Università di Camerino

Università degli Studi di Camerino

SCUOLA DI SCIENZE E TECNOLOGIE
Corso di Laurea in Informatica (Classe L-31)

Salesforce DevOps: Metodologie a confronto

Laureando
Lorenzo Evangelisti
Matricola 113910

Relatore
Fausto Marcantoni

Correlatori
Antonio Tangaro
Enrico Murru

A.A. 2021/2022

Indice

CAPITOLO 1

Introduzione

In collaborazione con **Engineering**, compagnia per cui lavoro da settembre 2021, nasce la volontà di approfondire i temi che riguardano DevOps applicato alla piattaforma Salesforce, al fine di determinare quali siano le tecnologie DevOps verso cui indirizzare i nostri clienti in fase di trasformazione tecnologica.
Engineering si occupa appunto di affiancare le imprese di ogni settore nei processi di trasformazione tecnologica, tema che comprende tante sfaccettature: tra queste c'è l'adozione e il mantenimento di una piattaforma personalizzata sviluppata su ambiente Salesforce.
Le imprese si approcciano a Salesforce per il bisogno di rendere più sottile il gap che li separa dai propri clienti, il che significa che per essere il più efficace possibile, l'ambiente Salesforce su cui andranno a lavorare deve essere architettato in maniera tale da presentarsi sia con una semplicità lampante, che con un'attenzione al dettaglio meticolosa.
Questa sfida viene superata con successo non solo grazie alla piattaforma Salesforce, ma anche grazie a vari tool di terze parti disponibili sullo store AppExchange, oltre che ad aggiornamenti della piattaforma attrverso app native.
In particolare, per quanto riguarda il flusso DevOps, esistono svariate tecnologie che permettono la gestione di esso, quali puntando più su un aspetto, quali su un altro.
Per questo, partendo da una dettagliata descrizione dei punti focali che girano attorno al mondo DevOps, passando per una presentazione della piattaforma Salesforce e di come DevOps viene applicato al flusso di lavoro su di essa, verranno valutate le tecnologie già in forte presenza nell'ambiente in rapporto ad una tecnologia emergente in particolare, pubblicata da Salesforce stesso: DevOps Center.
Sarà quindi possibile rispondere a questa domanda: "In base alle risorse di un'impresa che ha bisogno di adottare una tecnologia di DevOps, quale tecnologia ha il maggior margine di successo?"

CAPITOLO 2

DevOps

Da **Mastering Salesforce DevOps** di Andrew Davies:
In practice, DevOps is not something you're ever "done with". In that sense it's like "collaboration" or "efficiency" - a principle to adopt and a goal to always improve upon.

Cos'è DevOps?

DevOps è un termine che indica un gruppo di concetti che, pur non essendo tutti nuovi, si sono catalizzati in un movimento e si stanno rapidamente diffondendo nella comunità tecnica. Come ogni termine nuovo e popolare, le persone hanno impressioni un po' confuse e talvolta contraddittorie su cosa sia. Sebbene molte discussioni su DevOps si concentrino sugli strumenti specifici o sulle implementazioni tecniche, il significato principale riguarda le persone che sviluppano e mantengono i sistemi tecnici e la cultura che circonda questo processo. Un ciclo infinito, come mostrato nella **Figura 1**, viene spesso utilizzato per illustrare il ritmo continuo delle attività che compongono il processo DevOps.

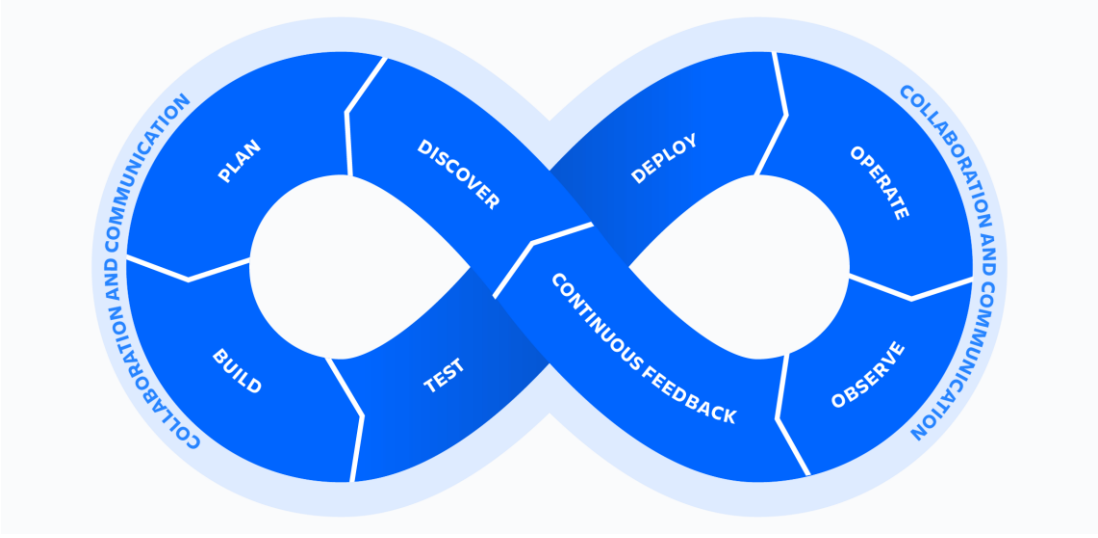

Figura 1. *Un ciclo infinito di DevOps*

Dev vs. Ops

La pratica di DevOps consiste nel considerare i processi informatici come centrali per la capacità di un'impresa di fornire valore ai propri clienti e di migliorare continuamente il processo di fornitura di nuove funzionalità, garantendo al contempo qualità e stabilità.
DevOps è l'unione dei termini "Dev" (Development) e "Ops" (Operations). Un'esigenza fondamentale per le aziende è la **capacità di innovare e sviluppare nuove funzionalità** per i propri clienti e dipendenti. Questo è il ruolo del team "Dev".
Un'altra esigenza critica è che **i sistemi esistenti siano stabili, affidabili e sicuri**. Questo è il ruolo del team "Ops", che di solito è composto da amministratori di sistema, amministratori di database, amministratori di siti web e così via. Il loro compito principale è assicurarsi che i server siano attivi e funzionanti, che il Service Level Agreement sia rispettato, che l'applicazione funzioni come previsto e così via.

Il termine DevOps implica la collaborazione tra questi due team, motivati dall'obiettivo comune di migliorare sia l'innovazione che la gestione delle risorse. Il raggiungimento di questo obiettivo comporta l'integrazione di pratiche di sviluppo di lunga durata, come la Continuous Integraton (CI), la Continuous Delivery (CD), i test automatizzati e l'infrastruttura come codice. Per questo motivo, è diventato un termine generico per i miglioramenti dei processi di sviluppo sia nuovi che vecchi.

Esiste una tensione naturale tra l'esigenza di innovazione (cambiamento) e quella di stabilità. Gli sviluppatori vogliono e devono continuare a cambiare il sistema, mentre l'Ops team vuole e ha bisogno che il sistema rimanga il più statico possibile, in modo da ottimizzare le prestazioni e ridurre i rischi che il cambiamento comporta. Nelle grandi organizzazioni, questi team lavoravano storicamente in silos che isolavano i team di sviluppo da quelli di Quality Assurance (QA) e Ops. QA e Ops in genere si occupano di un numero enorme di applicazioni e funzionalità, a volte con scarsa comprensione dello scopo e del valore aziendale del software che stanno abilitando. L'obiettivo finale per tutti questi team è la soddisfazione del cliente, ma gli obiettivi specifici dei "Devs" includono la correzione rapida dei bug e la creazione di nuove funzionalità, mentre per

le loro controparti "Ops" l'obiettivo potrebbe essere quello di mantenere il 99,99% di uptime dei server.
Questi obiettivi possono spesso essere in conflitto l'uno con l'altro, portando a inefficienza e a puntare il dito quando le cose vanno male. Il conflitto cronico tra i team Dev e IT è una ricetta per il fallimento dei team IT e dell'organizzazione che servono.
Il concetto di DevOps si basa sulla costruzione di una cultura della collaborazione, della comunicazione e dell'automazione tra team che storicamente hanno funzionato in silos. L'obiettivo è che sia gli sviluppatori che le operazioni condividano la responsabilità di facilitare l'innovazione, pur garantendo la stabilità. La cultura generativa promossa dalla comunità DevOps enfatizza valori come la proprietà e la responsabilità. I team di sviluppo e operativi collaborano strettamente, condividono molte responsabilità e combinano i loro flussi di lavoro. In questo modo si riducono le inefficienze e si risparmia tempo (ad esempio, scrivendo codice che tenga conto dell'ambiente in cui viene eseguito).
La **Figura 2** illustra DevOps come l'unificazione dell'attenzione e delle attività di questi team precedentemente eterogenei.

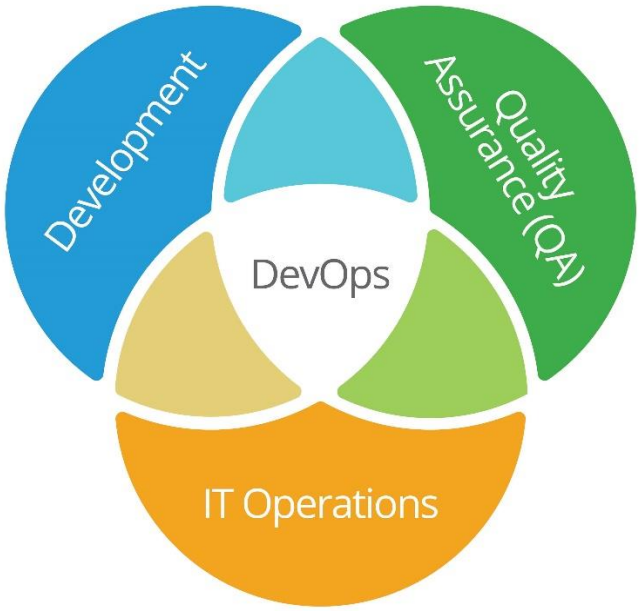

Figura 2. *DevOps riunisce le attività che tradizionalmente venivano svolte da team indipendenti di sviluppo, QA e operazioni IT.*

Le tre vie di DevOps

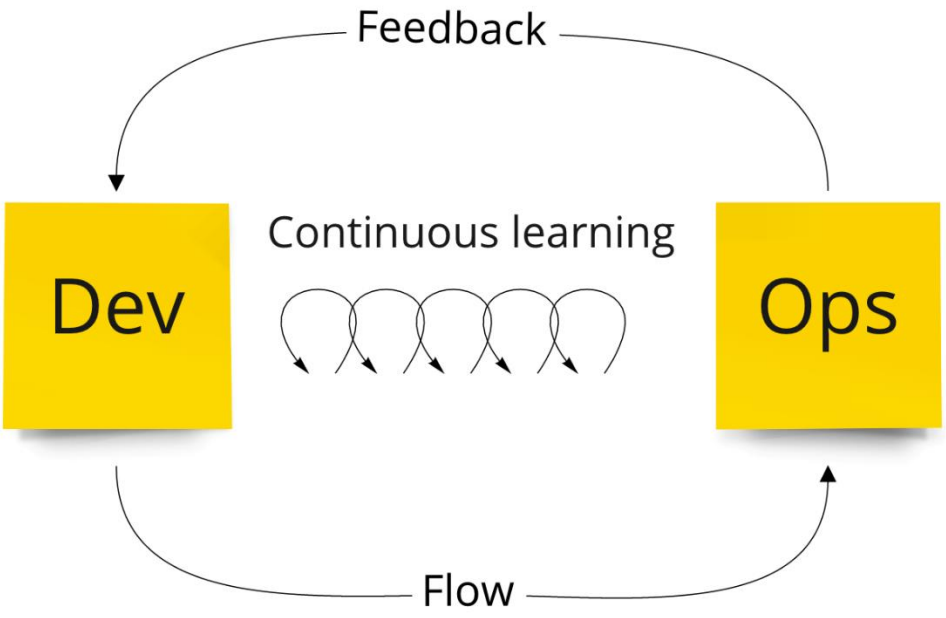

Figura 3. *Le tre vie di DevOps – Flow (o Continuous Delivery), Continuous Feedback e Continuous Learning (o Continuous Improvement).*

Flow (o CD - Continuous Delivery)

La prima via è il flusso di lavoro da sinistra a destra, dallo sviluppo alla QA, alle operazioni IT e al cliente. Per capire il processo di Continuous Delivery, è utile comprendere alcuni concetti di base coinvolti in questo processo.
Il fondamento della Continuous Delivery (e quindi dell'implementazione di DevOps) è l'uso del Version Control. Il Version Control è un meccanismo per tracciare e unire modifiche ai metadati senza problemi. Esistono molti tipi di Version Control ma quello più comunemente usato dagli sviluppatori oggi è Git. L'uso del Version Control offre molti vantaggi, ma apre anche le porte a tutte le pratiche menzionate in seguito. Il motivo è che il codice memorizzato nel Version Control può essere consultato da script automatici, e ogni commit tracciato nel Version Control riflette una modifica candidata ai test e alla distribuzione.

Continuous Integration (CI) significa che i team lavorano su un ramo comune nella base di codice ed eseguono test automatizzati con ogni commit in quel ramo. Il termine "CI" è spesso usato per riferirsi a strumenti come Jenkins, che eseguono azioni automatizzate in base a una pianificazione o alle modifiche del codice.
Ma il significato di fondo di CI è che non ci sono mai due sviluppatori che lavorano su basi di codice che si discostano in modo significativo l'una dall'altra. La CI è una pratica di sviluppo software collaudata per decenni, in contrasto con un approccio in cui i team sviluppano in modo isolato per settimane, mesi o anni prima di entrare in una lunga e dolorosa fase di integrazione.
Quando si utilizza la CI, è tipico che tutto il codice prodotto dai team di sviluppo confluisca in una repository centrale, dove una build automatica lo convalida. Questa pratica aiuta i team di sviluppo a rilevare, identificare e risolvere problemi e bug prima di rilasciare le modifiche ai clienti. Questo "sistema CI" avverte il team in caso di errori di compilazione o di test, e i controlli vengono ripetuti anche per le modifiche più piccole, per garantire che il sistema continui a funzionare in modo impeccabile.
L'individuazione precoce e le correzioni sono fondamentali per garantire la qualità nella fase più prematura possibile. Gli sviluppatori dovrebbero concentrarsi sulla creazione di un semplice processo di Continuous Integration il più presto possibile nel ciclo di vita dello sviluppo. Continuous Delivery significa che tutta la configurazione necessaria per ricreare l'applicazione viene memorizzata e distribuita dal Version Control in modo automatizzato. Si basa sulle due pratiche precedenti (Version Control e Continuous Integration) ed è la principale capacità tecnica alla base di DevOps.
DevOps combina la Continuous Delivery con una gestione snella per massimizzare il flusso di software di valore e la stabilità dei sistemi allo stesso tempo.

