Luca Cabibbo

Architettura dei Sistemi Software

Continuous Delivery

dispensa asw650

ottobre 2024

How long would it take your organization to deploy a change that involves just one single line of code?

Mary and Tom Poppendieck

- Riferimenti

- Luca Cabibbo. **Architettura del Software**: Strutture e Qualità. Edizioni Efesto, 2021.
  - Capitolo 38, Continuous Delivery
- Humble, J. and Farley, D. **Continuous Delivery**: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, 2010.
- Bass, L., Weber, I., and Zhu, L. **DevOps**: A Software Architect's Perspective. Addison-Wesley, 2015.
- Nygard, M. **Release It!** Design and Deploy Production-Ready Software, second edition. Pragmatic Bookshelf, 2018.
- Kim, G., Humble, J., Debois, P., and Willis, J. **The DevOps Handbook**: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. IT Revolution, 2016.

- Obiettivi e argomenti

- Obiettivi
  - introdurre la Continuous Delivery
  - presentare la Deployment Pipeline
  - presentare alcune strategie di deployment e rilascio
  - discutere principi e pratiche associate al rilascio del software
- Argomenti
  - introduzione
  - continuous delivery
  - deployment pipeline
  - strategie di deployment e rilascio
  - ulteriori aspetti e pratiche
  - strumenti
  - discussione

* Introduzione

- Per ricordare
  - la *delivery* (*rilascio*, *consegna* o *distribuzione*) di un sistema software (o di un servizio software) è l'attività di rilascio e consegna del software (o di una sua nuova versione o incremento) ai suoi utenti finali
  - *DevOps* è un insieme di pratiche che hanno lo scopo di ridurre il tempo tra quando viene effettuato il commit di un cambiamento di un sistema software e quando il cambiamento viene effettivamente rilasciato in produzione, garantendo allo stesso tempo un'alta qualità
  - la *Continuous Delivery* è la principale pratica tecnica DevOps, per effettuare la delivery del software in modo automatizzato

* Continuous Delivery

- La **Continuous Delivery** (**CD**) è un insieme di principi e pratiche per ridurre costi, tempi e rischi del rilascio di versioni incrementali del software ai suoi utenti
  - è la principale pratica tecnica DevOps
  - si concentra sulle attività della delivery del software – dal commit di un cambiamento nel software a quando gli utenti possono effettivamente utilizzare il software modificato
  - per fare in modo che il sistema software possa essere rilasciato nell'ambiente di produzione in ogni momento – ovvero, in modo "continuo"
  - gli obiettivi complessivi sono
    - aumentare il valore di business del software
    - ridurre i rischi associati ai rilasci

Importanza della delivery

- La delivery del software
  - è un'attività importante
    - perché lo sviluppo di nuovo software non produce nessun beneficio e nessun valore fino a quando il software non è stato rilasciato nelle mani degli utenti
    - perché può sostenere l'innovazione
  - è un'attività rischiosa
    - perché il rilascio di una nuova versione del software costituisce uno dei passi più delicati nel ciclo di vita dello sviluppo del software

Processo di delivery

- La delivery del software è un processo complesso – che richiede lo svolgimento di numerosi passi e azioni
  - quattro categorie principali di attività
    - *build* (*costruzione*) – la compilazione e l'assemblaggio del codice in un formato adatto per l'installazione
    - *deployment* (*installazione*) – l'installazione del software in un ambiente di esecuzione – può richiedere anche il *provisioning* (*preparazione*) dell'ambiente
    - *test* (*verifica*) – l'esecuzione di test per verificare le funzionalità e le qualità del software
    - *release* (*rilascio*) – l'effettivo rilascio del software agli utenti, nell'ambiente di produzione

Note terminologiche

- Ecco la terminologia che utilizziamo – nella pratica, alcuni di questi termini vengono però utilizzati anche con significati differenti
  - una *delivery* è un rilascio nell'ambiente di produzione – che talvolta viene chiamato un "deployment"
  - un *deployment* è l'installazione del software in un ambiente di esecuzione – che talvolta viene chiamata una "delivery"
  - *release* è un verbo e indica un'azione – ma talvolta "release" viene usato (come nome) per indicare una versione di un'applicazione
  - la *release* è l'ultima attività di una *delivery* – ma talvolta "release" viene usato per indicare l'intera "delivery"

Importanza del processo di delivery

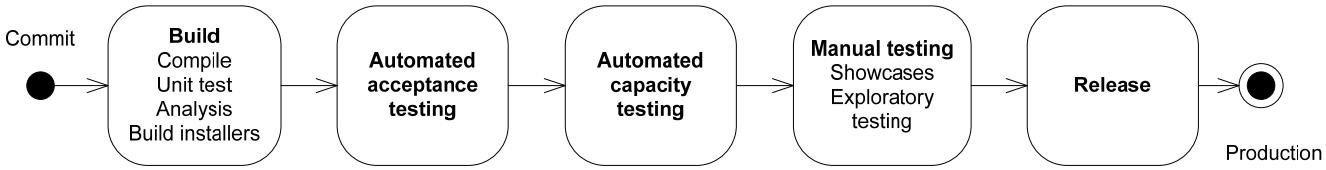
- La delivery del software può essere effettuata in modi diversi
  - alcune organizzazioni la effettuano in modo manuale
    - è un processo lungo, complesso, soggetto a errori e stressante
    - per questo, i rilasci vengono effettuati poco frequentemente
  - altre organizzazioni adottano invece la continuous delivery – ed effettuano la delivery in modo automatizzato, rapido, frequente e affidabile
    - questo consente alle organizzazioni di ottenere un vantaggio competitivo – ad es., per effettuare esperimenti e imparare più velocemente le preferenze degli utenti, e quindi conquistare quote di mercato
    - viene utilizzata una deployment pipeline

