

## Ciclo di vita del software:

Data Engineer

ACADEMIA  
DEL LEVANTE



**Ci saranno sempre aggiornamenti e nuove versioni.**

**Planning:** requisiti del software, da chi vengono dati? Dai clienti. Si mettono insieme tutti gli analisti che acquisiscono tutti i dati con i quali poi danno specifiche dei software.

**Specifiche:** all'uscita dall'analisi devo avere delle specifiche, ovvero dei dati che servono al programmatore per capire cosa fare.

Esempio:

### Student Management System – Specifiche tecniche

- User Interface(UI): HTML, AJAX, JQUERY, JAVASCRIPT
- Database: MySQL
- Programming Language: PHP
- PHP Framework: Laravel
- Server: 8 Core Linux Server

Ecc, ecc, ecc...

**Fase di design:** Linguaggio di modellazione (UML), servono a rappresentare le specifiche in maniera grafica.

**Implementazione:** Il Data Analyst deve creare l'algoritmo (cuore computazionale del programma).

Poi la parte di interfaccia ecc lo farà il dev, e poi l'ultimo passaggio è la creazione dei database, lo può fare il data analyst o il sistemista di database o il programmatore dev.

**Hardware:** Come e dove verrà distribuito. Dipende dal traffico.

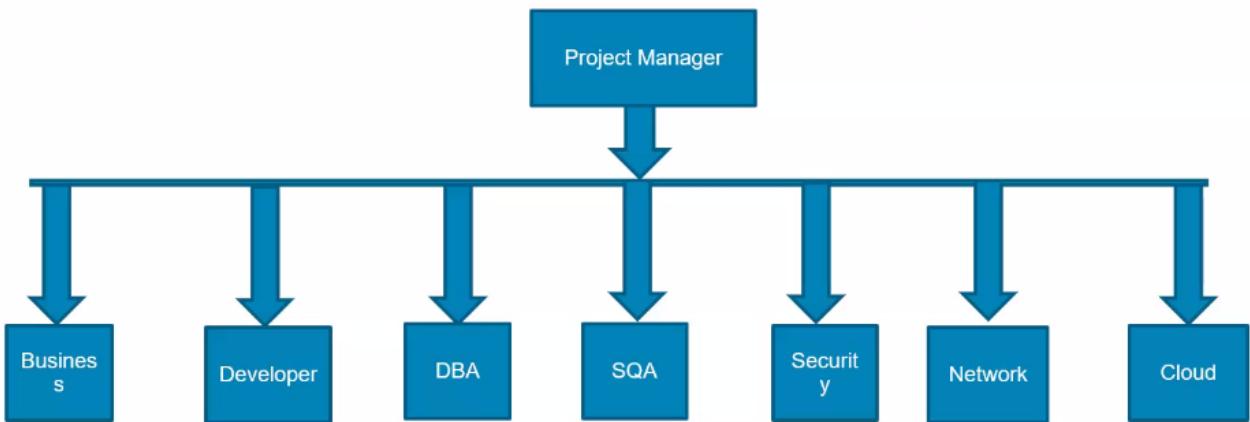
**Sicurezza:** Come raccogliere i dati personali o le password (quindi criptare ecc).

**Chi si occupa del flusso dati e interviene sulla rete?** Saranno i data analyst

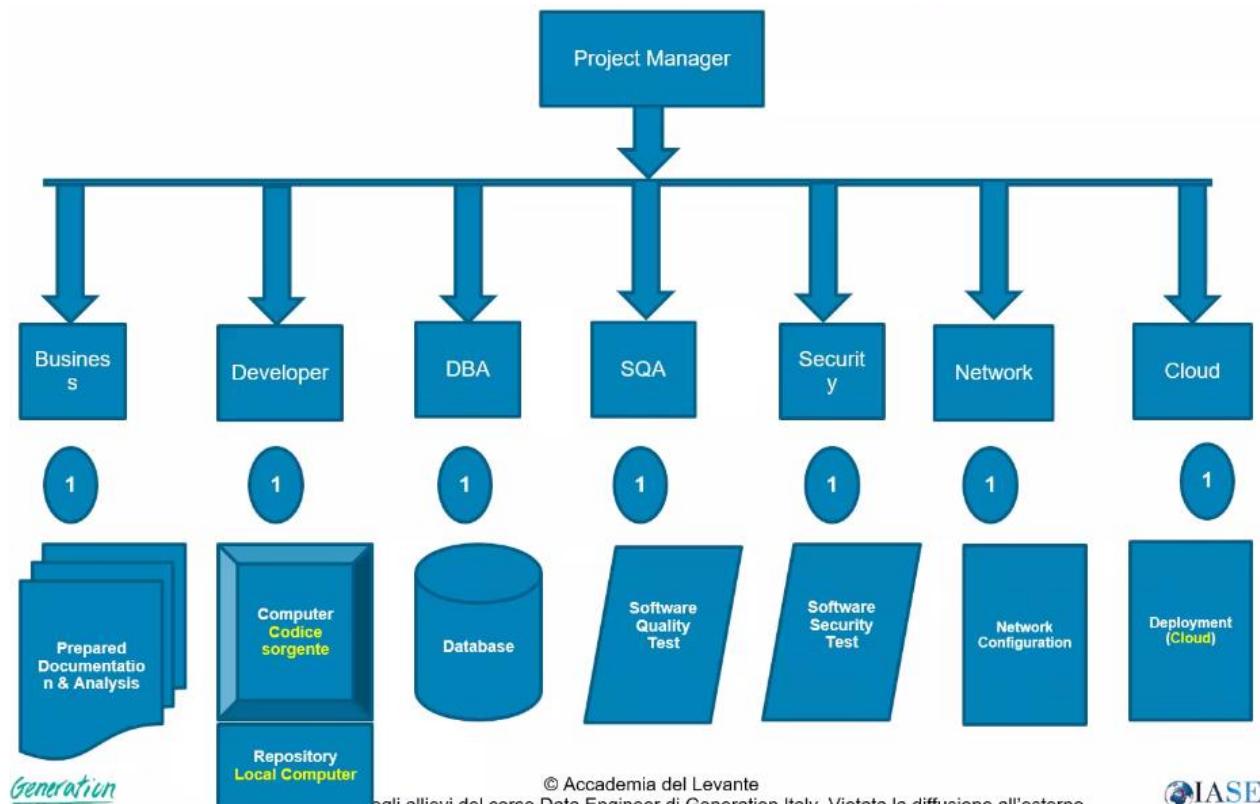
Saranno loro a dare un minimo/massimo sul flusso dati.

Poi c'è il testing, l'integrazione e la manutenzione.

# Project Team Members

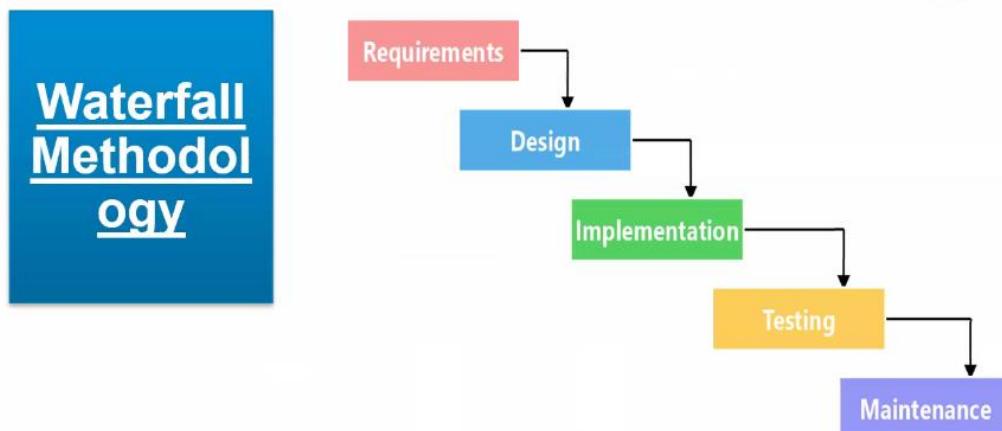


Cosa fa ciascuna figura in un team?



