9

Software Development in Cloud

Learning Objectives

The main objective of this chapter is to introduce the concept of Software as a Service (SaaS) and its development using Platform-as-a-Service (PaaS) technology. After reading this chapter, you will

- Understand how SaaS applications are different from traditional software/application
- Understand how SaaS benefits the service providers and the end users
- Understand the pros and cons of different SaaS delivery models
- Understand the challenges that are introduced by SaaS applications
- Understand how to develop a cloud-aware SaaS applications using PaaS technology

Preamble

This chapter gives an insight about SaaS applications that are different and offer many advantages when compared to traditional applications. We cannot choose the SaaS delivery option for all kinds of applications. The suitability of SaaS is also discussed in this chapter as well as the many SaaS deployment and delivery models available for SaaS development. Even though SaaS applications offer more advantages to the consumers, they bring a lot of challenges to developers with regard to SaaS application development, and this chapter will further discuss these challenges. An overview of other cloud service models such as Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) that can be leveraged to develop multitenant-aware, scalable, and highly available SaaS applications is detailed in this chapter. This chapter also gives an idea about achieving secured multitenancy at the database level with different multitenancy models. Finally, the chapter discusses about the monitoring and SLA maintenance of SaaS applications.

9.1 Introduction

SaaS is a promising software delivery and business model in the information technology (IT) industry. The applications will be deployed by the SaaS provider in their managed or hosted infrastructure. The end users can access the hosted application as a service whenever there is a need. For using SaaS, the end users need to pay the full license fee. They can pay for what they used or consumed. To access the SaaS application, the customers need not install the software on their devices. It can be accessed from the service provider infrastructure using a simple web browser over the Internet. In the traditional software delivery model, the relationship between the end user and the software is one to one and licensing based. The end users have to buy the software from the vendors by paying a huge licensing amount. Some heavyweight applications need a high computing power. So, the end users have to buy the required hardware also. So, the initial investment to use the software is high, and if you look at the usage, it will be very low. To overcome this disadvantage of the traditional software delivery model, companies started developing the SaaS application, which has one-to-many relationships with the end users and the software. SaaS follows the multitenant architecture, and it allows many customers to share a single instance of the software, popularly known as multitenancy. Because of its cost-effective nature, many customers started moving to SaaS applications rather than the traditional licensing-based software. SaaS applications benefit not only the end users but also the service providers. In the traditional application service provider (ASP) model, the service providers host the applications on their own data center to benefit the end user. As the application delivery is one to one, the ASPs manage dedicated infrastructure for each customers, which forces them to invest a huge amount in managing the infrastructure, reducing their return on investment (ROI). Then the ASPs and the independent software vendors (ISVs) started using the alternate software delivery model (SaaS), which increased their ROI and at the same time benefitted the users. Now, most of the traditional ASPs and ISVs started realizing the business benefits of SaaS and started the SaaS development business.

9.1.1 SaaS Is Different from Traditional Software

The SaaS delivery model is different from the traditional license-based traditional software. The following discusses the many characteristics that differentiate the SaaS application from traditional applications:

- SaaS provides web access to commercial software on pay-as-you-use basis.
- SaaS applications are developed, deployed, and managed from a central location by the service provider.
- It allows the same instance of the SaaS application to be shared by multiple customers or tenants.
- Updates are performed by the service provider, not by the user.
- SaaS applications allow service integration with other third-party services through application programming interfaces (APIs) provided by them.

9.1.2 SaaS Benefits

SaaS applications provide cost-based benefits to the customers. It is also an on-demand, easy, and affordable way to use the application without a need to buy it. Additionally, SaaS solutions are easy to adopt and integrate with other software. Some of the notable benefits of SaaS are mentioned in the following:

1. *Pay per use*: SaaS applications are consumed by the end users on a pay-per-use basis. SaaS applications are on a subscription basis and allow the customers to use and disconnect the service as they wish. In the traditional software delivery model, the customers need to pay the full amount even if they use it very less.
2. *Zero infrastructure*: For installing and running traditional software, customers need to buy the required hardware and software, increasing the capital expenditure. In SaaS customers, there is no need to buy and maintain the infrastructure, operating system, development platforms, and software updates.
3. *Ease of access*: The SaaS application requires a simple web browser and an Internet connection to access it. The template-based responsive user interface (UI) will adapt automatically to the end user device, increasing the user experience and ease of access.
4. *Automated updates*: In the traditional software delivery model, the customers need to perform the bulk update, which is an overhead. But in SaaS, the service provider will perform the automated

updates. So, the customers can access the most recent version of the application without any updates from their side.

5. *Composite services*: Using SaaS applications, we can integrate other required third-party web services or cloud services through their API. It allows us to create a composite service.
6. *Dynamic scaling*: SaaS applications are used by a diverse user community. The load on the application will be dynamic and unpredictable. But with the dynamic load balancing capability, SaaS applications can handle any additional loads effectively without affecting their normal behavior.
7. *Green IT solutions*: SaaS applications are supporting the Green IT solutions. Since SaaS applications are multitenant and share the same resources and application instances, buying additional hardware and resources can be eliminated. The high resource utilization allows the application to consume less energy and computing power. SaaS solutions are becoming smarter and have energy-aware features in it that does not consume much resource for its operation.

9.1.3 Suitability of SaaS

SaaS applications are used by many individuals and organizations for its cost-effective nature. Their adoption by large enterprises also increases in a fair amount. But we cannot use SaaS applications in all places. The following are some applications where SaaS may not be the best option:

- Some real-time applications where fast processing of data is needed
- Applications where the organization's data are more confidential and the organization does not want to host their data externally
- Applications where existing on-premise applications fulfill the organization's needs

The following are examples where SaaS is the best option:

- For applications where the end user is looking for on-demand software rather than full-term/licensing-based software
- For a start-up company that cannot invest more money on buying licensed software
- For applications that need the accessibility from handheld devices or thin clients
- For applications with unpredictable and dynamic load

9.2 Different Perspectives on SaaS Development

The SaaS model provides web access to the commercial software. Since SaaS applications are deployed and managed by the service providers, customers are getting access to software services without any overhead of maintaining underlying infrastructure and platform. SaaS providers also can reduce the maintenance overhead by choosing other appropriate cloud services such as IaaS and PaaS, reducing the capital and operation expenditure of maintaining the servers. SaaS can be deployed and delivered from the traditional infrastructure or cloud infrastructure. There are different SaaS deployment and delivery models available to benefit the service provider and customer, which is discussed in this subsection.

