Moving an Application to the Cloud – An Evolutionary Approach

Alexander Gunka
BOC Information Systems GmbH
Operngasse 20B
1040 Vienna, Austria
+43-1-905 10 81 - 0
alexander.gunka@boc-eu.com

Stepan Seycek
BOC Information Systems GmbH
Operngasse 20B
1040 Vienna, Austria
+43-1-905 10 81 - 0
stepan.seycek@boc-eu.com

Harald Kühn
BOC Information Systems GmbH
Operngasse 20B
1040 Vienna, Austria
+43-1-905 10 81 - 0
harald.kuehn@boc-eu.com

ABSTRACT

When planning to move a legacy style application to the cloud various challenges arise. The potential size and complexity of such a project might especially discourage small or medium companies trying to benefit from the advantages the cloud promises. In addition, the field they have to address is still young and very dynamic and related technologies are rapidly changing.

Based on on-going work in the context of the MODAClouds EU project, this paper describes an evolutionary, iterative approach to accomplish the migration of an existing application to a cloud based environment. Model based techniques are used to support the steps of this transition process by providing a baseline for the development of appropriate deployment architectures and the selection of suitable cloud providers. In addition they provide necessary abstractions in order to be less dependent on a specific technology stack or cloud provider.

In order to show how we imagine the developed approach to be applied in practice we describe an existing traditional 3-tier application based on the meta-modeling platform ADOxx and how it could be moved to the cloud from the perspective of a medium-sized software manufacturing company.

Categories and Subject Descriptors

C.0 [**General**]: *Modeling of computer architecture, System architectures*

C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *distributed applications*

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *Restructuring, reverse engineering, and reengineering*

Keywords

Cloud; Cloud deployment; Evolutionary Approach; ADOxx

1. INTRODUCTION

Cloud computing offers a range of novel opportunities, especially for small and medium-sized enterprises (SME). In particular SMEs can now exploit the economies of scale offered by cloud computing, giving them affordable access to sophisticated computing resources previously available only to large companies [1]. However, for SMEs who want to move parts of their applications to the cloud taking high efforts to redesign existing mature applications, while considering all kinds of cloud related aspects, can be a risky step.

This paper contributes to help solving this issue by describing an evolutionary strategy to migrate an existing application to the cloud. The main steps of this transition are 1) porting the application to an IaaS, 2) adding load balancing on the presentation and business logic layer, and 3) partially moving the app to a PaaS. Along with each of the steps described multi-cloud strategies will be considered in order to be able to mitigate risks by employing redundant, independent systems and to ensure service proximity for customers at different locations. Furthermore decision support is needed to help identifying the best-matching cloud platforms for each step. The MODAClouds EU [2] project is expected to provide tools and methods and modeling techniques which will help to address these issues.

The evolutionary approach is first described in a general way and later applied to an existing application developed by the authors' company.

2. SCENARIO

BOC is a medium-sized software manufacturing and consulting company with locations in different European countries. With the Management Office suite BOC offers a comprehensive tool set that supports the introduction and implementation of standardized management approaches in the areas of Strategy and Performance Management, Business Process Management, Supply Chain Management, and IT Management. The company is involved in the MODAClouds [2] EU project as a case study partner. The main objective of the provided case study is to move an instantiation of a process modeling tool based on our ADOxx meta-modeling platform to the cloud.

The case study application follows a classical 3-tier architecture supporting relational database systems at the database layer, a Windows application server and a web application based on Java Servlets and hosted inside a JEE Web Container (typically Apache Tomcat) (see Figure 1). The web application provides browser-based access to business process modeling including graphical online modeling and the creation of various reports.

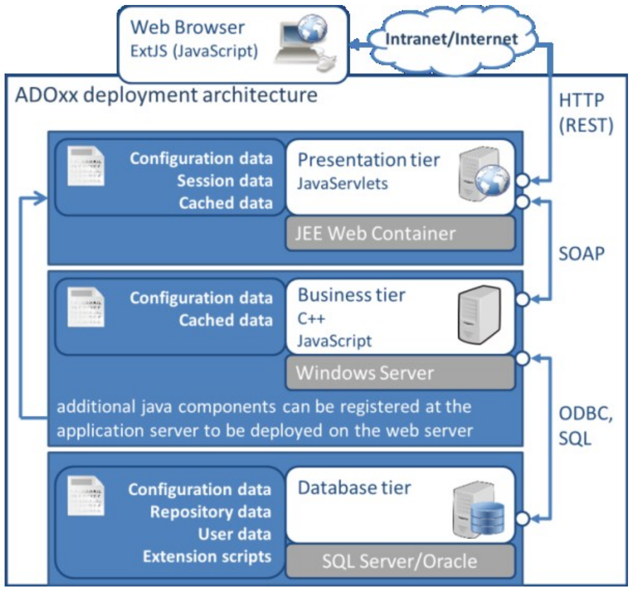

Figure 1 - ADOxx 3-tier architecture

Currently the application is typically installed on site leaving the provision and maintenance of the underlying (virtual) infrastructure and system environment to the user organization. The deployment and setup process includes several manual steps and is typically supported by technical staff either directly on site or using remote access to the customer's infrastructure.

