Comprensione e implementazione dei microfrontend su AWS

AWS Linee guida prescrittive

AWS Linee guida prescrittive: Comprensione e implementazione dei microfrontend su AWS

Table of Contents

Introduzione 1
Panoramica 1
Concetti fondamentali 6
Progettazione basata sul dominio 6
Sistemi distribuiti 8
Cloud computing 8
Architetture alternative 10
Monoliti 10
Applicazioni di livello N 10
Micro-servizi 11
Scelta dell'approccio adatto alle proprie esigenze 11
Decisioni di architettura 13
Confini del micro-frontend 13
Come suddividere un'applicazione monolitica in microfrontend 14
Approcci di composizione con micro-frontend 16
Composizione lato client 17
Composizione laterale 19
Composizione lato server 19
Routing e comunicazione 21
Routing 21
Comunicazione tra micro-frontend 21
Gestisci le dipendenze dei microfrontend 22
Non condividete nulla, ove possibile 22
Quando condividi codice 23
Stato condiviso 23
Framework e strumenti 25
Considerazioni generali sul quadro 25
Integrazione API – BFF 27
Styling e CSS 29
Sistemi di progettazione – Un approccio basato sulla condivisione 29
CSS completamente incapsulato – Un approccio «share nothing» 31
Shared Global CSS – Un approccio basato sulla condivisione totale 31
Organizzazione 33
Sviluppo agile 33

Composizione e dimensioni del team ... 34
DevOps cultura ... 35
Orchestrazione dello sviluppo di microfrontend tra più team ... 36
Implementazione ... 37
Governance ... 39
Contratti API ... 39
Interattività incrociata ... 40
Equilibra autonomia e allineamento ... 41
Creazione di micro-frontend ... 41
nd-to-end Test elettronici per microfrontend ... 41
Rilascio di micro-frontend ... 42
Registrazione di log e monitoraggio ... 42
Avviso ... 42
Bandiere di funzionalità ... 44
Individuazione dei servizi ... 45
Suddivisione dei pacchetti ... 46
Versioni di Canary ... 46
Team della piattaforma ... 48
Passaggi successivi ... 49
Risorse ... 53
Collaboratori ... 54
Cronologia dei documenti ... 55
Glossario ... 56
... 56
A ... 57
B ... 60
C ... 62
D ... 65
E ... 69
F ... 71
G ... 72
H ... 73
I ... 74
L ... 77
M ... 78
O ... 82

P 85
Q 87
R 88
S 91
T 94
U 96
V 96
W 97
Z 98
..... xcix

Comprensione e implementazione di micro-frontend su AWS

Amazon Web Services (collaboratori)

Luglio 2024 (cronologia dei documenti)

Poiché le organizzazioni puntano all'agilità e alla scalabilità, l'architettura monolitica convenzionale spesso diventa un collo di bottiglia, ostacolando lo sviluppo e l'implementazione rapidi. I micro-frontend mitigano questo problema suddividendo interfacce utente complesse in componenti più piccoli e indipendenti che possono essere sviluppati, testati e implementati in modo autonomo. Questo approccio migliora l'efficienza dei team di sviluppo e facilita la collaborazione tra backend e frontend, favorendo l'allineamento dei sistemi distribuiti. end-to-end

Questa guida prescrittiva è personalizzata per aiutare i leader IT, i proprietari di prodotti e gli architetti di diversi domini professionali a comprendere l'architettura micro-frontend e creare applicazioni micro-frontend su Amazon Web Services ().AWS

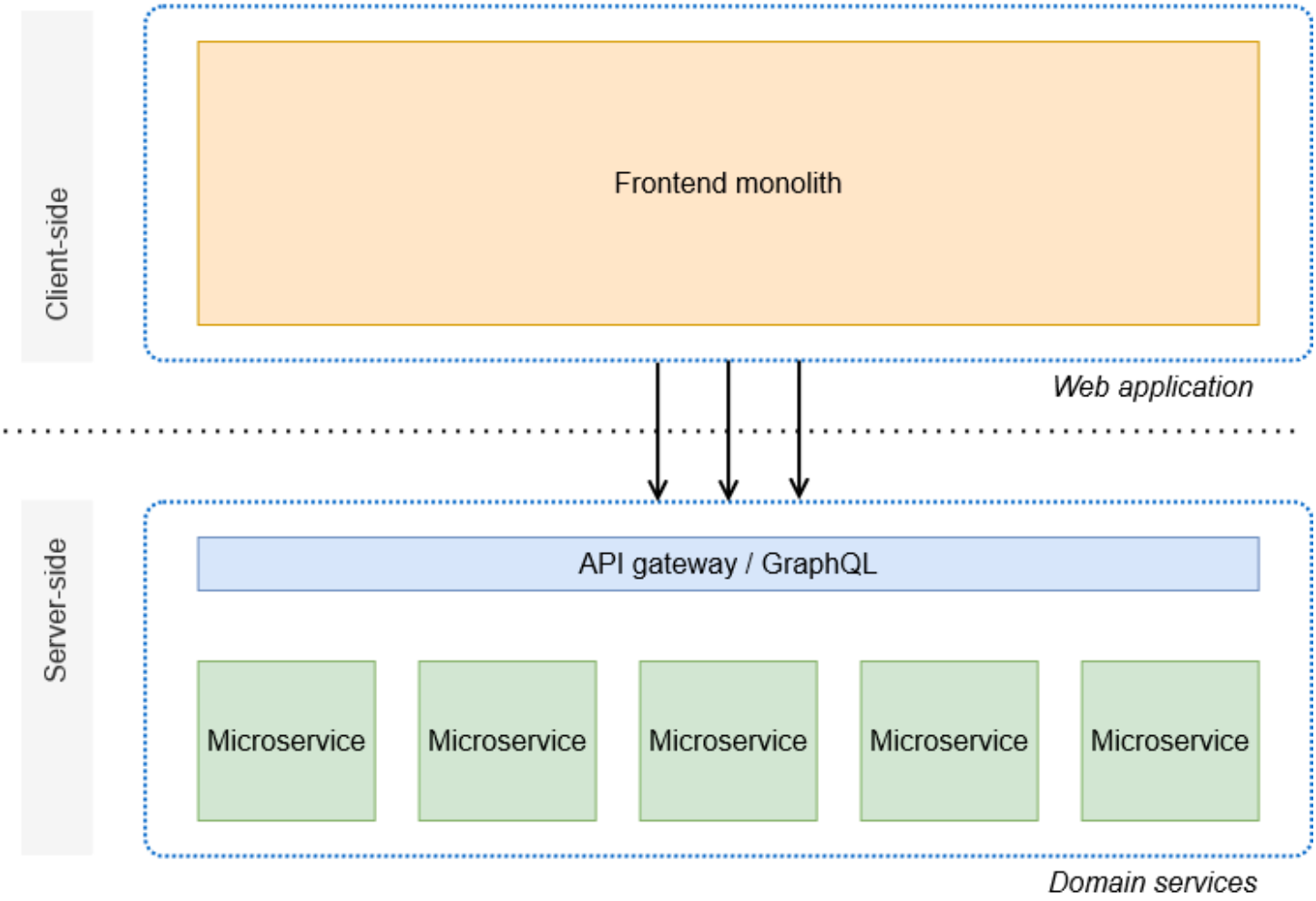
Panoramica

I microfrontend sono un'architettura basata sulla scomposizione dei frontend delle applicazioni in artefatti sviluppati e distribuiti in modo indipendente. Quando si suddividono frontend di grandi dimensioni in artefatti software autonomi, è possibile incapsulare la logica aziendale e ridurre le dipendenze. Ciò consente una consegna più rapida e frequente di incrementi di prodotto.

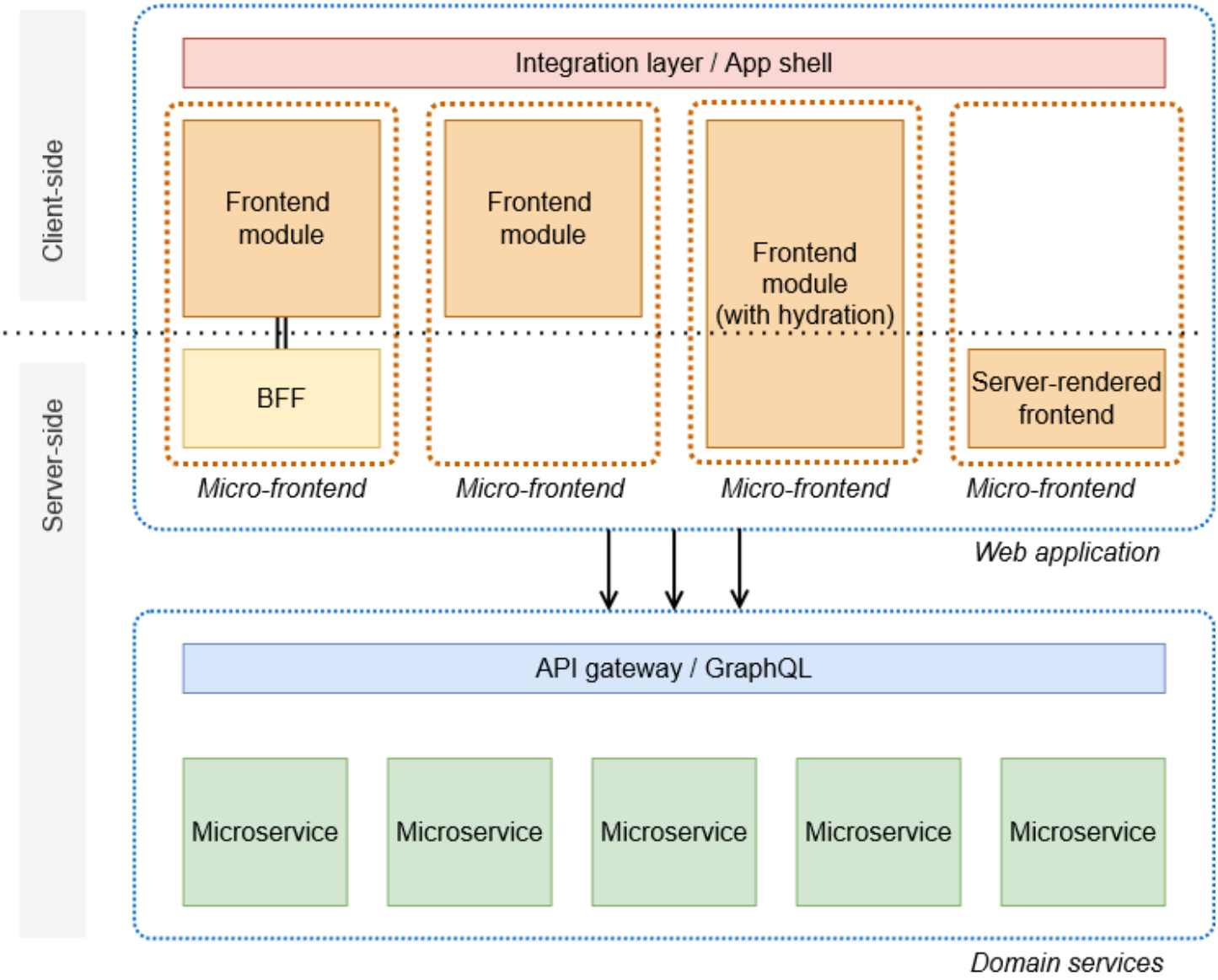
I micro-frontend sono simili ai microservizi. In effetti, il termine micro-frontend deriva dal termine microservice e mira a trasmettere la nozione di microservizio come frontend. Mentre un'architettura di microservizi in genere combina un sistema distribuito nel backend con un frontend monolitico, i microfrontend sono servizi di frontend distribuiti autonomi. Questi servizi possono essere configurati in due modi:

- Solo frontend, si integra con un livello API condiviso dietro il quale esegue un'architettura di microservizi
- Full stack, il che significa che ogni micro-frontend ha la propria implementazione di backend.

Il diagramma seguente mostra un'architettura di microservizi tradizionale, con un monolite di frontend che utilizza un gateway API per connettersi ai microservizi di backend.



Il diagramma seguente mostra un'architettura di micro-frontend con diverse implementazioni di microservizi.



Come illustrato nel diagramma precedente, è possibile utilizzare i micro-frontend con architetture di rendering lato client o lato server:

- I micro-frontend renderizzati lato client possono utilizzare direttamente le API esposte da un API Gateway centralizzato.
- Il team può creare un backend-for-frontend (BFF) all'interno del contesto limitato per ridurre la chiacchierata del frontend nei confronti delle API.
- Sul lato server, i microfrontend possono essere espressi con un approccio lato server potenziato sul lato client utilizzando una tecnica chiamata idratazione. Quando una pagina viene renderizzata dal browser, la pagina associata JavaScript viene idratata per consentire interazioni con gli elementi dell'interfaccia utente, come fare clic su un pulsante.

- I micro-frontend possono eseguire il rendering sul backend e utilizzare collegamenti ipertestuali per indirizzare verso una nuova parte di un sito Web.

I micro-frontend sono la soluzione ideale per le organizzazioni che desiderano eseguire le seguenti operazioni:

- Scala con più team che lavorano sullo stesso progetto.
- Abbraccia la decentralizzazione del processo decisionale, permettendo agli sviluppatori di innovare entro i confini dei sistemi identificati.

Questo approccio riduce significativamente il carico cognitivo sui team, in quanto diventano responsabili di parti specifiche del sistema. Aumenta l'agilità aziendale perché è possibile apportare modifiche a una parte del sistema senza interrompere il resto.

I micro-frontend sono un approccio architettonico distinto. Sebbene esistano diversi modi per creare micro-frontend, hanno tutti tratti comuni:

- Un'architettura di micro-frontend è composta da più elementi indipendenti. La struttura è simile alla modularizzazione che avviene con i microservizi sul backend.
- Un microfrontend è completamente responsabile dell'implementazione del frontend all'interno del suo contesto limitato, che comprende quanto segue:
  - Interfaccia utente
  - Dati
  - Stato o sessione
  - Logica aziendale
  - Flusso

Un contesto limitato è un sistema internamente coerente con confini accuratamente progettati che mediano ciò che può entrare e uscire. Un microfrontend dovrebbe condividere la minor quantità possibile di logica e dati aziendali con altri microfrontend. Ovunque sia necessario, la condivisione avviene tramite interfacce chiaramente definite come eventi personalizzati o flussi reattivi. Tuttavia, quando si tratta di questioni trasversali come un sistema di progettazione o la registrazione delle librerie, la condivisione intenzionale è benvenuta.

Uno schema consigliato è quello di creare micro-frontend utilizzando team interfunzionali. Ciò significa che ogni micro-frontend è sviluppato dallo stesso team che lavora dal backend al frontend. La proprietà del team è fondamentale, dalla codifica all'operatività del sistema in produzione.

Questa guida non intende raccomandare un approccio particolare. Anziché, discute diversi modelli, migliori pratiche, compromessi e considerazioni architettoniche e organizzative.

Concetti fondamentali

L'architettura Micro-frontend è fortemente ispirata a tre concetti architettonici precedenti:

- La progettazione basata sul dominio è il modello mentale per strutturare applicazioni complesse in domini coerenti.
- I sistemi distribuiti sono un approccio per la creazione di applicazioni sotto forma di sottosistemi liberamente accoppiati, sviluppati indipendentemente e eseguiti su una propria infrastruttura dedicata.
- Il cloud computing è un approccio per gestire l'infrastruttura IT come servizi con un modello. pay-as-you-go

Progettazione basata sul dominio

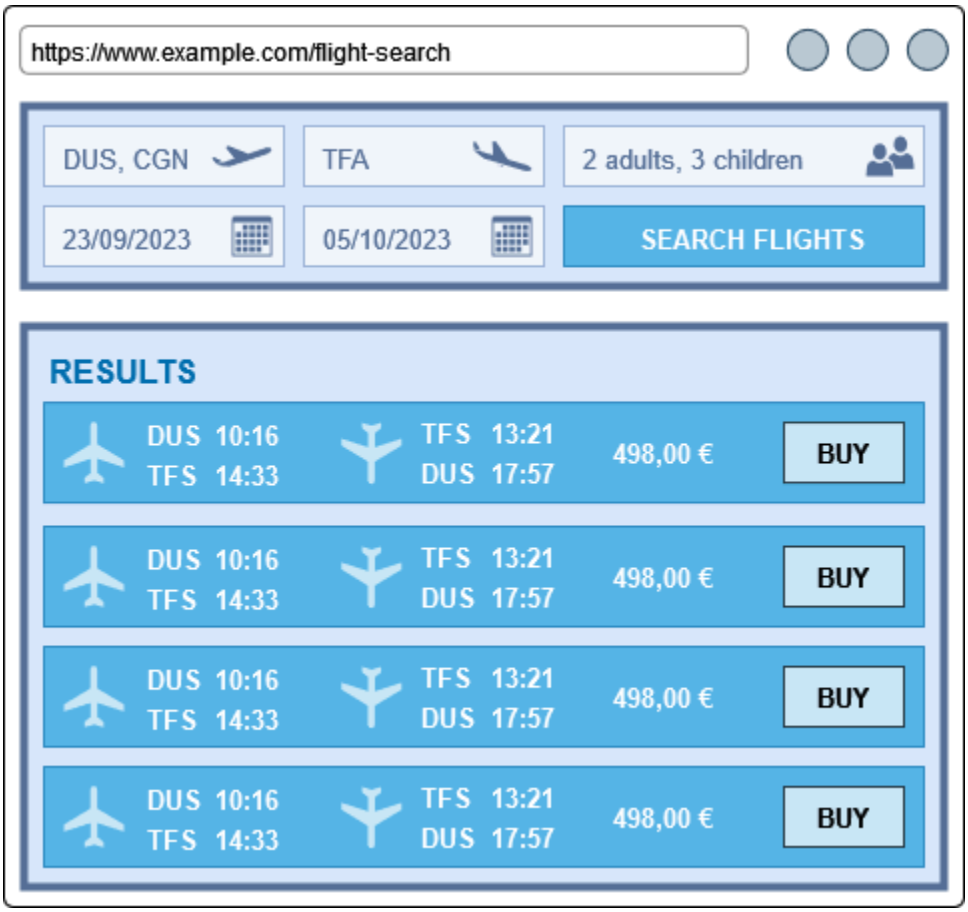
La progettazione basata sul dominio (DDD) è un paradigma sviluppato da Eric Evans. Nel suo libro del 2003 Domain-Driven Design: Tackling Complexity in the Heart of Software, Evans postula che lo sviluppo del software dovrebbe essere guidato da preoccupazioni aziendali piuttosto che tecniche. Evans propone che i progetti IT sviluppino innanzitutto un linguaggio onnipresente che aiuti gli esperti tecnici e di settore a trovare una comprensione condivisa. Sulla base di quel linguaggio, possono formulare un modello della realtà aziendale compreso da entrambe le parti.

Per quanto ovvio possa essere questo approccio, molti progetti software presentano una disconnessione tra business e IT. Queste disconnessioni spesso causano gravi malintesi, che portano a sforamenti di budget, riduzione della qualità o fallimento del progetto.

Evans introduce molti altri termini importanti, uno dei quali è il contesto limitato. Un contesto limitato è un segmento autonomo di un'applicazione IT di grandi dimensioni che contiene la soluzione o l'implementazione per una sola azienda. Un'applicazione di grandi dimensioni sarà costituita da più contesti limitati che sono liberamente accoppiati tramite modelli di integrazione. Questi contesti limitati possono persino avere i propri dialetti della lingua onnipresente. Ad esempio, un utente nel contesto di pagamento di un'applicazione potrebbe avere aspetti diversi da un utente nel contesto della consegna perché il concetto di spedizione sarebbe irrilevante durante il pagamento.

Evans non definisce quanto piccolo o grande debba essere un contesto limitato. La dimensione è determinata dal progetto software e potrebbe evolversi nel tempo. I buoni indicatori dei confini di un contesto sono il grado di coesione tra le entità (oggetti di dominio) e la logica aziendale.

Nel contesto dei microfrontend, la progettazione basata sul dominio può essere illustrata con l'esempio di una pagina web complessa come una pagina di prenotazione di voli.



In questa pagina, gli elementi costitutivi principali sono un modulo di ricerca, un pannello di filtri e l'elenco dei risultati. Per identificare i confini, è necessario identificare contesti funzionali indipendenti. Inoltre, prendete in considerazione aspetti non funzionali, come la riutilizzabilità, le prestazioni e la sicurezza. L'indicatore più importante del fatto che «cose che stanno insieme» sono i loro modelli di comunicazione. Se alcuni elementi di un'architettura devono comunicare frequentemente e scambiarsi informazioni complesse, è probabile che condividano lo stesso contesto limitato.

I singoli elementi dell'interfaccia utente, come i pulsanti, non sono contesti limitati, perché non sono indipendenti dal punto di vista funzionale. Inoltre, l'intera pagina non è adatta a un contesto limitato, perché può essere suddivisa in contesti indipendenti più piccoli. Un approccio ragionevole consiste nel trattare il modulo di ricerca come un contesto limitato e l'elenco dei risultati come il secondo contesto limitato. Ciascuno di questi due contesti limitati può ora essere implementato come un micro-frontend separato.

Sistemi distribuiti

Per facilitare la manutenzione e supportare la capacità di evoluzione, la maggior parte delle soluzioni IT non banali è modulare. In questo caso, modulare significa che i sistemi IT sono costituiti da elementi costitutivi identificabili che vengono disaccoppiati tramite interfacce per ottenere la separazione delle preoccupazioni.

Oltre ad essere modulari, i sistemi distribuiti dovrebbero essere sistemi indipendenti a pieno titolo. In un sistema puramente modulare, ogni modulo è idealmente incapsulato ed espone le sue funzioni tramite interfacce, ma non può essere distribuito in modo indipendente o addirittura funzionare da solo. Inoltre, i moduli generalmente seguono lo stesso ciclo di vita degli altri moduli che fanno parte dello stesso sistema. Gli elementi costitutivi di un sistema distribuito, d'altra parte, hanno ciascuno il proprio ciclo di vita. Applicando il paradigma di progettazione basato sul dominio, ogni elemento costitutivo si rivolge a un dominio o sottodominio aziendale e vive in un contesto limitato.

Quando i sistemi distribuiti interagiscono durante la fase di costruzione, un approccio comune consiste nello sviluppare meccanismi per identificare rapidamente i problemi. Ad esempio, potreste adottare linguaggi tipizzati e investire molto nei test unitari. Più team possono collaborare allo sviluppo e alla manutenzione di moduli, spesso distribuiti come librerie che i sistemi possono utilizzare con strumenti come npm, Apache Maven e pip. NuGet

In fase di esecuzione, i sistemi distribuiti interagenti sono in genere di proprietà di singoli team. L'uso eccessivo delle dipendenze causa complessità operativa a causa della gestione degli errori, del bilanciamento delle prestazioni e della sicurezza. Gli investimenti nei test di integrazione e nell'osservabilità sono fondamentali per ridurre i rischi.

Gli esempi più diffusi di sistemi distribuiti oggi sono i microservizi. Nelle architetture a microservizi, i servizi di backend sono guidati dal dominio (anziché da problemi tecnici come l'interfaccia utente o l'autenticazione) e sono di proprietà di team autonomi. I microfrontend condividono gli stessi principi, estendendo l'ambito della soluzione al frontend.

Cloud computing

Il cloud computing è un modo per acquistare l'infrastruttura IT sotto forma di servizi con un pay-as-you-go modello anziché costruire i propri data center e acquistare hardware per gestirli localmente. Il cloud computing offre diversi vantaggi:

- La tua organizzazione acquisisce una notevole agilità aziendale grazie alla possibilità di sperimentare nuove tecnologie senza dover assumere anticipatamente impegni finanziari ingenti e a lungo termine.
- Utilizzando un provider di servizi cloud come AWS, l'organizzazione può accedere a un ampio portafoglio di servizi a bassa manutenzione e altamente integrabili (come gateway API, database, orchestrazione dei container e funzionalità cloud). L'accesso a questi servizi consente al personale di concentrarsi su attività che differenziano l'organizzazione dalla concorrenza.
- Quando la tua organizzazione è pronta per implementare una soluzione a livello globale, puoi implementarla sull'infrastruttura cloud in tutto il mondo.

Il cloud computing supporta i microfrontend fornendo un'infrastruttura altamente gestita. Ciò semplifica la end-to-end proprietà per i team interfunzionali. Sebbene il team debba avere una solida conoscenza delle operazioni, le attività manuali di fornitura dell'infrastruttura, aggiornamenti del sistema operativo e networking costituirebbero una distrazione.