Continuous Feedback

La seconda via è il ciclo di feedback che va da destra a sinistra; il flusso costante di feedback in tutte le fasi del flusso di valore per garantire che si possa evitare che i problemi si ripetano o consentire un rilevamento e un

recupero più rapidi. I processi necessari includono la creazione di suite di test automatizzati e veloci, il monitoraggio della sicurezza, della qualità e dell'affidabilità del lavoro che passa attraverso le pipeline, assicurando che il codice sia sempre in uno stato distribuibile.

Continuous Improvement (o Continuous Learning)

I sistemi di Continuous Delivery e Feedback descritti in precedenza sono chiamati "sistema di lavoro", ovvero il meccanismo che consente al lavoro di valore di fluire verso l'ambiente di produzione. In qualsiasi sistema in evoluzione, l'entropia fa sì che le cose si rompano con il passare del tempo. Solo applicando gli sforzi per migliorare continuamente il "sistema di lavoro" si può continuare a sostenere e potenziare l'intera organizzazione. DevOps richiede una sperimentazione continua per migliorare il "sistema di lavoro" e per dare priorità a questo sistema rispetto al lavoro stesso. La maturazione della pipeline DevOps implica l'assunzione di rischi e l'apprendimento da successi e fallimenti. Le pratiche necessarie includono la creazione di una cultura dell'innovazione, dell'assunzione di rischi e della fiducia.

Lean Management

DevOps combina fondamentalmente due movimenti: le pratiche tecniche di Continuous Delivery e le pratiche di Lean Management e Lean Product Development che hanno avuto origine con la Toyota. Il sistema di produzione Toyota ha stabilito lo standard mondiale per la produzione di massa di un prodotto di alta qualità. E i loro processi di sviluppo dei prodotti hanno stabilito lo standard mondiale per la rapidità di innovazione per soddisfare le svariate esigenze dei clienti. Il Lean Software Development è nato dalla consapevolezza che l'applicazione delle stesse pratiche allo sviluppo del software aumentava la qualità e, di conseguenza, le prestazioni dell'azienda.

I pilastri dell'approccio Lean (che si tratti di produzione o di software) sono due: just-in-time e stop-the-line.
Il **just-in-time** si concentra su velocità, produttività ed efficienza. Nel caso del software, l'idea è quella di ridurre al minimo il tempo necessario per consegnare una funzionalità richiesta attraverso fasi come l'automazione dei deploy e l'identificazione di altri colli di bottiglia nel processo che non aggiungono valore. Una chiave per una consegna rapida è quella di suddividere il lavoro in piccoli batch. Per questo motivo, la frequenza dei rilasci è un ottimo indicatore del fatto che i team stiano seguendo questo approccio.

Stop-the-line significa che non appena si riscontra un difetto o un'anomalia, la linea di produzione (o la pipeline di distribuzione) si ferma e la priorità assoluta diventa l'identificazione e l'eliminazione della fonte del problema per garantire che non si ripeta. L'attenzione è rivolta alla stabilità e alla qualità. Nel caso del software, questo si ottiene inserendo dei punti di controllo, come i test e le convalide automatiche, che vengono eseguiti ogni volta che la base del codice viene modificata.

Il tempo che un cliente trascorre in attesa della consegna di una funzionalità si chiama **Lead Time**. Cosa succede mentre il cliente aspetta? Tre cose:

- Viene svolto un lavoro che aggiunge valore.
- Viene svolto un lavoro che (a posteriori) non aggiunge valore (e a volte non è nemmeno necessario).
- La feature è in attesa che qualcun altro la costruisca, la riveda, la testi e/o la distribuisca.

A volte è difficile dire quali azioni aggiungono valore e quali no. Ma è sempre vero che il tempo trascorso in attesa non aggiunge alcun valore. La velocità sostenibile più elevata alla quale tutto il lavoro di valore può essere svolto si chiama **Process Time**. La differenza tra il Lead Time e il Process Time è uno dei principali tipi di spreco.
Se si pensa al ciclo di vita dei generi alimentari, questi vengono prima coltivati o prodotti, poi immagazzinati o spediti, infine acquistati e consumati. Più tempo passa tra la coltivazione e il consumo, più è probabile che vadano a male.

Inoltre, il software invecchia rapidamente. E soprattutto, più lungo è il Lead Time, più lungo è il tempo necessario per ottenere un feedback dagli utenti. La costruzione di un software è estremamente complessa e l'unico modo affidabile per garantire che le cose siano costruite bene è quello di ottenere un feedback rapido e ripetuto.
L'essenza di DevOps (e del Lean Software Development) è eliminare gli sprechi, in particolare il tempo di attesa. La riduzione dei tempi di attesa consente un feedback rapido, un feedback rapido consente l'innovazione e l'innovazione consente il successo.
Uno dei modi più semplici per eliminare le perdite di tempo è automatizzare le distribuzioni utilizzando la Continuous Delivery.

Misurazione delle performance

Una metrica fondamentale utilizzata per determinare il successo dell'implementazione di DevOps è il "Lead Time": quanto tempo impiega un'organizzazione per distribuire una singola riga di codice in produzione? È importante che tutte le modifiche apportate in produzione siano ben testate e realizzate in modo controllato. Ma maggiori ritardi nella consegna delle funzionalità in produzione significano maggiori ritardi nell'ottenere un feedback.
Un feedback ritardato significa che i miglioramenti e le correzioni dei bug vengono ritardati, causando un cambio di contesto inefficiente in quanto gli sviluppatori correggono ripetutamente lavoro che potrebbero non aver toccato per giorni, settimane o addirittura mesi.

È stato dimostrato che l'uso di DevOps (Continuous Delivery e Lean Software Development) migliora la Software Delivery Performance (la velocità con cui è possibile rilasciare funzionalità in modo stabile).

Secondo lo *State of DevOps Report* del 2018 di Puppet by Perforce, che introduce per la prima volta nel dettaglio il "DevOps Maturity Model":
La Software Delivery Performance e l'Operational Performance consentono alle imprese di sfruttare il software per ottenere risultati migliori. Questi risultati sono misurati attraverso molti fattori, tra cui la produttività, la redditività e la quota di mercato, nonché da fattori non commerciali come l'efficacia, l'efficienza e la soddisfazione dei clienti. La nostra analisi mostra che gli "Elite Preformers" hanno 1,53 volte più probabilità di raggiungere o superare gli obiettivi di performance organizzativa.

Inoltre, lo *State of DevOps Report* del 2018 mostra che i team con una Software Delivery Performance (SDP) elevata, superano quelli con una bassa SDP con:

1. Rilasci del codice 46 volte più frequenti
2. Lead Time 2.555 volte più rapido dal commit al deploy
3. Change failure rate 7 volte più basso (1/7 di probabilità che una modifica fallisca)
4. Tempo di recupero da "downtime" 2.604 volte più veloce

Uno dei principali contributi dello *State of DevOps Report* è stato quello di concentrarsi costantemente sulle stesse metriche fondamentali anno dopo anno. Sebbene le domande dei loro sondaggi si siano evolute e siano emerse nuove conclusioni nel corso del tempo, le quattro metriche chiave utilizzate come parametri di riferimento sono rimaste invariate:

1. **Lead Time** - Tempo che passa dal commit al deploy di uno sviluppo
2. **Frequenza di deploy** (in produzione)
3. **Change Fail Percentage** - Percentuale di fallimento di uno sviluppo
4. **Mean Time to Restore** - Tempo medio di ripristino (da un fallimento in produzione)

Le prime due metriche riguardano l'innovazione e il rapido rilascio di nuove funzionalità. La terza e la quarta metrica riguardano la stabilità e la riduzione dei difetti e dei tempi di inattività. In quanto tali, queste metriche sono in linea con il duplice obiettivo di DevOps, ovvero "muoversi velocemente e non rompere le cose".
Sono anche in linea con i due principi fondamentali della gestione Lean, derivati dal sistema di produzione Toyota: "just-in-time" e "stop-the-line". Come già detto, il just-in-time è il principio secondo cui la massima efficienza deriva dalla riduzione degli sprechi nel sistema di lavoro e che il modo per ridurre gli sprechi è ottimizzare il sistema per gestire lotti sempre più piccoli e consegnarli con velocità crescente. Stop-the-line significa che il sistema di lavoro è regolato non solo per accelerare le consegne, ma anche per identificare immediatamente i difetti per evitare che vengano rilasciati, aumentando così la qualità del prodotto e riducendo la probabilità di fallimenti di produzione.

Il **Lead Time** è importante perché più breve è il tempo di consegna, più rapidamente si può ricevere un feedback sul software e quindi più velocemente si possono rilasciare innovazioni e miglioramenti. Una sfida nel misurare il Lead Time è costituita da due parti: il tempo di sviluppo di una funzionalità e il tempo di consegna della stessa. Il tempo di sviluppo di una funzionalità inizia dal momento in cui viene richiesta, ma ci sono alcuni motivi legittimi per cui una funzionalità può essere deprioritizzata e rimanere nel backlog di un prodotto per mesi o anni. Esiste un'elevata variabilità intrinseca nel tempo necessario per passare da "feature richiesta" a "feature sviluppata". Per questo motivo, il Lead Time nello

State of DevOps Report si concentra sulla misurazione del tempo necessario per consegnare una funzionalità solo una volta che sia già stata sviluppata. La parte del ciclo di vita relativa alla consegna del software è una parte importante del Lead Time totale ed è anche molto più coerente. Misurando il Lead Time dal “codice committato” al “codice deployato”, si possono iniziare a sperimentare miglioramenti del processo che riducano l'attesa e l'inefficienza, consentendo così un feedback più rapido.

La **frequenza di deploy** è la frequenza con cui le modifiche al codice o alle configurazioni vengono distribuite in produzione. La frequenza di deploy è importante perché è inversamente correlata alla dimensione del batch di deploy. I team che deployano in produzione una volta al mese, necessariamente deployano un batch di modifiche più grande rispetto ai team che deployano una volta alla settimana. Le modifiche non sono tutte uguali. All'interno di ogni batch di modifiche, ce ne saranno alcune estremamente preziose e altre quasi insignificanti. I batch di grandi dimensioni implicano che le funzionalità di valore rimangano in attesa insieme a tutte le altre modifiche, ritardando così la fornitura di valore e benefici. I batch di grandi dimensioni aumentano anche il rischio di fallimenti nell'implementazione e rendono molto più difficile diagnosticare quale dei molti cambiamenti sia responsabile se si verifica un errore. I team tendono naturalmente a raggruppare le modifiche quando gli sviluppi sono gravosi e noiosi. Misurando la frequenza di deploy, è possibile seguire i progressi del team, mentre si lavora per rendere le distribuzioni meno dolorose e consentire dei batch più piccoli.

La **percentuale di fallimento** (Change Fail Percentage) delle modifiche misura la frequenza con cui un deploy in produzione fallisce. Per fallimento si intende un'interruzione o un degrado del sistema provocato da uno sviluppo, o il fatto che quest’ultimo richieda un successivo hotfix o rollback. I sistemi software moderni sono complessi e in rapida evoluzione, quindi un certo numero di guasti è inevitabile. Tradizionalmente si ritiene che esista un compromesso tra la frequenza delle modifiche e la stabilità dei sistemi, ma i team più efficaci identificati nello *State of DevOps Report* sono caratterizzati sia da un alto tasso di innovazione che da un basso tasso di fallimenti. La misurazione del tasso di fallimento consente al team

di monitorare e mettere a punto i processi per garantire che i test eliminino la maggior parte dei guasti prima che si verifichino.

Il **tempo medio di ripristino** (Mean Time to Restore, o MTTR) è strettamente correlato al tempo di rilascio delle funzionalità. In effetti, i team che possono rilasciare rapidamente le funzionalità possono anche rilasciare rapidamente le patch. Il tempo di ripristino indica la quantità di tempo in cui un sistema di produzione rimane inattivo, in uno stato degradato o con funzionalità non complete. Questi incidenti sono tipicamente situazioni di stress e spesso hanno implicazioni finanziarie. Risolvere questi incidenti è una priorità fondamentale per i team operativi. La misurazione di questa metrica permette al team di stabilire una linea di base sul tempo di recupero e di lavorare per risolvere gli incidenti sempre più velocemente.