* Deployment pipeline

- Una **Deployment Pipeline** consiste di una sequenza di passi (o fasi) da svolgere tra il commit di un cambiamento del software e il rilascio della nuova versione del sistema software
  - è la principale pratica tecnica della CD
  - ha lo scopo di automatizzare l'intero processo di delivery, per muovere il software dal sistema di controllo delle versioni alle mani degli utenti

Esempio di deployment pipeline

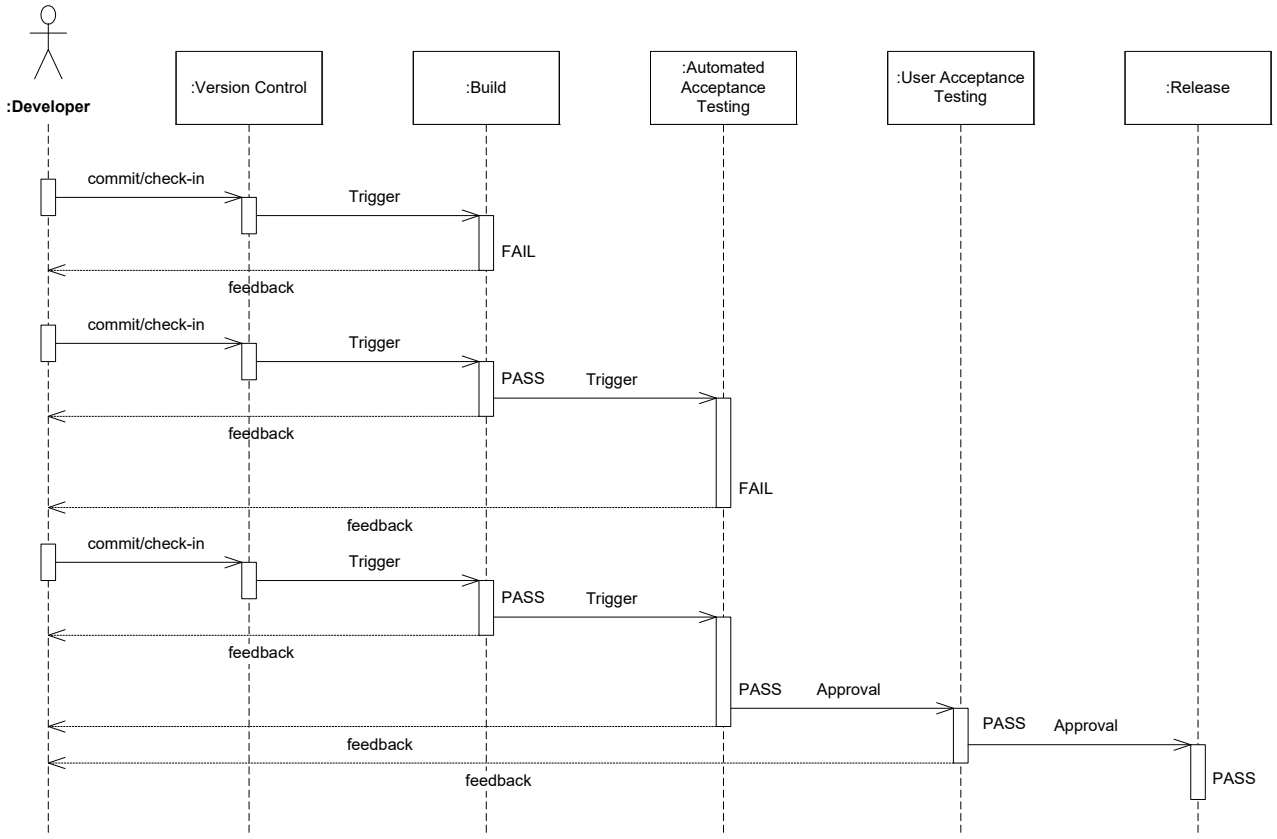
- Ecco un esempio di deployment pipeline



Funzionamento della deployment pipeline

- Funzionamento della deployment pipeline
  - ogni commit di un cambiamento del codice avvia una nuova istanza della pipeline
  - le fasi della pipeline vengono svolte una alla volta e in sequenza
    - il successo di una fase abilita l'esecuzione della fase successiva
    - il fallimento di una fase causa invece l'interruzione dell'istanza della pipeline
  - il completamento di tutti le fasi dell'istanza può portare al rilascio della nuova versione del software nell'ambiente di produzione
  - la pipeline può essere attivata più volte – ogni volta dà luogo a una nuova istanza della pipeline

