IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 8 Issue 5, May 2021
ISSN (Online) 2348 – 7968 | Impact Factor (2020) – 6.72
www.ijiset.com

A Brief Review of Micro Services and SAAS with Multi-Tenancy

Dr. Shruti Makarand Kanade

Dr. Babasaheb Amberdkar Marathwada University, Auranagabad. MH

Address: Plot no 30, Builder's Housing Society, Nandanvan Colony, Aurangabad - 431001

Email: shrutikanade@gmail.com Contact Number: 9921144299

Abstract: Off the shelf enterprise software such as ERP, CRM, and SCM is now available as a Cloud based multi-tenant SAAS model. Enabling customer-specific business advantage for every different tenant gives an added value to the vendor. For this, software vendors need explicit, novel customized solutions. A challenge when targeting this multi-tenant approach, is to ensure that the performance and resource consumption of one tenant does not adversely affect other tenants. An optimal degree of tenant isolation is the key to this challenge.

Micro-service architecture is a variant of the SOA structural style. It arranges an application as a collection loosely coupled services. In a micro-services architecture, services are fine-grained and the protocols are lightweight.

In this paper, a brief overview of micro-service architecture is given. Then, the need and feasibility of customization of SAAS application using MSA is scrutinized. The benefits and conditions for this customization are taken into account. There are mainly three types of approaches for customizing multi-tenant SaaS based on MSA, that are, Intrusive, Non-Intrusive and Database Customization.

The Database customization approach is studied extensively. The conditions and trade-offs for the database customized approach are analyzed. Shared and non-shared databases are analyzed using isolation and data storing approaches.

Keywords: SAAS Applications, Multi-tenancy, Micro-services, Database customization approach.

INTRODUCTION

Cloud Computing is defined by Badger et al. 2008 as "a model for enabling convenient, on-demand network access to a shared pool of configurable resources like networks, servers, storage, applications and services, that can be rapidly provisioned and released with minimum management effort or service provider interaction ". Many of us will see this paradigm shift in IT. The exciting features for end users/ lay man are the reduction in capital costs, agility offered by on-demand IT services and computing power. Analyzing the pros-cons of this ubiquitous cloud computing is very essential. Critical challenges include regulatory, security and privacy issues in cloud computing.

NIST has identified three basic types of cloud services offerings. These models are:

1. SaaS (Software as a Service): This offers renting applications functionality from a service provider rather than the user buying, installing and running the software.
2. PaaS (Platform as a Service): It provides a platform on cloud to develop and execute applications.
3. IaaS (Infrastructure as a Service): Here, vendors offer computing power and storage space as required by the customer.

MULTI-TENANCY: THE ONLY ARCHITECTURE FOR TRUE SAAS

As stated by Larry Aiken: "The true multi-tenant application is the de-facto architecture for highest level of SaaS efficiency. Using this approach, all application infrastructure is shared with a single logical instance of database. This makes it possible that the business logic is also leveraged across all users".

To summarize, the basic benefits of multi-tenant SaaS are:

1. Easy Upgrades: Developers are able to update a single, central application. The changes are instantly available to all users. The process of hosting a new cloud and application for a new customer is easily done.
2. Easy Customization: An additional layer for customization can be allowed, while keeping existing functionality constant for all users and can be given to new customers.
3. Ongoing Cost Savings: No dedicated set of resources must be configured. The opportunity to save money becomes greater as application scales up. This reduces cost of doing business.

MICRO-SERVICES --- DEFINITION AND BENEFITS

Some defining characteristics of micro-services are:

- Services in a Micro-Service Architecture (MSA) are often processes that communicate over a network to fulfil goals using protocols like HTTP
- Services are organized around business capabilities.
- Services can be implemented using different programming languages, databases, networks and software environments, depending on what fits best.
- Services are small in size, message enabled, bounded by contexts. They are autonomously developed, deployed, decentralized and also built and released with automated processes.

So, micro-service is a self-contained piece of business functionality with clear interfaces which implement a layered architecture.

It is common for MSA to be adopted for cloud applications as well as server-less computing and applications using light-weight container deployment.

It is to be figured out how big the individual micro-service needs to be. The right answer to this depends on business and organizational context. It is a bad practice to make the service too small. This is because runtime overhead and operational complexity outgain the benefits of MSA.

CUSTOMIZATION IN MICRO-SERVICES BASED SAAS

Here, the need for customization model in SaaS is analyzed. Enterprise software vendors are moving from single-tenant-on premise applications to multi-tenant, cloud based SaaS. Due to this, the traditional way of on-premises customization faces new challenges. This is due to the fact that customer does not have exclusive control on the application.

Three basics are essential. They are as follows:

1. Isolation: One of the tenant's customization must not impact other tenants.
2. Performance: Customization must not impact overall performance of SaaS
3. Cost: Customization of multi-tenant SaaS should ensure the economy of scale brought about by SaaS model

BENEFITS OF MSA FOR CUSTOMIZING SAAS

Micro-services for customizing purposes can be packaged and deployed in isolation from main product. The development and deployment of customized micro-services is independent. Inclusion of continuous integration and delivery practices is possible. Launching to the market can be done in lesser time. Customers can choose the technology for customization. Each micro-service can be operated independently which includes upgrades and scaling.