Insieme, queste metriche costituiscono la **Software Delivery Performance** di un team. L'obiettivo di qualsiasi iniziativa DevOps dovrebbe essere quello di migliorare la Software Delivery Performance aggiornando strategicamente capacità specifiche come la Continuous Delivery e l'uso di test automatizzati.
Il modo in cui un team misura queste capacità è un'altra sfida. Le metriche automatizzate possono essere implementate nel tempo, anche se il meccanismo per farlo dipenderà dal modo in cui vengono effettuati i deploy.
I sondaggi forniscono un'approssimazione ragionevole per queste metriche, specialmente se le risposte sono date da membri del team che ricoprono ruoli diversi. Le linee guida per la somministrazione di tali sondaggi esulano dallo scopo di questa tesi, ma le risposte oneste dei team sono il fattore più critico. Bisogna evitare qualsiasi politica che possa spingere il team a esagerare le risposte verso l'alto o il basso. Non bisogna usare mai questi sondaggi per premiare o punire; dovrebbero essere usati semplicemente per informare. È buono permettere ai team di monitorare i propri progressi e di sfidarsi a migliorare per il proprio beneficio e per quello della compagnia. Come si legge nel Manifesto Agile: "*At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.*"

Ottimizzazione del Value Stream (Flusso di valore)

Un concetto fondamentale del Lean Management del software è quello di comprendere il processo di sviluppo e consegna del software in termini di come ogni aspetto del processo contribuisca (o meno) al valore per l'utente finale. Come già detto, il valore riflette il beneficio che l'utente finale riceve e per il quale sarebbe disposto a pagare. Mentre il software si muove attraverso il processo di sviluppo e consegna, diversi individui e team aggiungono valore attraverso i loro contributi. I loro contributi continui al processo sono descritti come un flusso di valore che fornisce un valore continuo agli utenti finali.
La mappatura del flusso di valore è uno strumento chiave del Lean Management. Si tratta di mappare ogni fase del processo di sviluppo e delivery e di raccogliere tre metriche per ogni fase: Lead Time, Process Time e percentuale di completamento e accuratezza. Il Lead Time è il tempo totale che intercorre tra l'ingresso di un work-item in una determinata fase e la sua uscita dalla stessa. Il Process Time è la quantità di tempo che sarebbe necessaria per quell'attività se potesse essere eseguita senza interruzioni. La percentuale di completamento e accuratezza è la percentuale di output di quella fase che può essere utilizzata, così com'è, senza richiedere rielaborazioni o chiarimenti. L'obiettivo di questa mappatura è identificare le fasi di rallentamento in cui i work-item rimangono in attesa o potrebbero essere completati in modo più efficiente, nonché le fasi che soffrono di problemi di qualità. È anche possibile che questo processo faccia emergere fasi che non aggiungono molto valore e che possono essere eliminate o semplificate. Il concetto centrale del Lean Management consiste nel valutare questo processo e identificare come eliminare gli sprechi da questo sistema, per massimizzare il flusso di valore.
I potenziali miglioramenti tecnici che possono essere apportati a un prodotto software o al modo di lavorare su tale prodotto sono illimitati. Ma non è saggio pensare che l'adozione di DevOps implichi l'adozione simultanea di tutti questi miglioramenti. L'obiettivo dovrebbe invece essere quello di identificare i colli di bottiglia del sistema che limitano il flusso complessivo di valore e di ottimizzare il sistema attorno a tali colli di bottiglia. Un approccio complementare, noto come *Teoria dei Vincoli*, può fornire una pratica semplificazione per aiutare a individuare le aree prioritarie di miglioramento.

Teoria dei Vincoli (o Ostacoli)

Il concetto noto come Teoria dei Vincoli è stato introdotto e reso popolare da Eliyahu M. Goldratt nel suo libro *The Goal*. Questo libro è ampiamente citato nella letteratura DevOps ed enfatizza un quadruplice approccio all'ottimizzazione del flusso di valore attraverso un sistema:

1. Identificazione del vincolo
2. Sfruttamento del vincolo
3. Subordinamento e sincronizzazione al vincolo
4. Aumento della performance del vincolo

In parole povere, l'idea è che le prestazioni complessive di qualsiasi sistema in un dato momento saranno sempre limitate da un singolo vincolo (o ostacolo). In un processo di sviluppo software, il vincolo potrebbe essere rappresentato dal fatto che non ci sono abbastanza sviluppatori, o che il processo di test richiede troppo tempo, o che il processo di distribuzione può essere eseguito solo in determinati momenti, o che non ci sono ulteriori richieste da parte dei clienti, e così via. Il punto più importante da notare è che **i miglioramenti apportati a qualsiasi parte del sistema al di fuori di questo vincolo non produrranno alcun beneficio significativo**. Per questo motivo, il passo più critico per apportare miglioramenti complessivi è quello di identificare prima il vincolo.

L'identificazione del vincolo di un sistema può essere un processo impegnativo, che combina misurazione, intuizione e sperimentazione. Un approccio per identificare il vincolo consiste nel cercare un accumulo di "scorte" in un sistema, poiché questo tende a indicare un punto in cui il processo rallenta. Il concetto di “scorte” è preso in prestito dalla manifattura, ma può essere esteso al processo di sviluppo del software e ridenominato "work-in-progress". Dove si accumulano i work-in-progress (WIP) in un processo di sviluppo? Si possono porre ad un team domande come: quanto è grande il backlog degli sviluppatori? Quanti lavori sono in attesa di revisione da parte di un tech lead? Quanto lavoro è in attesa di essere testato? Quanti elementi sono in attesa di essere deployati? Domande come queste e l'osservazione delle tendenze nel tempo possono aiutarvi a individuare il vincolo principale del vostro processo.

Una volta individuato il vincolo, il miglioramento più grande che si può attuare è **sfruttare tale vincolo**, assicurandosi che venga utilizzato appieno. All'epoca in cui Goldratt scrisse *The Goal*, nel settore manifatturiero era comune concentrarsi sul massimo utilizzo di ogni singolo sistema coinvolto nel processo produttivo. L'equivalente in un team di sviluppo software è assicurarsi che tutti siano sempre occupati. Ma in pratica questa ottimizzazione locale di ogni parte del sistema non ottimizza le prestazioni del sistema complessivo, perché le prestazioni complessive sono sempre definite da un unico vincolo. È questo vincolo che deve essere ottimizzato, ottenendo la massima produttività sostenibile da quel vincolo.

La terza fase consiste nel **subordinare e sincronizzare il vincolo**. Ciò significa che ogni altra parte del sistema deve essere vista come un mezzo per abilitare e supportare il vincolo. La priorità per gli altri sistemi dovrebbe essere quella di garantire che il vincolo non sia mai in attesa di "materie prime" (come le specifiche per il lavoro) e che il lavoro prodotto dal vincolo sia rapidamente trasferito alla fase successiva del processo.

Se un vincolo persiste nonostante la massimizzazione del suo throughput, rimane solo un'opzione: **aumentare la performance di quel vincolo**. Se il vincolo è costituito da uno o più individui, il miglioramento delle prestazioni di tale vincolo può consistere nel fornire loro strumenti migliori, nella formazione o nel training, nell'assumere altri membri per quel team o addirittura nel sostituirli e spostarli in un ruolo diverso, se necessario.

Questo quadruplice processo è una danza continua. Non c'è mai un momento in cui un sistema non sia soggetto a vincoli. Anche quando è il mercato stesso a costituire un vincolo, la prassi è la stessa: sfruttare quel mercato, subordinare le altre attività all'alimentazione della domanda del mercato e infine lavorare per aumentare la domanda attraverso il marketing e la pubblicità.
Anche lo stesso processo di ottimizzazione è soggetto a vincoli invisibili. Ci vogliono tempo, sforzi e consapevolezza della situazione per capire come si sta comportando il flusso di valore. E possono volerci tempo, sforzi e denaro per apportare le modifiche necessarie a migliorare tali prestazioni.

Dal punto di vista della teoria dei vincoli, questa danza continua è l'essenza di una gestione efficace.

CAPITOLO 3

Salesforce DevOps

Cos’è Salesforce?

Salesforce è una piattaforma software basata su cloud che fornisce una gamma di applicazioni e servizi aziendali per la gestione delle relazioni con i clienti, le vendite, il marketing e altri processi aziendali. È costruita su un'architettura multi-tenant che consente a più organizzazioni di condividere la stessa infrastruttura, mantenendo allo stesso tempo un rigoroso isolamento e la sicurezza dei dati.
Salesforce fornisce una vasta gamma di strumenti e servizi per gli sviluppatori per creare applicazioni personalizzate e integrazioni sulla piattaforma, tra cui il suo linguaggio di programmazione Apex, il framework Lightning Web Components e una gamma di API e SDK. Gli sviluppatori possono utilizzare questi strumenti per creare applicazioni e automatizzazioni personalizzate che sono strettamente integrate con la funzionalità principale di Salesforce.

Salesforce è un **CRM SaaS Multi-Tenant**.

CRM

Un **CRM** (Customer Relationship Management) è un software o un sistema utilizzato dalle aziende per gestire le interazioni con i propri clienti. In sostanza, un CRM consente alle aziende di organizzare, automatizzare e sincronizzare le attività di vendita, marketing e assistenza clienti, al fine di migliorare l'esperienza complessiva del cliente e aumentare la fidelizzazione.
I sistemi CRM raccolgono dati sui clienti da diverse fonti, come le interazioni dei clienti con il sito web dell'azienda, le conversazioni telefoniche o via e-mail con il servizio clienti, i social media e molto altro ancora. Questi dati vengono quindi elaborati e utilizzati per fornire alle aziende informazioni sulle esigenze dei propri clienti, sui loro comportamenti di acquisto e sui loro feedback sul prodotto o servizio.
Tra le funzionalità di base di un sistema CRM ci sono la gestione dei contatti e dei lead, la gestione delle opportunità di vendita, la gestione dei ticket e delle richieste di assistenza clienti, la gestione delle campagne di marketing e la generazione di report e analisi. Inoltre, molte piattaforme CRM sono personalizzabili per soddisfare le esigenze specifiche dell'azienda e integrarsi con altri sistemi software utilizzati dall'azienda stessa.

SaaS

SaaS, acronimo di "Software as a Service" (Software come servizio), è un modello di distribuzione software in cui il provider di servizi informatici gestisce e distribuisce il software ai propri clienti tramite internet. In pratica, anziché installare il software sul proprio computer o server, l'utente accede all'applicazione tramite un browser web o un'applicazione dedicata, utilizzando un sistema di login fornito dal provider SaaS.
Il provider SaaS gestisce l'intera infrastruttura hardware e software, inclusi server, reti, database e applicazioni, garantendo la disponibilità, la sicurezza e l'aggiornamento del software. L'utente paga solitamente un

canone mensile o annuale per l'utilizzo del servizio, e può spesso scegliere tra diverse opzioni di abbonamento in base alle proprie esigenze.
Il modello SaaS offre numerosi vantaggi rispetto ai modelli di distribuzione software tradizionali: ad esempio, riduce la necessità di acquistare hardware e software costosi e di mantenere personale IT dedicato, in quanto tutti i servizi sono forniti dal provider SaaS. Inoltre, grazie alla distribuzione via internet, l'accesso all'applicazione è possibile da qualsiasi dispositivo connesso a Internet, semplificando la mobilità e la collaborazione tra team geograficamente distribuiti.

Multi-Tenancy

La **Multi-Tenancy** è un modello di architettura software in cui un'unica istanza di un'applicazione viene utilizzata simultaneamente da più clienti (tenants). In pratica, il software viene progettato per supportare più organizzazioni o gruppi di utenti, ognuno dei quali ha accesso solo alle proprie risorse e dati, senza poter accedere o interferire con le risorse degli altri.
Nel modello Multi-Tenant, il software e l'infrastruttura sottostante sono condivisi tra tutti i tenant, ma ogni tenant è isolato e protetto dalle altre organizzazioni. Ciò consente di ridurre i costi per i provider di servizi informatici, poiché l'infrastruttura è condivisa e gestita centralmente, e di fornire ai clienti un accesso rapido e flessibile al software e ai dati di cui hanno bisogno.
La Multi-Tenancy viene spesso utilizzata in ambito SaaS. Tuttavia, può essere applicata anche ad altre architetture software, come ad esempio nei data center condivisi, dove più clienti utilizzano gli stessi server per ospitare i propri siti web o applicazioni.

Oggi Salesforce è molto più di un CRM dedicato alla vendita, e grazie anche a tante acquisizioni di successo (ad oggi sono state acquisite nel gruppo più di 70 società), Salesforce è diventata una delle più importanti compagnie cloud al mondo, spaziando appunto dalle soluzioni CRM alle

soluzioni di analytics/business intelligence, marketing, commerce B2C e B2B, il tutto corredato da tanta intelligenza artificiale.
Negli ultimi anni, è letteralmente esploso un nuovo mercato in cui c'è una fortissima esigenza di competenze tecniche e funzionali legate all'ecosistema Salesforce. Nell'ecosistema di prodotti il CRM è al centro di tutto, non a caso la piattaforma è denominata "Customer 360 Platform", ovvero vuole mettere al centro del sistema informativo il cliente.
Ciò che differenzia maggiormente Salesforce dalle altre aziende competitor è la volontà genuina di portare le persone, siano esse clienti, consulenti, dipendenti o partners, ad essere parte di una community dove il profitto economico è solo uno dei tanti fattori chiave che portano il successo all'azienda che adotta Salesforce.

Strumenti di Salesforce

Come è già stato detto, Salesforce offre una vasta gamma di strumenti e servizi per gli sviluppatori, andiamo a vedere quali sono quelli fondamentali:

SObject

In Salesforce, un SObject (abbreviazione di "Salesforce Object") è un oggetto di database che rappresenta un tipo di record (ovvero un'istanza di un SObject). Ad esempio, gli oggetti standard come Account, Contact, Opportunity, Lead, Case, ecc. sono tutti SObject predefiniti di Salesforce. Inoltre, gli amministratori di Salesforce possono anche creare oggetti personalizzati che rappresentano tipi di record specifici per le esigenze aziendali.
Ogni SObject ha un insieme predefinito di campi, che rappresentano le proprietà dei record associati a quell'oggetto. Ad esempio, l'oggetto Account ha campi come "Nome", "Indirizzo", "Telefono", ecc. Esistono campi che hanno la funzione di relazionare SObjects tra di loro e possono essere di due tipi:

1. **Lookup**: la lookup è un campo che permette ad un record di uno specifico SObject di puntare ad un altro record di un altro (o lo stesso) specifico SObject. Essa crea una relazione di parentela "da

padre a figlio – uno a molti", dove i figli sono i record sui quali è definita la relazione di lookup, e il padre è il record puntato dalla lookup. Il record padre è ignaro di quanti e quali figli esso abbia.

2. **Master-Detail**: la master-detail si comporta in parte come la lookup, ma introduce dei nuovi comportamenti; se il record padre viene eliminato, con esso vengono eliminati tutti i record figli. Infine, grazie alla relazione di parentela stretta instaurata con la master-detail, i figli sono perfettamente visibili dal record padre, che può mostrare valori che rispecchiano la quantità e le qualità dei figli che gli appartengono. Questi valori prendono il nome di Roll-up Summary.

Inoltre, ogni SObject ha un insieme di metadati che definiscono il comportamento dell'oggetto. Questi metadati includono le autorizzazioni degli utenti per visualizzare, modificare o eliminare i record, le relazioni tra gli oggetti, le regole di validazione dei campi, i flussi di lavoro, i trigger, le pagine di layout e così via.
Gli SObject di Salesforce sono fondamentali per la creazione e la gestione dei record all'interno del sistema Salesforce.