Poiché i micro-frontend vivono in contesti limitati, i team possono scegliere il servizio più adatto per eseguirli. Ad esempio, i team possono scegliere tra funzioni cloud e contenitori per l'elaborazione e possono scegliere tra diverse versioni di database SQL e NoSQL o cache in memoria. I team possono persino creare i propri micro-frontend su un toolkit altamente integrato, ad esempio dotato di elementi costitutivi preconfigurati per l'infrastruttura AWS Amplifyserverless.

Confronto tra micro-frontend e architetture alternative

Come per tutte le strategie architettoniche, la decisione di adottare i microfrontend deve basarsi su criteri di valutazione guidati dai principi dell'organizzazione. I micro-frontend presentano vantaggi e svantaggi. Se l'organizzazione decide di utilizzare i micro-frontend, è necessario disporre di strategie per affrontare le sfide dei sistemi distribuiti

Quando si sceglie un'architettura applicativa, le alternative più popolari ai microfrontend sono i monoliti, le applicazioni n-tier e i microservizi in combinazione con un frontend applicativo a pagina singola (SPA). Questi sono tutti approcci validi e ognuno di essi presenta vantaggi e svantaggi.

Monoliti

Una piccola applicazione che non richiede modifiche frequenti può essere fornita molto rapidamente come monolite. Anche in situazioni in cui è prevista una crescita significativa, un monolite è un primo passo naturale. Successivamente, il monolite può essere ritirato o rifattorizzato in una struttura più flessibile. Partendo da un monolite, l'organizzazione può andare sul mercato, ottenere il feedback dei clienti e migliorare il prodotto più rapidamente.

Tuttavia, le applicazioni monolitiche tendono a degradarsi se non vengono mantenute con cura o se la codebase cresce di dimensioni nel tempo. Quando più team contribuiscono in modo significativo alla stessa base di codice, raramente tutti contribuiscono alla sua manutenzione e alle sue operazioni. Ciò si traduce in uno squilibrio delle responsabilità, che influisce sulla velocità e causa inefficienze. Allo stesso tempo, l'accoppiamento involontario tra i moduli di un monolite porta a effetti collaterali indesiderati man mano che il codice base si evolve. Questi effetti collaterali possono causare malfunzionamenti e interruzioni.

Applicazioni di livello N

Un'applicazione più complessa con un ritmo di evoluzione relativamente statico può essere costruita come un'architettura a tre livelli (presentazione, applicazione, dati), con un livello REST o GraphQL tra il frontend e il backend. Questa soluzione è molto più flessibile e i team dei diversi livelli possono svilupparsi in modo indipendente in una certa misura. Lo svantaggio di un'applicazione di livello n è che è molto più difficile implementare le funzionalità. Il frontend e il backend sono disaccoppiati tramite un contratto API, quindi le modifiche più importanti devono essere implementate insieme o l'API deve essere versionata.

Considerate il seguente scenario comune: se il rilascio di una nuova funzionalità richiede una modifica dello schema dei dati, i proprietari dei prodotti potrebbero impiegare giorni per concordare una serie di funzionalità con un team di frontend. Quindi il team di frontend chiederà al team di backend di sviluppare e rilasciare la funzionalità da parte sua. Il team di backend collaborerà con i proprietari dei dati per rilasciare un aggiornamento dello schema del database. Successivamente, il team di backend rilascerà una nuova versione dell'API, in modo che il team di frontend possa sviluppare e rilasciare le modifiche. In questo scenario, la propagazione di tutte le modifiche alla produzione potrebbe richiedere settimane o addirittura mesi, poiché ogni team ha il proprio backlog, le proprie priorità e i propri meccanismi per lo sviluppo, il test e il rilascio delle modifiche.

Micro-servizi

In un'architettura di microservizi, il backend è scomposto in piccoli servizi, ognuno dei quali si occupa di una particolare problematica aziendale all'interno di un contesto limitato. Ogni microservizio è inoltre fortemente disaccoppiato dagli altri servizi in quanto presenta un contratto di interfaccia chiaramente definito.

Vale la pena ricordare che i contesti limitati e i contratti di interfaccia dovrebbero esistere anche in monoliti e architetture n-tier ben realizzati. In un'architettura a microservizi, tuttavia, la comunicazione avviene tramite la rete, in genere il protocollo HTTP, e i servizi dispongono di un'infrastruttura di runtime dedicata. Ciò supporta lo sviluppo, la distribuzione e il funzionamento indipendenti di ogni servizio di backend.

Scelta dell'approccio adatto alle proprie esigenze

I monoliti e le architetture n-tier raggruppano più aspetti di dominio in un unico artefatto tecnico. Ciò semplifica la gestione di aspetti quali le dipendenze e il flusso di dati interno, ma rende più difficile l'implementazione di nuove funzionalità. Per mantenere una base di codice coerente, un team spesso investe tempo nel refactoring e nel disaccoppiamento a causa della grande base di codice che deve gestire.

Le applicazioni sviluppate da alcuni team potrebbero non richiedere la complessità aggiuntiva derivante dal passaggio ai microfrontend. Ciò è particolarmente vero se le squadre non pagano le penalità dovute all'elevato numero di accoppiamenti e ai lunghi tempi di attesa per rilasciare le modifiche.

In sintesi, le architetture più complesse e distribuite sono spesso la scelta giusta per applicazioni complesse e in rapida evoluzione. Per le applicazioni di piccole e medie dimensioni, un'architettura

distribuita non è necessariamente superiore a un'architettura monolitica, soprattutto se l'applicazione non si evolverà drasticamente in un breve periodo di tempo.

Decisioni architettoniche nei microfrontend

I team che applicano un modello di architettura micro-frontend per le proprie applicazioni devono prendere diverse decisioni sull'architettura nella fase iniziale:

- Identificazione dei micro-frontend e definizione dei confini
- Composizione di pagine e visualizzazioni con micro-frontend
- Routing, gestione dello stato e comunicazione tra microfrontend
- Gestione delle dipendenze per questioni trasversali

Le sezioni seguenti trattano questi argomenti in modo più approfondito.

Quando si prendono decisioni sull'architettura, è essenziale disporre delle metriche corrette e comprendere i modelli di utilizzo, le caratteristiche delle applicazioni e i compromessi. Ad esempio, un sito di e-commerce presenta caratteristiche e modelli di utilizzo diversi rispetto a uno strumento di montaggio video o a dashboard di osservabilità.

Le applicazioni destinate al pubblico con traffico elevato e profondità di sessione breve possono essere ottimizzate per le metriche iniziali di caricamento della pagina come Time to Interactive (TTI) e First Contentful Paint (FCP). Al contrario, un'applicazione a cui gli utenti accedono all'inizio della giornata e con cui continuano a interagire per tutto il giorno potrebbe essere ottimizzata per l'esperienza all'interno dell'applicazione. Il team dell'applicazione potrebbe ottimizzare la metrica First Input Delay (FID) dopo ogni navigazione anziché il caricamento iniziale della pagina.

I siti Web pubblici devono soddisfare diversi ambienti di browser. Le applicazioni aziendali con vincoli noti sull'ambiente client possono ottimizzare la composizione dei microfrontend in base ai propri vincoli.

Non esiste un'unica scelta giusta per le decisioni relative all'architettura. Comprendi i compromessi, il contesto in cui opera l'azienda, i modelli di utilizzo e le metriche per guidare le decisioni adatte a ogni singola applicazione.

Identificazione dei confini dei microfrontend

Per migliorare l'autonomia del team, le funzionalità aziendali fornite da un'applicazione possono essere scomposte in diversi microfrontend con dipendenze minime l'uno dall'altro.

Seguendo la metodologia DDD discussa in precedenza, i team possono suddividere un dominio applicativo in sottodomini aziendali e contesti limitati. I team autonomi possono quindi possedere la funzionalità dei propri contesti limitati e fornire tali contesti come micro-frontend. Per ulteriori informazioni sulla separazione delle preoccupazioni, consulta il diagramma Serverless Land.

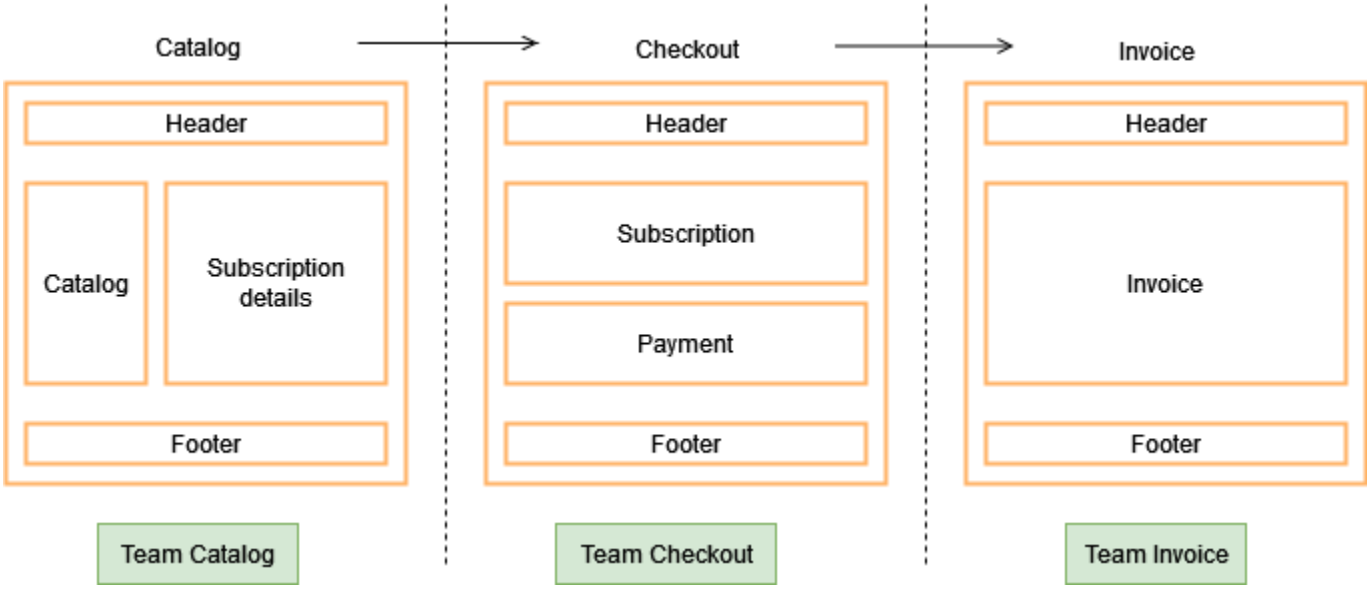
Un contesto limitato ben definito dovrebbe ridurre al minimo la sovrapposizione funzionale e la necessità di comunicazioni in fase di esecuzione tra i contesti. La comunicazione richiesta può essere implementata con metodi basati sugli eventi. Ciò non è diverso dall'architettura basata sugli eventi per lo sviluppo di microservizi.

Un'applicazione ben progettata dovrebbe inoltre supportare la fornitura di future estensioni da parte di nuovi team per offrire un'esperienza coerente ai clienti.

Come suddividere un'applicazione monolitica in microfrontend

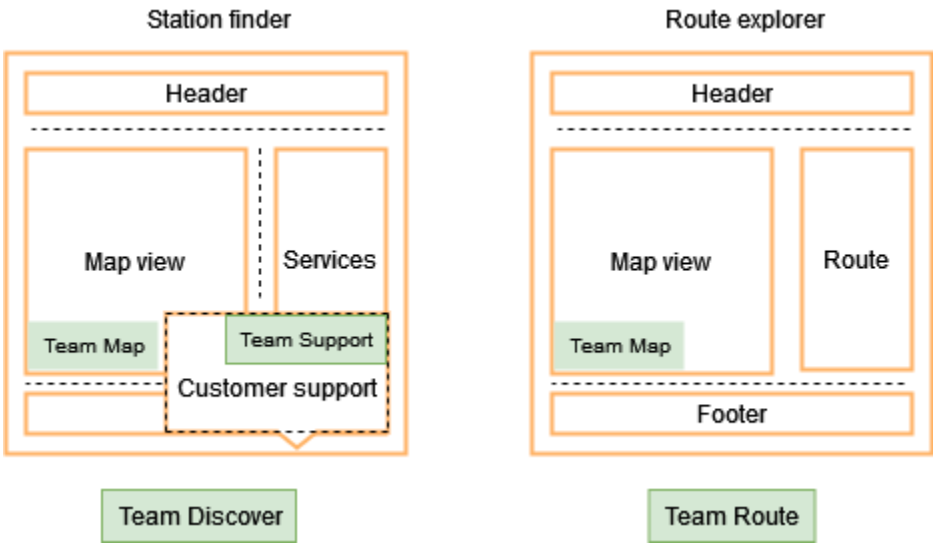
La sezione Panoramica includeva un esempio di identificazione di contesti funzionali indipendenti su una pagina Web. Emergono diversi modelli per suddividere la funzionalità sull'interfaccia utente.

Ad esempio, quando i domini aziendali costituiscono fasi del percorso dell'utente, è possibile applicare una suddivisione verticale sul frontend, in cui una raccolta di visualizzazioni del percorso dell'utente viene fornita sotto forma di microfrontend. Il diagramma seguente mostra una suddivisione verticale, in cui le fasi Catalog, Checkout e Invoice vengono fornite da team separati come microfrontend separati.



Per alcune applicazioni, la divisione verticale da sola potrebbe non essere sufficiente. Ad esempio, potrebbe essere necessario fornire alcune funzionalità in molte visualizzazioni. Per queste

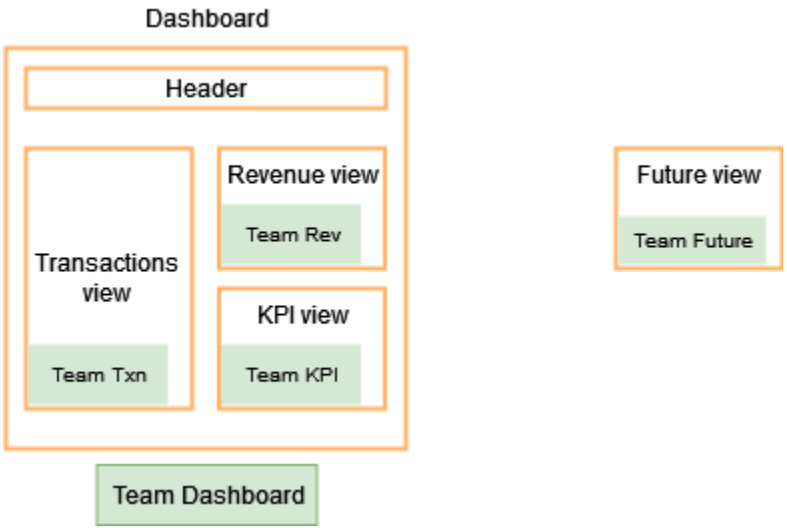
applicazioni, è possibile applicare una suddivisione mista. Il diagramma seguente mostra una soluzione mista suddivisa in cui i microfrontend per Station finder e Route explorer utilizzano entrambi la funzionalità di visualizzazione della mappa.



Le applicazioni di tipo portale o di dashboard in genere riuniscono le funzionalità di frontend in un'unica visualizzazione. In questi tipi di applicazioni, ogni widget può essere fornito come micro-frontend e l'applicazione di hosting definisce i vincoli e le interfacce che i micro-frontend devono implementare.

Questo approccio fornisce ai microfrontend un meccanismo per gestire problemi come il dimensionamento delle viewport, i provider di autenticazione, le impostazioni di configurazione e i metadati. Questi tipi di applicazioni ottimizzano l'estensibilità. Nuove funzionalità possono essere sviluppate da nuovi team per scalare le funzionalità del dashboard.

Il diagramma seguente mostra un'applicazione dashboard sviluppata da tre singoli team che fanno parte di Team Dashboard.



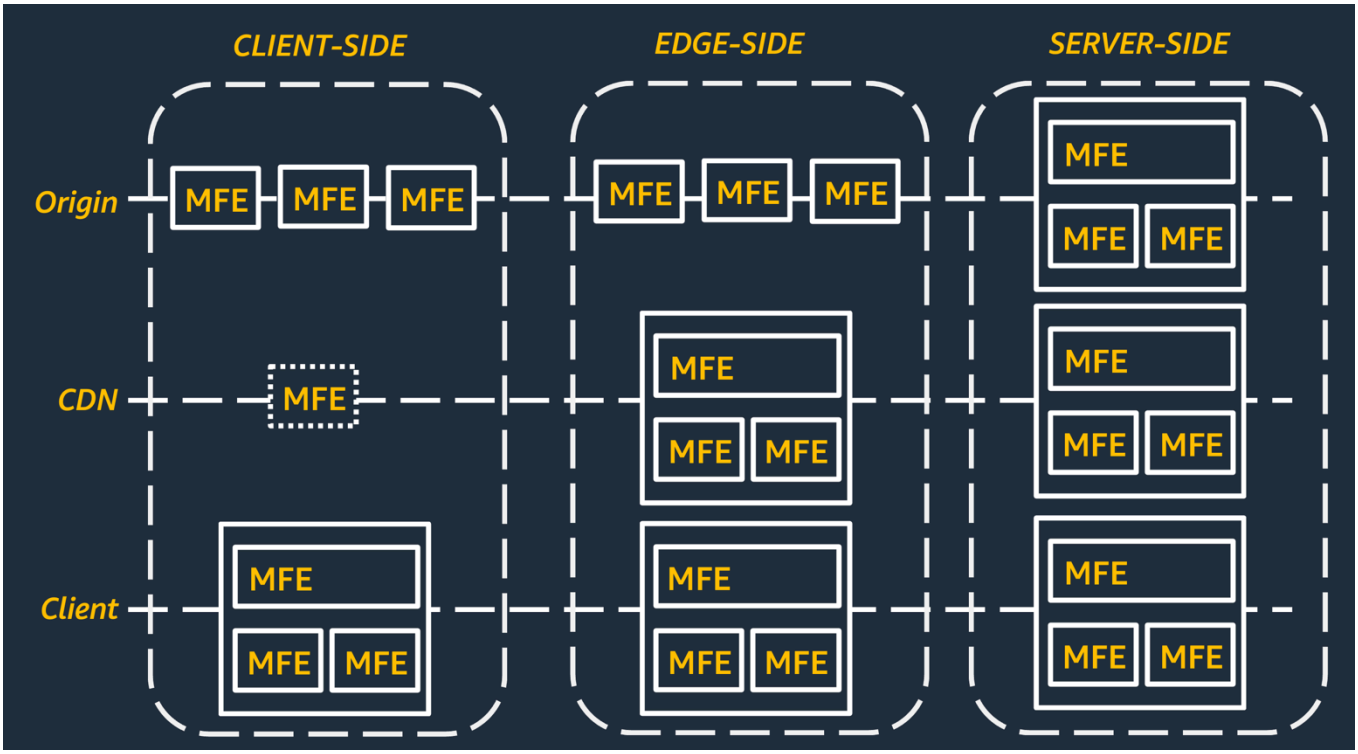
Nel diagramma, la vista futura rappresenta le nuove funzionalità sviluppate dai nuovi team per scalare Team Dashboard e le funzionalità della dashboard.

Le applicazioni di portale e dashboard di solito creano funzionalità utilizzando una suddivisione mista nell'interfaccia utente. I micro-frontend sono configurabili con impostazioni ben definite, inclusi vincoli di posizione e dimensione.

Composizione di pagine e visualizzazioni con micro-frontend

È possibile comporre viste di un'applicazione con composizione lato client, composizione lato bordo e composizione lato server. I modelli di composizione hanno caratteristiche diverse in termini di competenze di squadra necessarie, tolleranza agli errori, prestazioni e comportamento della cache.

Il diagramma seguente mostra come avviene la composizione a livello client, edge-side e lato server di un'architettura di microfrontend.



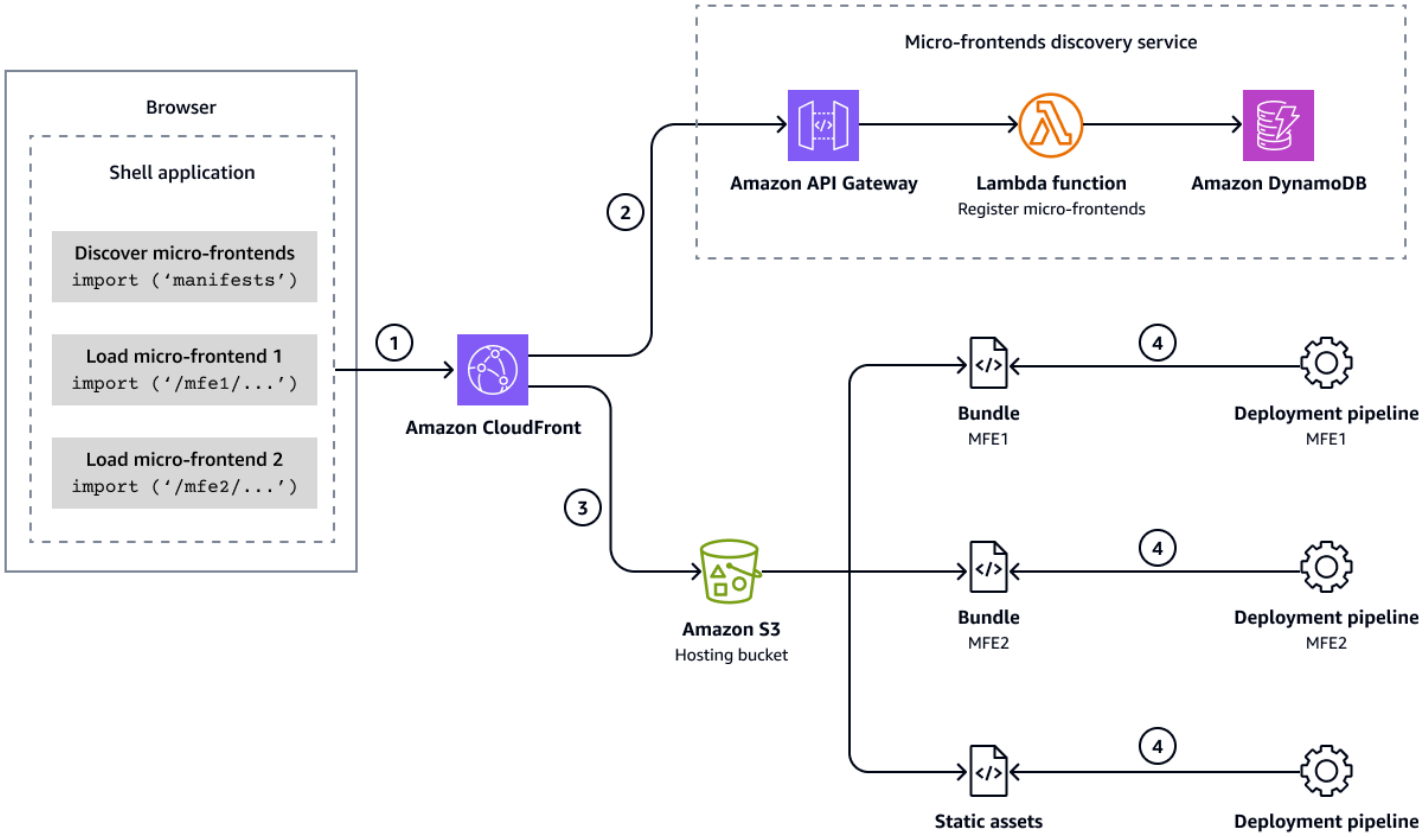
I livelli lato client, lato bordo e lato server sono discussi nelle sezioni seguenti.

Composizione lato client

Carica e aggiungi dinamicamente i micro-frontend come frammenti DOM (Document Object Model) sul client (browser o visualizzazione web mobile). Gli artefatti del micro-frontend, ad esempio i file CSS, possono essere caricati dalle JavaScript reti di distribuzione dei contenuti (CDN) per ridurre la latenza. La composizione lato client richiede quanto segue:

- Un team che possieda e gestisca un'applicazione shell o un framework di microfrontend per consentire l'individuazione, il caricamento e il rendering dei componenti di microfrontend in fase di esecuzione nel browser
- Livelli di competenza elevati nelle tecnologie di frontend come HTML, CSS e comprensione approfondita degli JavaScript ambienti dei browser
- Ottimizzazione della quantità di JavaScript file caricati in una pagina e disciplina per evitare conflitti tra namespace globali

Il diagramma seguente mostra un' AWS architettura di esempio per la composizione lato client senza server.