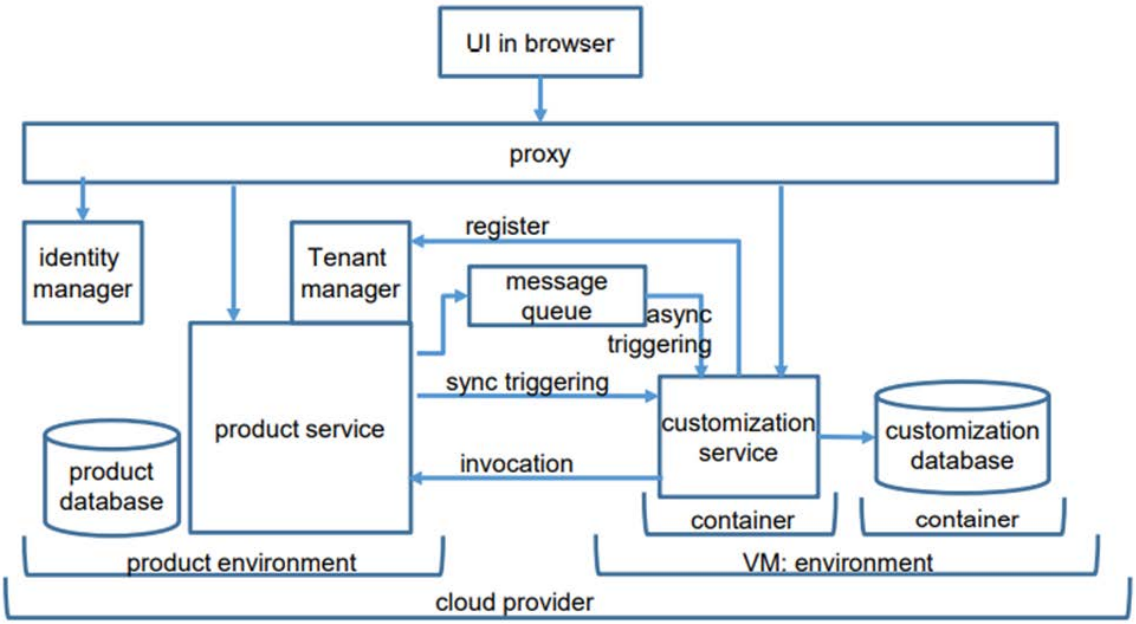

FIGURE1. Micro-services Layered Architecture

Also, there are certain principles for writing customized micro-service. First of all, each customized micro-service must be independent, light weight which runs in its own environment. A single micro-service will serve a single tenant only. Each micro-service has its own database. There should be no sharing of database or files between the product application and the customized micro-service. Product and customized micro-service are compulsorily hosted on same infra-structure. This micro-service can be assessed synchronously via REST API. And asynchronously using queue.

There are mainly three types of approaches for customizing multi-tenant SaaS based on MSA.

1. Intrusive
2. Non-Intrusive
3. Database Customization.

INTRUSIVE APPROACH

This approach requires that the main body of customization code runs as a separated micro-service. It is isolated from the main service of the main product. Here, specific parts of the customization code are sent back to the main product. This code is dynamically complied and executed within the execution context of main service.

A customization for a particular tenant is invariably running as a standalone service. It is dynamically registered to the product service for this single tenant. At runtime, the customized micro-service are triggered by the product service. This happens when the later reaches a registered extension point. The customization micro-service are hosted by the same vendor cloud which registers the product service. Deep customization is possible. This is because the internal main product code APIs are accessible to the customized code. But it is not secure and has technical limitations

NON-INTRUSIVE APPROACH

This approach prescribes that the entire customization code runs in a separate micro-service, isolated from the main service, however the customization code can call multiple product APIs for executing the logic.

Each Customization for a tenant is running as a stand-alone micro-service and dynamically registered to the main product service for this tenant. These customized micro-service must be registered before-hand with the product's tenant manager. The APIs of the customized micro-service are available for authorized access via API Gateway. At runtime, the main product service triggers the corresponding customization for that tenant by calling REST APIs of the tenant's customization micro-service. More the extension points the product service has, the more customization it supports. This customization micro-service can call multiple product APIs. The key point is that authorized access is enabled for the tenant customized micro-service to main product. So, tenant's customization micro-service has access to necessary execution context of main application if needed. It secures communication between product application and customization Micro-service. No technology limitation as any framework supporting REST API is required. But there is no deep customization.

DATABASE APPROACH

Let's have a look at the Database per service pattern. Let us suppose that we need a MSA for an online stores application. Data is to be saved and persisted in the database. As an example, it can be said that order service stores information about orders, vendor service stores data about vendors and so on. We need to define database architecture in a micro-service application.

There is nothing beneficial from MSA if all the services share the same database tables. This is because the services are getting tightly coupled. If a single database table changes, all services will have to change. The reason for using MSA is to reduce dependencies.