Record Type

Un SObject può avere uno o più Record Type.
Un Record Type è una entità che permette di definire un insieme di campi e di impostazioni che definiscono il comportamento di un record in base alle esigenze dell'azienda.
In pratica, i Record Type consentono di personalizzare la visualizzazione e le opzioni di un record a seconda del tipo di record, del ruolo dell'utente che accede al record e del processo di business associato. Ad esempio, un'azienda che possiede un SObject "Cliente" potrebbe avere un Record Type per i clienti potenziali, un altro per i clienti attivi e un altro ancora per i clienti che hanno chiuso il contratto. Quindi un record dell'SObject "Cliente" di questa ipotetica azienda apparterrebbe ad uno di questi tre Record Type.
I Record Type vengono utilizzati per creare processi personalizzati per ogni tipo di record, definire le autorizzazioni degli utenti, definire le azioni disponibili e la loro visibilità e creare flussi di lavoro personalizzati per i record.

Rappresentano una potente funzionalità di Salesforce che consente di personalizzare l'esperienza dell'utente e migliorare l'efficienza dei processi aziendali.

Validation Rule

Una Validation Rule è una regola che consente di definire la logica di validazione dei dati inseriti in un record prima che venga salvato nel database. Le Validation Rule permettono di definire le condizioni che un record deve soddisfare affinché sia considerato valido e possa essere salvato. Esse possono essere applicate su qualsiasi oggetto di Salesforce e possono essere utilizzate per verificare che i dati inseriti rispettino determinate condizioni, come ad esempio la presenza di un valore obbligatorio in un campo, la correttezza del formato di una data o la congruenza tra i valori di più campi.
Se una Validation Rule non viene soddisfatta quando un utente tenta di salvare un record, Salesforce visualizzerà un messaggio di errore che indica la causa del fallimento della validazione. Questo aiuta gli utenti a correggere i dati e a garantire che solo i record validi siano salvati nel database.
Le Validation Rule sono uno strumento importante per garantire l'integrità dei dati all'interno di Salesforce e possono essere utilizzate in combinazione con altre funzionalità come i trigger per creare processi personalizzati che migliorano l'efficienza e l'accuratezza dei processi aziendali.

Trigger

Un trigger è un blocco di codice personalizzato che viene eseguito in risposta a determinati eventi che si verificano sui record del database. Un trigger è un'azione automatizzata che viene scatenata quando un record viene creato, modificato o eliminato.
I trigger sono scritti in Apex, il linguaggio di programmazione proprietario di Salesforce, e sono associati a un SObject specifico di Salesforce. Quando si verifica un evento su uno o più record associati a quell'oggetto, il trigger viene attivato e può eseguire una serie di azioni personalizzate, come ad esempio:

- Modificare i valori dei campi del record
- Creare, modificare o eliminare altri record di Salesforce
- Invocare servizi esterni o effettuare chiamate API
- Generare notifiche o messaggi di avviso
- Eseguire calcoli complessi o processi di business

I trigger sono un'importante funzionalità di Salesforce che consente di personalizzare il comportamento dei record in base alle esigenze aziendali. Ad esempio, una compagnia potrebbe utilizzare un trigger per automatizzare il processo di assegnazione dei Case ai dipendenti in base alla loro disponibilità o competenze, o per aggiornare automaticamente un campo di un record in base a un calcolo complesso.
Tuttavia, è importante utilizzare i trigger con attenzione e comprensione, in quanto possono avere un impatto significativo sulle prestazioni del sistema e sulla qualità dei dati. È consigliabile utilizzare le best practice di sviluppo e i test per garantire che i trigger siano affidabili e non causino problemi nel sistema.

Salesforce DX

Salesforce DX (Developer Experience) è una piattaforma di sviluppo per la creazione di applicazioni Salesforce con un'esperienza moderna di sviluppo. È stata progettata per migliorare la produttività dei team di sviluppo e offrire un ciclo di vita del software più efficiente, con un maggiore controllo sulla gestione del codice sorgente e delle versioni.
Salesforce DX fornisce una suite di strumenti di sviluppo, tra cui un ambiente di sviluppo integrato (IDE) basato su cloud, strumenti di gestione del codice sorgente, testing automatizzato, Continuous Integration e Continuous Delivery (CI/CD) e strumenti di collaborazione per i team di sviluppo.
Inoltre, Salesforce DX utilizza la metodologia delle componenti, che consente ai team di sviluppo di creare e gestire le applicazioni Salesforce come un insieme di componenti front-end riutilizzabili, semplificando la gestione del codice sorgente e migliorando la collaborazione tra i team di sviluppo.
Salesforce DX consente anche di utilizzare la Salesforce CLI (Command Line Interface), uno strumento di linea di comando per la gestione

dell'ambiente Salesforce, che permette di automatizzare attività di sviluppo.

DevOps su Salesforce

Il DevOps applicato a Salesforce è di conseguenza un approccio che mira a migliorare l'efficienza e la qualità dello sviluppo e del rilascio delle applicazioni Salesforce attraverso l'automazione e l'integrazione continua dei processi.
I concetti fondamentali di DevOps sono adottati anche dai team che lavorano sulla piattaforma Salesforce, e sono stati ampiamente affrontati nel capitolo 2.

Quindi, cosa ha bisogno di implementare un'impresa per applicare tali concetti al proprio flusso di rilascio delle funzionalità? Per poter rispondere a questa domanda, vanno capiti i concetti di Org Strategy e Branching Strategy.

ORG Strategy

Concetto di Salesforce Org

In Salesforce, un'**Org** (abbreviazione di Organization) è l'ambiente di lavoro in cui vengono creati, gestiti e personalizzati gli oggetti, le funzionalità e i dati dell'applicazione.
Un'Org di Salesforce è un'istanza indipendente del software Salesforce, con il proprio insieme di dati, utenti, permessi e configurazioni. Può essere utilizzata da un singolo utente o da un'intera organizzazione, a seconda delle esigenze.

Esistono diverse tipologie di Org, ad esempio l'Org di sviluppo (utilizzata per la creazione e il testing delle applicazioni), l'Org di produzione (utilizzata per l'accesso e l'utilizzo delle applicazioni da parte degli utenti finali) e l'Org di sandbox (utilizzata per testare le modifiche dell'applicazione senza influire sulla versione di produzione).
Inoltre, ogni Org ha il proprio URL, che identifica l'ambiente di lavoro specifico e consente l'accesso ai dati e alle funzionalità dell'applicazione.

Essendo Salesforce un SaaS, ogni tipo di Org è associata ad una specifica subscription:

1. Salesforce **Developer Edition** Org: è un'Org gratuita utilizzata per la creazione, lo sviluppo e il testing di applicazioni Salesforce. Questo tipo di Org non può essere utilizzato per scopi produttivi e ha alcune limitazioni sui dati e sui volumi di attività.
2. Salesforce **Trial** Org: è un'Org gratuita utilizzata per provare l'applicazione Salesforce per un periodo di tempo limitato (generalmente 30 giorni). Questo tipo di Org consente di sperimentare tutte le funzionalità dell'applicazione e di testare il sistema in un ambiente protetto.
3. Salesforce **Production** Org: è un'Org a pagamento utilizzata per l'accesso e l'utilizzo delle applicazioni Salesforce da parte degli utenti finali. Questo tipo di Org contiene tutti i dati e le configurazioni necessarie per l'utilizzo dell'applicazione in un ambiente produttivo.

4. Salesforce **Sandbox** Org: è un'Org utilizzata per testare le modifiche dell'applicazione senza influire sulla versione di produzione. Questo tipo di Org consente di copiare i dati e le configurazioni dell'Org di produzione in un ambiente di testing protetto, al fine di verificare le modifiche prima di applicarle alla versione di produzione.

Esistono anche altri tipi di Org subscription, come l'**Enterprise Edition**, l'**Unlimited Edition** e la **Partner Edition**, ma queste sono generalmente utilizzate da sviluppatori e partner Salesforce per specifiche esigenze di sviluppo e personalizzazione.

Branching strategy

La **branching strategy** è una strategia di gestione del codice sorgente in cui si definiscono regole per il modo in cui i rami (branch) del repository vengono creati, utilizzati e integrati tra di loro.
In altre parole, la branching strategy è un insieme di linee guida che stabiliscono come i diversi branch del repository devono essere utilizzati, quando e da chi devono essere creati, e come devono essere uniti tra di loro per garantire che il codice sia stabile e che gli sviluppatori possano lavorare in modo efficiente e coordinato.
Una buona branching strategy permette di gestire in modo efficace il flusso di lavoro degli sviluppatori, di mantenere il codice stabile e di ridurre i rischi di conflitti o di errori nella gestione del codice sorgente.
Su Salesforce, generalmente, la branching strategy segue la Org strategy di pari passo. Questo perché in questa maniera si fa sì che ad ogni branch principale corrisponda un ambiente di sviluppo (**Figura 4**).

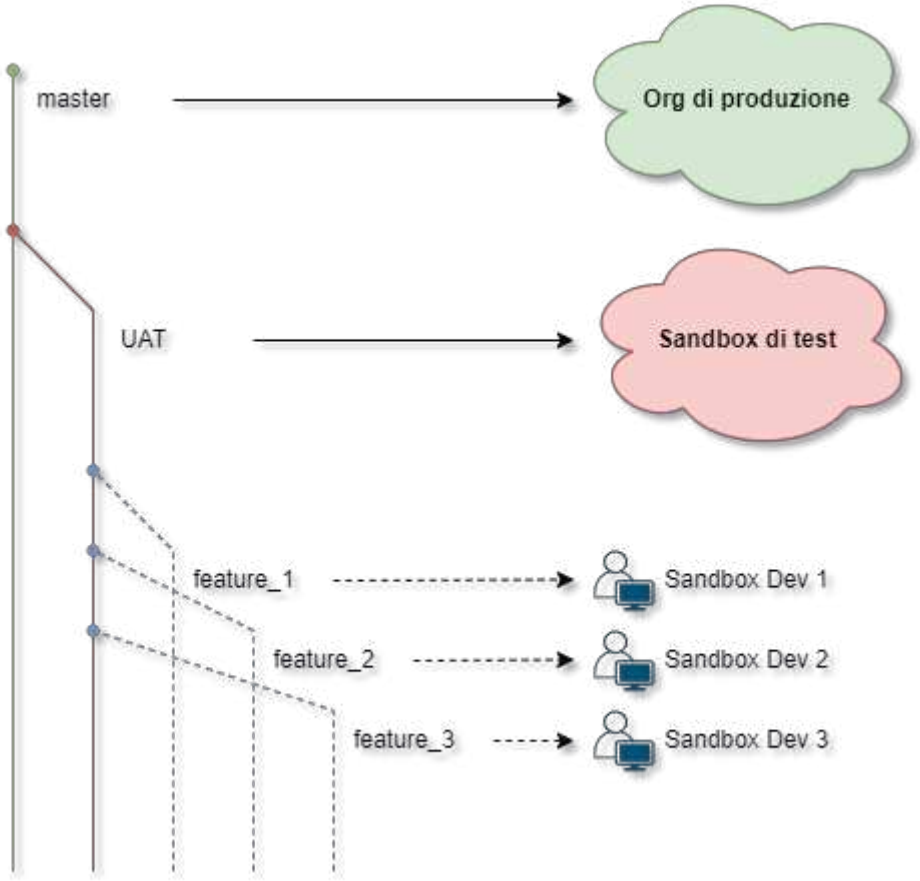

Figura 4. *Uno schema di una semplice branching strategy su un sistema Salesforce con tre ambienti (una Org di produzione, una sandbox di test e molteplici sandbox di sviluppo parallele).*

In questa strategia, i team di Dev lavorano su più sandbox di sviluppo parallele, ognuno su un "feature branch" staccato dal branch di UAT (User Acceptance Testing, branch specchio della sandbox di test). Ogni feature branch rappresenta lo sviluppo di una nuova funzionalità o un miglioramento del sistema, e viene sviluppato in isolamento dagli altri team.

Quando il lavoro sul feature branch è stato completato e testato sull'ambiente di sviluppo, viene effettuato un merge sulla sandbox di test per verificare l'interoperabilità tra i diversi feature branch. Se tutto funziona correttamente, viene quindi effettuato un merge su ambiente di produzione.

Per mantenere allineate le sandbox di sviluppo con quella di test e di produzione, viene effettuato un refresh degli ambienti a partire da quello

di produzione. La cadenza con cui viene effettuato questo refresh è arbitraria, e può variare dalle settimane ai mesi.
Una strategia semplice come questa consente di gestire in modo efficace lo sviluppo parallelo di diverse funzionalità, di isolare eventuali errori e di mantenere una versione stabile e funzionante del sistema in produzione.

Per implementare tale flusso di sviluppo, una compagnia che adotta Salesforce piò velocezzare questo processo appoggiandosi ad una (o più) delle integrazioni per DevOps disponibili sullo store AppExchange di Salesforce. Nel capitolo successivo tratteremo delle alternative esistenti, con un focus su Copado, l'integrazione leader nel settore, e DevOps Center, un'app nativa sviluppata da Salesforce e rilasciata al pubblico nel mese di dicembre 2022.

DevOps Management su Salesforce

Overview generale delle piattaforme

Come già menzionato, Salesforce permette l'integrazione di piattaforme di DevOps Management. Queste piattaforme permettono una perfetta gestione della catena di DevOps da parte degli amministratori di sistema Salesforce, rendendo questa struttura elastica in base alle esigenze del gruppo di lavoro o della compagnia che le utilizza.

Le piattaforme di DevOps Management hanno tutte (nessuna esclusa) degli aspetti fondamentali comuni:

1. Gestione della Pipeline di sviluppo
2. Creazione ed avanzamento dei task di sviluppo
3. Validazione su ambiente di deploy
4. Automatizzazione del deploy

Su ogni Org e su ogni sandbox create dalla Org di produzione vengono continuamente inserite nuove funzionalità che dovranno, in un momento o nell'altro, confluire nell'Org di produzione stessa. Ciò si rispecchia anche nei branch del Version Control: nel momento in cui uno sviluppatore "stacca" un branch da UAT (branch/ambiente di test), andrà a versionare le proprie modifiche su quel nuovo branch, per poi, una volta effettuati i dovuti test, iniziare una catena di merge finchè quel nuovo sviluppo diventerà parte integrante dell'ambiente di produzione. A questo concetto si riferisce la **gestione della pipeline di sviluppo** (figure 5 e 6), che serve a parallelizzare branch e Org, in maniera tale da "standardizzare" per ogni team il flusso di sviluppo verso produzione.