Deployment pipeline: esempio (è diverso dall'esempio precedente)



Avvio della deployment pipeline

- La pipeline viene avviata quando uno sviluppatore (o team di sviluppo) effettua un commit
  - lo sviluppatore effettua un commit nel sistema di controllo delle versioni quando pensa di aver concluso un proprio compito – ad es., aggiungere o modificare una funzionalità
  - questo avvia automaticamente una nuova istanza della deployment pipeline

Build

- Durante la fase di build vengono in genere svolte diverse attività
  - compilazione e assemblaggio del software – in un formato “binario” ed “eseguibile”, adatto alla distribuzione e all’installazione
  - esecuzione di un ampio insieme di test – test unitari e test di integrazione
  - analisi statica del codice – per fornire un feedback sulla bontà del codice
  - gli elaborati prodotti vengono salvati in un repository condiviso – per poter essere utilizzati nelle fasi successive della pipeline
  - l’ambiente utilizzato è un server di integrazione

Test automatizzati

- Vengono poi eseguiti dei test end-to-end e di accettazione automatizzati, che operano sull’intero sistema, per verificare il software in modo completo
  - questi test vengono eseguiti in uno o più ambienti di test
    - preliminarmente, questo richiede la preparazione di ciascun ambiente di test e il deployment del software in quell’ambiente
  - i test (funzionali e di qualità) possono essere suddivisi su più passi successivi della pipeline
    - ogni ulteriore passo effettua dei test via via più accurati – può richiedere un ambiente di test sempre più simile all’ambiente di produzione – e di solito richiede un tempo sempre maggiore

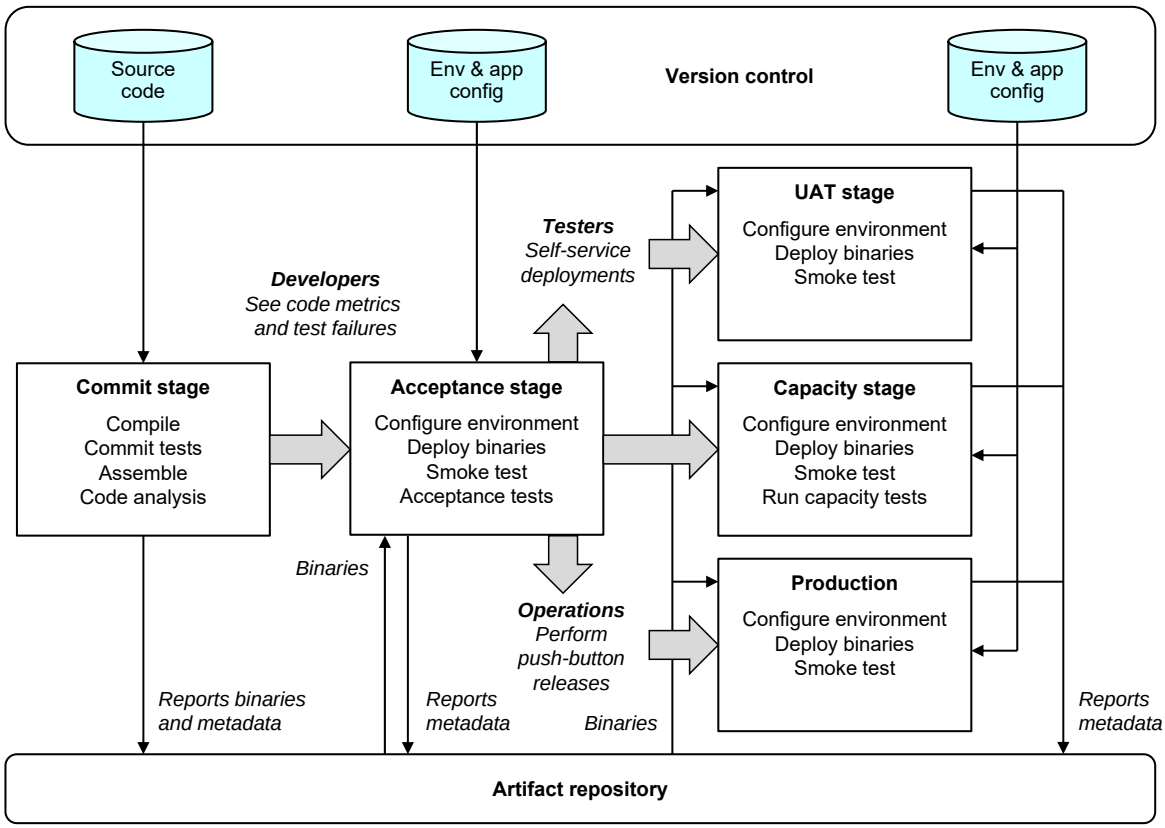
Test manuali

- Talvolta sono necessari anche dei test di accettazione manuali – per verificare ciò che non è stato o non può essere verificato in modo automatizzato
  - i test manuali seguono quelli automatizzati
  - anche questi test richiedono un ambiente di test separato, simile a quello di produzione

