Increasing Flexibility of Hybrid Clouds by Application Portability

Manohar Jonnalagedda, Michael C. Jaeger, Uwe Hohenstein and Gerald Kaefer

Corporate Research and Technology
Siemens AG
80200 München
Germany
manohar.jonnalagedda.ext@siemens.com
michael.c.jaeger@siemens.com
uwe.hohenstein@siemens.com
gerald.kaefer@siemens.com

Abstract: Many demands in engineering software, including legal, privacy, cost and technical issues influence the deployment of the software. With cloud computing, applications can be deployed in a provider cloud as an alternative to on-premises infrastructure. Thus, software should be ready for local or provider cloud deployment or a hybrid (mixed) setup. For the user of the application, the location of application parts should remain transparent. The goal is to provide flexibility to this regard while maintaining the basic advantages of cloud computing. This paper introduces and discusses some of the key issues when designing an application for a hybrid setup, in terms of software architecture, communication and security between modules. We give recent trends and recommendations on how to solve these issues so as to achieve application portability.

1 Introduction

The cloud is used as a metaphor for the Internet, most probably due to the depiction of the Internet in cloud forms. The cloud represents a set of services that provide computing, networking, and software capabilities without a clear location. Cloud computing refers to the provision of these services in three categories: (1) Software-as-a-Service (SaaS): Giving a user access to software which runs on the cloud. All computation taking place on the cloud, the client's hardware does not need to be high-performance. (2) Platform-as-a-Service (PaaS): Giving developers access to platforms and frameworks that allow them to leverage the computing power of the cloud and develop software that runs efficiently on the cloud. And, (3) Infrastructure-as-a-Service (IaaS): Giving access to highly scalable and elastic resources. This allows the developer to build applications that can scale (seamlessly) to many thousands of users, without having to buy the hardware for it.

While the first category is end-user oriented, it is really the latter two points that have made cloud computing attractive to businesses [gar10]. Cloud computing gives the offers highly available, elastic infrastructure for economical prices, along with reduced administration efforts. Companies are therefore looking to gain the benefits by developing software for the cloud. Such software must however adhere to several design constraints for cloud compatibility.[1]

From a business point of view, an idealistic goal for new applications is to decide at deployment time if the software shall run in the provider cloud, on-premises or in a hybrid (mixed) setup, so as to have an optimal trade-off between privacy and economic use of cloud resources. In this paper, Application Portability refers to the degree of ease with which a software can be deployed and run on both environments. In particular, this means the ease in which modules of an application can be split and deployed on on-premises or cloud platforms, ending up in a mixed setup. Portability has many different meanings, like being able to move an application from one cloud provider to another, with minimum application change. However, these other definitions are not in the scope of the paper. We introduce the engineering issues for application portability (flexibility at deployment regarding the location and the partitioning of the application) and describe various trends in the field. Section 2 explains the business drivers and motivation for this kind of portability. Section 3 shows the issues with this portability while Section 4 outlines possible solutions. Section 5 concludes the paper.

2 Motivation

The global pressure on cost reduction drives the industry to the adoption of rationalisation and automation. Cloud computing has gained a lot of popularity because of its promise to lower operational costs, and to provide large-scale, highly available systems. However, some applications cannot be moved to the cloud. Reasons for this may be business, privacy, legal, or operational-related. Moving business-critical information onto the public cloud is a matter of reticence to many big organisations as, if on the cloud, this information can physically lie in any continent, which may go against local laws and policies, or even make the information liable to laws in another country (for example the Patriot Act [pat01]). Consider an image management application which allows for uploading images taken with different devices to be uploaded onto a portal. The images can be accessed by a closed community, or shared publicly. Imagine a vendor providing such software to different organizations:

1. *The local police department:* with high-definition cameras to take pictures of various crime scenes and other surveillance pictures, it is interested in keeping high-quality pictures in a private setup. It goes without saying that the police would like the portal to run on-premises.
2. *A medical diagnostics application:* a medical institute has some high-tech equipment which takes pictures related to different illnesses inside the human

[1] 1Applications requiring hardware access, for example, cannot be easily deployed onto the cloud, as the cloud gives access to virtualized systems.

body. These images are large in size, and plenty in number. The institute might therefore like to store the images themselves on the cloud, although metadata concerning each image, being confidential, would be kept on-premises.
3. *A group of college friends:* these people would like their images of a trip to the Bahamas to be uploaded directly onto a public platform, so that their other friends can also view the pictures. They do not mind where the photos themselves are hosted.

