SAAS ENABLEMENT: DA MONO-TENANT A MULTI-TENANT.

Software as a Service (SaaS)

beSharp | 13 Dicembre 2019

Bentornati nella nostra serie di articoli dedicata al tema SaaS Enablement.

Dopo il nostro deep-dive sulla scomposizione di un monolite in microservizi, siamo pronti per addentrarci in un altro aspetto del modello SaaS: l'approccio multi-tenant.

Perché è così importante quando si parla di Software as a Services? E come implementarlo efficacemente?

Introduzione

Oggi sempre di più, pattern come quello SaaS sono fondamentali per determinare il successo di un prodotto.

La semplice adozione di questo modello, però, non è abbastanza.

Distribuire un prodotto pensato e progettato come mono-tenant secondo il SaaS Delivery Model è uno degli errori più comuni.

Infatti, installare una copia del software per ogni cliente su un'infrastruttura dedicata potrebbe tecnicamente funzionare, ma finirebbe per diventare presto insostenibile sia dal punto di vista gestionale, che finanziario.

Costruire efficacemente una soluzione SaaS significa soprattutto garantire la customer satisfaction su qualsiasi scala.

Per aumentare il profitto continuando a garantire, allo stesso tempo, la miglior user experience possibile è necessario poter adattare potenza di calcolo e spesa al traffico effettivo e al numero reale degli utenti. Raggiungere questo scenario-tipo è possibile sfruttando la condivisione dell'infrastruttura tra tutti gli utenti.

Mono-tenant VS multi-tenant: cosa cambia?

Un software mono-tenant funziona come un client individuale. Per ciascun cliente esistono un database indipendente e una istanza del software dedicata. Ogni elemento è dedicato esclusivamente ad un solo customer.

Con un software multi-tenant, al contrario, l'istanza del software in funzione su una piattaforma SaaS serve contemporaneamente tutti gli utenti. Ogni cliente si serve dello stesso software dell'applicazione e attinge dallo stesso database. La condivisione resta comunque a livello infrastrutturale e non implica la condivisione dei dati del Tenant che restano isolati e invisibili agli altri Tenant.

Perché optare per un modello multi-tenant

Vediamo ora alcuni esempi di vantaggi di un approccio multi-tenant sull'approccio mono-tenant:

- Maintenance e update: col modello multi-tenant la gestione degli aggiornamenti è notevolmente semplificata. Aggiornando il software di base, gli utenti potranno disporre in automatico dell'ultima versione.
- Costi: dovendo mantenere una sola istanza dell'infrastruttura, sarà possibile contenere i costi e sfruttare al massimo i vantaggi derivanti da un'economia di scala.
- Scaling: scale up e scale down saranno automatici, immediati e perfettamente aderenti al traffico effettivo delle tue applicazioni.

Come migrare verso il modello Multi-tenant

Prima di cominciare a modificare il codice applicativo, è necessario assicurarsi di poter fare su di esso cambiamenti senza che l'applicazione (o parte di essa) smetta di funzionare.

Il primo passo, quindi, è dotarsi di una suite di test; I test automatici vi aiuteranno a procedere in modo semplice e sicuro.

Il secondo step consiste invece in una serie di valutazioni sulle attività necessarie al re-engineering del data model dell'applicazione. Solo dopo si potrà procedere con le modifiche a data model e relativo codice dell'applicazione.

Passiamo ora al terzo step: la modifica del codice. Ecco Alcuni consigli:

- ogni risorsa dovrà mantenere una referenza con il cliente, utile per poter recuperare le giuste informazioni.
- Creiamo un modello per storare le informazioni di ciascun cliente e poi linkiamo tutte le risorse a quel modello. Per evitare che i dati di ciascun cliente possano essere recuperate dagli altri clienti, è consigliabile archiviare il customer ID all'interno del token di autenticazione per filtrare i dati all'interno delle API in modo sicuro.
- Durante il processo di refactoring del data model, teniamo presente che, al crescere dei dati, la velocità di esecuzione delle query potrebbe diminuire in modo drastico. Per evitare questo è consigliabile organizzare i dati utilizzando indici sulle colonne e partizioni.

- utilizziamo table partition and column indexing. Un altro trucco consiste nell'utilizzo di una cache per i dati che vengono richiesti in modo intensivo (es. database Key-Value NoSQL).

Come abbiamo descritto nel nostro blog post precedente, se il nostro punto di partenza è un'applicazione monolitica, la prima cosa da fare per facilitare il lavoro è scomporla in microservizi dividendo, per prima cosa, backend e frontend.

Nella trasformazione in multi-tenant è importante tenere a mente che, per poter scalare nel modo più efficiente possibile, il layer applicativo dovrebbe seguire il paradigma Stateless.

Cosa significa? Restate sintonizzati per scoprirlo: nel nostro prossimo articolo parleremo proprio di questo!

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