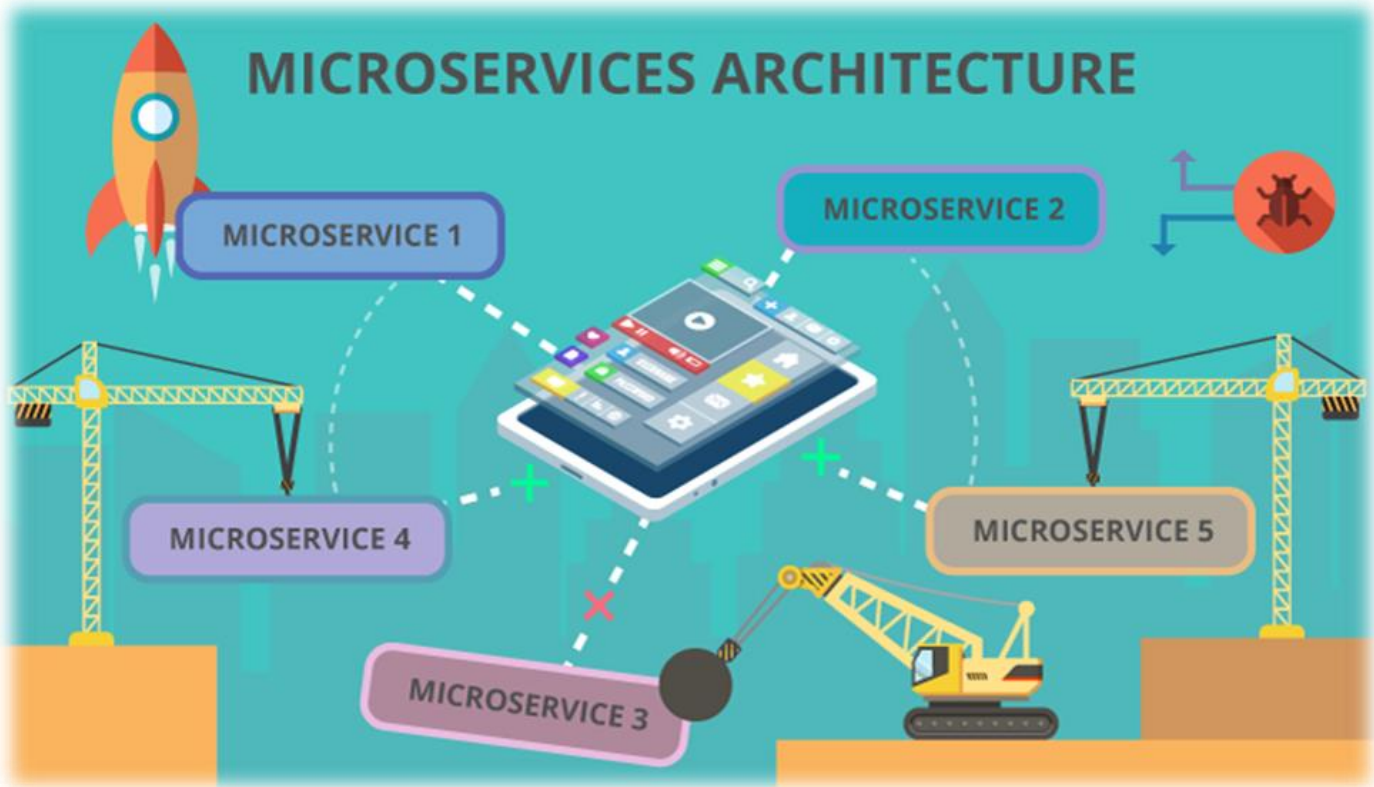

MICROSERVICES ARCHITECTURE
MICROSERVICE 1
MICROSERVICE 2
MICROSERVICE 4
MICROSERVICE 5
MICROSERVICE 3

TOPIC

Descrizione generale

I Microservizi

- I microservizi sembrano il sacro Graal dello sviluppo per i cloud e vengono propagandati come una evoluzione inevitabile di qualsiasi ambiente applicativo.

- Slogan simili ad altri che abbiamo già sentito per molti altri tipi di sviluppo «destrutturato».

- In questo caso, però, l'esperienza maturata dalle grandi imprese che hanno fatto il salto ai microservizi per sostenere il loro sviluppo dimostra che il valore effettivamente c'è e che non stiamo parlando dell'ennesima moda.

- In estrema sintesi, nell'approccio a microservizi le funzioni di una applicazione monolitica sono segmentate in tante piccole applicazioni specifiche (appunto i microservizi) che sono collegate fra loro da API RESTful.

I SOA Vs SOA Microservices

- Nelle architetture a microservizi, le applicazioni sono composte da «numerosi componenti distinti collegati in rete» (i microservi, per l'appunto)

- Lo stile delle architetture dei microservizi è «un'evoluzione delle architetture SOA» (Services Oriented Architecture).

- Le applicazioni realizzate sulla base dei servizi SOA tendono a essere focalizzate sui problemi di integrazione tecnica ove il livello dei servizi implementati è spesso articolato su API (application programming interfaces) tecniche ad elevato livello di granularità.

- Al contrario, l'approccio basato sui microservizi consente di implementare funzionalità aziendali chiare attraverso API aziendali caratterizzate da un livello di granularità inferiore.

I SOA Vs SOA Microservices

- La principale differenza tra questi due approcci risiede nel modo in cui vengono implementati.

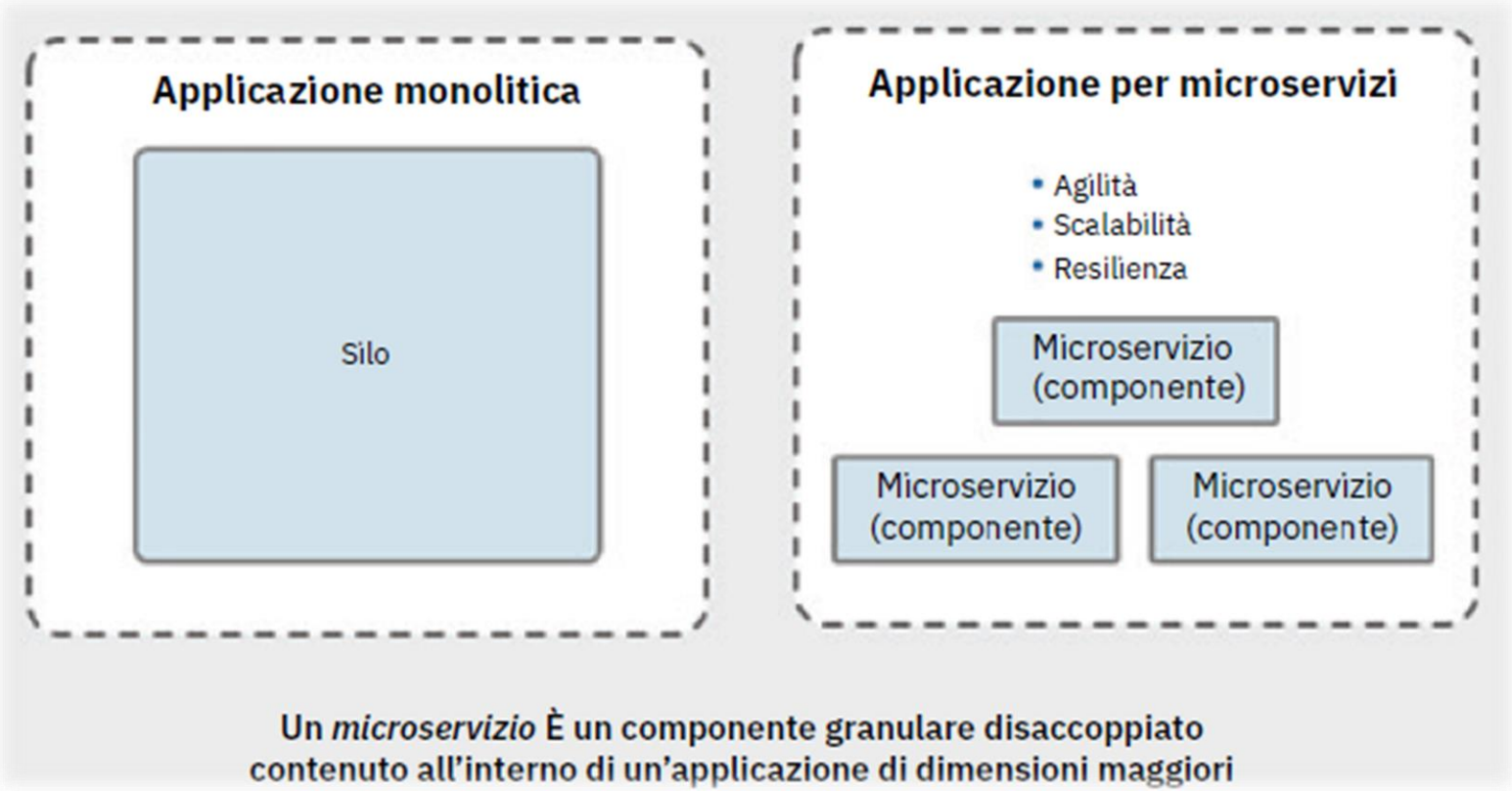
- Da molti anni, le applicazioni vengono realizzate in maniera monolitica, con un team di sviluppatori che realizza una singola applicazione di grandi dimensioni in grado di assolvere a tutte le esigenze di una data azienda.

- Una volta completata, l'applicazione viene implementata più volte su numerosi server delle applicazioni geograficamente dislocati.

- Con il modello di architettura per microservizi, gli sviluppatori realizzano e preparano in maniera indipendente molteplici applicazioni di dimensioni più piccole, ognuna delle quali assolve a funzioni specifiche dell'applicazione principale.

