E-BOOK

COME RENDERE AGILE UN'ARCHITETTURA MONOLITICA

Evoluzione intenzionale dei team, dei processi e delle applicazioni

Burr Sutter, *director of developer experience*
Deon Ballard, *content marketing*

facebook.com/redhatinc
@redhat
linkedin.com/company/red-hat

it.redhat.com

Le applicazioni non sono più di competenza esclusiva dei team IT. Ormai è assodato che oggi tutte le aziende offrono soluzioni software, e che la capacità di fornire con rapidità nuovi servizi e nuove funzionalità ai clienti è uno dei principali elementi di differenziazione dalla concorrenza che un'azienda può offrire. Un IT agile può consentire alle startup di ottenere un notevole vantaggio competitivo rispetto a grandi competitor.

Diverse generazioni fa (in anni tecnologici), i team IT all'interno di un'organizzazione erano team dedicati alla manutenzione dell'infrastruttura e dei servizi utilizzati dall'azienda. Alcune aziende offrivano servizi rivolti all'esterno, in particolare web service, ma si trattava in genere di una situazione rara e limitata a pochi servizi. Il team IT non era un reparto strategico o in grado di generare ricavi, ma piuttosto un ambiente di supporto considerato come un centro di costo.

Gli sviluppatori, a causa di un ambiente concentrato sull'infrastruttura, non avevano sempre ben chiara la funzione del codice. I cicli di rilascio erano lunghi, e le modifiche richiedevano molto tempo. Il codice creato dagli sviluppatori passava alla fase operativa o di verifica, per essere rilasciato dopo mesi. A causa del lungo lead time, gli ingegneri svolgevano le attività di sviluppo senza la soddisfazione di creare qualcosa e vederlo funzionare in una situazione reale.

La trasformazione digitale e i cambiamenti culturali e tecnologici che ne derivano, come il modello DevOps, hanno permesso un ritorno al piacere di creare codice. Oggi gli sviluppatori possono creare qualcosa e vederlo funzionare. È una svolta epocale che permette di ottenere risultati immediati. La possibilità di vedere la propria applicazione in funzione offre agli sviluppatori un feedback loop e consente loro di riprogettare e migliorare il codice, migliorando i progetti.

La difficoltà maggiore consiste nel trasformare le risorse a disposizione. Il passaggio da un ambiente tradizionale ad un ambiente moderno basato su microservizi e DevOps si compone di diverse fasi. Alcune organizzazioni hanno la fortuna di iniziare da zero, ma per la maggior parte delle aziende, la sfida sta principalmente nell'ottenere un ambiente agile.

COSA SIGNIFICA CAMBIAMENTO?

La trasformazione digitale è un cambiamento strategico per le aziende perché consente loro di adattare i servizi principali alle pressioni della concorrenza o all'emergere di nuove normative, e di eseguire aggiornamenti non appena viene individuata una vulnerabilità.

Tuttavia, non esiste una definizione univoca del concetto di trasformazione digitale. A volte, il termine viene usato per indicare nuove architetture come i microservizi, nuovi processi come DevOps, o nuove tecnologie come i container e le interfacce di programmazione di un'applicazione (API). Quando un termine può assumere più significati, in realtà non ha nessun significato in particolare. La trasformazione digitale non è qualcosa che si può acquistare, ma è un obiettivo che ogni organizzazione deve definire esclusivamente per sé stessa.

Non esistono modelli architetturali o piattaforme tecnologiche che funzionano perfettamente in qualsiasi ambiente. Le organizzazioni in grado di realizzare la trasformazione digitale sono quelle che hanno ben chiari i loro obiettivi e la loro cultura. Ogni azienda può trovarsi in una situazione diversa. Ad esempio:

- Walmart ha eseguito il deployment del suo codice durante il Black Friday, quando 200 milioni di persone erano online.[1]
- Amazon esegue aggiornamenti del codice ogni secondo (50 milioni di volte all'anno) su centinaia di applicazioni e milioni di istanze cloud.[2]
- Etsy esegue 60 deployment al giorno con un'applicazione monolitica.[3]
- Netflix esegue il deployment centinaia di volte al giorno su un'architettura distribuita complessa, in cui una singola modifica al codice passa dal check-in alla produzione in 16 minuti.[4]

Ognuna di queste aziende sta lavorando con team, tecnologie di base, code base e architetture totalmente diverse. Il punto è che per far funzionare tutto non si sono concentrate su un modello o una tecnologia specifici. Hanno tutte iniziato valutando i team, il debito tecnico esistente e le strategie aziendali, per poi portare le loro aziende nella direzione scelta in maniera intenzionale e coerente. Infine, sono riuscite a ottenere dei risultati.

Questo è il metodo giusto per ottenere la trasformazione digitale. A prescindere dal livello di digitalizzazione di un'azienda, è possibile completare il processo di trasformazione (correggendo debito tecnico, difetti di progettazione e così via) seguendo una strategia e passaggi definiti. Perché ciò avvenga, è necessario stabilire con chiarezza gli obiettivi e quanto occorrerà per raggiungerli.

ANALIZZA LO STATO DEL TUO STACK TECNOLOGICO

La valutazione dell'ambiente esistente e la ridefinizione della strategia aziendale non sono per nulla semplici. È un quadro difficile da ottenere. In una nota parabola, sei ciechi incontrano un elefante e ognuno di loro prova a capire di che animale si tratti toccandone una parte diversa del corpo. Uno di loro tocca una zanna e crede sia una lancia, un altro tocca il costato e crede sia un muro, un altro ancora tocca un orecchio e crede sia un ventaglio. È interessante notare che la parabola ha finali diversi a seconda della fonte. In alcune versioni, gli uomini non riescono ad accettare le dichiarazioni degli altri e finiscono per litigare. In altre, un uomo in grado di vedere descrive l'aspetto generale dell'animale, unificando le loro percezioni.

I vari finali sono probabilmente la parte più realistica della parabola. La morale della favola è che ognuno ha punti di vista diversi, informazioni limitate e supposizioni basate sul proprio punto di vista. L'insieme dei diversi punti di vista ci dice qualcosa sulla comunicazione e sui rapporti all'interno del team: alcuni riescono ad andare oltre il proprio punto di vista, altri no. Il nostro modo di vedere una situazione complessa (l'elefante) dipende da chi siamo, dove ci troviamo, cosa stiamo cercando, cosa sappiamo e cosa *non* sappiamo.

1 *O'Maidin, Cian. "Why Node.js Is Becoming The Go-To Technology In The Enterprise." NearForm, 10 mar. 2014, www.nearform.com/blog/node-js-becoming-go-technology-enterprise/. Accesso 1 set. 2017.*

2 *McKendrick, Joe. "How Amazon Handles a New Software Deployment Every Second." ZDNet, 24 mar. 2015, www.zdnet.com/article/how-amazon-handles-a-new-software-deployment-every-second/.*

3 *"The Great Microservices Vs Monolithic Apps Twitter Melee." High Scalability, 28 luglio 2014, http://highscalability.com/blog/2014/7/28/the-great-microservices-vs-monolithic-apps-twitter-melee.html.*

4 *Bukoski, Ed, et al. "How We Build Code at Netflix." The Netflix Tech Blog, 9 mar. 2016, https://medium.com/netflix-techblog/how-we-build-code-at-netflix-c5d9bd727f15.*

Tutto questo è ulteriormente complicato dal fatto che la maggior parte delle organizzazioni si trova a fronteggiare più situazioni complesse. Netflix è un'azienda unica, non solo per la sua architettura di microservizi avanzata e i suoi ambienti di verifica e distribuzione basati su container, ma per aver saputo creare i processi e lo stack tecnologico sulla base di una strategia aziendale definita, senza alcun debito tecnico. È nata come startup.

Qualsiasi organizzazione decida di preservare le applicazioni (e i team) in uso si troverà ad affrontare uno scenario complesso, che riguarda:

- strutture organizzative e modelli di comunicazione;
- processi di sviluppo, verifica, creazione e rilascio;
- debito tecnico e applicazioni di servizi esistenti;
- obiettivi o visioni strategiche divergenti, tra i vari reparti.

Si tratta di ostacoli che possono essere osservati da diversi punti di vista.

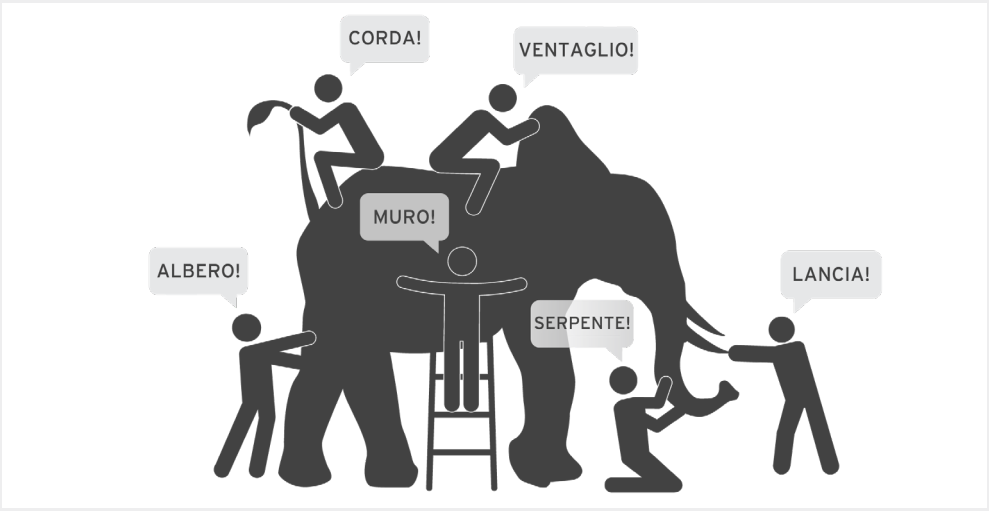

Figura 1. L'allegoria dei sei ciechi e dell'elefante

VISIONE DARWINIANA DELLA TRASFORMAZIONE DIGITALE

C'è una tendenza a vedere le iniziative IT o di trasformazione digitale come binarie: puoi scegliere un percorso oppure un altro completamente diverso. Questo approccio potrebbe non essere efficace per un paio di motivi. In primo luogo, perché a volte si può scegliere tra più alternative o optare per una soluzione ibrida. In secondo luogo, spesso non basta cambiare un solo fattore. Dal percorso di trasformazione, possono dipendere il tipo di modifiche strutturali richieste o cambiamenti culturali e di processo.

Niente succede in un vuoto spazio-temporale.

Sarebbe molto più costruttivo considerare la trasformazione digitale come un continuum caratterizzato da diverse fasi propedeutiche alla fase evolutiva successiva.

