IMPLEMENTATION AND EMPIRICAL ASSESSMENT OF A WEB APPLICATION CLOUD DEPLOYMENT TOOL

Américo Sampaio¹, Roberto Costa Jr¹,², Nabor C. Mendonça¹, Ricardo Hollanda Filho²

¹Universidade de Fortaleza (UNIFOR), ²E-NOVAR Soluções Tecnológicas

{americo.sampaio,nabor}@unifor.br, {roberto,ricardo.luiz}@enovar.com.br

Abstract

Cloud computing is becoming mainstream in software development, with many organizations now considering migrating their applications to the cloud. However, the task of deploying applications in infrastructure-as-a-service (IaaS) clouds is generally daunting due to a number of manual steps necessary to configure all application components as a set of virtual machine images. This work presents a tool, called TREXCLOUD, which greatly simplifies the task of deploying fully functional Web applications in IaaS clouds. With TREXCLOUD, users can easily select a set of files that contain the application components (e.g., .war files, database backup files, context files), which are then used by the tool to automatically deploy and configure the whole application stack in a supported IaaS cloud provider. An empirical study, in which nine users were asked to deploy two different Java web applications in the Amazon cloud using three different deployment tools, shows how TREXCLOUD enables a significant reduction in deployment effort (over 90% in the best case) when compared to existing state-of-the-practice approaches.

Keywords: cloud computing, infrastructure-as-a-service, deployment tool, empirical assessment

1. INTRODUCTION

Cloud computing is a recent computing paradigm that is transforming the way we build, deploy and manage software. The benefits of being able to easily provision and re-size the infrastructure used by an application, leveraged by a pay-per-use resource consumption model, are motivating many organizations to adopt infrastructure-as-a-service (IaaS) clouds (Armbrust et al., 2010) (Zhang, Cheng & Boutaba, 2010).

One popular use for IaaS clouds is to deploy modern web enterprise software such as Java Enterprise Edition (EE) applications (Hajjat et al., 2010) (Tavis & Fitzsimons, 2012). To be able to deploy this type of application in the cloud, one has to pack all application components (e.g., load balancer, web server, application server, database server, and business data) as a collection of virtual machines (VMs) images. In order to get the complete application “up and running”, a set of manual steps is necessary, such as creation of VM images, installation of the required software components in each VM as well as configuration of the components’ inter-dependencies (e.g., to configure the application server with the IP address of the database server). Besides, each IaaS cloud provider usually offers its own (often proprietary) deployment solutions, which tend to make application deployment in different clouds a hard and labor-intensive task. For example, Amazon offers a proprietary image format for packaging application components called Amazon Machine Image (AMI) (AMI, 2013). In case the organization decides to move its applications from Amazon to another IaaS provider (say, Rackspace), little reuse can be achieved and the deployment process has to be mostly re-done from scratch.

To tackle some of the difficulties discussed above, over the last few years some tools have been proposed specifically aimed at facilitating the IaaS cloud deployment task. Most of those tools, e.g., Chef (2013), Heroku (2013) and Rightscale (2013), have a broad scope and allow the deployment of any kind of application in the cloud. In general, those tools are difficult to use as they are based on “low-level” scripting languages and therefore are more recommended for experienced system administrators. On the other hand, there are tools with a narrower scope, such as Amazon Elastic Beanstalk (Beanstalk, 2013), which focuses on providing support for the deployment of web-based applications in the Amazon EC2 cloud. However, that tool still suffers from two fundamental limitations: (i) it offers no support for automatic database configuration, which has to be done manually by the tool user; and (ii) it is a provider-specific solution and thus cannot be used to deploy applications in different IaaS clouds.

This paper presents the TREXCLOUD tool, which allows the deployment and configuration of complete Web applications (including all components, from presentation logic to database) in different IaaS clouds in a simplified manner, without the need to execute manual and complex procedures. The tool currently supports two IaaS cloud providers (Amazon and Rackspace) and offers extensibility mechanisms to programmatically introduce new ones. An empirical evaluation of the tool, conducted in an industrial setting, shows promising results, indicating that the tool enables users with varied IT skills and backgrounds to quickly (in a matter of a few minutes) deploy and run different Web applications in the Amazon cloud. This represents a significant reduction in deployment effort (over 90% in the best case) when compared to Amazon’s own deployment tools (i.e., Amazon EC2 Dashboard and Amazon Elastic Beanstalk).