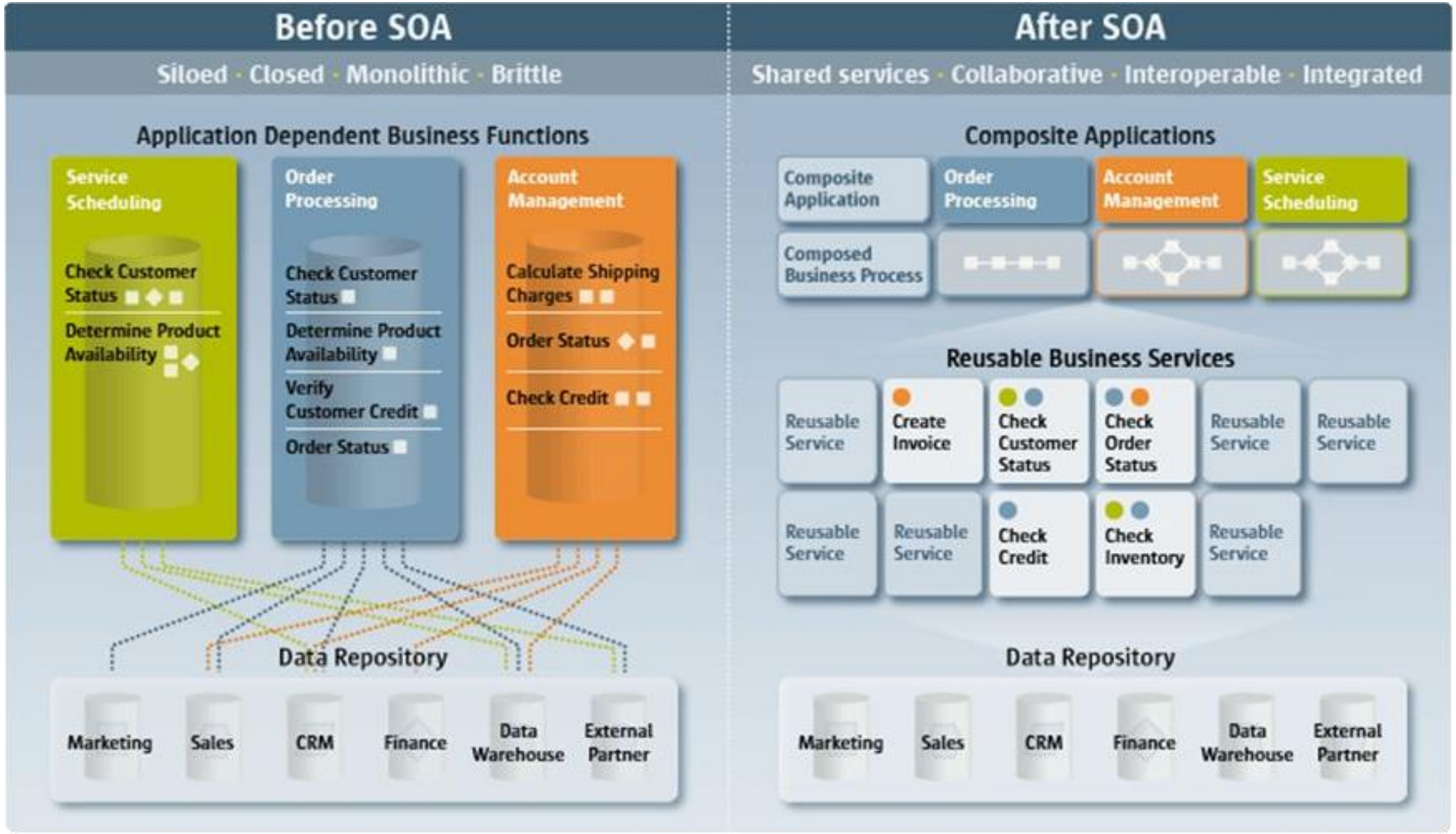
Silos Based Vs SOA Microservices

- In altre parole, le architetture basate sui microservizi abbattono le barriere che ci sono per le grandi applicazioni a silos, rendendole più gestibili e completamente separate

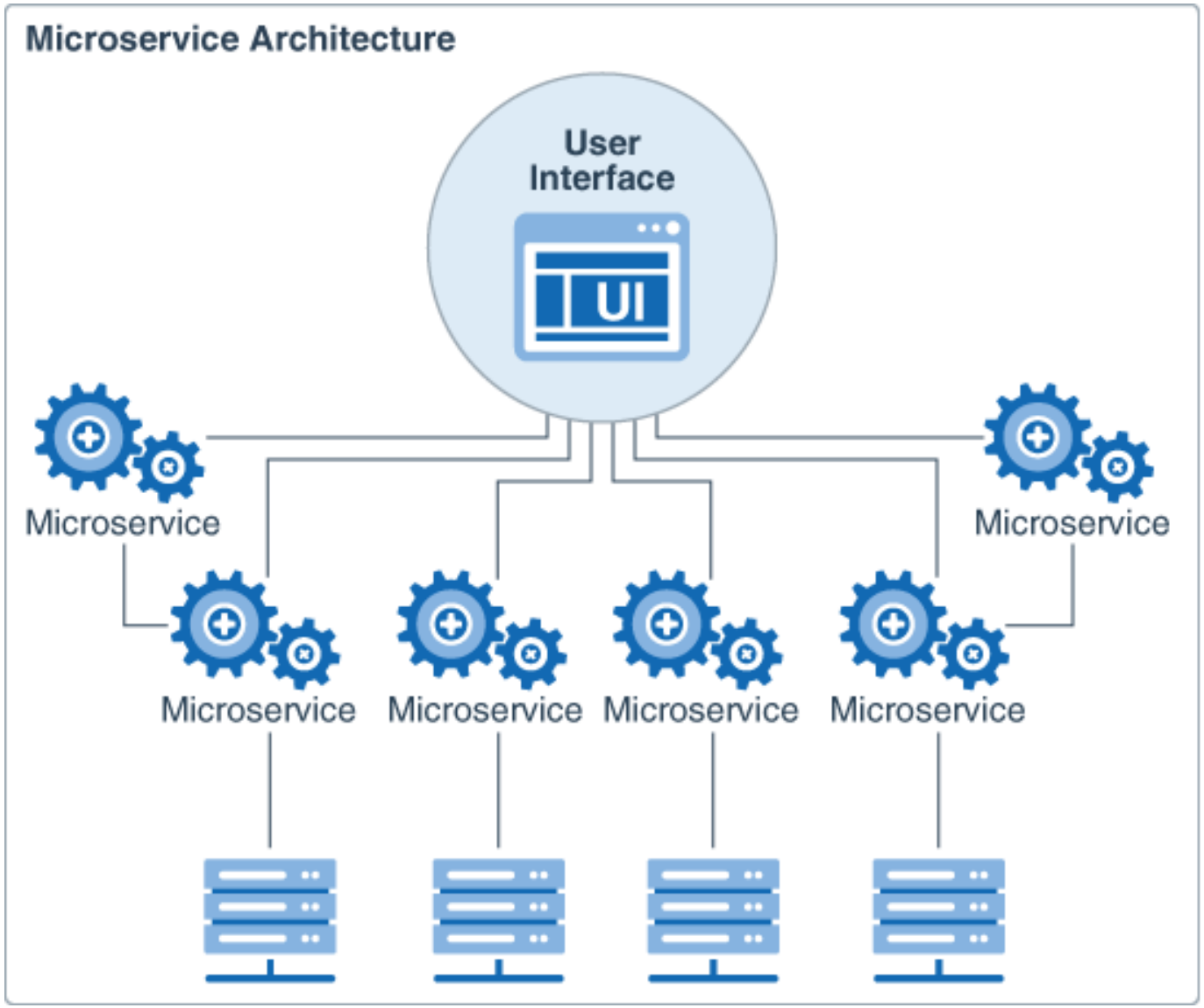


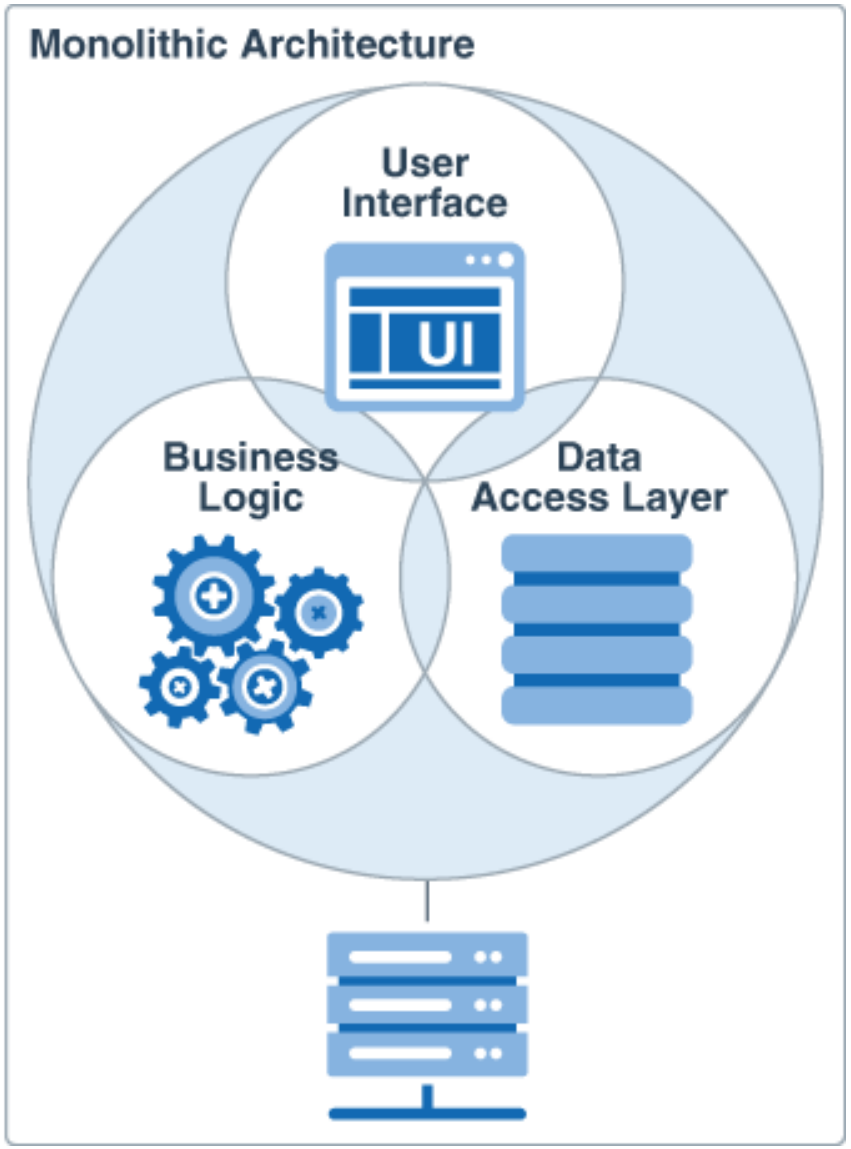
Timeline delle eveoluzioni architetturali

