente di gestione web. Ansible Tower offre funzionalità di gestione visiva e uno strumento per il flusso di lavoro basato sul team. Tuttavia, Ansible Tower richiede notevoli miglioramenti. Ad esempio, quasi l'85% delle attività che potrebbero essere completate tramite la riga di comando possono essere eseguite tramite l'interfaccia utente.

Ci si potrebbe anche imbattere in un'altra menzione degli svantaggi di Ansible derivanti dalla sua interfaccia utente. L'errore di sincronizzazione tra la GUI e la riga di comando può portare a risultati di query in conflitto. In generale, Ansible Tower é ancora in fase di sviluppo e non potrebbe fare tutto come un'interfaccia a riga di comando

- Mancanza di qualsiasi nozione di stato: un'altra importante menzione tra gli svantaggi di Ansible é la mancanza di qualsiasi nozione di stato. Ansible non ha alcuna nozione di

stato come altri strumenti di automazione ad esempio Puppet. Ansible non tiene traccia delle dipendenze ed esegue semplicemente attività sequenziali e si interrompe quando le attività finiscono, falliscono o si verifica un errore.

Queste caratteristiche non sono l'ideale per gli utenti che desiderano che lo strumento di automazione mantenga un catalogo dettagliato per l'ordinazione. Il catalogo può aiutare a raggiungere uno stato specifico senza alcuna influenza dei cambiamenti delle condizioni ambientali. Tuttavia, ad Ansible manca e presenta un formidabile svantaggio

- Supporto Windows limitato: la prossima menzione di spicco tra gli svantaggi di Ansible è il supporto di Windows semicostruito. Ansible versione 1.7 supporta Windows e nodi Linux/Unix. Nel caso di Windows, Ansible utilizza un telecomando PowerShell nativo anziché SSH. Di conseguenza, una macchina di controllo Linux è obbligatoria per la gestione degli host Windows. Il supporto limitato per Windows in Ansible presenta una delle formidabili battute d'arresto con lo strumento di gestione della configurazione e automazione
- Ansible ha poca esperienza: la mancanza di esperienza di supporto alle imprese riduce anche l'attrattiva di Ansible. Ansible non ha un'esperienza lavorativa a tutti gli effetti con grandi imprese come i suoi concorrenti, come Puppet e Chef. Anche se Ansible rivendica la possibilità di opzioni di supporto esteso di livello aziendale, l'esperienza pratica limitata riduce la responsabilità di Ansible
- Ansible è una novità sul mercato: è stato lanciato nel 2012, a differenza dei suoi rinomati concorrenti. Di conseguenza, non ha una grande comunità di sviluppatori o utenti. Inoltre, la nuova presenza di Ansible sul mercato implica la possibilità di bug sconosciuti, problemi software e scenari marginali[9]

Si può concludere che Ansible fornisce un pacchetto semplice e potente per la gestione e l'automazione della configurazione. Tuttavia, Ansible è nuovo sul mercato ora e deve fare i conti con la concorrenza di fonti rinomate. Le diverse funzionalità di Ansible, come il provisioning, l'orchestrazione, la distribuzione delle applicazioni, la sicurezza e la conformità, ne mostrano il potenziale. Queste capacità di Ansible possono tradursi efficacemente in uno strumento DevOps completo. Il futuro corso di Ansible dipenderebbe dal rafforzamento dei vantaggi di Ansible mentre si lavora sulle battute d'arresto.

9 Jenkins. Spiegazione del server per l'integrazione continua

Jenkins offre un modo semplice per configurare un ambiente di integrazione continua o distribuzione continua(CI/CD) per quasi tutte le combinazioni di linguaggi e repository di codice sorgente utilizzando le pipeline, oltre ad automatizzare altre attività di sviluppo di routine. Sebbene Jenkins non elimini la necessità di creare script per i singoli passaggi, esso offre un modo più rapido e robusto per integrare l'intera catena di strumenti di compilazione, test e distribuzione.

9.1 Hudson e Jenkins

Nel 2004, Kohsuke Kawaguchi era uno sviluppatore Java presso Sun. Kawaguchi si era stancato di rompere le build nel suo lavoro di sviluppo e voleva trovare un modo per sapere, prima di inviare il codice al repository, se il codice avrebbe funzionato. Quindi Kawaguchi ha costruito un server di automazione in e per Java per renderlo possibile, chiamato Hudson. Hudson è diventato popolare alla Sun e si è diffuso ad altre società come open source.

Andando avanti velocemente fino al 2011, una disputa tra Oracle(che aveva acquisito Sun) e la comunità open source indipendente di Hudson ha portato a un fork con un cambio di nome, Jenkins. Nel 2014 Kawaguchi è diventato CTO di CloudBees, che offre prodotti a consegna continua basati su Jenkins.

Entrambi i fork continuarono ad esistere, sebbene Jenkins fosse molto più attivo. Oggi il progetto Jenkins è ancora attivo. Il sito web di Hudson è stato chiuso il 31 gennaio 2020.

Nel marzo 2019 la Linux Foundation, insieme a CloudBees, Google e una serie di altre società, ha lanciato una nuova fondazione software open source chiamata Continuous Delivery Foundation (CDF). I collaboratori di Jenkins hanno deciso che il loro progetto avrebbe dovuto unirsi a questa nuova fondazione. Kawaguchi scrisse all'epoca che nulla di significativo sarebbe cambiato per gli utenti.

Nel gennaio 2020 Kawaguchi ha annunciato che si sarebbe trasferito nella sua nuova startup, Launchable. Ha anche detto che si sarebbe ufficialmente ritirato da Jenkins, pur rimanendo nel Technical Oversight Committee della Continuous Delivery Foundation, e passando il suo ruolo in CloudBees a un consulente[10].

9.2 Automazione con Jenkins

Oggi Jenkins è il principale server di automazione open source con circa 1.600 plug-in per supportare l'automazione di tutti i tipi di attività di sviluppo. Il problema che Kawaguchi stava originariamente cercando di risolvere, l'integrazione continua e la distribuzione continua del codice Java (ovvero la creazione di progetti, l'esecuzione di test, l'analisi del codice statico e la distribuzione) è solo uno dei tanti processi che le persone automatizzano con Jenkins. Questi 1.600 plug-in coprono cinque aree: piattaforme, interfaccia utente, amministrazione, gestione del codice sorgente e, più frequentemente, gestione della build[10].

9.3 Come funziona Jenkins

Jenkins è distribuito come archivio WAR e come pacchetti di installazione per i principali sistemi operativi, come pacchetto Homebrew, come immagine Docker e come codice sorgente. Il codice sorgente è principalmente Java, con alcuni file Groovy, Ruby e Antlr.

È possibile eseguire Jenkins WAR standalone o come servlet in un server di applicazioni Java come Tomcat. In entrambi i casi, produce un'interfaccia utente Web e accetta chiamate alla sua API REST.

Quando si esegue Jenkins per la prima volta, occorre creare un utente amministrativo con una password casuale, che bisogna poi incollare nella sua pagina Web iniziale per sbloccare l'installazione. Una volta installato, Jenkins consente di accettare l'elenco dei plug-in predefinito o di sceglierne altri.

9.4 Jenkins pipelines

Una Jenkins Pipeline(o semplicemente "Pipeline") è una suite di plugin che supporta l'implementazione e l'integrazione di pipeline di distribuzione continua in Jenkins.

Una pipeline di distribuzione continua è un'espressione automatizzata del processo per ottenere il software dal controllo della versione fino a utenti e clienti.