In general, a micro-service should be responsible for its own data. But that is a perfect world scenario. In a real world scenario, some or other services are highly related to each other. For example, customer address must be shared by Shipping Details service and Shopping Checkout service. If the Checkout service queries the shopping details directly, loose coupling between is broken. Otherwise a shared database can be introduced. There will always be some compromise on the architecture. Having a shared database for micro-services that take care of similar business logic is an important point. Here, Shipping Details service and shopping checkout service can share a database. But Store Details would have a separate database.

Some business transactions must enforce conditions that involve multiple micro-service. Other business transactions must update data owned by multiple micro-services. Some queries must join data owned by multiple services. Database need to be replicated and shared and have different data storage requirements.

The solution can be:

Use a single database that is shared by multiple micro-services. Every service accesses data owned by other services using local ACID transactions.

For example, the order service can use the ACID transaction to ensure that a new order will not violate customer's credit limit.

```
BEGIN TRANSACTION
…
SELECT ORDER_TOTAL
 FROM ORDERS WHERE CUSTOMER_ID = ?
…
SELECT CREDIT_LIMIT
FROM CUSTOMERS WHERE CUSTOMER_ID = ?
…
INSERT INTO ORDERS …
…
COMMIT TRANSACTION
```

FIGURE 2. Example of ACID Transaction

A challenge while implementing multi-tenancy in SaaS application, is to ensure that the performance and resource consumption of one tenant does not adversely affect other tenants. There is a need to achieve an optimal degree of tenant isolation. There are generally the following requirements to be met: disk space reduction, use of locking, low cloud resource consumption, customization and use of plug-in architecture and choice of multi-tenant pattern. The degree of isolation is reduced when there is no strategy to reduce disk space, and customization and also when plug –in architecture is not adopted. The degree of isolation improves when the application knows how to handle high work load, locking of data, and preventing clashes between tenants.

The following section gives trade-off for required degree of tenant isolation for a multi-tenant SaaS application:

1. **Tenant isolation versus resource sharing:** As the degree of isolation increases, the ability to share resources decreases. A low degree of isolation leads to efficient utilization of resources. A high degree of isolation implies duplicating resources for each tenant as sharing is not allowed. A reduction in number of users that can access the component is observed.
2. **Tenant isolation versus number of users:** As degree of isolation increases, no of users that access decreases. As number of user increases, the physical contention also increases because more requests contend for available shared resources.
3. **Tenant isolation versus customizability:** The higher the degree of isolation, the easier it is to customize the component. Each time a multi-tenant application or its deployment environment changes, then a tedious, complex and maintenance process may be required.
4. **Tenant isolation versus size of generated data:** The more data is generated, it is more difficult to achieve a higher degree of isolation. Most systems create additional copies of files on shared repository. As time passes, these files occupy disk space. This results in adverse performance experience by tenants which leads to lower degree of isolation. A lot of time is spent fetching data from repository.
5. **Tenant isolation versus scope of control:** If scope of control is restricted to the SaaS layer, the architect may be able to implement a low degree of isolation. So for higher degree of isolation, the scope of control should extend to all layers of cloud stack
6. **Tenant isolation versus business constraints:** As degree of isolation increases from top to bottom layers, ease and flexibility to implement business requirements reduces. Shared components can be used to handle business requirements compensated at application level.

CONCLUSION

This paper concentrates on multi-tenant SAAS application which incorporate micro-services in their layered architecture. It gives the definition and benefits of micro-services. The main focus is on how to customize micro-service based SAAS applications and their benefits. Three approaches for customization that is, intrusive, non-intrusive and database approach are discussed. The pros and cons of each approach are studied. In particular, database approach can be secured using ACID properties for each transaction. Then, lastly the trade-off for required degree of tenant isolation for a multi-tenant SAAS application is given.

REFERENCES

1. Laud Charles Ochei, Julian M. Bass & Andrei Petrovski, Degrees of tenant isolation for cloud-hosted software services: a cross-case analysis, Journal of Cloud Computing, 17 December 2018
2. https://www.signiant.com/resources/tech-articles/the-benefits-of-saas-multi-tenant-architecture
3. Rajiv Shrivastava, 10 Challenges and Solutions for Micro-services — Tips and Tricks, DZone
4. Saba Annees, 4 Challenges You Need to Address with Microservices Adoption, May 12, 2016
5. Espen Tønnessen Nordli , Phu H. Nguyen, Franck Chauvel, Hui Song, Event-Based Customization of Multi-tenant SaaS Using Micro-services, International Conference on Coordination Languages and Models, Jan 2020.
6. https://microservices.io/patterns/data/shared-database.html
7. Guo, C.J., Sun, W., Huang, Y., Wang, Z.H., Gao, B.: A framework for native multi-tenancy application development and management. In: The 9th IEEE International Conference on E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007, CEC/EEE 2007. pp. 551–558. IEEE (2007).
8. Kabbedijk, J., Bezemer, C.P., Jansen, S., Zaidman, A.: Defining multi-tenancy: a systematic mapping study on the academic and the industrial perspective. J. Syst. Softw. **100**, 139–148 (2015).