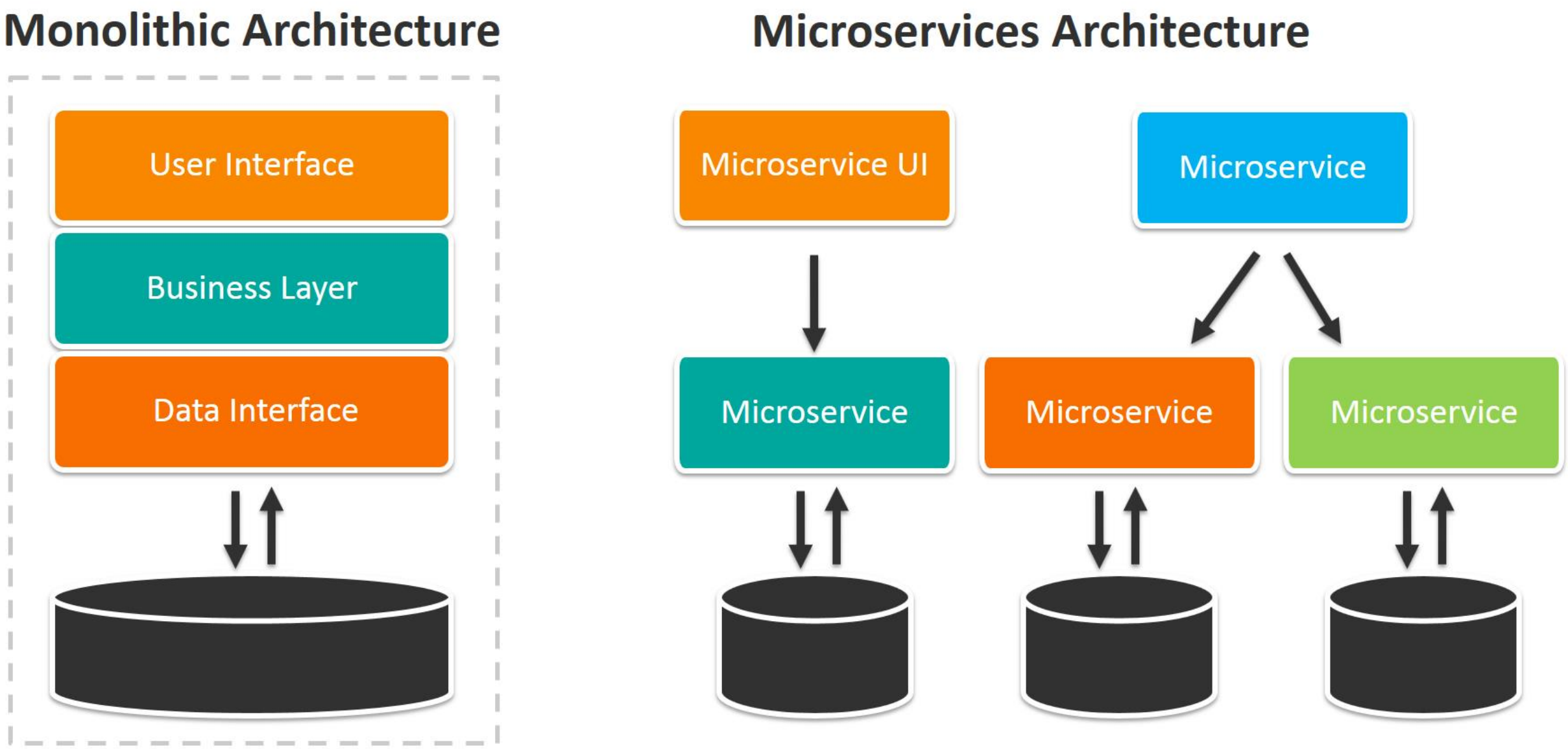
Monolithic vs SOA



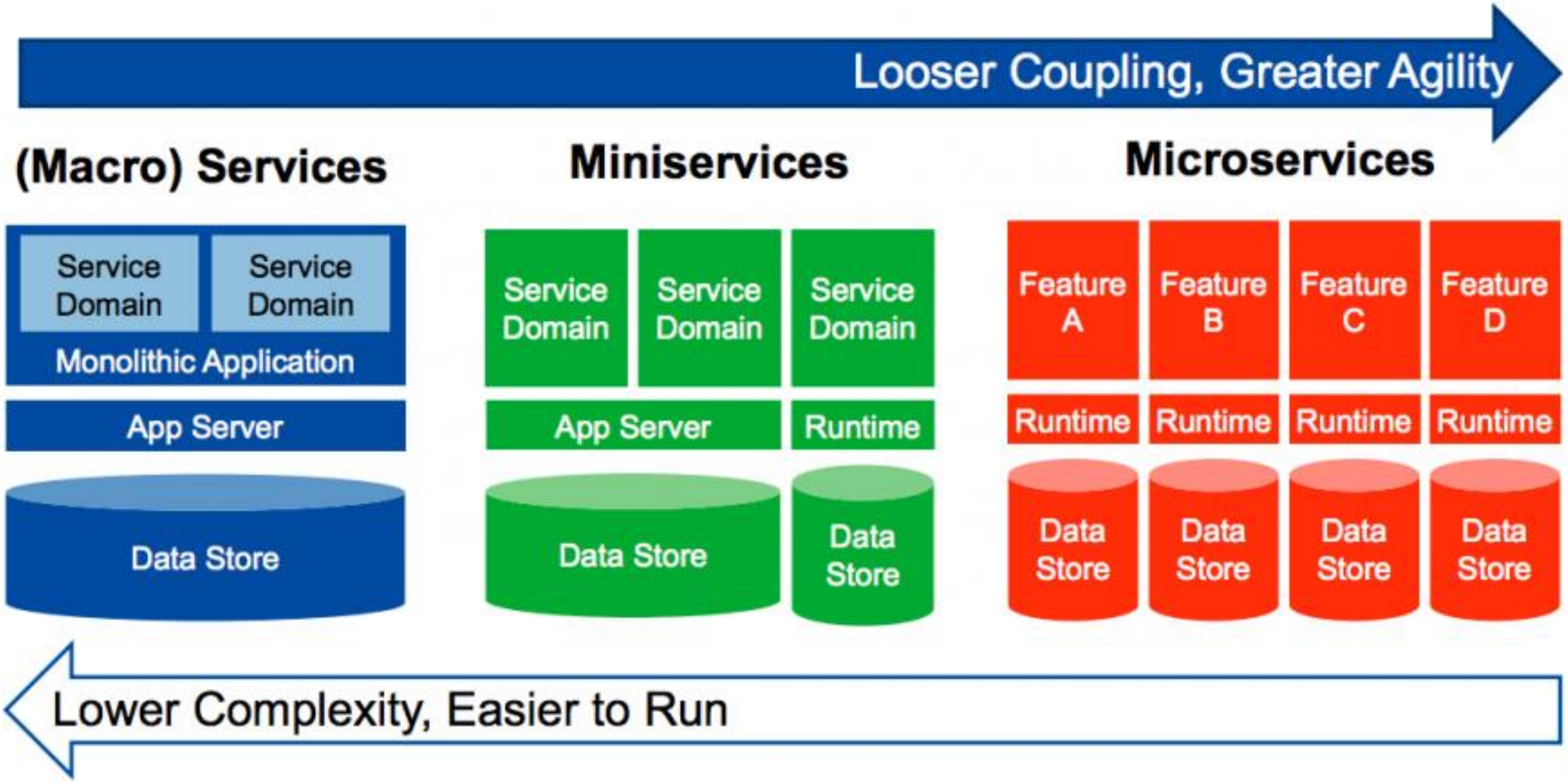
Monolithic vs Microservices Architecture







Looser Coupling & Complexity



TOPIC

Le 5 Regole alla base dell’implementazione delle applicazioni basate su architetture a microservizi

Regola 1

- «Suddividere applicazioni monolitiche di grandi dimensioni in molteplici servizi di dimensioni inferiori»:
  - Un singolo servizio accessibile dalla rete rappresenta la più piccola unità implementabile per un'applicazione di microservizi.
  - Ciascun servizio esegue il suo processo specifico.
  - Questa regola è definita come «un servizio per ciascun container» dove il termine «container» è riferito ai «container Docker» o a qualunque altro tipo di implementazione semplificato, come i runtime di Cloud Foundry.

Regola 2

- «Ottimizzazione dei servizi per singole funzioni»
  - In un approccio SOA monolitico di tipo tradizionale, una singola runtime delle applicazioni esegue molteplici funzioni aziendali.
  - Nell'approccio basato sui microservizi esiste una sola funzione aziendale per servizio.
  - Ciò rende ogni servizio più piccolo e semplice da creare e gestire.
  - Questo sistema viene definito come Single Responsibility Principle (SRP).

- «Comunicazioni tra API REST e broker dei messaggi»

  - Uno degli svantaggi dell'approccio SOA risiede nel fatto che esistono svariati standard e opzioni per l'implementazione dei servizi SOA.

  - L'approccio basato sui microservizi implica rigide limitazioni relative ai tipi di connettività di rete implementabili con un dato servizio per conseguire il massimo livello di semplicità.

  - Analogamente, i microservizi tendono a evitare i tipi di accoppiamenti introdotti dalle comunicazioni implicite attraverso un database.

  - Tutte le comunicazioni tra un servizio e l'altro devono essere effettuate attraverso l'API di servizio, o almeno devono utilizzare un pattern di comunicazione esplicito, come il cosiddetto Pattern Claim Check [Hohpe e Woolf]
    (https://www.enterpriseintegrationpatterns.com/patterns/messaging/StoreInLibrary.html).

- «Applicazione per servizio CI/CD»

  - Nelle applicazioni di grandi dimensioni, composte da numerosi servizi, tali differenti servizi evolvono a velocità differenti.

  - Ciascun servizio utilizza una singola, costante pipeline di integrazione/distribuzione che procede a un ritmo naturale indipendente.

  - Nel caso dell'approccio monolitico ciò non è possibile, in quanto in questo caso ogni aggiornamento del sistema viene rilasciato forzatamente sulla base dei componenti più lenti da aggiornare.

- «Applicazione delle decisioni di alta disponibilità (HA)/clustering per ciascun servizio»

  - Durante la realizzazione di sistemi di grandi dimensioni, il clustering non è basato su un singolo approccio di tipo universale.

  - L'approccio monolitico basato sullo scaling di tutti i servizi dell'architettura monolitica allo stesso livello porta al sovrautilizzo di alcuni server e al sottoutilizzo di altri, o, ancora peggio, tale approccio causa gravi carenze in alcuni servizi da parte di altri servizi che monopolizzano le risorse condivise disponibili, come i thread pool.

  - La realtà è che in un sistema di grandi dimensioni non tutti i servizi devono essere scalati.

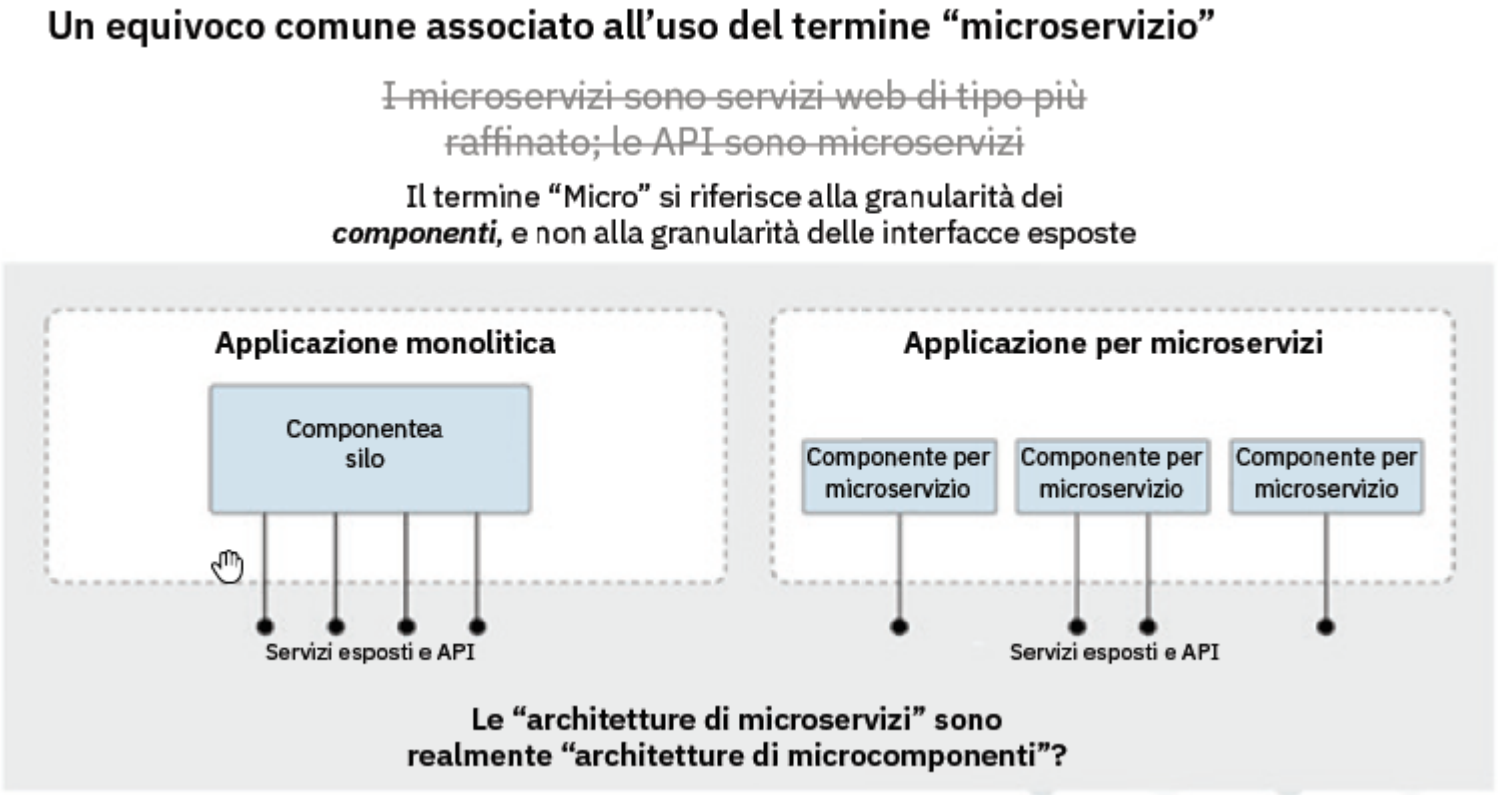
  - Molti di essi possono essere implementati in un piccolo numero di server per preservare le risorse; altri servizi invece necessitano di scaling su piattaforme di grandi dimensioni.