© Accademia del Levante  
Gli allievi del corso Data Engineer di Generation Italy. Vietata la diffusione all'esterno.

Modalità waterfall:



Metodologia del Waterfall Model, nota anche come Linear Sequential Life Cycle Model. Il modello a cascata è seguito in ordine sequenziale, quindi il team di sviluppo del progetto passa alla fase successiva di sviluppo o test solo se il passaggio precedente è stato completato con successo.

## Vantaggi del Waterfall Model

- È uno dei modelli più facili da gestire. Per sua natura, ogni fase prevede risultati specifici e un processo di revisione.
- Funziona bene per progetti di piccole dimensioni in cui i requisiti sono facilmente comprensibili.
- Consegnà rapida del progetto
- Il processo e i risultati sono ben documentati.
- Metodo facilmente adattabile
- Questa metodologia di gestione del progetto è utile per gestire le dipendenze.

**Un grandissimo numero di progetti è stato realizzato tramite il Waterfall Model**

## Ma cosa accade se...?

- Il cliente cambia alcuni requisiti?
- Si rileva un problema di bug/vulnerabilità?
- Il cliente vuole aggiungere un ulteriore modulo/funzionalità?
- Si vuole altro....

**Waterfall Model methodology is unable to solve this types of problems!**

## Limiti del Waterfall Model

- Non è un modello ideale per un progetto di grandi dimensioni
  -
- Se il requisito non è chiaro all'inizio, il metodo è meno efficace.
- Molto difficile tornare indietro per apportare modifiche alle fasi precedenti.
- Il processo di test inizia una volta terminato lo sviluppo. Pertanto, è molto probabile che i bug vengano rilevati più avanti nello sviluppo e siano costosi da risolvere.

È un software MONOLITICO, tutto d'un pezzo.

Un **software monolitico** è un'applicazione in cui tutte le funzionalità e i componenti sono strettamente integrati in un unico blocco di codice. Questo significa che l'intera applicazione viene sviluppata, distribuita ed eseguita come un'unica entità.

# Agile

Agile è un approccio iterativo alla gestione dei progetti e allo sviluppo software che aiuta i team a fornire valore ai propri clienti più velocemente e con meno problemi.



## Risolve i problemi che si rilevano con la waterfall!

Per ogni modulo faccio questo ciclo. Infatti, il software è diviso in moduli.

I **moduli** di un software sono parti indipendenti di un programma che svolgono funzioni specifiche e possono interagire tra loro. Un software modulare è diviso in più moduli, ognuno dei quali gestisce un aspetto del sistema, rendendo il codice più organizzato, riutilizzabile e facile da manutenere.

### Caratteristiche dei moduli software

- **Indipendenza:** Ogni modulo ha un compito specifico e può essere sviluppato e testato separatamente.
- **Riutilizzabilità:** Un modulo può essere riutilizzato in altri progetti senza bisogno di riscrivere il codice.
- **Manutenibilità:** Se un modulo deve essere aggiornato o corretto, non è necessario modificare l'intero software.
- **Interfacciamento:** I moduli comunicano tra loro tramite API, funzioni o protocolli definiti.

### Esempio di software modulare

Un sistema di e-commerce potrebbe avere:

- **Modulo di autenticazione:** Gestisce login e registrazione.
- **Modulo di catalogo prodotti:** Contiene le informazioni sugli articoli in vendita.
- **Modulo di pagamento:** Gestisce le transazioni e i metodi di pagamento.
- **Modulo di spedizione:** Calcola costi e tempi di consegna.

## Vantaggi dell' Agile Model

- È focalizzato sul processo del cliente. Quindi, fa in modo che il cliente sia continuamente coinvolto durante ogni fase.
- I team agili sono estremamente motivati e auto-organizzati, quindi è probabile che forniscano un risultato migliore dai progetti di sviluppo.
- Il metodo di sviluppo software agile garantisce il mantenimento della qualità dello sviluppo
- Il processo è completamente basato sul progresso incrementale. Pertanto, il cliente e il team sanno esattamente cosa è completo e cosa no. Ciò riduce il rischio nel processo di sviluppo.

In questo caso più modulo possono venire sviluppati in parallelo!

## Precedenti problemi? Risolti!

- Cambiamenti di requisiti voluti dal client epossono essere facilmente e velocemente integrati.
- bug/problems di vulnerabile dell'applicazione sono velocemente risolti.
- È possibile aggiungere ulteriori moduli a progetto in corso modulo



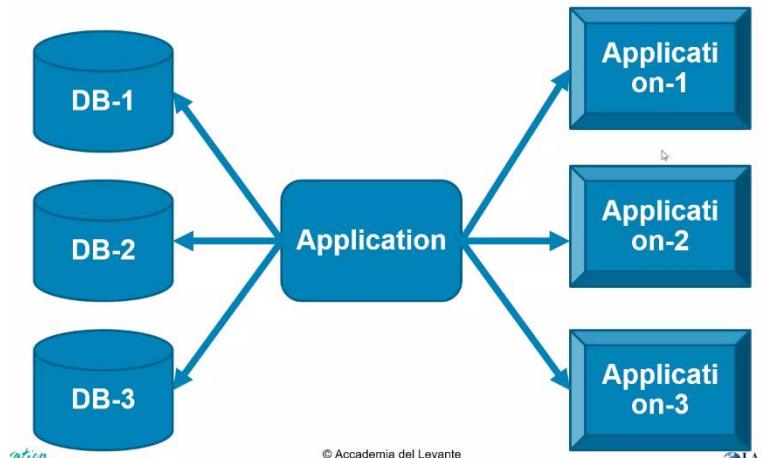
Se il software diventa in Cloud allora diventa un SaaS (Software as a service)

- Molte scuole sono interessate al “**School Management System**”
- La vostra application business e stata alalrgata
- Vi piacerebbe implementare il vostro business come un **SaaS** Model.

**Esempio:**

Gmail, Google Drive

## Software as a Service (SaaS) Model

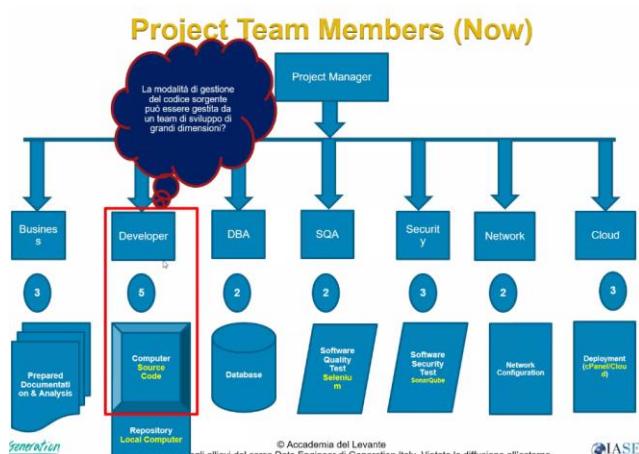


Un **SaaS (Software as a Service)** è un modello di distribuzione del software in cui le applicazioni sono ospitate su server remoti e accessibili via Internet, senza bisogno di installarle sul proprio dispositivo.