An ideal engineering effort of this new software would let the business decide whether the application will run locally, on the cloud or in a hybrid setup. The business driven goal is the engineering of applications whose components offer relocation transparency in the sense of the RM-ODP [ISO96]. Dividing the application in such a manner brings forward some key engineering issues, which are discussed in the next section.

3 Issues

Engineering issues mainly arise when one wants to develop and application that should be able to run in hybrid setup: ideally, we would like to develop the image sharing software from the example only once, and deploy different modules of the application to different platforms, as dictated by the business requirements.

This raises the following issues:

- How do we design the application so that it is easy to break it into smaller, independent modules?
- How do we develop the different modules so they run on both platforms?
- How do we store the data relevant to the application?
- How do different modules of the application communicate with each other? If they communicate over the Internet, moreover, how do we ensure security?

4 Trends and Recommendation

The previous section has presented many issues that arise for engineering applications in a mixed cloud setup. The general rationale is that the finished application should be portable enough so that the entire application, or parts of it, can be deployed on a provider cloud. Thus, the proposed business goal of Section 2 is to decide at deployment time for a given setup during run time.[2]

[2] The freedom to migrate services into or off the provider during run time might be also required but has different motivations. For example, an organisation could migrate services to the provider cloud in the case that on-premises infrastructure does not have any further capacity to host applications (known as cloud-bursting). However, this degree of dynamic migration is not the scope of this work. Therefore, the following trends and recommendations do not cover dynamic migration but do provide flexibility at the deployment.

4.1 Software Architecture

For Platform as a Service (PaaS) offerings we can see several paradigms characteristic for cloud use. One of the standard paradigms is divide and conquer, which has also been adopted in Google's Mapreduce framework for efficiently processing large problem sets [DG08]. Cloud computing offers horizontal scalability in general: adding more working units to increase capacity, the ideal goal being to achieve almost infinite scalability. The common consensus seems to be that good cloud applications should be designed with knowledge of horizontal scalability. If the problem can be partitioned, then solving the problem can be infinitely spread among a set of working units.

Secondly, with the proliferation of development platforms for the cloud, each of them privileges certain building blocks for writing software optimally for the cloud. The Windows Azure platform, for example, provides the concept of roles: self-contained entities implementing an elementary unit of processing capability. In general, it is a good practice to implement elementary units as opposed to classic large blocks of functionality: the latter would lead to large units in the software system, thereby not allowing for partitioning of the problem and take advantage of the horizontal scaling of the cloud. The partition of the application in many elementary units is a basic condition for the split deployment of the application. In conjunction with elementary application units, the second larger design issue for cloud computing applications is decoupling. For horizontal scalability, statelessness is preferred, as state at a certain key point of the application can block it, hence losing all the advantages of elasticity and availability. Furthermore, strong coupling would prevent the application from the ability to scale dynamically. In order to elastically add working entities during run time, the application architecture must show decoupled units of stages where problem parts can be processed in an independent manner. For splitting the application into two deployment locations, the decoupling allows for the use of service buses or other communication mechanisms in order to overcome the distance between the two parts (described in section 4.3).

To summarize, horizontal scaling of elementary working units in the staging of an application is an elementary architectural consideration in order to leverage the cloud resources in an optimal way. This paradigm also enables the slip deployment of the application in a hybrid cloud setup. Horizontal scaling in conjunction with a staged application flow is also known as pipeline, which has been adopted in both hardware and software architectures.

4.2 Data Models

Cloud computing has brought to the eyes of everybody a shift in thinking about data that has been taking place in the IT world for a long time. Traditionally, industrial applications have considered relational database management systems (RDBMS) the de facto standard for persisting data. SQL database systems (DBS) are therefore highly prevalent in the industry: they have the advantage of offering transactional services, a general schema for storing structured data, and indexing mechanisms which optimize queries. They concentrate mainly on consistency. In recent years, many companies have been challenged by the sheer quantity of information stored and accessed over the

Internet: the necessity to make their services highly available has been the main driving force. This has seen the development of many No-SQL type of DBS, lightweight systems whose main selling point is distributed services and high availability.