- La combinazione di queste cinque regole e dei loro vantaggi rappresenta la principale ragione per cui le architetture basate sui microservizi sono diventate così popolari. applicazioni commerciali su larga scala; nell'ambito dei microservizi, una definizione di questo nuovo termine per definire le architetture viene fornita da Martin Fowler:

> «*In breve, lo stile che caratterizza le architetture basate sui microservizi è incentrato su un approccio che prevede lo sviluppo di una singola applicazione come suite di piccoli servizi, ciascuno dei quali viene eseguito con un suo processo indipendente e comunica attraverso meccanismi snelli e semplici, spesso utilizzando un'API di risorse HTTP.*
>
> *Questi servizi sono realizzati sulla base delle capacità aziendali e possono essere implementati in maniera indipendente da dispositivi interamente automatizzati.*
>
> *Questi servizi richiedono procedure di gestione centralizzate minime, che possono essere scritte in differenti linguaggi di programmazione e possono utilizzare differenti tecnologie di storage dei dati*».

Osservazioni: Differenza tra API e microservizi

- Una delle principali differenze tra microservizi e altre piattaforme come SOA e API, risiede nel focus verso componenti implementati e componenti in esecuzione; i microservizi sono incentrati sulla granularità dei componenti installati, anziché sulle interfacce, come illustrato in figura sotto.



TOPIC

I 3 Fattori alla base dello sviluppo di microservizi

Le Organizzazioni

- Molte organizzazioni, spinte dalle esigenze operative, realizzano applicazioni monolitiche, in cui use case e funzioni sono implementati in una singola applicazione di grandi dimensioni.

- Sebbene ciò semplifichi alcune operazioni, la modifica di queste applicazioni affinché siano in grado di funzionare come piattaforme di microservizi richiede un grande impegno.

- Sono tre i fattori che sono alla base dello sviluppo a microservizi e di cui devono tener conto le organizzazioni che vogliono introdurlo nei propri processi aziendali.

Fattore 1: Il modo in cui i team sono organizzati

- Lo sviluppo dei microservizi risulta più gestibile quando eseguito con un approccio di ingegnerizzazione incentrato sulla scomposizione di un'applicazione in moduli associati a singole funzioni, con interfacce ben definite, implementabili e gestibili in maniera indipendente da piccoli team che controllano l'intero ciclo di vita del servizio.

- I microservizi accorciano i tempi di implementazione, minimizzando comunicazioni e coordinazione tra le persone e riducendo l'entità e i rischi associati alle modifiche.

Fattore 2: Il modo in cui vengono realizzate le applicazioni

- Le applicazioni basate sui microservizi implicano alcune considerazioni relative al modo in cui esse sono realizzate e agli ambienti in cui vengono eseguire.

- L'ambiente è spesso definito come applicazioni native in cloud o applicazioni a 12 fattori (twelve-factor app – es. App SaaS).

- Un'architettura basata sui microservizi trae vantaggio ed è in grado di compensare gli svantaggi di un ambiente cloud standardizzato, inclusi gli aspetti correlati a flessibilità di scalabilità, immutabilità di implementazione, istanze scartabili e infrastrutture con un ridotto grado di prevedibilità.

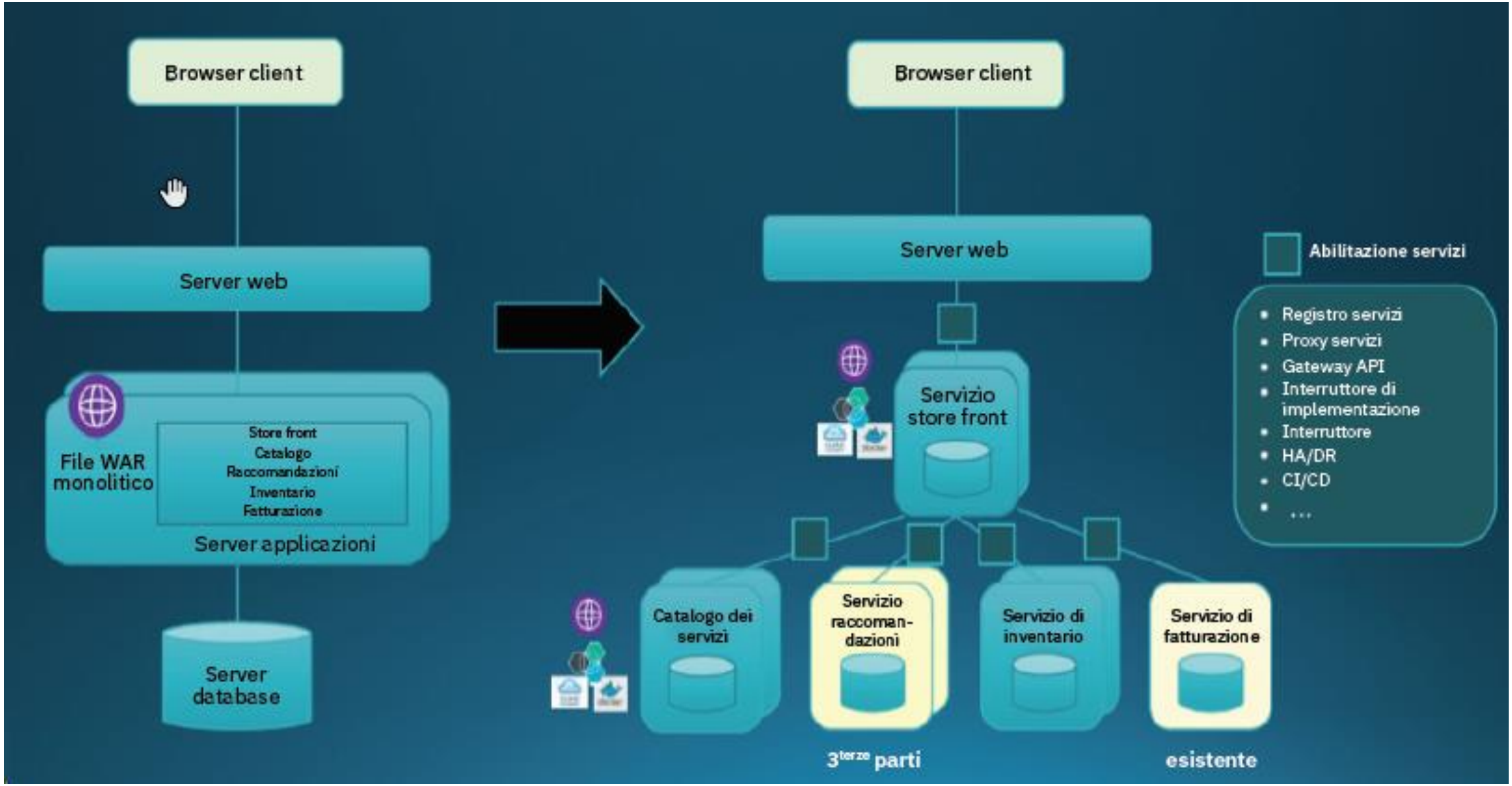
Fattore 3: In che modo le applicazioni sono implementate ed eseguite

- L'uso dei container come metodo standard per l'esecuzione delle applicazioni, è alla base dei metodi di creazione dei pacchetti e di esecuzione delle applicazioni basate sui microservizi.

- I container non sono una nuova tecnologia; essi offrono funzionalità a livello di sistema operativo che rendono possibile l'esecuzione di sistemi operativi Linux multipli (o container) su un singolo host di controllo Linux. I container Linux sono un'alternativa semplificata rispetto alle macchine virtuali complete.

- Anche se i container non rappresentano una novità assoluta, i framework come Docker ne hanno favorito l'adozione, dando vita a un metodo per creare un'immagine per l'esecuzione dei container; ciò offre agli utenti un metodo standard per racchiudere un'intera applicazione con tutte le relative dipendenze, con la possibilità di spostarla tra un ambiente e l'altro ed eseguendola senza alcuna modifica. Altri framework come Cloud Foundry utilizzano i container per eseguire le applicazioni, eliminando però le astrazioni correlate alla virtualizzazione.

Evoluzione dell'architettura monolitica in una basata sui microservizi



Osservazioni

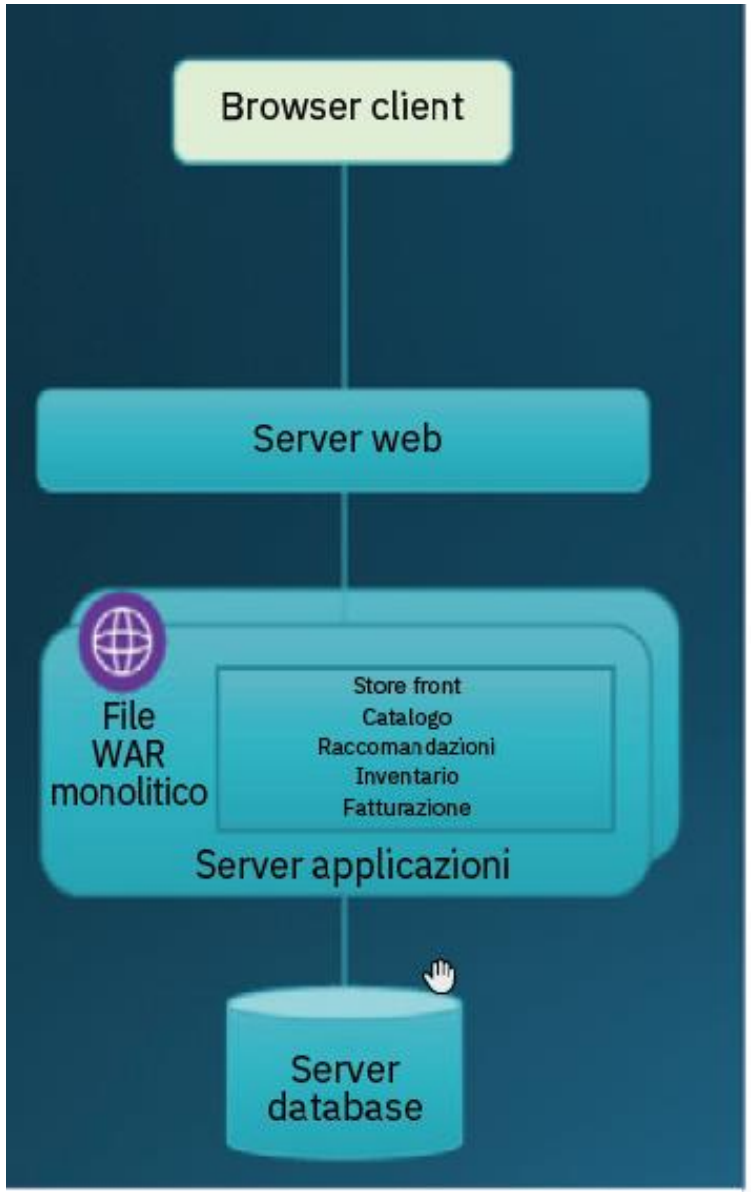
- La figura precedente mostra un singolo archivio aziendale che ospita tutti i componenti di un sito web che ospita uno shop al dettaglio.
- L'archivio delle applicazioni contiene tutte le funzioni aziendali, come catalogo e inventario, unitamente a logica del sito web, logica aziendale e logica di persistenza per ciascun componente.
- Inoltre, l'applicazione condivide un singolo database, che spesso contiene modelli di dati accoppiati e condivisi con altre applicazioni.
- Si noti come, sul lato dell'immagine, le funzionalità aziendali siano ora implementate in applicazioni separate, ciascuna delle quali utilizza i suoi dati specifici.
- Questo nuovo stile architetturale introduce complessità associate alle comunicazioni tra microservizi, proprietà dei dati, sincronizzazione e resilienza dei dati.