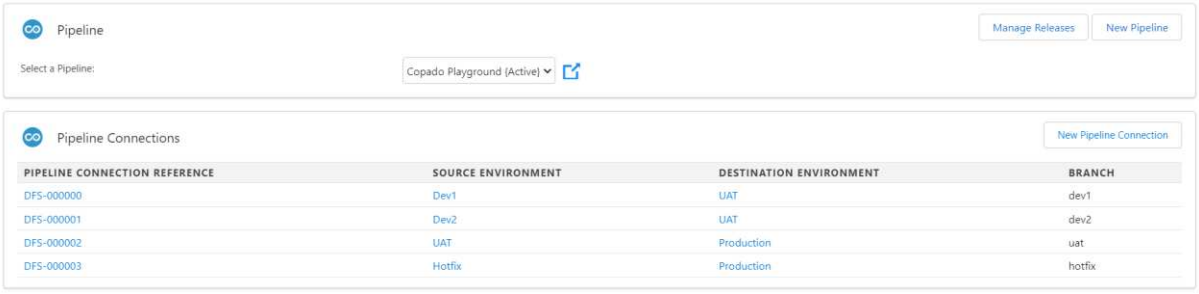

Figura 5. *Esempio di gestione della pipeline su Copado.*

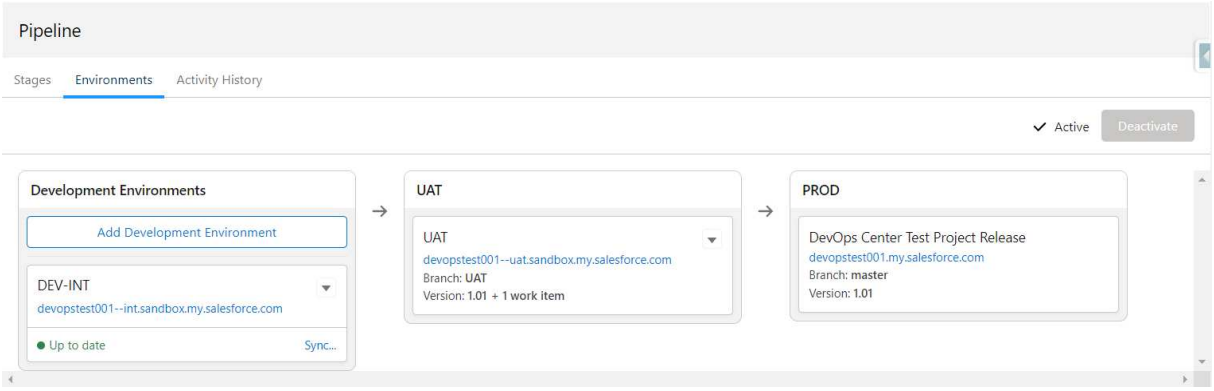

Figura 6. *Esempio di gestione della pipeline su Salesforce DevOps Center.*

Per poter usufruire della pipeline si ha bisogno di un metodo per tracciare ogni singolo sviluppo di funzionalità. Ciò è possibile attraverso la **creazione dei task di sviluppo**, che sono quelle entità che riassumono al proprio interno non solo l'intento di uno specifico sviluppo, ovvero quale sarà la nuova funzionalità a cui fa riferimento un task, ma anche il tracciamento di tutte le modifiche, i test e i tentativi di deploy che riguardano quello specifico sviluppo. In altre parole, i task di sviluppo rispecchiano in tutto e per tutto i branch che gli sviluppatori hanno creato a partire dall'ambiente di test e su cui stanno facendo versioning delle proprie modifiche, e non solo: quando tale branch sarà deployato in ambiente di test, il task di sviluppo continuerà a seguire la vita di quelle modifiche, come se incluse in una sorta di pacchetto, fino al loro arrivo a destinazione in produzione.

Essendo queste piattaforme perfettamente integrate in Salesforce, è possibile scegliere che tipo di **validazione** effettuare sui componenti software che passano da un ambiente al successivo. È ovvio che se si vuole portare avanti una determinata modifica in produzione, questa deve aver superato sufficienti test, ma potrebbe non bastare. Infatti, mentre in ambiente di test uno sviluppo funziona, non è detto che in produzione esistano i metadati necessari (quindi SObjects e loro campi, Classi Apex e Componenti front-end) per restituire il risultato atteso. L'Org di produzione potrebbe non essere ancora perfettamente allineata con la sandbox di test, per cui, ogni sviluppo, prima di poter essere distribuito nell'Org di destinazione deve superare dei test di validazione sui metadati. Inoltre, sebbene però la seguente sia una funzionalità intrinseca a

Salesforce, ogni classe Apex deve essere coperta dalle classi di test per almeno il 75% della propria interezza nel momento in cui sta essendo deployata in produzione. Quindi per le classi Apex viene eseguita una validazione particolare che non riguarda solo i metadati presenti all'interno del merge tra l'ambiente di destinazione e la modifica in atto, ma riguarda anche l'esecuzione di esse. Le piattaforme di DevOps Management forniscono la possibilità di scegliere se effettuare questo tipo di validazione prima di effettuare quella vera e propria all'interno di Salesforce: ciò permette un vantaggio, nel senso che il processo di rilevazione di un errore di validazione avviene in maniera molto più rapida.

Quando la validazione viene passata con successo, ecco che avviene un **deploy automatizzato** nell'ambiente di destinazione. In questa fase vengono realizzati contemporaneamente sia il merge delle modifiche verso il branch di destinazione (UAT se l'ambiente di destinazione è quello di test, master se l'ambiente di destinazione è quello di produzione) sia il deploy di tutti i metadati coinvolti sulla Org di destinazione.

Queste automatizzazioni rendono molto più snello il compito di chi si occupa di Release Management e danno più spazio al monitoraggio delle singole modifiche: in caso di errore, sarà molto semplice risalire a chi ha effettuato una certa modifica e quando è stata inserita nell'ambiente. Ricordiamo che uno dei principali obiettivi del Lean Management per quanto riguarda gli ambienti di produzione è minimizzare il downtime in caso di errori post-deploy, per cui queste modalità di gestione automatizzata dei processi di test e di rilascio hanno esattamente questo scopo.

Procediamo ora ad elencare le funzionalità specifiche che le piattaforme di DevOps Management per Salesforce singolarmente offrono, partendo da Copado per arrivare alla più nuova Salesforce DevOps Center.

CAPITOLO 4

Copado

Copado è stata fondata nel 2013 da due release manager europei di Salesforce con sede a Madrid, Spagna, per alleviare la pesantezza, la complessità e i rischi relativi al processo di distribuzione di Salesforce. Da allora hanno ottenuto capitale di crescita da Salesforce Ventures e Insight Ventures, hanno attirato oltre 150 clienti a livello globale e hanno assunto un esperto team di senior leadership statunitense per costruire la propria attività negli Stati Uniti.

Copado utilizza Salesforce come interfaccia utente, per l'autenticazione e per archiviare i dati su organizzazioni, metadati e implementazioni. Ma delega l'elaborazione del backend a Heroku. Questo permette a Copado di sfruttare la potenza e la velocità di Heroku per gestire il reperimento dei metadati, l'elaborazione e le distribuzioni. Questa architettura consente ai clienti di personalizzare gli aspetti del front-end di Copado e di accedere ai dati e alla logica aziendale direttamente all'interno di Salesforce.

Breve finestra su Heroku

Heroku è una piattaforma di hosting cloud che consente di eseguire applicazioni web in vari linguaggi di programmazione, come Ruby, Java, Python, Node.js e molti altri.

Heroku è stata fondata nel 2007 e inizialmente si concentrava sul supporto per il linguaggio di programmazione Ruby. Nel corso degli anni, tuttavia, la piattaforma si è evoluta e ora supporta una vasta gamma di linguaggi di programmazione e di strumenti. Heroku semplifica il processo di deployment e gestione delle applicazioni web, consentendo ai team di sviluppo di concentrarsi sulla scrittura di codice e sulla realizzazione di funzionalità, anziché sulla gestione dell'infrastruttura. Heroku si occupa di molte attività di sistema, come la configurazione del server, il bilanciamento

del carico, la gestione della scalabilità (verticale e orizzontale) e la gestione delle risorse, in modo da consentire ai team di concentrarsi sullo sviluppo dell'applicazione stessa.

Heroku è basata su un modello di pricing "pay-as-you-go", in cui il costo è proporzionale alle risorse utilizzate, rendendo la piattaforma accessibile anche alle piccole startup e ai team di sviluppo. Inoltre, Heroku offre un'ampia gamma di servizi aggiuntivi, come database gestiti, caching, email, autenticazione e autorizzazione, che semplificano la creazione di applicazioni web complete e funzionali.

È interessante notare che Copado non memorizza i dati su Heroku, ma i dynos (unità di calcolo e di esecuzione di un'applicazione sul servizio di hosting di una piattaforma cloud come Heroku) di Heroku sono creati all'occorrenza per eseguire operazioni sui metadati. Le informazioni su tali metadati vengono poi archiviate in Salesforce.

Ciclo di vita di una User Story

A partire dalla Pipeline mostrata in **Figura 5**, che descrive la Org strategy e la branching strategy di un progetto di test per Copado, simuliamo il ciclo di vita di un task di sviluppo, che in Copado prende il nome di “User Story”.

Per prima cosa, una User Story va creata:

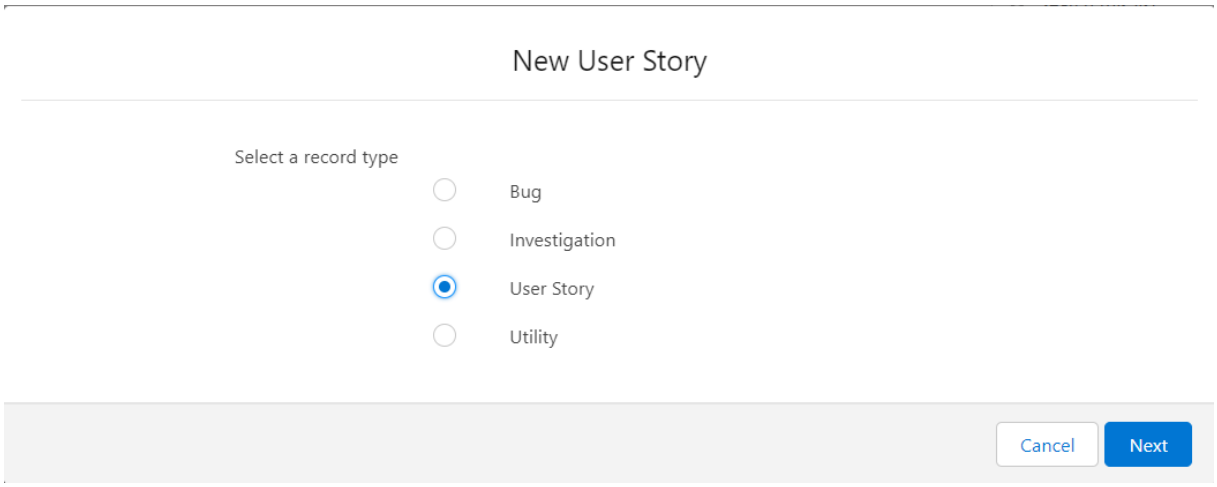

Figura 7. *Creazione di una nuova User Story su Copado.*

Una User Story è a tutti gli effetti un SObject, e come tale ha a disposizione un certo numero di Record Type.
Una User Story di **Bug** serve a tracciare bug interni creati dai team di sviluppo, QA, o di Customer Support.
Una User Story di tipo **Investigation** non permette di agire sul repository remoto e quindi non è utilizzabile per tracciare sviluppi veri e propri. È quindi usata per pura investigazione.
Una User Story di tipo standard (Record Type **User Story**) traccia sviluppi di nuove funzionalità e viene usata anche per debiti tecnici e lavori di refactoring del codice.
Infine, una User Story di tipo **Utility** indica un bundle di stories che possono essere promosse e deployate insieme.
Procediamo selezionando il Record Type *User Story* e premendo *Next*. Verrà visualizzata una finestra modale nella quale inserire le informazioni relative al task e si potrà:

1. Inserire un **titolo** per il task. Questo dovrebbe includere una brevissima descrizione dei requisiti del task.
2. Selezionare un **progetto**. Se si seleziona uno sprint e non un progetto, Copado imposterà il campo di lookup “Project” al progetto referenziato nello sprint selezionato.
3. Selezionare la Release (ovvero una finestra di rilascio per un blocco di sviluppo correlati tra loro) in cui la User Story sarà inclusa.
4. Selezionare uno sprint.
5. Selezionare la Credential dell’ambiente di sviluppo dove si trovano le modifiche che si vogliono committare. Quando si salva la User Story, si può navigare con semplicità alla Org per fare review delle modifiche cliccando il bottone “Open Org”, disponibile solo nei Record Type Bug e User Story.
6. L’ambiente di sviluppo si auto-popolerà al selezionamento della Credential.
7. Se la checkbox “Ready to Promote” viene impostata a true, la User Story sarà idonea alla promozione.
8. Se la checkbox “Promote and Deploy” viene impostata a true, verrà eseguito immediatamente una promozione per la User Story corrente, al salvataggio.

La sezione *User Story Definition* (**Figura 8**) va compilata con informazioni di rilievo riguardo cosa si vuole raggiungere con questo sviluppo:

1. **As a**: Tipo di utenza o ruolo. La persona per cui si sta creando la User Story.
2. **Want to**: Goal (cosa vuole raggiungere la persona specificata nel campo precedente).
3. **So that**: Ragione (qual è il beneficio della feature richiesta da questa persona).

Inoltre, oltre a selezionare quale sviluppatore lavorerà su questa User Story e quale business analyst la revisionerà operativamente, si possono aggiungere specifiche funzionali dettagliate riguardo a come lo sviluppo funzionerà e informazioni tecniche con dettagli specifici riguardo a cosa si sta costruendo e quali sono le idee di implementazione.

