



Article

Best practices for cloud computing multi-tenancy

Building a scalable, available cloud computing multi-tenancy architecture

☆ Save 👍 Like

Site feedback

By IBM Developer Staff

Updated July 4, 2011 | Published July 5, 2011

Cloud computing multi-tenancy is used for most if not all software as a service (SaaS) applications, because compute resources are scalable and allocation of these resources is defined by actual usage. Having said that, there are multiple types of SaaS applications that users can access through the Internet, from small Internet-based widgets to large enterprise software applications that have increased security requirements based on the type of data being stored on the software vendor's infrastructure outside the corporate network. These application requests require multi-tenancy for many reasons, the most obvious of which is cost: Adding a single stack of application servers and a database for each customer would not be sufficient in most cases, although it makes sense when there are higher security requirements.

This overview investigates and describes the types of multi-tenancy available and provides implementation use cases.

Multi-tenancy concepts

The concepts of multi-tenancy include three levels of consumer integration:

- Data center layer
- Infrastructure layer
- Application layer

The latest additions to cloud computing topology design are infrastructure and application-layer customer integration for multi-tenancy. This integration is specifically geared toward cost savings and developing highly scalable SaaS applications, which they do by compromising on security and customer segmentation requirements. Such designs are useful in many situations, although they may not be suitable for financial applications.

Site feedback

Renting space in a data center and supplying servers, routers, and cables that support multiple customer software requests have been around since Silicon Valley was in its infancy, so data center-layer multi-tenancy is well known. This configuration provides the highest level of security requirements if implemented correctly, with firewalls and access controls to meet business requirements as well as defined security access to the physical location of the infrastructure providing the SaaS. In most cases, data center-layer multi-tenancy can be a service provider that rents cages to companies that host their hardware, network, and software in the same building.

Multi-tenancy at the infrastructure layer is easiest thought of in software stacks, where one stack is dedicated to a specific customer. This configuration saves costs compared to data center-layer multi-tenancy, because stacks are deployed based on actual customer accounts. In this case, you can grow hardware requirements based on actual service use. In addition, high-availability may be an option for each customer at the infrastructure layer. Because the stack is known for each consumer, software and hardware best practices provide options for implementation.


Application-layer multi-tenancy requires architectural implementations at both the software layer and the infrastructure layer. Modifications are required for the existing software architecture, including multi-tenant patterns in the application layer. For example, multi-tenant applications require application methods and database tables to access and store data from different user accounts, which compromises on security. If done accurately, however, the benefit is cost savings. For widgets and simple web applications, application-layer multi-tenancy can be a solution, because a single developer can create software faster and scale more affordably. Disadvantages include more complex application architecture and implementation; rather than the infrastructure dealing with multi-tenancy, the application

teams and architects need to keep these programming patterns scalable, reliable, and future-proof in case the infrastructure architecture changes.

Services

Multi-tenant services specify access through HTTP RESTful interfaces or WSDL web service endpoints constructed into software applications and accessed directly. These services are critical to building service-oriented applications for multi-tenant patterns, because they can be reused for many transaction types. For example, the consumer of a multi-tenant application-layer service can call the servers by invoking a URL that in return produces XML as the response code:

[Site feedback](#)

https://visa.com/services/paymentOverview?account=OnlineShoesInc&pass=1234&=1_month 

```
<Response >
  < Report >
    <Title >Online Shoes Inc Report</Title>
    <Data><x>01/01/2011</x><y>20.11</y></Data>
    <Data><x>02/01/2011</x><y>22.24</y></Data>
    <Data><x>03/01/2011</x><y>20.21</y></Data>
  </Report>
</Response>
```



Show more ▾

The most critical part for multi-tenancy is that the account be specified in the URL as a parameter so that the infrastructure is aware which customer is requesting access to its data. This is the routing mechanism for multi-tenancy at the service level.

Application servers

The application server is a critical component in multi-tenancy at the application and infrastructure layer, because multi-tenancy affects installation, configuration, and application code. For the infrastructure layer, multi-tenancy for application servers means scaling fast and wide, with additional servers provisioned with an application server installation and configuration as well as application code. This layer of multi-tenancy does not require code changes unless specific requirements are set with the application; therefore, scaling is simple

and done mostly by the IT operations organizations rather than developers re-engineering application source code. Typically, if new customers are added, you can add a stack with the same configuration, meeting security requirements more easily.

One example is a stack that has the Web layer (HTTP server), application layer (application server), and database layer (database server) preconfigured for deployment to either physical hardware or virtualized instances of operation systems. This is a typical way to plan for growth for web-based applications, because demand can be high one day and low another. These instances can be scaled back during low-volume periods and scaled up as needed. The turnaround for these infrastructures, from growth requirements-gathering to running real customer transactions, is in most cases immediate, because stacks are preconfigured and automated for deployment.