The remainder of the paper is structured as follows. Section 2 provides the required background on IaaS cloud deployment and explains the motivation for our work.

http://hipore.com/ijcc

doi: 10.29268/stcc.2013.1.1.4
Section 3 analyzes related deployment approaches. Section 4 describes the TREXCLOUD tool in more details. Section 5 reports on the method and results of the empirical study conducted to evaluate the tool. Section 6 provides a discussion and, finally, Section 7 offers some conclusions and directions for future work.

2. PROBLEM AND MOTIVATION

Cloud computing services are generally categorized in three types (Armbrust et al., 2010) (Zhang, Cheng & Boutaba, 2010). Infrastructure-as-a-Service (IaaS) clouds, such as Amazon EC2 and Rackspace, offer computational (e.g., virtual machines), network and storage resources to customers. Other types are Software-as-a-Service (SaaS) and Platform-as-a-Service (PaaS) clouds, which offer applications (e.g., Salesforce.com) or development and service platforms (e.g., Google App Engine), respectively. This work focuses on offering a tool to support the deployment of web applications in IaaS clouds.

Typically, the process of deploying a web application in an IaaS cloud can be organized in two main steps: (i) creating images; and (ii) installing and configuring software components. The following subsections describe each of those steps in more details, along with a discussion of the need for a new tool to support them.

2.1 CREATING IMAGES.

This step corresponds to creating the virtual machine images that will be used to host the application components in the cloud. For example, in the case of a typical Java EE application, a minimal configuration would require two images: one for the application server and one for the database.

The application server image would host the Java Container (e.g., Spring, JBoss, Tomcat, etc.) necessary for the software being delivered. When considering a specific IaaS cloud one would have to either build a new VM image from scratch (a cumbersome manual process), or reuse one of the several pre-built images usually offered by the IaaS provider. For example, Amazon offers many public AMIs for several known Java Containers (in several operating system distributions) that users can customize according to their technical needs and preferences (AMI, 2013). Even though reusing an existing image is easier than building one from scratch, in both cases the user still needs to follow a set of specific configuration steps, such as registering the image and configuring it with the provider's security certificates.

The database image in turn would host the specific database and would be applied to the database. Once the images are created and registered for use with the IaaS provider, the next step would be to install in each image the application logic that will be deployed in the cloud. Considering the previous Java EE example application, the user would need to perform the following steps.

In the application server image, the user would need to copy and install the presentation and business logic that is executed by the application server. In general, this is done by starting up the virtual machine, logging into it and copying the files (e.g., .WAR files) that contain the application logic. If necessary, some application server configuration files also will have to be created or modified.

In the database image, the configuration would be similar. The user would need to either create a new database from scratch, or upload an existing one. Moreover, this is also the step where some tuning and configuration settings would be applied to the database.

Besides configuring each image, the user also has to configure the dependencies between each component. In the example above, a common step would be to configure the application server with the IP address of the database server virtual machine. It is worth mentioning that this step usually can only be performed after the database server has been deployed and initialized, as most IaaS virtual machines have dynamic IP addresses. Therefore, in every new deployment the database server would have a different IP address, which would have to be properly configured (either manually or automatically) as part of the application server configuration.

2.3 MOTIVATION.

It is clear from the above description that the process of deploying applications in IaaS clouds includes a series of non-trivial steps that can be very time-consuming if done manually. Nevertheless, its importance cannot be ignored, as application deployment is one of the most basic tasks when it comes to using an IaaS cloud (Hajjat et al., 2010) (Rodero-Merino et al., 2010) (Binz et al., 2012). For this reason, there are a number of tools currently available for deploying applications in IaaS clouds, which vary in functionality and deployment style. Some tools (e.g., Appzero (2013), Chef (2013), CloudFoundry (2013), Heroku (2013) and Rightscale (2013)) are not only deployment tools as they also combine other functionality such as being able to monitor and automatically scale the application, which is outside the scope of this work. Regarding deployment styles, while there are tools focused at using more powerful and complex mechanisms to be able to deploy several kinds of applications (e.g., scripting languages), there are other tools which have a narrower scope and use simpler mechanisms to deploy specific applications (e.g., Amazon Elastic Beanstalk (Beanstalk, 2013)).