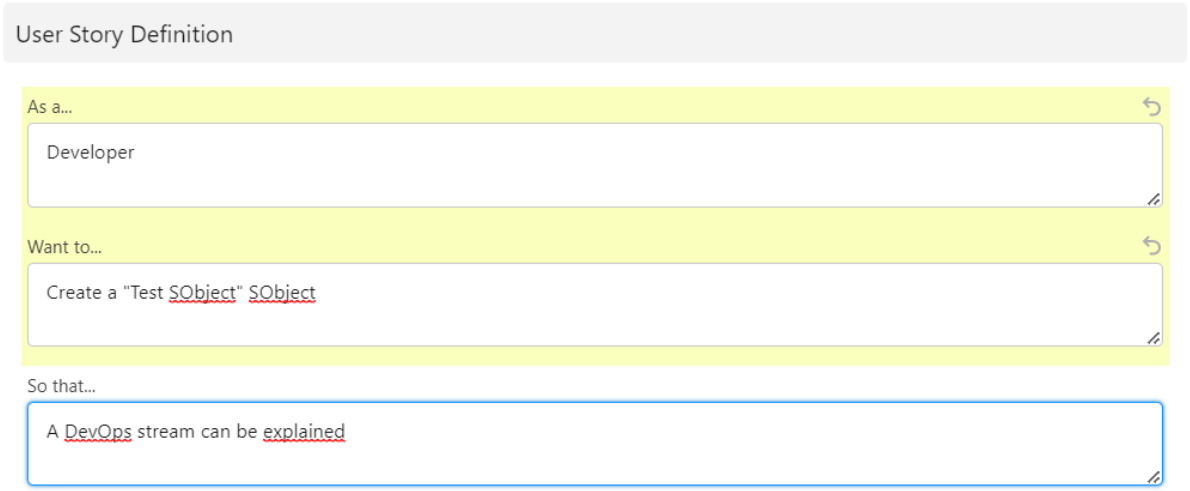

Figura 8. *Sezione User Story Definition.*

Nella sezione *Information* (**Figura 9**) invece, vengono inserite le informazioni relative all'ambiente in cui la funzionalità verrà sviluppata, in maniera tale che Copado possa autonomamente tracciare le modifiche su di esso.

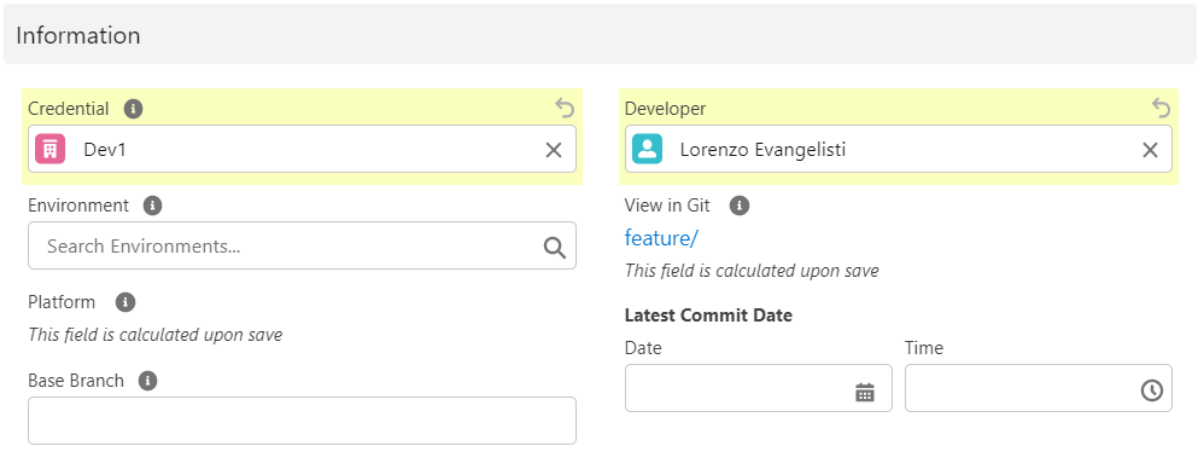

Figura 9. *Sezione Information.*

Struttura di una User Story

Al salvataggio della User Story, Copado effettua un reindirizzamento alla pagina di layout del task.

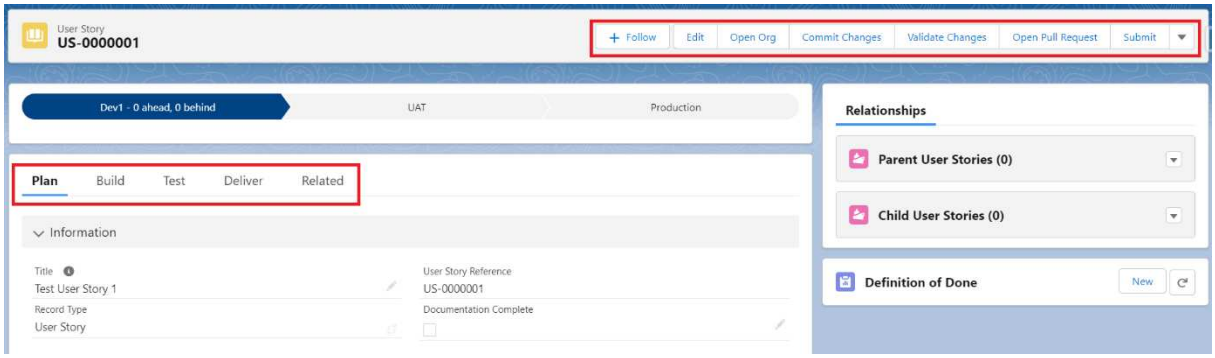

Figura 10. *Layout di una User Story.*

Come si può vedere in **Figura 10**, la User Story di Copado è suddivisa in 5 sezioni, ed ogni sezione è destinata a determinati profili:

- **Plan**: Scrum Master e Product Manager. Questa sezione è rilevante anche per i team di sviluppo e di QA.
- **Build**: Sviluppatori
- **Test**: Quality Assurance
- **Deliver**: Sviluppatori e Release Manager
- **Related**: Applicabile a tutti i profili.

La sezione **Plan** contiene tutte le informazioni generali della User Story che sono state definite alla creazione di essa.
La sezione **Build** contiene invece informazioni relative all'ambiente di sviluppo ed al branch su cui le modifiche verranno versionate. In altre parole, è relativa alla Org strategy ed alla branching strategy.
La sezione **Test** rende possibile selezionare le automazioni con cui testare lo sviluppo a cui fa riferimento la User Story.
La sezione **Deliver** permette di gestire il rilascio delle modifiche in ambiente di test e poi di produzione.
La sezione **Related** mostra la cronologia dei commit relativi allo sviluppo a cui la User Story si riferisce e anche delle modifiche effettuate direttamente sui campi della stessa.

La User Story di Copado è al centro del flusso di sviluppo, poiché permette una panoramica completa di tutto ciò che riguarda la feature che si vuole implementare.

Flusso di sviluppo

Nella sandbox di sviluppo vengono effettuate le modifiche incluse nel requisito della User Story. Nel nostro caso specifico effettueremo la creazione di un SObject chiamato *"Test SObject"* (**Figura 11**).

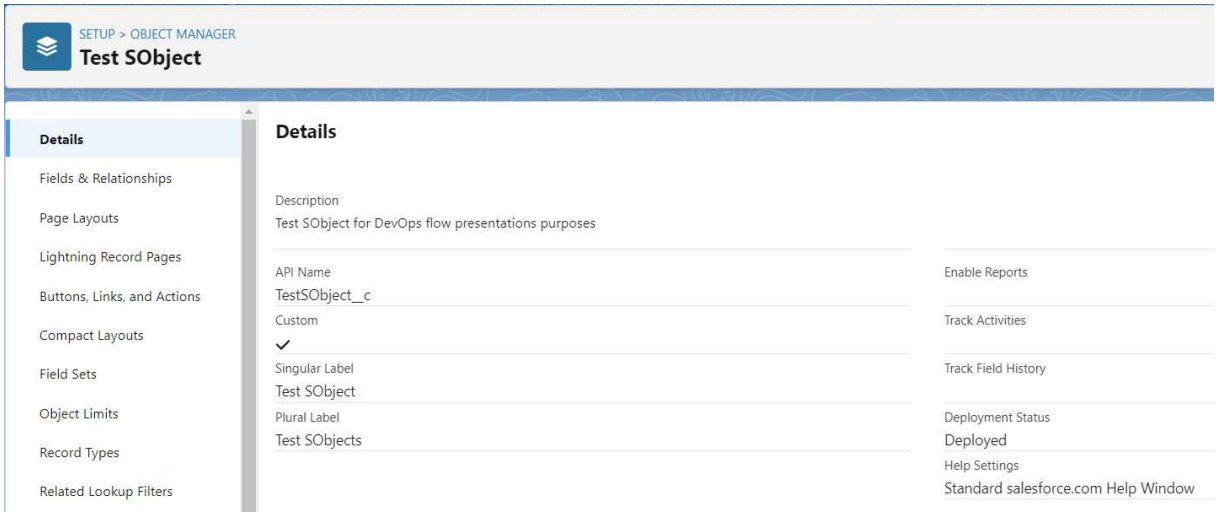

Figura 11. *Il nuovo oggetto Test SObject. Da notare che l'API Name (TestSObject__c dove "__c" sta per "custom") differisce dalla Label, questo perché mentre l'API Name viene utilizzato dal sistema Salesforce per identificare in modo univoco gli oggetti e i campi, la Label viene utilizzata per fornire una descrizione significativa dell'elemento dell'interfaccia utente agli utenti finali.*

Una volta creato l'SObject, lo sviluppatore si sposterà di nuovo sulla Org di produzione all'interno dell'applicazione Copado Release Manager e, cliccando su *Commit Changes* potrà visualizzare tutte le differenze.

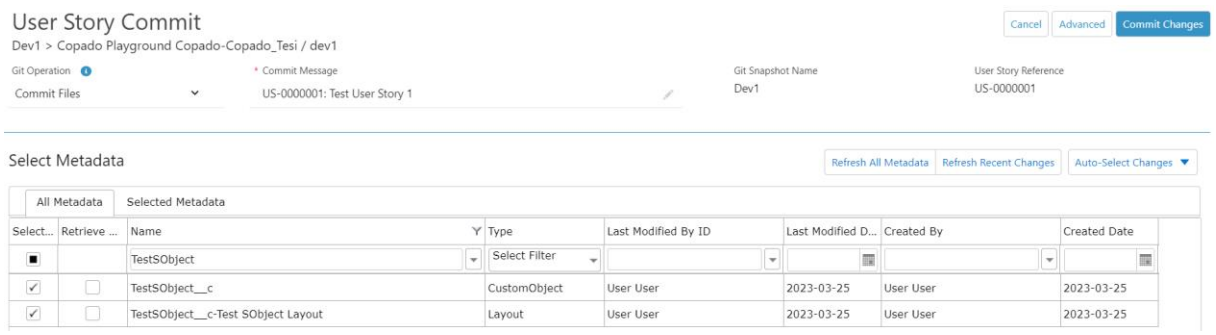

Figura 12. *Fase di commit delle modifiche.*

Selezionati i metadati da deployare verso l'ambiente di destinazione, cliccando di nuovo sul pulsante *Commit Changes,* partirà il processo di automazione di commit che provvederà ad aggiornare il branch di sviluppo con i dati scelti.

La prossima fase è quella dell'effettivo deploy sull'ambiente di test. Ciò viene effettuato visitando la sezione ***Build*** della User Story per una revisione dei metadati da deployare. Se si ritiene che la User Story sia pronta per passare allo step successivo si preme prima su *Validate Changes* per effettuare la validazione automatica dei metadati sull'ambiente di destinazione. In caso di fallimento di questa, bisognerà risolvere eventuali errori di validazione dovuti a dei disallineamenti o ad una malformazione dei metadati. In caso di successo, al contrario, premendo *Open Pull Request* si verrà reindirizzati alla pagina del repository (**Figura 13**) dove si potrà richiedere il merge del branch su cui sono presenti gli sviluppi verso il branch di destinazione (branch di test, in questo caso specifico).

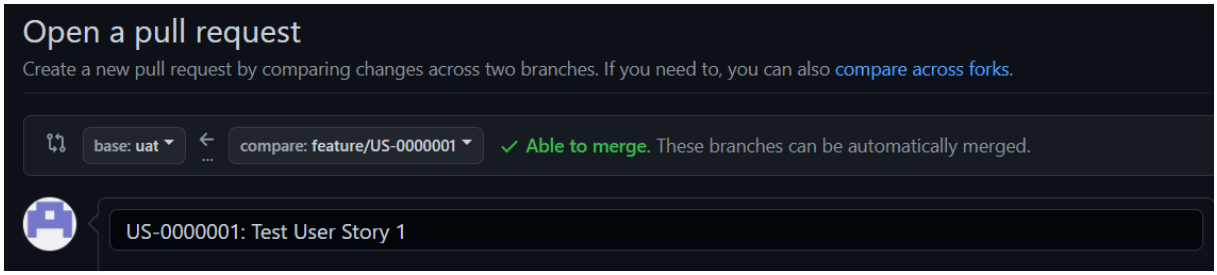

Figura 13. *Apertura di una Pull Request verso UAT.*

Dopo l'apertura della Pull Request, in caso di assenza di conflitti di merge, chi di competenza (generalmente una figura chiamata Release Manager) la approverà. A questo punto, tornando sulla pagina principale della User Story, premendo su *Submit* l'utente sarà reindirizzato alla pagina in **Figura 14**. Premendo poi su *Submit Changes*, Copado si prenderà cura di deployare le modifiche sull'ambiente di test.

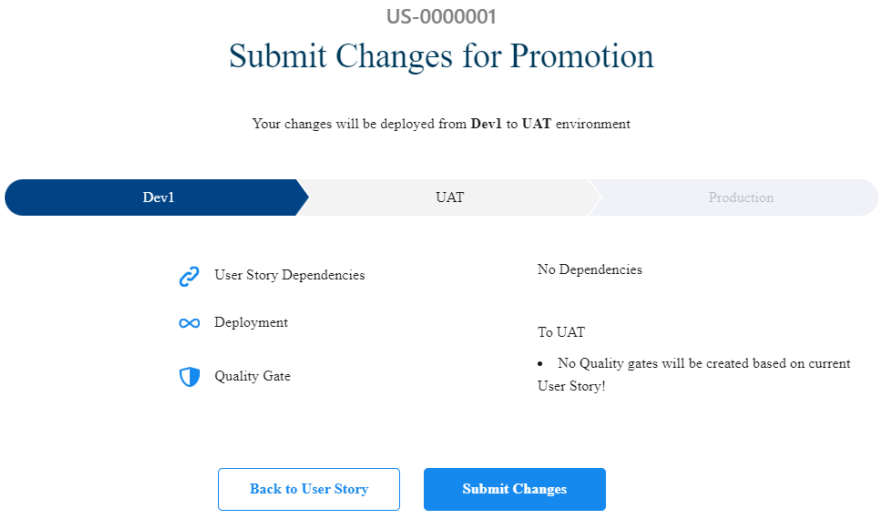

Figura 14. *Promozione di una User Story in ambiente di UAT.*

Si può monitorare lo stato della promozione nell'ambiente di destinazione navigando nel record di promozione presente nella sezione *Related* della User Story, in maniera tale da cogliere eventuali problemi di deploy sul nascere. Concludendo, la stessa procedura va effettuata di nuovo, stavolta per portare la User Story in ambiente di produzione.

