COME SFRUTTARE AL MASSIMO I MICROSERVIZI

Microservices | Software as a Service (SaaS)

beSharp | 29 Novembre 2019

In quest'ultimo articolo della nostra analisi in tre parti (trovate qui la prima parte e qui la seconda parte) su come scomporre un'applicazione monolitica, sfrutteremo alcune intuizioni su come un microservizio può essere creato a partire da un'applicazione esistente, cosa possiamo fare per risolvere problemi legati al codice legacy ad alta tossicità e in generale quali regole possono essere applicate per decidere quando migrare a microservizi. Cominciamo!

COME SCRIVERE UN MICROSERVIZIO

Quando vogliamo creare un microservizio abbiamo davanti due scelte:

1. **Trasferire il codice legacy sul microservizio.**
2. **Riscrivere il codice da zero**.

TRASFERIRE IL CODICE

In un primo momento trasferire direttamente il codice può sembrare una soluzione logica, soprattutto perché il team di sviluppo ha tendenzialmente un **atteggiamento cognitivo favorevole** verso il proprio codice MA può nascondere alcuni aspetti critici dai **costi elevati** e dal **basso valore intellettuale**:

- C'è una **grande quantità di codice boilerplate necessario alle dipendenze d'ambiente**, come l'accesso alla configurazione dell'applicazione a runtime, l'accesso agli archivi di dati, il caching, ed è solitamente sviluppato con framework obsoleti. Questo codice può essere tranquillamente riscritto perché il microservizio sicuramente affronterà le cose in modo diverso in un ambiente di sviluppo a sua volta diverso.
- Le funzionalità esistenti possono essere costruite intorno a concetti di dominio incerti. Ciò comporta una fase di refactoring consistente.
- Codice legacy che ha subito molte iterazioni di cambiamento potrebbe avere un'elevata **tossicità** e un **basso valore di riutilizzo**.

Quindi, per riassumere, verificate sempre se il codice che volete riutilizzare ha un **alto valore di complessità intellettuale** e **bassa tossicità** in termini di accoppiamento con altri servizi e debito tecnico**: questo è un buon candidato** per l'estrazione e il riutilizzo. Un esempio può essere un complesso **sistema di raccomandazioni** per un cliente di un e-commerce con molte interazioni, personalizzazioni, algoritmi di apprendimento automatico e così via.

SCRIVERE IL CODICE DA ZERO

Scrivere codice da zero per un microservizio è un ottimo approccio nel senso che costringe il team di Sviluppo e il team di Business a rivedere la logica di base della funzionalità che si vuole riscrivere permettendo:

- Un approccio al servizio con una comprensione del settore più ampia e con molta più esperienza, risultando molto probabilmente in una soluzione più piccola, semplice e ordinata.
- La ricerca di un **aggiornamento tecnologico**, implementando il nuovo servizio con un linguaggio di programmazione e uno stack tecnologico più moderno e sicuramente più adatto.

Questo approccio può essere vantaggioso anche nel senso che il microservizio sarà probabilmente **sviluppato più rapidamente**. Un esempio può essere rappresentato dalle **operazioni CRUD** perché non contengono proprietà intellettuali e possono dipendere tranquillamente da nuove tecnologie e framework.

COME APPROCCIARSI A CODICE CON ACCOPPIAMENTO FORTE ALL'INTERNO DEL MONOLITE

Dopo aver riscritto la maggior parte della vostra applicazione mediante microservizi, dovrete affrontare il resto del codice legacy per il quale la conoscenza del dominio è incerta. In questi casi, trasferire il codice e anche **la base dati** può essere molto difficile, anche se si consiglia di affrontare il più velocemente possibile questa parte della vostra infrastruttura, perché il disaccoppiamento dello strato di base dati è di gran lunga la parte più importante del vostro lavoro di migrazione verso un ecosistema a microservizi.

Possiamo tranquillamente dire che se i dati sono in qualche modo accoppiati ad altri servizi non si è sviluppato un vero e proprio microservizio.

Una tecnica che può aiutarci in questo processo si chiama **Reification,** che è un processo per ridefinire i confini del contesto attraverso la decomposizione di parti incerte del dominio logico decostruendo quindi il codice legacy in strutture ben separate e più semplici che derivano anche da modelli di dati più definiti.

È inoltre possibile utilizzare strumenti di **analisi del codice strutturale e di dipendenza come Structure101** per identificare le criticità di accoppiamento e fattori di vincolo nel codice del monolite.

COME AWS PUò AIUTARCI CON I SUOI SERVIZI

AWS può davvero aiutarci fornendoci gli strumenti per una facile transizione verso un approccio a microservizi, almeno dal punto di vista infrastrutturale.