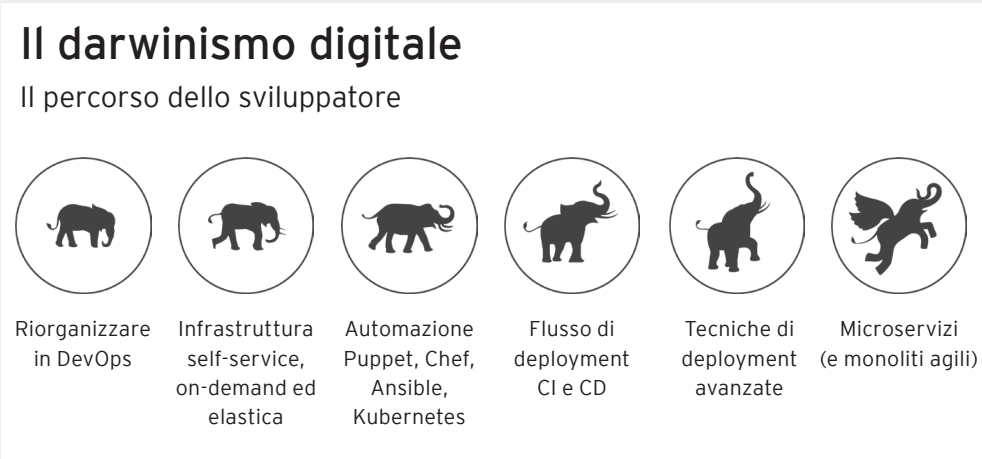

Figura 2. Le fasi del darwinismo digitale

Cambiamento di DevOps e processi

La base dell'evoluzione digitale è DevOps. Le applicazioni, come la strategia aziendale, sono il riflesso dei team e della comunicazione che ha portato alla loro creazione. La metodologia DevOps e approcci ai processi simili, coinvolgono le parti interessate nelle discussioni sullo sviluppo. Inoltre, offrono più informazioni sulla manutenzione del software e delle infrastruttura (e su come clienti e partner stanno effettivamente utilizzando le applicazioni). Creano, inoltre, un feedback loop più stretto tra i team, richiedendo linee di comunicazione aperte. La comunicazione aperta è fondamentale per qualsiasi altra fase evolutiva.

Infrastruttura self-service

Questa fase è un cambiamento incentrato sulla tecnologia, che permette di introdurre efficienze solitamente associate a piattaforme tecnologiche moderne. Container e cataloghi self-service permettono agli sviluppatori e ai gruppi di verifica e operazioni di avviare ambienti coerenti molto rapidamente, riducendo, in alcuni casi, il lead time per le nuove istanze da giorni a minuti. Perché un tecnico deve aspettare giorni per ottenere una risorsa?

I MICROSERVIZI SONO ASSOLUTAMENTE NECESSARI?

La fase finale dell'evoluzione digitale è rappresentata dai microservizi per la complessità della loro manutenzione. Ma devono necessariamente essere la fase finale dell'evoluzione della tua organizzazione?

No, non necessariamente.

In presenza di tutti i requisiti necessari a consentire l'innovazione, un'architettura monolitica può consentire rilasci settimanali e l'utilizzo di tecniche di deployment avanzate, di flussi CI/CD e di un'architettura scalabile e distribuita.

È consigliabile prendere in considerazione i microservizi solo in presenza di team di dimensioni tali da garantire rilasci più volte in una settimana o secondo tempistiche specifiche. Un team o un codice di piccole dimensioni non devono necessariamente essere inseriti un codebase di microservizi.

Automazione e orchestrazione dello sviluppo

L'automazione dello sviluppo riguarda due aspetti. Per il primo, rappresentato dalle tecnologie, è possibile utilizzare motori di deployment avanzati come Red Hat® Ansible o Puppet; il secondo aspetto riguarda la trasformazione dei processi. Molte organizzazioni hanno adottato processi rigidi relativamente alla gestione del cambiamento e dei rischi. Per sfruttare la nuova tecnologia è indispensabile trasformare quei processi in metodologie più agili.

Flussi di integrazione continua/distribuzione continua (CI/CD)

La distribuzione continua consente di effettuare modifiche alla distribuzione del software in modo rapido e iterativo. L'idea di un flusso prevede processi e tecnologie che riducono il rischio che un codice di scarsa qualità (o danneggiato) arrivi al deployment. Serve, inoltre, a mostrare la maturità dei passaggi precedenti: DevOps e comunicazione aperta tra i team, fasi di test e sviluppo, e processi di test e deployment automatizzati. Nel momento in cui si dispone di processi consolidati, è possibile eseguire il deployment del codice con maggiore rapidità.

Percorsi di deployment avanzati

Dopo aver predisposto i processi e l'infrastruttura che consentono deployment rapidi, è possibile iniziare a usare i sistemi di deployment per mitigare eventuali rischi dovuti ad aggiornamenti, valutare l'efficacia della funzionalità e fornire un campo di prova reale per le nuove idee. In presenza di ambienti separati è possibile effettuare il bilanciamento del carico tra di essi durante i deployment (deployment blue-green). Due ambienti diversi possono essere utilizzati al fine di testare l'interazione degli utenti (test A/B) o per l'esecuzione di aggiornamenti in sequenza su percentuali ridotte di utenti aumentando gradualmente il numero di utenti coinvolti (deployment di tipo canary).

Microservizi (o sistemi distribuiti)

Un microservizio è una piccola applicazione che esegue una sola funzione discreta. L'architettura applicativa complessiva può aver bisogno di eseguire decine o centinaia di funzioni diverse, e ognuna di queste funzioni viene definita ed orchestrata in un microservizio. Un'architettura di microservizi (o architettura distribuita) è complessa e semplice allo stesso tempo. La manutenzione, l'aggiunta e il ritiro di un solo servizio sono molto più semplici, ma l'architettura è più complessa. Se realizzata correttamente, una "progettazione basata su microservizi è la dimostrazione definitiva di tutto ciò che si è imparato sulla buona progettazione di applicazioni."[5] Tale architettura altamente distribuita offre percorsi di scalabilità più semplici, rende più facile l'introduzione di nuovi servizi o aggiornamenti e riduce il rischio di malfunzionamenti del sistema. L'elasticità che caratterizza questo tipo di architettura spiega perché i microservizi sono spesso associati ad aziende fortemente innovative come Netflix e Google.

"LA DISTRIBUZIONE CONTINUA È IRREALIZZABILE": LA LEGGE DI CONWAY, DEVOPS E CULTURA

Nel corso di una sessione generale di DevNation del 2016 Rachel Laycock ha dichiarato che "il processo di 'operazionalizzazione' del software è molto complesso".[6] Laycock ha condiviso un'esperienza di mancata riuscita della distribuzione continua in una grande azienda di servizi finanziari, che soffriva a causa dell'esecuzione di aggiornamenti (anche critici) lunghi intere settimane. L'azienda pensava che la soluzione fosse passare alla distribuzione continua. Dopo sei mesi passati a cercare di modificare i suoi processi l'azienda decise di tornare sui suoi passi, e Laycock comunicò ad uno dei vice presidenti che non era assolutamente possibile ottenere la distribuzione continua.

5 *Cotton, Ben. "From Monolith to Microservices." 3 gen. 2017, https://www.nextplatform.com/2017/01/03/from-monolith-to-microservices/.*

6 *Laycock, Rachel. "Continuous Delivery." Sessione pomeridiana. Red Hat Summit - DevNation 2016, 1 luglio 2016, San Francisco, California. https://www.youtube.com/watch?v=y87SUSOfgTY*

"Distribuire in modo rapido e frequente significa applicare un approccio CI/CD. Adottare i principi DevOps significa offrire supporto per le soluzioni distribuite."[9]

– BURR SUTTER

Il problema è in parte da attribuire alla tecnologia e all'architettura. L'azienda di servizi finanziari aveva un codebase con più di 70 milioni di linee di codice e un'architettura totalmente informe. Il software era il problema più evidente, e quindi il più facile da identificare. Tuttavia, il vero ostacolo era l'incapacità dell'organizzazione di modificare i comportamenti dei vari reparti. Era questo ad vanificare ogni tentativo di cambiamento.

La cultura prima di tutto

Un aspetto fondamentale da considerare durante il percorso dell'evoluzione darwiniana (del software) è che non si tratta solo e semplicemente di una trasformazione dal punto di vista tecnologico. In realtà deve esserci un'alternanza tra modifiche ai processi e ai team, e modifiche dell'infrastruttura, mentre la trasformazione della cultura organizzativa è decisamente più importante. Laycock ha concluso così il suo discorso:[7]

"Puoi creare la migliore architettura, ma coinvolgere persone e processi non basta. È necessario creare un ambiente [culturale] in grado di supportare la distribuzione continua e la disciplina dell'architettura, poiché le modifiche ai processi o alla struttura non sono durature. Le persone non seguono le regole. Il vero problema, quindi, è la Legge di Conway. Cosa significa? Mantenere le attuali strutture organizzative significa continuare a distruggere la propria architettura, per quanto sia ben progettata. *Su questo non c'è dubbio*".

Per ottenere un cambiamento evolutivo attraverso il software o la tecnologia, occorre pensare che l'ambiente circostante si evolve in modo naturale. In un'azienda, tale ambiente è rappresentato dalla cultura. I cambiamenti necessari per supportare l'evoluzione possono essere supportati dagli organi esecutivi, ma non possono essere imposti. Le persone devono aver voglia di cambiare. Deve avvenire per libera scelta, non per imposizione.

Gartner ci offre dei dati: "Nel 90% dei casi, se le organizzazioni cercano di usare un approccio DevOps, senza prima affrontare il problema della cultura organizzativa, il processo sarà destinato a fallire".[8]

Cambiare l'infrastruttura o l'architettura applicativa è facile, ma per cambiare in modo efficace ciò che si produce, bisogna prima cambiare la propria cultura.

Secondo la Legge di Conway "qualsiasi organizzazione che progetta un sistema finirà inevitabilmente per generare un progetto la cui struttura riflette il modello di comunicazione dell'organizzazione". A partire da questa affermazione è possibile ricavare due interpretazioni:

- Le modifiche all'architettura o all'infrastruttura non cambieranno alcunché, a meno che non si cambi anche la struttura alla base della comunicazione.
- Trasformare tale struttura consentirà di migliorare i processi e l'infrastruttura, in maniera pressoché indipendente dall'infrastruttura.

La Legge di Conway sostiene che "qualsiasi organizzazione che progetta un sistema finirà inevitabilmente per generare un progetto la cui struttura è la copia della struttura di comunicazione dell'organizzazione".[10]

DevOps: il primo passo

La metodologia agile è nata come approccio alla progettazione di software, al fine di riunire tutti i team coinvolti (QA, gestione del prodotto, sviluppo, documentazione) in un'entità più coesa. L'idea era quella di chiarire gli obiettivi mediante brevi iterazioni concentrate su attività specifiche espresse come obiettivi dell'utente (definite storie). Questo approccio ha portato all'abbandono del tradizionale metodo di sviluppo software a cascata, che prevedeva il passaggio di un progetto da un team all'altro.

7 *Laycock, Rachel. "Continuous Delivery." Sessione pomeridiana. Red Hat Summit - DevNation 2016, 1 luglio 2016, San Francisco, California. https://www.youtube.com/watch?v=y87SUSOfgTY.*

8 *"Gartner Highlights Five Key Steps to Delivering an Agile I&O Culture." 20 apr. 2015, www.gartner.com/newsroom/id/3032517.*

9 *DevNation Federal, 8 giugno 2017, Washington, DC, https://www.youtube.com/watch?v=tQOo2qaUc6w&t=1s*

10 *Conway, Melvin E. (aprile 1968), "How do Committees Invent?", Datamation*

"Le strutture dei team sono la prima bozza della tua architettura."

– MICHAEL NYGARD, RELEASE IT!