9.2.1 SaaS from Managed Infrastructure and Platform

This model uses the traditional infrastructure and the platform for developing and deploying the SaaS application. Cloud computing characteristics will be satisfied only at the SaaS layer. In the other two layers (platform and infrastructure), the cloud characteristics will not be satisfied. This means that the underlying infrastructure and platform are not cloud enabled. The degree of multitenancy is also very low in this type of model as multitenancy is not achieved at the PaaS and IaaS levels. The developed SaaS application will be delivered to the customers in a one-to-many model. Figure 9.1 illustrates the concept of delivering SaaS from the self-managed infrastructure and platform.

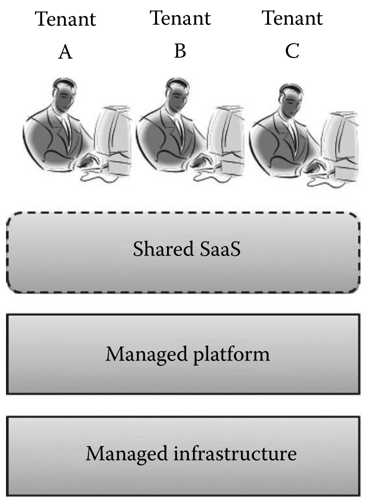

FIGURE 9.1
SaaS delivery from managed infrastructure and platform.

Pros

- This type of SaaS delivery model ensures more security to the user data as the infrastructure and platform are maintained by the SaaS provider.
- The SaaS provider gets full control over the infrastructure and development platform.
- There is no problem of vendor lock-in. The application can be easily migrated to any other infrastructure without any major modification.

Cons

- More overhead in maintaining the underlying infrastructure and platform.
- The service providers have to invest more on the infrastructure. So, this model is not suitable for SaaS development companies that do not have much amount to invest.
- Since the development environment is managed by the service provider, there is an additional overhead of maintaining the scalability and availability of the application.
- Resource utilization will be very low.

9.2.2 SaaS from IaaS and Managed Platform

In this type of service delivery model, the SaaS providers can use the infrastructure provided by any IaaS provider. The infrastructure provider may be a public or private IaaS provider. The public infrastructure will be chosen if the SaaS application does not require more security to the data. If the application needs more security and at the same time more resource utilization, they can choose the private IaaS model. Here, the multitenancy will be achieved at the infrastructure and application layers. The service providers have to manage their own development platform. Since the infrastructure is given by a third party, there is a possibility of vendor lock-in at the IaaS layer. Figure 9.2 illustrates SaaS delivery from the IaaS and self-managed platform.

Pros

- Ensures high resource utilization at the infrastructure level.
- Reduces the capital investment on the infrastructure.
- Capital investment can be reduced.

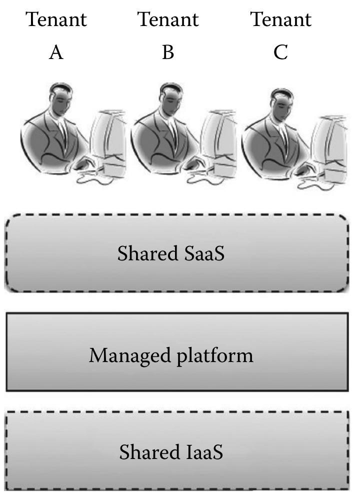

FIGURE 9.2
SaaS delivery from shared IaaS and managed platform.

Cons

- Still there is additional overhead in maintaining the development platform.
- Has to enable highly scalable and available features manually.
- There is a possibility of vendor lock-in at the infrastructure layer.

9.2.3 SaaS from Managed Infrastructure and PaaS

SaaS can be developed and delivered from self-managed infrastructure and shared PaaS. Normally, the application developed using this model will be deployed as an on-premise application. PaaS used here is generally a private PaaS. There are many PaaS providers who allow the customers to build their own private PaaS on their managed data center. This model will best suit the community deployment of SaaS. In the community deployment model, the infrastructure will be maintained by a group of organizations. The development platform (PaaS) can be accessed as a service by the different organizations. This type of model will reduce the overhead in maintaining the platform. SaaS-specific features such as high scalability and availability will be handled by the PaaS itself. But the overhead in maintaining the infrastructure will remain unsolved. This type of model also provides high security to the data. The vendor lock-in at the PaaS level can be possible in this type of model as the provider will be the third-party provider. This type of problem can be avoided by building our own PaaS platform on the managed

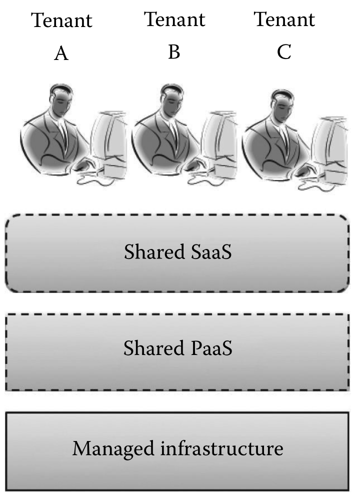

FIGURE 9.3
SaaS delivery from managed infrastructure and shared PaaS.

infrastructure. Figure 9.3 depicts the idea of developing and delivering SaaS from managed infrastructure and shared PaaS.

Pros

- The scalability and availability of the application will be provided by PaaS by default. So, the SaaS provider can concentrate more on application development.
- Security will be moderated, and there is full governance over user data.