La composizione lato client avviene nell'ambiente del browser tramite un'applicazione shell. Il diagramma mostra i seguenti dettagli:

1. Dopo aver caricato l'applicazione shell, invia una richiesta iniziale ad Amazon CloudFront per scoprire i micro-frontend da caricare tramite un endpoint manifest.
2. I manifesti contengono informazioni su ogni micro-frontend (ad esempio nome, URL, versione e comportamento di fallback). I manifesti sono forniti dal servizio di scoperta dei microfrontend. Nel diagramma, questo servizio di discovery è rappresentato da Amazon API Gateway, una AWS Lambda funzione e Amazon DynamoDB. L'applicazione shell utilizza le informazioni del manifesto per richiedere ai singoli microfrontend di comporre la pagina all'interno di un determinato layout.
3. Ogni pacchetto di micro-frontend è composto da file statici (come JavaScript CSS e HTML). I file sono ospitati in un bucket Amazon Simple Storage Service (Amazon S3) e serviti tramite. CloudFront
4. I team possono implementare nuove versioni dei loro micro-frontend e aggiornare le informazioni del manifesto utilizzando pipeline di distribuzione di loro proprietà.

Composizione laterale

Utilizza tecniche di transclusione come Edge Side Includes (ESI) o Server Side Includes (SSI) supportate da alcuni CDN e proxy davanti ai server di origine per comporre una pagina prima di inviarla via cavo ai client. ESI richiede quanto segue:

- Un CDN con funzionalità ESI o una distribuzione proxy davanti a micro-frontend lato server. Le implementazioni proxy come HAProxy, Varnish e NGINX supportano SSI.
- Comprensione dell'uso e dei limiti delle implementazioni ESI e SSI.

I team che iniziano nuove applicazioni in genere non scelgono la composizione laterale come modello di composizione. Tuttavia, questo modello potrebbe fornire un percorso per le applicazioni legacy che si basano sulla transclusione.

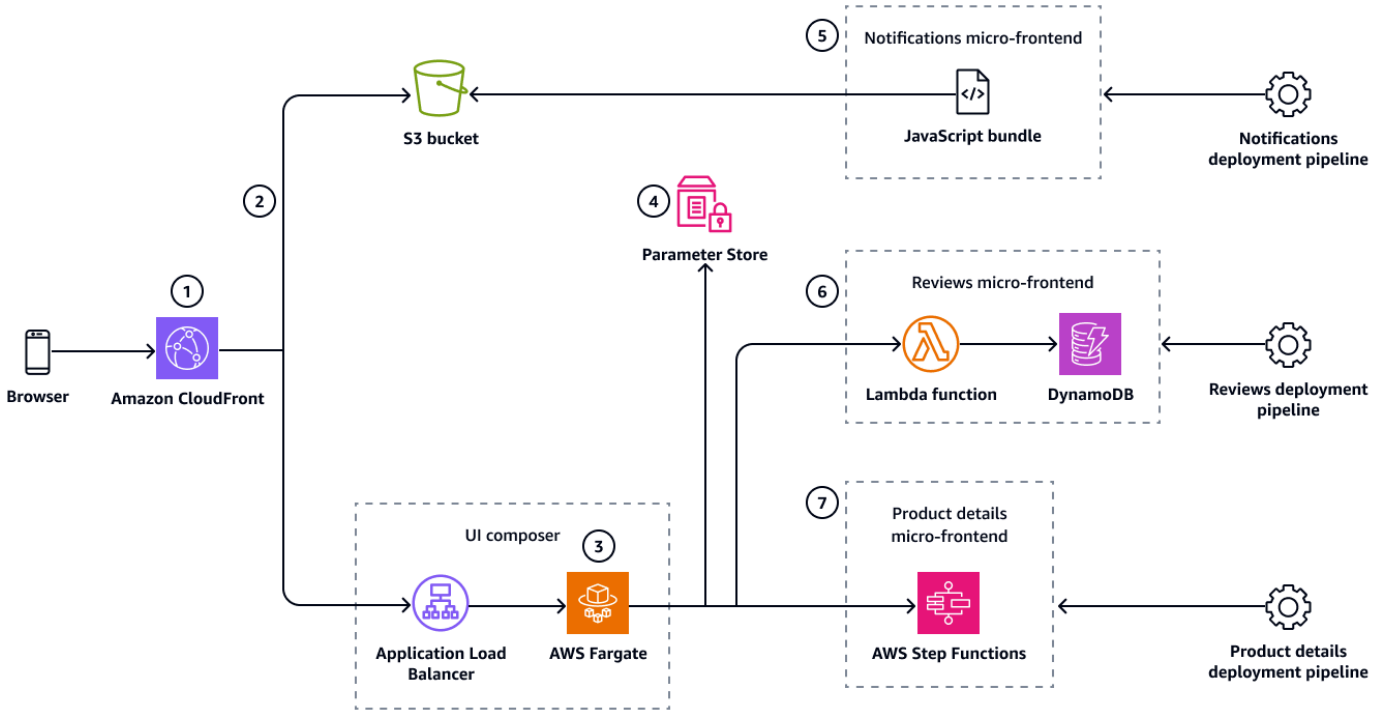
Composizione lato server

Utilizzate i server di origine per comporre le pagine prima che vengano memorizzate nella cache periferica. Questo può essere fatto con tecnologie tradizionali, come PHP, Jakarta Server Pages (JSP) o librerie di modelli, per comporre le pagine includendo frammenti di microfrontend. È inoltre possibile utilizzare JavaScript framework, come Next.js, in esecuzione sul server per comporre pagine sul server con rendering lato server (SSR).

Dopo aver eseguito il rendering delle pagine sul server, è possibile memorizzarle nella cache sui CDN per ridurre la latenza. Quando vengono implementate nuove versioni di micro-frontend, le pagine devono essere nuovamente renderizzate e la cache deve essere aggiornata per fornire le versioni più recenti ai clienti.

La composizione lato server richiede una conoscenza approfondita dell'ambiente server per stabilire i modelli di implementazione, l'individuazione di microfrontend lato server e la gestione della cache.

Il diagramma seguente mostra la composizione lato server.



Il diagramma include i seguenti componenti e processi:

1. Amazon CloudFront fornisce un punto di accesso unico all'applicazione. La distribuzione ha due origini: la prima per i file statici e la seconda per il compositore dell'interfaccia utente.
2. I file statici sono ospitati in un bucket Amazon S3. Vengono utilizzati dal browser e dal compositore dell'interfaccia utente per i modelli HTML.
3. Il compositore dell'interfaccia utente viene eseguito su un cluster di contenitori in. AWS Fargate Con una soluzione containerizzata, puoi utilizzare le funzionalità di streaming e il rendering multithread, se necessario.
4. Parameter Store, una funzionalità di AWS Systems Manager, viene utilizzato come sistema di rilevamento di microfrontend di base. Questa funzionalità fornisce un archivio chiave-valore utilizzato dal compositore dell'interfaccia utente per recuperare gli endpoint di micro-frontend da utilizzare.
5. Il micro-frontend delle notifiche archivia il pacchetto ottimizzato nel bucket S3. JavaScript Questo viene visualizzato sul client perché deve reagire alle interazioni dell'utente.
6. Il micro-frontend delle recensioni è composto da una funzione Lambda e le recensioni degli utenti sono archiviate in DynamoDB. Il microfrontend delle recensioni viene renderizzato completamente sul lato server e genera un frammento HTML.

7. Il micro-frontend dei dettagli del prodotto è un microfrontend a basso codice che utilizza. AWS Step Functions Express Workflow può essere richiamato in modo sincrono e contiene la logica per il rendering del frammento HTML e un livello di memorizzazione nella cache.

Per ulteriori informazioni sulla composizione lato server, consultate il post del blog Server-side rendering micro-frontends — the architecture.

Routing e comunicazione tra microfrontend

Le opzioni di routing dipendono dall'approccio di composizione. La comunicazione può essere ottimizzata riducendo l'accoppiamento tra i componenti del frontend.

Routing

Le applicazioni che utilizzano la composizione lato client con divisione verticale possono utilizzare il routing lato server (applicazione multipagina) o il routing lato client (applicazione a pagina singola). Se utilizzano una suddivisione mista per la composizione dell'interfaccia utente, il routing lato client è necessario per supportare gerarchie di routing più profonde dei micro-frontend su una pagina.

Le applicazioni che utilizzano la composizione edge-side e la composizione lato server si allineano meglio con il routing lato server o il routing con l'edge computing come Lambda @Edge con Amazon. CloudFront

Comunicazione tra micro-frontend

Con le architetture micro-frontend, consigliamo di ridurre l'accoppiamento tra i componenti del frontend. Un approccio per ridurre l'accoppiamento consiste nel passare dalle chiamate di funzione sincrone alla messaggistica asincrona.

I runtime del browser e le interazioni con gli utenti sono asincroni per natura. Gli eventi possono essere scambiati tra produttori e consumatori tramite messaggi. Gli eventi forniscono un'interfaccia ben definita per la comunicazione tra microfrontend.

Se segui le pratiche DDD per identificare i tuoi contesti limitati per i microfrontend, il passaggio successivo consiste nell'identificare gli eventi che devono essere comunicati oltre i confini.

Il meccanismo di messaggistica per gli eventi può essere costituito da eventi DOM nativi (`CustomEvents`), emettitori di eventi o JavaScript librerie di flussi reattivi fornite dai team della

piattaforma. I micro-frontend pubblicano eventi e si iscrivono a eventi pertinenti per il loro contesto limitato. Con questo metodo, gli editori e gli abbonati non devono conoscersi a vicenda. Il contratto è la definizione dell'evento. Per una rappresentazione visiva di ciò, vedere la sezione Comunica con gli eventi del diagramma Bounded context with Event Architectures.

Gestione delle dipendenze per questioni trasversali

La gestione consapevole delle dipendenze è fondamentale per il successo di un'architettura distribuita come i microfrontend. La gestione delle dipendenze è una delle parti più impegnative dello sviluppo di microfrontend.

In un'architettura micro-frontend, due aspetti importanti della gestione delle dipendenze sono la riduzione delle prestazioni dovuta al trasferimento di artefatti di codice di grandi dimensioni al client e il sovraccarico delle risorse di calcolo. Idealmente, l'organizzazione deve stabilire come vengono mantenute le dipendenze in un'architettura frontend distribuita.

Tre strategie valide per imporre il mantenimento delle dipendenze sono «share nothing», utilizzando standard web come mappe di importazione e federazione dei moduli. Altri approcci sono anti-pattern perché violano i principi di base delle architetture distribuite.

Non condividete nulla, ove possibile

L'approccio share-nothing postula che nessuna dipendenza tra artefatti software indipendenti debba essere condivisa affatto, o almeno non durante l'integrazione o il runtime. Ciò significa che se due micro-frontend dipendono dalla stessa libreria, ognuno deve essere inserito nella libreria in fase di compilazione e spedito separatamente. Inoltre, ogni microfrontend deve verificare che la libreria non inquini i namespace globali e le risorse condivise.

Ciò porta a ridondanze, ma è un compromesso consapevole con la massima agilità. Senza dipendenze di runtime condivise, i team hanno la massima flessibilità per far evolvere il software in qualsiasi modo ritengano utile, purché lo facciano nell'ambito della loro soluzione e non violino alcun contratto di interfaccia.

Su una piattaforma in cui i microfrontend seguono il principio «share-nothing», è importante mantenere i microfrontend il più leggeri possibile. Richiede sviluppatori esperti e diligenti nell'ottimizzare i propri microfrontend in termini di prestazioni e che non sacrifichino l'esperienza utente per l'esperienza degli sviluppatori.

Quando condividi codice

Quando decidi di condividere del codice, puoi condividerlo come librerie o moduli di runtime. Ad esempio, il team principale del frontend fornisce librerie per l'utilizzo di microfrontend tramite CDN. I business value team possono caricare le librerie in fase di esecuzione oppure utilizzare gli archivi di pacchetti per pubblicare le proprie librerie. I team di Micro-frontend possono sviluppare utilizzando una versione specifica della libreria pacchettizzata in fase di compilazione, in modo simile alle applicazioni mobili che utilizzano framework ibridi.

Una terza opzione consiste nell'utilizzare un registro dei pacchetti privato per supportare l'integrazione in fase di compilazione di librerie comuni. Ciò riduce il rischio che una modifica sostanziale del contratto di libreria provochi errori in fase di esecuzione. Tuttavia, questo approccio più conservativo richiede una maggiore governance per sincronizzare tutti i microfrontend con le versioni più recenti della libreria.

Per migliorare i tempi di caricamento delle pagine, i micro-frontend possono esternalizzare le dipendenze della libreria da caricare dai blocchi memorizzati nella cache di un CDN come Amazon. CloudFront

Per gestire le dipendenze di runtime, i micro-frontend possono utilizzare import-maps (o librerie come`System.js`) per specificare da dove viene caricato ogni modulo in fase di runtime. webpack Module Federation è un altro approccio per puntare a una versione ospitata di un modulo remoto e risolvere dipendenze comuni tra micro-frontend indipendenti.

Un altro approccio consiste nel facilitare il caricamento dinamico delle mappe di importazione con una richiesta iniziale a un endpoint di rilevamento.

Stato condiviso

Per ridurre l'accoppiamento dei microfrontend, è importante evitare una gestione globale dello stato accessibile da tutti i microfrontend nella stessa visualizzazione, in modo simile alle architetture monolitiche. Ad esempio, avere un negozio Redux globale accessibile da tutti i microfrontend aumenta l'accoppiamento.

Uno schema per eliminare lo stato condiviso consiste nell'incapsularlo all'interno di microfrontend e comunicare con messaggi asincroni, come discusso in precedenza.

Se assolutamente necessario, introduci interfacce ben definite per lo stato globale e opta per la condivisione in sola lettura per evitare comportamenti imprevisti:

- Quando è presente una divisione verticale, puoi utilizzare i componenti URL e l'archiviazione del browser per accedere alle informazioni dall'ambiente host.
- In caso di suddivisione mista, puoi anche utilizzare gli eventi o le JavaScript librerie personalizzati standard DOM, come emettitori di eventi o flussi bidirezionali, per passare informazioni ai microfrontend.

Se hai bisogno di condividere diverse informazioni tra i microfrontend, ti consigliamo di rivisitare i confini dei microfrontend. La necessità di condivisione potrebbe essere causata dall'evoluzione del business o da una progettazione iniziale scadente.

È anche possibile utilizzare sessioni lato server, in cui ogni micro-frontend recupera i dati richiesti utilizzando un identificatore di sessione. Per ridurre l'accoppiamento, è importante eliminare lo stato condiviso e mantenere separati i dati di sessione specifici del microfrontend.

Framework e strumenti

Non mancano i framework di frontend, come Angular e Next.js, ma la maggior parte di essi non sono creati pensando ai microfrontend. Pertanto, a volte mancano meccanismi per affrontare le sfide dell'architettura micro-frontend.

Considerazioni generali sul quadro

Questa guida non ha lo scopo di consigliare o confrontare singoli framework. Poiché più micro-frontend vengono spesso eseguiti sulla stessa pagina dell'applicazione Web, le prestazioni di caricamento e di runtime sono le principali preoccupazioni. È importante scegliere un framework che introduca il minor sovraccarico possibile.

I framework sono suddivisi in base al livello di rendering:

- Rendering lato client (CSR)
- Rendering lato server (SSR)

Le architetture frontend includono altre funzionalità, come la generazione statica di siti (SSG). Tuttavia, SSG viene eseguito una sola volta. I micro-frontend sono composti principalmente in fase di esecuzione, quindi CSR e SSR sono le opzioni principali.

Rendering lato client

Per quanto riguarda la CSR, ci sono due opzioni popolari:

- Framework SPA singolo
- Federazione dei moduli

Single SPA è una scelta leggera per la composizione di micro-frontend. Risolve le sfide più comuni nelle architetture di microfrontend, come la composizione di più microfrontend nella stessa pagina ed evitare conflitti di dipendenza.

Module Federation è nato come plug-in, offerto da webpack 5, e risolve la maggior parte delle sfide nelle architetture di microfrontend, inclusa la gestione delle dipendenze tra diversi artefatti. Module Federation 2.0 funziona nativamente con Rspack, webpack, esbuild e ora con. JavaScript

Prendi in considerazione l'idea di non utilizzare affatto un framework. I browser moderni, con una quota di mercato complessiva del 98 percento secondo caniuse.com, offrono funzionalità come elementi personalizzati in modo nativo e sono adeguati per un'applicazione di micro-frontend. Se necessario, combinate elementi personalizzati con librerie leggere per la propagazione di eventi, l'internazionalizzazione o altri problemi specifici.

Rendering lato server

Sul lato SSR, le due opzioni principali sono più complicate:

- Abbraccia un framework esistente come Next.js e applica un principio di micro-frontend che utilizza Module Federation.
- Usa HTML- over-the-wire per scambiare frammenti HTML che rappresentano micro-frontend e componi questi frammenti all'interno di un modello in fase di esecuzione. Un esempio di questo approccio è Podium.

Integrazione API – Backend per frontend

Il pattern Backends for Frontends (BFF) viene in genere utilizzato in ambienti di microservizi. Nel contesto dei microfrontend, un BFF è un servizio lato server che appartiene a un microfrontend. Non tutti i micro-frontend devono avere un BFF. Tuttavia, se utilizzi un BFF, deve essere eseguito all'interno dello stesso contesto limitato e non essere condiviso tra altri contesti limitati.

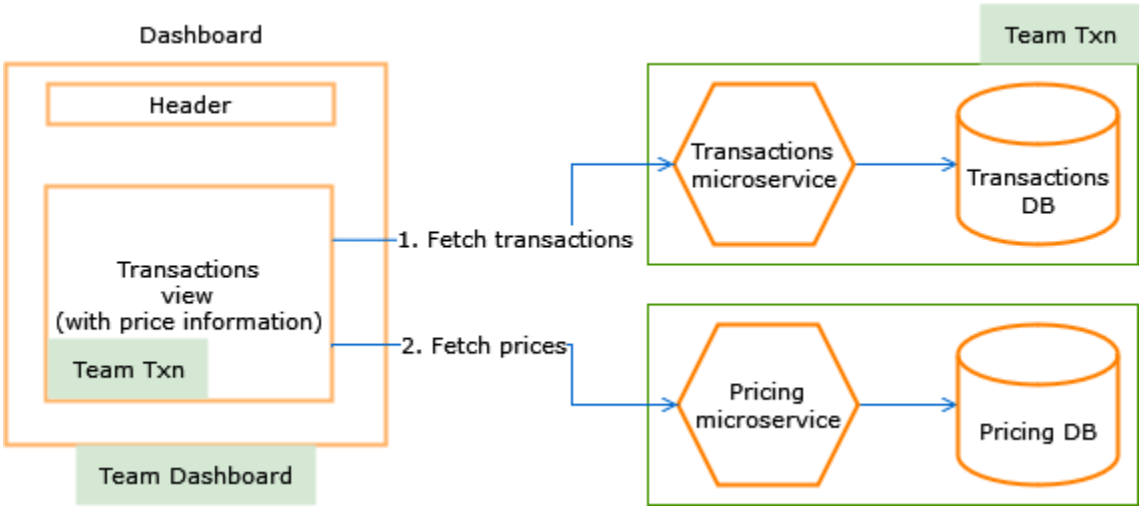
A differenza di un servizio tradizionale, un BFF non segue un modello di dominio. Si tratta invece di un livello API che consente al micro-frontend di preelaborare i dati prima che raggiungano il client. Le aree in cui ciò è utile includono le seguenti:

- Autorizzazione verso API private
- Aggregazione di dati provenienti da diverse fonti
- Trasformazione dei dati per ridurre il carico di rete e facilitare il consumo di dati da parte del cliente

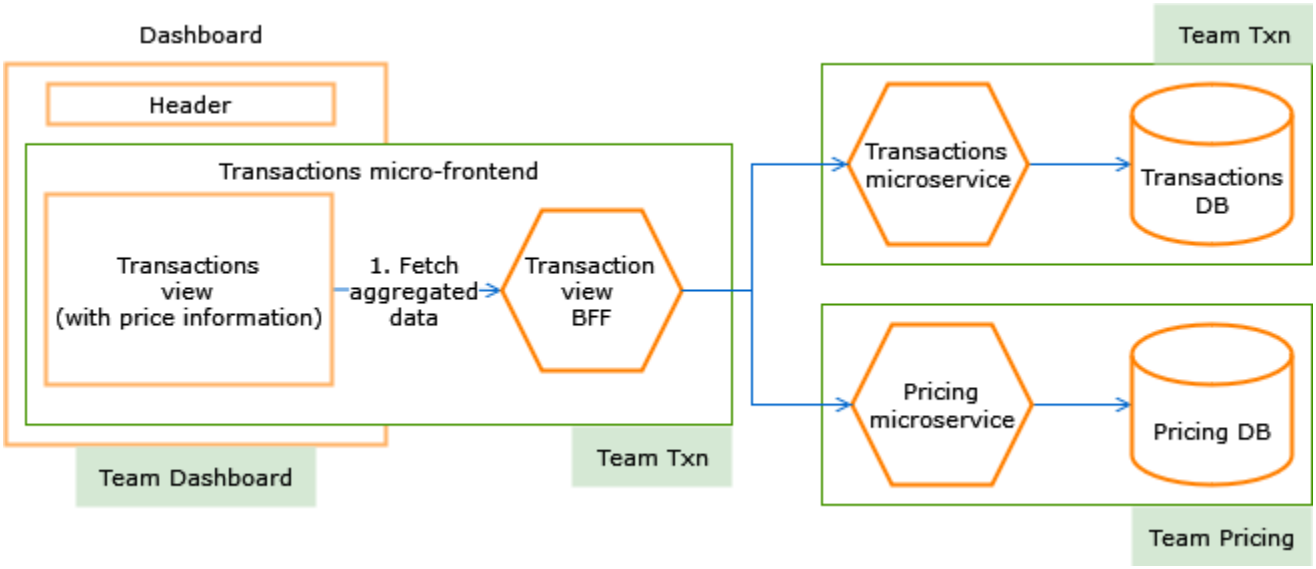
Pertanto, un BFF è di proprietà del micro-frontend, non del livello di servizio del dominio. I BFF possono essere implementati utilizzando quanto segue:

- API AWS AppSync GraphQL
- Un set di funzioni AWS Lambda
- Come contenitore in esecuzione su Amazon ECS, Amazon EKS o AWS AppRunner

Il diagramma seguente mostra che senza il pattern BFF, i microfrontend devono connettersi ai singoli endpoint delle API di microservizi per recuperare e aggregare i dati.



Invece, con il pattern BFF illustrato nel diagramma seguente, i microfrontend possono comunicare con il proprio backend e recuperare dati aggregati.



I team possono sviluppare BFF per diversi canali come dispositivi mobili, web o visualizzazioni specifiche, con requisiti per ottimizzare le interazioni di backend riducendo le chiacchiere.

Styling e CSS

I Cascading Style Sheets (CSS) sono un linguaggio per determinare centralmente la presentazione di un documento anziché la formattazione rigida per testo e oggetti. La funzionalità a cascata del linguaggio è progettata per controllare le priorità tra gli stili utilizzando l'ereditarietà. Quando lavori su micro-frontend e crei una strategia per gestire le dipendenze, la funzionalità a cascata del linguaggio può essere una sfida.

Ad esempio, due micro-frontend coesistono sulla stessa pagina, ognuno dei quali definisce il proprio stile per l'elemento HTML. `body` Se ognuno recupera il proprio file CSS e lo allega al DOM utilizzando un `style` tag, il file CSS sostituisce il primo se entrambi hanno una definizione per elementi HTML comuni, nomi di classi o ID di elementi. Esistono diverse strategie per affrontare questi problemi, a seconda della strategia di dipendenza scelta per la gestione degli stili.

Attualmente, l'approccio più diffuso per bilanciare prestazioni, coerenza ed esperienza degli sviluppatori consiste nello sviluppo e nella manutenzione di un sistema di progettazione.

Sistemi di progettazione – Un approccio basato sulla condivisione

Questo approccio utilizza un sistema per condividere lo stile quando appropriato, supportando al contempo divergenze occasionali per bilanciare coerenza, prestazioni ed esperienza degli sviluppatori. Un sistema di progettazione è una raccolta di componenti riutilizzabili, guidata da standard chiari. Lo sviluppo del sistema di progettazione è in genere guidato da un team con il contributo e il contributo di più team. In termini pratici, un sistema di progettazione è un modo per condividere elementi di basso livello che possono essere esportati come JavaScript libreria. Gli sviluppatori di Micro-frontend possono utilizzare la libreria come dipendenza per creare interfacce semplici componendo risorse disponibili predefinite e come punto di partenza per creare nuove interfacce.