### Caratteristiche principali del SaaS

- Basato su cloud:** Il software è eseguito sui server del provider e accessibile tramite browser.
- Nessuna installazione locale:** Gli utenti non devono scaricare o aggiornare manualmente il software.
- Aggiornamenti automatici:** Il provider gestisce aggiornamenti e manutenzione senza interruzioni per l'utente.
- Abbonamento:** Di solito, il servizio è offerto con un modello di pagamento a **canone mensile o annuale**.
- Scalabilità:** Gli utenti possono adattare le risorse in base alle loro esigenze (es. più spazio di archiviazione o utenti aggiuntivi).
- Accesso da qualsiasi dispositivo:** Basta una connessione Internet per usare il servizio ovunque.

I SaaS sono aggiornati in tempo reale.



Il codice sorgente può essere gestito da un team di sviluppatori, non posso adottare solo una repository in locale.

## Perchè il sistema di controllo versione è important?

Poiché sappiamo che un prodotto software è sviluppato in collaborazione da un gruppo di sviluppatori, questi potrebbero trovarsi in luoghi diversi e ognuno di essi contribuisce ad alcuni tipi specifici di funzionalità/caratteristiche. Quindi, per contribuire al prodotto, hanno apportato modifiche al codice sorgente (aggiungendo o rimuovendo linee di codice). Un sistema di controllo della versione è un tipo di software che aiuta il team di sviluppatori a comunicare e gestire (tracciare) in modo efficiente tutte le modifiche apportate al codice sorgente insieme alle informazioni su chi ha apportato e quali modifiche sono state apportate

**Il sistema di controllo della versione tiene traccia delle modifiche apportate a un particolare software e scatta un'istantanea di ogni modifica.**

### **GitHub:**

È uno dei software di revisione di codice più utilizzati.

Come si carica una versione su git? Con un “import”.

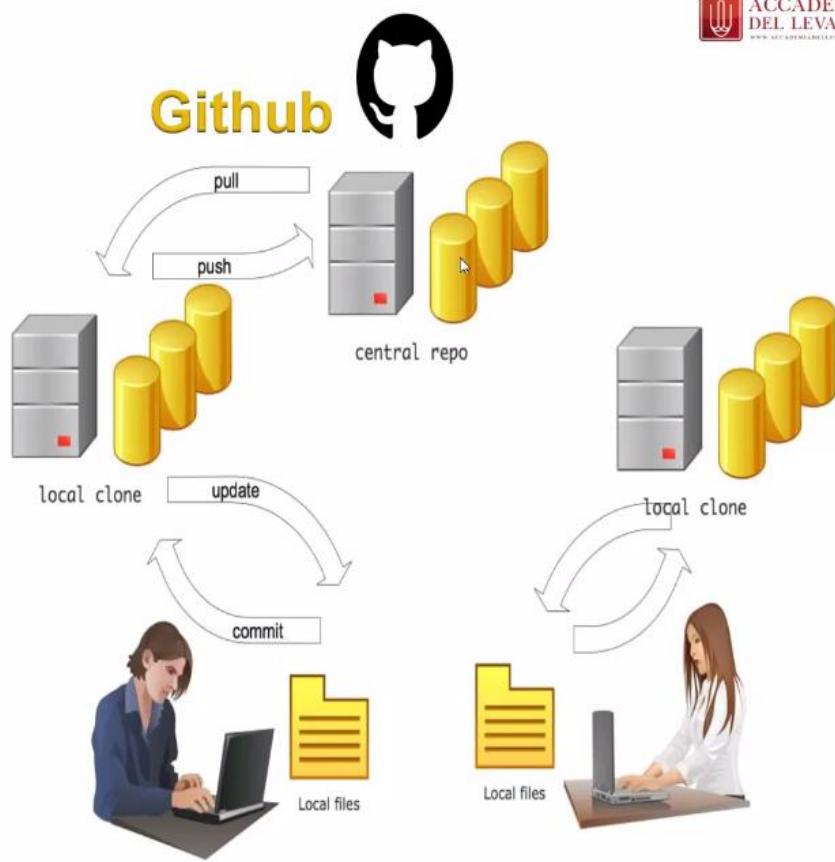
## Version Control Systems

Il controllo della versione (Version Control), noto anche come controllo del codice sorgente, è la pratica di tenere traccia e gestire le modifiche al codice sorgente. I sistemi di controllo della versione sono strumenti software che aiutano i team software a gestire le modifiche al codice sorgente nel tempo. Con l'accelerazione degli ambienti di sviluppo, i sistemi di controllo delle versioni aiutano i team software a lavorare più velocemente e in modo più intelligente.

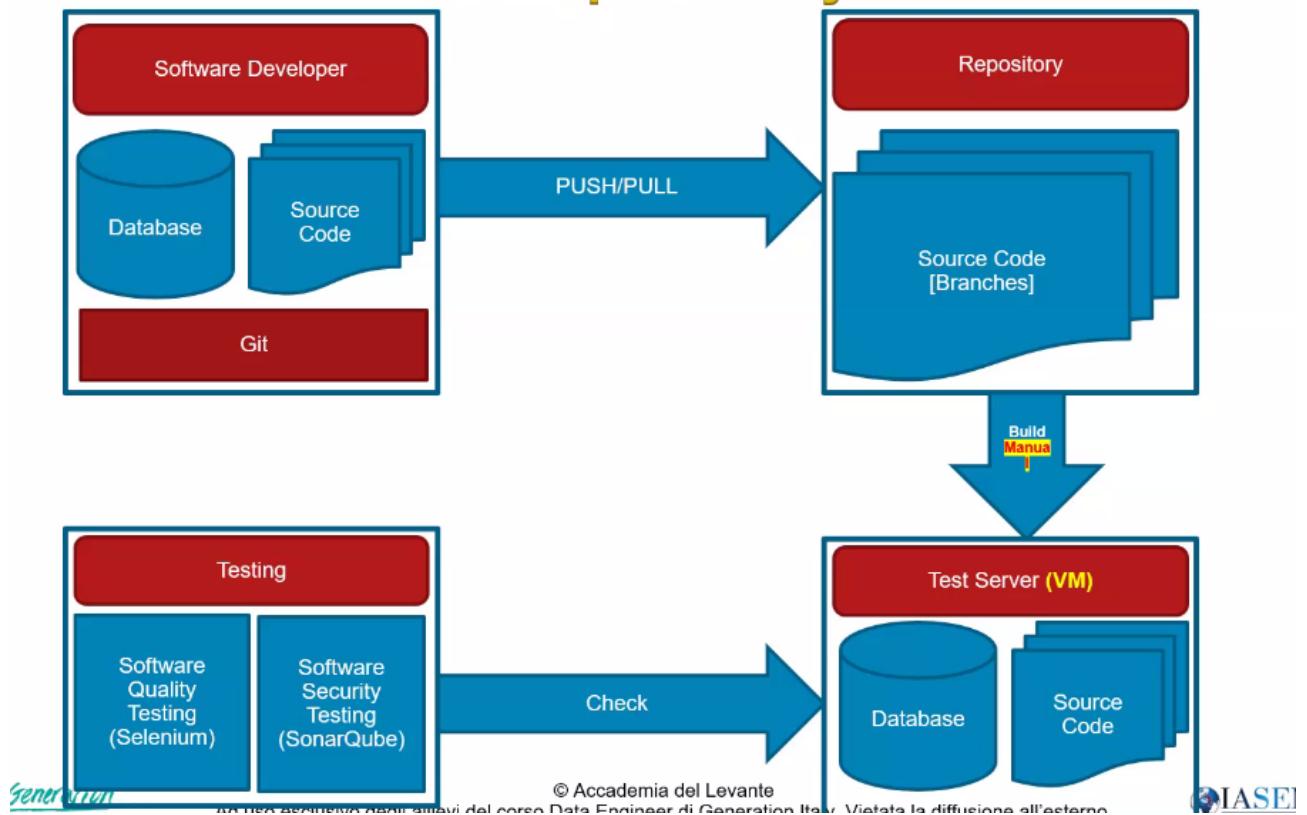


### Data Engineer

GitHub, Inc. is a provider of Internet hosting for software development and version control using Git. It offers the distributed version control and source code management (SCM) functionality of Git, plus its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, continuous integration, and wikis for every project.