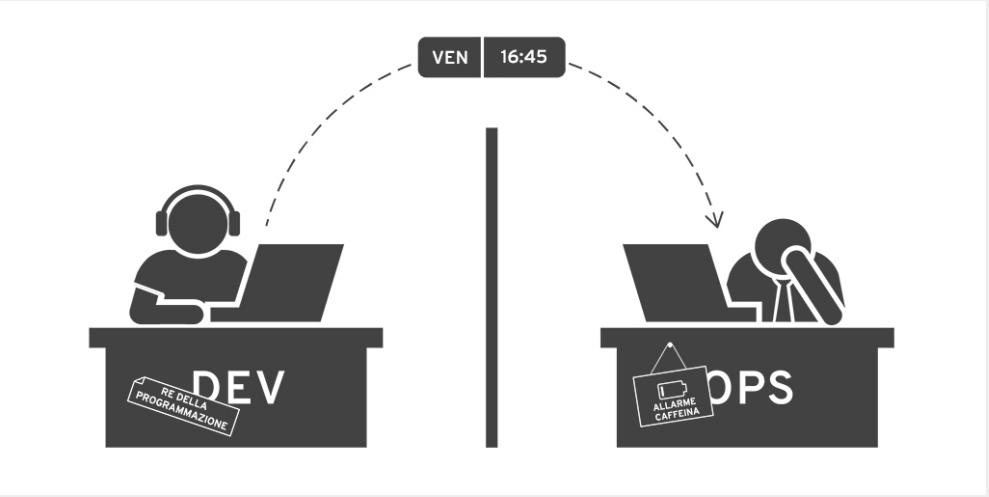

Figura 3. Il muro divisorio pre-DevOps

Tuttavia, riguardava solo metà del ciclo di vita effettivo di un'applicazione. Una volta completato il processo di sviluppo, avveniva il passaggio al team operativo che si occupava del deployment e della manutenzione (solitamente effettuati nel fine settimana).

Tuttavia, i team operativi non sempre conoscono approfonditamente l'applicazione, e corrono il rischio di eseguire un deployment inefficace. D'altro canto, gli sviluppatori possono non avere piena familiarità con l'ambiente operativo e, pertanto, creare un'applicazione non eseguibile nell'ambiente di produzione. Per risolvere questo problema e al fine di mitigare i rischi, molte organizzazioni adottano un costoso processo di gestione della trasformazione, nel tentativo di spiegare e giustificare i cambiamenti.

L'obiettivo dell'approccio DevOps è cambiare la cultura organizzativa, allineare processi di sviluppo e processi operativi, e ottenere la collaborazione dei team. La distanza tra diversi team è una realtà, ma è al tempo stesso artificiale. Un team può essere composto da più persone che condividono lo stesso incarico; in questo caso l'obiettivo delle metodologie DevOps è ridefinire il team in modo da includere tutti coloro che condividono il ciclo di vita di un'applicazione, e semplificare la comunicazione tra questi gruppi.

Il cambiamento culturale va persino oltre l'approccio DevOps, le metodologie agili o di altro tipo. Lo scopo principale è creare collaborazione tra i team. Solo cambiando i modelli di comunicazione, è possibile raggiungere risultati visibili.

Per ottenere grandi cambiamenti, a volte basta iniziare a piccoli passi. L'evoluzione della cultura organizzativa è alla base di tutti i cambiamenti tecnologici e di processo. Scopri come sviluppare una cultura DevOps attraverso i due approcci indicati di seguito.

- Consenti ai tuoi sviluppatori di osservare come lavorano i team operativi e il rollout in produzione, affinché possano acquisire un nuovo punto di vista.
- Stabilisci quanti passaggi o richieste di servizio occorrono ad uno sviluppatore per richiedere un nuovo sistema virtuale.

Vedere in prima persona come lavorano altri team può essere un modo efficace per comprendere come modificare i propri processi o per migliorare la comunicazione.

PUNTI SALIENTI DEL RAPPORTO "PUPPET'S STATE OF DEVOPS"

- Lead time 2.555 volte più veloci
- Aumento del numero di deployment pari a 200 volte
- Ripristino dei sistemi 24 volte più veloce.
- Tasso di fallimento dei cambiamenti 3 volte più basso
- Riduzione del 22% del tempo di rilavorazione

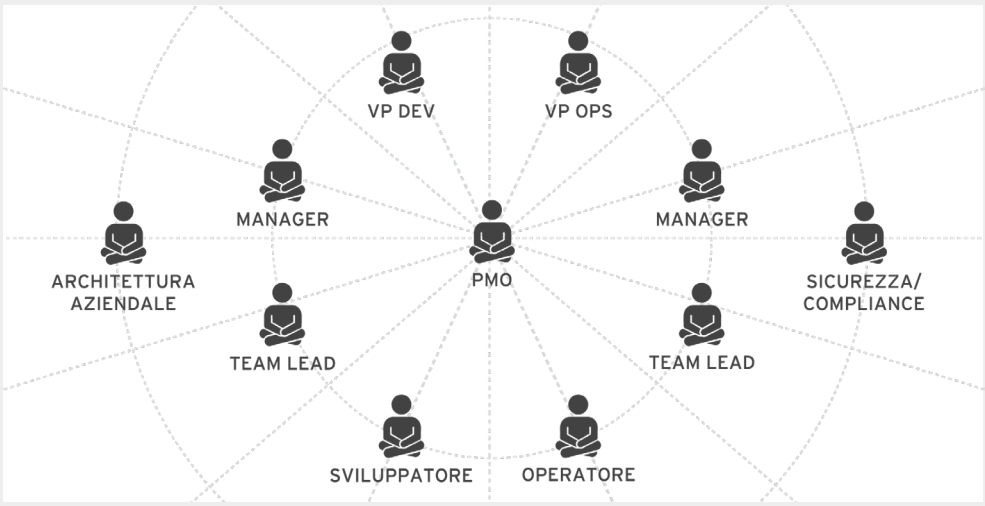

Figura 4. Panoramica sui percorsi di comunicazione tra team

Il rapporto "Puppet's State of DevOps" mostra quanto trasformare la struttura dei team e il modo di comunicare può essere efficace.[11] La ricerca rivela che i team DevOps riscontrano:

- Lead time 2.555 volte più veloci.
- Un aumento di 200 volte dei deployment.
- Un ripristino dei sistemi 24 volte più veloce.
- Un tasso di fallimento delle modifiche 3 volte più basso.
- Una riduzione del 22% dei tempi di rilavorazione.

La velocità è uno dei vantaggi principali offerti da un approccio DevOps. Considerando l'intero processo di rilascio, per poter distribuire un'applicazione è necessario coinvolgere tutti i team e, a seconda del livello di efficacia dei processi, dell'infrastruttura e del codebase, il processo di rilascio può richiedere un grande impegno.

11 Kim, Gene, et al. "State of DevOps Report." Puppet, 2016, https://puppet.com/resources/whitepaper/2016-state-of-devops-report.

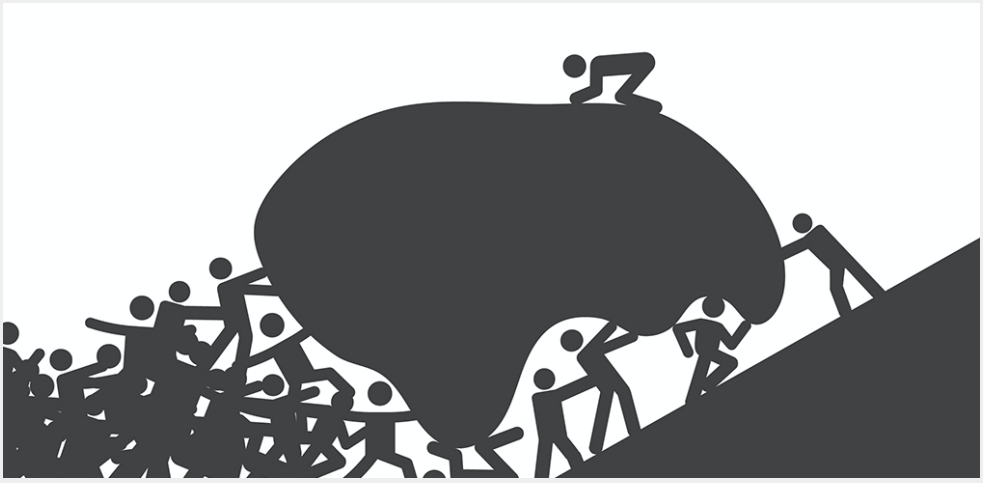

Figura 5. Sforzo richiesto al team per il rilascio di un'applicazione

Il carico di lavoro era gestibile quando i rilasci avvenivano con una frequenza relativamente bassa, Tuttavia, da quando il software è diventato un fattore chiave per le aziende, l'intero processo di rilascio deve verificarsi più volte in un anno. (In aziende fortemente innovative come Amazon, il processo di rilascio può verificarsi centinaia di volte al giorno.) Dunque, i team lavorano al passaggio in produzione delle applicazioni ripetutamente.

Figura 6. Sforzo per un solo rilascio, con molti altri rilasci all'orizzonte

L'approccio DevOps previene situazioni di emergenza in cui è richiesto l'intervento di tutti i team, offrendo iterazioni più uniformi e più sostenibili tra le fasi di pianificazione, sviluppo, verifica e deployment. L'intero ciclo di vita è ripetibile: ecco perché i team DevOps riscontrano un eccezionale aumento della produttività.

Modificare la struttura del tuo team e delle comunicazioni è indispensabile, indipendentemente dalla natura della tua architettura applicativa (monolitica o composta da microservizi distribuiti). È necessario garantire una comunicazione continua, dalle fasi precedenti alla pianificazione di un'applicazione, fino alle quelle successive al deployment. Un team dedicato ai servizi può facilmente diventare un silo di servizi, a meno che non vi sia un'apertura alla comunicazione e al feedback.[12]

PROGETTAZIONE DI UN'ARCHITETTURA APPLICATIVA: I MICROSERVIZI

La trasformazione digitale si ottiene nel momento in cui si dispone di una nuova architettura applicativa. È proprio questo il passaggio più evidente e l'obiettivo più facile da identificare. Vale la pena dare un'occhiata all'architettura, anche se la trasformazione delle piattaforme e dei processi deve avere la precedenza.

L'importanza della progettazione, debito tecnico e strategia

Un enorme ostacolo al cambiamento è rappresentato dall'incapacità di immaginare un'alternativa alle applicazioni esistenti. C'è un motivo per cui molte iniziative di trasformazione digitale prendono in considerazione l'adozione di un approccio di tipo "rip and replace": a volte sembra più facile ricominciare da zero.

Il problema, però, è che il debito tecnico è riconducibile alla progettazione. Rimuovere un'applicazione senza aver stabilito come sostituirla significa che, in futuro, ci si troverà di nuovo alle prese con un'architettura impenetrabile.

"Sperare che tutto vada per il meglio," ha spiegato Rachel Laycock, "non è un buon metodo di progettazione. Le intenzioni devono essere chiare."[13]

Qualsiasi cosa si decida di sviluppare può diventare un debito tecnico. Spesso, i team iniziano a sviluppare nuove applicazioni senza un progetto definito nel dettaglio, e questo non è necessariamente un errore. Nel descrivere il debito tecnico e l'importanza di una buona progettazione, Martin Fowler sottolinea che iniziare senza un progetto ben definito spesso consente di accelerare il time-to-market.[14] Tuttavia, al contrario di un progetto non definito, un progetto ben definito è in grado di offrire benefici a lungo termine.