To provide test installations of the software to the customers BOC uses virtual machines on dedicated servers in its in-house datacenter. By providing a standardised environment for the software, it is possible to minimize platform related issues during operation. Owning and controlling the system also allows to deploy, monitor, and maintain the software in a more efficient way. In addition, the benefit of economies of scale can be already exploited at this stage e.g. by using one database for multiple customers and scaling the database according to the needs.

Due to the already mentioned advantages BOC gradually wants to extend its hosting offers but does not want to invest in infrastructure and operations just for the sake of being able to handle significant workloads in the future. The clouds possibilities seem to be what BOC is looking for.

While moving the application to the cloud, in addition to the topics mentioned before, a number of other objectives related to the scenario can be identified. These objectives include:

- Extending our present hosting solutions gradually and moving from a testing environment towards production without the need for large initial investments.
- Minimizing deployment efforts by using a standardized environment and automated deployment.
- Benefiting from economies of scale by providing one elastic solution for multiple customers.
- Gradually moving our web based application towards a pay as you go Software as a Service solution.
- Monitoring and maintaining our solutions in a more efficient way.
- Defining high availability SLAs based on SLAs of the underlying infrastructure.
- Defining and testing disaster recovery scenarios.
- Ensuring privacy of the customers' data.
- Considering multi-site deployments in order to be able to provide service proximity.
- Obtaining infrastructure requirements for a given user count – needed for accurate cost calculation.
- Being able to meet these objectives in a cloud provider independent way to avoid vendor or technology lock-in.

Regarding all those objectives being able to accomplish at least some of them already in rather short term is regarded as crucial. Without having a large development team available from the beginning, which has been trained in all cloud specific aspects of application development, and considering the high dynamics of the field a policy of rather small steps seems to be most suitable. Above all, BOC's present internal software development processes already follow agile principles. For all those reasons the company favours an iterative, evolutionary approach to accomplish the migration. The first steps include choosing a particular cloud provider constituting the "best match" for our existing application and deploying it in the cloud as fast as possible. Further steps reconfigure and rearrange the application to take better advantage of additional cloud features.

3. BACKGROUND

Several approaches which support describing and analysing an application and its architecture regarding cloud related aspects and migrating applications to the cloud already exist.

The REMICS project [3] aims at supporting the migration of legacy systems to the cloud based on a Service-Oriented Architecture with a set of model-driven engineering tools and methods. REMICS seeks for overcoming several gaps identified by analysing existing methodologies for developing service-oriented systems from scratch by integrating a set of developing methods, languages, transformations and tools "in an agile, model-driven service-oriented methodology for modernizing legacy systems".

The MODAClouds project [2] will leverage results of the REMICS project and extend them by providing an additional abstraction level for cloud provider independent modeling by developing a domain-specific language named MODACloudML. In addition, MODAClouds aims at "providing methods, a decision support system, an open source IDE and run-time environment for the high-level design, early prototyping, semi-automatic code generation, and automatic deployment of applications" [2] to the cloud. One of the main innovations targeted by MODAClouds in this context is to provide techniques for migration, data mapping, and synchronization among multiple clouds.

4. RELATED WORK

According to Gartner [4] organizations seeking to move applications into the cloud have the following five options: Re-host on infrastructure as a service (IaaS), refactor for platform as a service (PaaS), revise for IaaS or PaaS, rebuild on PaaS, or replace with software as a service (SaaS). The options differ in the targeted cloud environment (IaaS or PaaS) but also in the degree of efforts taken in order to adapt the application to exploit cloud capabilities.

For Gartner *re-hosting an application on IaaS* means to take an application without any changes of the implementation, change its configuration according to the target infrastructure and redeploy it to a cloud IaaS environment. Gartner highlights this option can provide a fast migration solution but that the primary disadvantage is that benefits like scalability will be missed compared to applications which have been explicitly designed to exploit these cloud characteristics.

Gartner describes *refactoring an application for PaaS* as running an existing application in a PaaS environment with only minor adaptations. Preserving the applications original architecture helps reducing adaptation efforts. On the other hand with this approach the potential gain in elasticity is limited depending on the design of the application.

Especially older "legacy" applications are not designed in a way that would allow them to really benefit from cloud characteristics like the possibility to leverage distributed computing resources. In order to make better use of the cloud's elasticity and hence to lay the foundation of a possible later pay-as-you-go SaaS solution, Gartner suggests *revising the application for either IaaS or PaaS* which means to modify and extend the existing code base to support legacy modernization requirements as also mentioned in [3].

Another option Gartner mentions is to *rebuild* and re-architect an existing solution from scratch targeting a *PaaS* environment in a "cloud-optimal" way. Since in this case optimization is very likely to be done taking into account only a limited set of cloud providers, measures have to be taken to avoid lock-in. MODAClouds [2] will address this challenge by providing an abstract layer for cloud provider independent modeling.