Consideriamo l'esempio di un micro-frontend che necessita di un modulo. L'esperienza tipica degli sviluppatori consiste nell'utilizzare componenti predefiniti disponibili nel sistema di progettazione per comporre caselle di testo, pulsanti, elenchi a discesa e altri elementi dell'interfaccia utente. Lo sviluppatore non ha bisogno di scrivere uno stile per i componenti effettivi, ma solo per il loro aspetto insieme. Il sistema da compilare e rilasciare può utilizzare webpack Module Federation o un approccio simile per dichiarare il sistema di progettazione come dipendenza esterna, in modo che la logica del modulo sia impacchettata senza includere il sistema di progettazione.

Più microfrontend possono quindi fare lo stesso per occuparsi delle preoccupazioni condivise. Quando i team sviluppano nuovi componenti che possono essere condivisi tra più microfrontend, tali componenti vengono aggiunti al sistema di progettazione una volta raggiunta la maturità.

Uno dei principali vantaggi dell'approccio del sistema di progettazione è l'elevato livello di coerenza. Sebbene i microfrontend possano scrivere stili e occasionalmente sovrascrivere quelli del sistema di progettazione, ce n'è ben poco bisogno. I principali elementi di basso livello non cambiano spesso e offrono funzionalità di base che sono estendibili di default. Un altro vantaggio sono le prestazioni. Con una buona strategia di compilazione e rilascio, è possibile produrre pacchetti condivisi minimi che sono strumentati dalla shell dell'applicazione. È possibile migliorare ulteriormente quando più pacchetti specifici per microfrontend vengono caricati in modo asincrono su richiesta, con un ingombro minimo in termini di larghezza di banda di rete. Ultimo ma non meno importante, l'esperienza di sviluppo è ideale perché le persone possono concentrarsi sulla creazione di interfacce avanzate senza dover reinventare la ruota (come la scrittura JavaScript e il CSS ogni volta che è necessario aggiungere un pulsante a una pagina).

Il rovescio della medaglia è che un sistema di progettazione di qualsiasi tipo è una dipendenza, quindi deve essere mantenuto e talvolta aggiornato. Se più micro-frontend richiedono una nuova versione di una dipendenza condivisa, puoi utilizzare uno dei seguenti:

- Un meccanismo di orchestrazione in grado di recuperare occasionalmente più versioni di quella dipendenza condivisa senza conflitti
- Una strategia condivisa per far sì che tutti i dipendenti utilizzino una nuova versione

Ad esempio, se tutti i microfrontend dipendono dalla versione 3.0 di un sistema di progettazione ed è disponibile una nuova versione denominata 3.1 da utilizzare in modo condiviso, è possibile implementare dei flag di funzionalità per consentire la migrazione di tutti i microfrontend con il minimo rischio. Per ulteriori informazioni, consultate la sezione Feature flags. Un altro potenziale svantaggio è che i sistemi di progettazione di solito si occupano di qualcosa di più dello stile. Includono anche JavaScript pratiche e strumenti. Questi aspetti richiedono il raggiungimento del consenso attraverso il dibattito e la collaborazione.

L'implementazione di un sistema di progettazione è un buon investimento a lungo termine. È un approccio popolare e dovrebbe essere preso in considerazione da chiunque lavori su architetture frontend complesse. In genere richiede ingegneri di frontend e team di prodotto e progettazione per collaborare e definire meccanismi per interagire tra loro. È importante pianificare l'orario per raggiungere lo stato desiderato. È anche importante avere la sponsorizzazione della leadership in

modo che le persone possano costruire qualcosa di affidabile, ben mantenuto e performante a lungo termine.

CSS completamente incapsulato – Un approccio «share nothing»

Ogni micro-frontend utilizza convenzioni e strumenti per superare la funzionalità a cascata dei CSS. Un esempio è garantire che lo stile di ogni elemento sia sempre associato a un nome di classe anziché all'ID dell'elemento e che i nomi delle classi siano sempre unici. In questo modo, tutto è limitato ai singoli microfrontend e il rischio di conflitti indesiderati è ridotto al minimo. La shell dell'applicazione è in genere responsabile del caricamento degli stili dei microfrontend dopo che sono stati caricati nel DOM, anche se alcuni strumenti raggruppano gli stili utilizzando. JavaScript

Il vantaggio principale di non condividere nulla è il rischio ridotto di introdurre conflitti tra i micro-frontend. Un altro vantaggio è l'esperienza dello sviluppatore. Ogni micro-frontend non condivide nulla con gli altri micro-frontend. Il rilascio e il test in modo isolato sono più semplici e veloci.

Uno dei principali svantaggi dell'approccio share-nothing è la potenziale mancanza di coerenza. Non esiste alcun sistema per valutare la coerenza. Anche se l'obiettivo è duplicare ciò che è condiviso, diventa difficile trovare un equilibrio tra velocità di rilascio e collaborazione. Una mitigazione comune consiste nel creare strumenti per misurare la coerenza. Ad esempio, puoi creare un sistema per acquisire schermate automatiche di più micro-frontend renderizzati in una pagina con un browser headless. È quindi possibile rivedere manualmente gli screenshot prima di una versione. Tuttavia, ciò richiede disciplina e governance. Per ulteriori informazioni, consulta la sezione Bilanciamento dell'autonomia con l'allineamento.

A seconda del caso d'uso, un altro potenziale svantaggio sono le prestazioni. Se tutti i micro-frontend utilizzano una grande quantità di stile, il cliente deve scaricare molto codice duplicato. Ciò influirà negativamente sull'esperienza dell'utente.

Questo approccio basato sulla condivisione di nulla dovrebbe essere preso in considerazione solo per le architetture di microfrontend che coinvolgono solo pochi team o per i microfrontend che possono tollerare una bassa coerenza. Può anche essere un naturale passo iniziale mentre un'organizzazione sta lavorando su un sistema di progettazione.

Shared Global CSS – Un approccio basato sulla condivisione totale

Con questo approccio, tutto il codice relativo allo styling viene archiviato in un repository centrale in cui i collaboratori scrivono CSS per tutti i micro-frontend lavorando su file CSS o utilizzando

preprocessori come Sass. Quando vengono apportate modifiche, un sistema di compilazione crea un singolo pacchetto CSS che può essere ospitato in un CDN e incluso in ogni microfrontend dalla shell dell'applicazione. Gli sviluppatori di microfrontend possono progettare e creare le proprie applicazioni eseguendo il codice tramite una shell dell'applicazione ospitata localmente.

Oltre all'ovvio vantaggio di ridurre il rischio di conflitti tra microfrontend, i vantaggi di questo approccio sono la coerenza e le prestazioni. Tuttavia, il disaccoppiamento degli stili dal markup e dalla logica rende più difficile per gli sviluppatori capire come vengono utilizzati gli stili, come possono evolversi e come possono essere obsoleti. Ad esempio, potrebbe essere più rapido introdurre un nuovo nome di classe piuttosto che conoscere la classe esistente e le conseguenze della modifica delle sue proprietà. Gli svantaggi della creazione di un nuovo nome di classe sono la crescita delle dimensioni del pacchetto, che influisce sulle prestazioni, e la potenziale introduzione di incongruenze nell'esperienza utente.

Sebbene un CSS globale condiviso possa essere il punto di partenza di una monolith-to-micro-frontends migrazione, raramente è utile per le architetture di micro-frontend che coinvolgono più di uno o due team che collaborano insieme. Consigliamo di investire in un sistema di progettazione il prima possibile e di implementare un approccio basato sulla condivisione di nulla durante lo sviluppo del sistema di progettazione.

Organizzazione e modalità di lavoro

Come tutte le strategie architettoniche, i microfrontend hanno implicazioni che vanno ben oltre la tecnologia che un'organizzazione sceglie di implementare. La decisione di creare applicazioni micro-frontend deve essere in linea con il business, il prodotto, l'organizzazione, le operazioni e persino la cultura (ad esempio, potenziare i team e decentralizzare il processo decisionale). In cambio, questo tipo di architettura micro-frontend supporta uno sviluppo veramente agile e basato sul prodotto, perché riduce significativamente il sovraccarico di comunicazione tra team altrimenti indipendenti.

Sviluppo agile

L'idea dello sviluppo agile del software è diventata così universale negli ultimi anni che praticamente tutte le organizzazioni affermano di lavorare in modo agile. Sebbene una definizione definitiva di agile non rientri nell'ambito di questa strategia, vale la pena esaminare gli elementi chiave rilevanti per lo sviluppo di microfrontend.

Il fondamento del paradigma agile è l'Agile Manifesto (2001), che postulava quattro principi principali (ad esempio, «Individui e interazioni rispetto a processi e strumenti») e dodici principi. I framework di processo come Scrum e Scaled Agile Framework (SAFe) sono emersi attorno all'Agile Manifesto e hanno trovato la loro strada nelle pratiche quotidiane. Tuttavia, la loro filosofia è in gran parte fraintesa o ignorata.

Nel contesto dell'architettura micro-frontend, è importante adottare i seguenti principi agili:

- «Fornite software funzionante frequentemente, da un paio di settimane a un paio di mesi, preferendo tempi più brevi».

  Questo principio sottolinea quanto sia importante lavorare in modo incrementale e fornire software alla produzione il più regolarmente e il più spesso possibile. Da un punto di vista tecnologico, ciò si riferisce all'integrazione continua e alla consegna continua (CI/CD). In CI/CD, gli strumenti e i processi per la creazione, il test e l'implementazione sono parti integranti di ogni progetto software. Il principio implica anche che l'infrastruttura di runtime e le responsabilità operative devono essere di proprietà del team. Tale proprietà è particolarmente importante nei sistemi distribuiti in cui sottosistemi indipendenti potrebbero avere requisiti significativamente diversi per l'infrastruttura e le operazioni.
- «Costruisci progetti attorno a persone motivate. Offri loro l'ambiente e il supporto di cui hanno bisogno e affidati a loro per portare a termine il lavoro».

«Le architetture, i requisiti e i progetti migliori emergono dai team che si organizzano autonomamente».

Entrambi questi principi enfatizzano i vantaggi della proprietà, dell'indipendenza e della responsabilità. end-to-end Un'architettura di micro-frontend avrà successo quando (e solo quando) i team possiedono veramente i propri microfrontend. La nd-to-end responsabilità, dall'ideazione alla progettazione e implementazione, fino alla consegna e al funzionamento, garantisce che il team possa effettivamente esercitarne la proprietà. Questa indipendenza è necessaria, sia dal punto di vista tecnico che organizzativo, affinché il team abbia autonomia sulla direzione strategica. Non è consigliabile utilizzare una piattaforma di micro-frontend in un'organizzazione centralizzata che utilizza un modello di sviluppo a cascata.

Composizione e dimensioni del team

Affinché un team addetto al software possa esercitare la propria responsabilità, deve autogovernarsi, comprese le modalità e i risultati del team, entro i limiti imposti dall'organizzazione.

Per essere efficaci, i team devono essere in grado di fornire software in modo indipendente e avere l'autorità di decidere il modo migliore per distribuirlo. Un team che riceve requisiti di funzionalità da un product manager esterno o progetti di interfaccia utente da un designer esterno, senza essere coinvolto nella pianificazione di questi elementi, non può essere considerato autonomo. Le funzionalità potrebbero violare i contratti o le funzionalità esistenti. Tali violazioni richiederanno ulteriori discussioni e trattative, con il rischio di ritardare la consegna e introdurre conflitti inutili tra i team.

Allo stesso tempo, i team non dovrebbero diventare troppo grandi. Mentre un team più numeroso dispone di più risorse e può far fronte alle assenze individuali, la complessità della comunicazione cresce esponenzialmente con ogni nuovo membro. Non è possibile stabilire una dimensione massima del team universalmente valida. Il numero di persone necessarie per un progetto dipende da fattori quali la maturità del team, la complessità tecnica, il ritmo dell'innovazione e l'infrastruttura. Ad esempio, Amazon segue la regola delle due pizze: una squadra troppo numerosa per essere nutrita con due pizze dovrebbe essere divisa in squadre più piccole. Questa può essere una sfida. La divisione dovrebbe avvenire lungo i confini naturali e dovrebbe dare a ciascun team autonomia e proprietà sul proprio lavoro.

DevOps cultura

DevOps si riferisce a una pratica di ingegneria del software in cui le fasi del ciclo di vita dello sviluppo sono strettamente integrate dal punto di vista organizzativo e tecnico. Contrariamente alla credenza popolare, DevOps riguarda molto la cultura e la mentalità e molto poco i ruoli e gli strumenti.

Tradizionalmente, un'organizzazione software avrebbe a disposizione team di specialisti, ad esempio per la progettazione, l'implementazione, il collaudo, l'implementazione e le operazioni. Ogni volta che un team completava il proprio lavoro, affidava il progetto al team successivo. Tuttavia, la fornitura del software attraverso silos di team specializzati causa attriti durante i passaggi di consegne. Allo stesso tempo, quando gli specialisti sono costretti a lavorare con un focus ristretto, non hanno conoscenze nei settori limitrofi e non hanno una visione sistemica del prodotto. Questi deficit possono portare a una scarsa coerenza del prodotto software.

Ad esempio, quando un architetto del software progetta una soluzione che deve essere implementata da qualcuno che fa parte di un team diverso, potrebbe trascurare aspetti intrinseci dell'implementazione (come una mancata corrispondenza delle dipendenze). Gli sviluppatori adottano quindi delle scorciatoie (come una monkey patch) oppure ne back-and-forth viene avviata una formalizzata tra l'architetto e il team di sviluppo. A causa del sovraccarico di gestione di questi processi, lo sviluppo non è più agile (nel senso di flessibile, adattivo, incrementale e informale).

Sebbene il termine DevOps si riferisca principalmente alla cultura, implica le tecnologie e i processi che lo rendono possibile nella pratica. DevOps DevOps è strettamente correlato alla CI/CD. Quando uno sviluppatore ha finito di implementare un incremento di software, lo invia a un sistema di controllo delle versioni come Git. Tradizionalmente, un sistema di compilazione creava e integrava il software, lo testava e lo rilasciava in un processo più o meno unificato e centralizzato. Con CI/CD, la creazione, l'integrazione, il test e il rilascio del software sono intrinseci e automatizzati. Idealmente, il processo fa parte del progetto software stesso attraverso file di configurazione adattati specificamente al progetto specifico.

Il maggior numero possibile di passaggi viene automatizzato. Ad esempio, le pratiche di test manuali dovrebbero essere ridotte, poiché quasi tutti i tipi di test possono essere automatizzati. Quando il progetto è configurato in questo modo, gli aggiornamenti di un prodotto software possono essere forniti più volte al giorno con la massima sicurezza. Un'altra tecnologia supportata DevOps è l'infrastruttura come codice (IaC).

Tradizionalmente, la configurazione e la manutenzione dell'infrastruttura IT richiedevano il lavoro manuale di installazione e manutenzione dell'hardware (installazione di cavi e server

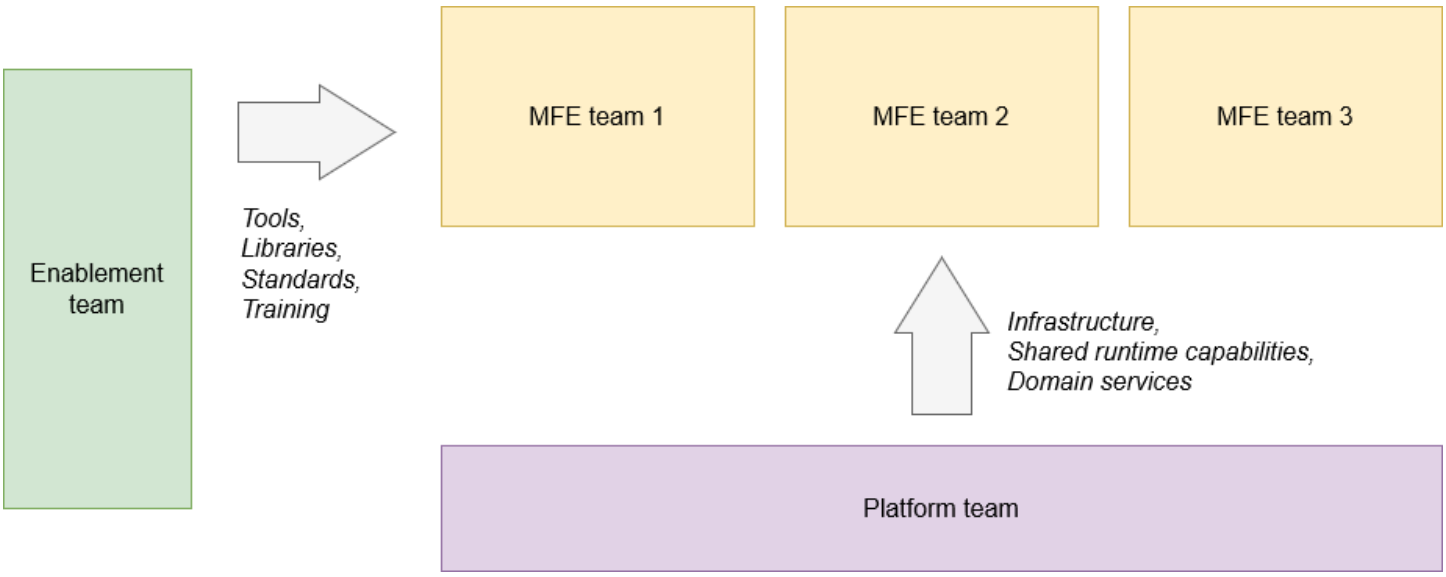
in un data center) e del software operativo. Ciò era necessario, ma presentava numerosi inconvenienti. L'installazione richiedeva molto tempo ed era soggetta a errori. L'hardware era spesso sovradimensionato o insufficiente, con conseguenti spese eccessive o un peggioramento delle prestazioni. Utilizzando IaC, è possibile descrivere i requisiti di infrastruttura per un sistema IT tramite un file di configurazione da cui è possibile distribuire e aggiornare automaticamente i servizi cloud.

Cosa c'entra tutto questo con i micro-frontend? DevOps, CI/CD e IaC sono i complementi ideali di un'architettura di micro-frontend. I vantaggi dei microfrontend si basano su processi di distribuzione rapidi e senza attriti. Una DevOps cultura può prosperare solo in ambienti in cui i team gestiscono progetti software con responsabilità. end-to-end

Orchestrazione dello sviluppo di microfrontend tra più team

Quando si ridimensiona lo sviluppo di microfrontend tra più team interfunzionali, emergono due problemi: in primo luogo, i team iniziano a sviluppare la propria interpretazione del paradigma, a fare scelte di framework e librerie e a creare le proprie librerie di strumenti e di supporto. In secondo luogo, i team completamente autonomi devono assumersi la responsabilità di funzionalità generiche come la gestione dell'infrastruttura a basso livello. Pertanto, è opportuno introdurre due team aggiuntivi in un'organizzazione di micro-frontend con più team: un team di abilitazione e un team di piattaforma. Questi concetti sono ampiamente adottati nelle moderne organizzazioni IT con sistemi distribuiti e sono ben documentati in Team Topologies.

Il diagramma seguente mostra il team di abilitazione che fornisce strumenti, librerie, standard e test a tre team di microfrontend. Il team della piattaforma fornisce infrastruttura, funzionalità di runtime condivise e servizi di dominio agli stessi tre team di microfrontend.



Il team della piattaforma supporta i team di microfrontend liberandoli dal sollevamento indifferenziato di carichi pesanti. Questo supporto può includere servizi di infrastruttura come runtime dei container, pipeline CI/CD, strumenti di collaborazione e monitoraggio. Tuttavia, la creazione di un team di piattaforma non dovrebbe portare a un'organizzazione in cui lo sviluppo è separato dalle operazioni. È vero il contrario: il team della piattaforma offre un prodotto ingegneristico e i team di microfrontend hanno la proprietà e la responsabilità di esecuzione dei propri servizi sulla piattaforma.

Il team di abilitazione fornisce supporto concentrandosi sulla governance e garantendo la coerenza tra i team di microfrontend. (Il team della piattaforma non dovrebbe essere coinvolto in questo.) Il team di abilitazione gestisce risorse condivise, come una libreria di interfaccia utente, e crea standard come la scelta del framework, i budget prestazionali e le convenzioni di interoperabilità. Allo stesso tempo, fornisce ai nuovi team o membri del team una formazione sull'applicazione degli standard e degli strumenti definiti dalla governance.

Implementazione

La stella polare dell'autonomia nei team di microfrontend è disporre di una pipeline automatizzata con un percorso di produzione indipendente dagli altri team di microfrontend. I team che seguono il principio «share-nothing» possono implementare pipeline indipendenti. I team che condividono librerie o dipendono da una piattaforma devono decidere come gestire le dipendenze nelle pipeline di distribuzione.

In genere, ogni pipeline esegue le seguenti operazioni:

- Crea risorse frontend
- Implementa le risorse nell'hosting per il consumo
- Assicura che i registri e le cache vengano aggiornati in modo che le nuove versioni possano essere consegnate ai clienti

Le fasi effettive della pipeline variano a seconda dello stack tecnologico e dell'approccio alla composizione della pagina.

Per quanto riguarda la composizione lato client, ciò significa caricare i pacchetti di applicazioni in bucket di hosting e rilasciarli all'uso tramite la memorizzazione nella cache di un CDN. Le applicazioni che utilizzano la memorizzazione nella cache del browser con gli addetti all'assistenza dovrebbero inoltre implementare metodi per aggiornare le cache degli addetti all'assistenza.

Per quanto riguarda la composizione lato server, ciò significa in genere distribuire la nuova versione del componente server e aggiornare il registro del microfrontend per rendere la nuova versione individuabile. È possibile utilizzare modelli di distribuzione blu/verde o canarino per implementare gradualmente nuove versioni.

Governance

In genere più persone lavorano su microfrontend e ognuna lavora con vincoli diversi verso obiettivi aziendali comuni. Sebbene la comunicazione e la collaborazione tra le persone siano fondamentali per il successo, l'eccessiva comunicazione e l'implementazione di processi eccessivamente complicati rallentano il ciclo di sviluppo. Ciò si traduce in una diminuzione del morale e abbassa il livello di qualità.

Le aziende di maggior successo che implementano microfrontend utilizzando più team creano meccanismi per bilanciare l'autonomia con l'allineamento. Consentono ai decisori di agire a livello locale e di agire gerarchicamente solo quando necessario. I meccanismi includono quanto segue:

- contratti API
- Interattività incrociata tramite eventi
- Bilanciamento dell'autonomia con l'allineamento
- Caratteristica delle bandiere
- Individuazione dei servizi

Contratti API

Ogni micro-frontend è un sistema in grado di incapsulare opinioni, logica e complessità. Le preoccupazioni trasversali di solito includono quanto segue:

- Sistemi di progettazione – Strumenti per sviluppare interfacce utente distribuite come librerie
- Composizione – Il modo in cui un micro-frontend deve interagire con la shell di un'applicazione per renderizzarlo ed ereditarne il contesto
- Gestione logica – Interazione con le API per gestire lo stato persistente
- Interattività con altri micro-frontend – Scenari come la pubblicazione e il consumo di eventi o la navigazione da un micro-frontend all'altro

Per accelerare il consumo e la risoluzione dei problemi, è normale investire nella standardizzazione del modo in cui queste interfacce vengono dichiarate e documentate, comprese le dipendenze dei microfrontend. I wiki curati da esseri umani sono un buon inizio. Un approccio più scalabile consiste nell'archiviare queste informazioni come metadati strutturati nel codice. È quindi

possibile centralizzarle per il consumo utilizzando l'automazione per tenere traccia delle modifiche cronologiche e fornire una ricerca completa.

Quando i micro-frontend coinvolgono un gran numero di team, è necessaria una strategia di coordinamento tra i team. La condivisione dei contratti API in modo unificato diventa un must perché riduce il sovraccarico di comunicazione e migliora l'esperienza degli sviluppatori.

OpenAPI è un linguaggio di specifica per le API HTTP che supporta la definizione di interfacce e contratti API in modo unificato. Puoi implementare le API REST utilizzando OpenAPI in Amazon API Gateway. Puoi anche utilizzare un'ampia varietà di framework open source che puoi ospitare in contenitori o macchine virtuali. Un vantaggio significativo è che OpenAPI può generare automaticamente la documentazione in un formato coerente, in modo che più team possano condividere le conoscenze con un investimento iniziale minimo.