Abbiamo visto come, usando una semplice interfaccia utente point-and-click, una User Story, ovvero un intero sviluppo di funzionalità, esegue un vero e proprio cammino dall'ambiente di sviluppo fino a quello di produzione. Tutto questo senza la benché minima percezione di uno strato sottostante di codice. La potenza di Copado però, non risiede solo nella creazione di questa "maschera" che permette che il flusso di Continuous Delivery sia fruibile da letteralmente chiunque.

Tra le funzionalità che non sono state affrontate ci sono:

1. Selezione dei test automatici da eseguire in fase di validazione
2. Ambienti di collaborazione interni all'applicazione stessa per i team di sviluppo
3. Monitoraggio dettagliato di ogni step del flusso di sviluppo
4. Funzionalità di audit e reportistica avanzate per mantenere il controllo sulle attività di sviluppo e di gestione delle applicazioni

In conclusione, è possibile schematizzare i punti di forza e di debolezza di Copado in questa maniera:

Rilascio: 2013

Architettura: Salesforce con Heroku come motore di elaborazione

Benefici:

- Ottimo selezionatore di metadati.
- Una buona scelta per coloro che utilizzano già Salesforce per gestire gli sviluppi di Salesforce.
- Ricca suite di strumenti, tra cui ALM, migrazione di dati e test automatizzati con Selenium (un framework di automazione di test funzionali per applicazioni web, che simula le azioni che un utente effettuerebbe dall'interfaccia grafica dell'applicazione).

Svantaggi:

- Tutti i log e gli altri file sono archiviati come allegati nel pacchetto Salesforce, e ciò li rende difficilmente leggibili.
- L'interfaccia utente è interamente costruita su Salesforce ed ha un aspetto un po' goffo. Ad esempio, le notifiche sui risultati dei processi (promozione, deploy) non sono molto evidenti.
- I lavori vengono eseguiti su Heroku ma non memorizzati su Heroku. Ciò significa che ogni lavoro richiede un tempo non banale per essere avviato (come, per esempio, la clonazione del repository).

CAPITOLO 5

Salesforce DevOps Center

DevOps Center è una grande novità tra le soluzioni di gestione del ciclo di vita del software. Purtroppo, sebbene le pretese siano ampie, ha bisogno di maturare ancora moltissimo per competere con Copado: vediamo perché.

Salesforce DevOps Center è un'applicazione centralizzata che consente ai team di gestire tutti i progetti, le modifiche e le release di Salesforce. È una soluzione gratuita per la gestione dei rilasci di Salesforce, progettata **per sostituire i change set** (un meccanismo di rilascio in graduale disuso che consiste in pacchetti di personalizzazione e configurazione delle Org atti ad essere trasferiti da una Org all'altra) e per riunire sviluppatori pro-code, amministratori low-code, tester e stakeholder aziendali. **DevOps Center è ora in open beta.**

Come Copado, DevOps Center offre funzionalità avanzate come il supporto per la gestione del codice sorgente attraverso una sincronizzazione con il sistema di Version Control, la compilazione automatizzata, l'automazione dei test, la Continuous Delivery e il monitoraggio delle applicazioni.

Al contrario di Copado invece, non si appoggia a nessun sistema di elaborazione esterno. DevOps Center è infatti completamente integrata in

Salesforce in quanto applicazione nativa, per cui l'elaborazione avviene in ambiente di sistema.

Ciclo di vita di un Work Item

Come fatto per le User Story di Copado, approfondiamo il ciclo di vita del task di sviluppo di Salesforce DevOps Center: il **Work Item.**
Stavolta la pipeline di riferimento sarà quella mostrata in **Figura 6,** che descrive la Org strategy e la branching strategy di un progetto di test per DevOps Center. In quanto nella strategia utilizzata l'ambiente di sviluppo è unico, alla creazione del Work Item non sarà richiesto selezionarlo.

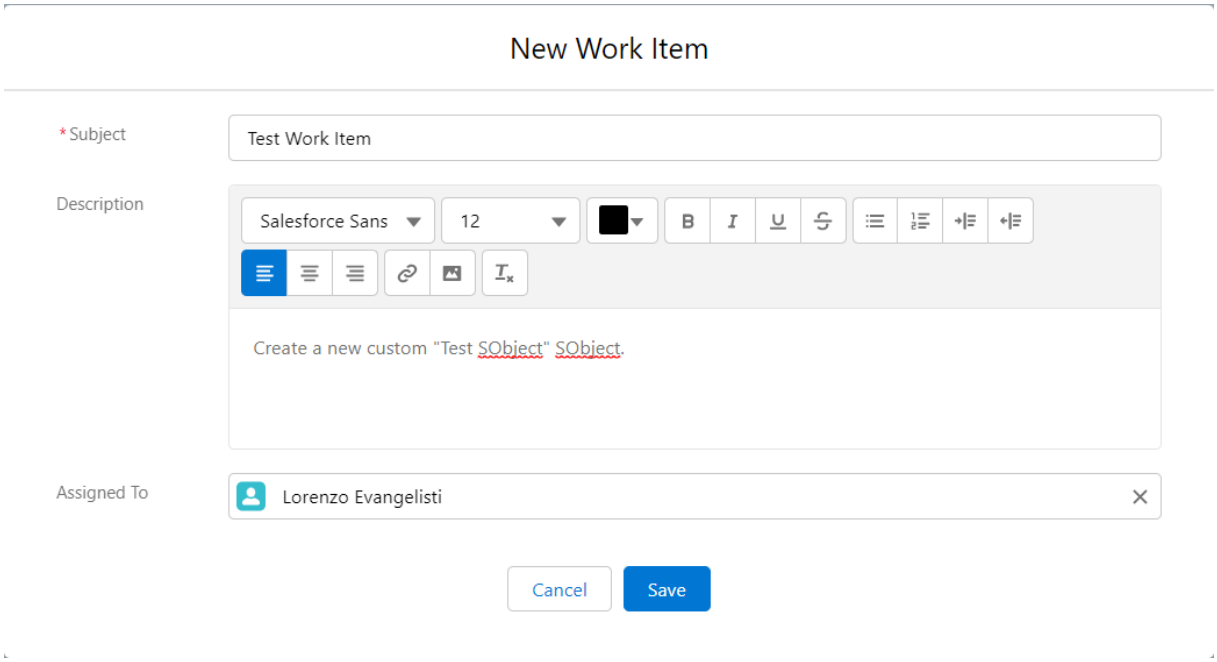

Figura 15. *Creazione di un Work Item.*

A differenza della User Story di Copado, il Work Item può essere popolato con pochissime informazioni: titolo, descrizione e utente a cui assegnarlo.

Struttura di un Work Item

Al salvataggio, il Work Item (e il relativo branch, figlio del branch specchio dell'ambiente di test) viene creato e bisogna visitarlo nella pagina dedicata.

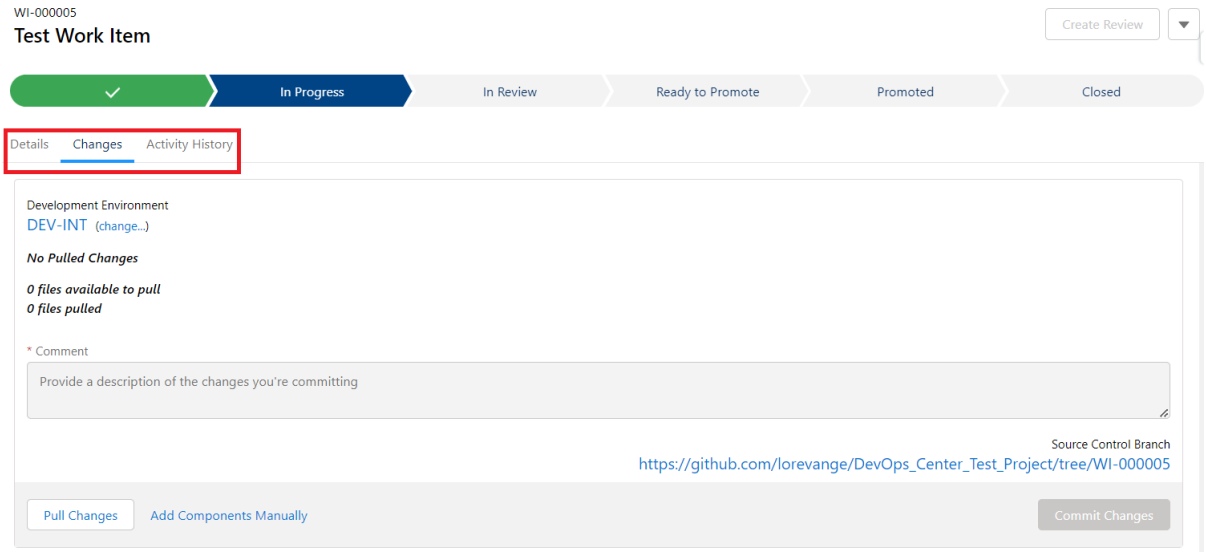

Figura 16. *Layout di un Work Item.*

La pagina dispone di tre sezioni: Details, Changes e Activity History.

La sezione **Details** presenta i tre campi che sono stati popolati alla creazione del Work Item, ovvero titolo, descrizione e utente a cui il Work Item è stato assegnato.
La sezione **Changes** mostra l'ambiente di sviluppo sul quale si stanno eseguendo le modifiche relative al task e le eventuali differenze con l'ambiente di test recuperate da quello di sviluppo.
La sezione **Activity History** presenta la cronologia delle azioni effettuate sul Work Item.

Flusso di sviluppo

Nella sandbox di sviluppo vengono effettuate le modifiche incluse nel requisito del Work Item. Di nuovo, nel nostro caso specifico effettueremo la creazione di un SObject chiamato *"Test SObject"* (ancora uguale a quello già mostrato in **Figura 11**).

A questo punto, tornando nell'applicazione DevOps Center disponibile nell'ambiente di produzione, all'interno della sezione *Changes* del Work Item creato, l'utente può premere su *Pull Changes.* Quest'azione recupererà tutte le differenze (quindi aggiunte, modifiche o cancellazione di metadati) tra l'ambiente di test e quello di sviluppo. In **Figura 17** vediamo come il sistema ha rilevato la creazione dell'oggetto e del suo layout standard, oltre a delle modifiche su tre profili (sicuramente aggiunte alla loro *Field-level-security*, ovvero la visibilità che questi profili hanno sull'oggetto appena creato) che non deployeremo.
Generalmente, per una questione di sicurezza, essendo i Profile degli SObject molto delicati, essi vengono aggiornati nell'ambiente di destinazione attraverso una *Manual Procedure*. Così facendo, si aggiunge un intervento umano atto ad evitare che il profilo sbagliato abbia o meno visibilità su determinati metadati.

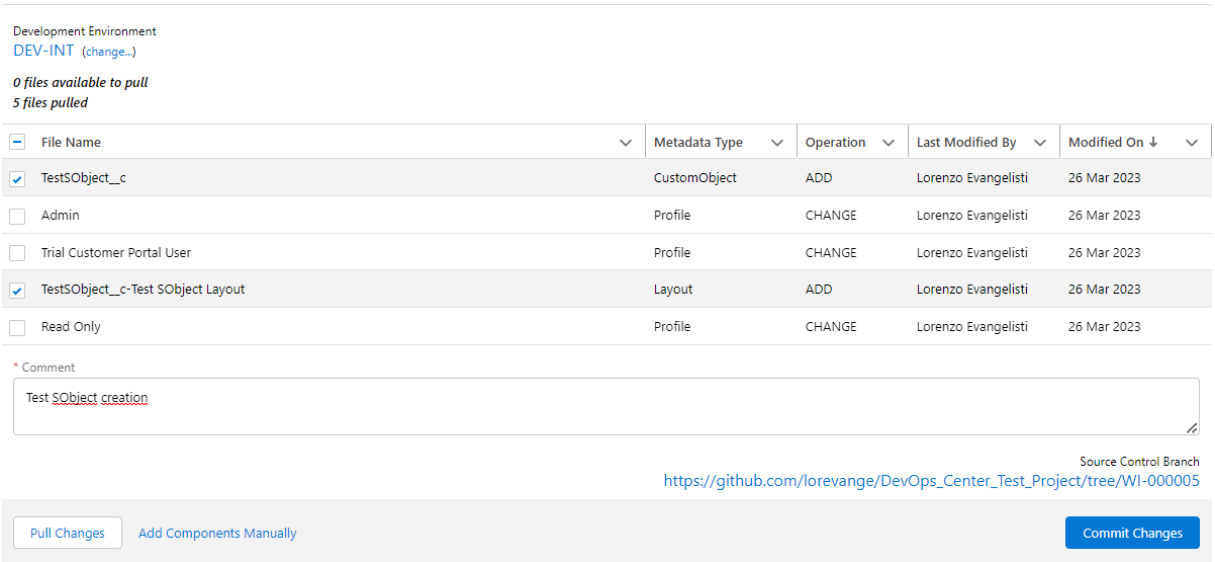

Figura 17. *Risultato della Pull Changes sul Work Item.*

Cliccando su *Commit Changes,* DevOps Center effettuerà il commit dei metadati selezionati sul branch automaticamente creato in fase di creazione del Work Item.

A questo punto, per far passare il Work Item nello stato di *Review,* va premuto il bottone *Create Review* in alto a destra, in maniera tale che un revisore (di solito un membro del team di Release Management) possa approvarlo per una eventuale promozione in ambiente di test.

Fatto ciò, verrà creata una *Change Request* sul repository (equivalente di una *Pull Request*) che il revisore potrà visionare usando il pulsante *View Change Request* dedicato (**Figura 18**). Se tutti i metadati risultano corretti, il revisore può procedere a premere l'interruttore *Ready to Promote,* sempre presente all'interno della pagina di layout del Work Item. Quindi Salesforce DevOps Center aggiornerà automaticamente la sezione *Stages* della pagina della Pipeline con il Work Item pronto per essere promosso in ambiente di test (figura 19).