Release

- La fase di release ha lo scopo di rilasciare il sistema agli utenti finali, nell'ambiente di produzione
  - non è solo deployment – si conclude quando le richieste degli utenti vengono effettivamente indirizzate alla nuova versione del software
  - anche la release viene eseguita automaticamente – ma in genere deve essere approvata ed avviata "manualmente" da un operatore umano
  - è l'attività più rischiosa della pipeline
    - ad es., si vuole un'interruzione di servizio limitata o nulla, e deve essere possibile un ripristino in caso di problemi
    - le fasi precedenti della deployment pipeline hanno proprio l'obiettivo di minimizzare questi rischi

Architettura di una deployment pipeline



Fasi e ambienti

- I diversi passi della pipeline vengono svolti in ambienti differenti
  - gli sviluppatori operano sui loro computer
  - la build viene di solito eseguita in un server di integrazione
  - la verifica viene effettuata in uno o più ambienti di test
  - il rilascio finale avviene nell'ambiente di produzione
  - la deployment pipeline si deve occupare anche della preparazione degli ambienti di test e di produzione – e non solo del deployment in questi ambienti

Fasi e verifica e feedback

- I passi successivi della deployment pipeline consentono di verificare la bontà del software in modo via via più accurato – e di fornire agli sviluppatori un feedback su di essa
  - gli sviluppatori effettuano compilazioni e test locali
  - durante la build, tutte le parti del sistema vengono integrate e verificate, per garantire che il sistema sia “tecnicamente funzionante”
  - nei diversi passi di test, viene effettuata una verifica sempre più accurata e completa del sistema
  - non viene verificato solo il software – ma anche il processo di delivery del software e le sue attività
  - i test però richiedono anche un tempo via via maggiore
    - i passi della pipeline vengono organizzati per bilanciare la velocità del feedback rispetto alla confidenza nella “prontezza” del sistema

Su “continuous”

- La parola *continuous* (*continuo*, *ininterrotto*) suggerisce che la deployment pipeline vada avviata spesso – anche più volte al giorno
  - “se è rischioso e può far male, allora fallo più spesso”
  - gli sviluppatori fanno commit frequenti e regolari, relativi a piccoli aggiornamenti
  - le attività della delivery (build, costruzione degli ambienti, deployment e test) vengono eseguite frequentemente
  - in questo modo, viene verificato in modo continuo che il sistema software sia “pronto per il rilascio”
    - non è il rilascio che è continuo, ma la “prontezza al rilascio” – e il feedback sulla “prontezza al rilascio”

- Discussione

- Ora che abbiamo descritto il funzionamento della deployment pipeline, discutiamone gli obiettivi generali, i principi su cui si basa, nonché i benefici che sostiene

Obiettivi

- Obiettivi della deployment pipeline
  - abilitare il rilascio automatizzato, ripetibile e affidabile del software
    - per poter eseguire, su richiesta, il deployment di ogni versione del software in ogni ambiente semplicemente premendo un pulsante
  - fornire feedback utile per identificare e risolvere problemi, nel modo più rapido possibile
  - sostenere una collaborazione stretta e proficua tra tutte le persone coinvolte nel rilascio del software
  - rendere visibili le attività legate al rilascio del software

Principi

- Alcuni principi della delivery del software
  - automatizzare quasi tutto – sia le singole attività che l'intera deployment pipeline
  - se è rischioso e può far male, allora fallo più spesso
  - tenere tutto sotto controllo di versione
  - costruisci la qualità dentro al sistema e al processo – non pensare di poterla verificare dopo
  - "done" (un termine di Scrum) significa "potenzialmente rilasciabile" (o rilasciato)
  - responsabilizza le persone
  - persegui un miglioramento continuo

Vantaggi

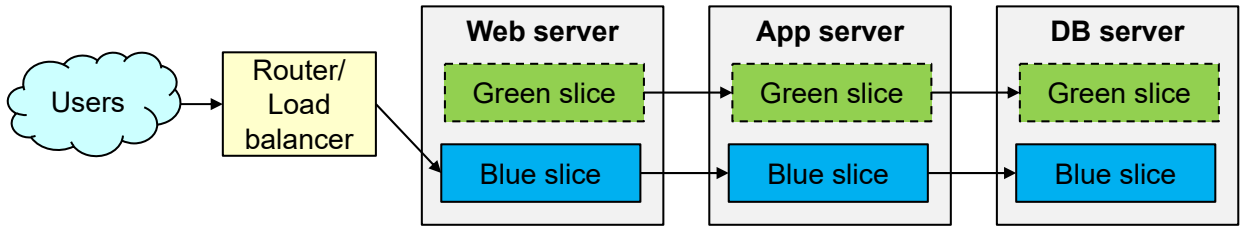
- Benefici sostenuti dalla deployment pipeline
  - l'automazione delle attività sostiene la ripetibilità del delivery
  - rilasci frequenti e riduzione del rischio
  - maggiore affidabilità dei rilasci e riduzione degli errori
  - flessibilità dei rilasci
  - i team sono più autonomi e più responsabili
  - miglioramento continuo del processo di delivery e dei cicli di (sviluppo e) rilascio del software
  - riduzione dello stress