# Development Cycle



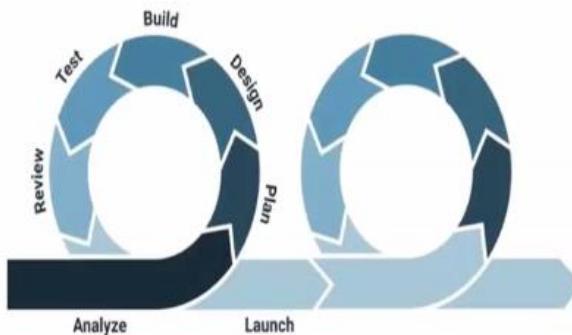
## In Real Development Project

- Ogni **sprint**, Può essere implementato con migliaia di Features, Richieste di modifica, Bug, Tasks, Defect-testing e problemi correlati alla vulnerabilità.
- In tal modo **ogni giorno** possono essere effettuati centinaia di **commit** di codice sorgente.
- Ma risulta difficile dover costruire **manualmente** la **build** dopo ogni commit del codice sorgente

↳

Per risolvere quest'ultimo problema, introduciamo la **CI/CD pipeline automatizzata**

## Fino ad ora abbiamo completato... la metà del DevOps



Questo è solo la **Dev** – Part del **DevOps**  
**Dev = Development**



## Problema dell'Agile?

### La parte network!

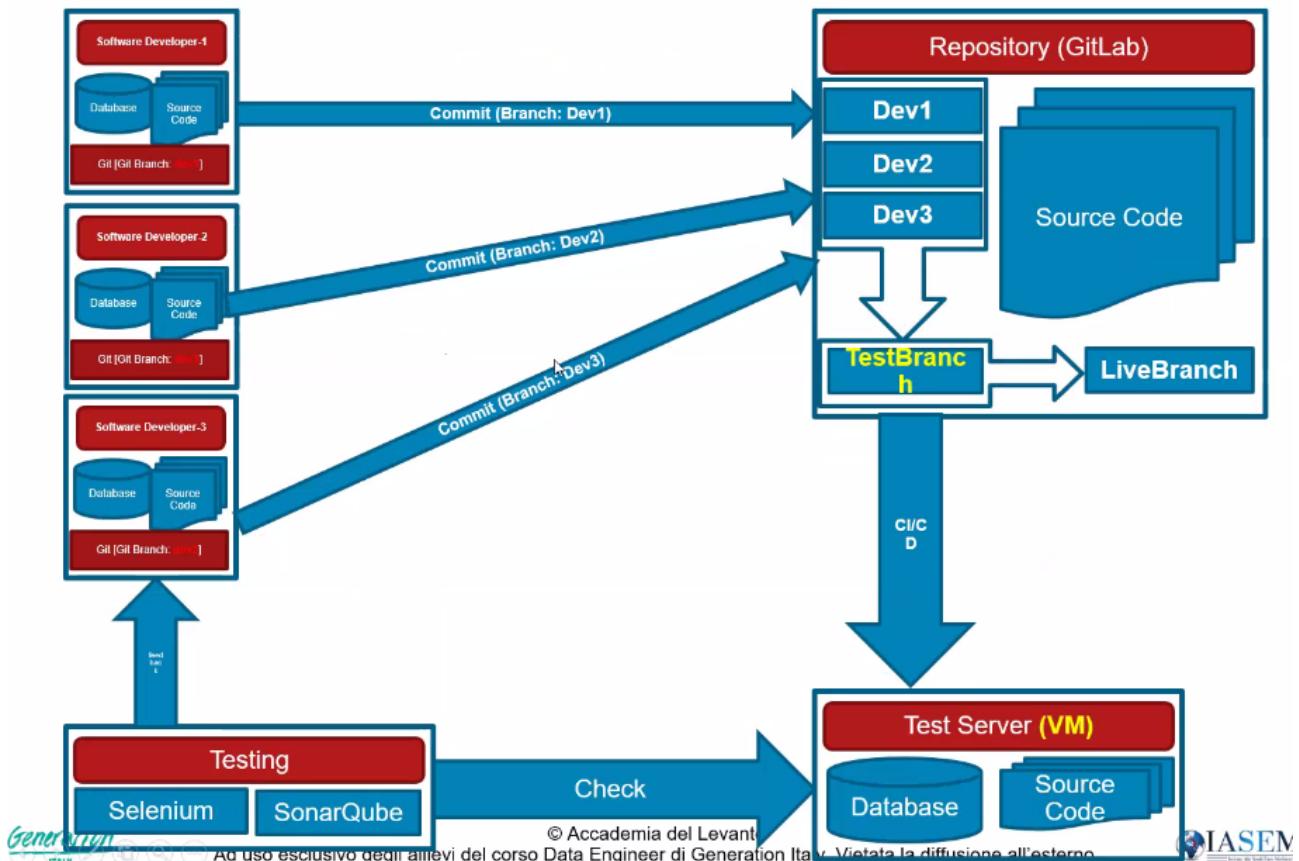
#### Cosa si intende per OPS (la parte arancione):

Il team di operazioni è responsabile di:

- Gestire l'infrastruttura: Configurare, monitorare e mantenere i server, le reti e altre risorse hardware e software necessari per eseguire l'applicazione.
- Deployment: Gestire il processo di rilascio del software in produzione, assicurandosi che l'applicazione funzioni correttamente e sia accessibile agli utenti finali.
- Monitoraggio e manutenzione: Monitorare l'applicazione e l'infrastruttura in tempo reale per rilevare eventuali problemi e garantire che tutto funzioni senza intoppi. Ciò include anche il gestire la sicurezza e risolvere i problemi operativi.
- Scalabilità e prestazioni: Assicurarsi che l'infrastruttura possa supportare la crescita dell'azienda e le piccole o grandi modifiche del software, come l'aggiunta di nuove risorse o la gestione di un aumento del traffico.

Come la parte di “git” si mischia con la metodologia “devops”?