Utilizzando l’esempio riportato, un riepilogo dei principali aspetti include:

- «Componentizzazione attraverso i servizi»: Questo aspetto viene discusso nelle precedenti sezioni del documento.

- «Organizzazione sulla base delle capacità aziendali»: L'aspetto relativo alla nozione di team di sviluppo è stata discussa in precedenza; l’aspetto relativo all’organizzazione sulla base delle capacità aziendali richiede un cambiamento del modo di pensare nell’ambito dei team di sviluppo.

Team di sviluppo organizzati in sili



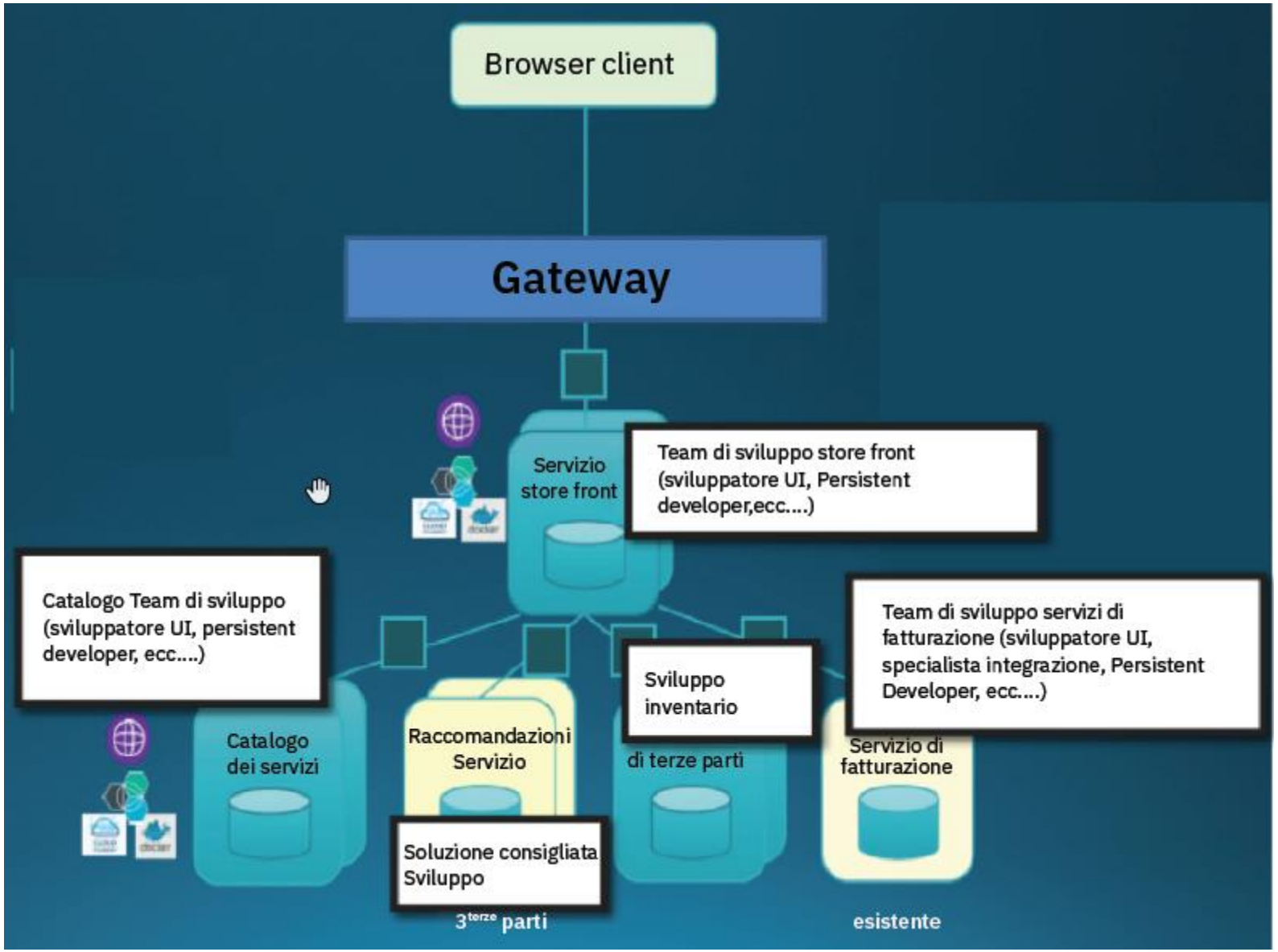
Team di sviluppo organizzati in silos

Sviluppatori di UI

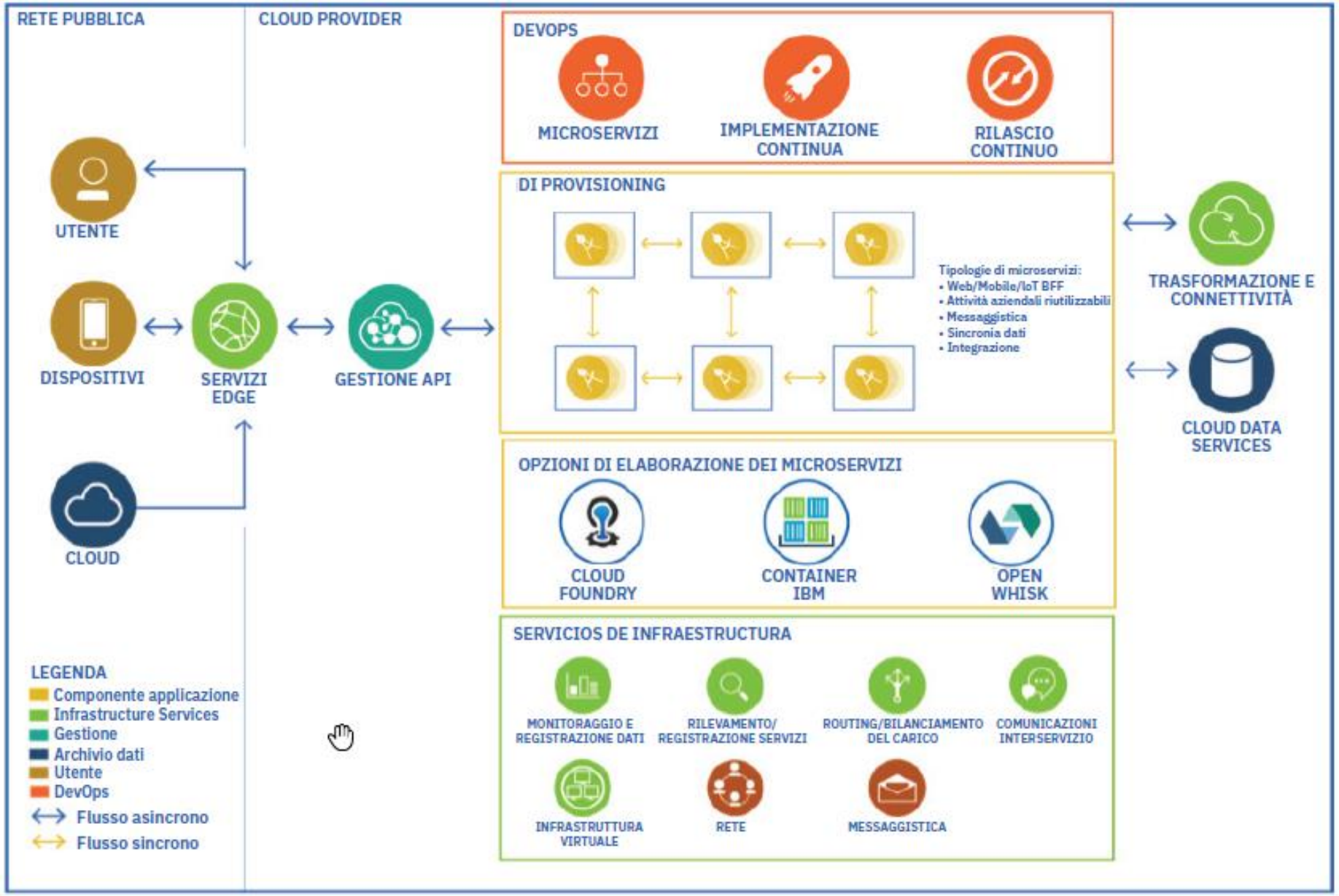
Sviluppatore middleware

DBA

Team di sviluppo organizzati in base alle capacità aziendali



Capacità di un’architettura basata sui microservizi



TOPIC

Approfondimenti

I Microservizi

- Ogni microservizio è distinto e isolato, nel senso che ha un suo database e non condivide dati con gli altri. Ciò permette di sviluppare microservizi dotati della base dati più opportuna a ciò che devono fare, senza il peso di una struttura comune.

- Ammettiamo di voler realizzare il portale ecommerce di un negozio.

- Nell'approccio tradizionale svilupperemmo una mega-applicazione partendo dalla struttura della base dati, collegata quasi certamente a un sistema relazionale che conterrà tutte le informazioni necessarie (prodotti, utenti e via dicendo) separate nelle varie tabelle.

- A partire dalla struttura delle informazioni costruiremmo la logica applicativa che le gestisce e, infine, il codice che serve all'interfaccia e a tutta l'user experience.

- L'approccio a microservizi è esattamente opposto: in un certo senso si parte dalla fine.

- Pensiamo al prodotto completo (il portale di ecommerce) e scomponiamolo nelle sue funzioni logiche.

- Serve certamente un servizio che mostri la scheda di un prodotto, come uno per completare i pagamenti, uno per gestire la registrazione di un nuovo utente, uno per accettare i commenti a un prodotto…

- Le funzioni possono essere decine e anche di più.

La logica della scomposizione

- I vantaggi di questo «smantellamento» delle applicazioni monolitiche - o delle applicazioni che si sviluppano da zero con questo modello - sono principalmente collegati alla elasticità e scalabilità del sistema e all'organizzazione dello sviluppo.

- Tutti i microservizi possono - e dovrebbero, infatti - essere sviluppati in maniera indipendente perché sono concepiti come entità autonome e isolate.

- Quindi nello sviluppo di uno di essi non c'è bisogno di preoccuparsi di cosa c'è nel resto dell'applicazione perché non influenza le scelte progettuali del singolo servizio.

- Se per la registrazione degli utenti basta un semplice database NoSQL, ad esempio, lo si adotta anche se la gestione degli articoli in vendite richiede magari una piattaforma diversa.