* Strategie di deployment e rilascio

- Presentiamo ora alcune strategie e tecniche per il deployment e il rilascio (lato server) di una nuova versione di un'applicazione o di un servizio software
  - in particolare, discutiamo alcune tecniche per il *rilascio senza interruzione di servizio* (*zero-downtime release*) – in cui la transizione da una versione del software alla successiva avviene, per gli utenti, in modo istantaneo
  - queste tecniche supportano anche il ripristino (rollback) di una versione precedente del software
  - si basano in genere su un disaccoppiamento del deployment dalla release

- Blue-green deployment

- *Blue-green deployment*
  - ci sono due versioni dell'ambiente di produzione – "blu" e "verde" – i cui nodi ospitano l'applicazione o il servizio da aggiornare
    - la versione corrente V_N è in esecuzione nell'ambiente verde
    - la nuova versione V_{N+1} viene installata nell'ambiente blu
  - quando il deployment di V_{N+1} è completato, le richieste dei client vengono indirizzate verso l'ambiente blu



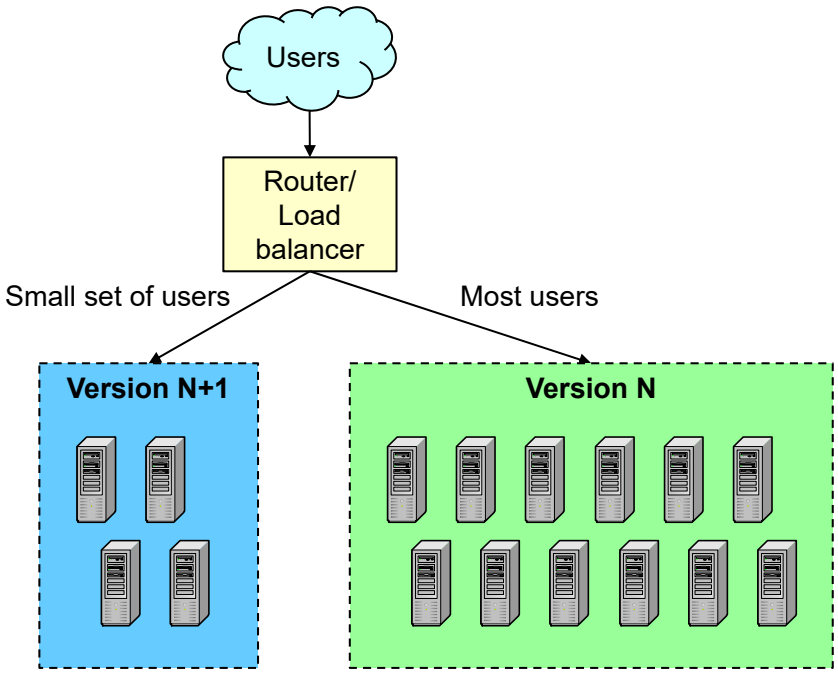
Blue-green deployment

- Conseguenze
  - l'interruzione di servizio è minimale o nulla
  - se il rilascio si conclude con successo, l'ambiente verde può essere deallocato
  - durante il rilascio richiede una grande quantità di risorse computazionali