Una pipeline Jenkins fornisce un set estensibile di strumenti per la modellazione di pipeline di distribuzione da semplici a complesse "come codice". La definizione della pipeline viene in genere scritta in un file di testo(chiamato Jenkinsfile) che a sua volta viene archiviato nel repository del controllo del codice sorgente di un progetto[11].

Sebbene sia possibile utilizzare l'interfaccia utente Web per creare script, la procedura consigliata corrente consiste nel creare uno script della pipeline denominato Jenkinsfile, e archivarlo nel repository.

Le pipeline Jenkins possono essere create in maniera dichiarativa oppure attraverso uno script. Una pipeline dichiarativa, la più semplice delle due, utilizza la sintassi compatibile con Groovy. Questa inizia con un blocco pipeline, definisce un agente e degli stage che includono gli step che vengono eseguiti.

- pipeline: è il blocco esterno obbligatorio per invocare il plug-in della pipeline Jenkins

- agent: definisce dove si desidera eseguire la pipeline
- any: usato insieme ad agent dice di utilizzare qualsiasi agente disponibile per eseguire la pipeline o la fase
- stages: contiene una sequenza di una o più direttive di stage
- steps: fanno il lavoro vero e proprio, sono le azioni da compiere relative ad uno stage
- post: definisce le azioni che verranno eseguite al termine dell'esecuzione o della fase della pipeline. È possibile utilizzare una serie di blocchi post-condizione all'interno della sezione post: always, changed, failure, success, unstable, e aborted

La pipeline dichiarativa può esprimere la maggior parte di ciò che è necessario per definire le pipeline ed è molto più facile da imparare rispetto alla sintassi della pipeline con script.

9.5 Perché usare Jenkins

Il plug-in Jenkins Pipeline descritto supporta un caso d'uso CI/CD, che è probabilmente l'uso più comune per Jenkins. Esistono considerazioni specifiche per altri casi d'uso.

I progetti Java erano la ragion d'essere originale per Jenkins. Jenkins supporta la costruzione con Maven e funziona anche con Ant, Gradle, JUnit, Nexus e Artifactory.

Android esegue una sorta di Java, ma introduce il problema di come eseguire i test sull'ampia gamma di dispositivi Android. Il plug-in dell'emulatore Android consente di creare e testare su tutti i dispositivi emulati che si desidera definire. Il plug-in Google Play Publisher consente di inviare build a un canale alfa in Google Play per il rilascio o ulteriori test su dispositivi reali.

È possibile specificare un container Docker come agente per una pipeline in cui eseguire Jenkins all'interno di un Docker container. I contenitori Docker sono molto utili in un ambiente Jenkins per migliorare velocità, scalabilità e coerenza.

Esistono due casi d'uso principali per Jenkins e GitHub. Uno è l'integrazione della build, che può includere un hook di servizio per attivare Jenkins su ogni commit nel proprio repository GitHub. Il secondo è l'uso dell'autenticazione GitHub per controllare l'accesso a Jenkins tramite OAuth.

Jenkins supporta molti altri linguaggi oltre a Java. Per C/C++, sono disponibili plug-in per acquisire errori e avvisi dalla console, generare script di compilazione con CMake, eseguire unit test ed eseguire analisi del codice statico. Jenkins ha anche una serie di integrazioni con gli strumenti PHP.

Sebbene il codice Python non debba essere compilato (a meno che non si stia utilizzando Cython, ad esempio), è utile che Jenkins si integri con gli strumenti di test e reporting Python, come Nose2 e Pytest, e per la qualità del codice strumenti come Pylint. Allo stesso modo, Jenkins si integra con strumenti Ruby come Rake, Cucumber, Brakeman e CI::Reporter[10].

9.6 Jenkins per CI/CD

Nel complesso, Jenkins offre un modo semplice per configurare un ambiente CI/CD per praticamente qualsiasi combinazione di linguaggi e repository di codice sorgente utilizzando le pipeline, oltre ad automatizzare una serie di altre attività di sviluppo di routine. Sebbene Jenkins non elimini la necessità di creare script per i singoli passaggi, esso offre un modo più rapido e robusto per integrare l'intera catena di strumenti di compilazione, test e distribuzione di quanto si potrebbe fare altrimenti.

9.7 Vantaggi e svantaggi di Jenkins

9.7.1 Vantaggi di Jenkins

Jenkins non è l'unica opzione per l'integrazione continua nel mercato; tuttavia, sono disponibili anche altri strumenti per fare lo stesso. Ma Jenkins è il più importante e il più utilizzato

grazie al suo utilizzo e alle sue straordinarie caratteristiche. É anche conosciuto come il cuore della pipeline DevOps. Ci sono molti vantaggi nell'usare la pipeline Jenkins, alcuni di essi sono riportati di seguito[12]:



Figura 8: Vantaggi di Jenkins

- Open Source e gratuito: gratuito, open source ed é indipendente dalla piattaforma
- Plug-in e integrazione: é una delle caratteristiche piú importanti che lo rendono il piú utilizzato. Ha il suo tipo di plug-in, che aiuta molto lo sviluppatore nell'esecuzione dei lavori. I plug-in Jenkins possono essere sviluppati da chiunque e per chiunque. Il plug-in di visualizzazione dashboard, il plug-in di analisi del test, il plug-in di build pipeline e molti altri simili supportano lo sviluppatore nell'uso dello strumento Jenkins
- Opzione di hosting: é un'altra importante caratteristica di Jenkins, che può essere installata su qualsiasi sistema operativo come Windows, MacOS, Linux, ecc. É possibile eseguire Jenkins sul cloud scaricandolo e distribuendolo su una VM. Si può anche utilizzare un contenitore Docker al suo interno
- Supporto della comunità: Jenkins ha un grande supporto dalla comunità degli sviluppatori. Si può presumere che la sua popolarità e il supporto della community abbiano piú di 1000k utenti in tutto il mondo
- Integrazione con altre piattaforme CI/CD: Jenkins supporta molte piattaforme CI/CD, non solo la pipeline. Può anche interagire con altri strumenti. Sono disponibili diversi plug-in, che consentono agli utenti di effettuare connessioni con altre piattaforme CI/CD
- Mantiene sincronizzato il team: Jenkins si concentra su un modo di lavorare centralizzato. Tutti i membri del team si muovono in sincronia
- Debug facile da eseguire: é molto facile scoprire gli errori in Jenkins. Lo sviluppatore può facilmente controllare il bug e risolverlo

- Meno tempo per consegnare il progetto: succede grazie alla sua funzione di integrazione continua
- Flessibile nella creazione dei progetti: é molto flessibile, può creare progetti(Jobs) sia in maniera libera che in fase di elaborazione della pipeline molto facilmente
- Gestione del codice sorgente(SCM): Jenkins supporta diversi tipi di repository di codice sorgente come SVN, Git, ecc. Lo sviluppatore può impostare trigger diversi dopo aver apportato modifiche ai codici
- Rende molto semplice il processo di conversione nella GUI dalla CLI
- Fornisce un supporto dati accurato alla gestione del progetto
- Supporta molti linguaggi, come Java, Python, ecc

9.7.2 Svantaggi di Jenkins

Di seguito sono riportati alcuni svantaggi dello strumento Jenkins:

- Tutti i plug-in non sono compatibili con la sintassi della pipeline dichiarativa
- Jenkins ha molti plug-in nella sua libreria, ma sembra che di tanto in tanto non vengano gestiti dai team di sviluppatori. É importante sapere se i plug-in che si stanno utilizzando stanno ricevendo aggiornamenti regolari o meno
- Molti plug-in hanno un problema con il processo di aggiornamento
- La forte dipendenza dai plug-in; a volte, non si riescono a trovare nemmeno gli strumenti di base senza i plug-in
- L'interfaccia utente di Jenkins non é facile da usare rispetto alle attuali tecnologie dell'interfaccia utente
- La gestione della dashboard di Jenkins diventa difficile quando si hanno tanti progetti da eseguire
- Le modifiche apportate da uno sviluppatore non saranno visibili a un altro sviluppatore del team; solo il gestore può avere il suo accesso. Rende difficile il monitoraggio nei progetti di grandi dimensioni
- Sebbene siano disponibili molti plug-in, alla fine non fornisce il supporto analitico per completare il ciclo di distribuzione. Porta alla creazione di difficoltà nel monitoraggio della distribuzione del progetto
- Il messaggio di errore a volte non é chiaro. Diventa difficile scoprire cosa é andato storto
- Non é facile installare e configurare lo strumento Jenkins

10 Ansible contro Jenkins

Dalle precedenti descrizioni di Ansible e Jenkins, diventa evidente che entrambi gli strumenti aiutano l'implementazione delle pratiche DevOps in ambienti complessi. Sebbene il loro scopo sia leggermente diverso, gran parte del loro utilizzo si sovrappone. Di seguito é riportato un confronto di questi strumenti.

Il piú delle volte, Ansible e Jenkins vengono usati insieme piuttosto che come sostituti l'uno dell'altro. Molti team in tutto il mondo utilizzano Ansible per l'orchestrazione, mentre Jenkins viene utilizzato per l'automazione di compilazione e rilascio.

Tabella 1: Confronto delle caratteristiche di Ansible e Jenkins

Caratteristiche	Ansible/Ansible Tower	Jenkins
Scopo	Gestione della configurazione, distribuzione delle applicazioni e automazione dell'infrastruttura	Automazione del processo di creazione e rilascio e orchestrazione di pipeline
Contributori	Redhat e la comunità open source	Cloudbees, Open Source Lab, CD.Foundation, Redhat, AWS, Github, Jfrog, la comunità open source
Disponibilità dell'interfaccia utente	Sì, Ansible Tower è la dashboard dell'interfaccia utente per Ansible	Sì, è una web dashboard

Tabella 2: Confronto delle caratteristiche di Ansible e Jenkins

Sistema operativo supportato	Linux, macOS e Windows. Per Windows, Ansible si integra con Powershell ed è un po' ingombrante da usare	Linux, Windows e macOS
Costruito in	Python	Java
Installazione	L'installazione di Ansible è facile. Impostare Tower può essere un po' impegnativo	Facile configurazione con pacchetti nativi per tutti i sistemi operativi
Sintassi	Sintassi YAML per playbook	Sintassi Groovy per Pipeline-as-a-code
Configurazione	Lavori e sistema facili da configurare	La configurazione è un po' complicata a causa delle numerose opzioni fornite
Peso dell'applicazione	Leggero rispetto a Jenkins	Applicazione pesante
Supporto plug-in	Numero decente di plugin disponibili	Il supporto per i plugin è vasto. Quasi tutti gli strumenti possono essere integrati con Jenkins
Adatto per la containerizzazione?	Sì	Sì
Licenza	Licenza commerciale proprietaria	Licenza MIT

In base ai requisiti, entrambi gli strumenti possono essere altamente efficaci nei loro domini. Sebbene Ansible Tower possa essere considerato un sostituto di Jenkins, non ha ancora l'enorme supporto per i plug-in di quest'ultimo. Al contrario, Jenkins non sarà in grado di gestire architetture complesse con la stessa facilità di Ansible Tower.

Oggi, i team devono considerare i limiti di entrambi gli strumenti e quindi decidere quale strumento si adatta meglio alle loro esigenze.

- Per ambienti complessi con un vasto numero di server, Ansible sarebbe lo strumento ideale. Il suo efficace sistema di gestione dell'inventario aiuta a gestire meglio il sistema
- Per attività più piccole che implicano la chiamata di più strumenti, come build, analisi della qualità del codice e test, Jenkins sarebbe un'opzione migliore
- Entrambi gli strumenti hanno un supporto della comunità ampio e affidabile. Ma Ansible ha l'esperienza e la potenza di Redhat alle spalle

11 Parte pratica

11.1 Dockerizzazione dell'applicazione

11.1.1 Definizione del Dockerfile per il database PostgreSQL

Il seguente dockerfile definisce come creare l'immagine per il database dell'applicazione. Lo script `.sh` che viene copiato nella cartella `/docker-entrypoint-initdb.d/` ha il compito di creare il database e l'utente che può operare su di esso.

```
1 FROM postgres:13.2
2 ADD initdb.sh /docker-entrypoint-initdb.d
3 WORKDIR /home
```

11.1.2 Definizione del Dockerfile per la webapp

Dopo aver definito l'immagine di base, viene utilizzato il manager di pacchetti Yum che si occupa di cercare i pacchetti e le loro dipendenze per poi eseguirne l'installazione.

Attraverso i successivi comandi RUN vengono installati e configurati Python e il server OpenSSH.

PAM(Pluggable Authentication Modules) viene usato per definire un meccanismo di autenticazione. L'idea è quella di limitare chi può collegarsi con il demone ssh in base a un elenco di utenti. Viene utilizzato per migliorare la sicurezza.

Nell'ultima riga viene lanciato il processo sshd che viene esposto sulla porta specificata.

```
1 FROM amazonlinux:2
2 RUN yum update -y
3 RUN amazon-linux-extras enable python3.8
4 RUN yum install -y sudo openssh-server
5 RUN sudo yum -y install python3.8
6 RUN mkdir /var/run/sshd
7 RUN echo 'root:screencast' | chpasswd
8 RUN sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin yes/'
   /etc/ssh/sshd_config
9
10 RUN sed 's@session\s*required\s*pam_loginuid.so@session optional pam_loginuid.so@g'
   -i /etc/pam.d/sshd
11 RUN ssh-keygen -A
12 ENV NOTVISIBLE "in users profile"
13 RUN echo "export VISIBLE=now" >> /etc/profile
```

```
14
15 EXPOSE 22
16 CMD /usr/sbin/sshd -D
```

11.1.3 Definizione del Docker Compose

Eseguendo il comando `docker-compose up` si esegue il setup dell'intera applicazione. All'interno del file `docker-compose.yml` troviamo:

- `version`: indica la versione della sintassi del file
- `services`: é la sezione che contiene i servizi da eseguire
- `web-aws-ansible`, `postgres` e `redis`: sono i nomi dei servizi da eseguire, il loro contenuto descrive come Docker deve avviare i container per questi servizi
- `build`: il compito del comando `build` é quello di preparare le immagini per creare contenitori, quindi se un servizio utilizza un'immagine precompilata, salterá questo comando
- `volumes`: viene usato per mappare i file dell'applicazione all'esterno del contenitore in maniera tale da agevolarne l'accesso e per ottenere la persistenza
- `ports`: mappa le porte dell'host con le porte dei container `HOST:CONTAINER`
- `depends_on`: implica le dipendenze tra i vari servizi spiegando al Compose quali container devono essere avviati per primi. Tuttavia `docker-compose` non aspetta che i container `postgres` e `redis` siano pronti prima di avviare il container `webapp`, per ottenere ciò é necessario utilizzare uno script che ritarda l'avvio o altro
- `deploy`: questa sezione definisce i requisiti runtime per un servizio
- `restart_policy`: configura se e come riavviare i contenitori in caso di uscita:
 - `condition`: é la condizione che indica in quale caso dev'essere eseguita la politica per il riavvio
 - `delay`: é il tempo di attesa tra i tentativi di riavvio
 - `max_attempts`: indica quante volte tentare di riavviare un container prima di arrendersi(come predefinito non si arrende mai)
 - `window`: indica quanto tempo attendere prima di decidere se un riavvio é riuscito, specificato come durata(di predefinito decide immediatamente)
 - `environment`: serve per definire delle variabili che vengono poi rese disponibili ai relativi container

```
1  version: "3.9"
2  services:
3  web-aws-ansible:
4  build: ./webapp
5  volumes:
6  - ./webapp/log:/home/app_pn/site/log
7  - ./webapp/site:/home/app_pn/site
8  ports:
9  - "5000:22"
10 - "80:80"
11 - "5678:5678"
12 depends_on:
```

```
13 - postgres
14 - redis
15 deploy:
16   restart_policy:
17     condition: on-failure
18   delay: 3s
19   max_attempts: 5
20   window: 60s
21   environment:
22     - PGPASSWORD=postgres
23   postgres:
24     build: ./postgres
25   ports:
26     - "5532:5432"
27   environment:
28     - POSTGRES_DB=xxxxxx
29     - POSTGRES_USER=xxxxxx
30     - POSTGRES_PASSWORD=xxxxxx
31   redis:
32   image: redis:6.2.1-alpine
```

11.2 Setup e deploy con Ansible

11.2.1 Inventari

Viene creato un file di inventario per ogni ambiente su cui eseguire l'applicazione. Gli ambienti configurati sono sviluppo, collaudo e produzione, su tutti i server é stato lanciato manualmente il Docker Compose precedentemente creato. Per ogni ambiente é definito un file di host che contiene gli ip delle macchine e un file per le variabili di configurazione. Nel file di inventario vengono specificati il tipo di connessione da usare con il comando `ansible_connection: docker` e il nome del container a cui connettersi. Quando viene lanciato un playbook si passa come parametro la variabile di ambiente `DOCKER_HOST`, questa serve per indicare ad Ansible su quale socket connettersi al demone Docker.

```
1 prod:
2   hosts:
3     127.0.0.1:
```

```
1 hosts:
2   children:
3     app:
4     hosts:
5     docker_compose_web-aws-ansible_1:
6     ansible_connection: docker
```

11.2.2 Playbook

Il playbook `deploy.yml` serve per rilasciare l'applicazione sui vari ambienti, definito come segue. Il comando `hosts: all` serve a includere gli host presenti nell'inventario mentre con `become: true` si acquisiscono i permessi necessari per eseguire le operazioni. Nella sezione di `pre_tasks` si installa `systemctl` che é il tool usato per gestione dei `systemd` necessari all'applicazione. Nella sezione `tasks` viene usato `dbus.service` per gestire la comunicazione tra `systemctl` e `systemd` dopodiché viene lanciato il ruolo dedicato al rilascio dell'applicazione `deploy_linux.yml`

```
1 - hosts: all
```

```

2   become: true
3   pre_tasks:
4   - name: install systemctl.py
5   copy: src="systemctl.py" dest="/usr/bin/systemctl"
6
7   - name: ensure run directory for ansible check_systemd
8   file: name="/run/systemd/system/" state="directory"
9   tasks:
10  - name: check some service for systemctl.py
11  service: name="dbus.service" state="stopped"
12
13  - name: "Including deploy role for Linux"
14  include: ../../roles/app/tasks/deploy_linux.yml

```

Il secondo playbook site.yml si occupa di apportare le configurazioni necessarie per l'installazione dell'applicazione. Il suo preambolo é identico a quello appena descritto, la sua sezione tasks si conclude eseguendo il ruolo main.yml che richiama a sua volta il ruolo linux.yml per configurare l'ambiente dell'applicazione da zero.

11.2.3 Role: deploy_linux.yml

Nel ruolo per il deploy si eseguono i seguenti passaggi:

1. Pulisce la cartella /tmp dove viene scaricato il repository
2. Clona il repository
3. Muove la release nella cartella di destinazione
4. Installa le dipendenze dell'applicazione attraverso il gestore di pacchetti PIP di Python
5. Aggiorna il file di settings dell'applicazione
6. Chiama il ruolo postgres.yml per eseguire il setup del database
7. Attiva il virtual enviromental per l'applicazione Python e usa un comando di gestione per eseguire copia dei file statici
8. Riavvia l'applicazione
9. Imposta i permessi dell'applicazione
10. Esegue un riavvio del demone systemd per ricaricare le configurazioni
11. Riavvia i processi batch dell'applicazione che sono:
 - processo beat: gestisce i processi anche su più macchine
 - processo worker: esegue i task dell'applicazione
 - processo reports-cluster: riceve i messaggi di reportistica dal Gateway

```

1   - name: "sw release: cleaning local repository"
2   local_action: file path=/tmp/app/{{ git_branch }} state=absent
3   become: false
4   run_once: true
5
6   - name: "sw release: git cloning repo app"
7   local_action: command git clone --branch {{ git_branch }} {{ git_url }} /tmp/app/{{
      git_branch }}
8   become: false

```

```

9     run_once: true
10
11     - name: "sw release: moving release on server {{ ansible_host }}"
12       synchronize:
13         src: /tmp/app/{{ git_branch }}/
14         dest: /home/app/site/app/
15         copy_links: yes
16         checksum: yes
17
18     - name: "Installing pip requirements"
19       pip:
20         virtualenv: /home/app/virtualenv
21         requirements: /home/app/site/app/requirements.txt
22
23     - name: "Updating /home/app/site/app/app/local_settings.py"
24       template:
25         backup: yes
26         src: ../templates/linux/local_settings.py
27         dest: /home/app/site/app/app/local_settings.py
28         owner: xxxxxx
29         group: xxxxxx
30         tags: localsettings
31
32     - name: "Database setup"
33       include_tasks: postgres.yml
34
35     - name: "Copy everything from static"
36       shell: . /home/app/virtualenv/bin/activate && /home/app/site/app/manage.py
37             collectstatic --noinput
38
39     - name: "Restarting app"
40       systemd:
41         state: restarted
42         name: httpd
43         tags: apache-restart
44
45     - name: "Fixing files permissions"
46       file:
47         path: /home/app/site
48         state: directory
49         owner: xxxxxx
50         group: xxxxxx
51         recurse: yes
52
53     - name: "Systemd: daemon-reload"
54       systemd:
55         daemon_reload: yes
56         tags: systemd
57
58     - name: "Restarting services: app-beat"
59       systemd:
60         name: app-beat
61         state: restarted
62         tags: systemd
63
64     - name: "Restarting services: app-worker"
65       systemd:
66         name: app-worker
67         state: restarted

```

```
67     tags: systemd
68
69     - name: "Restarting services: app-reports-ter"
70       service:
71         name: app-reports-ter
72         state: restarted
73         tags: systemd
```

11.2.4 Role: linux.yml

Il ruolo contiene le operazioni da effettuare per configurare l'ambiente da zero e rilasciare poi l'applicazione.