To explain the reasons why we have decided to build another deployment tool (i.e., TREXCLOUD), we will first
providing an overview of the industrial setting where the tool was originally implemented and developed. E-NOVAR is a small software company located in Fortaleza, Brazil, whose main business focuses on building customized solutions for different industries. Each software product developed by E-NOVAR aims at offering a solution that is targeted for a specific problem that the customer faces. Since its inception, the company has implemented more than 50 products for different industries (e.g., financial, healthcare, media, gaming, retail etc.) and using several different technologies (e.g., Java, PHP, .NET, Objective C, Android, etc.).

Some of E-NOVAR customers are companies that operate with a very lean IT team and are not interested in investing on a dedicated datacenter infrastructure. For these companies, E-NOVAR was contracted to manage the computing infrastructure required for their applications. In order to host its customers’ applications, E-NOVAR started using some local hosting companies in Fortaleza, which offer a reliable yet expensive hosting service. In an attempt to reduce its hosting costs, more recently the company has started to consider using cloud services.

Since E-NOVAR works with many different technologies and solutions, the best option was to use IaaS clouds due to their flexibility when compared to other cloud models. Initially, the company migrated a few applications to the Amazon EC2 cloud and experienced several benefits. Besides a significant reduction in operational costs, E-NOVAR also perceived other clear benefits, such as flexibility in pricing options, easiness to scale the resources up and down, as well as the good performance obtained from the cloud environment.

However, the migration process itself was cumbersome and mostly manual. As mentioned previously, many virtual machine images had to be created and configured with the required software components. Moreover, whole databases had to be moved from the previous hosting service to the Amazon cloud. This situation soon led to the conclusion that a customized software deployment tool was required to facilitate and expedite the cloud migration task. After experimenting with several automated and semi-automated deployment technologies, the company decided to build its own tool, as the technologies available were not able to fulfill all the company’s cloud deployment requirements, described as follows.

REQ01: User-friendliness. The tool shall provide a user-friendly way to perform the deployment. By this we mean that the tool shall not require any advanced infrastructure knowledge in order to be used (such as that required to write “low-level” configuration scripts), which could prevent less technical users from successfully performing the deployments.

REQ02: Fully automated support. The tool shall provide fully automated support for all steps involved in deploying the application. This means that all application components shall be deployed and completely configured by the tool. As explained previously, this requirement involves the execution of a series of steps such creating, deploying and configuring all the application virtual machines and their software components inter-dependencies (e.g., configuring the application server with the IP address of the database server virtual machine).

REQ03: Database support. As most industrial applications require a relational database, it is important that the tool provides the necessary support to install and configure whole databases as part of the cloud deployment process. The tool shall enable the user to reuse a database backup file from which the database will be created and populated by the tool. The tool also shall configure the database virtual machine and make it ready to be used by other components (e.g., application servers).

REQ04: Multicloud support. The tool shall provide deployment support for multiple IaaS providers. The users shall be required only to provide the tool with their IaaS cloud credentials and access information in order to be able to perform a deployment with that provider. The users shall be able to choose from different IaaS providers when performing a deployment. This is an important requirement since it avoids the users from being locked in to a specific provider.

The next section discusses whether (and how) several existing deployment approaches fulfill the set of requirements described above.

3. RELATED APPROACHES

This section provides a detailed analysis of several cloud deployment tools with respect to the four requirements described in the previous section. As there are many deployment tools currently available, we focus on five of those tools, namely Appzero (2013), Amazon Elastic Beanstalk (2013), Rightscale (2013), Heroku (2013), and Claudia (2013). These five tools were selected based on their popularity and similarity to our work. Other related approaches are also briefly discussed in the end of the section.