[Site feedback](#)

At the application layer, multi-tenancy for application servers requires application code changes, because multiple customers are sharing the same application server. Response times may be affected regardless of whether the customer is running one transaction or a thousand at the same time, because other customers are running not only on the same server hardware but also within the same logical system memory. Depending on the application, there can be additional security requirements.

Transactions

Multi-tenant infrastructures and applications require transactions to authenticate each customer on submission of a request. This process helps authenticate and authorize the types of transaction resources a user can access.

The extraction of authentication and authorization services out of the application layer can help with scalability, maintainability, and re-usability for multi-tenant transactions. Because most application services added to infrastructures require authorization, a separated subnet, cloud, or cluster of application servers specifically for authentication meets requirement for scalability, maintainability, and re-usability. The same goes for authorization services, because this architecture can be reused and scaled within a cloud or subnet based on transaction growth.

The database

As the core component of most applications, the database is critical for multi-tenancy in terms of scalability. Because scalability databases require extended planning for the infrastructure and application layers, you need to understand the requirements of the application as well as best practices for scalable database infrastructures. If the infrastructure stack requires a single database for each customer account, scalability is straightforward, as these best practices have been established for single databases with failover. Also consider cost, because most commercial databases may have exponential license cost associated with adding to a per-account model.

If the multi-tenancy architecture is an application-layer implementation, you must understand the application well enough for database planning. Specific database schemata patterns apply to multi-tenant architectures, and these must be planned for accordingly. One common application design approach for saving cost and scaling on single licenses is to have the customer account name in the table name — for example, `customer123_payment`, where *customer123* is a unique identifier for the user account. This design increases the number of tables significantly over adding a database instance per customer or creating a column in each table to validate whether the customer is accessing the proper data.

[Site feedback](#)

Building multi-tenant services

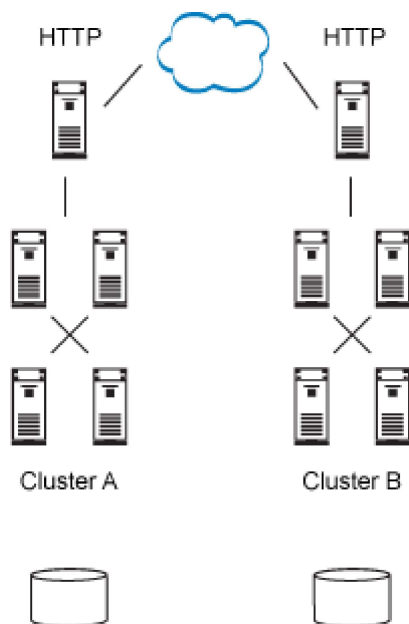
The requirements for building multi-tenant services are:

- Define RESTful or WSDL-based services.
- Define response times and performance goals.
- Determine scalability and-high availability requirements.
- Define the services required for each transaction.
- Determine load volumes for services based on transactions consumers.
- Create a deployment and network topology for services.
- Create deployment automation scripts for configuration and installation.
- Create a Unified Modeling Language (UML) sequence diagram for development implementation.

Planning the network topology

The network topology shown in [Figure 1](#) describes how services are deployed and accessed through the network using cloud-scalable resources. Other services can be added to the application layer where the servers are running as necessary. In Figure 1, Cluster A and Cluster B are exactly the same and distributed across two data centers or cloud providers. Each cluster has the same services deployed across all nodes for high availability and scalability.