1. Abilita l'installazione di Python
2. Installa i pacchetti necessari
3. Installa il client Postgres
4. Crea l'utente per l'applicazione
5. Assegna i permessi all'utente
6. Crea la home directory dell'applicazione
7. Crea la cartella per i log applicativi
8. Crea la directory .well-known dedicata a contenere i prefissi degli URL noti che vengono usati dall'applicazione
9. Installa il virtual host
10. Configura il web server Apache
11. Crea i processi batch dell'applicazione che sono beat, worker ed reports
12. Riavvia i processi systemd
13. Installa i driver psycopg2 per PostgreSQL
14. Installa il virtual enviromental
15. Crea il virtual enviromental per l'applicazione
16. Installa mod_wsgi e lo configura. É un modulo di Apache che esegue l'hosting dell'applicazione
17. Crea una directory per le settings dell'applicazione e assegna i permessi

Alla termine di questi step la configurazione dell'ambiente é completata e si esegue quindi il deploy attraverso il ruolo deploy_linux.yml

```
1     - name: "Enable python 3.8 on amazon linux"
2     shell: "amazon-linux-extras enable python3.8"
3
4     - name: Installing packages
5     yum:
6       name: "{{ packages }}"
7       bugfix: yes
8       state: latest
```

```

9     vars:
10    packages:
11      - python-pip
12      - python3-pip
13      - python38-devel
14      - python-setuptools
15      - httpd-devel
16      - gcc
17      - rsync
18      - which
19      - postgresql-devel
20    tags:
21      - packages
22
23    - name: Installing postgresql client
24      command: amazon-linux-extras install postgresql11 -y
25
26    - name: "Creating app_pn user"
27      user:
28        name: "app_pn"
29        comment: "app_pn"
30        uid: "1100"
31        state: present
32        shell: /bin/bash
33        tags: user
34
35    - name: "fixing /home/app_pn/ permissions"
36      file:
37        path: /home/app_pn/
38        state: directory
39        owner: xxxxxx
40        group: xxxxxx
41        mode: +x
42        tags: user
43
44    - name: "Creating /home/app_pn/site/app/ dir"
45      file:
46        path: /home/app_pn/site/app
47        state: directory
48        owner: app_pn
49        group: app_pn
50        tags: user
51
52    - name: "Creating /home/app_pn/site/log for django logs"
53      file:
54        path: /home/app_pn/site/log
55        state: directory
56        owner: xxxxxx
57        group: xxxxxx
58        tags: user
59
60    - name: "Creating /home/app_pn/.well-known"
61      file:
62        path: /home/app_pn/site/.well-known
63        state: directory
64        owner: xxxxxx
65        group: xxxxxx
66        tags: user
67

```

```

68     - name: "Installing virtual host"
69     template:
70     src: templates/linux/apache_app_pn.conf
71     dest: /etc/httpd/conf.d/app_pn.conf
72     tags: apache
73
74     - name: "Installing event mpm conf"
75     template:
76     src: templates/linux/mpm_event.conf
77     dest: /etc/httpd/conf.d/mpm_event.conf
78     tags: apache
79
80     - name: "Installing worker mpm conf"
81     template:
82     src: templates/linux/mpm_worker.conf
83     dest: /etc/httpd/conf.d/mpm_worker.conf
84     tags: apache
85
86     - name: "enabling mpm"
87     template:
88     src: templates/linux/00-mpm.conf
89     dest: /etc/httpd/conf.modules.d/00-mpm.conf
90     tags: apache
91
92     - name: "logrotate"
93     template:
94     src: templates/linux/httpd
95     dest: /etc/logrotate.d/httpd
96     tags: apache
97
98     - name: "Enabling Apache at boot"
99     service: name=httpd enabled=yes
100    tags: apache
101
102    - name: "Systemd: creating celery beat service"
103    template:
104    src: templates/linux/systemd/app-beat.service
105    dest: /etc/systemd/system/app-beat.service
106    mode: 0655
107    owner: root
108    group: root
109    tags: systemd
110
111    - name: "Systemd: creating celery worker service"
112    template:
113    src: templates/linux/systemd/app-worker.service
114    dest: /etc/systemd/system/app-worker.service
115    mode: 0655
116    owner: root
117    group: root
118    tags: systemd
119
120    - name: "Systemd: creating app reports TER"
121    template:
122    src: templates/linux/systemd/app-reports-ter.service
123    dest: /etc/systemd/system/app-reports-ter.service
124    mode: 0655
125    owner: root
126    group: root

```

```

127     tags: systemd
128
129     - name: "Systemd: daemon-reload"
130     systemd:
131     daemon_reload: yes
132     tags: systemd
133
134     - name: "Installing psycopg2"
135     pip:
136     name: psycopg2-binary == 2.8.6
137     tags: pip
138
139     - name: "Installing virtualenv"
140     pip:
141     name: virtualenv
142     executable: pip3.8
143     tags: pip
144
145     - name: "Creating virtualenv"
146     shell: sudo su - app_pn -c "virtualenv --python=/usr/bin/python3.8 ~/virtualenv"
147     tags: pip
148
149     - name: "Installing mod_wsgi"
150     pip:
151     name: mod_wsgi
152     virtualenv: /home/app_pn/virtualenv
153     tags: pip
154
155     - name: "Installing mod_wsgi config"
156     template:
157     src: "templates/linux/{{mod_wsgi_conf}}"
158     dest: /etc/httpd/conf.modules.d/mod_wsgi.conf
159     tags: apache
160
161     - name: "Creating /home/app_pn/site/app/app (...)"
162     file:
163     path: /home/app_pn/site/app/app
164     state: directory
165     owner: xxxxxx
166     group: xxxxxx
167     mode: 0755
168     tags: pip
169
170     - include_tasks: deploy_linux.yml
171     tags: deploy

```

11.2.5 Role: postgres.yml

Questo ruolo si occupa della configurazione del database e viene chiamato durante l'esecuzione del file `deploy_linux.yml`. Le operazioni che vengono effettuate sono:

- Crea il database
- Crea l'utente
- Attiva il virtual enviromental e genera i file di migrazione
- Attiva il virtual enviromental ed esegue i file di migrazione generati

- Sposta il file di set minimo nella cartella di destinazione
- Esegue il file di set minimo per popolare il database

```

1  - name: Create the database specified in vars
2  postgresql_db:
3  db: "{{ app_postgres_db_name }}"
4  login_host: "{{ app_postgres_master }}"
5  login_user: "{{ app_postgres_user }}"
6  login_password: "{{ app_postgres_password }}"
7  state: present
8
9  - name: Create app_plugin user
10 postgresql_user:
11 db: "{{ app_postgres_db_name }}"
12 login_host: "{{ app_postgres_master }}"
13 login_user: "{{ app_postgres_user }}"
14 login_password: "{{ app_postgres_password }}"
15 name: app_plugin
16 priv: ALL
17 state: present
18
19 - name: "Updating database"
20 shell: . /home/app_pn/virtualenv/bin/activate && /home/app_pn/site/app/manage.py
      makemigrations
21 tags: schema
22
23 - name: "Updating database schema"
24 shell: . /home/app_pn/virtualenv/bin/activate && /home/app_pn/site/app/manage.py
      migrate
25 tags: schema
26
27 - name: "updating setMinimo.sql"
28 template:
29 backup: yes
30 src: ../templates/linux/setMinimo.sql
31 dest: /home/app_pn/setMinimo.sql
32 owner: xxxxxx
33 group: xxxxxx
34 tags: setMinimo
35
36 - name: Run queries from SQL script
37 postgresql_query:
38 db: "{{ app_postgres_db_name }}"
39 login_host: "{{ app_postgres_master }}"
40 login_user: "{{ app_postgres_user }}"
41 login_password: "{{ app_postgres_password }}"
42 path_to_script: /home/app_pn/setMinimo.sql