Cons

- Even though overhead in maintaining the development platform is reduced, the overhead in maintaining the infrastructure still remains unsolved.
- This type of model will be suitable only for private/public SaaS applications. As the load on the public SaaS is high and unpredictable, the service providers may have to buy a new additional infrastructure to handle extra load. This is not possible for small SaaS development companies.

9.2.4 SaaS from IaaS and PaaS

This type of SaaS development and delivery model gets all the benefits of cloud computing. The infrastructure for developing and deploying a SaaS application will be provided by the IaaS provider, reducing the overhead in maintaining the underlying infrastructure. In the same way, the development platform also can be provided by the PaaS provider. The IaaS and PaaS provider might be public or private. The public IaaS and PaaS give more benefits than the private IaaS and PaaS. Normally, the public IaaS and PaaS will be selected to reduce the maintenance overhead and initial investment. Multitenancy is provided at all layers, that is, infrastructure, platform, and application layers. This type of multitenancy is called as high-level multitenancy, which is not available in other SaaS development and delivery models. Figure 9.4 illustrates the idea of developing and delivering SaaS from the IaaS and PaaS.

Pros

- The best delivery model that suits public SaaS applications.
- Ensures high resource utilization as it enables multitenancy at all layers of the application. It also supports Green IT applications.
- Dynamic scaling of IaaS and PaaS provider ensures the high scalability of the application. SaaS development companies need not worry about the scalability of the application.
- The high availability of the applications is ensured by the replica and the backup and recovery mechanism provided by the service provider.
- No overhead in maintaining the infrastructure and development platform. This enables SaaS development companies to develop more applications in a short span of time.

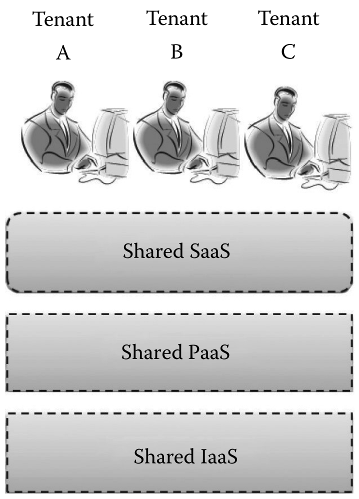

FIGURE 9.4
SaaS delivery from shared IaaS and PaaS.

Cons

- Since this type of model mostly uses public IaaS and PaaS models, the application will be hosted as off-premise applications. So, there is no governance over customer data.
- There is a possibility of cross tenant attacks as multitenancy is enabled at all levels of the application.

There are different development and deployment models discussed in this subsection. The deployment model of SaaS (public or private or community) will be selected based on the security requirement of the user data. The investment of buying and managing the infrastructure also will be considered before selecting the deployment model.

9.3 New Challenges

Many on-premise web applications are replaced by SaaS applications because of the business benefits of SaaS applications. Adoption of SaaS by large enterprises is increasing so as the adoption by individual users. This is due to the security issues of the data that are stored in the cloud. Large enterprises are worrying about data security, data governance, and availability. In this subsection, we shall discuss the challenges that make SaaS development difficult.

9.3.1 Multitenancy

Enabling multitenancy in a SaaS application is a challenge that all developers are facing today. Multitenancy can be achieved at the infrastructure, platform, database, and application levels for better resource utilization. Multitenancy is a one-to-many model that allows multiple customers to share a single instance of code and database of the SaaS application. So, developers need to learn the required knowledge of developing a multitenant software. For example, there are several multitenancy levels available with respect to database, namely, separate database, shared database and separate schema, and shared database and shared schema. The developer has to choose the correct multitenancy level based on the customer's requirement before developing the application. The developers can make use of any PaaS to reduce the overhead in enabling the multitenancy of the SaaS application.

9.3.2 Security

The first important consequence of the multitenancy model of the SaaS application is data security. Since the environment is shared, there is a possibility of data breaches. One tenant can easily get access to the other tenants' data, which will result in a serious issue. This is the reason why most of the enterprises are not ready to adopt SaaS applications for their business needs and isolating tenants' data is one way to overcome this, which is the biggest challenge for any SaaS developer. As SaaS applications are web based, the security threats that are applicable to traditional applications are applicable to SaaS applications as well. Internal and external security attacks should be detected and prevented by using proper intrusion detection and intrusion prevention systems. The developers can incorporate other security mechanisms such as strong encryption, auditing, authentication, access control, and authorization to secure SaaS applications.

9.3.3 Scalability

A scalable SaaS application should handle the extra load on the application efficiently. A SaaS application is said to be highly scalable if it handles the additional load properly. The scalability of the application can be maintained by predicting the load on the system and providing enough resources to handle the load. Since SaaS application users are diverse and can connect and disconnect from the system at any time, predicting the load on the application is a big challenge to the developer. So, the software architect should design an architecture that can handle any kind of load on the application. The architect should use vertical scaling, horizontal scaling, and load balancers to develop a highly scalable application. The scalability at the platform and infrastructure level also can decide the efficiency of the application. So, the application architecture should use IaaS and PaaS to utilize the advantages of cloud services.

9.3.4 Availability

Since SaaS users are storing their data in the service provider data center, ensuring the 99.99% availability of the data is a challenge to SaaS developers, and a better way to address this is to maintain a proper backup or replica mechanism. We cannot predict the failure of data stored in advance and cannot take preventive measures. Most of the traditional application developers rely on third-party tools to ensure the backup and disaster recovery. A developer can use distributed NoSQL databases that support automatic replication and disaster recovery rather than relational databases.

9.3.5 Usability

A new challenge introduced by the SaaS application is multiple-device support. The user may access the application from laptops, mobiles, tablets, and other handheld devices. Traditional web application developers used to develop rich Internet applications that are not adaptable to mobile devices.

So, by keeping the diversity of the end user devices, developers should develop a responsive web UI that adapts automatically to all user devices. The developers cannot develop a separate version of the application for different devices. The other challenge to the developer is to maintain the per tenant customization settings. The UI customization settings of each user should be managed properly for each device and should not affect the others.