Quando più team lavorano su micro-frontend, spesso formano gruppi. In questi gruppi, le persone possono incontrarsi e imparare gli uni dagli altri mentre pensano e contribuiscono al quadro generale. Queste iniziative in genere definiscono e documentano i limiti di proprietà, discutono questioni trasversali e identificano tempestivamente qualsiasi duplicazione degli sforzi per risolvere problemi comuni.

Interattività incrociata mediante eventi

In alcuni scenari, potrebbe essere necessario interagire tra loro più microfrontend per reagire ai cambiamenti di stato o alle azioni degli utenti. Ad esempio, più micro-frontend sulla pagina possono includere menu comprimibili. Quando l'utente sceglie un pulsante, viene visualizzato un menu. Il menu è nascosto quando l'utente fa clic in un altro punto, incluso un altro menu che viene visualizzato all'interno di un microfrontend diverso.

Tecnicamente, una libreria di stato condivisa come Redux può essere utilizzata da più micro-frontend e coordinata da una shell. Tuttavia, ciò crea un accoppiamento significativo tra le applicazioni, con conseguente codice più difficile da testare e potrebbe rallentare le prestazioni durante il rendering.

Un approccio comune ed efficace consiste nello sviluppo di un bus di eventi distribuito come libreria, orchestrato dalla shell dell'applicazione e utilizzato da più microfrontend. In questo modo, ogni microfrontend pubblica e ascolta eventi particolari in modo asincrono, basando il proprio comportamento solo sul proprio stato interno. Quindi, più team possono gestire una pagina wiki condivisa che descrive gli eventi e documenta i comportamenti concordati dai progettisti dell'esperienza utente.

In un'implementazione dell'esempio event-bus, un componente dropdown utilizza il bus condiviso per pubblicare un evento chiamato `drop-down-open-menu` con un payload di. `{"id": "homepage-aboutus-button"}` Il componente aggiunge un listener all'evento per garantire che, se viene generato un `drop-down-open-menu` evento per un nuovo ID, il componente a discesa venga renderizzato in modo da nascondere la sua sezione comprimibile. In questo modo, il micro-frontend può reagire alle modifiche in modo asincrono con un aumento delle prestazioni e un migliore incapsulamento, facilitando la progettazione e il test dei comportamenti da parte di più team.

Consigliamo di utilizzare API standard implementate nativamente dai browser moderni per migliorare la semplicità e la manutenibilità. Il riferimento agli eventi MDN fornisce informazioni sull'utilizzo degli eventi con applicazioni renderizzate lato client.

Bilanciamento dell'autonomia con l'allineamento

Le architetture micro-frontend sono fortemente orientate all'autonomia del team. Tuttavia, è importante distinguere tra aree che possono supportare flessibilità e approcci diversi per risolvere i problemi e aree in cui la standardizzazione è necessaria per raggiungere l'allineamento. I dirigenti e gli architetti senior devono identificare tempestivamente queste aree e dare priorità agli investimenti per bilanciare sicurezza, prestazioni, eccellenza operativa e affidabilità dei microfrontend. Il raggiungimento di questo equilibrio implica quanto segue: creazione, test, rilascio e registrazione di microfrontend, monitoraggio e invio di avvisi.

Creazione di micro-frontend

Idealmente, tutti i team sono fortemente allineati per massimizzare i vantaggi in termini di prestazioni per l'utente finale. In pratica, questo può essere difficile e potrebbe richiedere uno sforzo maggiore. Ti consigliamo di iniziare con alcune linee guida scritte a cui più team possono contribuire attraverso un dibattito aperto e trasparente. I team possono quindi adottare gradualmente il modello software Cookiecutter, che supporta la creazione di strumenti che forniscono un modo unificato per strutturare un progetto.

Utilizzando questo approccio, è possibile incorporare opinioni e vincoli. Il lato negativo è che questi strumenti richiedono investimenti significativi per la creazione e la manutenzione e per garantire che i blocchi vengano risolti rapidamente senza influire sulla produttività degli sviluppatori.

nd-to-end Test elettronici per microfrontend

I test unitari possono essere lasciati ai proprietari. Consigliamo di implementare fin dall'inizio una strategia per testare in modo incrociato i microfrontend eseguiti su una shell unica. La strategia

include la capacità di testare le applicazioni prima e dopo un rilascio di produzione. Consigliamo di sviluppare processi e documentazione per personale tecnico e non tecnico per testare manualmente le funzionalità critiche.

È importante assicurarsi che le modifiche non compromettano l'esperienza del cliente funzionale o non funzionale. Una strategia ideale consiste nell'investire gradualmente in test automatizzati, sia per le funzionalità chiave che per le caratteristiche dell'architettura come sicurezza e prestazioni.

Rilascio di micro-frontend

Ogni team potrebbe avere il proprio modo di distribuire il proprio codice, elaborare opinioni e la propria infrastruttura. Il costo della complessità legato alla manutenzione di tali sistemi è in genere un deterrente. Consigliamo invece di investire tempestivamente per implementare una strategia condivisa che possa essere applicata mediante strumenti condivisi.

Sviluppa modelli con la piattaforma CI/CD preferita. I team possono quindi utilizzare i modelli preapprovati e l'infrastruttura condivisa per apportare modifiche alla produzione. È possibile iniziare a investire in questo lavoro di sviluppo sin dall'inizio, perché questi sistemi raramente necessitano di aggiornamenti significativi dopo un periodo iniziale di test e consolidamento.

Registrazione di log e monitoraggio

Ogni team può avere metriche aziendali e di sistema diverse che desidera monitorare per scopi operativi o di analisi. Il modello software Cookiecutter può essere applicato anche qui. L'erogazione degli eventi può essere riassunta e resa disponibile come libreria utilizzabile da più micro-frontend. Per bilanciare flessibilità e autonomia, sviluppa strumenti per la registrazione di metriche personalizzate e la creazione di dashboard o report personalizzati. Il reporting promuove una stretta collaborazione con i proprietari dei prodotti e riduce il ciclo di feedback dei clienti finali.

Standardizzando la distribuzione, più team possono collaborare per tenere traccia delle metriche. Ad esempio, un sito di e-commerce può tracciare il percorso dell'utente dal microfrontend «Dettagli del prodotto» al microfrontend «Cart», al microfrontend «Purchase» per misurare il coinvolgimento, il tasso di abbandono e i problemi. Se ogni microfrontend registra gli eventi utilizzando un'unica libreria, è possibile utilizzare questi dati nel loro insieme, esplorarli in modo olistico e identificare tendenze approfondite.

Avviso

Analogamente alla registrazione e al monitoraggio, gli avvisi traggono vantaggio dalla standardizzazione con spazio per un certo grado di flessibilità. Team diversi potrebbero reagire

in modo diverso agli avvisi funzionali e non funzionali. Tuttavia, se tutti i team dispongono di un metodo consolidato per avviare avvisi basato su metriche raccolte e analizzate su una piattaforma condivisa, l'azienda può identificare i problemi tra i team. Questa funzionalità è utile durante gli eventi di gestione degli incidenti. Ad esempio, gli avvisi possono essere avviati nel modo seguente:

- Numero elevato di eccezioni JavaScript lato client in una particolare versione del browser
- Tempo di rendering notevolmente ridotto oltre una determinata soglia
- Numero elevato di codici di stato 5xx quando si utilizza una particolare API

A seconda della maturità del sistema, è possibile bilanciare gli sforzi su diverse parti dell'infrastruttura, come illustrato nella tabella seguente.

Adozione	Ricerca e sviluppo	Salita	Maturità
Crea micro-fro ntend.	Sperimenta, documenta e condividi quanto appreso.	Investi in strumenti per costruire nuovi micro-frontend. Evangelizza l'adozione.	Consolida gli utensili per ponteggi. Spingi per l'adozione.
Prova i micro-fro ntend dall'iniz io alla fine.	Implementa meccanism i per testare manualmente tutti i micro-fro ntend correlati.	Investi in strumenti per test automatizzati di sicurezza e prestazioni. Esamina i segnali di funzionalità e l'individuazione dei servizi.	Consolida gli strumenti per l'individ uazione dei servizi, i test in produzion e e i test automatizzati. end-to-end
Rilascia i micro-fro ntend.	Investite in un'infrastruttura CI/CD condivisa e in versioni automatizzate in più ambienti. Evangelizza l'adozione.	Consolida gli strumenti per l'infrastruttura CI/CD Implementa meccanismi di rollback manuali. Spingi per l'adozione.	Crea meccanismi per avviare rollback automatici in aggiunta alle metriche e agli avvisi di sistema e aziendali.

Adozione	Ricerca e sviluppo	Salita	Maturità
Osserva le prestazic ni dei microfror tend.	Investite in un'infrastruttura e una libreria di monitoraggio condivise per la registrazione coerente degli eventi di sistema e aziendali.	Consolida gli strumenti per il monitoraggio e gli avvisi. Implementa dashboard interteam per monitorare lo stato generale e migliorare la gestione degli incidenti.	Standardizza gli schemi di registraz ione. Ottimizza i costi. Implementa gli avvisi basati su metriche aziendali complesse.

Caratteristica: bandiere

I flag di funzionalità possono essere implementati nei microfrontend per facilitare il coordinamento del test e del rilascio delle funzionalità in più ambienti. La tecnica del feature flag consiste nel centralizzare le decisioni in un negozio basato su valori booleani e nel guidare il comportamento in base a ciò. Viene spesso utilizzata per diffondere silenziosamente modifiche che possono essere tenute nascoste fino a un momento specifico, sbloccando al contempo nuove versioni per nuove funzionalità che altrimenti sarebbero bloccate, riducendo la velocità del team.

Prendiamo ad esempio i team che lavorano su una funzionalità di micro-frontend che verrà lanciata in una data specifica. La funzionalità è pronta, ma deve essere rilasciata insieme a una modifica su un altro micro-frontend rilasciato in modo indipendente. Il blocco del rilascio di entrambi i microfrontend sarebbe considerato un anti-pattern e aumenterebbe il rischio una volta implementato.

I team possono invece creare un feature flag booleano in un database che entrambi utilizzano durante il rendering (magari tramite una chiamata HTTP a un'API Feature Flags condivisa). I team possono anche rilasciare la modifica in un ambiente di test in cui il valore booleano è impostato per verificare i requisiti funzionali e non funzionali tra progetti prima di passare `True` alla produzione.

Un altro esempio di utilizzo dei flag di funzionalità è l'implementazione di un meccanismo per sovrascrivere il valore di un flag impostando un valore specifico tramite il `QueryString` parametro o memorizzando una particolare stringa di test in un cookie. I proprietari dei prodotti possono iterare sulle funzionalità senza bloccare il rilascio di altre funzionalità o correggere errori fino alla data di lancio. Alla data specificata, la modifica del valore del flag sul database rende immediatamente

visibile la modifica in produzione, senza la necessità di rilasci coordinati tra i team. Dopo il rilascio di una funzionalità, i team di sviluppo puliscono il codice per rimuovere il vecchio comportamento.

Altri casi d'uso includono il rilascio di un sistema di feature flag basato sul contesto. Ad esempio, se un singolo sito Web serve i clienti in più lingue, una funzionalità potrebbe essere disponibile solo per i visitatori di un determinato paese. Il sistema di segnalazione delle funzionalità può dipendere dal fatto che il consumatore invii il contesto del paese (ad esempio utilizzando l'intestazione `Accept-Language` HTTP) e il comportamento può variare a seconda del contesto.

Sebbene i feature flag siano uno strumento potente per facilitare la collaborazione tra sviluppatori e proprietari di prodotti, si affidano alla diligenza delle persone per evitare un significativo degrado del codice base. Mantenere attivi i flag su più funzionalità può aumentare la complessità nella risoluzione dei problemi, aumentare la dimensione dei JavaScript pacchetti e, in ultima analisi, accumulare debiti tecnici. Le attività di mitigazione comuni includono quanto segue:

- Ogni funzionalità viene sottoposta a test unitari mediante un flag per ridurre la probabilità di bug, il che può introdurre cicli di feedback più lunghi nelle pipeline CI/CD automatizzate che eseguono i test
- Creazione di strumenti per misurare gli aumenti delle dimensioni dei pacchetti durante le modifiche al codice, che possono essere mitigati durante le revisioni del codice

AWS offre una gamma di soluzioni per ottimizzare i test A/B sull'edge utilizzando le CloudFront funzioni di Amazon o Lambda @Edge. Questi approcci aiutano a ridurre la complessità dell'integrazione di una soluzione o del prodotto SaaS esistente che stai utilizzando per affermare le tue ipotesi. Per ulteriori informazioni, consulta A/B testing.

Individuazione dei servizi

Il modello di scoperta del frontend migliora l'esperienza di sviluppo durante lo sviluppo, il test e la fornitura di microfrontend. Il pattern utilizza una configurazione condivisibile che descrive il punto di ingresso dei micro-frontend. La configurazione condivisibile include anche metadati aggiuntivi che vengono utilizzati per implementazioni sicure in ogni ambiente utilizzando versioni canarie.

Lo sviluppo frontend moderno prevede l'utilizzo di un'ampia varietà di strumenti e librerie per supportare la modularità durante lo sviluppo. Tradizionalmente, questo processo consisteva nel raggruppare il codice in singoli file che potevano essere ospitati in un CDN con l'obiettivo di ridurre al minimo le chiamate di rete durante l'esecuzione, incluso il caricamento iniziale (quando un'app si apre

in un browser) e l'utilizzo (quando un cliente esegue azioni come la scelta di pulsanti o l'inserimento di informazioni).

Suddivisione dei pacchetti

Le architetture Micro-frontend risolvono i problemi di prestazioni causati da pacchetti molto grandi generati raggruppando singolarmente un ampio set di funzionalità. Ad esempio, un sito di e-commerce di grandi dimensioni può essere raggruppato in un file da 6 MB. JavaScript Nonostante la compressione, la dimensione del file potrebbe influire negativamente sull'esperienza dell'utente durante il caricamento dell'app e il download del file da un CDN ottimizzato per i dispositivi periferici.

Se dividi l'app in home page, dettagli del prodotto e micro-frontend del carrello, puoi utilizzare un meccanismo di raggruppamento per creare tre pacchetti singoli da 2 MB. Questa modifica potrebbe migliorare le prestazioni al primo caricamento del 300% quando gli utenti utilizzano la home page. I pacchetti di micro-frontend relativi al prodotto o al carrello vengono caricati in modo asincrono solo se e quando l'utente visita la pagina del prodotto di un articolo e decide di acquistarlo.

Sono disponibili molti framework e librerie basati su questo approccio, con vantaggi sia per i clienti che per gli sviluppatori. Per identificare i limiti aziendali che possono comportare il disaccoppiamento delle dipendenze nel codice, è possibile mappare diverse funzioni aziendali a più team. La proprietà distribuita introduce indipendenza e agilità.

Quando si dividono i pacchetti di build, è possibile utilizzare una configurazione per mappare i micro-frontend e gestire l'orchestrazione per il caricamento iniziale e per la navigazione dopo il caricamento. Quindi, la configurazione può essere utilizzata durante il runtime anziché durante la fase di compilazione. Ad esempio, il codice frontend lato client o il codice backend lato server possono effettuare una chiamata di rete iniziale a un'API per recuperare dinamicamente l'elenco dei micro-frontend. Recupera anche i metadati necessari per la composizione e l'integrazione. È possibile configurare strategie di failover e caching per garantire affidabilità e prestazioni. La mappatura dei microfrontend aiuta a rendere le implementazioni individuali di microfrontend individuabili dai microfrontend precedentemente distribuiti e orchestrati da un'app shell.

Versioni di Canary

Una versione canaria è un modello consolidato e popolare per l'implementazione di microservizi. Le release di Canary raggruppano gli utenti destinatari di una versione in più gruppi e rilasciano le modifiche gradualmente anziché sostituirle immediatamente (nota anche come distribuzione blu/verde). Un esempio di strategia di rilascio di Canary consiste nell'applicare una nuova modifica al

10% degli utenti target e aggiungerne il 10% ogni minuto, con una durata totale di 10 minuti per raggiungere il 100%.

L'obiettivo di una release Canary è ottenere un feedback tempestivo sulle modifiche, monitorando il sistema per ridurre l'impatto di eventuali problemi. Quando l'automazione è in atto, le metriche aziendali o di sistema possono essere monitorate da un sistema interno che può interrompere l'implementazione o avviare un rollback.

Ad esempio, una modifica potrebbe introdurre un bug che, nei primi due minuti di una versione, comporta una perdita di ricavi o un peggioramento delle prestazioni. Il monitoraggio automatico può avviare un allarme. Grazie al Service Discovery Pattern, l'allarme può interrompere l'implementazione e ripristinarlo immediatamente, interessando solo il 20 percento degli utenti anziché il 100 percento. L'azienda trae vantaggio dalla portata ridotta del problema.

Per un esempio di architettura che utilizza DynamoDB come storage per implementare un'API di amministrazione REST, consulta la soluzione Frontend Service Discovery on AWS su. GitHub Utilizza il AWS CloudFormation modello per integrare l'architettura nelle tue pipeline CI/CD. La soluzione include un'API REST Consumer per l'integrazione della soluzione con le applicazioni frontend.

Hai bisogno di un team di piattaforma?

Alcune aziende hanno un team responsabile della proprietà e della manutenzione del codice, dell'infrastruttura e dei processi che vengono adottati da altri team per lavorare sui microfrontend. Le responsabilità comuni includono:

- Crea e gestisci una pipeline CI/CD che può essere utilizzata con repository contenenti micro-frontend. Crea e testa le modifiche al codice e rilasciale in più ambienti.
- Crea e gestisci strumenti relativi all'osservabilità come dashboard condivisi, meccanismi di avviso e sistemi per reagire ai problemi.
- Crea e gestisci librerie condivise per la gestione degli eventi, il consumo di servizi condivisi e le dipendenze di terze parti.
- Crea e gestisci strumenti che monitorino continuamente qualità non funzionali come prestazioni, sicurezza e affidabilità del sistema.
- Crea e gestisci sistemi di progettazione.
- Crea, gestisci e supporta la shell dell'applicazione per il sistema micro-frontend.

A seconda delle dimensioni del progetto, è possibile gestire queste responsabilità utilizzando uno dei seguenti approcci:

- Crea un team di piattaforma dedicato la cui unica responsabilità è lavorare su strumenti condivisi.
- Crea un gruppo composto da membri di più team. I membri del gruppo dividono il loro tempo tra il lavoro su micro-frontend e il lavoro su strumenti condivisi. Questa squadra è nota anche come «tiger team».

Sebbene l'approccio tiger-team sia un modo efficace per rimanere concentrati sul cliente, un tiger team spesso si trasforma in un team di piattaforma se il progetto acquisisce visibilità e responsabilità. Sia per i team di piattaforma che per i tiger team, le aziende di maggior successo che lavorano sui microfrontend formano questi team in modo che più persone con background e competenze diversi possano contribuire. I membri del team possono includere ingegneri di backend, ingegneri di frontend, progettisti di esperienza utente (UX) e responsabili tecnici dei prodotti. Questa diversità spinge le persone a impegnarsi continuamente in dibattiti salutari e a progettare pensando alla semplicità.

Passaggi successivi

Questa guida ha trattato i modelli architettonici e organizzativi, i compromessi per le decisioni chiave e i problemi di governance relativi ai microfrontend. Le tabelle riassumono i compromessi delle pratiche discusse in questo documento in termini delle seguenti dimensioni:

- Autonomia – La capacità di ogni team di microfrontend di evolvere in modo indipendente la propria implementazione e il rilascio agli utenti finali.
- Coerenza – L'esperienza complessiva dell'applicazione in cui ogni micro-frontend si comporta come previsto. Un'elevata coerenza significa che i microfrontend sono coerenti con il resto dell'applicazione e non danneggiano l'esperienza utente dell'intera applicazione.
- Complessità – La quantità di infrastruttura, codice e impegno necessari per implementare e testare i microfrontend, l'applicazione complessiva e i controlli di governance.

Pratica	Autonomia	Coerenza	Complessità
Creazion con micro-fro ntend anziché applicazi oni monolitic he	Elevata	Media	Elevata

Pratiche di condivisione del codice	Autonomia	Coerenza	Complessità
Non condividere nulla	Elevata	Bassa	Bassa
Condividete preoccupazioni trasversali	Media	Elevata	Media
Condividete la logica aziendale	Bassa	Elevata	Media
Condividi tramite le librerie in fase di compilazione	Media	Elevata	Bassa
Condividi in fase di	Elevata	Elevata	Elevata

Pratiche di condivisione del codice	Autonomia	Coerenza	Complessità
esecuzione			

Pratiche di individuazione tramite microfrontend	Autonomia	Coerenza	Complessità
Configurazione durante la creazione dell'applicazione	Bassa	Elevata	Bassa
Individuazione lato server	Elevata	Elevata	Media
Individuazione lato client (runtime)	Elevata	Elevata	Media

Visualizza le pratiche di composizione	Autonomia	Coerenza	Complessità
Composizione lato server	Elevata	Media	Elevata
Composizione laterale	Media	Media	Elevata
Composizione lato client	Elevata	Media	Media

Per ulteriori informazioni sui concetti introdotti in questa guida, consulta la sezione Risorse.

Risorse

- Micro-frontend nel contesto
- Progettazione basata sul dominio
- Immagini EDA
- Scoperta del frontend
- Frontend Service Discovery attivo AWS
- Manifesto Agile
- Riferimento all'evento MDN
- OpenAPI

Collaboratori

Le seguenti persone hanno contribuito a questa guida.

- Matteo Figus, Principal Solutions Architect, AWS
- Alexander Guensche, Senior Solutions Architect, AWS
- Harun Hasdal, architetto senior delle soluzioni, AWS
- Luca Mezzalira, responsabile di Go to Market Specialist Solutions Architect Serverless UK, AWS

Cronologia dei documenti

La tabella seguente descrive le modifiche significative apportate a questa guida. Per ricevere notifiche sugli aggiornamenti futuri, puoi abbonarti a un feed RSS.

Modifica	Descrizione	Data
Pubblicazione iniziale	—	12 luglio 2024

AWS Glossario del Prontuario

I seguenti termini sono comunemente utilizzati in strategie, guide e pattern forniti dal AWS Prontuario. Per suggerire voci, utilizza il link Fornisci feedback alla fine del glossario.

Numeri

7 R

Sette strategie di migrazione comuni per trasferire le applicazioni sul cloud. Queste strategie si basano sulle 5 R identificate da Gartner nel 2011 e sono le seguenti:

- Rifattorizzare/riprogettare: trasferisci un'applicazione e modifica la sua architettura sfruttando appieno le funzionalità native del cloud per migliorare l'agilità, le prestazioni e la scalabilità. Ciò comporta in genere la portabilità del sistema operativo e del database. Esempio: esegui la migrazione del database Oracle on-premise ad Amazon SQL Aurora edizione compatibile con Postgre.
- Ridefinire la piattaforma (lift and reshape): trasferisci un'applicazione nel cloud e introduci un certo livello di ottimizzazione per sfruttare le funzionalità del cloud. Esempio: esegui la migrazione del database Oracle on-premise ad Amazon Relational Database Service (AmazonRDS) per Oracle nel. Cloud AWS
- Riacquistare (drop and shop): passa a un prodotto diverso, in genere effettuando la transizione da una licenza tradizionale a un modello SaaS. Esempio: esegui la migrazione del tuo sistema di gestione delle relazioni con i clienti (CRM) su Salesforce.com.
- Eseguire il rehosting (lift and shift): trasferisci un'applicazione sul cloud senza apportare modifiche per sfruttare le funzionalità del cloud. Esempio: esegui la migrazione del database Oracle on-premise su Oracle su un'EC2istanza di. Cloud AWS
- Trasferire (eseguire il rehosting a livello hypervisor): trasferisci l'infrastruttura sul cloud senza acquistare nuovo hardware, riscrivere le applicazioni o modificare le operazioni esistenti. Si esegue la migrazione dei server da una piattaforma locale a un servizio cloud per la stessa piattaforma. Esempio: migrazione di Microsoft Hyper-V applicazione a. AWS
- Riesaminare (mantenere): mantieni le applicazioni nell'ambiente di origine. Queste potrebbero includere applicazioni che richiedono una rifattorizzazione significativa che desideri rimandare a un momento successivo e applicazioni legacy che desideri mantenere, perché non vi è alcuna giustificazione aziendale per effettuarne la migrazione.

- Ritirare: disattiva o rimuovi le applicazioni che non sono più necessarie nell'ambiente di origine.

A

ABAC

Vedi Controllo degli accessi basato su attributi.

servizi astrattivi

Vedi servizi gestiti.