```

11.3 Creazione della pipeline Jenkins

Il flusso CI/CD viene implementato attraverso una pipeline dichiarativa Jenkins che gestisce il software sui vari ambienti che sono stati predisposti per realizzare questo scenario di test. Su ognuno di essi si trovano in esecuzione i container Docker, l'ambiente di sviluppo é una macchina remota Linux mentre i server di collaudo e produzione vengono simulati sulla macchina locale. La pipeline viene lanciata manualmente attraverso l'interfaccia grafica di Jenkins, quando questo avviene é previsto l'inserimento di un parametro che per ogni ambiente indica se occorre

eseguire il setup dell'applicazione o meno. A seconda del parametro viene quindi eseguito il giusto playbook di Ansible. Il primo stage configura e/o rilascia quindi l'applicazione sul server di sviluppo, per poter raggiungere la macchina remota che è un server aziendale è necessario eseguire il seguente tunnel ssh: `ssh -L localhost:2377:localhost:2375 remote_ip` il quale rende disponibile la connessione con Docker sulla socket localhost:2377 della macchina locale, tale socket viene quindi passata come parametro ai playbook Ansible dello stage attraverso una variabile di ambiente. Successivamente all'esecuzione dei playbook si entra nello stage adibito al testing delle API sull'ambiente di sviluppo. Si utilizza quindi Newman per eseguire le richieste e i test presenti nella collezione Postman insieme alle variabili di ambiente definite. Se tutto è andato a buon fine l'esecuzione procede nello stage di collaudo dove vengono nuovamente eseguiti i playbook Ansible. Nell'ambiente di collaudo si ipotizza che dopo aver distribuito il software debbano essere eseguiti dei test di tipo applicativo, dopo aver deployato in questo ambiente si entra quindi in uno step di approvazione che blocca l'esecuzione della pipeline in attesa di una conferma prima di procedere con il rilascio sul server di produzione. Se quindi l'approvazione non avviene il rilascio viene interrotto, altrimenti con un flag di ok si procede con la consegna sul server di produzione e la pipeline termina correttamente la sua esecuzione.

```
1 pipeline {
2   agent any
3
4   parameters {
5     string(name: 'release_type_dev')
6     string(name: 'release_type_stg')
7     string(name: 'release_type_prod')
8   }
9
10  stages {
11    stage('Release in develop environment') {
12      steps {
13        script {
14          if(params.release_type_dev == 'site')
15            sh 'env "DOCKER_HOST=localhost:2377" ansible-playbook -i
              /home/daniele/Scrivania/progetto_sd/ansible/inventories/dev/inventory-dev-docker.yml
              /home/daniele/Scrivania/progetto_sd/ansible/playbooks/apppush/site.yml
              -v'
16
17          else if(params.release_type_dev == 'deploy')
18            sh 'env "DOCKER_HOST=localhost:2377" ansible-playbook -i
              /home/daniele/Scrivania/progetto_sd/ansible/inventories/dev/inventory-dev-docker.yml
              /home/daniele/Scrivania/progetto_sd/ansible/playbooks/apppush/deploy.yml
              -v'
19        }
20      }
21    }
22    stage('Test APIs') {
23      steps {
24        sh 'newman run
          /home/daniele/Scrivania/jenkins/app_pn.postman_collection.json -g
          /home/daniele/Scrivania/jenkins/variables.postman_environment.json'
25      }
26    }
27    stage('Release in stage environment') {
28      steps {
29        script {
30          if(params.release_type_stg == 'site')
31            sh 'ansible-playbook -i
              /home/daniele/Scrivania/progetto_sd/ansible/inventories/stage/inventory-stage-docke
              /home/daniele/Scrivania/progetto_sd/ansible/playbooks/apppush/site.yml
```

```

32         -v'
33     else if(params.release_type_stg == 'deploy')
34     sh 'ansible-playbook -i
        /home/daniele/Scrivania/progetto_sd/ansible/inventories/stage/inventory-stage-docker.
        /home/daniele/Scrivania/progetto_sd/ansible/playbooks/apppush/deploy.yml
        -v'
35     }
36 }
37 }
38 stage('Waiting for approval to proceed') {
39     steps {
40         script {
41             try {
42                 userInput = input(
43                     id: 'proceed', message: 'Pending approval', parameters: [
44                         [$class: 'BooleanParameterDefinition', defaultValue: true,
45                             description: '', name: 'Release into production?']
46                     ])
47             } catch(err) {
48                 def user = err.getCauses()[0].getUser()
49                 userInput = false
50                 echo "Aborted by: [${user}]"
51             }
52         }
53     }
54     stage('Release in production environment') {
55         steps {
56             script {
57                 if(params.release_type_prod == 'site')
58                 sh 'ansible-playbook -i
                    /home/daniele/Scrivania/progetto_sd/ansible/inventories/prod/inventory-prod-docker.
                    /home/daniele/Scrivania/progetto_sd/ansible/playbooks/apppush/site.yml
                    -v'
59
60                 else if(params.release_type_prod == 'deploy')
61                 sh 'ansible-playbook -i
                    /home/daniele/Scrivania/progetto_sd/ansible/inventories/prod/inventory-prod-docker.
                    /home/daniele/Scrivania/progetto_sd/ansible/playbooks/apppush/deploy.yml
                    -v'
62             }
63         }
64     }
65 }
66 }

```

11.4 Funzionamento del sistema

Gli sviluppatori che vogliono rilasciare una nuova funzionalità portano le modifiche sul branch usato dai playbook ansible per recuperare l'applicazione dal repository durante la distribuzione. Per lanciare Jenkins é necessario che gli sviluppatori si colleghino alla macchina su cui esso é in esecuzione, la build puó essere eseguita attraverso l'interfaccia grafica oppure dalla riga di comando. Questa fase si potrebbe anche evitare in quanto é possibile configurare la pipeline per essere lanciata in esecuzione in seguito al ricevimento di una merge o di un commit. Durante l'esecuzione della pipeline il nodo orchestratore si collega ai vari ambienti dell'applicazione usando Ansible dove su ogni macchina esegue i comandi necessari a installare, configurare e distribuire

il software che viene recuperato dal repository. Su ogni ambiente dopo l'esecuzione dei playbook il nodo orchestratore invoca in sequenza tutte le chiamate della collection creata usando Newman. Ogni chiamata deve rispondere correttamente e superare i test, se questo non avviene l'esecuzione della pipeline si interrompe. L'esito delle varie esecuzioni rimane memorizzato in Jenkins ed è sempre consultabile. L'esecuzione può interrompersi anche a causa degli errori che vengono generati in seguito all'esecuzione dei comandi impartiti con i playbook. La pipeline è parametrizzata con delle variabili per poter distinguere quale tipo di playbook usare tra i vari ambienti. Prima del rilascio nell'ultimo ambiente che rappresenta l'ambiente di produzione la pipeline mette in pausa la sua esecuzione e chiede una conferma che si contrassegna attraverso una checkbox prima di procedere con il rilascio. Poiché il software viene distribuito su dei container è necessario che questi siano stati precedentemente creati e che stiano funzionando correttamente. Per poter distribuire l'applicazione per la prima volta dopo la creazione di un container è necessario eseguire il playbook responsabile dell'installazione dell'app.