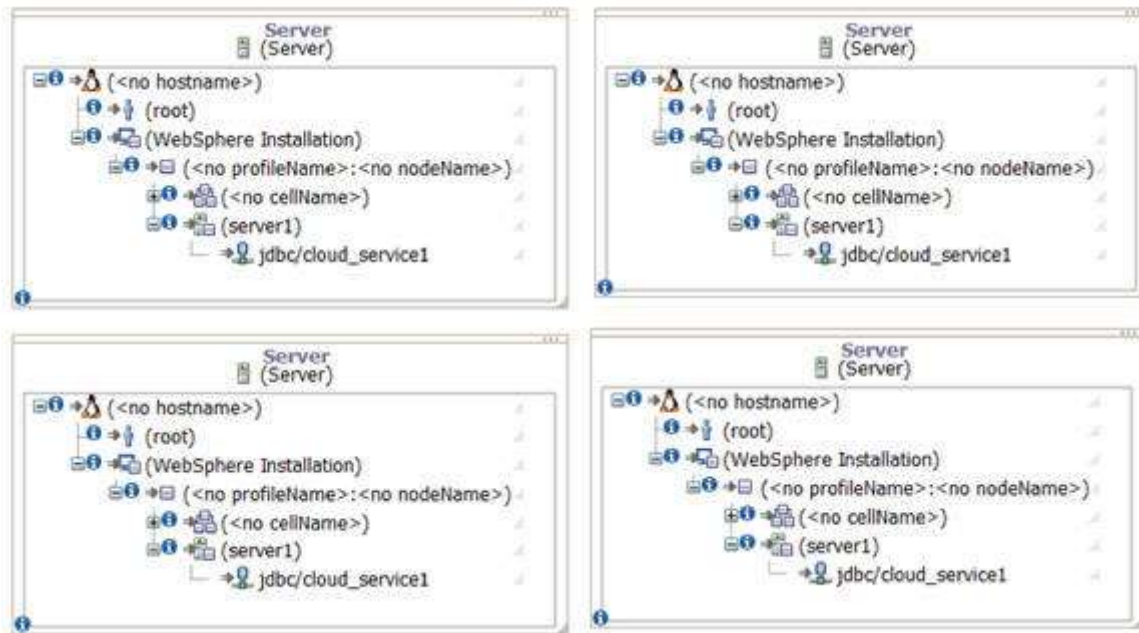
Figure 1. Network topology design for multi-tenant services



Planning the deployment topology

The deployment topology is required to identify the resources services required for the hardware and operating systems. Because each service will likely require JDBC, reference libraries, systems users, and messaging functionality, add these requirements to the deployment topology. Cloud infrastructures can become large, so it's best to prepare in advance. For instance, if a service calls to a specific data source or database, add that source to the deployment topology as shown in [Figure 2](#). In this figure, *cloud_server1* is the data source defined in the sample deployment topology.

Figure 2. Deployment topology design for multi-tenant services



Challenges

Multiple customers accessing the same hardware, application servers, and databases may affect response times and performance for other customers. For application-layer multi-tenancy specifically, resources are shared at each infrastructure layer and have valid security and performance concerns that you must plan for. For example, multiple service requests accessing resources at the same time increase wait times but not necessarily CPU time, or the number of connections to an HTTP server has been exhausted, and the service must wait until it can use an available connection or — in a worst-case scenario — drops the service request.

Building multi-tenant application servers

The requirements for building multi-tenant application servers are:

- Determine the application server for implementation.
- Define response times and performance goals.
- Determine scalability and high-availability requirements
- Define service and application deployment topologies.

- Determine load volumes for services based on transaction consumers.
- Create deployment and network topologies for services.
- Create deployment automation scripts for configuration and installation.
- Create a UML sequence diagram for development implementation.

Planning the network topology

Similar to the network topology for services, the network topology for multi-tenant application servers describes how application servers are shared across transaction consumers. In this example, you scale by adding a transaction path and cluster. All transactions, regardless of the consumer, are distributed across clusters A, B, and C.

[Site feedback](#)

Challenges

Because most application server deployments for enterprise infrastructures require complex installations and configurations, automation of this process is paramount. Scalability and new-customer on-boarding depends on provisioning application server resources as well as the guaranteed replication of infrastructure stacks and application-layer infrastructure. For example, operating system scripts for the automated installation of application servers are required to shorten the mean time to deployment for new accounts and services being added to the infrastructure.

These application server configurations can be extremely complex, as in most cases they align with development-created resources (for example, database connection source names and other Java™ Naming and Directory Interface [JNDI] resource names). If the resources that developers are required to subscribe to are defined in advance, the automation is simplified for scalability.

Building multi-tenant transactions

The requirements for building multi-tenant transactions are

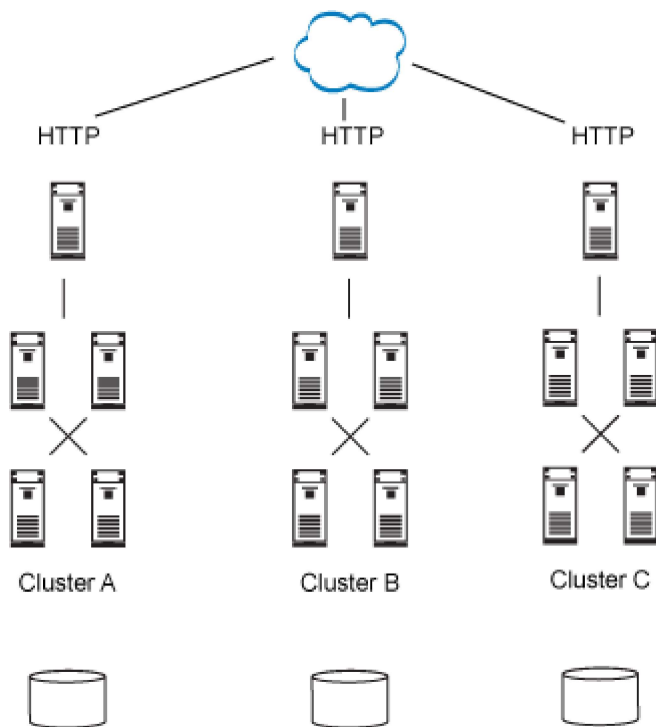
- Determine the transaction protocol for implementation.