ACID

Vedi atomicità, consistenza, isolamento, durabilità.

migrazione attiva-attiva

Un metodo di migrazione del database in cui i database di origine e di destinazione vengono mantenuti sincronizzati (utilizzando uno strumento di replica bidirezionale o operazioni di doppia scrittura) ed entrambi i database gestiscono le transazioni provenienti dalle applicazioni di connessione durante la migrazione. Questo metodo supporta la migrazione in piccoli batch controllati anziché richiedere una conversione una tantum. È più flessibile ma richiede più lavoro rispetto alla migrazione attiva-passiva.

migrazione attiva-passiva

Un metodo di migrazione di database in cui i database di origine e di destinazione vengono mantenuti sincronizzati, ma solo il database di origine gestisce le transazioni provenienti dalle applicazioni di connessione mentre i dati vengono replicati nel database di destinazione. Il database di destinazione non accetta alcuna transazione durante la migrazione.

funzione di aggregazione

Una SQL funzione che opera su un gruppo di righe e calcola un singolo valore restituito per il gruppo. Esempi di funzioni aggregate includono `SUM` e. `MAX`

Intelligenza artificiale

Vedi intelligenza artificiale.

AIOps

Guarda le operazioni di intelligenza artificiale.

anonimizzazione

Il processo di eliminazione permanente delle informazioni personali in un set di dati. L'anonimizzazione può aiutare a proteggere la privacy personale. I dati anonimi non sono più considerati dati personali.

anti-modello

Una soluzione utilizzata di frequente per un problema ricorrente in cui la soluzione è controproducente, inefficace o meno efficace di un'alternativa.

controllo delle applicazioni

Un approccio alla sicurezza che consente l'uso solo di applicazioni approvate per proteggere un sistema dal malware.

portfolio di applicazioni

Una raccolta di informazioni dettagliate su ogni applicazione utilizzata da un'organizzazione, compresi i costi di creazione e manutenzione dell'applicazione e il relativo valore aziendale. Queste informazioni sono fondamentali per il processo di scoperta e analisi del portfolio e aiutano a identificare e ad assegnare la priorità alle applicazioni da migrare, modernizzare e ottimizzare.

intelligenza artificiale (IA)

Il campo dell'informatica dedicato all'uso delle tecnologie informatiche per svolgere funzioni cognitive tipicamente associate agli esseri umani, come l'apprendimento, la risoluzione di problemi e il riconoscimento di schemi. Per ulteriori informazioni, consulta la sezione Che cos'è l'intelligenza artificiale?

operazioni di intelligenza artificiale (AIOps)

Il processo di utilizzo delle tecniche di machine learning per risolvere problemi operativi, ridurre gli incidenti operativi e l'intervento umano e aumentare la qualità del servizio. Per ulteriori informazioni su come AIOps viene utilizzato nella strategia di AWS migrazione, consulta la guida all'integrazione delle operazioni.

crittografia asimmetrica

Un algoritmo di crittografia che utilizza una coppia di chiavi, una chiave pubblica per la crittografia e una chiave privata per la decrittografia. Puoi condividere la chiave pubblica perché non viene utilizzata per la decrittografia, ma l'accesso alla chiave privata deve essere altamente limitato.

atomicità, consistenza, isolamento, durabilità () ACID

Un insieme di proprietà del software che garantiscono la validità dei dati e l'affidabilità operativa di un database, anche in caso di errori, interruzioni di corrente o altri problemi.

controllo dell'accesso basato su attributi () ABAC

La pratica di creare autorizzazioni dettagliate basate su attributi utente, come reparto, ruolo professionale e nome del team. Per ulteriori informazioni, vedere ABACfor AWS nella documentazione AWS Identity and Access Management ()IAM.

fonte di dati autorevole

Una posizione in cui è archiviata la versione principale dei dati, considerata la fonte di informazioni più affidabile. È possibile copiare i dati dalla fonte di dati autorevole in altre posizioni allo scopo di elaborarli o modificarli, ad esempio anonimizzandoli, oscurandoli o pseudonimizzandoli.

Zona di disponibilità

Posizione separata all'interno di una isolata dagli errori Regione AWS che si verificano in altre zone di disponibilità che offre connettività di rete non costosa e a bassa latenza ad altre zone di disponibilità nella stessa regione.

AWS Framework per l'adozione del cloud (AWS CAF)

Un framework di linee guida e buone pratiche di AWS per aiutare le organizzazioni a sviluppare un piano efficiente ed efficace per passare con successo al cloud. AWS CAForganizza le linee guida in sei aree di interesse chiamate prospettive: azienda, persone, governance, piattaforma, sicurezza e operazioni. Le prospettive relative ad azienda, persone e governance si concentrano sulle competenze e sui processi aziendali; le prospettive relative alla piattaforma, alla sicurezza e alle operazioni si concentrano sulle competenze e sui processi tecnici. Ad esempio, la prospettiva relativa alle persone si rivolge alle parti interessate che gestiscono le risorse umane (HR), le funzioni del personale e la gestione del personale. Per questa prospettiva, AWS CAF fornisce linee guida per lo sviluppo del personale, la formazione e le comunicazioni per aiutare l'organizzazione nell'adozione efficace del cloud. Per ulteriori informazioni, consulta il AWS CAFsito Web e il AWS CAFwhite paper.

AWS Framework per la qualificazione del carico di lavoro ()AWS WQF

Uno strumento che valuta i carichi di lavoro di migrazione dei database, consiglia strategie di migrazione e fornisce stime del lavoro. AWS WQFè incluso in AWS Schema Conversion Tool ()AWS SCT. Analizza gli schemi di database e gli oggetti di codice, il codice dell'applicazione, le dipendenze e le caratteristiche delle prestazioni e fornisce report di valutazione.

B

bot non valido

Un bot che ha lo scopo di interrompere o causare danni a individui o organizzazioni.

BCP

Vedi la pianificazione della continuità operativa.

grafico comportamentale

Una vista unificata, interattiva dei comportamenti delle risorse e delle interazioni nel tempo. Puoi utilizzare un grafico comportamentale con Amazon Detective per esaminare tentativi di accesso non riusciti, API chiamate sospette e azioni simili. Per ulteriori informazioni, consulta Dati in un grafico comportamentale nella documentazione di Detective.

sistema big-endian

Un sistema che memorizza per primo il byte più importante. Vedi anche endianness.

Classificazione binaria

Un processo che prevede un risultato binario (una delle due classi possibili). Ad esempio, il modello di machine learning potrebbe dover prevedere problemi come "Questa e-mail è spam o non è spam?" o "Questo prodotto è un libro o un'auto?"

filtro Bloom

Una struttura di dati probabilistica ed efficiente in termini di memoria che viene utilizzata per verificare se un elemento fa parte di un set.

distribuzioni blu/verdi

Una strategia di implementazione in cui si creano due ambienti separati ma identici. La versione corrente dell'applicazione viene eseguita in un ambiente (blu) e la nuova versione dell'applicazione nell'altro ambiente (verde). Questa strategia consente di ripristinare rapidamente il sistema con un impatto minimo.

bot

Un'applicazione software che esegue attività automatizzate su Internet e simula l'attività o l'interazione umana. Alcuni bot sono utili o utili, come i web crawler che indicizzano le informazioni su Internet. Alcuni altri bot, noti come bot dannosi, hanno lo scopo di disturbare o causare danni a individui o organizzazioni.

botnet

Reti di bot infettate da malware e controllate da un'unica parte, nota come bot herder o bot operator. Le botnet sono il meccanismo più noto per scalare i bot e il loro impatto.

ramo

Un'area contenuta di un repository di codice. Il primo ramo creato in un repository è il ramo principale. È possibile creare un nuovo ramo a partire da un ramo esistente e quindi sviluppare funzionalità o correggere bug al suo interno. Un ramo creato per sviluppare una funzionalità viene comunemente detto ramo di funzionalità. Quando la funzionalità è pronta per il rilascio, il ramo di funzionalità viene ricongiunto al ramo principale. Per ulteriori informazioni, consulta Informazioni sulle filiali (documentazione). GitHub

accesso break-glass

In circostanze eccezionali e tramite una procedura approvata, un mezzo rapido per consentire a un utente di accedere a un sito a Account AWS cui in genere non dispone delle autorizzazioni necessarie. Per ulteriori informazioni, vedere l'indicatore Implementate break-glass procedures nella guida Well-Architected AWS .

strategia brownfield

L'infrastruttura esistente nell'ambiente. Quando si adotta una strategia brownfield per un'architettura di sistema, si progetta l'architettura in base ai vincoli dei sistemi e dell'infrastruttura attuali. Per l'espansione dell'infrastruttura esistente, è possibile combinare strategie brownfield e greenfield.

cache del buffer

L'area di memoria in cui sono archiviati i dati a cui si accede con maggiore frequenza.

capacità di business

Azioni intraprese da un'azienda per generare valore (ad esempio vendite, assistenza clienti o marketing). Le architetture dei microservizi e le decisioni di sviluppo possono essere guidate dalle capacità aziendali. Per ulteriori informazioni, consulta la sezione Organizzazione in base alle funzionalità aziendali del whitepaper Esecuzione di microservizi containerizzati su AWS.

pianificazione della continuità operativa (BCP)

Un piano che affronta il potenziale impatto di un evento che comporta l'interruzione dell'attività, come una migrazione su larga scala, sulle operazioni e consente a un'azienda di riprendere rapidamente le operazioni.

C

CAF

Vedi AWS Cloud Adoption Framework.

distribuzione canary

Il rilascio lento e incrementale di una versione agli utenti finali. Quando sei sicuro, distribuisci la nuova versione e sostituisci la versione corrente nella sua interezza.

CCoE

Vedi Centro di eccellenza del Cloud.

CDC

Vedi Change Data Capture.

Change Data Capture (CDC)

Il processo di tracciamento delle modifiche a un'origine dati, ad esempio una tabella di database, e di registrazione dei metadati relativi alla modifica. È possibile utilizzare CDC per vari scopi, ad esempio il controllo o la replica delle modifiche in un sistema di destinazione per mantenere la sincronizzazione.

ingegneria del caos

Introduzione intenzionale di guasti o eventi dirompenti per testare la resilienza di un sistema. Puoi usare AWS Fault Injection Service (AWS FIS) per eseguire esperimenti che stressano i tuoi AWS carichi di lavoro e valutarne la risposta.

CI/CD

Vedi integrazione e distribuzione continua.

classificazione

Un processo di categorizzazione che aiuta a generare previsioni. I modelli di ML per problemi di classificazione prevedono un valore discreto. I valori discreti sono sempre distinti l'uno dall'altro. Ad esempio, un modello potrebbe dover valutare se in un'immagine è presente o meno un'auto.

crittografia lato client

Crittografia dei dati in locale, prima che vengano Servizio AWS ricevuti dal di destinazione.

Centro di eccellenza del Cloud (CCoE)

Un team multidisciplinare che guida le iniziative di adozione del cloud in tutta l'organizzazione, tra cui lo sviluppo di best practice per il cloud, la mobilitazione delle risorse, la definizione delle tempistiche di migrazione e la guida dell'organizzazione attraverso trasformazioni su larga scala. Per ulteriori informazioni, consulta i CCoEpost sul blog Cloud AWS Enterprise Strategy.

cloud computing

La tecnologia cloud generalmente utilizzata per l'archiviazione remota di dati e la gestione dei dispositivi IoT. Il cloud computing è comunemente collegato alla tecnologia di edge computing.

modello operativo del cloud

In un'organizzazione IT, il modello operativo utilizzato per creare, maturare e ottimizzare uno o più ambienti cloud. Per ulteriori informazioni, consulta Building your Cloud Operating Model.

fasi di adozione del cloud

Le quattro fasi che le organizzazioni in genere attraversano quando migrano verso Cloud AWS:

- Progetto: esecuzione di alcuni progetti relativi al cloud per scopi di dimostrazione e apprendimento
- Fondamento: effettuare investimenti fondamentali per dimensionare l'adozione del cloud (ad esempio, creazione di una zona di destinazioneCCoE, definizione di un modello operativo)
- Migrazione: migrazione di singole applicazioni
- Reinvenzione: ottimizzazione di prodotti e servizi e innovazione nel cloud

Queste fasi sono state definite da Stephen Orban nel post del blog The Journey Toward Cloud-First & the Stages of Adoption sul blog Enterprise Strategy. Cloud AWS Per informazioni su come si relazionano alla strategia di AWS migrazione, consulta la guida di preparazione alla migrazione.

CMDB

Vedi database di gestione della configurazione.

repository di codice

Una posizione in cui il codice di origine e altri asset, come documentazione, esempi e script, vengono archiviati e aggiornati attraverso processi di controllo delle versioni. Alcuni repository cloud comuni sono GitHub e. AWS CodeCommit Ogni versione del codice è denominata ramo. In una struttura a microservizi, ogni repository è dedicato a una singola funzionalità. Una singola pipeline CI/CD può utilizzare più repository.

cache fredda

Una cache del buffer vuota, non ben popolata o contenente dati obsoleti o irrilevanti. Ciò influisce sulle prestazioni perché l'istanza di database deve leggere dalla memoria o dal disco principale, il che richiede più tempo rispetto alla lettura dalla cache del buffer.

dati freddi

Dati a cui si accede raramente e che in genere sono storici. Quando si eseguono interrogazioni di questo tipo di dati, le interrogazioni lente sono in genere accettabili. Lo spostamento di questi dati su livelli o classi di archiviazione meno costosi e a basso rendimento può ridurre i costi.

visione artificiale (CV)

Un campo dell'intelligenza artificiale che utilizza l'apprendimento automatico per analizzare ed estrarre informazioni da formati visivi come immagini e video digitali. Ad esempio, AWS Panorama offre dispositivi che aggiungono CV alle reti di telecamere locali e Amazon SageMaker fornisce algoritmi di elaborazione delle immagini per CV.

linea di configurazione

Per un carico di lavoro, una modifica della configurazione rispetto allo stato previsto. Potrebbe causare la non conformità del carico di lavoro e in genere è graduale e involontaria.

database di gestione della configurazione (CMDB)

Un repository che archivia e gestisce le informazioni su un database e il relativo ambiente IT, inclusi i componenti hardware e software e le relative configurazioni. In genere si utilizzano i dati provenienti da una CMDB fase di individuazione e analisi del portafoglio della migrazione.

Pacchetto di conformità

Una serie di AWS Config regole di e azioni correttive che puoi riunire per personalizzare i controlli di conformità e sicurezza. Puoi distribuire un pacchetto di conformità come singola entità in un Account AWS e in una regione, o all'interno di un'organizzazione, utilizzando un modello. YAML Per ulteriori informazioni, consulta Pacchetti di conformità nella documentazione di AWS Config .

integrazione e distribuzione continua (continuous integration and continuous delivery, CI/CD)

Il processo di automazione delle fasi di origine, creazione, test, gestione temporanea e produzione del processo di rilascio del software. Il processo CI/CD è comunemente descritto come una pipeline. CI/CD può aiutare ad automatizzare i processi, migliorare la produttività, migliorare la qualità del codice e velocizzare le distribuzioni. Per ulteriori informazioni, consulta Vantaggi

della distribuzione continua. CD può anche significare continuous deployment (implementazione continua). Per ulteriori informazioni, consulta Distribuzione continua e implementazione continua a confronto.

CV

Vedi visione artificiale.

D

dati a riposo

Dati stazionari nella rete, ad esempio i dati archiviati.

classificazione dei dati

Un processo per identificare e classificare i dati nella rete in base alla loro criticità e sensibilità. È un componente fondamentale di qualsiasi strategia di gestione dei rischi di sicurezza informatica perché consente di determinare i controlli di protezione e conservazione appropriati per i dati. La classificazione dei dati è un componente del pilastro della sicurezza nel Framework AWS Well-Architected. Per ulteriori informazioni, consulta Classificazione dei dati.

linea di dati

Una variazione significativa tra i dati di produzione e i dati utilizzati per addestrare un modello di machine learning o una modifica significativa dei dati di input nel tempo. La deriva dei dati può ridurre la qualità, l'accuratezza e l'equità complessive nelle previsioni dei modelli ML.

dati in transito

Dati che si spostano attivamente attraverso la rete, ad esempio tra le risorse di rete.

rete di dati

Un framework architettonico che fornisce la proprietà distribuita e decentralizzata dei dati con gestione e governance centralizzate.

minimizzazione dei dati

Il principio della raccolta e del trattamento dei soli dati strettamente necessari. Praticare la riduzione al minimo dei dati in the Cloud AWS può ridurre i rischi per la privacy, i costi e l'impronta di carbonio delle analisi.

perimetro dati

Una serie di barriere preventive nell' AWS ambiente che aiutano a garantire che solo le identità attendibili accedano alle risorse attendibili delle reti previste. Per ulteriori informazioni, consulta Building a data perimeter on. AWS

pre-elaborazione dei dati

Trasformare i dati grezzi in un formato che possa essere facilmente analizzato dal modello di ML. La pre-elaborazione dei dati può comportare la rimozione di determinate colonne o righe e l'eliminazione di valori mancanti, incoerenti o duplicati.

provenienza dei dati

Il processo di tracciamento dell'origine e della cronologia dei dati durante il loro ciclo di vita, ad esempio il modo in cui i dati sono stati generati, trasmessi e archiviati.

soggetto dei dati

Un individuo i cui dati vengono raccolti ed elaborati.

data warehouse

Un sistema di gestione dei dati che supporta la business intelligence, come l'analisi. I data warehouse contengono in genere grandi quantità di dati storici e vengono generalmente utilizzati per interrogazioni e analisi.

linguaggio di definizione del database (DDL)

Istruzioni o comandi per creare o modificare la struttura di tabelle e oggetti in un database.

linguaggio di manipolazione del database () DML

Istruzioni o comandi per modificare (inserire, aggiornare ed eliminare) informazioni in un database.

DDL

Vedi linguaggio di definizione del database.

deep ensemble

Combinare più modelli di deep learning per la previsione. È possibile utilizzare i deep ensemble per ottenere una previsione più accurata o per stimare l'incertezza nelle previsioni.

deep learning

Un sottocampo del ML che utilizza più livelli di reti neurali artificiali per identificare la mappatura tra i dati di input e le variabili target di interesse.

defense-in-depth

Un approccio alla sicurezza delle informazioni in cui una serie di meccanismi e controlli di sicurezza sono accuratamente stratificati su una rete di computer per proteggere la riservatezza, l'integrità e la disponibilità della rete e dei dati al suo interno. Quando adotti questa strategia in AWS, puoi aggiungere più controlli a diversi livelli della AWS Organizations struttura per proteggere le risorse. Ad esempio, un defense-in-depth approccio potrebbe combinare l'autenticazione a più fattori, la segmentazione della rete e la crittografia.

amministratore delegato

In AWS Organizations, un servizio compatibile può registrare un account AWS membro di per amministrare gli account dell'organizzazione e gestire le autorizzazioni per quel servizio. Questo account è denominato amministratore delegato per quel servizio specifico. Per ulteriori informazioni e un elenco di servizi compatibili, consulta Servizi che funzionano con AWS Organizations nella documentazione di AWS Organizations .

implementazione

Il processo di creazione di un'applicazione, di nuove funzionalità o di correzioni di codice disponibili nell'ambiente di destinazione. L'implementazione prevede l'applicazione di modifiche in una base di codice, seguita dalla creazione e dall'esecuzione di tale base di codice negli ambienti applicativi.

Ambiente di sviluppo

Vedi ambiente.

controllo di rilevamento

Un controllo di sicurezza progettato per rilevare, registrare e avvisare dopo che si è verificato un evento. Questi controlli rappresentano una seconda linea di difesa e avvisano l'utente in caso di eventi di sicurezza che aggirano i controlli preventivi in vigore. Per ulteriori informazioni, consulta Controlli di rilevamento in Implementazione dei controlli di sicurezza in AWS.

mappatura del flusso di valore di sviluppo () DVSM

Un processo utilizzato per identificare e dare priorità ai vincoli che influiscono negativamente sulla velocità e sulla qualità nel ciclo di vita dello sviluppo del software. DVSMestende il processo di

mappatura del flusso di valore originariamente progettato per pratiche di produzione snella. Si concentra sulle fasi e sui team necessari per creare e trasferire valore attraverso il processo di sviluppo del software.

gemello digitale

Una rappresentazione virtuale di un sistema reale, ad esempio un edificio, una fabbrica, un'attrezzatura industriale o una linea di produzione. I gemelli digitali supportano la manutenzione predittiva, il monitoraggio remoto e l'ottimizzazione della produzione.

tabella delle dimensioni

In uno schema a stella, una tabella più piccola che contiene gli attributi dei dati quantitativi in una tabella dei fatti. Gli attributi della tabella delle dimensioni sono in genere campi di testo o numeri discreti che si comportano come testo. Questi attributi vengono comunemente utilizzati per il vincolo delle query, il filtraggio e l'etichettatura dei set di risultati.

disastro

Un evento che impedisce a un carico di lavoro o a un sistema di raggiungere gli obiettivi aziendali nella sua sede principale di implementazione. Questi eventi possono essere disastri naturali, guasti tecnici o il risultato di azioni umane, come errori di configurazione involontari o attacchi di malware.

disaster recovery (DR)

La strategia e il processo utilizzati per ridurre al minimo i tempi di inattività e la perdita di dati causati da un disastro. Per ulteriori informazioni, consulta Ripristino di emergenza dei carichi di lavoro in AWS: ripristino nel cloud in AWS Well-Architected Framework.

DML

Vedi linguaggio di manipolazione del database.

progettazione basata sul dominio

Un approccio allo sviluppo di un sistema software complesso collegandone i componenti a domini in evoluzione, o obiettivi aziendali principali, perseguiti da ciascun componente. Questo concetto è stato introdotto da Eric Evans nel suo libro,Domain-Driven Design: Tackling Complexity in the Heart of Software (Boston: Addison-Wesley Professional, 2003). Per informazioni su come utilizzare la progettazione basata sul dominio con il modello del fico strangolatore (Strangler Fig), consulta la sezione Modernizzazione della versione precedente di Microsoft. ASP NET(ASMX) servizi web in modo incrementale utilizzando contenitori e Amazon API Gateway.

DOTT

Vedi disaster recovery.

rilevamento delle deviazioni

Tracciamento delle deviazioni da una configurazione di base. Ad esempio, è possibile AWS CloudFormation utilizzarlo per rilevare deviazioni nelle risorse di sistema oppure AWS Control Tower per rilevare cambiamenti nella landing zone che potrebbero influire sulla conformità ai requisiti di governance.

DVSM

Vedi la mappatura del flusso di valore dello sviluppo.

E

EDA

Vedi analisi esplorativa dei dati.

edge computing

La tecnologia che aumenta la potenza di calcolo per i dispositivi intelligenti all'edge di una rete IoT. Rispetto al cloud computing, l'edge computing può ridurre la latenza di comunicazione e migliorare i tempi di risposta.

crittografia

Un processo di elaborazione che trasforma i dati in chiaro, leggibili dall'uomo, in testo cifrato.

chiave crittografica

Una stringa crittografica di bit randomizzati generata da un algoritmo di crittografia. Le chiavi possono variare di lunghezza e ogni chiave è progettata per essere imprevedibile e univoca.

endianità

L'ordine in cui i byte vengono archiviati nella memoria del computer. I sistemi big-endian memorizzano per primo il byte più importante. I sistemi little-endian memorizzano per primo il byte meno importante.

endpoint

Vedi service endpoint.

servizio endpoint

Un servizio che puoi ospitare in un cloud privato virtuale (VPC) da condividere con altri utenti. È possibile creare un servizio endpoint con AWS PrivateLink e concedere autorizzazioni ad altri Account AWS o a AWS Identity and Access Management (IAM) principali. Questi account o principali possono connettersi al servizio endpoint in privato creando endpoint di interfaccia. VPC Per ulteriori informazioni, consulta Creazione di un servizio endpoint nella documentazione di Amazon Virtual Private Cloud (AmazonVPC).

pianificazione delle risorse aziendali () ERP

Un sistema che automatizza e gestisce i processi aziendali chiave (come la contabilità e MESla gestione dei progetti) per un'azienda.

crittografia envelope

Il processo di crittografia di una chiave di crittografia con un'altra chiave di crittografia. Per ulteriori informazioni, consulta Crittografia a busta nella documentazione di AWS Key Management Service (AWS KMS).

ambiente

Un'istanza di un'applicazione in esecuzione. Di seguito sono riportati i tipi di ambiente più comuni nel cloud computing:

- ambiente di sviluppo: un'istanza di un'applicazione in esecuzione disponibile solo per il team principale responsabile della manutenzione dell'applicazione. Gli ambienti di sviluppo vengono utilizzati per testare le modifiche prima di promuoverle negli ambienti superiori. Questo tipo di ambiente viene talvolta definito ambiente di test.
- ambienti inferiori: tutti gli ambienti di sviluppo di un'applicazione, ad esempio quelli utilizzati per le build e i test iniziali.
- ambiente di produzione: un'istanza di un'applicazione in esecuzione a cui gli utenti finali possono accedere. In una pipeline CI/CD, l'ambiente di produzione è l'ultimo ambiente di implementazione.
- ambienti superiori: tutti gli ambienti a cui possono accedere utenti diversi dal team di sviluppo principale. Si può trattare di un ambiente di produzione, ambienti di preproduzione e ambienti per i test di accettazione da parte degli utenti.

epica

Nelle metodologie agili, categorie funzionali che aiutano a organizzare e dare priorità al lavoro. Le epiche forniscono una descrizione di alto livello dei requisiti e delle attività di implementazione.