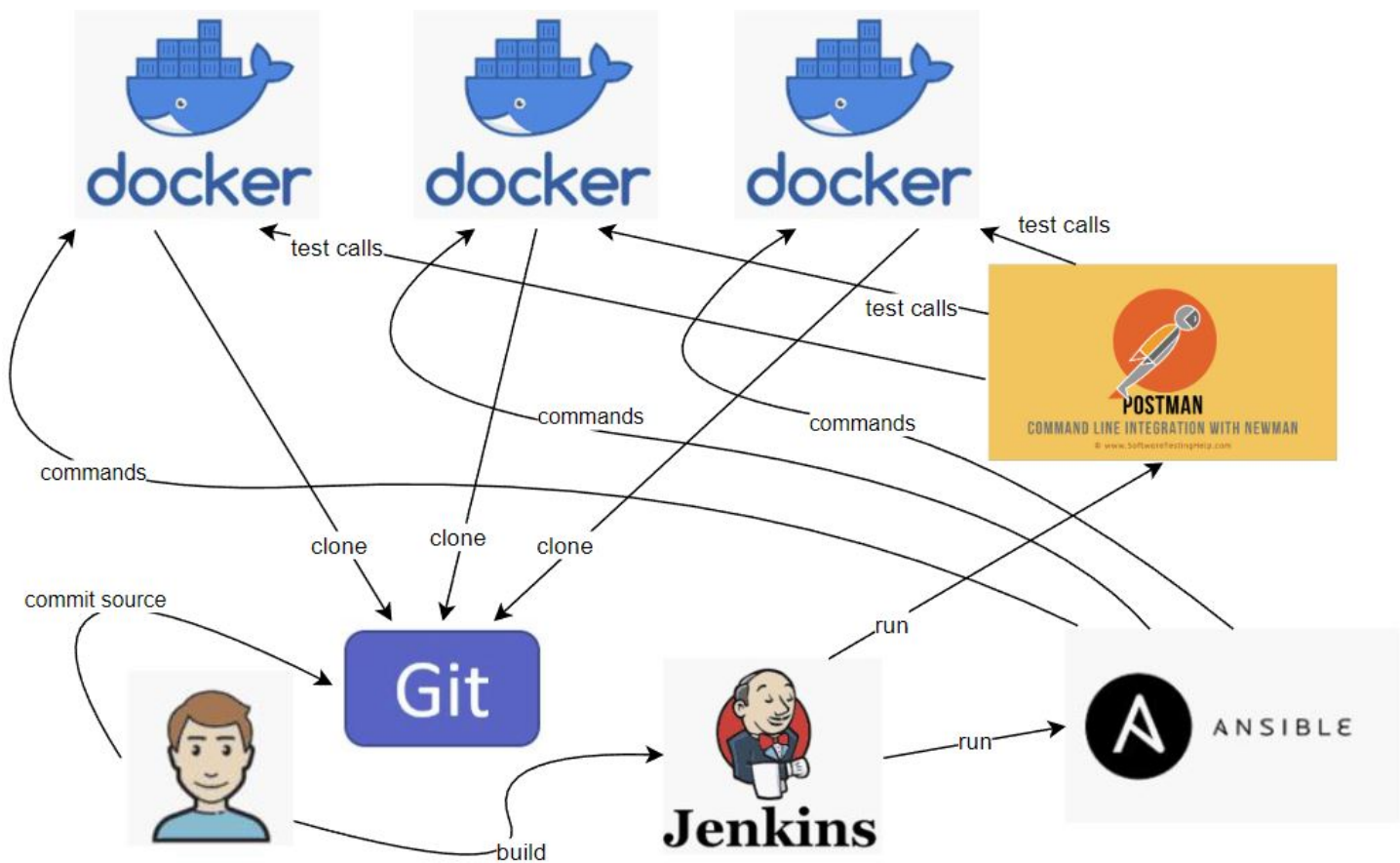


Figura 9: Diagramma a blocchi del sistema

11.5 Test del sistema

Per verificare la corretta esecuzione del sistema realizzato ci si può basare innanzi tutto sull'interfaccia grafica di Jenkins, attraverso la quale è possibile controllare l'esecuzione dei playbook, dei test con Newman e anche monitorare la corretta esecuzione della pipeline durante i vari stage.

Per verificare che i file siano stati correttamente copiati ci si può collegare ai container e verificarne l'effettivo aggiornamento all'interno delle relative directory. Si possono anche eseguire dei test applicativi al fine di verificare che la versione del software deployata corrisponda con quello che ci si aspetta. Per farlo occorre collegarsi alla piattaforma di Oracle, creare un'istanza del plugin e testare manualmente le varie funzionalità dell'applicazione. Per controllare la corretta

	executed	failed
iterations	1	0
requests	19	0
test-scripts	36	0
prerequisite-scripts	19	0
assertions	23	0
total run duration: 21.5s		
total data received: 248KB (approx)		
average response time: 1094ms [min: 20ms, max: 5s, s.d.: 1128ms]		

Figura 10: Output di Newman

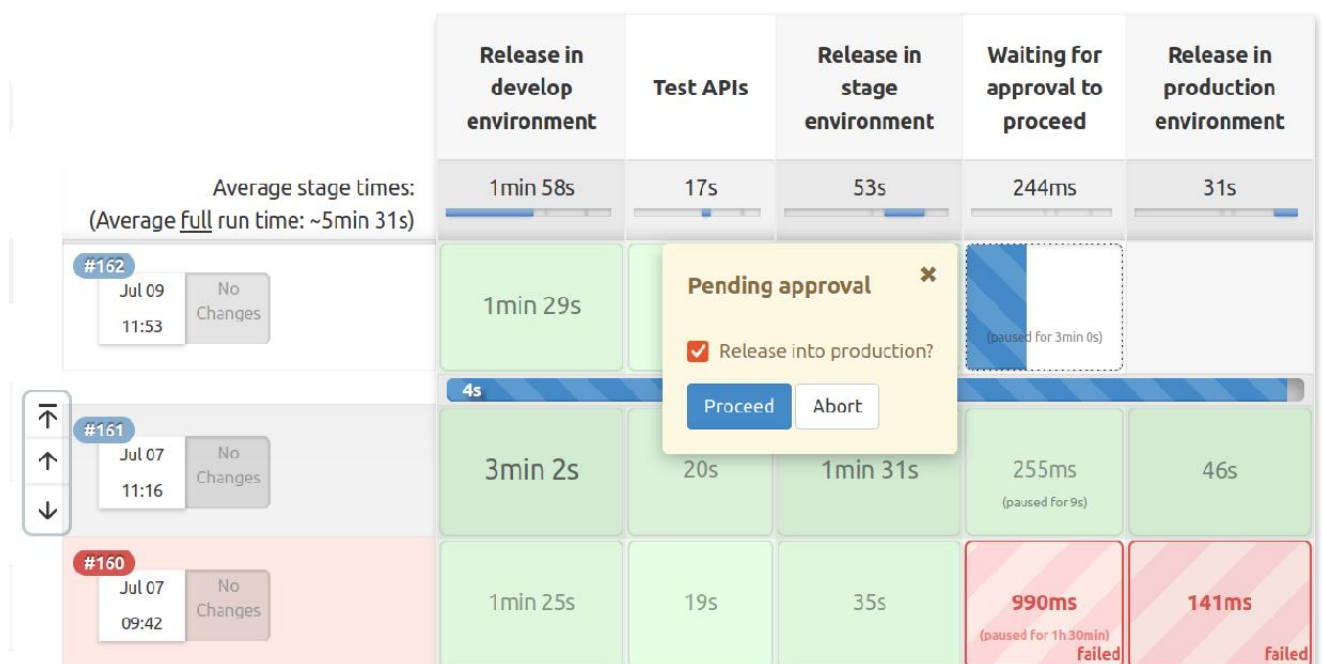


Figura 11: Pipeline Jenkins in esecuzione

esecuzione dell'app si verifica la presenza dei dati sul database, nei registri di log e anche la presenza dei messaggi sulle code del message broker RabbitMQ.

