# Migrating Medical Communications Software to a Multi-Tenant Cloud Environment

Pieter-Jan Maenhaut*†, Hendrik Moens*, Marino Verheye‡, Piet Verhoeve‡, Stefan Walraven§, Eddy Truyen§, Wouter Joosen§, Veerle Ongenae† and Filip De Turck*
*iMinds – INTEC, Ghent University, Dept. of Information Technology
Gaston Crommenlaan 8 bus 201, 9050 Ghent, Belgium
†Faculty of Applied Engineering Sciences (INWE), University College Ghent,
Valentin Vaerwyckweg 1, 9000 Ghent, Belgium
‡Televic Healthcare, Leo Bekaertlaan 1, 8870 Izegem, Belgium
§ iMinds – DistriNet, KU Leuven, Dept. Computer Science
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
Email: pieterjan.maenhaut@intec.ugent.be

***Abstract*—The rise of cloud computing has paved the way for many new applications. Many of these new cloud applications are also multi-tenant, ensuring multiple end users can make use of the same application instance. While these technologies make it possible to create many new applications, many legacy applications can also benefit from the added flexibility and cost-savings of cloud computing and multi-tenancy.**

**In this paper, we describe the steps required to migrate a .NET-based medical communications application to the Windows Azure public cloud environment, and the steps required to add multi-tenancy to the application. We then discuss the advantages and disadvantages of our migration approach. We found that the migration to the cloud itself requires only a limited amount of changes to the application, but that this also limited the benefits, as individual instances would only be partially used. Adding multi-tenancy requires more changes, but when this is done, it has the potential to greatly reduce the cost of running the application.**

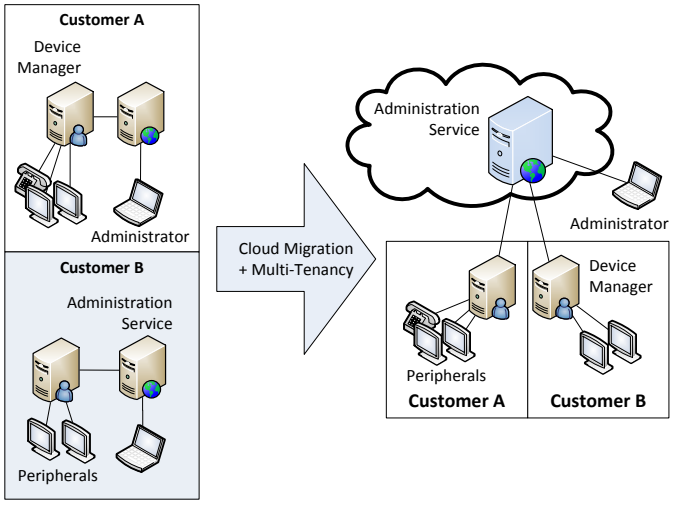



Fig. 1: An overview of the Medical Communications (MC) application before and after the migration.

## I. Introduction

In recent years, there has been a growing interest in cloud computing [1], a technology that enables elastic, on-demand resource provisioning. A concept that is often used together with cloud computing to reduce costs is multi-tenancy [2], where multiple end users make use of a single instance. Without multi-tenancy, the cost savings using cloud computing are minimal for applications requiring continuous availability, as for every client a separate Virtual Machine (VM) instance must be provisioned. This instance must be available at all times, even if it is only used sporadically. In such a case, the cost reductions are limited to a reduced maintenance cost. When during the cloud migration multi-tenancy is added to the application, this problem is mitigated, as all tenants can make use of the same application instance.

In this paper, we will describe the steps needed to add multi-tenancy to a commercial Medical Communications (MC) application, and to migrate it to the Microsoft Windows Azure public cloud environment. A MC application offers various functionalities within hospitals surrounding its central *nurse call* component. A nurse call system consists of physical terminals that are installed within various hospital locations which can be used by patients to call hospital personnel.