Ad esempio le epiche AWS CAF di sicurezza includono la gestione delle identità e degli accessi, i controlli investigativi, la sicurezza dell'infrastruttura, la protezione dei dati e la risposta agli incidenti. Per ulteriori informazioni sulle epiche, consulta la strategia di migrazione AWS , consulta la guida all'implementazione del programma.

ERP

Vedi la pianificazione delle risorse aziendali.

analisi esplorativa dei dati () EDA

Il processo di analisi di un set di dati per comprenderne le caratteristiche principali. Si raccolgono o si aggregano dati e quindi si eseguono indagini iniziali per trovare modelli, rilevare anomalie e verificare ipotesi. EDAviene eseguita calcolando statistiche di riepilogo e creando visualizzazioni di dati.

F

tabella dei fatti

Il tavolo centrale con schema a stella. Memorizza dati quantitativi sulle operazioni aziendali. In genere, una tabella dei fatti contiene due tipi di colonne: quelle che contengono misure e quelle che contengono una chiave esterna per una tabella di dimensioni.

fallire velocemente

Una filosofia che utilizza test frequenti e incrementali per ridurre il ciclo di vita dello sviluppo. È una parte fondamentale di un approccio agile.

limite di isolamento dei guasti

Nel Cloud AWS, un limite come una zona di disponibilità Regione AWS, un piano di controllo o un piano dati che limita l'effetto di un errore e aiuta a migliorare la resilienza dei carichi di lavoro. Per ulteriori informazioni, consulta AWS Fault Isolation Boundaries.

ramo di funzionalità

Vedi filiale.

caratteristiche

I dati di input che usi per fare una previsione. Ad esempio, in un contesto di produzione, le caratteristiche potrebbero essere immagini acquisite periodicamente dalla linea di produzione.

importanza delle caratteristiche

Quanto è importante una caratteristica per le previsioni di un modello. Di solito viene espresso come punteggio numerico che può essere calcolato con varie tecniche, come Shapley Additive Explanations (SHAP) e gradienti integrati. Per ulteriori informazioni, consulta la sezione Interpretabilità dei modelli di machine learning con:AWS.

trasformazione delle funzionalità

Per ottimizzare i dati per il processo di machine learning, incluso l'arricchimento dei dati con fonti aggiuntive, il dimensionamento dei valori o l'estrazione di più set di informazioni da un singolo campo di dati. Ciò consente al modello di ML di trarre vantaggio dai dati. Ad esempio, se suddividi la data "2021-05-27 00:15:37" in "2021", "maggio", "giovedì" e "15", puoi aiutare l'algoritmo di apprendimento ad apprendere modelli sfumati associati a diversi componenti dei dati.

FGAC

Vedi Controllo granulare degli accessi.

controllo granulare degli accessi () FGAC

L'uso di più condizioni per consentire o rifiutare una richiesta di accesso.

migrazione flash-cut

Un metodo di migrazione del database che utilizza la replica continua dei dati tramite acquisizione dei dati delle modifiche per migrare i dati nel più breve tempo possibile, anziché utilizzare un approccio graduale. L'obiettivo è ridurre al minimo i tempi di inattività.

G

blocco geografico

Vedi restrizioni geografiche.

limitazioni geografiche (blocco geografico)

In Amazon CloudFront, un'opzione per impedire agli utenti di paesi specifici di accedere alle distribuzioni di contenuti. Puoi utilizzare un elenco consentito o un elenco di blocco per specificare i paesi approvati e vietati. Per ulteriori informazioni, consulta Limitazione della distribuzione geografica dei contenuti nella CloudFront documentazione di.

Flusso di lavoro di GitFlow

Un approccio in cui gli ambienti inferiori e superiori utilizzano rami diversi in un repository di codice di origine. Il flusso di lavoro Gitflow è considerato obsoleto e il flusso di lavoro basato su trunk è l'approccio moderno e preferibile.

strategia greenfield

L'assenza di infrastrutture esistenti in un nuovo ambiente. Quando si adotta una strategia greenfield per un'architettura di sistema, è possibile selezionare tutte le nuove tecnologie senza il vincolo della compatibilità con l'infrastruttura esistente, nota anche come brownfield. Per l'espansione dell'infrastruttura esistente, è possibile combinare strategie brownfield e greenfield.

guardrail

Una regola di livello elevato che consente di governare risorse, policy e conformità tra le unità organizzative ()OUs. I guardrail preventivi applicano le policy per garantire l'allineamento agli standard di conformità. Vengono implementati utilizzando le policy di controllo dei servizi e i limiti IAM delle autorizzazioni. I guardrail di rilevamento rilevano le violazioni delle policy e i problemi di conformità e generano avvisi per porvi rimedio. Sono implementati utilizzando Amazon AWS Config AWS Security Hub GuardDuty AWS Trusted Advisor, Amazon Inspector e controlli personalizzati AWS Lambda .

H

AH

Vedi disponibilità elevata.

migrazione di database eterogenea

Migrazione del database di origine in un database di destinazione che utilizza un motore di database diverso (ad esempio, da Oracle ad Amazon Aurora). La migrazione eterogenea fa in genere parte di uno sforzo di riprogettazione e la conversione dello schema può essere un'attività complessa. AWS offre AWS SCT che aiuta con le conversioni dello schema.

alta disponibilità (HA)

La capacità di un carico di lavoro di funzionare in modo continuo, senza intervento, in caso di sfide o disastri. I sistemi HA sono progettati per il failover automatico, fornire costantemente prestazioni di alta qualità e gestire carichi e guasti diversi con un impatto minimo sulle prestazioni.

modernizzazione storica

Un approccio utilizzato per modernizzare e aggiornare i sistemi di tecnologia operativa (OT) per soddisfare meglio le esigenze dell'industria manifatturiera. Uno storico è un tipo di database utilizzato per raccogliere e archiviare dati da varie fonti in una fabbrica.

migrazione di database omogenea

Migrazione del database di origine in un database di destinazione che condivide lo stesso motore di database (ad esempio, da Microsoft SQL Server ad Amazon RDS for SQL Server). La migrazione omogenea fa in genere parte di un'operazione di rehosting o ridefinizione della piattaforma. Per migrare lo schema è possibile utilizzare le utilità native del database.

dati caldi

Dati a cui si accede frequentemente, come dati in tempo reale o dati di traduzione recenti. Questi dati richiedono in genere un livello o una classe di storage ad alte prestazioni per fornire risposte rapide alle query.

hotfix

Una soluzione urgente per un problema critico in un ambiente di produzione. A causa della sua urgenza, un hotfix viene in genere creato al di fuori del tipico flusso di lavoro di DevOps rilascio.

periodo di hypercare

Subito dopo la conversione, il periodo di tempo in cui un team di migrazione gestisce e monitora le applicazioni migrate nel cloud per risolvere eventuali problemi. In genere, questo periodo dura da 1 a 4 giorni. Al termine del periodo di hypercare, il team addetto alla migrazione in genere trasferisce la responsabilità delle applicazioni al team addetto alle operazioni cloud.

I

IaC

Considera l'infrastruttura come codice.

Policy basata su identità

Una policy collegata a uno o più IAM principali che definisce le relative autorizzazioni all'interno dell' Cloud AWS ambiente.

applicazione inattiva

Un'applicazione che prevede un uso di memoria medio CPU compreso tra il 5% e il 20% in un periodo di 90 giorni. In un progetto di migrazione, è normale ritirare queste applicazioni o mantenerle on-premise.

IIoT

Vedi Internet delle cose industriale.

infrastruttura immutabile

Un modello che implementa una nuova infrastruttura per i carichi di lavoro di produzione anziché aggiornare, applicare patch o modificare l'infrastruttura esistente. Le infrastrutture immutabili sono intrinsecamente più coerenti, affidabili e prevedibili delle infrastrutture mutabili. Per ulteriori informazioni, visitare il sito Deploy using immutable infrastructure in Well-Architected Framework. AWS

in ingresso (ingress) VPC

In un'architettura AWS multi-account, una VPC che accetta, ispeziona e instrada le connessioni di rete dall'esterno di un'applicazione. Nel documento Architettura di riferimento per la AWS sicurezza di si consiglia di configurare l'account di rete con informazioni in entrata, in uscita e di ispezione VPCs per proteggere l'interfaccia bidirezionale tra l'applicazione e Internet in generale.

migrazione incrementale

Una strategia di conversione in cui si esegue la migrazione dell'applicazione in piccole parti anziché eseguire una conversione singola e completa. Ad esempio, inizialmente potresti spostare solo alcuni microservizi o utenti nel nuovo sistema. Dopo aver verificato che tutto funzioni correttamente, puoi spostare in modo incrementale microservizi o utenti aggiuntivi fino alla disattivazione del sistema legacy. Questa strategia riduce i rischi associati alle migrazioni di grandi dimensioni.

Industria 4.0

Un termine introdotto da Klaus Schwab nel 2016 per riferirsi alla modernizzazione dei processi di produzione attraverso progressi in termini di connettività, dati in tempo reale, automazione, analisi e AI/ML.

infrastruttura

Tutte le risorse e gli asset contenuti nell'ambiente di un'applicazione.

infrastruttura come codice (IaC)

Il processo di provisioning e gestione dell'infrastruttura di un'applicazione tramite un insieme di file di configurazione. Il processo IaC è progettato per aiutarti a centralizzare la gestione dell'infrastruttura, a standardizzare le risorse e a dimensionare rapidamente, in modo che i nuovi ambienti siano ripetibili, affidabili e coerenti.

Internet delle cose industriale (IIoT)

L'uso di sensori e dispositivi connessi a Internet nei settori industriali, come quello manifatturiero, energetico, automobilistico, sanitario, delle scienze della vita e dell'agricoltura. Per ulteriori informazioni, consulta Creazione di una strategia di trasformazione digitale dell'Internet delle cose industriale (IIoT).

ispezione VPC

In un'architettura AWS multi-account, una soluzione centralizzata VPC che gestisce le ispezioni del traffico di rete tra VPCs (in uguali o diverse Regioni AWS), Internet e le reti on-premise. Nel documento Architettura di riferimento per la AWS sicurezza di si consiglia di configurare l'account di rete con informazioni in entrata, in uscita e di ispezione VPCs per proteggere l'interfaccia bidirezionale tra l'applicazione e Internet in generale.

Internet of Things (IoT)

La rete di oggetti fisici connessi con sensori o processori incorporati che comunicano con altri dispositivi e sistemi tramite Internet o una rete di comunicazione locale. Per ulteriori informazioni, consulta Cos'è l'IoT?

interpretabilità

Una caratteristica di un modello di machine learning che descrive il grado in cui un essere umano è in grado di comprendere in che modo le previsioni del modello dipendono dai suoi input. Per ulteriori informazioni, consulta la sezione Interpretabilità dei modelli di machine learning con AWS.

IoT

Vedi Internet of Things.

Libreria di informazioni IT (ITIL)

Una serie di best practice per offrire servizi IT e allinearli ai requisiti aziendali. ITILfornisce le basi perITSM.

gestione dei servizi IT (ITSM)

Attività associate alla progettazione, implementazione, gestione e supporto dei servizi IT per un'organizzazione. Per informazioni sull'integrazione delle operazioni cloud con ITSM gli strumenti, consulta la guida all'integrazione delle operazioni.

ITIL

Vedi la libreria di informazioni IT.

ITSM

Vedi Gestione dei servizi IT.

L

controllo accessi basato su etichette () LBAC

Un'implementazione del controllo di accesso obbligatorio (MAC) in cui agli utenti e ai dati stessi viene assegnato esplicitamente un valore di etichetta di sicurezza. L'intersezione tra l'etichetta di sicurezza dell'utente e l'etichetta di sicurezza dei dati determina quali righe e colonne possono essere visualizzate dall'utente.

zona di destinazione

Una zona di destinazione è un AWS ambiente multi-account ben progettato, scalabile e sicuro. Questo è un punto di partenza dal quale le organizzazioni possono avviare e distribuire rapidamente carichi di lavoro e applicazioni con fiducia nel loro ambiente di sicurezza e infrastruttura. Per ulteriori informazioni sulle zone di destinazione, consulta la sezione Configurazione di un ambiente AWS multi-account sicuro e scalabile.

migrazione su larga scala

Una migrazione di 300 o più server.

LBAC

Vedi Controllo degli accessi basato su etichette.

Privilegio minimo

La best practice di sicurezza per la concessione delle autorizzazioni minime richieste per eseguire un'attività. Per ulteriori informazioni, consulta Applicazione delle autorizzazioni del privilegio minimo nella documentazione di. IAM

eseguire il rehosting (lift and shift)

Vedi 7 R.

sistema little-endian

Un sistema che memorizza per primo il byte meno importante. Vedi anche endianità.

ambienti inferiori

Vedi ambiente.

M

machine learning (ML)

Un tipo di intelligenza artificiale che utilizza algoritmi e tecniche per il riconoscimento e l'apprendimento di schemi. Il machine learning analizza e apprende dai dati registrati, come i dati dell'Internet delle cose (IoT), per generare un modello statistico basato su modelli. Per ulteriori informazioni, consulta la sezione Machine learning.

ramo principale

Vedi filiale.

malware

Software progettato per compromettere la sicurezza o la privacy del computer. Il malware potrebbe interrompere i sistemi informatici, divulgare informazioni sensibili o ottenere accessi non autorizzati. Esempi di malware includono virus, worm, ransomware, cavalli di Troia, spyware e keylogger.

servizi gestiti

Servizi AWS per cui AWS gestisce il livello di infrastruttura, il sistema operativo e le piattaforme e si accede agli endpoint per archiviare e recuperare i dati. Amazon Simple Storage Service (Amazon S3) e Amazon DynamoDB sono esempi di servizi gestiti. Questi sono noti anche come servizi astratti.

sistema di esecuzione della produzione () MES

Un sistema software per tracciare, monitorare, documentare e controllare i processi di produzione che convertono le materie prime in prodotti finiti in officina.

MAP

Vedi Migration Acceleration Program.

meccanismo

Un processo completo in cui si crea uno strumento, si promuove l'adozione dello strumento e quindi si esaminano i risultati per apportare le modifiche. Un meccanismo è un ciclo che si rafforza e si migliora man mano che funziona. Per ulteriori informazioni, consulta Creazione di meccanismi nel Framework AWS Well-Architected.

account membro

Tutti Account AWS gli diversi dall'account di gestione che fanno parte di un'organizzazione in AWS Organizations. Un account può essere membro di una sola organizzazione alla volta.

MES

Vedi Manufacturing Execution System.

Message Queuing Telemetry Transport () MQTT

Un protocollo di comunicazione machine-to-machine (M2M) leggero, basato sul modello di pubblicazione/sottoscrizione, per dispositivi IoT con risorse limitate.

microservizio

Un piccolo servizio indipendente che comunica tramite linee ben definite APIs ed è in genere di proprietà di piccoli team autonomi. Ad esempio, un sistema assicurativo potrebbe includere microservizi che si riferiscono a funzionalità aziendali, come vendite o marketing, o sottodomini, come acquisti, reclami o analisi. I vantaggi dei microservizi includono agilità, dimensionamento flessibile, facilità di implementazione, codice riutilizzabile e resilienza. Per ulteriori informazioni, consulta la sezione Integrazione dei microservizi utilizzando servizi AWS serverless.

architettura di microservizi

Un approccio alla creazione di un'applicazione con componenti indipendenti che eseguono ogni processo applicativo come microservizio. Questi microservizi comunicano tramite un'interfaccia ben definita utilizzando il formato leggero. APIs Ogni microservizio in questa architettura può essere aggiornato, distribuito e dimensionato per soddisfare la richiesta di funzioni specifiche di un'applicazione. Per ulteriori informazioni, consulta la sezione Implementazione di microservizi su AWS.

Programma MAP di accelerazione della migrazione ()

Un AWS programma che offre consulenza, formazione e servizi per aiutare le organizzazioni a costruire una solida base operativa per il passaggio al cloud e per contribuire a compensare il costo iniziale delle migrazioni. MAPinclude una metodologia di migrazione per eseguire le migrazioni precedenti in modo metodico e un set di strumenti per automatizzare e accelerare gli scenari di migrazione comuni.

migrazione su larga scala

Il processo di trasferimento della maggior parte del portfolio di applicazioni sul cloud avviene a ondate, con più applicazioni trasferite a una velocità maggiore in ogni ondata. Questa fase utilizza le migliori pratiche e le lezioni apprese nelle fasi precedenti per implementare una fabbrica di migrazione di team, strumenti e processi per semplificare la migrazione dei carichi di lavoro attraverso l'automazione e la distribuzione agile. Questa è la terza fase della strategia di migrazione AWS.

fabbrica di migrazione

Team interfunzionali che semplificano la migrazione dei carichi di lavoro attraverso approcci automatizzati e agili. I team della fabbrica di migrazione includono in genere operazioni, analisti e proprietari aziendali, ingegneri della migrazione, sviluppatori e DevOps professionisti che lavorano a tappe. Tra il 20% e il 50% di un portfolio di applicazioni aziendali è costituito da schemi ripetuti che possono essere ottimizzati con un approccio di fabbrica. Per ulteriori informazioni, consulta la discussione sulle fabbriche di migrazione e la Guida alla fabbrica di migrazione al cloud in questo set di contenuti.

metadati di migrazione

Le informazioni sull'applicazione e sul server necessarie per completare la migrazione. Ogni modello di migrazione richiede un set diverso di metadati di migrazione. Esempi di metadati di migrazione includono la sottorete di destinazione, il gruppo di sicurezza e AWS l'account.

modello di migrazione

Un'attività di migrazione ripetibile che descrive in dettaglio la strategia di migrazione, la destinazione della migrazione e l'applicazione o il servizio di migrazione utilizzati. Esempio: eseguire il rehosting della migrazione ad Amazon EC2 con AWS Application Migration Service.

Valutazione del portfolio di migrazione (MPA)

Uno strumento online che fornisce informazioni per la convalida del business case per la migrazione a. Cloud AWS MPAoffre una valutazione dettagliata del portfolio (dimensionamento

corretto dei server, prezzi, analisi TCO dei costi di migrazione) e pianificazione della migrazione (analisi e raccolta dei dati delle applicazioni, raggruppamento delle applicazioni, prioritizzazione delle migrazioni e pianificazione delle ondate). Lo MPAstrumento (richiede il login) è disponibile gratuitamente per tutti i AWS consulenti e i consulenti APN partner.

Valutazione della preparazione alla migrazione () MRA

Il processo di acquisizione di informazioni sullo stato di idoneità al cloud di un'organizzazione, l'identificazione dei punti di forza e di debolezza e la creazione di un piano d'azione per colmare le lacune identificate, utilizzando. AWS CAF Per ulteriori informazioni, consulta la guida di preparazione alla migrazione. MRAè la prima fase della strategia di AWS migrazione.

strategia di migrazione

L'approccio utilizzato per eseguire la migrazione di un carico di lavoro verso. Cloud AWS Per ulteriori informazioni, consulta la voce 7 R in questo glossario e consulta Mobilita la tua organizzazione per accelerare le migrazioni su larga scala.

ML

Vedi machine learning.

modernizzazione

Trasformazione di un'applicazione obsoleta (legacy o monolitica) e della relativa infrastruttura in un sistema agile, elastico e altamente disponibile nel cloud per ridurre i costi, aumentare l'efficienza e sfruttare le innovazioni. Per ulteriori informazioni, consulta la sezione Strategia per modernizzare le applicazioni nel Cloud AWS.

valutazione della preparazione alla modernizzazione

Una valutazione che aiuta a determinare la preparazione alla modernizzazione delle applicazioni di un'organizzazione, identifica vantaggi, rischi e dipendenze e determina in che misura l'organizzazione può supportare lo stato futuro di tali applicazioni. Il risultato della valutazione è uno schema dell'architettura di destinazione, una tabella di marcia che descrive in dettaglio le fasi di sviluppo e le tappe fondamentali del processo di modernizzazione e un piano d'azione per colmare le lacune identificate. Per ulteriori informazioni, consulta la sezione Valutazione della preparazione alla modernizzazione per le applicazioni in. Cloud AWS

applicazioni monolitiche (monoliti)

Applicazioni eseguite come un unico servizio con processi strettamente collegati. Le applicazioni monolitiche presentano diversi inconvenienti. Se una funzionalità dell'applicazione registra un

picco di domanda, l'intera architettura deve essere dimensionata. L'aggiunta o il miglioramento delle funzionalità di un'applicazione monolitica diventa inoltre più complessa man mano che la base di codice cresce. Per risolvere questi problemi, puoi utilizzare un'architettura di microservizi. Per ulteriori informazioni, consulta la sezione Scomposizione dei monoliti in microservizi.

MPA

Vedi Migration Portfolio Assessment.

MQTT

Vedi Message Queuing Telemetry Transport.

classificazione multiclasse

Un processo che aiuta a generare previsioni per più classi (prevedendo uno o più di due risultati). Ad esempio, un modello di machine learning potrebbe chiedere "Questo prodotto è un libro, un'auto o un telefono?" oppure "Quale categoria di prodotti è più interessante per questo cliente?"

infrastruttura mutevole

Un modello che aggiorna e modifica l'infrastruttura esistente per i carichi di lavoro di produzione. Per migliorare la coerenza, l'affidabilità e la prevedibilità, il AWS Well-Architected Framework consiglia l'uso di un'infrastruttura immutabile come best practice.

O

OAC

Vedi Origin Access Control.

OAI

Vedi Origin Access Identity.

OCM

Vedi gestione delle modifiche organizzative.

migrazione offline

Un metodo di migrazione in cui il carico di lavoro di origine viene eliminato durante il processo di migrazione. Questo metodo prevede tempi di inattività prolungati e viene in genere utilizzato per carichi di lavoro piccoli e non critici.

OI

Vedi l'integrazione delle operazioni.

OLA

Vedi accordo a livello operativo.

migrazione online

Un metodo di migrazione in cui il carico di lavoro di origine viene copiato sul sistema di destinazione senza essere messo offline. Le applicazioni connesse al carico di lavoro possono continuare a funzionare durante la migrazione. Questo metodo comporta tempi di inattività pari a zero o comunque minimi e viene in genere utilizzato per carichi di lavoro di produzione critici.

OPC-UA

Vedi Open Process Communications - Unified Architecture.