9.3.6 Self-Service Sign-Up

Many traditional social networking sites use the feature of self-service sign-up for the users. But it is not mandatory for all the traditional web applications. But for the SaaS application, it is a mandatory feature to be incorporated, creating a challenge to the developers. SaaS applications should be available to the users as soon as they register. The application should not encourage any admin approval to allow the user to access the service. In the same way, the application should allow the users to unsubscribe from using the services at any point in time.

9.3.7 Automated Billing

The other challenge that is imposed by the unique characteristics of SaaS is automated billing. Most of the existing SaaS application users are facing the problem of abusive billing even after disconnected from the services. So, the developers should incorporate a proper mechanism to avoid abusive billing and maintain the usage history of each tenant. The tenant may subscribe to different services of the same service provider, and a mechanism should be available to the customers to see their total and per service usage report and billing.

9.3.8 Nondisruptive Updates

The frequent updates of the application may result in the unavailability of the application. Whenever the update is performed, it should not affect the normal behavior of the user. The other consequence of the update may lead to service-level agreement (SLA) violation. The update operation decides the downtime of the application, which is an important SLA parameter. The update should be performed in such a way that it does not increase the downtime that is mentioned in the SLA. So, scheduling and performing the nondisruptive updates of SaaS applications are the biggest challenges that developers face to avoid SLA violation.

9.3.9 Service Integration

A good SaaS application should be able to integrate with other third-party services. Normally, we cannot integrate third-party services directly. We have to use the APIs of service providers to perform service integration. So, the architecture of the SaaS application should allow service integration of other services through their APIs. Service integration can be achieved by following proper service-oriented architecture (SOA). Another benefit of service integration is that we can automate the implementation of some functionalities. Many SaaS providers fail to support service integration, which they need to improve while developing SaaS applications.

9.3.10 Vendor Lock-In

The major problem of public cloud services is vendor lock-in. There is no global standard followed among the service providers. Each service provider follows their own way of providing infrastructure and platform services, and the migration of the SaaS application from one service provider to the other becomes difficult and leads to vendor lock-in. So, developers should select the PaaS or IaaS that is interoperable with other service providers.

9.4 Cloud-Aware Software Development Using PaaS Technology

PaaS is a widely used cloud service model that enables the developers to develop an application online. PaaS provides the development PaaS on a 0 demand basis. The developers need not install any heavyweight software in their machine to use PaaS. The developers can develop and deploy an application online through the client tools such as web UI, Command Line Interface, web CLI, and representational state transfer Representational State Transfer (REST) API provided by the service provider. Normally, the development of SaaS application imposes a lot of challenges as discussed in the previous section. With the traditional development environment, it is very difficult to develop a successful SaaS that satisfies all the SaaS-specific requirements. To overcome these challenges, SaaS development companies can use the popular PaaS that provides many SaaS-specific features by default. By using PaaS, the developers can concentrate more on application functionalities rather than struggle with enabling SaaS-specific features. In this subsection, we shall discuss developing cloud-aware SaaS applications using PaaS technology.

Benefits of PaaS: PaaS is used by many small SaaS development companies and ISVs. The following discusses the many characteristics that increase PaaS adoption by an organization:

- PaaS provides the services to develop, test, deploy, host, and maintain applications in one place.
- Most of the service providers offer *polyglot* PaaS where the developers can use a variety of application development environments in one integrated development environment (IDE).

- The variety of client tools such as web UI, web CLI, REST APIs, and IDE integration increases the ease of application development and deployment.
- PaaS offers a built-in multitenant architecture for the applications developed using PaaS.
- PaaS provides a software load balancer that ensures the dynamic scaling of the application.
- The replicas maintained by PaaS providers ensure the high availability of the application.
- PaaS providers also allow integrating with other web or cloud services to develop composite cloud services.
- PaaS increases the collaboration between the development team as the application will be deployed at a central place.
- The other important SaaS-specific features like monitoring tools and automated billing will be offered by PaaS itself.

Before PaaS and after PaaS: PaaS changes the software development process totally when compared to the traditional software development model. In the traditional software development, the development process will start from requirements analysis that involves all the stakeholders of the system. The second phase is the design phase that includes software architecture, UI, and database design. The implementation phase is the actual development, or coding the application with the available development platform. Here, the development platform will be a license-based heavyweight software that requires machines with high computing power, forcing companies to invest more on the development platform and hardware. The testing is the next phase of software development that mainly ensures the nonfunctional requirements like security, scalability, availability, and portability. There are many tools available in carrying out the testing process of the developed application. After testing a product, it can be deployed in suitable infrastructure and can be delivered to the end users. Normally, if the application is a stand-alone application, it will provide a license. If it is a web application, it will be delivered through the Internet. Generally, each customer of the application will get a separate copy of the application. In the case of a web application, the application will be hosted in the service provider infrastructure or customer on-premise infrastructure. The next step is to maintain the scalability and availability of the application. The company should keep enough additional servers to handle the extra load of the application. But in real time, the load on the application will be dynamic and unpredictable. So, we cannot decide the power of the hardware, which we need to add later. Here, buying and maintaining the additional hardware will be an overhead to development companies. The next important thing is availability of the application. To ensure high availability, companies need to keep replicas. Again replica management is a big issue for the companies. In the end, the maintenance process of the application will be carried out by companies. Normally, updates will be performed by customers from their machines. Each copy of the application should be updated separately. Additionally, sometimes updates through a slow Internet connection will disrupt the normal behavior of the system. So, to avoid all these problems in traditional software development methodologies, companies started using the PaaS technology for better productivity.