12 Cotton, Ben. "From Monolith to Microservices." 3 gen. 2017, https://www.nextplatform.com/2017/01/03/from-monolith-to-microservices/.

13 Laycock, Rachel. "Continuous Delivery." Sessione pomeridiana. Red Hat Summit - DevNation 2016, 1 luglio 2016, San Francisco, California. https://www.youtube.com/watch?v=y87SUSOfgTY

14 Fowler, Martin. "Design Stamina Hypothesis." 20 luglio 2007, https://martinfowler.com/bliki/DesignStaminaHypothesis.html.

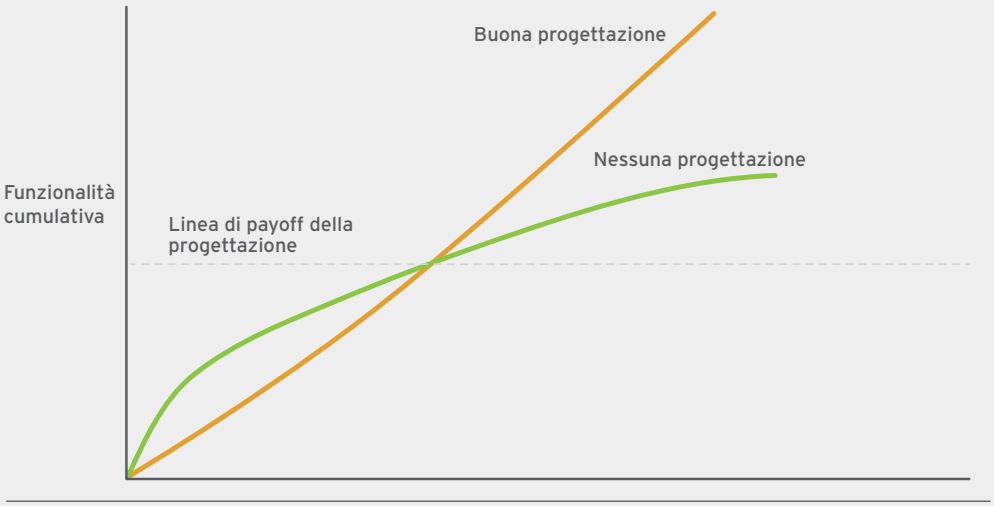

Figura 7. Grafico concettuale della teoria della progettazione efficiente

Prima di iniziare a pianificare un'architettura, occorre avere ben chiare le priorità e gli obiettivi strategici. In Information Week, Rob Zuber ha scritto:[15]

"Se non hai le idee chiare sui tuoi obiettivi di business, sul tuo prodotto o sul tuo ruolo, la suddivisione prematura dello stack in una serie di servizi senza una funzione precisa avrà un esito negativo... Questo è il momento giusto per pensare all'architettura, ma è importante farlo gradualmente, in modo da testare le idee e convalidarle piuttosto che cercare di cambiare rapidamente il modello di sviluppo. Un approccio di questo tipo risulterà nel fallimento del progetto a cui lavori da mesi, e ti troveresti a dover comunicare che, pur avendo imparato molto dall'esperienza, il team ha deciso di non portarlo a termine. Non è questo il tuo obiettivo. Il tuo obiettivo è offrire valore e avere un impatto sui tuoi clienti e sulla tua attività, grazie a processi pianificati e ben ponderati".

In un recente studio di IDC, Stephen Elliot ha affermato che, prima di definire un'architettura, è necessario identificare clienti ed utenti finali, ciò di cui necessitano, gli obiettivi di business desiderati, e come verrà misurato il successo.[16]

In altre parole, non devi iniziare pensando a come ottenere qualcosa o a quale architettura vuoi usare, ma devi piuttosto concentrarti sul tuo obiettivo strategico e, quindi, progettare l'architettura che lo supporterà. In questo modo viene data priorità alle applicazioni.

Definizione di microservizi e monoliti

Oggi esistono tre tipologie di architettura applicativa basate sui rapporti tra i servizi: i monoliti (ad alto grado di accoppiamento), i microservizi (disaccoppiati) e le sempre meno diffuse Service-Oriented Architecture (a basso grado di accoppiamento).

15 Zuber, Rob. "Transitioning to Microservices: The Story of Two Monoliths." InformationWeek, 25 maggio 2017, https://www.informationweek.com/devops/transitioning-to-microservices-the-story-of-two-monoliths-/a/d-id/1328972.

16 Elliot, Stephen. "Enabling DevOps with Modern Application Architecture." IDC Perspective, dic. 2016.

STRATEGIE INTENZIONALI

- Chi sono i tuoi clienti o i tuoi utenti?
- Quali sono i loro obiettivi?
- Di quale infrastruttura disponi?
- Qual è il ciclo di vita di quella infrastruttura?
- Quali servizi o funzionalità sono richieste per un singolo flusso di lavoro?
- Qual è il ciclo di vita di quel flusso di lavoro?
- Qual è il tuo percorso di deployment, e con che frequenza deve essere distribuito?
- Su quali capacità aziendali influisce?

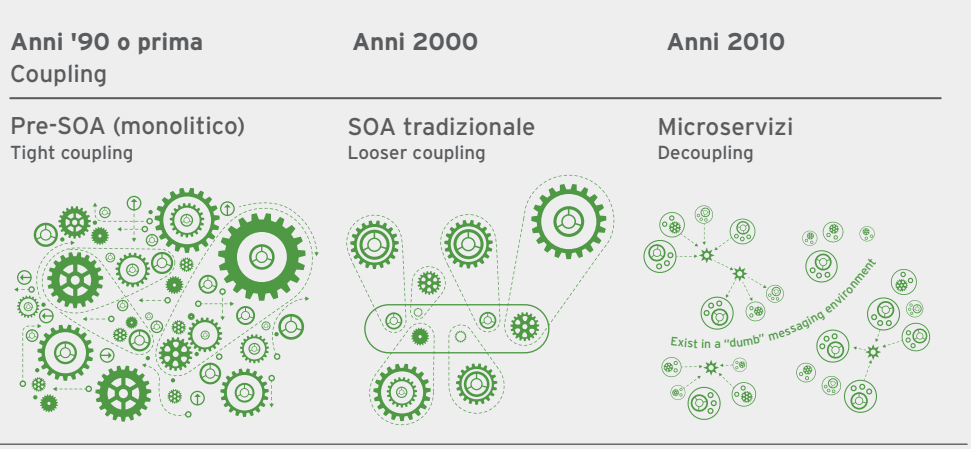

Figura 8. Coupling in architetture monolitiche, service-oriented e di microservizi

In parole povere, un monolite è uno stack applicativo singolo che contiene tutte le funzionalità presenti all'interno di quell'applicazione. Presenta un elevato grado di accoppiamento sia nell'interazione tra i servizi sia nel modo in cui questi ultimi sono sviluppati e distribuiti. Aggiornare o ottenere la scalabilità di un singolo aspetto di un'applicazione monolitica ha implicazioni sull'intera applicazione e sulla sua infrastruttura di base.

La scalabilità dinamica e il failover sono i potenziali problemi di un monolite, generalmente risolvibili con progettazioni caratterizzate da una scalabilità semplice, come la scalabilità orizzontale (che duplica una funzione in un cluster) o la scalabilità verticale (che replica le istanze ed espande l'hardware). Un problema sottovalutato della scalabilità riguarda i team operativi e di sviluppo. Se per rilasciare un'applicazione monolitica occorrono da 6 a 9 mesi e un team di 50 persone, un approccio scalabile potrebbe consentire il rilascio di cinque applicazioni più piccole, sfruttando cinque team e aggiornamenti singoli a distanza di poche settimane uno dall'altro.

L'architettura monolitica è con ogni probabilità l'architettura applicativa più datata, per la sua semplicità iniziale e per i rapporti più definiti tra servizi e interdipendenze. Inoltre, questa architettura rispecchia maggiormente un'infrastruttura IT commodity-based limitata con processi di sviluppo e rilascio più rigidi.

Poiché i monoliti rappresentano un tipo di architettura più datato, sono frequentemente associati alle applicazioni esistenti. Al contrario, per offrire una maggiore flessibilità, le architetture più moderne cercano di suddividere i servizi per funzionalità o capacità aziendale. Questo è comune soprattutto con interfacce customer-facing, come API, app mobili o app per il web, che sono solitamente più leggere e richiedono aggiornamenti più frequenti per rispondere alle aspettative degli utenti.

Le architetture distribuite moderne vengono definite microservizi. Pur avendo aspetti in comune con altre architetture modulari, come le Service-Oriented Architecture (SOA), i microservizi passano dal basso accoppiamento tra i servizi alla totale indipendenza dei servizi. La definizione di un singolo servizio è generalmente chiara. I servizi possono essere aggiunti, aggiornati o rimossi da un'architettura più ampia con facilità. Ciò rappresenta un vantaggio sia per la scalabilità dinamica sia per la tolleranza degli errori: infatti, la scalabilità consente di adattare ogni servizio in base alle esigenze, senza che sia necessaria un'infrastruttura pesante o senza che possa provocare un failover e compromettere altri servizi. Come ha scritto Ben Cotton, "la progettazione basata su microservizi è la dimostrazione finale di tutto ciò che si è appreso sulla buona progettazione di applicazioni."[17]

17 Cotton, Ben. "From Monolith to Microservices." 3 gen. 2017, https://www.nextplatform.com/2017/01/03/from-monolith-to-microservices/.

Grazie alla sua fluidità, l'architettura di microservizi è spesso associata a tecnologie dinamiche come container e cloud, che permettono di avviare ed eliminare con facilità istanze singole, anche in maniera programmatica.

L'immediatezza che caratterizza gli ambienti di elaborazione distribuiti offre vantaggi diretti, sia all'organizzazione sia ai team che possono riscontrarne l'impatto sul loro lavoro, anche negli aspetti indicati di seguito.

- Maggiore tolleranza degli errori e riduzione al minimo delle interruzioni di servizio.
- Protocolli semplici come JSON/REST e HTTP/OAuth per un'integrazione più facile.
- Servizi poliglotti, che offrono maggiore flessibilità agli sviluppatori.
- Time-to-market più rapido per applicazioni e funzionalità.
- Recupero dei dati semplificato e condivisione tra servizi, senza la necessità di usare bus di messaggi o conversioni più pesanti.[18]

In Coding the Architecture, Cotton ha affermato che un'architettura monolitica si differenzia da un'architettura di microservizi quando il tempo di esecuzione dell'applicazione stessa è monolitico e statico.[19] Molte applicazioni sono organizzate in numerosi pacchetti o moduli, che possono essere basati su processi di creazione e distribuzione indipendenti uno dall'altro, ma è l'applicazione stessa che interagisce come un'unica entità.