Finally Gartner mentions that from a software customer's point of view *replacing* a traditional application *with an existing SaaS solution* might be another option.

While Gartner focuses on choosing one out of several alternative options this paper points out how these options can be seen as steps of an evolution where one step builds on the other. The fact that MODAClouds will provide support for both PaaS and IaaS deployment is expected to help accomplishing smooth transitions when moving from one step to the other.

Another article [5] introduces a phase-driven step-by-step strategy for migrating applications to the cloud. After the *cloud assessment phase* (1) a *proof of concept* is performed (2). Then *application data* (3) and the *application* itself (4) are migrated. Auto-scaling, edge caching, auto-recovery, and elasticity are considered in the *leverage the cloud* phase (5). Finally the application is *optimized* in terms of cost savings (6). Even though the article mainly focuses on AWS the approach might be somewhat generalizable. During phase (6) other services might be introduced in order to increase efficiency. At that point, in contrast to the approach introduced in this paper, the article only focuses on additional AWS features and services and does neither take into account adapting the deployment architecture nor the possibility of moving to another cloud provider.

5. EVOLUTIONARY APPROACH FOR CLOUD MIGRATION

This section discusses arguments for moving an existing application from an on-premise environment to the cloud and suggests a step-by-step transition approach that shall guarantee business continuity during the migration process.

5.1 Motivation

The first argument pro cloud and especially pro multi-cloud is *availability*. While a standard setup of a service implemented on 'bare metal equipment' – one server with hardware maintenance contracts and a backup/restore strategy hosting the application – is well suited for a non-critical service, the availability that can be guaranteed for such a system is not sufficient for business critical applications or SaaS business cases with corresponding SLAs. In order to satisfy the availability requirements in the on-premise environment, major capital expenditures must be considered as all single points of failure need to be eliminated. A multi-cloud based infrastructure can introduce redundancy of all components with much lower additional costs as usually only utilized resources are charged by the cloud vendors. Another important motivation for cloud deployments is *scalability*. As usually the utilization of an application is not constant over time, the infrastructure must provide sufficient performance for covering high peak scenarios. Cloud infrastructures promise to provide on-demand horizontal scalability by spawning additional nodes when needed and to tear them down in times of lower loads. Combined with the usual pricing model where the customer is charged for the time where computing instances are running, this not only provides the required technical elasticity, it also delivers an infrastructure with the desired *cost efficiency*.

Obviously these facts are a clear indicator that a company that decides to take the step to provisioning its software to its clients in an SaaS business model should do this technically by *migrating the service to the cloud* in order to be ready to react to load changes seamlessly without having to pay for resources not utilized most of the time.

5.2 The Migration Process

What does "migrate to the cloud" mean? This heavily depends on the application's architecture. In order to benefit from the cloud's elasticity a software system must be designed according to certain cloud patterns. If it has a web frontend, the web application must be ready for *load balancing*. That means that management of state must have been considered with load balancing in mind. If the application relies on persistent state held in a *database*, which usually is the case for typical enterprise applications, then there should be an abstraction layer allowing for plugging in different database engines. If the application consists of multiple modules that leverage *message oriented middleware* for asynchronous processing again an abstraction will be needed so that it can be integrated with the various message broker implementations.

Of course, for an application that has not been developed with a cloud service portfolio in mind, *challenges* arise in terms of architectural change. The aforementioned design requirements result in major software changes, which makes for efforts and risks that need to be evaluated and quantified.

Based on the results of this *assessment* a strategy for the actual migration has to be developed. In many cases it will turn out that a big bang approach encompassing the change of infrastructure and redesigning the application's architecture in order to get out the maximum flexibility from the cloud deployment introduces too many *risks* that cannot be properly handled and may affect the *business continuity* for services related to the considered application. Especially if the application provides a service central to the company's business a restrained strategy will be chosen.

5.2.1 Transition to IaaS

In the given situation the first decision in favour of an evolutionary service migration should be to separate development related issues from operations related ones. This can be accomplished by moving the whole application as is to an *IaaS* instance provided in the cloud as suggested as the option with the lowest risk by Gartner [4]. This does not sound like a very challenging task; however, there are several infrastructural aspects that need to be considered here and the multi-cloud approach results in additional complexity (as introduction of redundancy usually does).

As the first step the *sizing* has to be done. The natural approach for this is to analyse the relevant parameters in the on-premise environment and then pick an IaaS offer that meets these requirements. At this point MODAClouds' decision support tool should provide assistance for selecting appropriate providers. Usually the *performance parameters* for determining the best matching IaaS offer will be *CPU power*, *RAM size*, (virtual) *disk size*, *disk IO performance* in terms of IOPS, latency and bandwidth, as well as *network* bandwidth and latencies for internal traffic within the cloud as well as to/from the internet on the public interfaces and the replication paths to the multi-cloud peers.