Figure 9.5 illustrates the software development process before and after PaaS. The main problem in traditional software development is the licensed platform, overhead in testing, deployment, scaling, and maintenance of the application. But PaaS hides all of the overheads that are present in traditional software development. Here, the PaaS platform will be provided to the developers on an on-demand basis through the Internet. So, the developers can use the platform to develop their application. The developed application will be automatically deployed on the service provider infrastructure by the PaaS tool itself. The other important parameters such as dynamic scaling and availability will be handled by the PaaS tool. Sometimes, because of poor Internet connection, the developers may not able to use PaaS online. So, to overcome this problem, some of the PaaS vendors allow the offline software development by

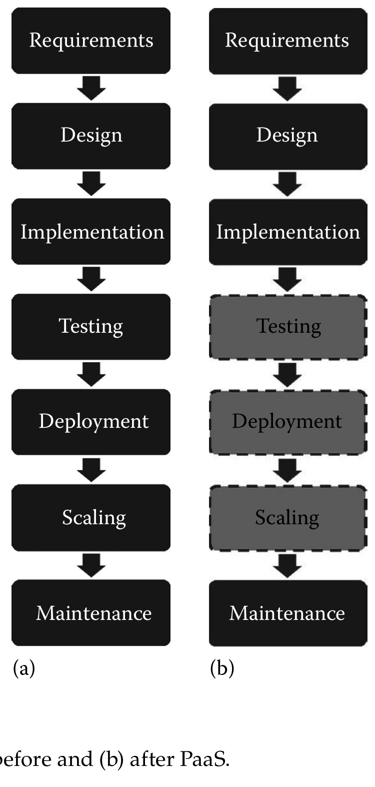

FIGURE 9.5
Software development (a) before and (b) after PaaS.

integrating their online repository with the local IDEs. This enables the developers to work offline in the local machine and push the application online whenever there is an Internet connection. The other advantage is centralized maintenance. Here, the application is hosted in a central location. So, the customers need not perform the updates from their machine as the service provider updates the application that will be effected to all the customers.

The developed SaaS application is different from the traditional application. SaaS application developers should enable the following features to the application: multitenancy, dynamic scaling, and high availability. The process of developing a SaaS application using PaaS technology is discussed in the following.

9.4.1 Requirements Analysis

The development team should cope up with frequently changing requirements of the SaaS application. Generally, in requirements analysis, only the customers and the development team will be involved. But in SaaS application development, the service provider for the PaaS tool should also be involved. The requirements collection team should collect the requirements from all the stakeholders. The requirement document should include functional, nonfunctional, and other SaaS-specific requirements. Before developing the application, the requirements collection team should analyze the suitability of the SaaS delivery model for the customers' requirements. Generally, the SaaS delivery model will be selected based on security and ROI. The next important thing in the requirements analysis phase is the deployment model of the SaaS application. If the application does not need more security, we can develop the SaaS application from any public PaaS provider. If it requires more security, then we have to select any private PaaS provider to develop the application. Normally, in the public deployment model, the overhead in maintenance will be low and security threats will be high. But in the private deployment model, overhead in maintenance will be high and security threats will be moderate. As the SaaS application is going to be delivered through the Internet, the SaaS development company should assign highly skilled security experts and software architects in order to succeed in the SaaS development business. Security experts should ensure the security at all layers of the application. The software architect should ensure the scalability and availability of the application. Once the requirements analysis is properly done, the development team may start designing the architecture.

9.4.2 Multitenant Architecture

An important characteristic of the SaaS application is multitenancy where multiple customers are allowed to share the same application. Achieving multitenancy depends on the software architecture. The software architecture should ensure the multitenancy of the SaaS application. Multitenancy can be achieved at the infrastructure, development platform, database, and application

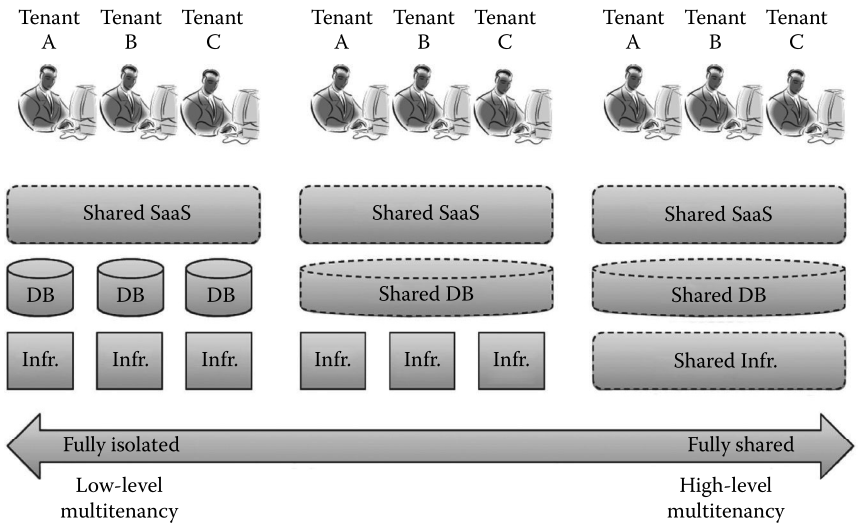

FIGURE 9.6
Different multitenancy levels.

levels. The architect can choose different levels of multitenancy to utilize the resources effectively. If the software architect selects an IaaS provider for the infrastructure needs, then the architect is relieved from enabling multitenancy at the infrastructure level. If the architect chooses the PaaS provider, the platform level and the infrastructure level multitenancy will be provided by the PaaS provider itself. So, the software architect's job gets reduced to enable the multitenancy features only at the software level. Depending on the multitenancy level, the isolation security of the data is decided. If multitenancy is achieved at all levels, then it is called as high-level multitenancy. If multitenancy is achieved only at the application level, then it is called as low-level multitenancy. The security threat to data will increase as the multitenancy level increases. Depending on the user security requirements, the architect should select the multitenancy level. The other advantage of multitenancy is resource utilization. If the company wants high resource utilization, they can go for a high-level multitenancy. Figure 9.6 illustrates the different levels of multitenancy and isolation available to ensure resource utilization and security to the software architect.

9.4.3 Highly Scalable and Available Architecture

Another important characteristic of the SaaS application is dynamic scaling. The dynamic scaling feature is not mandatory in the case of traditional web applications. But in the case of the SaaS application, dynamic scaling is very important.