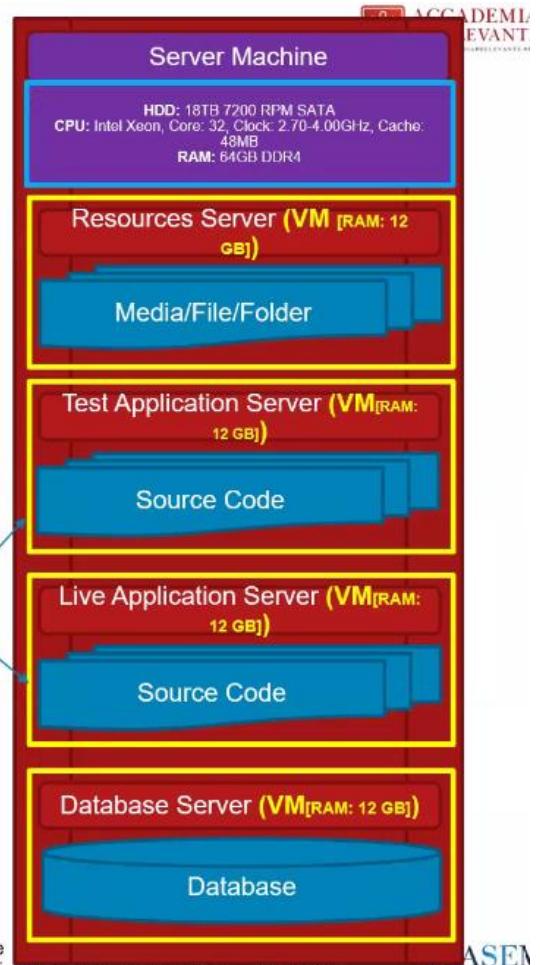
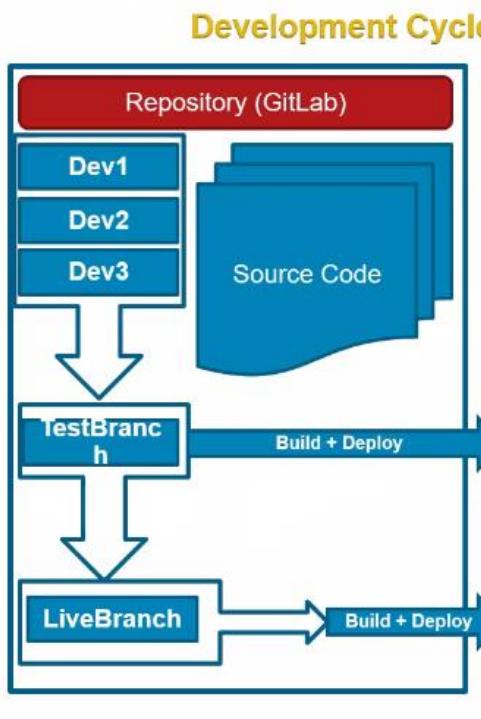
Diamo un’occhiata ad uno... Development Cycle:



- ➔ Ogni team Dev lavora su un singolo modulo;
- ➔ Ogni team lavora in modo indipendente.

## → Quando un team finisce fa il commit al repository

Data Engineer



Generation

Ad uso esclusivo degli allievi del corso Data Engineer di Generation IT&DEV. VIETATA LA DISTRIBUZIONE ALL'esterno.

ASEM

C'è la macchina fisica e la macchina virtuale.

**CI/CD** è un insieme di pratiche fondamentali in DevOps che aiutano a migliorare l'efficienza e la qualità nel ciclo di vita del software, automatizzando e ottimizzando le fasi di sviluppo e rilascio.

CI (Continuous Integration) - Integrazione continua

La CI è una pratica in cui gli sviluppatori integra frequentemente (di solito più volte al giorno) il loro codice nel repository centrale. L'idea è che, ogni volta che un nuovo codice viene aggiunto, venga immediatamente testato per garantire che non ci siano conflitti o bug introdotti.

Caratteristiche principali della CI:

- **Integrazione frequente:** Ogni sviluppatore aggiunge il suo codice al repository principale frequentemente, in modo che le modifiche vengano verificate subito.
- **Automazione dei test:** Ogni integrazione del codice è seguita da una serie di test automatici che verificano che il codice non rompa funzionalità esistenti.
- **Prevenire conflitti:** Integrare il codice spesso riduce i conflitti quando diversi sviluppatori lavorano sullo stesso codice, migliorando la coesione del progetto.

In pratica, CI garantisce che ogni nuova modifica venga testata immediatamente e che il software rimanga funzionante anche dopo ogni cambiamento.

## **CD (Continuous Delivery/Continuous Deployment) - Consegn... continua / Distribuzione continua**

Il CD rappresenta due pratiche strettamente collegate ma leggermente diverse:

### **Continuous Delivery (Consegna continua):**

Con Continuous Delivery, il codice viene automaticamente preparato per essere rilasciato in produzione. Ogni volta che il codice passa attraverso i test (e altre fasi di controllo qualità), il sistema lo rende pronto per il deployment. Tuttavia, il rilascio effettivo in produzione può richiedere ancora l'intervento manuale dell'operatore, che decide quando distribuire la nuova versione.

Caratteristiche principali:

- Automazione del rilascio: Il codice viene automaticamente testato e preparato per il rilascio, ma la distribuzione finale è ancora un'azione manuale.
- Rilascio frequente e sicuro: Il codice è sempre pronto per essere distribuito, riducendo i tempi di attesa tra lo sviluppo e la messa in produzione.

### **Continuous Deployment (Distribuzione continua):**

Continuous Deployment porta la Consegna continua a un livello successivo. Qui, ogni modifica che passa i test viene automaticamente rilasciata in produzione senza l'intervento umano. Questo significa che ogni nuova funzionalità o correzione viene distribuita direttamente agli utenti finali senza ritardi.

Caratteristiche principali:

- Automazione completa: Ogni modifica, una volta superati i test, è automaticamente distribuita in produzione senza necessità di intervento manuale.
- Distribuzione rapida e continua: Le nuove versioni del software arrivano agli utenti molto più velocemente, migliorando il ciclo di feedback.

Differenza tra Continuous Delivery e Continuous Deployment:

- Continuous Delivery: Il codice è pronto per il rilascio, ma il rilascio stesso è ancora manuale.
- Continuous Deployment: Ogni modifica che supera i test è distribuita automaticamente in produzione.

Perché CI/CD è importante?

- Velocità e frequenza di rilascio: Permette di rilasciare nuove funzionalità e correzioni in modo rapido e frequente, riducendo il ciclo di vita del software.
- Affidabilità e qualità: L'automazione dei test e del rilascio aiuta a prevenire bug e garantire che il software sia sempre di alta qualità.
- Riduzione dei rischi: Le modifiche vengono distribuite in piccoli blocchi, riducendo il rischio di problemi gravi che potrebbero derivare da un grande rilascio.

Strumenti tipici per CI/CD:

- Jenkins

- GitLab CI/CD

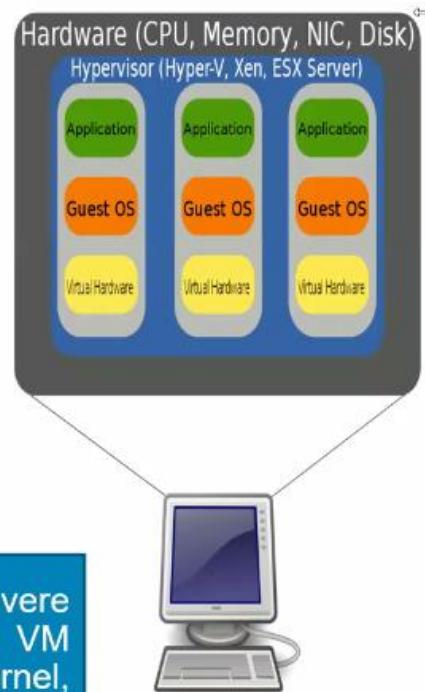
#### Esempio di flusso CI/CD:

1. Sviluppo: Gli sviluppatori scrivono e modificano il codice.
2. Integrazione continua (CI): Ogni volta che il codice viene aggiornato, viene eseguito un processo di integrazione per unire tutte le modifiche e testarlo.
3. Test automatici: I test vengono eseguiti automaticamente su ogni modifica.
4. Consegnare continua (CD): Il codice testato viene preparato e reso pronto per il rilascio.
5. Distribuzione continua: Il codice può essere automaticamente distribuito in produzione se supera tutti i test.

In sostanza, CI/CD è una metodologia che consente di rilasciare software più velocemente e con maggiore qualità, automatizzando i processi di testing, integrazione e distribuzione.