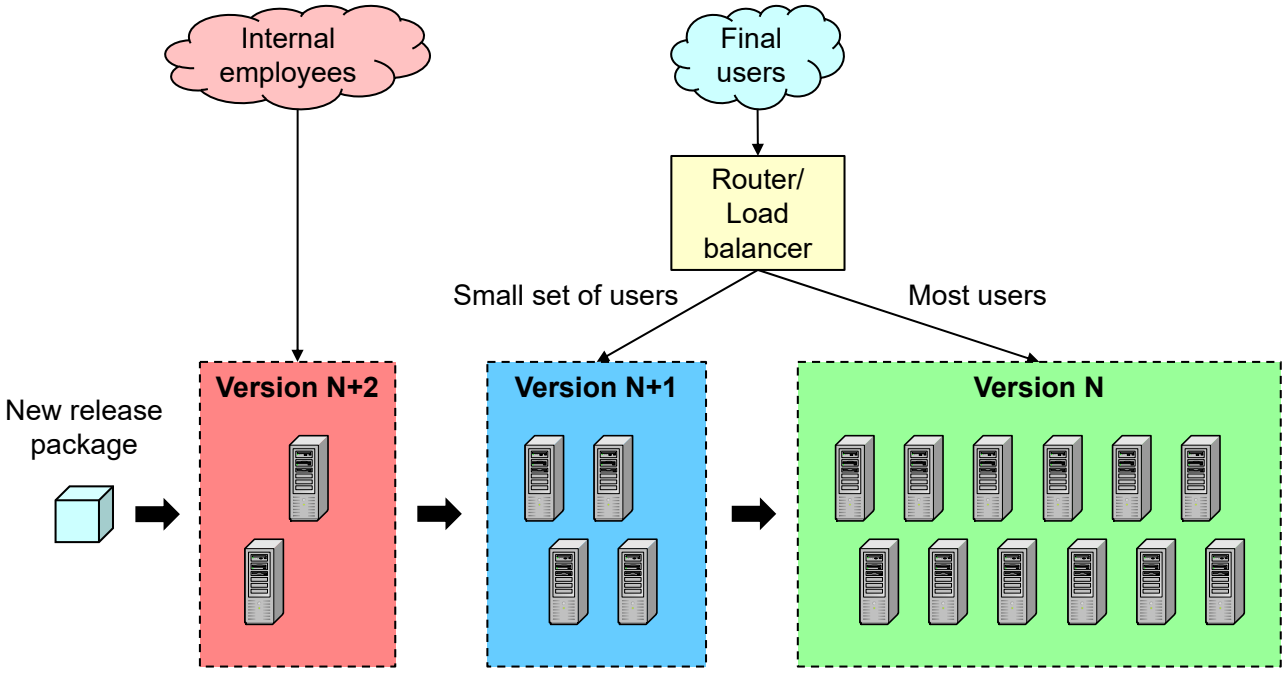
- Canary releasing

- *Canary releasing*
  - utile quando la versione corrente V_N del software è eseguita (in modo replicato) su più nodi
  - la nuova versione V_{N+1} del software viene installata solo su alcuni di questi nodi (o su dei nuovi nodi)
  - inizialmente, solo le richieste di alcuni utenti vengono reindirizzate verso questa nuova versione
  - se non ci sono problemi, il rilascio prosegue anche sugli altri nodi

Canary releasing



Canary releasing



- A/B testing

- *A/B testing* (o *split testing*)
  - ha l'obiettivo di effettuare un esperimento, nell'ambiente di produzione, per mettere a confronto due versioni (A e B) di un'applicazione o di un servizio software
    - ad es., per apprendere le preferenze degli utenti tra A e B
  - A è la versione corrente, e la versione B viene di solito sperimentata solo su un sottoinsieme degli utenti, per un breve periodo di tempo
    - il monitoraggio degli utenti consente di confrontare il loro comportamento rispetto alle due versioni
  - è supportata dal canary releasing

A/B testing

- Un esempio
  - Amazon sperimenta in Italia il pagamento in 5 rate senza interessi – ma solo per pochi utenti (scelti in modo casuale) e solo su pochi prodotti

- Rollback

- *Rollback*
  - se vengono rilevati problemi nel rilascio, allora è necessario effettuare un rollback (ripristino) della versione precedentemente funzionante dell'applicazione o del servizio
  - il modo in cui può essere effettuato dipende dalla strategia usata per il rilascio – ad es., blue-green deployment o canary releasing
  - al limite, viene ripetuto il rilascio della versione precedente del software
  - è una cattiva pratica cercare di risolvere i problemi direttamente nell'ambiente di produzione (emergency fix) – utilizzando un approccio "pets"
    - è preferibile iniettare cambiamenti solo dal punto iniziale della deployment pipeline – utilizzando invece un approccio "cattle"

- Rolling upgrade

- *Rolling upgrade*
  - consiste nell'installare la nuova versione V_{N+1} dell'applicazione o servizio software in un piccolo numero di nodi alla volta – e nel frattempo disattivare uno stesso numero di nodi in cui è installata la versione attuale V_N
- Conseguenze
  - richiede una quantità ridotta di risorse computazionali
  - tuttavia, presenta l'inconveniente di una possibile "inconsistenza" delle versioni – perché, durante l'upgrade, ci sono alcuni nodi che forniscono la versione V_N e altri nodi che forniscono la versione V_{N+1} del software

- Feature toggling

- *Feature toggling*
  - il software sviluppato contiene, al suo interno, funzionalità (feature) aggiuntive oppure implementate in più varianti o versioni, che possono essere attivate o disattivate mediante dei parametri o flag di configurazione (feature toggle)
    - ad es., la variante A potrebbe corrispondere alla versione attuale V_N e la variante B alla nuova versione V_{N+1} del software
  - la transizione da una versione V_N ad una versione V_{N+1} non avviene al momento del deployment – ma la release si conclude quando vengono cambiati i parametri di configurazione
  - consente di evitare inconsistenze tra versioni quando si utilizza il rolling upgrade

Feature toggling

- *Feature toggling*
  - i parametri di configurazione possono essere definiti a granularità diversa
  - il feature toggling supporta
    - l'esecuzione di test A/B
    - il rollback (senza ripetizione del deployment)
    - la tattica della "degradation"
    - il "dark launching"

* Ulteriori aspetti e pratiche

- Discutiamo ora alcuni ulteriori aspetti relativi alle attività da svolgere nei diversi passi della deployment pipeline

- Gestione delle configurazioni

- La **gestione delle configurazioni** (*configuration management*)
  - è il processo di gestione degli elaborati significativi per un sistema software e delle relazioni tra di essi – nonché dei loro cambiamenti e della loro evoluzione
- Gli interessi principali sono
  - il controllo delle versioni (version control)
  - la gestione delle dipendenze
  - la gestione dei dati di configurazioni del software
  - la gestione degli ambienti di esecuzione