Like multitenancy, achieving dynamic scaling also depends on the software architecture. As the load on the SaaS application becomes unpredictable and increases or decreases any time, the architecture should ensure the same performance on varying loads. The scalability of the SaaS application can be achieved using horizontal scaling, vertical scaling, software load balancer, and hardware load balancer. In horizontal scaling, identical resources (application server, database server, and infrastructure) will be added to the application to handle the additional load. In vertical scaling, the capacity of the server (application, database, and infrastructure) will be increased as the load increases. The software load balancers also can be used to ensure the dynamic scalability of the SaaS application. The role of the software load balancer is to distribute additional user request across different application and database servers. The hardware load balancer will distribute the load across different virtual machines when there is a need for more computing power. If the infrastructure and development platform is consumed from any service providers (IaaS, PaaS), they will provide the hardware and software load balancers to balance the load. If the platform and the infrastructure are self-managed, then the SaaS development company should rely on third-party tools or they have to develop their own. Figure 9.7 illustrates the typical SaaS architecture used to achieve high scalability and availability of the SaaS application.

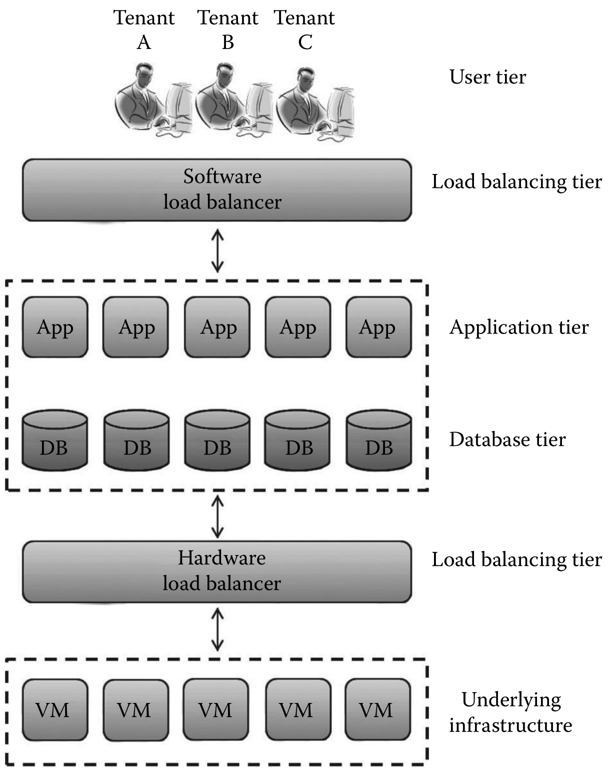

FIGURE 9.7
Typical architecture of the SaaS application.

Like multitenancy and scalability, the availability of the application is also an important characteristic of a SaaS application. The availability of the application decides its uptime and downtime. The availability of the application is an important parameter of the SLA. Ensuring the 99.99% availability of the application depends on the replica mechanism that is specified in the software architecture. As shown in Figure 9.7, multiple copies of virtual machines and database applications should be maintained for high availability. While maintaining the replica, another important aspect is recovery time after any failure. The application should be fault tolerant, and the recovery time should be minimal to avoid SLA violation. The replica should be maintained near the customer location to reduce the recovery time after any failure or disaster.

9.4.4 Database Design

Achieving multitenancy, scalability, and availability at the database level is an important criterion for successful SaaS development. The database design for multitenancy should consider the security requirement of the data.

Multitenancy at database level can be achieved by sharing the database instance, sharing the database table, and sharing the database schema. Depending on the security of the application, the database should be designed to secure multitenancy. The database-level multitenancy can be achieved in three different ways as illustrated in Figure 9.8. If the database designer selects a separate database for different tenants as shown in Figure 9.8b, the security will be ensured. If the shared database and separate schema are

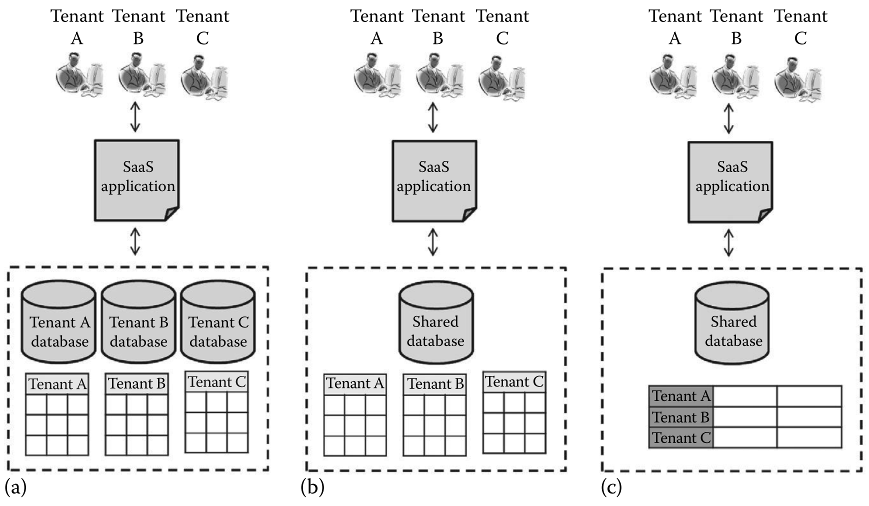

FIGURE 9.8
Database-level multitenancy: (a) separate database, (b) shared database and separate schema, and (c) shared database and shared schema.

selected as shown in Figure 9.8a, the security to the data will be moderated. The third multitenancy model shares the database and the schema for the tenants as shown in Figure 9.8c.

The scalability and availability of the database decide the performance of the application. Most of the SaaS applications are interactive and involve a large number of database of read and write requests from the users. When the number of requests exceeds the actual capacity of the database server, additional requests should be redirected to the other server. When requests are redirected, a change of data in one database server should reflect in other database servers also. Mostly the type of data used in the SaaS application will be diverse and includes structured, semistructured, and unstructured data in huge amounts. So, for SaaS applications, Not Only Structured Query Language (NoSQL) databases will be a better option than traditional relational, object-oriented databases. The developers can leverage the advantages of NoSQL databases to achieve scalability and availability at the database level in an efficient way.