Migrating management infrastructure to a cloud environment increases the flexibility of the application, as it is no longer necessary to physically update the servers which were on-premise, making it easier to change the application features used by the client. By also adding multi-tenancy, resources can be shared between different tenants. This makes it possible to reduce costs and to offer features to clients which would previously have been unable to afford the required server hardware. The system setup before migration, and result of the migration are illustrated in Figure 1.

In the next section we will discuss related work. Afterwards, in Section III we outline the use case. Next, in Section IV we will discuss the high-level architecture of the MC application, both before and after the cloud migration. In Section V, we elaborate on the steps required to add multi-

tenancy and migrate to the cloud. This is followed by a discussion of the used approach in Section VI. Finally, we will end with our conclusions in Section VII.

## II. Related work

In [3], an approach for partially migrating applications to the cloud is presented. The approach focuses on identifying components to migrate, taking into account various rules such as performance and security. We similarly focus on a migration to a hybrid cloud, taking into account legal and performance limits. We however discuss the concrete steps needed to migrate a specific application use case.

In [4] a checklist that can be used to determine whether applications are compatible with a chosen Platform as a Service (PaaS) provider is presented. The approach is evaluated by three case studies where a Java and two Python applications are migrated to Google App Engine. Similarly, we focus on how a complex .NET application can be executed on Windows Azure, but more specifically we focus on how the resulting application can be made multi-tenant, increasing cost savings.

Cost savings and other organizational benefits and risks of migration to Infrastructure as a Service (IaaS) are discussed in [5]. We however focus on migration to a PaaS platform, rather than an IaaS platform. Furthermore, we describe how multi-tenancy can be added, making it possible to better utilize individual application instances.

In [6], the migration of an on-premises web application to Windows Azure is described, with a comparison of the application's performance when deployed to a traditional Windows server versus its deployment to Windows Azure. While the cloud migration of a .NET application requires limited effort, Azure has no built-in support for multi-tenancy, so it must be added during the migration process. In this paper, we discuss both the steps needed to migrate a specific application use case, and the steps needed to add multi-tenancy to the application.

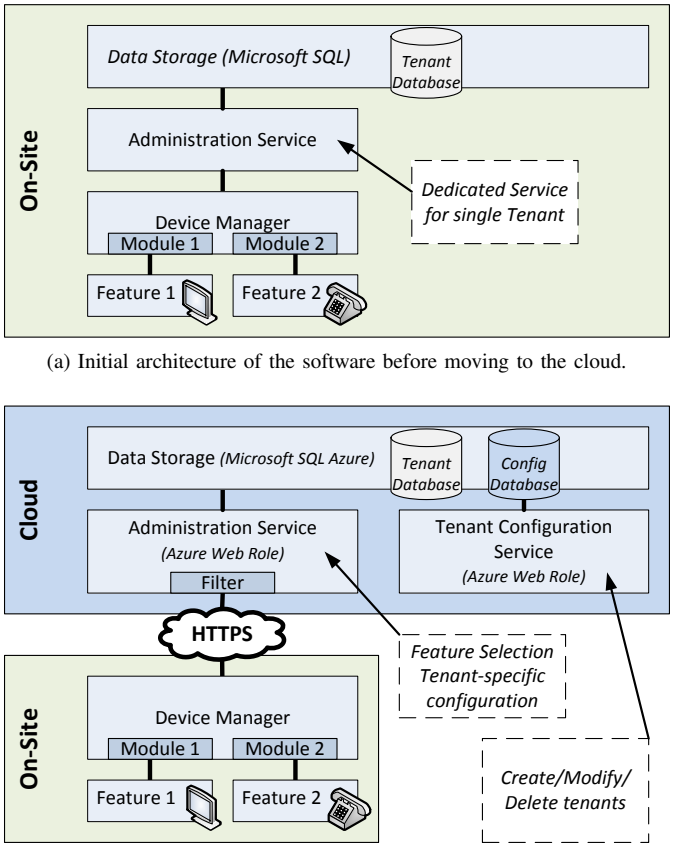
Within our approach, we make use of customization by specification of features with associated code modules, which we have previously discussed in [7] and [8].