Controllo delle versioni

- Il *controllo delle versioni* (*version control*) riguarda la gestione di tutto il codice per il sistema software (codice sorgente, script, file di configurazione, codice per l'infrastruttura) e delle sue versioni
  - il codice viene gestito in un sistema di controllo delle versioni – ad es., Git o SVN
    - memorizza le diverse versioni del codice
    - registra i cambiamenti da una versione all'altra
    - consente la condivisione del codice e favorisce la collaborazione tra gli sviluppatori

Gestione delle dipendenze

- La *gestione delle dipendenze* riguarda le relazioni tra i componenti o moduli sviluppati dai team di sviluppo e le librerie esterne
  - spesso, un componente o modulo dipende da una specifica versione di una libreria esterna
  - componenti o moduli diversi possono dipendere da versioni differenti di una stessa libreria esterna

Gestione dei dati di configurazione

- La *gestione dei dati di configurazione* riguarda le informazioni di configurazioni del sistema software e dei servizi che lo compongono
  - il comportamento del software dipende spesso da parametri specificati sotto forma di dati di configurazioni
  - questi dati di configurazione vanno tenuti al di fuori del codice sorgente – in modo che possano variare separatamente
  - ogni configurazione è relativa a una specifica versione del software e a uno specifico ambiente di esecuzione
  - nella deployment pipeline, tutti gli ambienti devono utilizzare esattamente gli stessi file binari e script – ma, in genere, possono far riferimento a dati di configurazione differenti
  - inoltre, i dati di configurazione supportano il feature toggling

Gestione degli ambienti

- La *gestione (automatizzata) degli ambienti* riguarda la creazione degli ambienti di esecuzione – nonché la loro manutenzione ed evoluzione
  - si basa su tre principi
    - uso di configurazioni eseguibili (*infrastructure-as-code*)
    - gli ambienti devono essere *autonomici* (*autonomic*)
    - utilizzo del *monitoraggio*

- Gestione dei dati

- I dati costituiscono un aspetto rilevante di molte applicazioni
  - la gestione dei dati nella deployment pipeline richiede di solito delle considerazioni speciali
  - infatti, i dati devono sopravvivere alle versioni del software che li hanno generati, acceduti o modificati
- Alcuni interessi principali
  - il rilascio di una nuova versione del software può richiedere una migrazione dei dati (o del loro formato)
  - la dimensione dei dati non consente di effettuare delle copie "istantanee" dei dati
  - i test devono utilizzare una base di dati separata da quella usata in produzione

- Monitoraggio

- Il rilascio di una nuova versione di un'applicazione o servizio software deve prevedere anche il monitoraggio del software nell'ambiente di produzione
  - per rilevare velocemente eventuali problemi successivi al rilascio
  - per abilitare una diagnosi del sistema, in caso di incidenti
  - per monitorare l'andamento del business
  - per anticipare e prevenire problemi
  - questi aspetti sono discussi nel capitolo sul monitoraggio

- Continuous Integration

- La pratica della **Continuous Integration** (**CI**)
  - un sottoinsieme della CD che riguarda l'automazione delle sole attività iniziali del rilascio – build e primi test automatizzati – eseguite in un integration server
  - l'obiettivo è garantire che l'intero sistema software sia "tecnicamente funzionante" in ogni momento
    - l'alternativa è che il sistema sia normalmente considerato "rotto" – fino a quando non si dimostra il contrario
  - non si interessa però alle attività successive della delivery e del deployment

- Continuous Deployment

- La pratica del **Continuous Deployment**
  - una variante della CD
    - nella CD, la decisione di rilasciare il software nell'ambiente di produzione viene di solito presa da un operatore umano responsabile del rilascio
  - nel continuous deployment, invece, anche il rilascio del software nell'ambiente di produzione avviene automaticamente
  - attenzione, non è una pratica adatta a tutti i sistemi software
    - spesso è preferibile separare la predisposizione al rilascio (un aspetto tecnico) dalla decisione di effettuare il rilascio (un aspetto di business)

- Continuous Development

- Il **Continuous Development** è una metodologia per lo sviluppo del software che estende le metodologie iterative e agili con le pratiche DevOps e della continuous delivery
  - si basa su delle iterazioni di sviluppo e rilascio brevi
  - ciascuna iterazione ha l'obiettivo di realizzare, verificare e rilasciare rapidamente un piccolo cambiamento incrementale
  - benefici
    - rilascio più rapido di nuove funzionalità e caratteristiche
    - migliore qualità e maggior valore del software
    - riduzione dei rischi
    - migliore utilizzo del tempo e maggiore produttività dei team di sviluppo