La domanda sorge spontanea. Qual è il tipo di architettura più appropriato? D'istinto siamo portati a pensare che ciò che è nuovo è sempre migliore, ma è importante fare un passo indietro e valutare bene cosa realmente si adatta maggiormente ai propri obiettivi aziendali. Tra gli ingegneri di Etsy e di Netflix si è scatenata una interessante discussione su Twitter in merito alla necessità dei microservizi per il deployment continuo e per l'autonomia degli sviluppatori. Gli ingegneri di Etsy hanno evidenziato il fatto di avere piccoli team di sviluppo agile basati su funzionalità e deployment estremamente rapidi (circa 60 al giorno) pur continuando ad eseguire un'applicazione monolitica. Avevano semplicemente trovato un sistema che funzionava per loro e per la loro cultura.

L'architettura e le tecnologie mutano e si evolvono. "Le best practice del passato, non saranno adatte al futuro", ha sottolineato Laycock durante il suo intervento in DevNation. "I microservizi sono una scelta, una risposta. Non sono la soluzione, ma una delle soluzioni possibili."[20]

Le organizzazioni, quindi, non dovrebbero chiedersi se *sostituire un'applicazione monolitica con i microservizi*, ma piuttosto *qual è l'obiettivo strategico e cosa fare per raggiungerlo.* Comprendere la struttura delle comunicazioni, della cultura, delle capacità aziendali e dei flussi di lavoro necessari influenza le modalità di accoppiamento dei servizi, i loro cicli di vita e, in ultima analisi, l'architettura applicativa.

---

18 ***Lambert, Natalie.*** *"Micro Services: Breaking down Software Monoliths." NetworkWorld, 22 nov. 2017, https://www.networkworld.com/article/3143971/application-development/micro-services-breaking-down-software-monoliths.html.*

19 *Annett, Robert. "What Is a Monolith?" Coding the Architecture, 19 nov. 2014, http://www.codingthearchitecture.com/2014/11/19/what_is_a_monolith.html.*

20 *Laycock, Rachel. "Continuous Delivery." Sessione pomeridiana. Red Hat Summit - DevNation 2016, 1 luglio 2016, San Francisco, California. https://www.youtube.com/watch?v=y87SUSOfgTY*

TABELLA 1. CONFRONTO TRA ARCHITETTURE MONOLITICHE E DI MICROSERVIZI

	MONOLITI	MICROSERVIZI
Sviluppo	• Sviluppo iniziale più rapido • Difficoltà a modificare o aggiungere funzionalità	• Progettazione iniziale complessa • Aggiunta o modifica di servizi più semplice
Flussi di lavoro delle applicazioni	• Inserimento facile dell'applicazione nei flussi di lavoro • Implementazione della funzionalità (ad esempio autenticazione o monitoraggio) in un'unica posizione	• Assegnazione più complessa dei servizi nei flussi di lavoro • Possibile nebulosità tra interdipendenze o requisiti tra servizi
Formazione e manutenzione	• Architettura semplice • Requisiti di sviluppo rigidi per ambiente e linguaggio	• Architettura flessibile con maggiore complessità di progettazione • Servizi poliglotti con API standardizzate o messaggistica di connessione
Scalabilità	• Scalabilità complessa dipendente dall'infrastruttura hardware • Scalabilità dell'intera applicazione per picchi di singoli servizi	• Servizi singoli facilmente scalabili senza influire sull'architettura complessiva • Utilizzo di infrastrutture software-defined (container, cloud) per una reattività dinamica
Aggiornamenti, failover, downtime	• Tutti i servizi sono caratterizzati da un basso accoppiamento • I servizi possono essere aggiornati insieme; il controllo versione è accoppiato • Rischio di errore del sistema in caso di errore di un singolo servizio	• I servizi sono disaccoppiati • I servizi possono essere aggiunti o aggiornati in maniera indipendente • Il rischio di errore è limitato a un numero ridotto di servizi
Automazione	• L'automazione non è richiesta	• Automazione e orchestrazione sono indispensabili

Gli svantaggi degli ambienti di elaborazione distribuiti

Un ingegnere scherzò sul fatto che i microservizi trasformano ogni interruzione in un mistero insoluto, [21] offrendo una interessante sintesi del problema più spinoso degli ambienti di elaborazione distribuiti: la diffusione della complessità.

21 @HonestStatusPage. Twitter, 7 ott. 2015, https://twitter.com/honest_update/status/651897353889259520?lang=en.

Aumento dei costi e costi di transazione

Le modifiche apportate all'infrastruttura, necessarie per la creazione di ambienti di elaborazione realmente distribuiti, possono comportare elevati costi di capitale. Non bisogna poi dimenticare i costi indiretti sostenuti per la formazione o l'acquisizione di nuove competenze, il cambiamento delle strutture dei team e la migrazione dei sistemi. È vero che questi costi possono dare luogo a risparmi futuri in termini di time-to-market e riduzione dei tempi di inattività (se la nuova architettura viene effettivamente implementata), ma questi benefici non sono immediati.

Aumento della complessità

Anziché un solo punto di errore (catastrofico), come nel caso di un'architettua monolitica, l'architettura di microservizi ha centinaia di potenziali punti di errore. Così come per un monolite, l'individuazione di un punto di errore può rivelarsi un compito arduo poiché la causa scatenante potrebbe non essere così ovvia; tuttavia, i microservizi aumentano la complessità proprio perché le interdipendenze tra i servizi sono ancora meno visibili.

Anche i cicli di rilascio più rapidi e la struttura dei team più agile contribuiscono all'aumento della complessità. Una comunicazione efficace è, pertanto, fondamentale con i microservizi. La complessità intrinseca di ogni architettura di software deve essere bilanciata. Quella complessità può essere nascosta nell'applicazione stessa (nel caso delle applicazioni monolitiche) o trasmessa alle strutture di comunicazione del team (nel caso dei microservizi).

Visione e progettazione dei sistemi

Per progettare un'architettura di microservizi efficace è necessaria una notevole visione sistemica. Le interazioni tra servizi, le capacità aziendali e le esperienze degli utenti devono essere chiare. La visione sistemica, inoltre, deve includere anche le strutture di comunicazione e i processi presenti all'interno della cultura organizzativa. Gartner sottolinea che senza la comprensione del sistema complessivo, le architetture di microservizi funzioneranno in modo molto simile al monolite stereotipato: "La mancata adozione di un approccio globale che prenda in considerazione l'architettura software, ma anche l'infrastruttura e i processi di sviluppo, non forniranno risultati ottimali e non permetteranno di superare molti degli svantaggi di un sistema software monolitico".[22]

Limiti delle risorse

Nel caso delle architetture monolitiche le risorse sono ovviamente limitate, per via della scarsa scalabilità. O il sistema ha una capacità hardware sufficiente a gestire i picchi di carico sul servizio più ampio, con un conseguente sottoutilizzo della capacità, o rischia di non avere una capacità sufficiente a gestire i picchi di utilizzo.

Con i microservizi, l'architettura è flessibile, e dal momento che ogni servizio è molto piccolo, è facile scalare nuovi servizi su risorse molto leggere o temporanee, come i container o le istanze cloud.

Poiché i requisiti delle singole risorse per un determinato servizio sono (relativamente) ridotti, è possibile ignorare o ridurre al minimo le richieste delle singole risorse dell'architettura complessiva. Questo discorso si basa su alcune supposizioni:

- La rete è sempre affidabile.
- La latenza è pari a zero.
- La larghezza di banda è infinita.
- La rete è sicura.

22 Knoernschild, Kirk. "Refactor Monolithic Software to Maximize Architectural and Delivery Agility." Gartner Key insights, 18 maggio 2017

- La topologia dell'infrastruttura resta immutata.
- C'è un solo amministratore.
- Il costo del trasporto è pari a zero.
- La rete è omogenea.

Seppure influisca solo su un esempio di utilizzo di nicchia per gli ambienti di elaborazione, esiste un altro limite delle risorse. Le applicazioni caratterizzate da prestazioni estremamente elevate possono avere requisiti eccessivamente esigenti per un'architettura di microservizi, come, ad esempio, la creazione di modelli climatici o la mappatura del DNA.

È importante conoscere gli svantaggi dei microservizi solo per ricordarsi che nessun sistema è perfetto. L'intento non deve essere quello di trovare una tecnologia (o un'architettura) perfetta che risolva tutti i problemi, ma, piuttosto, quello di concentrarsi sulla cultura e sulla comunicazione e ridefinire i processi per creare un'organizzazione matura ed efficace. A quel punto, l'organizzazione avrà la capacità di progettare un'architettura funzionale in grado di soddisfare gli obiettivi specifici.

Realizzare un'applicazione essenziale

Come è giusto che sia, visto che si tratta di considerazioni fondamentali, la gran parte delle discussioni sulla trasformazione digitale verte sull'infrastruttura e sulla cultura. Tuttavia, è necessario ricordare che l'infrastruttura e la cultura sono i mezzi per raggiungere un fine: la creazione di un'applicazione utile per gli utenti ed essenziale per l'organizzazione.

Le applicazioni essenziali possiedono determinate caratteristiche, ovvero:

- Reattività agli utenti
- Attinenza alla funzione o allo scopo aziendale principale
- Adattamento o reattività ai cambiamenti dinamici all'interno dell'ambiente
- Connessione tra gli ambienti
- Leggerezza e flessibilità per consentire di aggiungere o mantenere funzionalità

Quando un'applicazione ha queste caratteristiche, può dirsi essenziale.

Queste caratteristiche possono essere presenti sia in architetture monolitiche sia in architetture di microservizi. L'architettura è una scelta di progettazione. Cambiare prospettiva sui vari elementi è fondamentale per creare un'applicazione moderna e flessibile, anche nel caso di un'architettura monolitica. Etsy è in grado di gestire la distribuzione continua, dozzine di aggiornamenti al giorno, carichi di utenti elevati e applicazioni mobili, grazie alla sua applicazione monolitica, poiché è stata realizzata appositamente.

Questa predisposizione deve essere riflessa in tutte le architetture, monolitiche o di servizi che siano, in numerose aree chiave, tra cui:

- Integrazione.
- Gestione e coerenza dei dati.
- Messaggistica e comunicazioni di servizio.
- Modelli di processi o flussi di lavoro coerenti.

Semplificando molto, gli ambienti monolitici tradizionali avevano la tendenza ad affrontare queste problematiche o come problemi che dovevano essere risolti, o come un aspetto poco chiaro dell'applicazione. L'integrazione, ad esempio, veniva talvolta considerata come un'alternativa alla fusione di applicazioni o sorgenti di dati diverse; era una specie di collante che teneva insieme i vari componenti dell'ambiente. Anche aspetti come i flussi di processo erano visti come un modo per accrescere la produttività rispetto all'intervento manuale e per automatizzare i flussi di lavoro, ma spesso non si trattava di una precisa decisione progettuale.

Con un'applicazione moderna, i vari aspetti dell'architettura, come l'integrazione o i flussi di processo, non sono secondari; sono di primaria importanza per il funzionamento dell'applicazione. Questo aspetto è ben visibile nelle architetture di microservizi (pur valendo anche per le architetture monolitiche).

Integrazione e messaggistica

Con i microservizi la difficoltà sta nel capire come accoppiare questi servizi in modo da conservarne l'autonomia e, al contempo, consentire una comunicazione libera. Si tratta, quindi, di un problema di integrazione e di messaggistica. L'integrazione definisce la modalità in cui questi servizi separati comunicano l'uno con l'altro. Questa è una scelta progettuale fondamentale. Dalla prospettiva dell'infrastruttura, i microservizi sono, talvolta, suddivisi in "container + API", riflettendo la necessità di accoppiare questi servizi con i container, che rappresentano i servizi, e le API, che rappresentano il tessuto connettivo. (Un approccio simile sarebbe "container + messaggistica", ovvero qualsiasi tecnologia che offre un modo per trasmettere dati tra i servizi.)