Aside from these performance parameters other peripheral operations aspects need to be considered. These aspects are:

- Snapshotting options
- Backup options
- RTO in case of file system corruption
- Vertical scaling capabilities
- Firewalling
- Availability of private networks
- Availability of VPN connectivity
- Privacy of data stored in the cloud
- Monitoring interfaces
- Suitability for data replication in multi-cloud setups

Once the choice has been made based on the required features, the contract conditions and costs, the technical preparation phase can start. The operations staff will verify whether the infrastructure provided by the cloud vendors really provides the *performance* that has been calculated by MODAClouds' decision support tool. Benchmarks run by independent analysts (see [6] and [7]) have shown that even if one cloud provider offers one product in different locations the performance of the provisioned systems may differ.

After the performance of the provisioned infrastructure has been validated, *firewalls* need to be configured to provide the desired isolation. Here some vendors offer centralized systems others rely on the operating system's packet filtering capabilities. The complexity of the firewall setup varies depending on the application's communication requirements with other services. In some cases it may just open inbound ports for access to the application's web frontend plus a management port like ssh, however, in a multi-cloud setup an IPSec tunnel connecting the private networks is obligatory.

As the next step *backup* has to be set up and *restore* tests must be conducted on all involved multi-cloud sites. For an application that is expected to run 24x7 off-line backup is not an option, therefore generating an application consistent backup by taking the whole machine down is not possible. In most cases a crash consistent backup will be performed utilizing the cloud provider's snapshotting functionality and transferring the snapshots to a secondary storage. At Amazon AWS for example backing up an EC2 instance data would require taking a snapshot of the underlying EBS volume and uploading the snapshot to a S3 bucket or Glacier vault.

Having the infrastructure including backup in place the actual deployment of the application stack can be started. In most cases a *database tier* is required as the persistence layer. Depending on the application's nature and its transaction patterns it might be necessary to equip the virtual machine with some high IO block storage. Here again the MODAClouds decision support tool shall provide information about the availability and the guaranteed performance of such storage tiers for the managed cloud providers. Of course also here the actually delivered performance must be validated by means of IO test tools like iometer [8] or fio [9].

As the final deployment step the application itself is installed on the prepared infrastructure. At this point, having all operational components configured, *monitoring* and *alarming* needs to be set up and tested. In most cases an existing monitoring solution can be taken as is because the application itself has not been changed and the environment prepared for it does not really differ from the on-premise datacenter, however, integrating some additional performance indicators might be a good idea for providing long term performance data.

With the application setup finished, testing can be done. Nowadays with agile patterns like behaviour driven development probability is high that the company already has a good coverage of *scenarios* as well as massive *load tests* available in an automated test suite. This is a very good starting point for verifying the new platform's functionality. Nonetheless *manual testing* will be required as well in order to be able to assess the application's responsiveness and the overall user experience. Additionally the multi-cloud take-over scenario must be tested with special attention being paid to split brain potential and data replication lag.

Assuming that the tests have not yielded any issues, the migration of *data and clients* may be started. Based on the structure and the separation of customers' data it might be possible to perform a one-by-one migration. Of course, if the application is for example some kind of collaboration platform cutting away portions of data is not an option and the whole customer base must be migrated at once. In any case, if the amount of data is large and the maintenance window for transferring it from the on-premise datacenter to the cloud environment would not be accepted by the clients, a backup and restore procedure can be applied consisting of restoring a full backup set at one point in time and then right before switching the traffic applying only the differential backup on top. Of course being able to switch back to the on-premise environment is an essential risk management measure. In most cases this just means that the transactions that have been committed on the cloud infrastructure must be replayed on the in-house database and the traffic must be switched back.

Having gone through all these steps brings the company to the point where they have their application ported to the cloud so the operations team does not need to deal with hardware availability and maintenance in the corporate datacenter any more. If *scaling* becomes an issue in this stage either *vertical scaling* provided by some of the cloud vendors, e.g. ProfitBricks [10] can be considered, or alternatively setting up *additional identical machines* and distributing the clients might be an option. But again, if the application relies on data exchange between the clients the database cannot be partitioned in this way. So it would be necessary to launch the new instances without private databases and interlink them with the database on the first instance. A private network within the cloud provider's infrastructure then is a must as the SQL traffic shall by no means be transported over any public network resource.

Nowadays this is definitely not considered an elegant way to scale an application hosted in the cloud.

In summary the steps required for moving an application to IaaS are:

1. vendor selection
2. sizing
3. performance verification
4. network configuration
5. backup/restore verification incl. disaster scenario
6. database setup
7. monitoring/alarming

8. automated and manual tests including multi-cloud fail-over
9. migration of data and customers

5.2.2 *Introducing load balancing*

The state of the art technique to accomplish application scaling is to use load balancing, at least for applications served on the web. So, having successfully moved the application to a (multi-)cloud hosted infrastructure and usually after having gained some operations experience and re-adjusted monitoring and maybe some flows in the application time has come to make the next step and let the developers contribute to a software architecture that is prepared for load balancing.