In [9], we describe an architecture of a multi-tenancy enablement layer, which amongst others can be used for data isolation, feature management and tenant-specific customizations. We use the discussed approaches in this paper, in which we specifically focus on the architectural changes required to migrate an existing MC application to a multi-tenant environment on Windows Azure.

## III. Use case: medical communications systems

The central functionality of a MC system is *nurse call*. The basic concept of a nurse call system is simple: a call device is located in every room. When a button is pressed on the device, a message is sent to a controller after which nurses are notified of the call.

A nurse call system consists of many different elements, that are installed within a hospital. These elements include amongst others 1) end user equipment installed in the rooms, which patients can use to contact hospital personnel, and terminals used by the personnel; 2) embedded servers, used to communicate between the terminals and management servers; and 3) servers for logging, registration and visualization.




(a) Initial architecture of the software before moving to the cloud.

(b) Architecture after migration to the cloud.

Fig. 2: The architecture of the MC application before and after the cloud migration.

While the center of the MC application is the nurse call system, additional services, such as intercom, video over IP, access control and other health services are offered as well.

Considering the medical use case, the MC application is subject to stringent security and performance constraints, which need to be taken into account when the components to migrate to the cloud are selected.

In this paper, we will describe the steps required to convert a dedicated nurse calling solution, with a historical and proprietary architecture to a cloud based multi-tenant implementation. This is important from the perspective of reducing the IT management cost and risks when applications get more sophisticated, and need to be deployed in an intra- and inter-hospital context.

## IV. Architecture

The MC application consists of two main components: the device manager and the administration service. Figure 2a shows the initial architecture of the application. After migrating the application to the cloud, a new component is introduced, the tenant configuration service. The architecture after migration is shown in Figure 2b.

A feature is a distinct functionality or set of functionalities that can be offered by an application. The concept of features plays a central role in the architecture, as every customer can have its own customized service, including different feature selections. For every feature, the customer currently needs some peripherals installed on-site. Example features include nurse call, voice over IP, video over IP, access control, etc. The device manager, deployed on-site, is responsible for controlling these features. For every feature, the corresponding module is loaded in the device manager.

The device manager communicates with the administration service to know which features should be activated and to get the specific configuration. As the administration service is deployed as Software as a Service (SaaS) in the public cloud, communication between the two should be secured, for example by using HTTPS. The administration service has a user interface to configure all tenant-specific information: which features should be activated, feature-specific configuration, user administration, etc.

All tenant-specific data is stored in the tenant database, running on SQL Azure. Tenant data includes configuration of features, tenant-specific configuration, and other information related to individual tenants. Tenant databases can be colocated into a single SQL Azure instance, to limit the number of SQL Azure instances. The configuration database is a shared database containing general configuration for the different tenants. For example, for each tenant, the connection string pointing to the tenant database is stored in this database.

The tenant configuration service is used to configure the different tenants. It is the only service that has write access to the configuration database. During authentication, the administration service reads the configuration database to get all needed information about the tenant before accessing the tenant database.

The initial application only consisted of two components: the device manager for activating and controlling the features and the administration service with the tenant database. The administration service was designed as a single tenant application, meaning that every tenant would have its own instance of this service. During migration, we decided to make the administration service multi-tenant, and introduced the configuration database and tenant configuration service for managing the different tenants.

## V. Cloud migration

In this section, we will describe the steps needed to migrate a part of the existing .NET application to the public cloud, and to add multi-tenancy to the administration service.

### A. Cloud Migration

**C1: Preparing the application.** The parts of the existing software to migrate need to be selected based on the quality attributes to guarantee the required QoS. Some changes need to be done before moving the .NET web application to the cloud. First of all, the SQL database has to be moved to SQL Azure. As a result, the connection strings should be altered to point to the SQL Azure instance. Next, Azure Web roles need to be added to the .NET project. Once the application is running correctly in the Azure simulator, the project can be packaged and deployed onto Windows Azure [10].