Nelle Service-Oriented Architecture questo aspetto è ancora diverso poiché messaggistica e integrazione non sono centralizzate in bus e i dati non sono convertiti tra i servizi.

Flussi di processo

Un'applicazione moderna è reattiva verso tutti gli utenti. Questo è particolarmente chiaro nelle applicazioni mobili, che sono consumer-focused, semplici e immediate. Quando un cliente avvia una transazione, come verificare un saldo bancario o cercare un biglietto aereo, è fondamentale lanciare immediatamente un flusso di lavoro. Il flusso di lavoro deve adattarsi alla decisione successiva dell'utente, pur restando conforme ai protocolli dell'organizzazione. Inizialmente, la gestione dei processi di business (BPM) tradizionale era usata per automatizzare le attività e accrescere l'efficienza, ma nel contesto di un'architettura applicativa moderna, diventa un elemento fondamentale per distribuire la funzionalità e contribuire all'esperienza dell'utente.

Dati

Ad un certo punto, lo stato di tutti i dati viene verificato. I dati devono essere archiviati e accessibili dai servizi all'interno dell'architettura (a prescindere dal fatto che sia monolitica o distribuita), pertanto è fondamentale che sia chiaro il modo in cui i dati si muovono tra i servizi e quali tipi di dati vengono generati.[23] Lo sviluppatore Christian Posta ha scritto che l'aspetto più difficile dei microservizi è proprio la gestione dei dati, ossia cercare di definire i confini naturali e i limiti transazionali tra i servizi.[24] La difficoltà è quella, quindi, di trovare un equilibrio tra coerenza, accessibilità e autonomia dei dati. La definizione dei domini naturali permette di formare i modelli di dati e le strutture di storage.

---

23 Brown, Kyle. "Refactoring to Microservices, Part 2: What to Consider When Moving Your Data." IBM developerWorks, 4 maggio 2016, https://www.ibm.com/developerworks/cloud/library/cl-refactor-microservices-bluemix-trs-2/index.html.

24 Posta, Christian. "The Hardest Part About Microservices: Your Data." 14 July 2016, http://blog.christianposta.com/microservices/the-hardest-part-about-microservices-data/.

"Chiunque provi a copiare Netflix, riesce a copiare solo quello che vede. Copia i risultati, non il processo."[27]

– ADRIAN COCKCROFT, EX CHIEF CLOUD ARCHITECT DI NETFLIX

DISTRIBUZIONE RAPIDA (MA RESPONSABILE)

Uno degli obiettivi principali della trasformazione digitale è ottenere un rilascio più rapido delle applicazioni. Ciò nonostante, la velocità è solo una somma dei miglioramenti nell'efficienza. Al fini della trasformazione la velocità è essenziale, poiché consente un'innovazione rapida, l'introduzione di nuove funzionalità e la capacità di testare nuove idee.

Nel 2013, durante un intervento presso la Computer Science and Engineering Technology Open House dell'Università del Minnesota, Ron Kohavi, illustre ingegnere e general manager del team di sperimentazione di Microsoft per l'intelligenza artificiale, ha sottolineato che meno di un terzo delle idee riesce a migliorare i parametri che era nata per migliorare.[25]

Per valutare la validità di un'idea è necessario testare in maniera approfondita ed efficace non solo la qualità del codice, ma anche quella dell'esperienza e delle preferenze dell'utente. Per stabilire se un'idea è buona, quindi, è necessario sperimentarla. Come sostiene Kohavi "i dati trionfano sull'intuizione".

Questo è l'obiettivo delle tecniche di distribuzione continua e di deployment avanzato. CI/CD è la piattaforma per il deployment rapido, mentre le tecniche di deployment sono gli strumenti per la sperimentazione ed il perfezionamento.

Dietro a queste due fasi c'è il cambiamento culturale, che incoraggia l'innovazione e previene errori e rischi. L'innovazione non è una destinazione o una meta, ma un processo alimentato dalla sperimentazione. Per accettare il rischio di un fallimento durante il percorso verso l'innovazione serve una cultura dell'umiltà.

Self-service, automazione e CI/CD

La trasformazione digitale richiede innanzitutto il passaggio a una cultura DevOps caratterizzata da piccoli team dinamici e intercomunicazione. La fase successiva riguarda la tecnologia, al fine di fornire un'infrastruttura in grado di supportare cicli di sviluppo rapidi.

Questa fase è composta da due aspetti strettamente correlati tra loro:

- Le infrastrutture self-service elastiche, ovvero la capacità di richiedere e ricevere un'istanza sulla base di specifiche ben definite pressoché immediatamente.
- L'automazione o l'orchestrazione, ovvero la capacità di creare e gestire in maniera automatica più istanze all'interno di un ambiente.

Sono tecnologie complementari: è impossibile automatizzare senza un ambiente elastico o gestire potenzialmente centinaia di istanze senza gli strumenti che le rendono coerenti e ripetibili.

In questa fase della trasformazione, il miglioramento della tecnologia può corrispondere a un aumento tangibile della produttività. Un cliente Red Hat che ha introdotto un catalogo self-service per permettere agli sviluppatori di richiedere con rapidità sistemi virtuali, è riuscito a ridurre il tempo necessario per ricevere un sistema richiesto da cinque giorni a circa 15 minuti, e a trasformare il processo da manuale ad automatico.[26] Questo cambiamento ha ridotto gli interventi manuali da parte dei team operativi, ed ha aumentato la produttività (e la motivazione) degli sviluppatori.

---

25 *"Online Controlled Experiments: Introduction, Insights, Scaling, and Humbling Statistics." SOBACO University of Minnesota, 18 ott. 2013, https://sobaco.umn.edu/content/online-controlled-experiments-introduction-insights-scaling-and-humbling-statistics.*

26 *"Red Hat Virtualization Increases Efficiency and Cost-Effectiveness." Studio sull'impatto economico totale di Forrester, 26 gen 2017, www.redhat.com/en/resources/virtualization-tei-forrester-analyst-paper.*

27 *https://twitter.com/kelseyhightower/status/641886057391345664*

"Docker dispone di un set di strumenti che offrono container dinamici. Si basa sui principi di un'infrastruttura immutabile in cui la configurazione delle applicazioni è fissata dallo sviluppatore e trasmessa alla produzione con modifiche minime alla configurazione."[28]

–IDC

Le tecnologie sono complementari, ma non prescrittive. Le infrastrutture elastiche possono essere cloud (pubblici o privati), macchine virtuali o container. L'automazione può essere un componente all'interno del sistema infrastrutturale (come il progetto Heat per l'orchestrazione con OpenStack®) o uno strumento esterno, come Red Hat CloudForms o Kubernetes con i container basati su Docker. A seconda delle competenze e dell'infrastruttura esistente della tua organizzazione puoi scegliere vari percorsi tecnologici.

Uno dei motivi per cui i container (come Docker o Red Hat OpenShift) sono stati spesso associati a CI/CD è rappresentato dagli ambienti di sistema rigidi e ripetibili offerti, i quali rendono il passaggio tra ambienti organizzativi completamente diversi molto meno problematico. (Quando una modifica al codice passa dallo sviluppo alla produzione, capita di sentir dire: "sul mio computer funzionava".) Nel caso delle macchine virtuali o delle istanze cloud, possono esserci differenze tra il sistema operativo e le versioni del pacchetto; i container, invece, devono avere impostazioni del sistema operativo identiche, e l'immagine del container selezionata deve essere la stessa tra i vari deployment.

La coerenza è una buona base per la prima parte dell'approccio CI/CD: l'integrazione continua. Con l'integrazione continua, i cambiamenti dello sviluppo vengono compilati costantemente ed integrati ad ogni check-in, consentendo di identificare i problemi più rapidamente. Per verificare la stabilità o la funzionalità, l'integrazione continua viene solitamente supportata da una serie di test automatizzati. Il processo continuo di check-in, creazione e verifica permette di garantire la qualità del codice.

Una volta adottata l'integrazione continua, la tua organizzazione potrà eseguire deployment continui, applicando le modifiche in produzione con maggiore rapidità. Questa velocità è vantaggiosa sia per gli sviluppatori sia per i team operativi. Gli sviluppatori e i leader aziendali saranno soddisfatti di vedere i nuovi prodotti arrivare sul mercato più velocemente. I team operativi, d'altro canto, hanno la possibilità di implementare correzioni ed intervenire su vulnerabilità ed esposizioni comuni (CVE) in un periodo di tempo molto più breve, offrendo quindi un sistema più sicuro ed efficiente.

La parola continuo può avere significati leggermente diversi a seconda del ritmo di sviluppo e delle esigenze aziendali specifiche. In presenza di grandi architetture monolitiche (processi e tecnologie robusti con un'architettura applicativa più tradizionale) si hanno aggiornamenti monolitici che avvengono con un unico rilascio a settimana. In questo caso, i requisiti dei processi agili costituirebbero un ostacolo.[29] Con i microservizi, invece, tutti i servizi possono essere aggiornati tramite cicli di perfezionamento sovrapposti, permettendo quindi di aggiornare l'intera architettura quotidianamente.

Deployment avanzati e innovazione

Le fasi del percorso verso un approccio CI/CD consistono nel garantire rapidità e qualità nella distribuzione delle applicazioni. Tuttavia, come riportato da Kohavi, molte delle idee non consentono di raggiungere i risultati attesi. Kohavi ha illustrato questo punto nel suo discorso di apertura mostrando i diversi design del motore di ricerca Bing, e chiedendo alla platea di indovinare quale versione era stata preferita dagli utenti. Kohavi ha effettuato un test di quattro domande ad eliminazione diretta, e uno solo tra le centinaia di partecipanti è rimasto in piedi.[30]

---

28 "The Emergence of Microservices as a New Architectural Approach to Building New Software Systems" nella serie INDUSTRY DEVELOPMENTS AND MODELS di IDC. Author, Al Hilwa, giugno 2015.

29 Spazzoli, Raffaele. "The Fast-Moving Monolith: How We Sped-up Delivery from Every Three Months, to Every Week." Red Hat Developer's, 27 ott. 2016, https://developers.redhat.com/blog/2016/10/27/the-fast-moving-monolith-how-we-sped-up-delivery-from-every-three-months-to-every-week/.

30 "Online Controlled Experiments: Introduction, Insights, Scaling, and Humbling Statistics." SOBACO University of Minnesota, 18 ott. 2013, https://sobaco.umn.edu/content/online-controlled-experiments-introduction-insights-scaling-and-humbling-statistics.

Con ciò ha voluto dimostrare che, basandosi sui dati anziché sull'intuito, è possibile dare risposte più esatte. La progettazione di Bing fu finalizzata mediante test approfonditi sugli utenti, che permisero di scoprire che circa un terzo delle idee era effettivamente riuscito a migliorare l'esperienza dell'utente (altrettante la avevano, invece, peggiorata).

L'infrastruttura applicativa per la sperimentazione

Nel 2006, quasi un decennio prima che il termine microservizi facesse la sua comparsa, lo sviluppatore Neal Ford definì la cosiddetta programmazione poliglotta nel suo blog Meme Agora.[31] Ford aveva identificato una svolta nell'ambiente di programmazione.