The major decision with respect to load balancing is *handling of non-persistent state*. Basically, there are four options available:

- avoiding non-persistent state
- letting the client manage the state and resubmit it with every request that yields a modification of the persisted state
- replicating the state between all nodes that form the load balanced tier e.g. by means of HTTP session replication within a web container cluster or a distributed, replicated cache solution like Infinispan [11] in applications that maintain additional state apart from the users' HTTP sessions
- utilizing a message oriented infrastructure which provides a centralized service with the purpose of distributing information among the load balanced tier members

If the software cannot provide any of these, persistent load balancing is required. This ensures that subsequent requests related to one session are always forwarded to the same worker node that has created the client's session. The drawback of this approach is that the load of the worker nodes is only taken in account for requests outside sessions and that a node failure results in lost session data. Furthermore, if persistent sessions are used, the developers must make sure that other state (held by singleton instances) is not relevant to other nodes as the singletons lose their uniqueness once there are more than one worker nodes running.

After having either adapted the software for sharing state among nodes or decided to go with persistent sessions the load balancer can be set up. Usually the cloud provider will provide a *highly available HTTP load balancer* that supports persistent sessions with configurable session token (cookie name or GET parameter name). MODAClouds' decision tool shall provide this information for managed cloud providers. In case a provider has no load balancing offerings, of course, a solution based on standard OSS tools like nginx [12] can be set up, however, this requires a high availability solution on IaaS and results in additional maintenance efforts.

With the load balancer in place one major step has to be made before additional workers can be spawned. Assumed that the infrastructure provides low latency private networks – this actually is a must-have for scaling systems – a dedicated, *centralized*, well sized *node* for providing the *database* service to all worker nodes is set up. This should be easy to do as long as the original database has been set up on dedicated block storage. This block storage simply needs to be re-assigned to the newly created database node after the DBMS has been installed there. Then only the connection for the database has to be reconfigured for the application.

Once the database has been successfully moved to its dedicated virtual machine, *additional instances* of the application can be created by means of the cloud provider's portal – MODAClouds' runtime could provide an abstraction for this – and registered with the load balancer. At this point sophisticated *performance monitoring* is essential to provide information about system components running into limits. Especially the load of the database machine has to be analysed in terms of IO and CPU as well as database specific performance indicators like cache hit ratio and query times.

The system is prepared for elastic scaling now, where nodes can be added on demand and removed when load goes down. The limitation in most cases will be the database backend as a standard relational database system does not provide the horizontal scalability that would be required push the limits here and the virtualization layer does not provide the computing power available with 'bare metal servers', however, a vertically scaling IaaS product like ProfitBricks [10], MD5 Cloud [13] or Global Technologies Company [14] used for hosting the database can be of some benefit.

In short introducing load balancing requires the following steps:

1. adaptation of state handling in the application for load-balancing
2. centralization of database on dedicated IaaS
3. deployment of additional presentation- and business tier IaaS instances
4. load balancer configuration
5. testing

5.2.3 *Moving modules to PaaS*

At this point the application environment has been taken to the most flexible point regarding scalable deployment that is possible without major design changes. After having eliminated all flaws that might have been introduced by the changes made for load balancing the utilization of *cloud provided services* should be considered as this is where operations efforts can be reduced and transparent scaling can be achieved.

As an example the web tier of a JEE application could be moved to *web container based PaaS* product like Amazon Beanstalk [15], Google App Engine [16] or Heroku [17] that provides clustering including HTTP session replication load balancing and automatic resource management. The business logic tier of the application implemented by means of EJBs could be handed over to *EJB container based PaaS* like OpenShift [18] or Jelastic [19]. This step is expected to introduce some challenges as JEE 5 and 6 do not provide any cluster-wide singleton specification. However, since singleton is a very common design pattern in business logic tiers, most application will use vendor specific extensions to achieve their centralized task and data management. In order to make the EJB artifacts cluster-enabled this has to be accomplished by other mechanisms like distributed replicated data grids, e.g. Infinispan [11].

Another aspect of the transition to a distributed deployment is that EJB clients will no more be able to access the session beans through their local interfaces. Instead the standard case will be *remote invocation* which has an impact on performance – on one hand because of the network round trip, on the other hand because of pass-by-value arguments with marshalling and un-marshalling instead of pass-by-reference method arguments. To cope with this some algorithms will require adjustments replacing many "small" remote calls with a call to a specialized method encapsulating bigger parts of the algorithm.

As for the *database* two strategies for going PaaS can be considered:

- Leverage a cloud provided RDBMS service
- Migrate to a cloud provided NoSQL service

Cloud provided RDBMS services such as Amazon's RDS usually offer QoS guarantees in terms of I/O performance and switching to a product with higher load capabilities is seamless. MODAClouds' decision support tool should consider this feature to provide valuable information on database deployment.

If the application does not rely on transactions encapsulating more than one object changing in the database and the consequences of the CAP theorem [20] are acceptable one of the available *NoSQL* products could be considered as the meanwhile quite mature systems provide high scaling capabilities, both in terms of data storage and in data processing. Systems like HBase [21] and Hadoop [22] on top HDFS [23] for example provide scalability over thousands of servers favouring local storage over SAN. This results in a highly efficient and cost-effective solution with configurable redundancy. If processing of collected data is part of the application Hadoop will provide distributed processing of the collected data taking data locality in account which results in reduced network traffic and higher performance.