**C2: Impact analysis on client networks.** A side effect of the migration to cloud environments, is that communication between the device manager and the administration service now needs to pass over the internet, and if we want to move the device manager to the cloud in future, even more traffic bandwidth will be needed. Because of this, it is important to perform an impact analysis when client configurations are changed. We have previously covered this in-depth in [11].

### B. Multi-Tenancy

**M1: Introduce multiple connection strings.** As application behaviour can differ for every tenant, the configuration for every tenant must be stored in a tenant database. Each tenant will have its own application data stored in a shared or dedicated SQL server instance. Using shared database instances is cheaper, while dedicated databases lead to a better security, but at a higher cost. To connect to the correct database, a connection string is associated with each tenant. These connection strings will be stored in the central configuration database.

**M2: Add tenant configuration database.** A new database, which we refer to as the *configuration database*, needs to be added to store general information about all tenants. While this database is shared between all tenants, it only contains minimal information, and is only accessed sporadically so it should not become a bottleneck. The connection strings introduced in the previous part should be stored in this database, together with some general tenant information.

**M3: Tenant configuration interface.** Migration to the cloud makes it possible to more flexibly select the application features used by different clients, as the tenant configuration is stored in a separate database, the tenant database. It is however also necessary to create a separate tenant configuration service, which can be used by administrators to quickly change the general tenant configuration.

**M4: Dynamic feature selection.** As explained in Section IV, the device manager consists of multiple modules for the different features, which are dynamically loaded on start-up. These modules can be configured from the administration service. Because every tenant has its own features, the user interface of the administration service is automatically adapted for the different clients based on the tenant configuration.

**M5: User roles and tenants.** Every tenant has its own users and custom roles, stored in the tenant database. These users and roles can be created and modified from the administration service by a user with an administrator role. By introducing multi-tenancy, tenant administrators which can create and modify the different tenants configurations using the tenant configuration service are needed. These tenant administrators will be stored in the shared configuration database and should also be able to access the administration service for every tenant if needed.

**M6: Mitigate security risks.** A major disadvantage of using multi-tenancy is an increased security risk, as by definition multiple tenants use the same application instance. These

risks are mitigated in multiple ways: 1) by implementing URL-based filtering of application requests, taking into account the permissions of the user and tenant; 2) by separating tenant configuration from tenant data; and for cases where this is not deemed sufficient 3) by offering single-tenant instances of specific components at a higher cost.

## VI. Discussion

Moving the administration service to the cloud introduces new opportunities for the MC software. First of all, there is the increased flexibility and elasticity. When the workload on the administration service increases, new instances can be created and deployed in an automatic way. Similarly, when the workload decreases, instances can again be deleted. For new customers, deployment time decreases as there is no need to physically install a new server, and by using Azure's platform, there is also no need to install, configure and manage the guest OS. The hardware maintenance cost is also eliminated as the virtual machines are automatically migrated when the hardware is broken, and the physical hardware is frequently updated.

An additional advantage of the cloud migration is that administration and access become easier, as the software is located centrally on the Azure platform. This ensures all managing server instances can be accessed in a uniform way, and no VPN access is needed for every customer. This ensures software updates and patches can easily be deployed to all instances at once. One disadvantage of the cloud is that the device manager needs a working internet connection to get the tenant configuration. However, in our architecture the device manager only checks the administration service in certain intervals to update the configuration, and can continue to work when there is no internet connection.

Using a public cloud provider comes at a small cost. For running the administration service in the cloud, there is the cost for using the Azure web roles, which is linear with the number of instances, and there is the cost for storing the data in SQL Azure. To minimise this impact and enable cost savings, we modified the architecture of the application to support multi-tenancy. Doing so does come with an overhead, as a number of steps are required to convert the single-tenant application. Most of the discussed steps can however be executed without too many changes to the architecture and code, and within a limited time frame.