Inizialmente le applicazioni aziendali erano caratterizzate da un solo linguaggio. Quando le applicazioni hanno iniziato a muoversi verso architetture server-client o basate sul web, per scrivere determinate azioni venivano usati alcuni linguaggi specifici (come JavaScript per l'interfaccia utente web o SQL per i database), ma nel complesso l'applicazione continuava a essere scritta in un solo linguaggio.

L'idea alla base della programmazione poliglotta di Ford è che non sarebbe più esistito un linguaggio di programmazione centrale, ma che all'interno dell'architettura applicativa complessiva, i singoli servizi o le singole funzioni potevano essere scritte in linguaggi completamente diversi più adatti a rispondere alle esigenze di un determinato servizio.

La programmazione poliglotta riflette perfettamente una delle esigenze principali di un ambiente di sviluppo agile: gli sviluppatori devono avere la possibilità di provare qualcosa di nuovo al di fuori della progettazione centrale di un'applicazione. Questo vale sia per le applicazioni monolitiche efficaci sia per le architetture di microservizi.

Al momento di creare una piattaforma per la sperimentazione è fondamentale prendere in considerazione la flessibilità e le opzioni a disposizione all'interno dell'ambiente di sviluppo. Un ambiente di sperimentazione deve, quindi, supportare:

- Più linguaggi.
- Più runtime.
- Ambienti di deployment flessibili (cloud, fisici o ibridi).
- Architetture applicative flessibili o modificabili; in questo modo un'applicazione può adattarsi ai cambiamenti dell'ambiente durante l'intero ciclo di vita.
- Standard open o capacità di standardizzare in maniera iterativa.

I container supportano la standardizzazione in modo efficace, anche se è possibile raggiungere gli stessi risultati anche con altri tipi di piattaforme. I container hanno requisiti specifici per quanto riguarda librerie, linguaggi e versioni. Un catalogo di container, tuttavia, può avere centinaia di migliaia di immagini diverse e supportare, quindi, linguaggi e runtime diversi. Oltre a permettere agli sviluppatori di individuare il runtime e il linguaggio giusti per l'esecuzione desiderata di un determinato servizio, questo tipo di piattaforma consente anche ai team delle operazioni di eseguire con efficacia il deployment di tali servizi.

Modelli di deployment innovativi

Le tecniche di deployment avanzate conferiscono struttura e trasparenza all'innovazione. Le metodologie di deployment mature generano un ambiente che consente sperimentazione, feedback e analisi reali. Una sperimentazione migliore porta a un'innovazione migliore.

Di seguito sono descritti modelli di deployment comuni che, a seconda della natura dell'applicazione e dell'ambiente dell'utente possono essere più o meno appropriati.

---

31 Ford, Neal. "Polyglot Programming." Meme Agora, 5 dic. 2006, http://memeagora.blogspot.com/2006/12/polyglot-programming.html.

"I dati trionfano sull'intuizione."

RON KOHAVI, GENERAL MANAGER OF ARTIFICIAL INTELLIGENCE EXPERIMENTATION, MICROSOFT UNIVERSITÀ DEL MINNESOTA, DIPART. DI COMPUTER SCIENCE AND ENGINEERING TECHNOLOGY OPEN HOUSE 2013 [23]

Ambienti blue-green

Un ambiente blue-green serve a mitigare i rischi legati alle modifiche. Una nuova build passa attraverso tutti gli ambienti nel flusso CI/CD. Per la produzione sono presenti due ambienti identici (blue e green), ma solo uno è attivo. Il cambiamento viene avviato nell'ambiente inattivo, in produzione; dopo la verifica di tale ambiente, il router sposterà il traffico verso l'ambiente aggiornato.

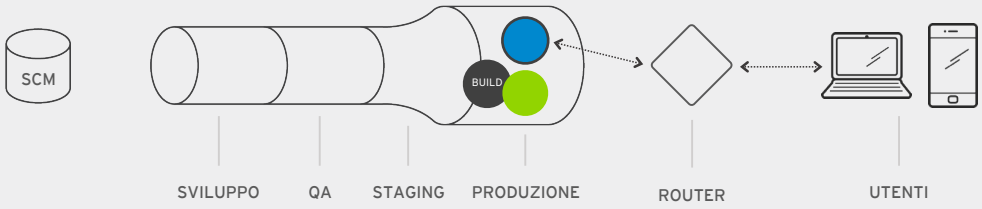

Figura 9. Flusso di deployment blue-green

Rilasci di tipo canary

Un rilascio di tipo canary è simile a un deployment blue-green, con l'unica differenza che il rilascio iniziale avviene solo per un sottogruppo di utenti all'interno dell'ambiente (il termine "canary" fa riferimento ai canarini un tempo utilizzati nelle miniere di carbone). Dopo aver ricevuto un feedback da un gruppo di utenti, è possibile estendere gradualmente il rilascio a tutti gli utenti.

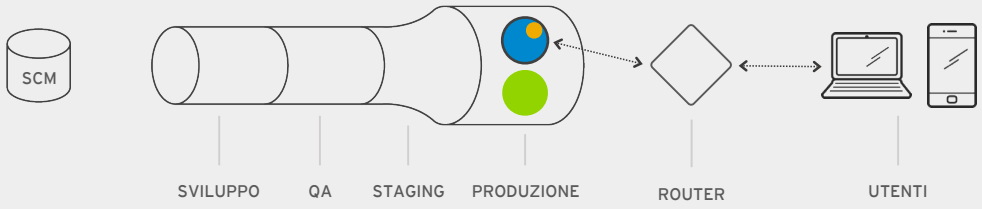

Figura 10. Flusso di deployment di tipo canary

Questo modello può essere incluso nelle tecniche di verifica per valutare le varie funzionalità o i progetti di applicazioni destinate a piccoli gruppi all'interno di un ambiente di produzione attivo e con traffico e modelli di utilizzo reali.

Test A/B

Il test A/B fornisce agli utenti due design diversi, al fine di valutare quale si avvicina maggiormente ai parametri prestazionali desiderati. La valutazione può essere richiesta in modo esplicito, ad esempio chiedendo agli utenti di valutare la propria esperienza o di fornire un feedback, o senza che l'utente ne sia consapevole. In questo caso, è possibile unire il test A/B con i rilasci di tipo canary, per fare un confronto tra i due design proposti oppure nascondere una funzionalità e valutare il modo in cui gli utenti interagiscono con i diversi design, usando l'ambiente attuale come riferimento.

Per esempio, nella Figura 11, l'applicazione mobile sembra identica agli utenti, ma gli ambienti A/B stanno usando algoritmi diversi per testare il livello di soddisfazione del prodotto.

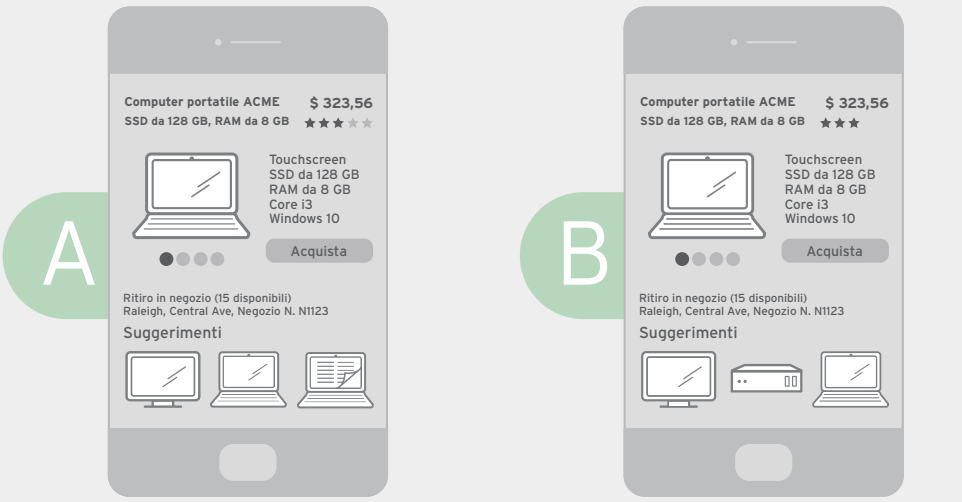

Figura 11. Test interfaccia A/B

Dopo aver selezionato una determinata progettazione è possibile rilasciarla in una porzione più ampia dell'ambiente, come avviene per il rilascio di tipo canary.

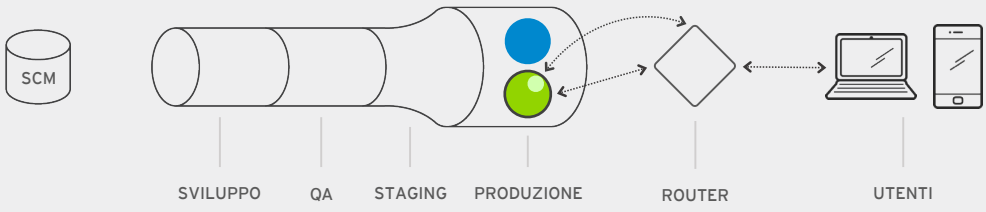

Figura 12. Flusso di rilascio test A/B

Se condotta in maniera efficace, questo tipo di verifica può trasformare un ambiente di produzione in un ambiente di sperimentazione, e permettere ai team di realizzare progetti più innovativi e di maggior valore.

Questo test è alla base del concetto secondo cui i dati consentono di prendere decisioni più accurate.

COME RENDERE AGILE UN'ARCHITETTURA MONOLITICA

Individua il tuo livello

Tra un ambiente a cascata più tradizionale e un ambiente di microservizi completamente distribuiti esistono tantissimi livelli intermedi. Laycock ha affermato che tutte le architetture software sono il risultato de "l'attrito tra coesione e accoppiamento".[32] Quando si inizia a pianificare una strategia di trasformazione digitale, l'attrito si manifesta nell'infrastruttura e nella cultura dell'azienda.

Innanzitutto, è necessario stabilire obiettivi realisticamente raggiungibili. Realisticamente non significa "facilmente", poiché il fine ultimo della trasformazione digitale è cambiare in maniera essenziale la cultura, i processi, l'architettura e la tecnologia. Significa capire cosa si sta cercando di ottenere con il cambiamento e definire con chiarezza cosa serve per raggiungere quell'obiettivo. Ecco alcune domande da porsi:

- Come sono strutturati, attualmente, reparti e team?
- Quali sono i modelli di comunicazione tra quei gruppi?
- Chi è attualmente coinvolto nei cicli di pianificazione?
- Guardando alla funzionalità, quanto si avvicina l'architettura applicativa corrente all'architettura applicativa desiderata?
- Qual è il livello di tolleranza dei rischi o degli errori dell'organizzazione?
- Qual è il grado di comprensione dei flussi di materiali e informazioni? (Ciò consente di tracciare una "mappa" della tua organizzazione.)
- Con che frequenza è necessario rilasciare un aggiornamento per soddisfare le esigenze operative e dei clienti?
- Quale nuova funzionalità è richiesta dagli obiettivi aziendali o dalle esigenze di sviluppo?

Definisci i tuoi principi operativi

I cambiamenti culturali sono alla base di tutti i cambiamenti di processo, tecnologia o architettura che la tua organizzazione dovrà mettere in atto per poter realizzare la trasformazione digitale.