La logica della scomposizione

- Allo stesso modo, il servizio che accetta un commento a un prodotto non si deve preoccupare dell'identificazione di chi lo sottopone, che sarà gestita da un altro. Una conseguenza di questo è che apportare modifiche a una applicazione scomposta in microservizi è semplice e veloce.

- Si interviene solo sui microservizi coinvolti, senza dover mettere mano a migliaia di righe di codice alla ricerca dei punti in cui fare modifiche.

- Un altro elemento importante abilitato dalla logica dei microservizi è che, essendo questi isolati e autonomi, vi si possono associare le risorse più indicate, invece di pensarle per tutta un'applicazione nel suo complesso.

- Questo vale in particolare negli ambienti virtualizzati e PaaS, luogo ideale dove far «vivere» i microservizi (ma non necessariamente l'unico, l'approccio è generico).

La logica della scomposizione

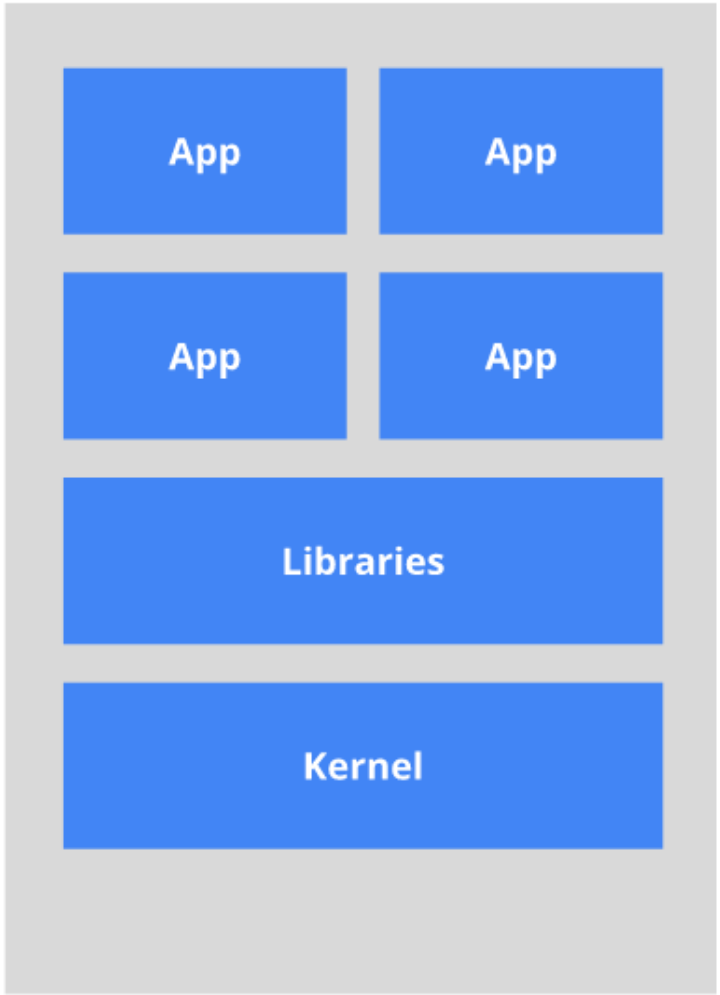
- Il concetto di microservizio si lega bene a quello di container: entrambi sono componenti «atomici» e stateless.

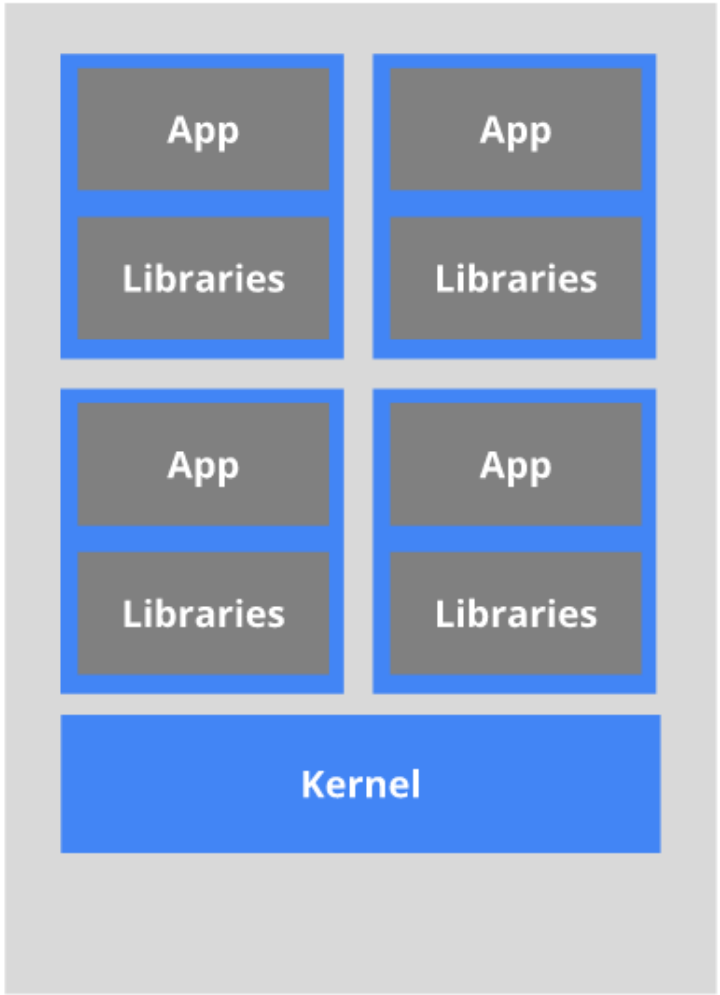
- Per questo appare solo logico incapsulare un microservizio in un container e gestire un ambiente virtualizzato in modo che il microservizio scali orizzontalmente attivando tante istanze quante ne servono in un certo momento, dotandole delle risorse più opportune.

- Proseguendo nel nostro esempio, è prevedibile che il microservizio collegato alla visualizzazione di un prodotto sia usato molto più intensamente di quello che gestisce la registrazione di un utente.

- Un ambiente PaaS pubblico o privato permette di adeguarsi a queste differenze.

Microservizi e Container





L’altro lato della medaglia

- Sarebbe troppo bello se tutti questi vantaggi non implicassero qualche complessità. E infatti lo sviluppo a microservizi ha i suoi punti critici.

- Anzi, essenzialmente uno: come l’applicazione monolitica viene disgregata in più servizi, anche il flusso dell’applicazione si scompone in vari passi che restano interdipendenti ma sono delegati a moduli autonomi e scollegati. Controllarlo diventa molto più difficile.

- Quando si verifica un problema nel flusso delle operazioni distribuite tra più microservizi - magari decine - diventa complesso capire chi ha causato cosa.

- Non c’è più la possibilità di scorrere il codice monolitico per eseguire magari un debug e, prima o poi, arrivare al punto critico che non ha funzionato a dovere.

L’altro lato della medaglia

- Strumenti e servizi dedicati al debug vero e proprio dei microservizi ve ne sono pochi.

- Si stanno sviluppando, è però probabile che chi passa ora ai microservizi dovrà assemblare da solo una propria cassetta degli attrezzi per affrontare questo aspetto che non è il solo, perché anche se i microservizi sono stati sviluppati a regola d’arte devono anche funzionare correttamente insieme.

- Questo rimanda al tema dell’orchestration, che è più noto e offre piattaforme già ben definite (Kubernetes su tutte).

I Microservizi

- Riepilogando abbiamo visto come i microservizi sono la logica applicativa preponderante per lo sviluppo di nuove applicazioni, quelle cosiddette cloud-native, e anche per l'integrazione delle applicazioni tradizionali agli ambienti più moderni.

- Però creare applicazioni e servizi basandosi sulla nuova logica non è semplice, perché scomporre un'applicazione monolitica in una collezione di microservizi impone agli sviluppatori di tenere conto di elementi specifici e prima non presenti, come la discovery dei servizi stessi, la tolleranza ai guasti della rete, la sicurezza e la compliance.

- Non esiste un approccio unico a queste problematiche e nemmeno può essere imposto, perché in molti casi non è possibile intervenire sul codice delle applicazioni preesistenti per modificarlo in tal senso.

- La soluzione potrebbe essere invece inserire un nuovo «strato» infrastrutturale tra i servizi applicativi e la rete.

- È la strada che Google, IBM e Lyft hanno deciso di seguire dando vita al progetto open source Istio (https://cloud.google.com/istio/?hl=it).

- Istio serve a creare quella che le tre aziende definiscono una «service mesh». È uno strato logico che separa gli aspetti relativi alla gestione dei servizi da quelli che interessano invece gli sviluppatori, in modo che le relative attività possano andare avanti in parallelo.

- Ad esempio, Istio presenta a chi gestisce le infrastrutture dei microservizi le informazioni necessarie a visualizzare lo stato delle applicazioni, dei microservizi e del traffico dei loro dati in rete.

- Questi dati servono a evidenziare eventuali criticità dell'architettura dei servizi ed evitare, o almeno identificare rapidamente, i colli di bottiglia.

- Il vantaggio per gli sviluppatori sta nel fatto che tutta questa parte di monitoraggio e gestione non richiede nessun contributo da parte loro perché non si trova nel codice dei microservizi.

I Microservizi

- Lo sviluppo è astratto dalle caratteristiche operative della rete e dell'insieme di servizi che «crea» un'applicazione.

- È Istio che si occupa di elementi come il routing intelligente verso i servizi, il bilanciamento del loro carico, la gestione di eventuali policy, la telemetria.

- Lo fa dialogando con Kubernetes, al momento, perché la sua prima versione è pensata per le applicazioni containerizzate.

- Il progetto Istio (il termine vuol dire vela, in greco) nasce principalmente dalla collaborazione tra Google, IBM e Lyft ma vede coinvolte anche Red Hat, Pivotal, Weaveworks, Tigera e Datawire.

- Non di meno sono ineressanti soluzioni che adottano sistemi ESB come WSO2 e consimili che fungono da «BUS» all'interno del proprio ecosistema software.

Cosa serve per abilitare un’architettura a msvc

- Organizzare le proprie applicazioni adottando una architettura modulare a microservizi ha diversi vantaggi e pone qualche problema a cui gli sviluppatori tradizionali non sono abituati.

- Il trend verso l’adozione dei microservice peraltro è ormai evidente e la gran parte delle aziende sta quantomeno considerando di adottarli in qualche misura.

- Gli analisti avvisano che prima di approcciare questo mondo bisogna avere chiaro di che si tratta e sviluppare una minima competenza nella gestione dei suoi elementi principali, in particolare alcune tecnologie che inevitabilmente ruotano attorno ai microservizi.

- Vediamone alcune.