Cloud computing, with its promise of elasticity and availability, is a natural fit for such No-SQL DBS. This explains why the first storage solutions have been BLOB (binary large objects) storages emulating a file system to some extent, and table storages. The latter keep a table view without requesting a fixed pre-defined table schema to be defined in advance. Hence, heterogeneous structures can be stored in one table. The main focus of table storages is to offer those 80% of database functionality that typical Web applications really require. They sacrifice consistency for the sake of availability and partition tolerance in the sense of the Brewer's CAP theorem [GL02]. Economic drivers (the need for providers to cater to the inertia of the industry) has however also seen the appearance of SQL-based persistence solutions appearing on the cloud.

In order to make cloud applications independent of the underlying persistence, introducing an access layer is useful in the way object/relational persistence frameworks such as HIBERNATE or ECLIPSELINK do for relational DBSs. However, the access layer should now be abstract from data models and access interfaces of storage types. Cloud applications can then easily switch from a DBS to No-SQL storage and vice versa. First proposals such as SIMPLEJPA or SIMPLEORM are available.

On-premises applications can use any form of cloud storage as well since there are either REST APIs [Fie00] to access cloud storages or usual protocols offered by DBSs. This makes them available from anywhere. There are also some reasonable scenarios where data is partially stored redundantly on-premises and in the cloud. In that case, some kind of synchronization is required. Here, synchronization frameworks such as Microsoft Azure Sync Framework are useful and provide much flexibility by means of definable synchronization rules.

Another important scenario is the following: an application runs completely on-premises, but wants to keep data in cloud storage in order to save storage costs, and must also keep data on-premises because of legal or privacy issues. Even more, the data should flexibly be distributed from one to the other. Then, the problem of distributed transactions arrives at the scene if modifications in both parts have to be done in one global transaction. Distributed transactions are still a major problem in Service-Oriented Architectures. No soon solution is thus expected for cloud storages.

As a consequence, the overall conclusions for persistence issues are that modules and the persistent data they use should stay together. Moreover, when designing such applications, a good practice is to consider Brewer's CAP theorem when trading full data consistency with scalability.

4.3 Communication and Security

The other important aspect of portability is ensuring security during communication between different sites where the application is hosted. In this paper, the topic of security

of the data residing on the cloud itself is not discussed. We identify two different layers of communication: at the application layer, and then at the network layer. At both layers, there are challenges when it comes to linking the cloud and on-premises platforms.

4.3.1 API communication / Application layer

API communication is one of the simplest and most elegant forms of communication, and very handy when developing highly available services, as it makes the software easier to use for clients/customers. Popular Web applications worldwide provide REST APIs for clients/customers/users to access data, process and use it in new, imaginative ways. The Programmable Web [pro] provides a list of applications offering such APIs.

This is relevant to application portability as we can imagine different modules of an application each presenting an API for access. Moreover:

- It is important to be aware that every cloud service requires Internet security hardening, because all interfaces are exposed to the Internet without firewall protection. Therefore, Web services security standards and identity federation are required building blocks on any design.
- It is good practice, when developing Web-based applications (whether it is hosted on the cloud or not) to provide an API that gives access to data: such an API generally requires little overhead on creating the application itself, and provides an easy-to-use, generalized interface to many potential clients without having to set up specific protocols with each of them.
- Two sites communicating uniquely via REST calls abstract the network layer beneath them, and alleviate the need for developing communication protocols. This effectively makes any module of the application portable, in the sense that as long as it provides an API interface, it can be accessed, wherever it resides.

Secure API design is an active subject of research and discussion: many services accept third-party login, for example they give access if the user provides Google login credentials. The issue here is that the user is giving his/her login and password to a third-party API, which is very compromising in terms of security and confidentiality. A better idea is to hand out tokens to the third-party website in question. This token, received from a trusted source, is accepted by the website, and access to data is thereby granted. This technique is named OAuth (which is an RFC specification) [oau]. Microsoft AppFabric's ACS [app] is an implementation of this standard and allows developers to write secure APIs that accept ACS tokens.