La creazione di un insieme di principi fondamentali, anche se basilari, supportati dalla dirigenza e dai team, può contribuire a rafforzare le iniziative di trasformazione digitale e a unificare i team.

32 Laycock, Rachel. "Continuous Delivery." Sessione pomeridiana. Red Hat Summit - DevNation 2016, 1 luglio 2016, San Francisco, California. https://www.youtube.com/watch?v=y87SUSOfgTY

"Un'architettura applicativa robusta, processi agili e DevOps maturi, e un team di gestione dei dati dedicato danno vita a un ambiente di microservizi efficace. Questo ambiente può ridurre il lead time dello sviluppo del 75%."

– GARTNER[33]

Anche se questi principi possono essere semplici, può rivelarsi utile far chiarezza sulle attitudini ed i comportamenti considerati più importanti per cambiare la cultura aziendale, soprattutto se alcuni di questi (come l'apertura alla sperimentazione e al rischio) sono in antitesi alla cultura attuale. Ecco alcuni principi:

- Il fallimento è normale.
- Sperimentare è una buona abitudine.
- I team devono avere la priorità.
- Cercare di migliorare continuamente.
- Non smettere mai di formarsi.
- Essere sempre affidabili.
- Essere trasparenti.

Figura 13. Sforzo ridotto per un rilascio ben orchestrato

33 Innovation Insights for Microservices, Anne Thomas and Aashish Gupta, 27 gennaio 2017

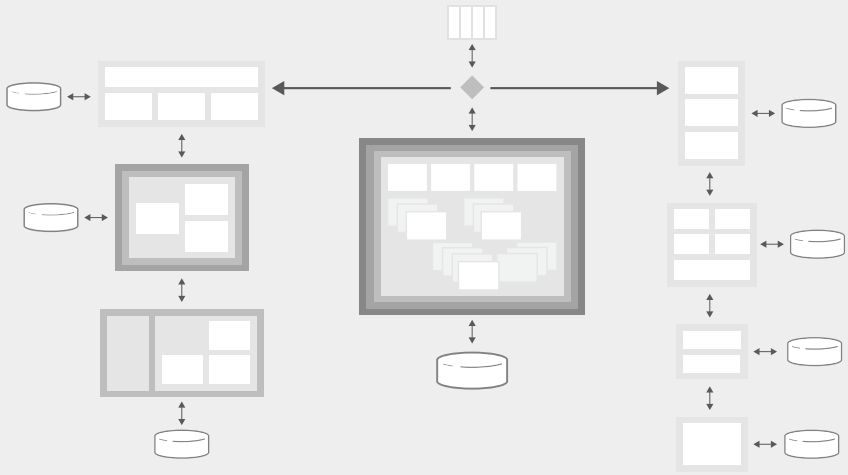

Figura 14. Suddividere i singoli servizi per trasformare l'applicazione monolitica

Sostituisci la tua applicazione monolitica

La trasformazione digitale inciderà sull'applicazione e sull'architettura esistenti. Se hai un'applicazione monolitica, hai a disposizione due modi per iniziare ad affrontare il debito tecnico:

- Suddividere i servizi esistenti nel momento in cui richiedono aggiornamenti o sostituzioni (metodo detto "strangling").
- Offrire nuove funzionalità creando servizi separati e indipendenti (metodo detto "starving").

Non è necessario un passaggio totale ai microservizi. Key Bank, cliente di Red Hat, aveva la necessità di velocizzare i tempi di rilascio, passando da una frequenza trimestrale ad una frequenza settimanale. Ciò è stato possibile pur mantenendo un'applicazione monolitica.[34] In presenza di un'applicazione monolitica, la logica applicativa principale può essere preservata, ma possono coesistere domini logici che consentono la separazione dei servizi, in particolare per le interfacce utente. Ad esempio, la creazione di un livello separato per i servizi, come un sistema API, un front end mobile ed altre interfacce utente permette ai servizi rivolti ai clienti o all'esterno di trarre vantaggio da iterazioni più rapide o cicli di vita più separati rispetto all'applicazione principale, offrendo, in ultima analisi, una maggiore innovazione con un minor rischio per l'azienda.

Gartner consiglia di adottare questo approccio incrementale per le architetture distribuite poiché è al tempo stesso "iterativo e focalizzato sulle aree importanti".[35]

A prescindere da quale sia la fase finale dell'evoluzione digitale per la tua organizzazione, il tuo approccio deve coprire tre aree basilari:

- Agilità nella progettazione dell'architettura.
- Sperimentazione.
- Automazione.

34 Spazzoli, Raffaele. "The Fast-Moving Monolith: How We Sped-up Delivery from Every Three Months, to Every Week." Red Hat Developers, 27 ott 2016, developers.redhat.com/blog/2016/10/27/the-fast-moving-monolith-how-we-sped-up-delivery-from-every-three-months-to-every-week/.

35 Olliffe, Gary. "How to Design Microservices for Agile Architecture." Gartner Key Insights, 30 gen. 2017.

Architetture progettate per ottenere agilità

La base della tua architettura deve essere agile indipendentemente dal fatto che il tuo obiettivo sia semplificare i processi per migliorare la tua applicazione monolitica o creare un'architettura di microservizi. Spesso, si ottiene agilità da una soluzione ibrida. Gartner raccomanda di avviare nuovi progetti monolitici e passare ai microservizi una volta raggiunta la maturità del progetto.[36] Gartner afferma, "Si potrebbe pensare che sia richiesto un impegno eccessivo in termini di sviluppo, ma la nostra ricerca suggerisce il contrario: un approccio monolitico iniziale permetterà di ridurre il rischio, migliorare la produttività iniziale e garantire il decoupling e la scomposizione dell'applicazione nel giusto insieme di microservizi".[37]

Una buona progettazione consente di ottenere un processo di sviluppo caratterizzato da trasparenza e semplicità. Il codice deve essere comprensibile. La funzionalità e lo scopo devono essere chiari. Una volta che l'applicazione è matura, può evolversi in architetture più distribuite. La presenza di buoni processi di sviluppo e deployment garantisce un modello agile sostenibile.

Investi nella sperimentazione

Una parte del budget deve essere riservata alla sperimentazione di nuove tecnologie e funzionalità applicative. IDC, ad esempio, consiglia di destinare il 2% del budget IT alla sperimentazione delle tecnologie container.[38]

Gartner afferma che la tecnologia non è un obiettivo di per sé e che, pertanto, eseguire il deployment nel cloud o creare microservizi non consentirà di ottenere un cambiamento. Perché la trasformazione sia possibile, è necessario avere ben chiaro il fine ultimo.[39]

Ciononostante, gli obiettivi strategici digitali sono molto spesso supportati da cambiamenti tecnologici. La riduzione del time-to-market, ad esempio, può richiedere il passaggio ai container, mentre le applicazioni JavaTM EE possono aver bisogno di piattaforme containerizzate com Red Hat JBoss® Enterprise Application Platform (EAP) per poter essere eseguite.

Una riserva di risorse può aiutare sviluppatori e team operativi a identificare le tecnologie utili, e a sviluppare le proprie competenze in modo da supportare qualsiasi tipo di infrastruttura verrà effettivamente implementato.

Automazione completa

L'automazione offre alcuni vantaggi evidenti: una maggiore efficienza grazie alla rimozione dei passaggi manuali, e una coerenza e una riproducibilità intrinseche. L'automazione di ogni passaggio dello sviluppo (dall'inizio) e del deployment (alla maturità dei processi) offrirà ai tuoi team un feedback loop sui cambiamenti in corrispondenza di ogni fase, e nel passaggio dallo sviluppo alle operazioni, fino ai clienti. Questo approccio consente di migliorare la qualità del codice.

Innanzitutto, fissa dei valori di riferimento su cui sviluppare una strategia di automazione che includa:

- Definizione dei parametri appropriati.
- Visualizzazione o creazione di diagrammi relativi agli attuali flussi di lavoro.
- Identificazione dei team coinvolti nelle diverse fasi.

---

36 Ibid.

37 Ibid.

38 Elliot, Stephanie, et al. "IDC TechBrief: Containers." IDC. gen. 2017.

39 Knoernschild, Kirk. "Refactor Monolithic Software to Maximize Architectural and Delivery Agility." Gartner Key insights, 18 maggio 2017.

CONCLUSIONE

Con il passare del tempo, le applicazioni aziendali hanno la tendenza a trasformarsi in monoliti: opachi, difficili da aggiornare e restii all'integrazione di nuove funzionalità. Sono proprio queste applicazioni a definire le operazioni di business che generano entrate. Si tratta di un problema ingombrante.

Si può, però, risolvere il problema rendendo l'ambiente più agile e adattabile, purché sia ben chiaro l'obiettivo che si intende raggiungere. La trasformazione digitale è, quindi, un processo evolutivo. Non esiste un unico risultato ideale; ogni percorso evolutivo, infatti, riflette lo scopo e la cultura dell'organizzazione stessa.

Analizza ogni fase dell'evoluzione digitale: DevOps, ambienti self-service o elastici, automazione, flussi CI/CD, deployment avanzati e microservizi. Modella la tua strategia di trasformazione digitale sulla base del livello evolutivo che più si avvicina alle esigenze del tuo business.

Concentrati sulla trasformazione della tua cultura organizzativa e allinea i cambiamenti tecnologici all'evoluzione dei processi corrispondenti affinché la tecnologia sia totalmente supportata dai team.

Mentre i tuoi processi maturano, inizia a valutare l'applicazione e l'architettura della tua azienda. Se necessario, isola o sviluppa servizi indipendenti, e dai vita a un'architettura agile in grado di adattarsi a priorità aziendali che cambiano ed emergono.

Infine, promuovi l'innovazione. Questo implica una certa tolleranza al rischio e all'errore (entro i limiti dei tuoi obiettivi aziendali e delle esigenze dei tuoi clienti) e l'abitudine a mettere da parte risorse in termini di tempo, denaro e infrastruttura. La sperimentazione è alla base dell'innovazione, e aumenta le probabilità di successo della trasformazione digitale. Inoltre, riporta un po' di quell'entusiasmo iniziale che ha spinto molti sviluppatori ed esperti delle operazioni a entrare nel mondo della tecnologia: la capacità di creare e di vedere quella creazione prendere forma.

INFORMAZIONI SU RED HAT

Red Hat è il leader mondiale nella fornitura di soluzioni software open source e si avvale di un approccio community-based per offrire tecnologie cloud, Linux, middleware, storage e di virtualizzazione caratterizzate da affidabilità e prestazioni elevate. L'azienda offre inoltre servizi di supporto, formazione e consulenza per i quali ha ottenuto diversi riconoscimenti. Principale punto di riferimento in una rete globale di aziende, partner e community open source, Red Hat consente di creare tecnologie specifiche e innovative che garantiscono libero accesso alle risorse per la crescita e preparano i clienti al futuro dell'IT.

EUROPA, MEDIO ORIENTE E AFRICA (EMEA)
00800 7334 2835
it.redhat.com
europe@redhat.com

TURCHIA
00800-448820640

ISRAELE
1-809 449548

EAU
8000-4449549

facebook.com/redhatinc
@redhat
linkedin.com/company/red-hat

it.redhat.com
#8980_0917