- Define response times and performance goals.
- Determine scalability and high-availability requirements.
- Define service and applications deployment topologies.
- Determine load volumes for services based on transaction consumers.
- Create deployment and network topologies for services.
- Create deployment automation scripts for configuration and installation.
- Create a UML sequence diagram for development implementation.

Planning the network topology

Similar to application server multi-tenancy, transaction multi-tenancy can be configured and programmed to leverage a nonseparated or separated approach. As shown in [Figure 3](#), the clusters A, B, and C can be configured to process transactions per customer as well as all customers if the right load-balancing infrastructure is in place. Also, there is a requirement to write application code to process all transactions, if necessary, for multi-tenancy. For example, cluster A could be configured to process one customer, and clusters B and C could be configuring in a similar fashion or programmed to process multiple customers based on planning configurations. The configurations could also depend on service levels defined in the service agreements.

Figure 3. Network topology for multi-tenant transactions



Challenges

Because the customer is sending transactions that consist of multiple services to create the end-to-end transaction, there are a few challenges. If these transactions and individual services are not sized appropriately, it may affect performance and response times for all transactions. For instance, consider transactions that use fraud services along side transactions do not: Without the right sizing, it's difficult to determine in advance whether load requirements are met. The exponential utilization of the authentication service will be problematic, particularly if used multiple times per transaction. A cap on service utilization per transaction ensures the ability to size appropriately as well as providing a measure to reuse for capacity planning.

Building a multi-tenant database

The requirements for building a multi-tenant database are:

- Determine the database model for implementation.
- Define response times and performance goals.
- Determine scalability and high-availability requirements.

- Determine load volumes for transaction consumers.
- Create deployment and network topologies for services.
- Create deployment automation scripts for configuration and installation.
- Create a UML sequence diagram for development implementation.

Challenges

The multi-tenant database delivers new types of challenges at the application layer by adding many tables per customer to scale based on single-database licenses. The number of tables increases, affecting the disk space utilization assigned to a specific table, also known as *table space*. Because customers use different as well as unexpected amounts of data, table space sizes can vary significantly, affecting database sizing and planning considerations.

Keep in mind that auto-sizing options are available for database table spaces as well as detailed monitoring configurations that notify you if specific table spaces are full and need resizing. Best practice for preventing these problems is to size based on the average amount of data you expect a user to store, then add a buffer to ensure that in exceptional cases transactions and services can still be processed until table spaces have been modified to support customer-specific growth.

In conclusion

For the small startup or single developer to crank out applications quickly at an affordable price and to create proofs of concept as well as widgets for web applications, cloud computing multi-tenancy is appropriate and cannot be avoided going forward. Scalability and time to market are often at an even higher priority for widget and web application consumers. For example, applications for mobile or personal use and even small business applications with fewer security requirements are used based on innovation, response time, and ease of use rather than security and high availability.

In the case of enterprise applications, additional measures must be taken to ensure that security and high availability are addressed and implemented, all of which affects time to market and innovation. In the end, if cloud computing multi-tenancy applications are more secure and have improved response times, these organizations are able to deliver enterprise solutions that meet larger organizational requirements. For example, if the steps described in

this article are implemented to create multi-tenancy and integrated into an enterprise organization, segregation is required to resolve security concerns for the availability of each type of service to prevent denial of service issues and service-specific requirements. Multi-tenancy prevents services falling between the cracks for end-to-end transactions that may be required in the future but were not planned for.



Legend ⓘ

Categories



Table of Contents



Resources



Site feedback



Build

Smart ↓

IBM Developer

- About
- FAQ
- Third-party notice

Explore

Follow Us

- Twitter
- LinkedIn
- Facebook
- YouTube

- Best practices for cloud computing multi-tenancy - IBM Developer
- Newsletters
- Code patterns
- APIs
- Articles
- Tutorials
- Open source projects
- Videos
- Events

-
- Community
 - Career Opportunitites
 - Privacy
 - Terms of use
 - Accessibility
 - Cookie preferences
 - Sitemap

Site feedback