Here are the steps required when leveraging PaaS infrastructure:

1. adaptation of software eliminating dependency on container vendor
2. adaptation of software for PaaS based storage
3. adaptation of software eliminating remote calls with high overhead caused by remote invocations
4. transition of the database from IaaS to a scalable RDBMS service (database as a service)

6. THE EVOLUTIONARY APPROACH APPLIED TO ADOXX

The ADOxx platform is a meta-modeling-based development and configuration platform for implementing modeling tools. It is the technological basis of the BOC Management Office. Its hybrid architecture has been designed for the rich client scenario and the web scenario both bound to the same relational database. The rich client deployment provides a fully-fledged desktop application leveraging either a central database in collaborative environments or a local database in case of single user installations.

This case study focuses on the web scenario implemented by means of the three tier application architecture described in Figure 1. Currently the scenario is hosted in the on-premise datacenter and the absolute need for a low risk strategy when transferring the service to the cloud makes it an obvious candidate for the evolutionary approach introduced in the previous section. While the final goal is to leverage as many cloud patterns as possible resulting in higher availability, elasticity and cost effectiveness, it is not considered a viable approach to do it all at once.

6.1 ADOxx on IaaS

As ADOxx already is layered according to Figure 1 and also operated with a dedicated database server the move from on-premise to cloud infrastructure differs slightly from the approach proposed in section 5.2.1. The requirements for the infrastructure provided on each of the multi-cloud peers are:

- Two small Windows instances for the Windows active directory with private IPv4 addresses.
- One medium sized Windows instance for the web tier and the business logic tier with a public IPv4 address for traffic to/from the internet and a private IPv4 address for database and active directory traffic.
- One high I/O Windows instance with sufficient CPU resources (vertically scalable if possible) for MS SQL Server. This instance shall have no public IP. The block storage backing the database shall be dedicated and equipped with I/O guarantees. Striping over multiple volumes will be considered.
- One smaller instance running Linux for monitoring purposes.
- An IPsec connection available to connect the in-house datacenter with the private network of the cloud installations and another one for the multi-cloud peer connectivity.
- Data located in Europe
- Operated by a European company

The MODAClouds decision support tool is expected to list cloud providers matching these requirements and BOC will evaluate the offerings from this list.

BOC will then contract with the chosen provider, and set up the basic infrastructure by means of the MODAClouds runtime. The operations team will then harden the operating systems regarding security, set up the monitoring system and configure the RDBMS. Next the backup for the operating systems and database will be established and restore tests will be performed.

At this point the infrastructure for deploying the application will be in place, so both the business logic tier and the web tier can be installed. A *windows installer package* will be used for this task as it provides all necessary steps in terms of configuration and initialization of the whole stack. Finally the monitoring system will be adjusted to provide full monitoring for all application layers. This procedure has to be performed for every of the current deployments reflecting the various product configurations in use. The installation step is completed by configuring reverse proxy mappings for standard HTTP and HTTPS to the ports locally bound by the installed web containers.

The QA departments will then run automated penetration tests and round out then with interactive smoke tests to verify correct and smooth operation of the deployed products. Operations will then verify correct behaviour and data consistency after a multi-cloud fail-over. After this quality gate has been successfully passed the client migration will be triggered.

Groups of clients of the BOC online services will be informed about the maintenance window needed for running an application level *export* of their respective *data*, transferring it to the new infrastructure and importing it there. They will also receive the *new DNS name* which shall be used for accessing the services. This is needed because not all clients will be migrated at once and also a DNS switch would not work reliably due to DNS cache lifetime violations. In parallel with the import of the data the active directory entries of the clients will be created in the cloud infrastructure based on the directory contents of the on-premise catalogue. During the whole process the services will still be available in the old infrastructure albeit in read-only mode, of course. After the import has been finalized and the flawless operation confirmed by quality assurance the clients will be granted access to the new infrastructure.

6.2 Load balancing ADOxx

After successful migration of all online service clients to the new platform, availability and scaling will be tackled by introducing load balancing on the presentation tier and the business logic tier. The aspect of managing non-persistent state will be handled by means of *sticky sessions*.

For the session between the client and the web application this will be done by configuring the corresponding session cookie as the persistence token of the load balancer. The drawback of being bound to one web application instance within one user session will be accepted as the already planned distributed state management will not yet be available at that time.

Load balancing between the business logic nodes exposing SOAP interfaces will be a little bit more complex. The nodes will be grouped to *QoS groups* representing highly interactive services on one hand and off-line processing tasks on other hand. The different groups will be accessible through different endpoint URIs. The nodes will also maintain sessions and set the corresponding cookies in their responses. The SOAP clients (i.e. web application instances) will then be able to decide for every request whether or not they rely on state held in the business tier and which service category (QoS group) the actual request belongs to.

For the load-balanced setup the Windows image holding the bundle consisting of reverse proxy, presentation tier and business tier needs to be duplicated. For this purpose a "gold master" image should be created from the existing instance and re-used with every extension step.