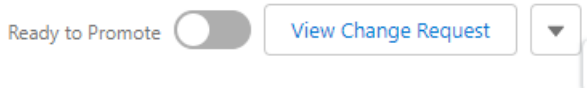

Figura 18. *Interruttore Ready to Promote e bottone View Change Request.*

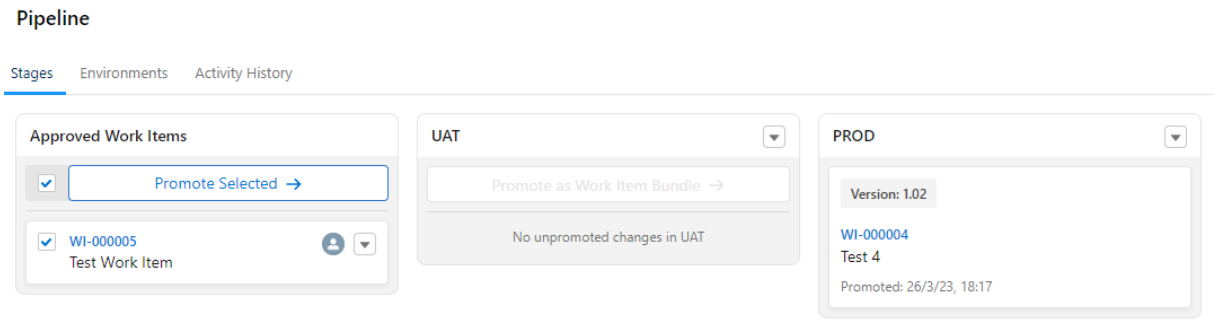

Figura 19. *Pipeline aggiornata successivamente al passaggio del Work Item nello stato Ready to Promote.*

In **Approved Work Items** sono presenti tutti i Work Item che hanno avuto avanzamenti (quindi commit) nelle sandbox di sviluppo e che sono passati allo stato *Ready to Promote.*

A questo punto, alla pressione di *Promote Selected*, verrà mostrata una finestra dove selezionare la metodologia di testing automatico da utilizzare. Successivamente, tutti i Work Item selezionati, a meno di errori a runtime, verranno **contemporaneamente:**

1. Inseriti nel branch di test UAT.
2. Deployati nell'ambiente di test.

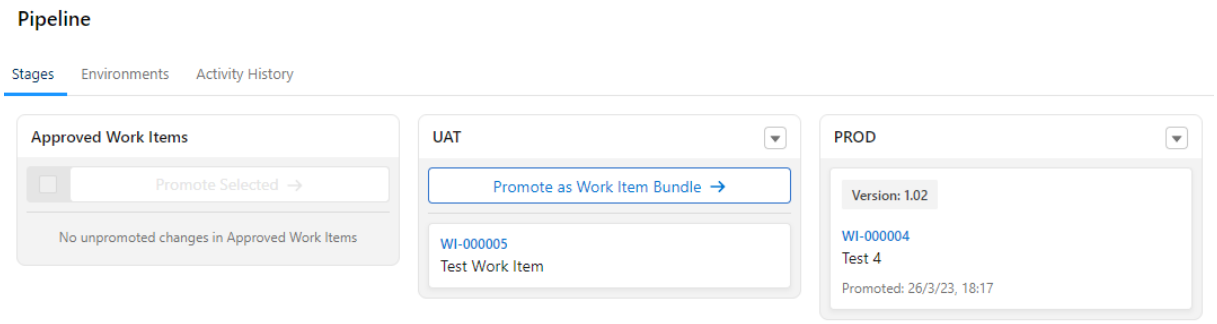

Figura 20. *Stato della Pipeline successivamente alla promozione del Work Item in UAT.*

Ora, contrariamente a quanto accade con Copado, la promozione dei Work Item da UAT a Produzione avverrebbe in maniera estremamente più semplice, se solo l'effort da applicare per testare tutti i Work Item presenti in UAT non fosse enorme. Infatti, DevOps Center non permette una gestione separata dei Work Item presenti in UAT. Ciò significa che tutti i Work Item presenti in UAT **devono essere promossi insieme verso l'ambiente di produzione**. Questa gestione implica un coordinamento tra il team di Release Management e tutte le persone che effettuano i test in UAT. Personalmente, credo che questa modalità di promozione verso Produzione sia eccessivamente pesante e che rappresenti un collo di bottiglia nella gestione del flusso di sviluppo di più Work Item.

Il processo di promozione in produzione però non richiede l'apertura e l'approvazione di una Pull Request: questo avviene automaticamente grazie a DevOps Center, che si occupa di tutto in maniera autonoma. Ciò nonostante, l'utente che effettua la promozione ha la possibilità di selezionare la metodologia di testing automatico da utilizzare, esattamente come avviene per la promozione in UAT (**Figura 21**).

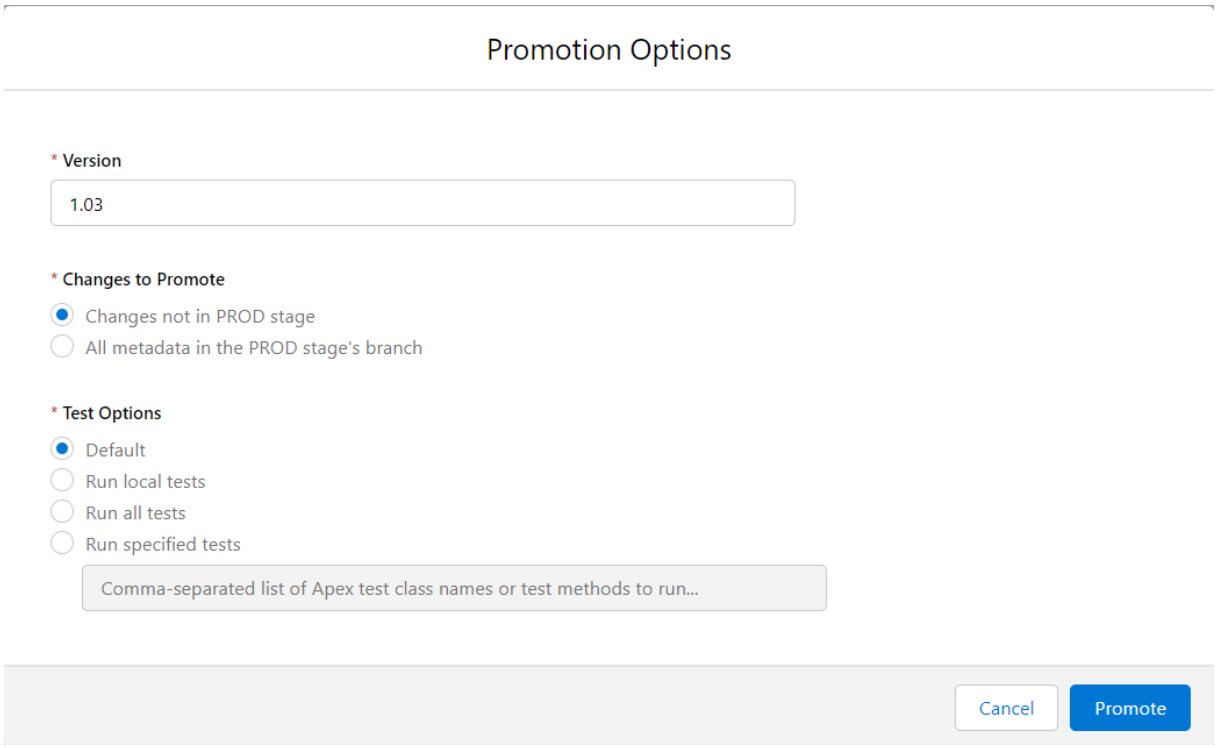

Figura 21. *Selezione della metodologia di testing automatico in fase di promozione in produzione di un bundle di Work Item.*

Successivamente, se tutti i test eseguiti vengono superati con successo, il bundle di Work Item viene deployato nella Org di produzione. La pipieline avrà l'aspetto mostrato in **Figura 22**.

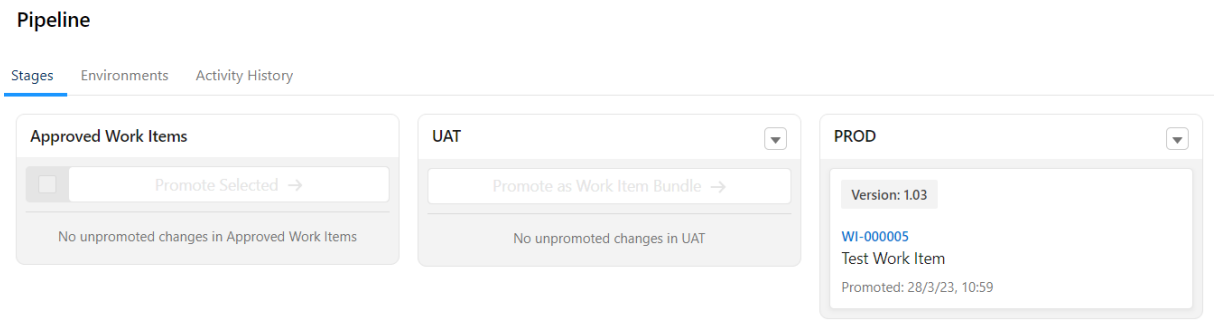

Figura 22. *Stato della Pipeline successivamente alla promozione del bundle di Work Item in produzione.*

Si conclude così il ciclo di vita di un Work Item. In ordine, è passato dalla creazione allo sviluppo, poi è stato sottomesso a revisione da parte del team di Release Management, che ha promosso il Work Item in ambiente di test. Effettuate le verifiche e i test in UAT, il Work Item è stato portato in produzione in un bundle, insieme a tutti gli altri Work Item in ambiente di test al momento della promozione.

Salesforce DevOps Center è un tool di gestione del flusso di sviluppo molto semplice ed intuitivo. Elimina del tutto l'utilizzo della linea di comando e gestisce del tutto autonomamente il Version Control. Insomma, è un'applicazione destinata agli admin che non fanno uso di codice, ma dedicano la loro totale attenzione ai metadati ed alla loro configurazione a sistema.

Come è stato fatto per Copado, è possibile schematizzare i punti di forza e di debolezza di DevOps Center in questo modo:

Rilascio: 2022 (open beta)

Architettura: Applicazione nativa di Salesforce

Benefici:

- Ottimo selezionatore di metadati.
- Una ottima scelta per coloro che iniziano ad approcciarsi a Salesforce.
- Ottimo per gli admin, in quanto estremamente low-code.

Svantaggi:

- Nessun tipo di logging.
- Promozione da UAT a Produzione limitante.

CAPITOLO 6

Conclusioni

Copado viagga ancora su binari ad alta velocità rispetto a DevOps Center. La possibilità di creare suite di test che ripercorrono azioni effettuate sulla UI dall'utente, il dettagliato tracciamento delle modifiche, la maggiore relazione tra i dati e la più articolata gestione delle Org e branching strategy pongono Copado su un piano molto più alto rispetto a DevOps Center.

Nonostante questo, Salesforce DevOps Center è un neonato, una soluzione rilasciata al pubblico nel recente dicembre 2022: può ancora fare passi da gigante. Esiste una comunità di utenti in forte crescita che comunica sulla piattaforma di Partner Community, e ciò sta dando spunti agli sviluppatori per le future implementazioni e i miglioramenti da attuare.

Possiamo con semplicità distinguere il tipo di cliente che farebbe meglio ad utilizzare Copado: compagnie medio/grandi, che possiedono già una struttura Salesforce ben radicata, oltre che un flusso di rilascio dei metadati consolidato. Copado fornirebbe loro la possibilità di sperimentare graduali cambiamenti in maniera molto semplice, e individuare rallentamenti e vincoli in modo più consistente.

DevOps Center può invece essere al momento adottato solo da parte di piccole compagnie. Non è ancora adatto ad essere utilizzato da decine di team di sviluppo contemporaneamente. Dalla sua parte ha il fatto di essere un'applicazione nativa su Salesforce: l'integrazione con esso rende DevOps Center una scelta più solida nel lungo termine.

Mentre le piattaforme di gestione dei flussi di rilascio competono sul mercato, la comunità del DevOps si allarga e abbraccia sempre più individui. In un'epoca in cui lo sviluppo del software è diventato centrale a quasi tutte le compagnie, di conseguenza lo è ancora di più la gestione di esso. È per questo che in sordina si compete, in un ambiente dove sembra che una soluzione possa valere l'altra. Il "come", il più delle volte, è più importante del "cosa". Il "cosa" può essere definito dalla comunità scientifica, dalle ricerche, dalle scoperte; ma può anche essere imposto

da una moda, dalla forza mediatica di individui o compagnie multinazionali, e non sempre ha uno scopo propositivo nell'avanzamento della specie. Il "come" è definito dal DevOps, qualsiasi sia l'ambito, il mercato, l'ambiente. Non necessariamente deve essere applicato allo sviluppo del software, ed è questo che lo rende così versatile e potente. DevOps è il modo in cui la produzione passa in secondo piano: "come lo fai" ha più importanza, e rende il prodotto finale il vero frutto dell'innovazione.

CAPITOLO 7

Bibliografia e Link usati

Andrew Davies, *Mastering Salesforce DevOps: A Practical Guide to Building Trust While Delivering Innovation, 1st Edition (2019)*

Copado Community, *https://success.copado.com/s/*

Documentazione Copado, *https://docs.copado.com/home/en-us/*

Salesforce DevOps HQ, *https://salesforcedevops101.com/*

Get Started with DevOps Center, *https://help.salesforce.com/s/articleView?id=sf.devops_center_get_started.htm&type=5*

2021 State of DevOps Report by Puppet, *https://www.puppet.com/success/resources/state-of-devops-report*

CAPITOLO 8

Ringraziamenti

Dedico questo spazio a ringraziare chi mi ha supportato nel mio percorso universitario:

Alla mia famiglia, che c'è sempre stata, c'è, e sempre ci sarà.

A Paloma, senza la quale non sarei mai dove sono e chi sono adesso.

A Enrico Murru e Antonio Tangaro, che in Engineering mi hanno guidato nel viaggio nel mondo di DevOps.

A tutti i miei amici che mi sopportano.

Ai membri de La Fabbrica del Ghiaccio, che mi permettono di sfogare la mia creatività attraverso la mia più grande passione.