#### Cos'è una virtual machine?

Una **macchina virtuale (VM)** è una risorsa di calcolo che utilizza software anziché un computer fisico per eseguire programmi e distribuire app. Ogni macchina virtuale esegue la propria CPU, memoria, interfaccia di rete e spazio di archiviazione, creati su un sistema hardware fisico (localizzato o fuori sede). Le macchine virtuali funzionano sul software hypervisor, che imita l'infrastruttura fisica e divide le risorse in più macchine virtuali. L'hypervisor viene anche definito macchina host o monitor della macchina virtuale.



Le VM tendono ad essere ingombranti e ad avere dimensioni di molti gigabyte perché ciascuna VM contiene il proprio sistema operativo guest, kernel, file binari, librerie e la relativa applicazione.

Le **virtual machine (VM)** sono una parte fondamentale di molte **pratiche DevOps**, e vengono spesso utilizzate per supportare e migliorare il processo di **CI/CD**. Ecco come le **VM** si collegano a **CI/CD** in un flusso di lavoro DevOps:

#### Automazione della creazione di ambienti di sviluppo e testing

Le **virtual machine** permettono di creare ambienti **isolati e riproducibili** dove il software può essere testato e sviluppato senza interferire con altri sistemi o ambienti.

- **Integrazione con CI:** Durante il processo di **integrazione continua (CI)**, le VM vengono utilizzate per creare ambienti identici a quello di produzione in cui il codice può essere automaticamente testato. Ogni volta che una modifica viene integrata nel codice, le VM vengono utilizzate per eseguire test in ambienti che emulano il più possibile la produzione.
- **Ambienti isolati:** Ogni sviluppatore o team di sviluppo può lavorare su una VM separata, garantendo che le modifiche non confliggano con quelle degli altri membri del team.

## 2 Creazione e gestione di ambienti di staging e produzione

Nel contesto del **Continuous Delivery (CD)**, le VM possono essere utilizzate per configurare e gestire ambienti **di staging e di produzione**:

- **Staging:** Le VM possono replicare esattamente l'ambiente di produzione, permettendo di testare le nuove versioni del software in un ambiente che imita quello finale. Questo è fondamentale per garantire che non ci siano sorprese quando il codice viene distribuito in produzione.
- **Produzione:** Molti ambienti di produzione, specialmente in cloud, utilizzano VM per eseguire il software. Le VM possono essere create, configurate e gestite automaticamente attraverso script e tool DevOps, con l'ausilio di strumenti come **Ansible**, **Terraform**, **Kubernetes** e **Docker**.

## 3 Scalabilità e distribuzione automatica

Le **virtual machine** sono particolarmente utili per garantire che il processo di distribuzione e scalabilità del software avvenga senza intoppi:

- **Auto-scaling:** Le VM possono essere scalate automaticamente in base al carico. Ad esempio, se il traffico aumenta, il sistema può creare nuove VM per supportare il carico extra, rendendo facile distribuire nuove versioni dell'applicazione in modo continuo senza interruzioni.
- **Orchestrazione automatica:** Strumenti come **Kubernetes** possono gestire automaticamente il provisioning di VM per distribuire l'applicazione in modo elastico e scalabile.

## 4 Esecuzione di test su VM durante il CI/CD

Nel processo di **CI/CD**, le VM sono ampiamente utilizzate per eseguire test automatizzati su diverse configurazioni e ambienti:

- **Test automatici:** Ogni volta che il codice viene integrato (CI), le VM possono essere avviate per eseguire test di unità, test di integrazione o test di sistema. Ogni VM può eseguire un set di test specifico, a seconda delle configurazioni dell'ambiente.
- **Test su diverse configurazioni:** Le VM possono emulare diverse configurazioni hardware e software per testare il codice su più ambienti (ad esempio, diverse versioni di un sistema operativo o configurazioni di rete).

## Virtualizzazione vs. Containerizzazione

Sia le **virtual machine** che i **container** sono utilizzati in DevOps, ma con scopi leggermente diversi. Entrambi offrono isolamento, ma i **container** (come Docker) sono generalmente più leggeri e veloci rispetto alle VM.

- **Virtual Machines:** Offrono una maggiore separazione a livello di sistema operativo, consentendo di creare ambienti completamente isolati. Sono più adatte a situazioni dove è

necessario testare su configurazioni diverse di sistema operativo o quando un'applicazione richiede un ambiente più complesso.

- **Container:** Sono più veloci da avviare e più leggeri, ideali per microservizi e per applicazioni che devono essere scalate rapidamente.

Molti ambienti DevOps utilizzano una combinazione di VM e container per sfruttare i punti di forza di entrambe le soluzioni. Ad esempio, i container potrebbero essere utilizzati per il deploy di microservizi, mentre le VM potrebbero essere usate per il testing o la gestione dell'infrastruttura di produzione.

#### Esempio di flusso CI/CD con VM:

1. **Codifica:** Gli sviluppatori scrivono codice nelle loro macchine locali.
2. **CI:** Quando il codice viene pushato su un sistema di versioning come **Git**, un server CI (ad esempio **Jenkins**) avvia il processo di integrazione. Questo include la creazione automatica di una VM per eseguire i test (unità, integrazione, ecc.).
3. **Staging/Pre-produzione:** Se i test sono superati, una VM in un ambiente di staging è configurata per testare la nuova versione dell'applicazione in un ambiente che replica la produzione.
4. **CD:** Una volta che la versione è pronta, una VM o una serie di VM vengono avviate per il deploy in produzione, utilizzando strumenti di automazione e gestione come **Ansible** o **Terraform**.
5. **Monitoraggio:** Le VM in produzione sono monitorate per garantire il corretto funzionamento dell'applicazione. Se necessario, nuove VM possono essere create per scalare l'applicazione.

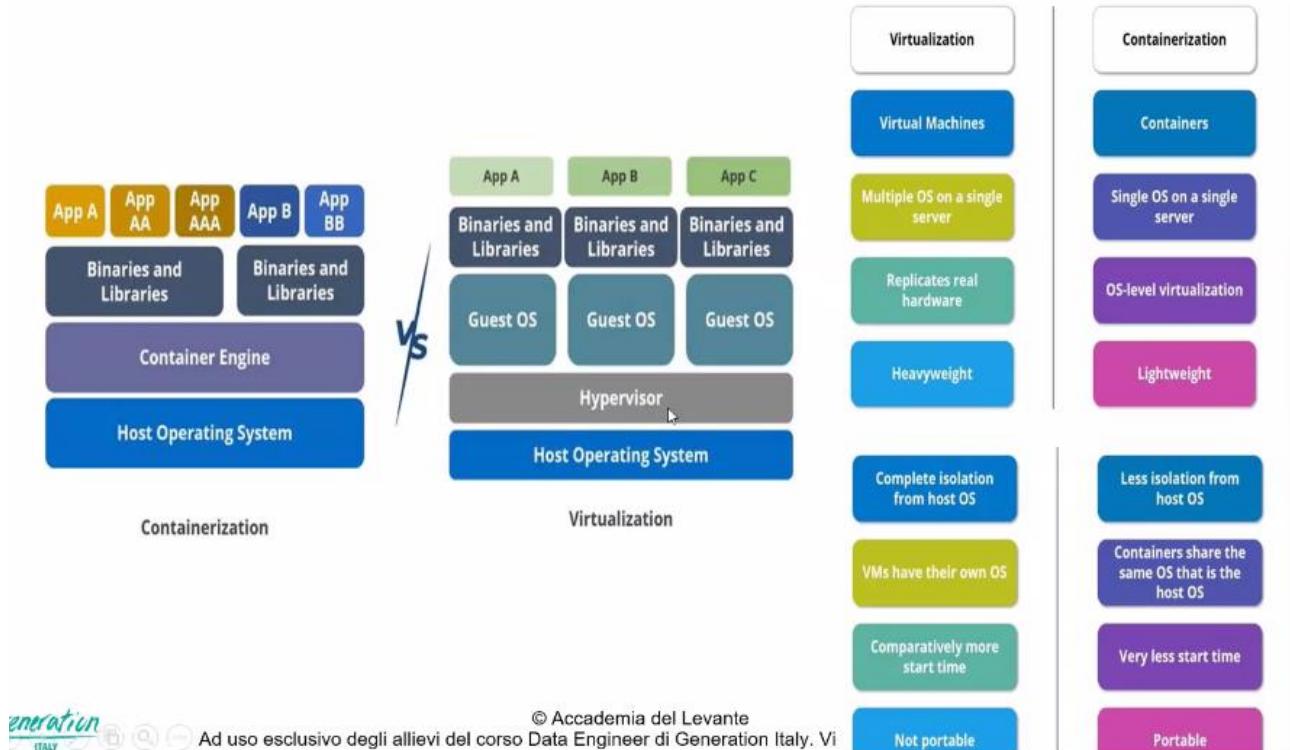
#### Strumenti di automazione con VM in CI/CD:

- **Vagrant:** Strumento per automatizzare la creazione di VM, utile in ambienti di sviluppo.
- **Terraform:** Strumento di Infrastructure as Code (IaC) che può creare e gestire VM automaticamente in un ambiente cloud (AWS, Azure, GCP).
- **Ansible:** Strumento di automazione che può configurare VM per garantire che l'ambiente sia sempre pronto e configurato correttamente.
- **Jenkins, GitLab CI:** Possono essere configurati per utilizzare VM durante il processo di build, test e deploy.

#### In sintesi:

Le **virtual machine** sono utilizzate in DevOps per creare ambienti isolati e replicabili, per eseguire test e gestire la distribuzione automatica delle applicazioni, spesso in combinazione con strumenti di automazione per supportare **CI/CD**. Le VM garantiscono che il software venga distribuito su ambienti di produzione e staging scalabili, affidabili e ben testati.

# Container vs. VM



Il container lo si crea con docker, all'interno ci sono solo le librerie. Non c'è il sistema operativo! Il container funziona indipendentemente dal sistema operativo che c'è.

Caratteristica	Container 	Macchina Virtuale (VM) 
Isolamento	Parziale, condividono il kernel del sistema operativo	Completo, ogni VM ha il proprio sistema operativo
Avvio	Veloce (secondi)	Lento (minuti)
Peso	Leggeri (MB)	Più pesanti (GB)
Consumo risorse	Minimo, condivide il kernel del sistema operativo	Alto, ogni VM ha il proprio OS e kernel
Flessibilità	Ideale per microservizi e deploy rapidi	Più adatta per applicazioni monolitiche o legacy
Compatibilità	Dipende dal sistema operativo host	Può eseguire qualsiasi OS, anche diversi dal sistema host
Gestione	Orchestrabili con strumenti come Kubernetes	Gestite con hypervisor come VMware, VirtualBox, Hyper-V
Sicurezza	Minore, perché condividono il kernel	Maggiore, ogni VM è completamente isolata
Esempi di utilizzo	Docker, Kubernetes	VMware, VirtualBox, AWS EC2 con VM

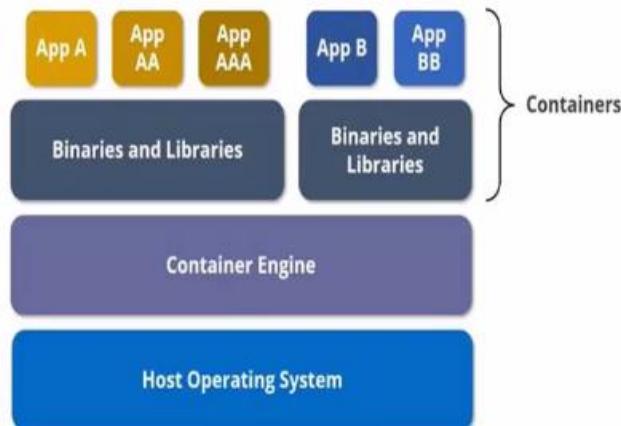
### In sintesi:

- Se hai bisogno di **velocità, leggerezza e scalabilità**, usa i **container**.
- Se hai bisogno di **maggior isolamento e compatibilità con più OS**, usa le **VM**. wow

# Container

Un container è un pezzo di software che impacchetta il codice e tutte le sue dipendenze.

La containerizzazione è il processo di confezionamento del codice software insieme a tutti i suoi componenti essenziali.



Cosa costruiscono quindi i devops?

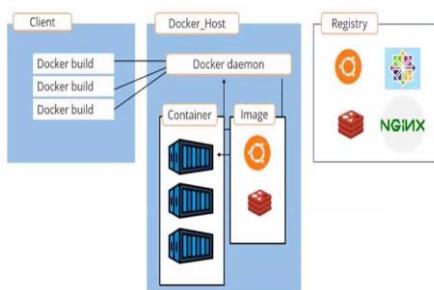
## Docker

Docker è una piattaforma di containerizzazione per creare pacchetti della tua applicazione insieme a tutte le sue dipendenze.

Docker è una piattaforma gratuita e aperta per lo sviluppo, la distribuzione e l'esecuzione di software.

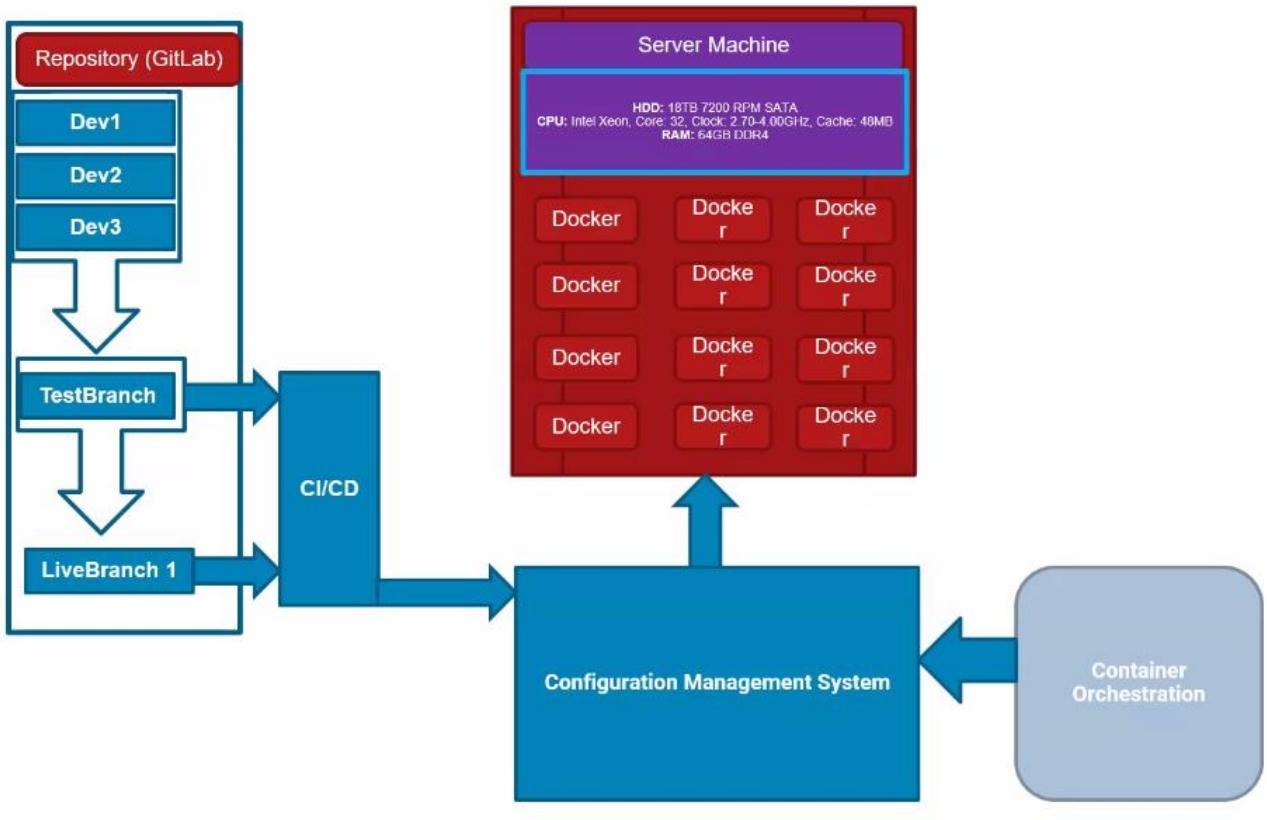
**Vantaggi di Docker:** alcuni dei vantaggi offerti da Docker nelle varie fasi del ciclo di vita dello sviluppo del software (SDLC) sono i seguenti:

1. Build
2. Test
3. Deploy
4. Maintain



Con docker facciamo i container, ossia per ogni modulo (microservizio) viene creato un container che vengono caricati sul server (virtuale o fisico) e si testa. I container sono trattati con l'orchestratore di container.

# Development Cycle (Cont)

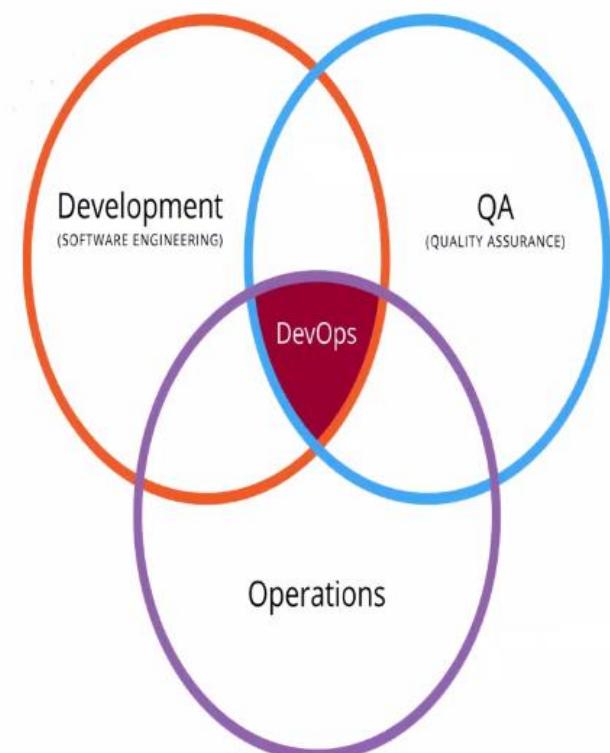
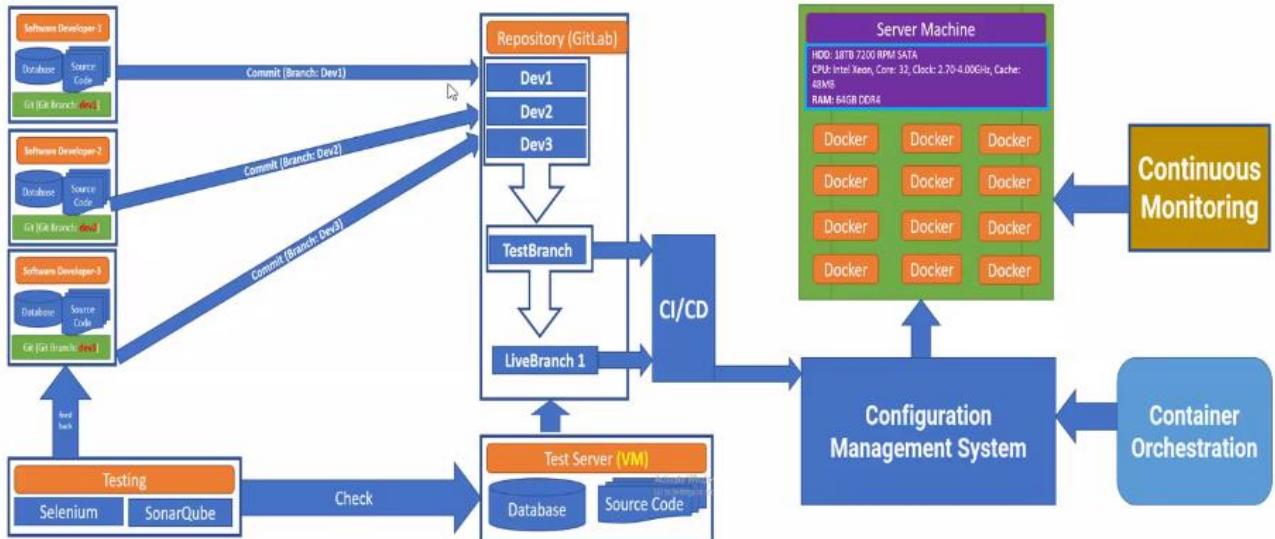


## Definizioni:

Un microservizio è un'architettura software in cui un'applicazione viene suddivisa in piccole parti indipendenti chiamate microservizi, ognuna delle quali svolge una funzione specifica e ben definita. Ogni microservizio è autonomo, può essere sviluppato, distribuito e scalato separatamente e comunica con gli altri microservizi attraverso interfacce ben definite, solitamente tramite API (Application Programming Interface).

ALLA FINE, ABBIAMO QUESTO:

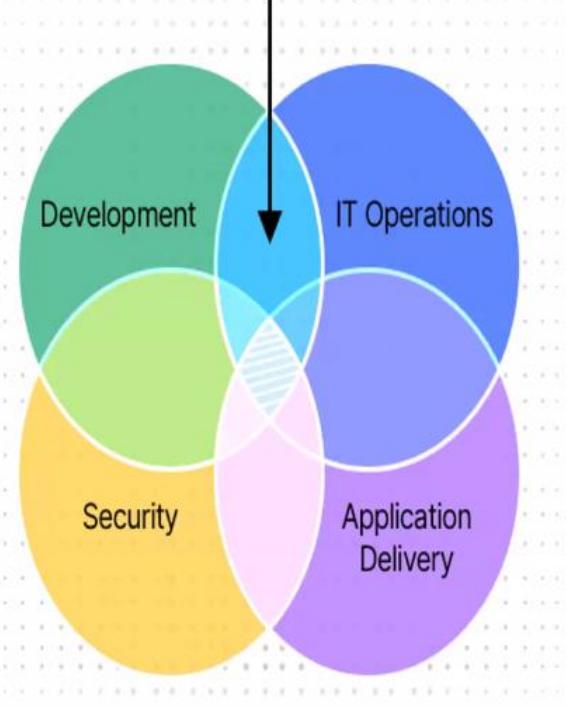
## DevOps Cycle



## DevOps

DevOps è un insieme di pratiche che mira a fornire rapidamente software di qualità superiore integrando i processi tra i team di sviluppo e operativi.

# DevSecOps



## DevSecOps

- DevSecOps (abbreviazione di sviluppo, sicurezza e operazioni) è una pratica di sviluppo che integra iniziative di sicurezza in ogni fase del ciclo di vita dello sviluppo del software per fornire applicazioni robuste e sicure.