A LIVELLO DI BUILDING, TESTING E DEPLOYING:

CodeCommit, CodeBuild e CodeDeploy.

MONITORING E DEBUGGING:

CloudWatch, ElasticSearch with Kibana integration, X-Ray.

DEPLOY E RECOVERY:

CloudFormation.

LOGIC:

AWS Lambda Functions e Lambda Layers, ECS e AWS Step Functions che coordinano i workers.

PERCHÉ EVITARE INVECE L'APPROCCIO A MICROSERVIZI

Abbiamo parlato molto di quanto siano efficaci i microservizi e di quanti benefici si avrebbero implementando questo paradigma, ma ci sono alcune situazioni in cui si può riconsiderare l'idea di utilizzare questo approccio.

Ad esempio, se avete un piccolo progetto che probabilmente non si evolverà in futuro, lo sforzo di disaccoppiamento potrebbe essere troppo grande, in quanto fare microservizi non è, come abbiamo visto, un compito semplice.

Se l'applicazione svolge una **funzione mission-critical**, come il **mantenimento di un database legacy insostituibile**, è probabile che non sarete in grado di sostituirlo completamente in pochi anni, in altre parole il vostro team strategico non può permettersi **una strategia a lungo termine** finalizzata alla ricostruzione della base dati.

Alcune applicazioni per **loro natura** richiedono una **stretta integrazione** tra i **singoli componenti e servizi**. Questo è spesso vero, ad esempio, per applicazioni che **elaborano rapidi flussi di dati in tempo reale**. Eventuali livelli aggiuntivi di comunicazione tra i servizi **possono rallentare l'elaborazione in tempo reale**.

CONCLUSIONI

Per riassumere quanto detto fino a questo punto, riprendiamo i punti chiave più importanti della nostra discussione:

- Comprendere il vostro obiettivo tecnologico nel modo più chiaro possibile per rendere la transizione il più semplice e coerente possibile.
- Gli step di transizione devono essere più **atomici** possibile e cerchiamo di utilizzare sempre l'**approccio a tre passi** della Tecnica **Strangler**, specialmente per **decommissionare** il codice

obsoleto una volta che non viene più utilizzato.

- **Definire quali microservizi implementare** per **primi** per iniziare a lavorare con il giusto atteggiamento e vedere i risultati del vostro lavoro il più presto possibile. Anche questo approccio è molto buono per iniziare a **sviluppare esperienza nel processo di migrazione** prima di attaccare parti più difficili della vostra code base.
- Utilizzare la Tecnica **Reify** per aiutarvi a ridefinire le parti di codice più oscure.
- **Cercare** sempre **di isolare parti significative della vostra base dati** in modo da suddividerla in database indipendenti, uno per ogni microservizio (se il microservizio ne ha bisogno, può ovviamente essere stateless). Questo passo è molto importante.
- Verificare sempre che il **microservizio rispetti la proprietà atomica di isolamento** sia a livello di infrastruttura che di logica.
- Non abbiate paura di fare un **uso estensivo di strumenti come Structure101 o in generale dei servizi AWS** per aiutarvi nel processo di migrazione.
- Cercare di **riscrivere i vostri microservizi da zero il più possibile** (a parte il codice ad **alto valore intellettuale**), di coinvolgere maggiormente il vostro team strategico e di definire un approccio migliore per risolvere un determinato problema. Riscrivere il codice evita anche di portare con sé un eventuale debito tecnico.
- Ricordavi che per iniziare il vostro viaggio di migrazione verso i microservizi il vostro team di Sviluppo e in generale la vostra azienda deve aderire alla DevOps Culture e deve essere in possesso di una certa esperienza preliminare.
- Infine ricordate che ci sono alcune situazioni in cui cercare di migrare verso un approccio a microservizi potrebbe non essere la soluzione ideale per casi come le applicazioni in tempo reale in cui la comunicazione tra livelli deve essere il più veloce possibile.

Con questo riassunto concludiamo il nostro lungo viaggio su come sia possibile scomporre una grande applicazione monolitica in diversi microservizi al fine di mantenerla meglio, svilupparla, implementarla ed eventualmente recuperarla molto velocemente, concentrandosi così molto di più, sul suo valore aziendale.

Ci auguriamo che siate soddisfatti. Contattateci per feedback, domande o spunti. Saremo felici di fare due chiacchiera insieme a voi 😉

<< Leggi la seconda parte | << Leggi la prima parte

beSharp

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all'avanguardia a realizzare progetti innovativi in campo IT.

Get in touch

beSharp.it
proud2becloud@besharp.it