SAAS ENABLEMENT: FROM MONO-TENANT TO MULTI-TENANT

Software as a Service (SaaS)

beSharp | 13 December 2019

Welcome back to our series of blog posts dedicated to SaaS Enablement. After our deep-dive about refactoring a monolith into microservices, here we'll discuss why a SaaS model definitely requires a multi-tenant approach and which are the best practices to implement it efficiently at scale.

Introduction

Today more than ever, patterns like SaaS are essential for the growth of successful products.

However, the mere adoption of SaaS is not enough.

Attempting to sell a product as SaaS while it is designed to be mono-tenant is one of the most common mistakes, usually done by installing a new copy of the software for each customer, in a dedicated piece of infrastructure.

Such an approach may technically work, but it quickly becomes financially and technically unsustainable.

Successfully build a SaaS application means you must be ready to guarantee customer satisfaction on any scale and at any cost.

To boost your profit while bestowing an outstanding user experience, you must be able to adjust your computing resources (and your spending) according to the traffic and the number of customers. And this results in greater efficiency at scale only if all your users share the same infrastructure.

It should be clear at this point that to successfully grow a SaaS application you have to migrate to a multi-tenant approach

Difference between Mono-Tenant and Multi-Tenant

Mono-Tenant software is a program that serves as an individual client. With a single tenancy, there is an independent database and instance of the software for each customer. Essentially, there is no sharing happening with this option.

Multi-Tenant software is the environment in which a single instance of the software that is running on a SaaS platform serves multiple clients or tenants. Each customer shares the software application and a single database. Each Tenant's data is isolated and remains invisible to other Tenants.

Why use a Multi-Tenant model instead of a Mono-Tenant model

There are a lot of benefits using a Multi-Tenant model instead of a Mono-Tenant model. For example:

- Maintenance and Update: Using the Multi-Tenant model requires only to update the base software. All customers automatically have the last version.
- Cost: You have to maintain only one instance of the application that requires less infrastructure instead of multiple, one for each customer. You can also do a better economy of scale.
- Scaling: It is simpler to scale up and down based on the traffic of your application.

How to migrate to Multi-Tenant Model

Before you can start editing your application code, you want to ensure you can make changes without breaking your functionalities.

Therefore, we strongly recommend you to have and maintain an automatic test suite. Automated tests help you to change the application model more easily and securely.

The first step you should consider to migrate to a Multi-Tenant model is to re-engineer the data model of your application. Start from analyzing that. After that, you can start to change the data model and the related code of your application. Remember that every resource should have a reference to the customer to permit you to retrieve the correct information. Create a model for store the customer info and then link all the resources to that model. To prevent a customer from retrieving data from another customer, you can store the customer id into the authentication token, to filter the customer data inside your APIs in a secure way.

During the data model re-engineering, think that the cardinality of your data could grow a lot making your queries very slow. To prevent this, you can organize your data using table partition and column indexing. Another trick is to use a cache database for data that is requested intensively. For example, you can use a Key-Value database or a NoSQL database.

As we already mentioned in our previous post, If you have a monolith application, during the code rewriting, begin to break it into microservices, this helps you to maintain and scale without considerable effort. When approaching this, consider starting from dividing Backend application from Frontend application. After that, you can convert your monolith application functionality into microservices going to separate and convert them one at a time until you have been all the features migrated.

One of the most important things about Multi-tenant transformation is to keep in mind that, to be able to scale seamlessly, your application layer should definitely be Stateless. But this will be covered in depth by our next blog post about this topic.

Stay tuned!

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