MODAClouds run-time is expected to provide support for this task as automatic scaling might be necessary in the deployment engine anyway.

6.3 Partial move to PaaS

Starting with the web tier transition from a (potentially clustered) web container hosted on an IaaS platform to a web container PaaS product promises transparent scaling and reduced operations efforts for the web containers, the underlying IaaS instances and the load balancer configuration. It follows Gartner's suggestion [4] for revising the application for PaaS.

What adaptation is necessary in order to make the ADOxx based web application PaaS ready? One task is definitely to solve the management of non-persistent state so that session stickiness for load balancing is not required any more. This is necessary as sticky sessions are not always available with PaaS offerings and of course *stateless nodes* offer better robustness compared to their stateful counterparts. Another change that might become necessary is the removal of any file based *persistence*. Whenever persistence is required a central database should be considered. In case of a store that is only relevant for the node itself a key/value or blob store could be considered. Alternatively a document based NoSQL service like MongoDB could be the right choice. Therefore the application must be analyzed spotting usage of local storage and adapted accordingly. As this refactoring pattern is expected in most web applications MODAClouds should provide an abstraction with vendor specific code generation for configuring and accessing the store. Finally all container runtime specifics must be eliminated. The result must be a Java EE 1.5 compatible artifact relying only on Java Servlet 2.5 and Java Server Pages 2.1 specifications. If a relational database is used an abstraction layer like Hibernate [24] shall be introduced.

With the web tier restructured according to the PaaS requirements the *deployment process* must be adjusted. As this part differs from vendor to vendor varying from git push to specialized APIs among others, MODAClouds shall provide an abstraction for this. Nevertheless this means that the MODA deployment approach must be integrated into BOC's continuous integration and release processes.

While the transfer of the presentation tier to PaaS seems very rewarding for the business logic tier the situation is different because of the technology used. It is written in C++ and deployed as a windows service controlled by the service control manager. Currently there are no PaaS products targeting this kind of application other than MS Azure worker role which actually is a Windows IaaS instance. Porting the whole tier to CLR in order to open the doors to .NET PaaS vendors is not an option as the application is neither in early state where such a change would be acceptable nor is it to be considered legacy where a rewrite would be justified. Therefore the *business tier* will be kept deployed on Windows *IaaS*.

Both, the load balanced IaaS architecture and the PaaS architecture are expected to run into limits with a central database hosted on an IaaS instance. With reduction of operations efforts and improved scalability in mind a migration to a *RDBMS PaaS (DBaaS)* provided by the cloud vendor is desirable. For the ADOxx business tier this means that an adaptation of the data access layer is required to match the used RDBMS. Utilizing the existing ODBC connector with vendor specific SQL statements is most probably the best matching solution as the technology can be considered mature and general and should be available for all major database services. A migration to NoSQL technologies does not seem to be an option now, as the business tier heavily relies on referential integrity and transactions with massive changes made in the object repository that need to be consistent. However, with the introduction of new functionality, e.g. the extensions of the analytical engine with graph based algorithms utilization of NoSQL engines as a supplement to the RDBMS store is considered and MODAClouds' abstraction is expected to provide modules for attaching to these services in the cloud.

6.4 Proposed target architecture

Figure 2 visualizes the steps an evolutionary migration of ADOxx to the cloud will go through. Starting with an on-premise setup on two servers the existing deployment artefacts will be moved to IaaS instances that are a drop-in replacement of the servers. In a second step cloud-provided load balancers will be applied to scale over IaaS instances of the presentation and business tiers. Finally, the presentation tier will be moved to a PaaS environment and will replace the IaaS database with a DBaaS (database as a service) provided by the cloud service provider.

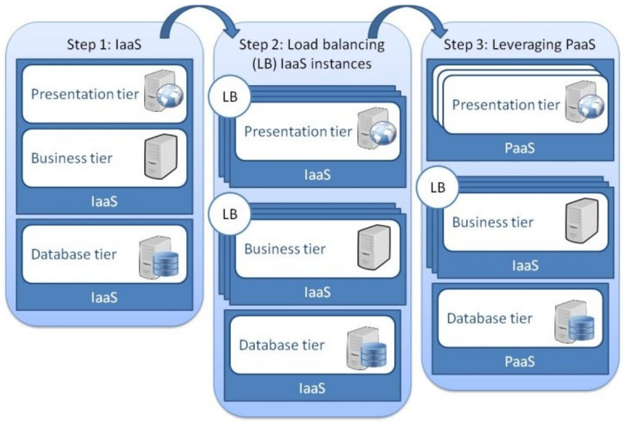

Figure 2 - Evolutionary steps migrating ADOxx to the cloud

7. SUMMARY AND OUTLOOK

Companies who consider moving existing applications to the cloud are confronted with a number of strategic decisions to be made. Business risks which are introduced when changing an existing system have to be carefully considered. An evolutionary approach as it has been proposed in this paper can help to reduce risks by only taking one step at a time. Each phase of the transformation process described adds additional benefits by better exploiting advantages like scalability, availability, and cost efficiency the cloud provides. During this process model based techniques will help ensuring consistency during the transition from one phase to another and reducing the risk of lock-in to a specific cloud provider or technology. The transition approach introduced here will be verified during the migration of BOC's ADOxx products from an on-premise environment to a multi-cloud infrastructure as a case study within the scope of the MODAClouds project. As a preparation an additional risk assessment will be conducted for each of the steps mentioned in this paper and the findings will be contributed to MODAClouds' decision support tool.

8. ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n° 318484 (MODAClouds).

ADOxx is a registered trademark of BOC Information Technologies Consulting AG.

9. REFERENCES

[1] Microsoft, "European SMEs in the Cloud" [Online]. Available: http://www.microsoft.eu/default.aspx?tabid=231. [Accessed 16 January 2013].

[2] MODAClouds, "MOdel-Driven Approach for design and execution of applications on multiple Clouds" 2012. [Online]. Available: www.modaclouds.eu. [Accessed January 2013].

[3] M. Parastoo and S. Thor, "Software Engineering Challenges for Migration to the Service Cloud Paradigm - On-going Work in the REMICS Project" July 2011. Available: http://www.remics.eu/system/files/REMICS-Mohagheghi-camera-ready.pdf

[4] Gartner, "Gartner Identifies Five Ways to Migrate Applications to the Cloud" 16 May 2011. [Online]. Available: http://www.gartner.com/it/page.jsp?id=1684114. [Accessed January 2013].

[5] J. Varia, "Migrating your Existing Existing Existing Applications to the AWS Cloud - A Phase-driven Approach to Cloud Migration" 2010. [Online]. Available: http://media.amazonwebservices.com/CloudMigration-main.pdf.

[6] CloudHarmony, "Benchmarking in the Cloud" 29 May 2010. [Online]. Available: http://blog.cloudharmony.com/2010/05/what-is-ecu-cpu-benchmarking-in-cloud.html. [Accessed 29 January 2013].

[7] M. B. Uddin, B. He and R. Sion, "Cloud Performance Benchmark Series" Stony Brook University, 2010. [Online]. Available: http://digitalpiglet.org/research/sion2010cloud-http.pdf. [Accessed 29 January 2013].

[8] Open Source Community, "Iometer" [Online]. Available: http://www.iometer.org/. [Accessed 29 January 2013].

[9] axboe, "fio" [Online]. Available: http://freecode.com/projects/fio. [Accessed 29 January 2013].

[10] ProfitBricks, "ProfitBricks" [Online]. Available: https://www.profitbricks.com. [Accessed January 2013].

[11] RedHat, "Infinispan" [Online]. Available: http://www.jboss.org/infinispan. [Accessed 29 January 2013].

[12] I. Sysoev, "nginx HTTP server" [Online]. Available: http://nginx.org/. [Accessed 29 January 2013].

[13] M. Hosting, "M5 Cloud Hosting" [Online]. Available: http://www.m5cloud.com/. [Accessed January 2013].

[14] G. T. C. P. Ltd., "Global Technologies Company" [Online]. Available: http://www.globaltechnologies.com.au. [Accessed January 2013].

[15] Amazon, "Amazon Elastic Beanstalk" [Online]. Available: http://aws.amazon.com/de/elasticbeanstalk/. [Accessed January 2013].

[16] Google, "Google App Engine" [Online]. Available: https://developers.google.com/appengine/. [Accessed 29 January 2013].

[17] Heroku, "heroku" [Online]. Available: http://java.heroku.com/. [Accessed 29 January 2013].

[18] RedHat, "OpenShift Enterprise" [Online]. Available: https://openshift.redhat.com/app/. [Accessed 29 January 2013].

[19] M. Sprava, "Benefit from EJB in the Cloud" [Online]. Available: http://blog.jelastic.com/2012/08/09/benefit-from-ejb-in-the-cloud/. [Accessed 29 January 2013].

[20] J. Browne, "Brewer's CAP Theorem" 2009. [Online]. Available: http://www.julianbrowne.com/article/viewer/brewers-cap-theorem. [Accessed 29 January 2013].

[21] Apache, "Apache HBase" [Online]. Available: http://hbase.apache.org/. [Accessed 29 January 2013].

[22] Apache, "Apache Hadoop" [Online]. Available: http://hadoop.apache.org/. [Accessed 29 January 2013].

[23] Apache, "Apache Hadopp/HDFS" [Online]. Available: http://hadoop.apache.org/docs/hdfs/current/hdfs_design.html. [Accessed 29 January 2013].

[24] JBoss, "Hibernate" [Online]. Available: http://www.hibernate.org/. [Accessed 29 January 2013].

[25] OMG, "Architecture Driven Modernization" [Online]. Available: http://adm.omg.org/. [Accessed January 2013].

[26] CSS Corporation, "Migrating Existing Applications to the Cloud" 2011. [Online]. Available: http://www.csscorp.com/cloud/solutions/migrating-existing-applications-to-the-cloud.php.