Comunicazioni a processo aperto - Architettura unificata (OPC-UA)

Un protocollo di comunicazione machine-to-machine (M2M) per l'automazione industriale. OPC-UA fornisce uno standard di interoperabilità con schemi di crittografia, autenticazione e autorizzazione dei dati.

accordo a livello operativo () OLA

Un accordo che chiarisce quali sono gli impegni reciproci tra i gruppi IT funzionali, a supporto di un accordo sul livello di servizio (). SLA

revisione della prontezza operativa () ORR

Un elenco di domande e best practice associate che aiutano a comprendere, valutare, prevenire o ridurre la portata degli incidenti e dei possibili guasti. Per ulteriori informazioni, vedere Operational Readiness Reviews (ORR) nel AWS Well-Architected Framework.

tecnologia operativa (OT)

Sistemi hardware e software che interagiscono con l'ambiente fisico per controllare le operazioni, le apparecchiature e le infrastrutture industriali. Nella produzione, l'integrazione di sistemi OT e di tecnologia dell'informazione (IT) è un obiettivo chiave per le trasformazioni dell'Industria 4.0.

integrazione delle operazioni (OI)

Il processo di modernizzazione delle operazioni nel cloud, che prevede la pianificazione, l'automazione e l'integrazione della disponibilità. Per ulteriori informazioni, consulta la guida all'integrazione delle operazioni.

trail organizzativo

Un trail creato da AWS CloudTrail che registra tutti gli eventi per tutti gli Account AWS in un'organizzazione di. AWS Organizations Questo percorso viene creato in ogni Account AWS che fa parte dell'organizzazione e tiene traccia dell'attività in ogni account. Per ulteriori informazioni, consulta Creazione di un trail per un'organizzazione nella CloudTrail documentazione di.

gestione del cambiamento organizzativo (OCM)

Un framework per la gestione di trasformazioni aziendali importanti e che comportano l'interruzione delle attività dal punto di vista delle persone, della cultura e della leadership. OCMaiuta le organizzazioni a prepararsi e passare a nuovi sistemi e strategie accelerando l'adozione del cambiamento, affrontando i problemi di transizione e promuovendo cambiamenti culturali e organizzativi. Nella strategia di AWS migrazione, questo framework si chiama accelerazione delle persone, a causa della velocità di cambiamento richiesta nei progetti di adozione del cloud. Per ulteriori informazioni, consulta la OCMguida di.

controllo dell'accesso all'origine (OAC)

Nel CloudFront, un'opzione avanzata per limitare l'accesso e proteggere i contenuti Amazon Simple Storage Service (Amazon S3). OACsupporta tutti i bucket S3 in generale Regioni AWS, la crittografia lato server con AWS KMS (SSE-KMS) e la crittografia dinamica `PUT` e `DELETE` le richieste al bucket S3.

identità di accesso origine (OAI)

In CloudFront, un'opzione per limitare l'accesso e proteggere i contenuti Amazon S3. Quando usiOAI, CloudFront crea un principale con cui Amazon S3 può autenticarsi. I principali autenticati possono accedere ai contenuti in un bucket S3 solo tramite una distribuzione specifica. CloudFront Vedi anche OAC, che fornisce un controllo degli accessi più granulare e avanzato.

ORR

Vedi la revisione della prontezza operativa.

NON

Vedi tecnologia operativa.

in uscita (egress) VPC

In un'architettura AWS multi-account, VPC che gestisce le connessioni di rete avviate dall'interno di un'applicazione. Nel documento Architettura di riferimento per la AWS sicurezza di si consiglia di configurare l'account di rete con informazioni in entrata, in uscita e di ispezione VPCs per proteggere l'interfaccia bidirezionale tra l'applicazione e Internet in generale.

P

limite delle autorizzazioni

Una policy di IAM gestione collegata ai IAM principali per impostare le autorizzazioni massime che l'utente o il ruolo possono avere. Per ulteriori informazioni, consulta Limiti delle autorizzazioni nella IAM documentazione di.

informazioni di identificazione personale () PII

Informazioni che, se visualizzate direttamente o abbinate ad altri dati correlati, possono essere utilizzate per dedurre ragionevolmente l'identità di un individuo. Alcuni esempi PII includono nomi, indirizzi e informazioni di contatto.

PII

Visualizza le informazioni di identificazione personale.

playbook

Una serie di passaggi predefiniti che raccolgono il lavoro associato alle migrazioni, come l'erogazione delle funzioni operative principali nel cloud. Un playbook può assumere la forma di script, runbook automatici o un riepilogo dei processi o dei passaggi necessari per gestire un ambiente modernizzato.

PLC

Vedi controllore logico programmabile.

PLM

Vedi la gestione del ciclo di vita del prodotto.

policy

Un oggetto in grado di definire le autorizzazioni (vedi politica basata sull'identità), specificare le condizioni di accesso (vedi politicabasata sulle risorse) o definire le autorizzazioni massime per tutti gli account di un'organizzazione in (vedi politica di controllo dei servizi). AWS Organizations

persistenza poliglotta

Scelta indipendente della tecnologia di archiviazione di dati di un microservizio in base ai modelli di accesso ai dati e ad altri requisiti. Se i microservizi utilizzano la stessa tecnologia di archiviazione di dati, possono incontrare problemi di implementazione o registrare prestazioni

scadenti. I microservizi vengono implementati più facilmente e ottengono prestazioni e scalabilità migliori se utilizzano l'archivio dati più adatto alle loro esigenze. Per ulteriori informazioni, consulta la sezione Abilitazione della persistenza dei dati nei microservizi.

valutazione del portfolio

Un processo di scoperta, analisi e definizione delle priorità del portfolio di applicazioni per pianificare la migrazione. Per ulteriori informazioni, consulta la pagina Valutazione della preparazione alla migrazione.

predicate

Una condizione di interrogazione che restituisce o, in genere, si trova in una clausola `WHERE`. `true` `false`

pushdown del predicate

Una tecnica di ottimizzazione delle query del database che filtra i dati della query prima del trasferimento. Ciò riduce la quantità di dati che devono essere recuperati ed elaborati dal database relazionale e migliora le prestazioni delle query.

controllo preventivo

Un controllo di sicurezza progettato per impedire il verificarsi di un evento. Questi controlli sono la prima linea di difesa per impedire accessi non autorizzati o modifiche indesiderate alla rete. Per ulteriori informazioni, consulta Controlli preventivi in Implementazione dei controlli di sicurezza in AWS.

principale

Un'entità in AWS che può eseguire operazioni e accedere alle risorse. Questa entità è in genere un utente root per un Account AWS, un IAM ruolo o un utente. Per ulteriori informazioni, consulta i termini e i concetti di Principal in Roles nella IAM documentazione.

Privacy fin dalla progettazione

Un approccio all'ingegneria dei sistemi che tiene conto della privacy durante l'intero processo di progettazione.

zone ospitate private

Un container che contiene informazioni su come si desidera che Amazon Route 53 risponda alle DNS query per un dominio e i relativi sottodomini all'interno di uno o più. VPCs Per ulteriori informazioni, consulta Utilizzo delle zone ospitate private nella documentazione di Route 53.

controllo proattivo

Un controllo di sicurezza progettato per impedire l'implementazione di risorse non conformi. Questi controlli analizzano le risorse prima del loro provisioning. Se la risorsa non è conforme al controllo, non viene fornita. Per ulteriori informazioni, consulta la guida di riferimento sui controlli nella AWS Control Tower documentazione e consulta Controlli proattivi in Implementazione dei controlli di sicurezza su. AWS

gestione del ciclo di vita del prodotto () PLM

La gestione dei dati e dei processi di un prodotto durante l'intero ciclo di vita, dalla progettazione, sviluppo e lancio, attraverso la crescita e la maturità, fino al declino e alla rimozione.

Ambiente di produzione

Vedi ambiente.

controllore logico programmabile () PLC

Nella produzione, un computer altamente affidabile e adattabile che monitora le macchine e automatizza i processi di produzione.

pseudonimizzazione

Il processo di sostituzione degli identificatori personali in un set di dati con valori segnaposto. La pseudonimizzazione può aiutare a proteggere la privacy personale. I dati pseudonimizzati sono ancora considerati dati personali.

pubblica/iscriviti (pub/sub)

Un pattern che consente comunicazioni asincrone tra microservizi per migliorare la scalabilità e la reattività. Ad esempio, in un sistema basato su microservizi MES, un microservizio può pubblicare messaggi di eventi su un canale a cui altri microservizi possono abbonarsi. Il sistema può aggiungere nuovi microservizi senza modificare il servizio di pubblicazione.

Q

Piano di query

Una serie di passaggi, come le istruzioni, utilizzati per accedere ai dati in un sistema di database SQL relazionale.

regressione del piano di query

Quando un ottimizzatore del servizio di database sceglie un piano non ottimale rispetto a prima di una determinata modifica all'ambiente di database. Questo può essere causato da modifiche a statistiche, vincoli, impostazioni dell'ambiente, associazioni dei parametri di query e aggiornamenti al motore di database.

R

RACImatrice

Vedi responsabile, responsabile, consultato, informato () RACI.

ransomware

Un software dannoso progettato per bloccare l'accesso a un sistema informatico o ai dati fino a quando non viene effettuato un pagamento.

RASCImatrice

Vedi responsabile, responsabile, consultato, informato () RACI.

RCAC

Vedi controllo dell'accesso a righe e colonne.

replica di lettura

Una copia di un database utilizzata per scopi di sola lettura. È possibile indirizzare le query alla replica di lettura per ridurre il carico sul database principale.

rearchitetta

Vedi 7 Rs.

obiettivo del punto di ripristino (RPO)

Il periodo di tempo massimo accettabile dall'ultimo punto di ripristino dei dati. Questo determina ciò che si considera una perdita di dati accettabile tra l'ultimo punto di ripristino e l'interruzione del servizio.

obiettivo del tempo di ripristino (RTO)

Il ritardo massimo accettabile tra l'interruzione del servizio e il ripristino del servizio.

rifattorizzazione

Vedi 7 R.

Regione

Una raccolta di AWS risorse in un'area geografica. Ogni Regione AWS è isolata e indipendente dalle altre per fornire tolleranza agli errori, stabilità e resilienza. Per ulteriori informazioni, consulta Specificare cosa può usare Regioni AWS il tuo account.

regressione

Una tecnica di ML che prevede un valore numerico. Ad esempio, per risolvere il problema "A che prezzo verrà venduta questa casa?" un modello di ML potrebbe utilizzare un modello di regressione lineare per prevedere il prezzo di vendita di una casa sulla base di dati noti sulla casa (ad esempio, la metratura).

rehosting

Vedi 7 R.

rilascio

In un processo di implementazione, l'atto di promuovere modifiche a un ambiente di produzione.

trasferisco

Vedi 7 Rs.

ripiattaforma

Vedi 7 Rs.

riacquisto

Vedi 7 Rs.

resilienza

La capacità di un'applicazione di resistere o ripristinare le interruzioni. L'elevata disponibilità e il disaster recovery sono considerazioni comuni quando si pianifica la resilienza in. Cloud AWS Per ulteriori informazioni, consulta Cloud AWS Resilienza.

policy basata su risorse

Una policy associata a una risorsa, ad esempio un bucket Amazon S3, un endpoint o una chiave di crittografia. Questo tipo di policy specifica a quali principali è consentito l'accesso, le azioni supportate e qualsiasi altra condizione che deve essere soddisfatta.

matrice di assegnazione di responsabilità () RACI

Una matrice che definisce i ruoli e le responsabilità di tutte le parti coinvolte nelle attività di migrazione e nelle operazioni cloud. Il nome della matrice deriva dai tipi di responsabilità definiti nella matrice: responsabile (R), responsabile (A), consultato (C) e informato (I). Il tipo di supporto (S) è facoltativo. Se includi il supporto, la matrice viene chiamata RASCImatrice e se la escludi, viene chiamata RACImatrice.

controllo reattivo

Un controllo di sicurezza progettato per favorire la correzione di eventi avversi o deviazioni dalla baseline di sicurezza. Per ulteriori informazioni, consulta Controlli reattivi in Implementazione dei controlli di sicurezza in AWS.

retain

Vedi 7 R.

andare in pensione

Vedi 7 Rs.

rotazione

Processo mediante il quale si aggiorna periodicamente il segreto per rendere più difficile l'accesso alle credenziali da parte di un utente malintenzionato.

controllo dell'accesso a righe e colonne () RCAC

L'uso di SQL espressioni di base e flessibili che hanno regole di accesso definite. RCACè costituito da permessi di riga e maschere di colonna.

RPO

Vedi obiettivo del punto di ripristino.

RTO

Vedi l'obiettivo del tempo di ripristino.

runbook

Un insieme di procedure manuali o automatizzate necessarie per eseguire un'attività specifica. In genere sono progettati per semplificare operazioni o procedure ripetitive con tassi di errore elevati.

S

SAML2.0

Uno standard aperto utilizzato da molti gestori dell'identità IdPs digitale (). Questa funzionalità consente l'accesso federato Single Sign-On (SSO), grazie al quale gli utenti possono accedere alla AWS Management Console o eseguire chiamate AWS API alle operazioni. In questo modo non è necessario creare un account utente IAM per tutti gli utenti nell'organizzazione. Per ulteriori informazioni sulla federazione SAML basata sulla versione 2.0, consulta Informazioni sulla federazione SAML basata sulla versione 2.0 nella IAM documentazione.

SCADA

Vedi controllo di supervisione e acquisizione dati.

SCP

Vedi la politica di controllo del servizio.

Secret

In AWS Secrets Manager, informazioni riservate o riservate, come una password o le credenziali utente, archiviate in forma crittografata. È costituito dal valore segreto e dai relativi metadati. Il valore segreto può essere binario, una stringa singola o più stringhe. Per ulteriori informazioni, consulta What is Secrets Manager? nella documentazione di Secrets Manager.

controllo di sicurezza

Un guardrail tecnico o amministrativo che impedisce, rileva o riduce la capacità di un autore di minacce di sfruttare una vulnerabilità di sicurezza. Esistono quattro tipi principali di controlli di sicurezza: preventivi, investigativi, reattivi e proattivi.

rafforzamento della sicurezza

Il processo di riduzione della superficie di attacco per renderla più resistente agli attacchi. Può includere azioni come la rimozione di risorse che non sono più necessarie, l'implementazione di best practice di sicurezza che prevedono la concessione del privilegio minimo o la disattivazione di funzionalità non necessarie nei file di configurazione.

sistema di gestione delle informazioni e degli eventi di sicurezza (SIEM)

Strumenti e servizi che combinano sistemi di gestione delle informazioni di sicurezza (SIM) e sistemi di gestione degli eventi di sicurezza (SEM). Un SIEM sistema raccoglie, monitora

e analizza i dati da server, reti, dispositivi e altre fonti per rilevare minacce e violazioni della sicurezza e generare avvisi.

automazione della risposta di sicurezza

Un'azione predefinita e programmata progettata per rispondere o porre rimedio automaticamente a un evento di sicurezza. Queste automazioni fungono da controlli di sicurezza investigativi o reattivi che aiutano a implementare le migliori pratiche di sicurezza. AWS Esempi di azioni di risposta automatizzate includono la modifica di un gruppo di VPC sicurezza, l'applicazione di patch a un'EC2istanza Amazon o la rotazione delle credenziali.

Crittografia lato server

Crittografia dei dati a destinazione, da parte del Servizio AWS che li riceve.

policy di controllo dei servizi (SCP)

Una policy che fornisce il controllo centralizzato sulle autorizzazioni per tutti gli account di un'organizzazione in AWS Organizations. SCPsdefinisci i guardrail o imposta i limiti alle azioni che un amministratore può delegare a utenti o ruoli. Puoi utilizzare SCPs come elenchi consentiti o elenchi di rifiuto, per specificare quali servizi o azioni sono consentiti o proibiti. Per ulteriori informazioni, consulta Policy di sicurezza dei servizi nella AWS Organizations documentazione di.

endpoint del servizio

Il punto URL di ingresso per un Servizio AWS. Puoi utilizzare l'endpoint per connetterti a livello di programmazione al servizio di destinazione. Per ulteriori informazioni, consulta Endpoint del Servizio AWS nei Riferimenti generali di AWS.

accordo sul livello di servizio () SLA

Un accordo che chiarisce ciò che un team IT promette di offrire ai propri clienti, ad esempio l'operatività e le prestazioni del servizio.

indicatore del livello di servizio () SLI

Misurazione di un aspetto prestazionale di un servizio, ad esempio il tasso di errore, la disponibilità o la velocità effettiva.

obiettivo a livello di servizio () SLO

Una metrica target che rappresenta lo stato di un servizio, misurato da un indicatore del livello di servizio.

Modello di responsabilità condivisa

Un modello che descrive la responsabilità condivisa AWS per la sicurezza e la conformità del cloud. AWS è responsabile della sicurezza del cloud, mentre tu sei responsabile della sicurezza nel cloud. Per ulteriori informazioni, consulta Modello di responsabilità condivisa.

SIEM

Vedi sistema di gestione delle informazioni di sicurezza e degli eventi di sicurezza.

singolo punto di errore (SPOF)

Un guasto in un singolo componente critico di un'applicazione che può disturbare il sistema.

SLA

Vedi accordo sul livello di servizio.

SLI

Vedi l'indicatore del livello di servizio.

SLO

Vedi l'obiettivo del livello di servizio.

split-and-seed modello

Un modello per dimensionare e accelerare i progetti di modernizzazione. Man mano che vengono definite nuove funzionalità e versioni dei prodotti, il team principale si divide per creare nuovi team di prodotto. Questo aiuta a dimensionare le capacità e i servizi dell'organizzazione, migliora la produttività degli sviluppatori e supporta una rapida innovazione. Per ulteriori informazioni, consulta la sezione Approccio graduale alla modernizzazione delle applicazioni nel. Cloud AWS

SPOF

Vedere Single Point of Failure.

schema stellare

Una struttura organizzativa di database che utilizza un'unica tabella dei fatti di grandi dimensioni per archiviare i dati transazionali o misurati e utilizza una o più tabelle dimensionali più piccole per memorizzare gli attributi dei dati. Questa struttura è progettata per l'uso in un data warehouse o per scopi di business intelligence.

modello del fico strangolatore

Un approccio alla modernizzazione dei sistemi monolitici mediante la riscrittura e la sostituzione incrementali delle funzionalità del sistema fino alla disattivazione del sistema legacy. Questo modello utilizza l'analogia di una pianta di fico che cresce fino a diventare un albero robusto e alla fine annienta e sostituisce il suo ospite. Il modello è stato introdotto da Martin Fowler come metodo per gestire il rischio durante la riscrittura di sistemi monolitici. Per un esempio di come applicare questo modello, consulta Modernizzazione della versione legacy di MicrosoftASP.NET(ASMX) servizi web in modo incrementale utilizzando contenitori e Amazon API Gateway.

sottorete

Un intervallo di indirizzi IP nel tuoVPC. Una sottorete deve risiedere in una singola zona di disponibilità.

controllo di supervisione e acquisizione dati () SCADA

Nella produzione, un sistema che utilizza hardware e software per monitorare gli asset fisici e le operazioni di produzione.

crittografia simmetrica

Un algoritmo di crittografia che utilizza la stessa chiave per crittografare e decrittografare i dati.

test sintetici

Test di un sistema in modo da simulare le interazioni degli utenti per rilevare potenziali problemi o monitorare le prestazioni. Puoi usare Amazon CloudWatch Synthetics per creare questi test.

T

tags

Coppie chiave-valore che fungono da metadati per l'organizzazione delle risorse. AWS Con i tag è possibile a gestire, identificare, organizzare, cercare e filtrare le risorse. Per ulteriori informazioni, consulta Tagging delle risorse AWS.

variabile di destinazione

Il valore che stai cercando di prevedere nel machine learning supervisionato. Questo è indicato anche come variabile di risultato. Ad esempio, in un ambiente di produzione la variabile di destinazione potrebbe essere un difetto del prodotto.

elenco di attività

Uno strumento che viene utilizzato per tenere traccia dei progressi tramite un runbook. Un elenco di attività contiene una panoramica del runbook e un elenco di attività generali da completare. Per ogni attività generale, include la quantità stimata di tempo richiesta, il proprietario e lo stato di avanzamento.

Ambiente di test

Vedi ambiente.

training

Fornire dati da cui trarre ispirazione dal modello di machine learning. I dati di training devono contenere la risposta corretta. L'algoritmo di apprendimento trova nei dati di addestramento i pattern che mappano gli attributi dei dati di input al target (la risposta che si desidera prevedere). Produce un modello di ML che acquisisce questi modelli. Puoi quindi utilizzare il modello di ML per creare previsioni su nuovi dati di cui non si conosce il target.

Transit Gateway

Un hub di transito di rete che è possibile utilizzare per collegare VPCs le reti locali. Per ulteriori informazioni, consulta Che cos'è un Transit Gateway di transito? nella AWS Transit Gateway documentazione di.

flusso di lavoro basato su trunk

Un approccio in cui gli sviluppatori creano e testano le funzionalità localmente in un ramo di funzionalità e quindi uniscono tali modifiche al ramo principale. Il ramo principale viene quindi integrato negli ambienti di sviluppo, preproduzione e produzione, in sequenza.

Accesso attendibile

La concessione di autorizzazioni a un servizio specificato dall'utente per eseguire attività all'interno dell'organizzazione in AWS Organizations e nei relativi account per conto dell'utente. Il servizio attendibile crea un ruolo collegato al servizio in ogni account, quando tale ruolo è necessario, per eseguire attività di gestione per conto dell'utente. Per ulteriori informazioni, consulta Utilizzo AWS Organizations con altri AWS servizi nella AWS Organizations documentazione.

regolazione

Modificare alcuni aspetti del processo di training per migliorare la precisione del modello di ML. Ad esempio, puoi addestrare il modello di ML generando un set di etichette, aggiungendo etichette e quindi ripetendo questi passaggi più volte con impostazioni diverse per ottimizzare il modello.

team da due pizze

Una piccola DevOps squadra che puoi sfamare con due pizze. Un team composto da due persone garantisce la migliore opportunità possibile di collaborazione nello sviluppo del software.

U

incertezza

Un concetto che si riferisce a informazioni imprecise, incomplete o sconosciute che possono minare l'affidabilità dei modelli di machine learning predittivi. Esistono due tipi di incertezza: l'incertezza epistemica, che è causata da dati limitati e incompleti, mentre l'incertezza aleatoria è causata dal rumore e dalla casualità insiti nei dati. Per ulteriori informazioni, consulta la guida Quantificazione dell'incertezza nei sistemi di deep learning.

tasks indifferenziati

Conosciuto anche come sollevamento di carichi pesanti, è un lavoro necessario per creare e far funzionare un'applicazione, ma che non apporta valore diretto all'utente finale né offre vantaggi competitivi. Esempi di attività indifferenziate includono l'approvvigionamento, la manutenzione e la pianificazione della capacità.

ambienti superiori

Vedi ambiente.

V

vacuum

Un'operazione di manutenzione del database che prevede la pulizia dopo aggiornamenti incrementali per recuperare lo spazio di archiviazione e migliorare le prestazioni.

controllo delle versioni

Processi e strumenti che tengono traccia delle modifiche, ad esempio le modifiche al codice di origine in un repository.

VPCscrutando

Una connessione tra due VPCs che consente di instradare il traffico tramite indirizzi IP privati. Per ulteriori informazioni, consulta Che cos'è il VPC peering? nella VPC documentazione di Amazon.

vulnerabilità

Un difetto software o hardware che compromette la sicurezza del sistema.

W

cache calda

Una cache del buffer che contiene dati correnti e pertinenti a cui si accede frequentemente. L'istanza di database può leggere dalla cache del buffer, il che richiede meno tempo rispetto alla lettura dalla memoria dal disco principale.

dati caldi

Dati a cui si accede raramente. Quando si eseguono interrogazioni di questo tipo di dati, in genere sono accettabili interrogazioni moderatamente lente.

funzione finestra

Una SQL funzione che esegue un calcolo su un gruppo di righe che si riferiscono in qualche modo al record corrente. Le funzioni della finestra sono utili per l'elaborazione di attività, come il calcolo di una media mobile o l'accesso al valore delle righe in base alla posizione relativa della riga corrente.

Carico di lavoro

Una raccolta di risorse e codice che fornisce valore aziendale, ad esempio un'applicazione rivolta ai clienti o un processo back-end.

flusso di lavoro

Gruppi funzionali in un progetto di migrazione responsabili di una serie specifica di attività. Ogni flusso di lavoro è indipendente ma supporta gli altri flussi di lavoro del progetto. Ad esempio, il flusso di lavoro del portfolio è responsabile della definizione delle priorità delle applicazioni, della pianificazione delle ondate e della raccolta dei metadati di migrazione. Il flusso di lavoro del portfolio fornisce queste risorse al flusso di lavoro di migrazione, che quindi migra i server e le applicazioni.

WORM

Vedi write once, read many.

WQF

Vedi AWSWorkload Qualification Framework.

scrivi una volta, leggi molte () WORM

Un modello di archiviazione che scrive i dati una sola volta e ne impedisce l'eliminazione o la modifica. Gli utenti autorizzati possono leggere i dati tutte le volte che è necessario, ma non possono modificarli. Questa infrastruttura di archiviazione dei dati è considerata immutabile.

Z

exploit zero-day

Un attacco, in genere malware, che sfrutta una vulnerabilità zero-day.

vulnerabilità zero-day

Un difetto o una vulnerabilità assoluta in un sistema di produzione. Gli autori delle minacce possono utilizzare questo tipo di vulnerabilità per attaccare il sistema. Gli sviluppatori vengono spesso a conoscenza della vulnerabilità causata dall'attacco.

applicazione zombie

Un'applicazione che prevede un utilizzo CPU della memoria inferiore al 5%. In un progetto di migrazione, è normale ritirare queste applicazioni.

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.