Cosa serve per abilitare un'architettura a msvc

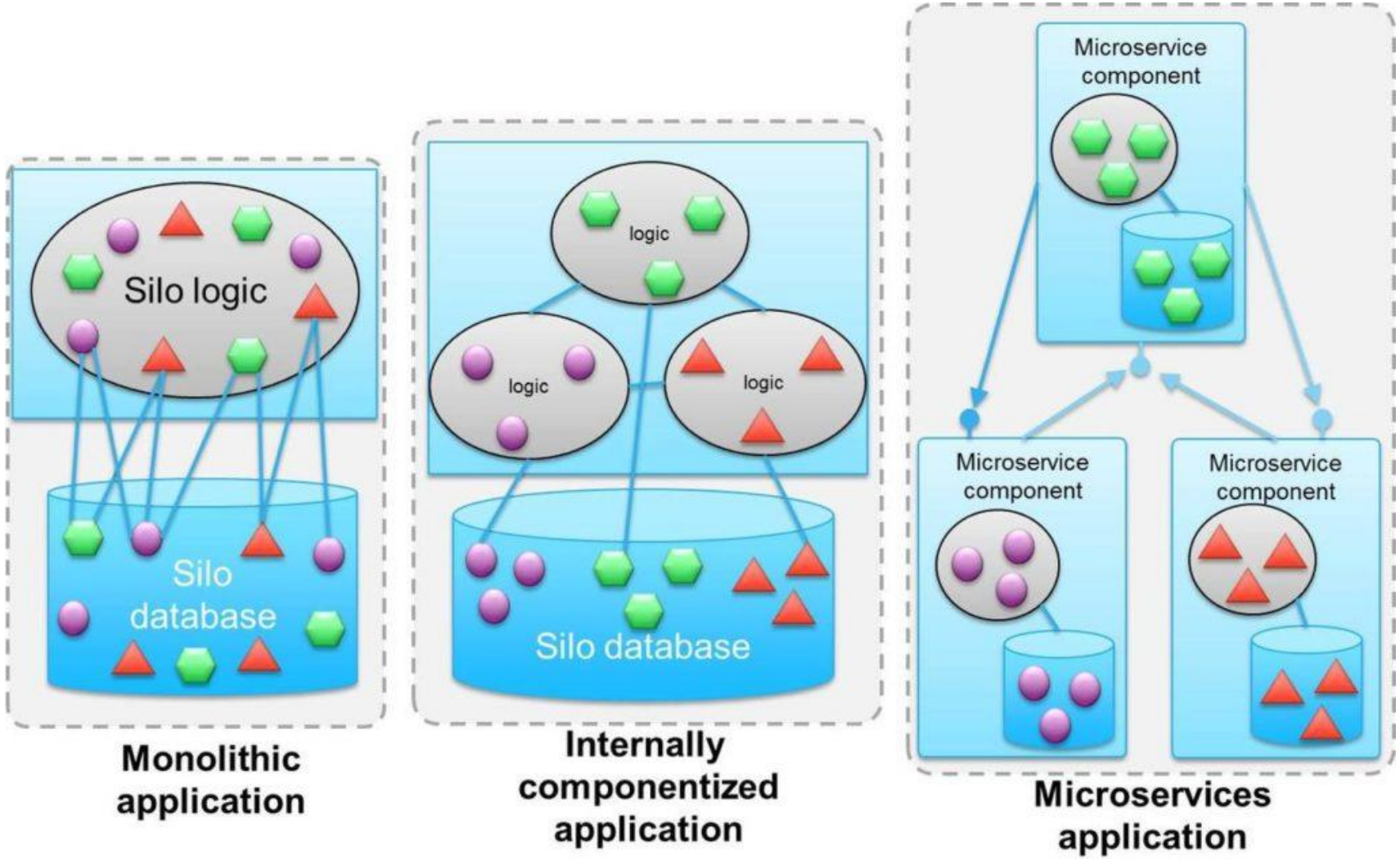
- Il legame tra microservizi, ambienti cloud e virtualizzazione a container è stretto e anche abbastanza ovvio.

- Un'architettura a microservizi si può creare anche senza cloud - c'è chi in fondo considera i microservice banalmente come l'evoluzione delle Service-Oriented Architecture, e di cloud non si parlava nell'epoca d'oro delle SOA - ma nella gran parte dei casi i microservizi sono la base delle applicazioni cloud-native.

- D'altro canto tutti i cloud provider offrono la possibilità di realizzare microservizi sui propri cloud e, più in generale, qualsiasi ambiente basato su OpenStack li supporta.

- Da notare poi che i cloud provider stanno facendo evolvere la loro offerta in campo serverless computing, una concezione dello sviluppo che si integra bene con il modello dei microservice.

Modelli Architetturali



Cosa serve per abilitare un'architettura a msvc

- Più tradizionalmente i microservizi si realizzano preferibilmente come elementi «containerizzati», anche perché tra container e micoservice ci sono molte similitudini concettuali.

- Quando vi trovate con una architettura di decine di microservizi che insieme devono realizzare il flusso logico che avete previsto, l'informazione principale che volgiamo sapere è quale microservice sta facendo cosa in un dato momento e se l'insieme dei servizi si sta effettivamente comportando come gli sviluppatori vogliono.

- Il che significa due cose:

  - Orchestration

  - Monitoring delle applicazioni.

Cosa serve per abilitare un'architettura a msvc

- L'Orchestration dei microservizi e quindi dei container che li abilitano è un elemento importante in una architettura applicativa destrutturata.

- Senza funzioni di orchestrazione diventa molto difficile capire cosa eventualmente non va, di specifico, se il (macro)servizio non sta funzionando.

- I tool per farlo, collegati alla containerizzazione, ci sono: ad esempio

  - Kubernetes,
  - Mesos
  - Docker Swarm.

Cosa serve per abilitare un'architettura a msvc

- La situazione cambia se ci portiamo a un livello di astrazione superiore e cerchiamo di valutare le performance di un'applicazione "scomposta" in microservizi.

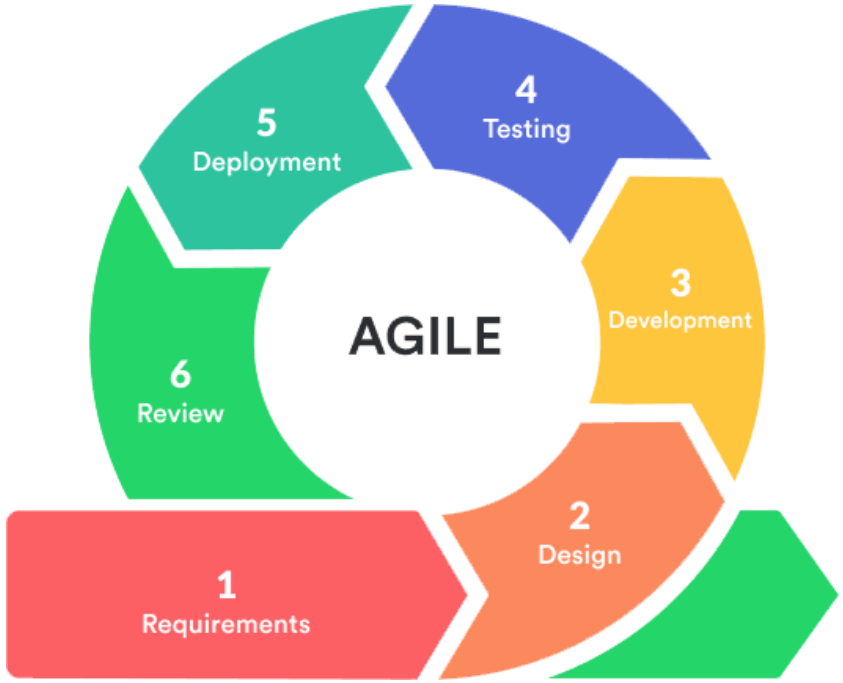
- Qui le soluzioni tradizionali di Application Performance Monitoring non ci aiutano perché sono state pensate per applicazioni monolitiche e non sempre riescono ad adattarsi alle architetture a microservice.

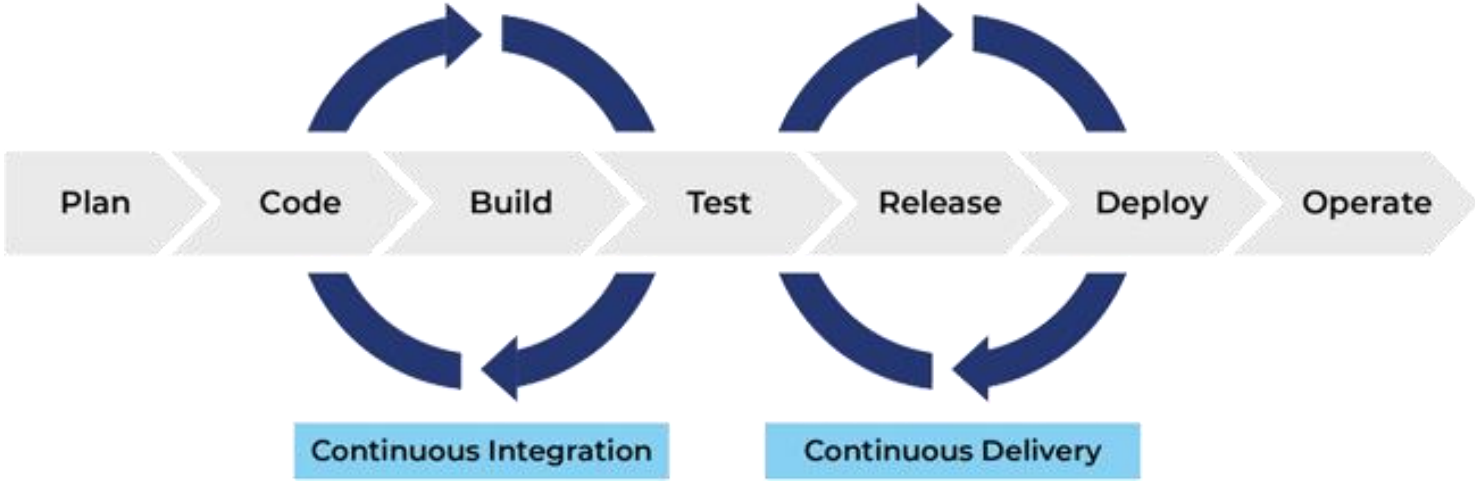
- I vendor del mondo APM stanno ovviamente lavorando al potenziamento dei loro tool in ottica microservice, lo scenario è quantomeno fluido.

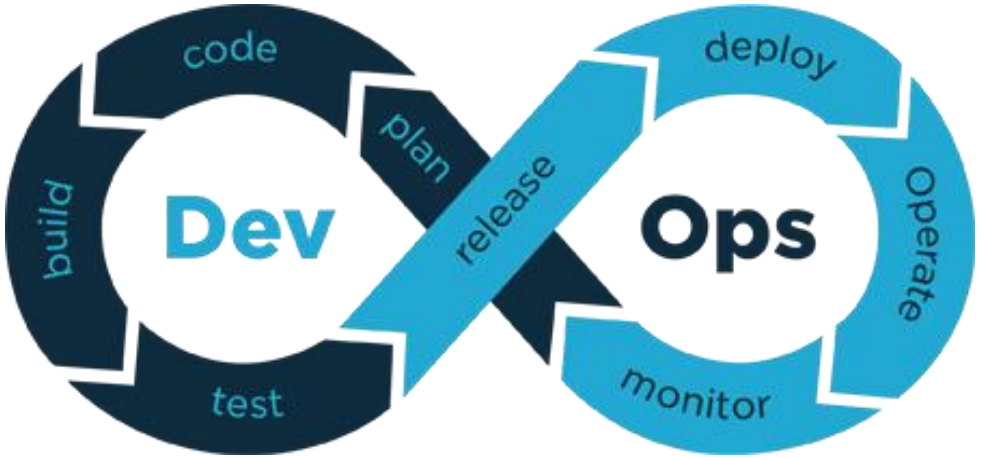
TOPIC

DevOps

- Al fine di creare un'architettura dei microservizi di successo, comprendente provisioning, implementazione continua e rilascio continuo, è necessario sviluppare una solida strategia di automazione mediante DevOps.