By introducing multi-tenancy, administration and maintenance become less complex as the number of instances remains limited, and costs can be reduced [9]. Additionally, the Azure platform has functionality which allows to update all instances at once, without downtime [12]. Note that the service must be deployed to at least two instances for this to work. The changes necessary to enable multi-tenancy come at a one-time cost, but these costs will be recovered on short-term as the number of instances needed will decrease significantly. On the long-term, the introduction of multi-tenancy becomes even more interesting as the number of customers grows.

## VII. Conclusions

In this paper, we described how a large part of an existing medical communications software can be migrated to a public cloud provider. We briefly described the different steps needed to convert the dedicated application to a cloud application, and the steps required to add multi-tenancy to the application. We found that migrating an application to Microsoft Azure only requires a limited number of changes, while the conversion from a single-tenant to a multi-tenant application requires more steps. We ended by describing the advantages of both the cloud and multi-tenancy for our use case, and conclude that in the long term, the benefits of a cloud migration outweigh the costs of implementing the described changes.

## Acknowledgment

This research is partly funded by the iMinds CUSTOMSS[13] project.

## References

[1] M. Armbrust, R. Fox, Armandoand Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds : A Berkeley view of cloud computing," University of California at Berkley, Tech. Rep., 2009.

[2] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao, "A framework for native multi-tenancy application development and management," in *9th IEEE International Conference on E-Commerce and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE 2007.*, 2007, pp. 551 – 558.

[3] M. Hajjat, X. Sun, Y.-w. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani, "Cloudward bound: Planning for beneficial migration of enterprise applications to the cloud," in *Proceedings of the ACM SIGCOMM 2010 conference*, 2010, pp. 243–254.

[4] Q. H. Vu and R. Asal, "Legacy application migration to the cloud: Practicability and methodology," in *2012 IEEE Eighth World Congress on Services*. Ieee, Jun. 2012, pp. 270–277.

[5] A. Khajeh-Hosseini, D. Greenwood, and I. Sommerville, "Cloud migration: A case study of migrating an enterprise IT system to IaaS," in *2010 IEEE 3rd International Conference on Cloud Computing*. IEEE, Jul. 2010, pp. 450–457.

[6] P. J. P. da Costa and A. M. R. da Cruz, "Migration to Windows Azure - analysis and comparison," *Procedia Technology*, vol. 5, no. 0, pp. 93 – 102, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2212017312004422

[7] H. Moens, E. Truyen, S. Walraven, W. Joosen, B. Dhoedt, and F. De Turck, "Feature placement algorithms for high-variability applications in cloud environments," in *Proceedings of the 13th Network Operations and Management Symposium (NOMS2012)*, 2012, pp. 17–24.

[8] H. Moens, E. Truyen, S. Walraven, W. Joosen, B. Dhoedt, and F. D. Turck, "Developing and managing customizable software as a service using feature model conversion," in *Proceedings of the 3rd IEEE/IFI Workshop on Cloud Management (CloudMan)*, 2012, pp. 1295–1302.

[9] S. Walraven, E. Truyen, and W. Joosen, "A middleware layer for flexible and cost-efficient multi-tenant applications," in *Middleware 2011*, 2011, pp. 370–389.

[10] D. Betts, S. Densmore, M. Narumoto, E. Pace, and M. Woloski, *Moving Applications to the Cloud on Microsoft Windows Azure*. Microsoft ; O'Reilly distributor, 2010.

[11] H. Moens, E. Truyen, S. Walraven, W. Joosen, B. Dhoedt, and F. De Turck, "Network-aware impact determination algorithms for service workflow deployment in hybrid clouds," in *Proceedings of the 8th Conference on Network and Service Managment (CNSM2012)*, 2012, pp. 28–36.

[12] (2012) Overview of updating a Windows Azure service. [Online]. Available: http://msdn.microsoft.com/en-us/library/windowsazure/hh472157.aspx

[13] (2012) CUSTOMSS: CUSTOMization of Software Services in the cloud. [Online]. Available: http://www.iminds.be/en/research/overview-projects/p/detail/customss