11.5.1 Limiti delle tecnologie impiegate e del sistema

In questa sezione vengono individuati e analizzati diversi casi di test, con l'obiettivo di far emergere le criticità del sistema realizzato.

1. il nodo che esegue il playbook Ansible viene improvvisamente spento durante il deploy:
Ansible non ha un modo automatico per annullare le proprie azioni ed eseguire un rollback

		Release in develop environment	Test APIs	Release in stage environment	Waiting for approval to proceed	Release in production environment
Average stage times: (Average <u>full</u> run time: ~5min 40s)		1min 58s	17s	53s	250ms	32s
#162	Jul 09 11:53 No Changes	1min 29s	23s	32s	164ms (paused for 3min 18s)	38s
#161	Jul 07 11:16 No Changes	3min 2s	20s	1min 31s	255ms (paused for 9s)	46s
#160	Jul 07 09:42 No Changes	1min 25s	19s	35s	990ms (paused for 1h 30min) failed	141ms failed

Figura 12: Esecuzione della pipeline completata

- in caso di errore, e nemmeno Jenkins. In questo caso i nodi che vengono gestiti potrebbero essere lasciati in uno stato di inconsistenza. Ansible offre alcune funzionalità per gestire personalmente gli errori come blocchi di azioni da implementare nelle sezioni di rescue, tuttavia se il sistema si guasta in qualche modo insolito anche le attività previste in questo tipo di sezioni potrebbero non funzionare
- perdita di connessione verso il server che ospita un container durante il deploy: in questo caso le attività previste dai playbook non vengono più eseguite e la pipeline termina l'esecuzione. Questo può portare a casi in cui un errore non correlato può lasciare un host in uno stato imprevisto. In questo caso occorre attendere per il ripristino della connessione ed eseguire nuovamente la build con Jenkins
 - mentre si sta eseguendo il deploy un membro del team esegue la push di un commit sul branch che viene utilizzato dai playbook per clonare il progetto: in questo caso non vi è alcuna garanzia che il commit che doveva avviare la build sarà ancora in head quando Jenkins avrà terminato la sua esecuzione. Per risolvere questo problema si potrebbe pensare di non usare direttamente il branch su cui vengono committate le modifiche per deployare sui vari ambienti. Si potrebbe piuttosto predisporre il deploy per usare un branch diverso in ogni ambiente, automatizzare il processo di merge tra le branch ed impedirlo nuovamente fino a che la build Jenkins non è stata completata
 - il software viene rilasciato in ambiente di produzione mentre un cliente sta eseguendo una campagna per l'invio di notifiche push: in questo caso viene creato un malfunzionamento perché l'applicazione viene riavviata, questo comporta lo spegnimento delle connessioni (con le code Rabbit per esempio), l'incapacità di rispondere alle chiamate HTTP e l'interruzione delle operazioni perché i processi che eseguono i task vengono riavviati. In questo modo si genera un disservizio per il cliente. In particolare si vanno a perdere le chiamate ricevute dalla piattaforma di Oracle e si rischia di non accodare più correttamente i messaggi sulle code del message broker. In questo caso per mitigare il problema risulta indispensabile una buona comunicazione tra il team di sviluppo e il team di customer care che segue le attività del cliente

12 Conclusioni

In questo progetto sono state studiate e messe in pratica diverse tecnologie che sono Postman, Newman, Docker, Ansible e Jenkins.

Alcune di queste sono state messe a confronto e sono stati evidenziati i relativi vantaggi e svantaggi dati dal loro utilizzo. La scelta delle tecnologie é stata guidata dal fatto che queste vengono utilizzate nell'azienda in cui lavorano i membri del team e si aveva quindi l'interesse nel migliorare il proprio livello di conoscenza a riguardo.

É stato creato un sistema di CI/CD che ha permesso di mettere in pratica quello che si é imparato e sono stati pensati diversi casi d'uso con l'obiettivo di individuare le criticità che possono emergere durante l'utilizzo di questo sistema.

Inizialmente si era pensato di poter trarre beneficio dalla creazione di questa pipeline per un progetto lavorativo, sono poi però emerse delle limitazioni durante lo sviluppo dovute a problemi di riservatezza nei confronti della compagnia che ha imposto dei limiti al fine di preservare la propria privacy. Sono anche sorti dei problemi a livello di permessi che hanno impedito la possibilità di configurare degli hook tra la pipeline e il repository aziendale utilizzato, in maniera tale da poter avviare la build di Jenkins automaticamente in seguito a una merge o all'invio di un commit; di conseguenza non é stato possibile soddisfare uno dei principi previsti dalle pratiche per la CI. Dallo scenario di test progettato é emerso che le tecnologie impiegate a prescindere da quali esse siano non sono e non possono essere la soluzione di tutti i problemi presenti nel mondo della CI/CD, possono però essere di grande aiuto nel supportare questo genere di attività. La coordinazione tra i membri dei team che seguono i progetti all'interno di un'azienda risulta come sempre un elemento fondamentale del quale non si può fare a meno affinché tutto funzioni correttamente.

12.1 Sviluppi futuri

Un punto interessante che sarebbe bello affrontare in futuro potrebbe essere automatizzare un numero maggiore di attività sul repository remoto attraverso Ansible. Ad esempio automatizzando il processo di merge tra le branch dei vari ambienti, si potrebbe sicuramente realizzare una pipeline più solida e sofisticata. Un altro scenario interessante potrebbe prevedere di includere nei playbook Ansible anche la parte relativa alla gestione dei container, automatizzando magari il trasferimento dei Dockerfile dal repository al server, la creazione delle immagini e l'avvio dei container.

Riferimenti bibliografici

- [1] Red Hat. What is ci/cd? <https://www.redhat.com/it/topics/devops/what-is-ci-cd>.
- [2] wikipedia.org. Integrazione continua. https://it.wikipedia.org/wiki/Integrazione_continua#Principi.
- [3] harness.io. continuous-delivery-vs-continuous-deployment. <https://harness.io/blog/continuous-delivery/continuous-delivery-vs-continuous-deployment>.
- [4] html.it. Docker e i container. <https://www.html.it/pag/62783/docker-e-i-container>.
- [5] dev.to. Understanding the dockerfile. <https://dev.to/aws-builders/understanding-the-dockerfile-format-3cc6>.
- [6] docs.docker.com. Panoramica di docker compose. <https://docs.docker.com/compose>.
- [7] opensource.com. What is ansible. <https://opensource.com/resources/what-ansible>.
- [8] redhat.com. Learning ansible. <https://www.redhat.com/en/topics/automation/learning-ansible-tutorial>.

- [9] whizlabs.com. Ansible advantages and disadvantages. <https://www.whizlabs.com/blog/ansible-advantages-and-disadvantages>.
- [10] infoworld.com. What is jenkins? the ci server explained. <https://www.infoworld.com/article/3239666/what-is-jenkins-the-ci-server-explained.html>.
- [11] jenkins.io. What is a jenkins pipeline? https://www.jenkins.io/doc/pipeline/tour/hello-world/#_footnotedef_1.
- [12] tutorialandexample. Advantages and disadvantages of jenkins. <https://www.tutorialandexample.com/advantages-and-disadvantages-of-jenkins>.