* Strumenti

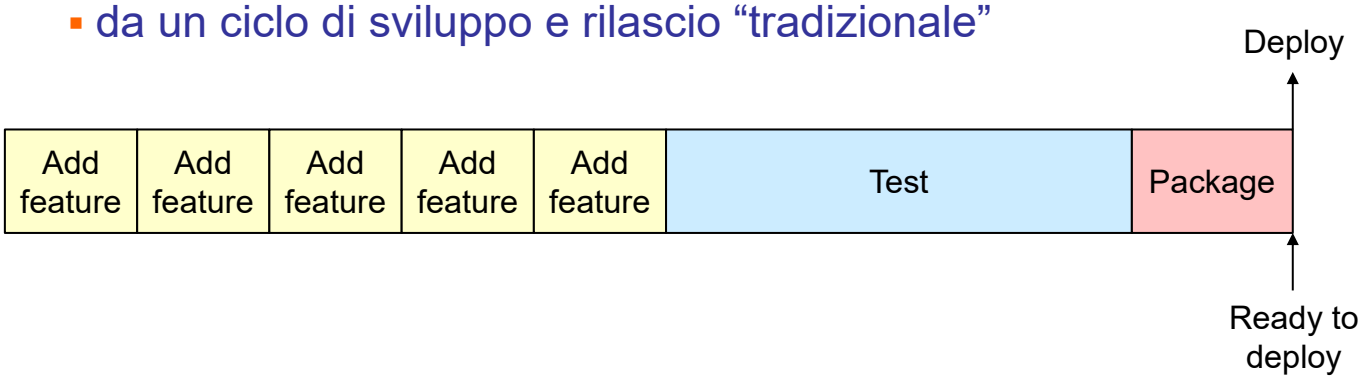
- Esistono numerosi strumenti di supporto alla Continuous Delivery
  - consentono di applicare la CD nel contesto di diversi tipi di infrastrutture
    - per il rilascio su computer fisici, macchine virtuali e container – in un proprio data center (on premises) o nel cloud
  - ecco alcuni strumenti rappresentativi
    - *Jenkins* è un server di automazione estensibile open source, per CI/CD – fornisce centinaia di plugin per consentire la build, il deployment e l'automazione di qualunque progetto
    - *CircleCI*, *Travis CI* e *GoCD* sono strumenti di CI/CD nativi per il cloud – ma possono essere anche utilizzati on premises, per es., con Docker e Kubernetes
    - *AWS CodePipeline*, *Google Cloud Build* e *Azure Pipelines* sono alcuni esempi di servizi cloud flessibili di supporto alla CI/CD e alle pratiche DevOps

* Discussione

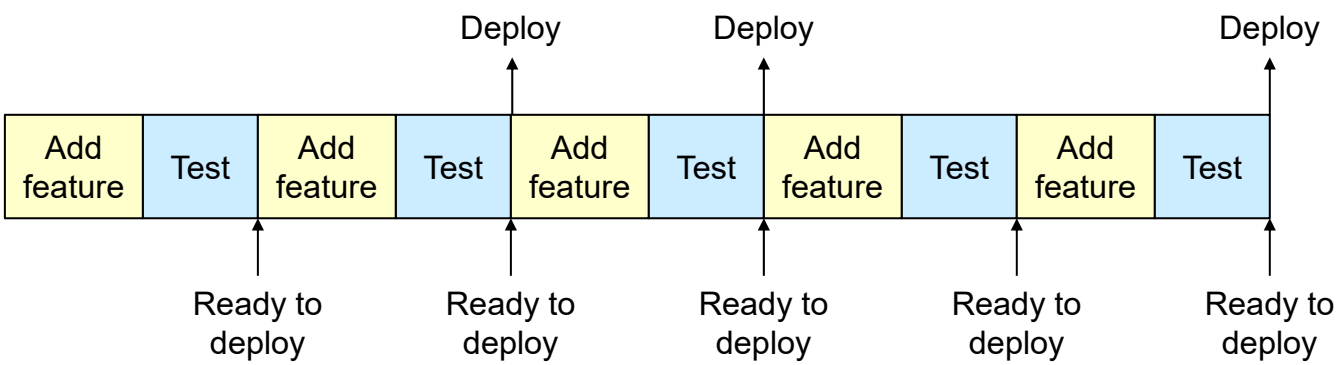
- Secondo Martin Fowler, si sta facendo Continuous Delivery quando
  - il software può essere rilasciato durante tutto il suo ciclo di vita
  - il team di sviluppo dà priorità alla “prontezza al rilascio” del software rispetto all’introduzione di nuove caratteristiche
  - è possibile ottenere un feedback veloce e automatizzato sulla “prontezza al rilascio” ogni volta che viene effettuato un cambiamento nel software
  - è possibile eseguire, a richiesta, il deployment di ogni versione del software in ogni ambiente semplicemente premendo un pulsante

Discussione

- Ovvero, si sta facendo Continuous Delivery quando si passa
  - da un ciclo di sviluppo e rilascio “tradizionale”



  - a dei cicli di sviluppo e rilascio “continui”



Discussione

- Gli obiettivi del Continuous Delivery si possono considerare raggiunti quando si è in grado di effettuare in modo automatizzato e continuo
  - l'integrazione del software realizzato dai team di sviluppo
  - la build e il test del software, per rilevare eventuali problemi
  - lo spostamento del software verso ambienti di esecuzione sempre più simili all'ambiente di produzione finale – per assicurarsi che il software funzioni in produzione
  - questo richiede
    - l'automatizzazione di tutte le attività della delivery del software – sulla base di una deployment pipeline
    - una collaborazione stretta e proficua tra tutti coloro che sono coinvolti nello sviluppo e nel rilascio del software – DevOps