9.4.5 SaaS Development

After designing the architecture and the databases, the developers need to implement the functional requirements given by the customers. As we discussed earlier, PaaS facilitates the developers in developing highly scalable, available, and multitenant-aware SaaS applications.

The PaaS tools allow the developers to develop the application online, and the application will be deployed on the service provider infrastructure as soon as the developer pushes the application online. Here, the end users or SaaS consumers can access the application online using the web UI provided by the SaaS provider. Figure 9.9 illustrates the overview of SaaS development using PaaS tools.

The PaaS providers also provide testing tools in the same development environment to facilitate the developers. So, the developers can use built-in testing tools provided by the PaaS providers to test SaaS applications. Some PaaS providers offer automated testing of the applications also. While developing the application, the developers should incorporate the following things for successful SaaS:

- Responsive UI design to support multiple devices
- Role Based Access Control (RBAC), Access Control List (ACL) mechanism to uniquely identify users and tenants
- Monitoring tools that will monitor the performance and notify the service provider frequently
- Control panel for the tenant and admin to manage the users
- User-centric customization panel that does not affect the settings of other tenants or users
- Self-service sign-up for the users

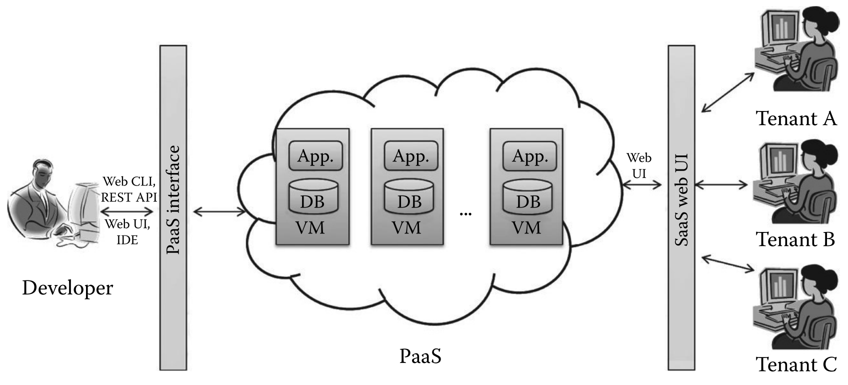

FIGURE 9.9
Overview of SaaS development using PaaS.

- Usage statistics and bill calculation
- Help documentation to use the service
- Service integration

9.4.6 Monitoring and SLA Maintenance

As soon as the SaaS application is developed and deployed using PaaS, the end users can access the SaaS application over the Internet from any end user device such as desktops, laptops, tablets, and mobiles. After delivering the application, any misbehavior, failure, security attacks, and disasters of SaaS applications should be monitored and prevented. Since a lot of customers are sharing the same instance of a single application, any misbehavior from one tenant will affect the other tenants and the underlying resources. Updates should be scheduled and performed in such a way that it does not affect the normal behavior of the system, so updating the SaaS application frequently will offer the its most updated version to the end users. But if you update the application with bulk updates frequently, it may lead to the unavailability of the application. Another important job in SaaS monitoring is to monitor the SLA violation by both the service provider and customer. If there is any SLA violation, the monitoring tool should notify the developers to correct the errors that lead to the SLA violation. The SaaS providers should define the SLA clearly to the end users before delivering any services. The SLA should include the availability, response time, and degree of support. The service providers also should provide 24 × 7 support to the end users. The development team should resolve the issues frequently as soon as feedback is received from the end users. There are many third-party monitoring tools are available to monitor SaaS applications. The SaaS development company can make use of those monitoring tools to reduce the overhead.

9.5 Summary

SaaS is one of the important services provided by cloud computing. Usage-based billing, high scalability, ease of access, and other benefits make the most of the customers to move from traditional applications to SaaS applications. Small business companies/start-up companies started using SaaS to reduce their investment on buying software that is underutilized in their organization. By using on-demand SaaS applications, any company can increase their ROI. If you look at the usage of SaaS in large enterprises, it is very low compared to the usage of individuals and small business companies. Large enterprises are hesitant to use SaaS applications for their organizations because of security issues. SaaS applications are multitenant. Whenever the users are sharing the application, there is a possibility of security attacks between the tenants. If you remove the multitenant features from SaaS applications at the infrastructure, platform, and software levels, it will result in a high development cost. Obviously, the customers need to pay more for the software they use. When someone decides to go for SaaS applications, they have to consider its cost and security requirements. Based on the cost and security requirements, the service providers can follow any of the SaaS development and deployment models discussed in this chapter. Other than the security issues, the SaaS application introduces a lot of challenges to the developers such as scalability, availability, usability, self-service sign-up, and automated billing. These challenges can be addressed by incorporating the best practices into software engineering and PaaS technology. SaaS changes the way the software is delivered, and PaaS changes the way the software is developed. PaaS automates the process of deployment, testing, and scaling and reduces manual work and the cost involved in developing the application. SaaS providers also can utilize IaaS of cloud computing to reduce the investment on buying infrastructure.

Review Points

- *SaaS* is one of the software delivery models that allow the end users to share the application that is centrally hosted by a provider (see Section 9.1).
- *SaaS* contains unique characteristics that differentiate it from traditional software (see Section 9.1.1).
- *SaaS benefits* include pay per use, zero infrastructure, ease of access, automated updates, and composite services (see Section 9.1.2).
- *SaaS does not suit* the application where fast processing of data is needed (see Section 9.1.3).