DevOps: Provisioning

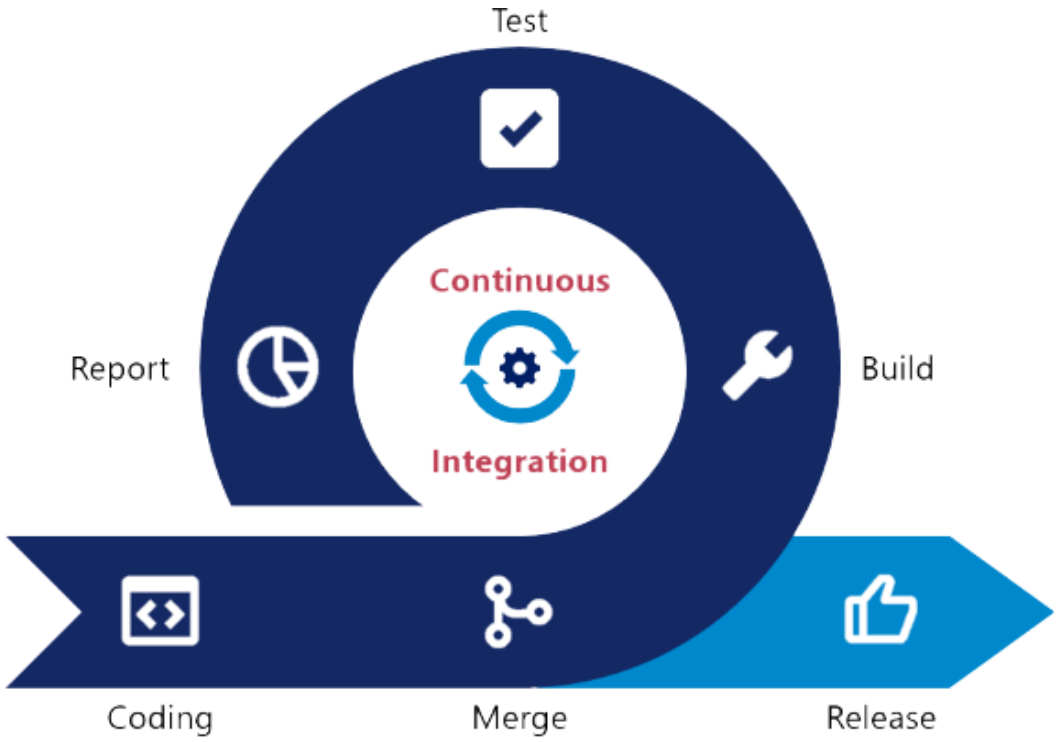
- Un’architettura per microservizi di successo richiede il provisioning automatico per gli ambienti delle applicazioni. Il layer della Platform as a Service normalmente fornisce questo servizio attraverso una serie di servizi gestiti, come i CaaS (Container as a Service), su IBM® Cloud®.

- Se si esegue un’infrastruttura basata su container sopra un layer IaaS (Infrastructure as a Service), utilizzando motori di orchestrazione Docker, come Kubernetes o Docker Datacenter (DDC), è necessario automatizzare il processo attraverso il quale tali ambienti sono implementati e aggiornati, mediante processi di provisioning automatico che risiedono in un ambiente basato su macchine virtuali.

DevOps: Implementazione continua

- Gli ambienti di sviluppo richiedono un processo automatizzato di versioning e implementazione delle immagini Docker o delle applicazioni per i microservizi.

- Se si dispone di una strategia multi-Cloud, il processo di rilascio e implementazione deve essere in grado di astrarre le differenze relative ai metodi utilizzati per eseguire le varie operazioni, come auto-scaling o regole per il cloud.



Dal punto di vista degli sviluppatori

- Chi adotta l'approccio a microservizi poggia su due elementi che fortunatamente sono ben noti: le API e Rest.

- Ogni microservice idealmente ha la sua API tramite cui interagisce con altri servizi e che può essere presentata all'esterno (esposta) per far usare il servizio da qualsiasi applicazione.

- Questo permette una grande elasticità di sviluppo perché permette di collegare una propria applicazione con qualsiasi microservice, l'unico problema è che quando le API gestite o presentate all'esterno crescono di numero si rende quasi obbligatoria una soluzione di API management.

Dal punto di vista degli sviluppatori

- Rest (o Representational State Transfer) è una vecchia conoscenza degli sviluppatori web ed è il sistema più comunemente adottato per la comunicazione tra microservizi, una scelta logica anche perché la maggior parte delle attuali applicazioni a microservice sono applicazioni web.

- In questo approccio i microservizi dialogano semplicemente via Http e si scambiano informazioni in formati standard come XML, Html o sempre più spesso Json.

- Fin qui tutto bene.

Dal punto di vista degli sviluppatori

- Ciò con cui alcuni sviluppatori possono avere meno confidenza è che per sfruttare appieno l'elasticità offerta dai microservizi è opportuno adottare modelli di sviluppo di tipo DevOps.

- Questo significa avere assimilato anche i concetti di Continuous Integration (CI) e Continuous Delivery (CD) e aver adottato tool specifici che automatizzino la gestione e il rilascio del codice.

- Per chi deve prendere la mano con lo sviluppo «agile» le piattaforme più diffuse sono:
  - Jenkins
  - Hudson
  - Chef.

Come fare DevOps: primo step, la chiarezza

- Da qualche tempo si ritiene che l'approccio tradizionale «a cascata» per lo sviluppo e l'implementazione di applicazioni in azienda non sia adatto a conseguire l'elasticità operativa che oggi tutte le imprese cercano.

- Questo modello prevede una distinzione netta tra i team di sviluppo e quelli delle operations, ossia coloro che gestiscono l'IT aziendale.

- Ad esempio, nella creazione di una nuova applicazione gli sviluppatori ne recepiscono le specifiche, scrivono codice, testano le varie versioni del software e poi rilasciano quella definitiva. A questo punto le operations la implementano e, se tutto va bene, badano in seguito alla sua manutenzione.

- Un approccio rigido si ritiene oggi non più applicabile, e DevOps si presenta come la risposta alla scelta di evitare questa rigidità.

Come fare DevOps: primo step, la chiarezza

- La rigidità del modello a cascata è legata in primo luogo alla separazione delle varie classi di attività e delle persone che le portano avanti. È una distinzione storica e che ha le sue ragioni perché i compiti di analisi, sviluppo, implementazione e manutenzione sono diversi e richiedono skill differenti.

- Ma specie nelle grandi imprese è diventata spesso una separazione quasi burocratica che rallenta tutto il ciclo di sviluppo.

- Ad esempio, il rilascio nell'ambiente di produzione di un nuovo software solo alla fine del ciclo di sviluppo evidenzia tardi eventuali problemi che negli ambienti di test non si potevano verificare. Così bisogna ritornare a macinare codice intervenendo su un prodotto finito e non su uno ancora in evoluzione.

Come fare DevOps: primo step, la chiarezza

- L'approccio DevOps fa in modo che la parte di sviluppo (development) e le operations interagiscano molto più strettamente - da qui l'acronimo DevOps - per lavorare tutti meglio.

- In realtà l'approccio concettualmente non è nuovo: è una estensione del modello di sviluppo «agile» e si può considerare più in generale come ispirato dai modelli «lean» nati per i processi produttivi del manufacturing.

- L'idea infatti è generale: un flusso produttivo (in questo caso dallo sviluppo alla gestione del software) si può rendere più efficiente con una maggiore integrazione dei vari elementi della sua catena.

Come fare DevOps: primo step, la chiarezza

- L'approccio DevOps abbandona lo sviluppo monolitico e sequenziale e adotta cicli molto più brevi e frequenti di sviluppo e implementazione, anche delle versioni preliminari del software o del servizio su cui si sta lavorando.

- In sintesi e semplificando, il team di sviluppo crea una successione frequente di versioni sempre più complete del software o del servizio, mettendole di volta in volta in un ambiente di (quasi) produzione e recependo immediatamente le segnalazioni e i feedback del team di operations.

I vantaggi di DevOps

- I vantaggi del modello DevOps sono diversi. Innanzitutto avere feedback pratici sin dalle prime fasi di sviluppo permette di capire subito in che direzione muoversi.

- Dato poi che l'iterazione tra le varie versioni è veloce, la quantità di codice su cui intervenire per correggere errori e ottimizzare il risultato è minore.

- Per questo motivo le versioni dei software sono anche generalmente più stabili e con meno bug, dato che i nuovi elementi introdotti di volta in volta sono contenuti.

- Nel complesso tutto questo porta a una riduzione dei tempi di rilascio definitivo, a una riduzione anche dei costi di sviluppo (che si stima intorno al 20 percento) e a risultati più soddisfacenti in termini di qualità del prodotto.

I vantaggi di DevOps

- È tutto così ovviamente positivo che viene da chiedersi perché non ci si è pensato prima.

- Ma DevOps è figlio dei suoi tempi e non potrebbe essere altrimenti.

- Concettualmente non sarebbe possibile se non si fossero assimilati prima i modelli di sviluppo agile e di Continuous Delivery e tecnicamente sarebbe molto più complesso da realizzare senza la presenza di piattaforme relativamente recenti.

- Ad esempio quelle di virtualizzazione e containerizzazione, come Docker, che facilitano e velocizzano le fasi di implementazione e test in ambienti controllati ma realistici e le piattaforme di orchestration come Jenkins, che permettono di automatizzare e integrare potenzialmente tutti i passi del processo di sviluppo-test-rilascio.

Semplicità apparente

- L'idea è evidentemente buona, gli strumenti ci sono, eppure l'adozione del modello DevOps resta un problema per molte imprese, indipendentemente dalla loro dimensione.

- Non è strano e si capisce perché, se ci si astrae dalle sigle e dagli acronimi e si guarda al ciclo di vita del software per quello che è: un processo articolato e complesso che comprende aspetti sì tecnologici ma anche organizzativi e strategici.

- Il primo ostacolo per molte imprese è che DevOps innanzitutto è un cambiamento culturale che deve interessare developers e operations. I primi devono imparare qualcosa di IT management e i secondi qualcosa dello sviluppo, tutti devono imparare a collaborare insieme e a «vivere» un modello fatto di esperimenti e test frequenti e anche insuccessi (seppure parziali) frequenti.

Semplicità apparente

- Nelle organizzazioni «a silo» in cui sviluppo e operations sono storicamente separati, è tutt'altro che banale. E per questo ci deve essere un impegno chiaro non solo della parte IT e delle operations ma proprio del top management.

- Secondo punto chiave: DevOps non è una ricetta univoca. Non si «fa DevOps» semplicemente perché si usano le piattaforme che vanno per la maggiore nelle presentazioni tecniche. Bisogna partire considerando i propri processi di sviluppo e gestione e capendo in che modo possono essere ottimizzati grazie al modello DevOps. Può darsi che gli strumenti più di tendenza siano quelli adatti, può anche darsi che altri siano più efficaci.

- Di solito, avvisano gli esperti di settore, conviene farsi aiutare in questa fase di auto-valutazione, come si farebbe per l'ottimizzazione di qualsiasi processo.

Semplicità apparente

- Se non si tengono presenti questi punti il rischio è andare alla ricerca della «soluzione DevOps» che si ritiene adatta alle proprie caratteristiche e adottarla come se fosse un nuovo ambiente di sviluppo chiavi in mano.

- In questo modo l'approccio porta più problemi che soluzioni e di sicuro nessuno dei suoi vantaggi.

DevOps: Rilascio continuo

- Un ambiente DevOps supporta una forte cultura basata sullo sviluppo guidato dai test e dai processi di test automatizzati.

- Ciò include i test delle unità, i test funzionali e prestazionali e i test di convalida dell'ambiente.