4.3.2 Bi-directional communication / Network layer

API communication is very pleasing in terms of simplicity and elegance, and we notice that two modules can call each other's API services in order to absorb data or insert data. For many applications however (chat applications and collaboration tools), such interfaces have too much overhead in communication. Such tools require that all sides are able to send and receive messages, and maintain a good performance level. For

achieving bi-directional communication, platforms need to be connected in a more stronger sense.

Large corporations having centers worldwide study ways of connecting these centers in secure ways. Enterprise Service Bus is a prevalent concept among them. Microsoft AppFabric's Service Bus aims to take the concept onto the cloud, enabling applications residing on the cloud to work with those on other platforms. The service bus concept is one way of looking at the communication issue between different platforms. Another way is to attempt to bring both the cloud and on-premises platforms into a single virtual network. Amazon's VPC [ama] is a product which goes in this direction: the on-premises network is connected to the Amazon Cloud resources via a secure VPN connection over the internet. This gives the illusion of a giant (seemingly infinite) enterprise network, with all the advantages of cloud computing (scalability, availability, computation). VPN connections as links between cloud and on-premises software have also been studied elsewhere. Wolinsky et al [WLJ+09] study network architectures in which network clients can be seamlessly added and removed from a virtual network. They conclude in their study that virtual networks provide excellent isolation, and good performance over Wide Area Networks.

4 Conclusions

In this paper, we have presented some of the problems and discussed some of the trends and recommendations when it comes to hybrid portable applications for the cloud. It is important to note that we focused on portability (flexibility in platform choice) at deployment time, as opposed to dynamic portability. We identified software architecture, data models, communication and security as key issues when it comes to developing such applications.

The points previously discussed show different characteristics, and it is apparent that not every solution fits every problem. We note particularly that portability may not be completely flexible when it comes to data models (cloud applications accessing on-premise data storage in particular). Some mechanisms that can be used for achieving migration transparency are contradictory in some application cases. This work has shown the general issues that arise when application need to provide the flexibility in deployment. Cloud computing, after the initial hype, has reached a certain maturity with many big industrial players actively looking for cloud solutions. Application Portability (or hybrid cloud) is a very important issue for them, and we expect to see many new solutions and ideas in this field in the coming months.

Literaturverzeichnis

[ama] Amazon Virtual Private Cloud (Amazon VPC). Available from *http://aws.amazon.com/vpc/*.

[app] Windows Azure AppFabric. Available from *http://www.microsoft.com/windowsazure/developers/appfabric/*.

[DG08] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[Fie00] Roy Thomas Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures.* Doctoral dissertation, University of California, Irvine, 2000.

[gar10] Gartner Top 10 Strategic Technologies 2010, October 2010. Available from *http://www.gartner. com/it/page.jsp?id=1210613*.

[GL02] Seth Gilbert and Nancy Lynch. Brewer's Conjecture and the Feasibility of Consistent Available Partition-Tolerant Web Services. In *In ACM SIGACT News*, 2002.

[ISO96] ISO/IEC. ITU.TS Recommendation X.902 — ISO/IEC 10746-1: Open Distributed Processing Reference Model - Part 1: Overview, August 1996.

[oau] OAuth Core 1.0. Available from *http://oauth.net/*.

[pat01] Uniting and Strengthening America by Providing Appropriate Tools Required to Intercept and Obstruct Terrorism (USA PATRIOT ACT), 2001. Available from *http://thomas.loc.gov/cgi-bin/ bdquery/z?d107:h.r.03162:*.

[pro] Programmable Web : Keeping you up to date with APIs, mashups and the Web as platform. Available from *http://www.programmableweb.com/*.

[WLJ+09] David Isaac Wolinsky, Yonggang Liu, Pierre St. Juste, Girish Venkatasubramanian, and Renato J. O. Figueiredo. On the design of scalable, self-configuring virtual networks. In *SC*. ACM, 2009.