- *SaaS delivery* can be of many forms: managed infrastructure and platform, IaaS and managed platform, managed infrastructure and PaaS, and IaaS and PaaS (see Section 9.2).
- *SaaS challenges* such as multitenancy, security, scalability, availability, and usability make SaaS development a difficult job for developers (see Section 9.3).
- *Multitenancy* is a one-to-many model where a single instance of an application can be shared by multiple users (see Section 9.3.1).
- *Scalability* of the SaaS application depends on how well the application will handle the extra load (see Section 9.3.3).
- *Availability* of the SaaS application can be improved by keeping proper backup and recovery mechanisms (see Section 9.3.4).
- *Usability* of the SaaS application depends on the adaptive and responsive UI design that supports multiple devices (see Section 9.3.5).
- *Self-service sign-up* feature of the SaaS application allows the end users to subscribe or unsubscribe from the service without the intervention of the provider (see Section 9.3.6).
- *Automated billing* feature maintains the usage history and provides the bill based on per tenant usage or per service usage (see Section 9.3.7).
- *Nondisruptive updates* ensure the uptime of the application and during the time of application update also (see Section 9.3.8).
- *Service integration* of the SaaS application allows any SaaS application to integrate with other services through an API (see Section 9.3.9).
- *Vendor lock-in* does not allow migration of application to other service providers, which is the problem with most of the public cloud providers (see Section 9.3.10).
- *PaaS* changes the way software is developed by providing development PaaS (see Section 9.4).
- *Cloud-aware software development* requires multitenant, highly scalable architecture (see Section 9.4).

Review Questions

1. What is Software as a Service (SaaS)? How is it different from traditional software?
2. Briefly explain the benefits of the SaaS application.
3. Is it wise to choose the SaaS delivery model for all kinds of applications? Justify your answer.

4. Explain the different SaaS development and deployment models with neat diagrams.
5. List out the pros and cons of different SaaS development and deployment models.
6. List the challenges that make SaaS development a difficult task. Also, explain any five challenges in detail.
7. Write short notes on the benefits provided by PaaS technology for developing SaaS applications.
8. Explain in detail how PaaS technology changes software development.
9. Briefly explain the requirements analysis for SaaS application.
10. Explain different multitenancy levels with neat diagrams.
11. Illustrate and explain the typical architecture of the SaaS application, which is to ensure better scalability and high availability.
12. How is database-level multitenancy achieved? Explain the different database-level multitenancies with neat diagrams.
13. List out important features that SaaS developers should incorporate while developing SaaS applications.
14. Write short notes on monitoring and SLA maintenance of SaaS applications.

Further Reading

6 best practices to cloud enable your apps. White Paper, Tier 3, Inc.

Best practices for cloud computing multi-tenancy. White Paper, IBM Corporation, 2003.

Betts, D., A. Homer, A. Jezierski, M. Narumoto, and H. Zhang. *Developing Multi-Tenant Applications for the Cloud on the Microsoft Windows Azure*. Microsoft Press, 2010.

Building successful enterprise SaaS apps for the cloud. White Paper. THINKstrategies, Inc., 2011.

Chauhan, N. S. and A. Saxena. A green software development life cycle for cloud computing. *IT Professional* 15(1): 28–34, 2013.

Chong, R. F. Designing a database for multi-tenancy on the cloud: Considerations for SaaS vendors. Technical article, IBM Developer Works. Available [Online]: http://www.ibm.com/developerworks/data/library/techarticle/dm-1201dbdesigncloud/dm-1201dbdesigncloud-pdf.pdf Accessed October 12, 2013.

da Silva, E.A.N. and D. Lucredio. Software engineering for the cloud: A research roadmap. *26th Brazilian Symposium on Software Engineering (SBES)*, September 23–28, 2012, pp. 71–80.

Deploying software as a service (SaaS). White Paper, WebApps, Inc. a.k.a. SaaS.com.

Gagnon, S., V. Nabelsi, K. Passerini, and K. Cakici. The next web apps architecture: Challenges for SaaS vendors. *IT Professional* 13(5): 44–50, 2011.

Goth, G. Software-as-a-service: The spark that will change software engineering? *Distributed Systems Online, IEEE* 9(7): 3, 2008.

Kang, S., J. Myung, J. Yeon, S. Ha, T. Cho, J. Chung, and S. Lee. A general maturity model and reference architecture for SaaS. *Proceedings of Database Systems for Advanced Applications (DASFAA 2010)*, Part II, LNCS 5982, pp. 337–346.

Lawton, G. Developing software online with platform-as-a-service technology. *Computer* 41(6): 13–15, June 2008.

Liu, F., J. Tong, J. Mao, R. B. Bohn, J. V. Messina, M. L. Badger, and D. M. Leaf. NIST cloud computing reference architecture. NIST Special Publication 500-292, September 2011. Available [Online]: http://www.nist.gov/customcf/get_pdf.cfm?pub_id=909505 (accessed September 3, 2013).

Mell, P. and T. Grance. The NIST definition of cloud computing. NIST Special Publication 800-145, September 2011. Available [Online]: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf (accessed September 3, 2013).

Yau, S. S. and H. G. An. Software engineering meets services and cloud computing. *Computer* 44(10): 47–53, October 2011.

Rodero-Merino, L., L. M. Vaquero, E. Caron, A. Muresan, and F. Desprez. Building safe PaaS clouds: A survey on security in multitenant software platforms. *Computers and Security* 31(1), 96–108, 2012. ISSN 0167-4048.

SaaS Architecture. White Paper, Progress Software Corporation.

SaaS Scalability. White Paper, Progress Software Corporation.

10

Networking for Cloud Computing

Learning Objectives

After studying this chapter, you should be able to

- Understand the general classification of data centers
- Present an overview of the data center environment
- Understand the basic networking issues in data centers
- Explain the performance challenges faced by TCP/IP in data center networks
- Describe the newly designed TCPs for data center networks and their novelty

Preamble

This chapter provides an introduction to networking in Cloud Enabled Data Centers (CEDCs) and the issues thereof. A general classification of data centers and a brief overview of the data center environment are provided to familiarize the reader with the CEDCs. Major issues related to networking in a cloud environment are presented with an emphasis on TCP/IP-related performance issues. Newly designed protocols tailored specifically for data center networks are explained in detail, while mentioning advantages and disadvantages of each.

10.1 Introduction

The Internet over the past few years has transformed from an experimental system into a gigantic and decentralized source of information. Data centers form the backbone of the Internet and host diverse applications ranging