3.1 Appzero.

Appzero (2013) is a tool that enables the creation and management of Virtual Application Appliances (VAAs). Application components (e.g., application logic, application server, and database server) are packed inside a VVA, and can be moved to either a virtual or a physical machine. After building a VAA, deploying it in an IaaS cloud is a simple process that is managed by the tool. Another characteristic of Appzero is that all application components in a VVA are completely isolated from the operating system. The main point is to encapsulate all the changes of state of the application inside the VAA. This allows moving a VAA from one cloud provider to another without losing the application state.

REQ01: User-friendliness. Appzero offers a flexible and customizable deployment solution. It enables to deploy
any kind of application, but requires a certain level of expertise to use the tool because users need to understand about the configuration details of each of the application components contained inside a VAA.

REQ02: Fully automated support. Appzero automates the whole deployment and configuration process. After the required VAAs are created, the application components can be easily deployed and configured using the tool in a fully automated fashion.

REQ03: Database support. Appzero supports the deployment and configuration of databases while creating their VAAs. To this end, the tool requires from its users the necessary technical knowledge to install and configure a database system inside the VAA. This means that for users with less technical knowledge this step could be very complicated to achieve.

REQ04: Multicloud support. Appzero offers support for multiple cloud providers, enabling users to easily move VAAs to public cloud providers as well as to private clouds.

3.2 Amazon Elastic Beanstalk.

Amazon Elastic Beanstalk (EB) (Beanstalk, 2013) is an easy to use deployment tool that was built to facilitate the deployment of web applications in the Amazon EC2 cloud. The tool provides support for several popular web platforms (e.g., Java, .NET, PHP, etc.) and facilitates both the deployment and scaling of applications in the cloud. EB requires as input file the application logic code (e.g., .WAR files of a Java web application) that is uploaded to the cloud and configured within the context of an EC2 virtual machine instance. Regarding database support, EB is easily integrated with Amazon's own database solution, called Amazon Relational Database Service (RDS). Integration with a different database system is not as straightforward, as can be seen in the tool's online documentation.

REQ01: User-friendliness. EB offers a very easy deployment approach, in which users just need to follow a set of deployment Wizards and provide the tool with the application logic code.

REQ02: Fully automated support. The tool enables automatic deployment and configuration of the application logic but lacks in automated support for database services other than RDS (see below).

REQ03: Database support. The tool supports RDS as well as other relational databases, such as MySQL. However, in the latter case the tool requires a set of manual procedures in order to configure the database.

REQ04: Multicloud support. EB only supports application deployment in the Amazon EC2 cloud.

3.3 Rightscale.

Rightscale (2013) is an advanced tool that supports deployment and auto-scaling of applications in multiple cloud providers (e.g., Amazon, Rackspace, Google, etc.). The tool requires the users to produce a set of configuration scripts, called Rightscripts, which are executed during the deployment and startup of the application instances in the cloud. Typically, those scripts encapsulate a series of configuration steps that have to be executed in order to configure the whole application in the cloud. The tool offers several Rightscript templates (developed by the Rightscale users community) that can be used to configure popular application development and execution environments (e.g., a LAMP stack, and .NET or Java EE applications). Even though those templates can be easily reused and modified by the users, they still require a reasonable level of expertise in order to be properly understood and customized with an appropriate set of configuration parameters (e.g., location of the application code repository, database configuration parameters, etc.).

REQ01: User-friendliness. The power and flexibility gained by writing and reusing configuration scripts is a clear advantage of Rightscale, allowing users to have a tight control over the whole deployment process. However, in order to build and reuse those scripts users need to have a good level of technical knowledge.

REQ02: Fully automated support. Once the necessary scripts are built, Rightscale is able to automatically deploy and configure the whole application stack.

REQ03: Database support. Users are able to deploy any type of database with Rightscale by building the required configuration scripts. Therefore, database support also depends on advanced technical knowledge from users.

REQ04: Multicloud support. Rightscale offers support for several public cloud providers (e.g., Amazon, Rackspace, etc.) as well as open source private cloud solutions (e.g., OpenStack).

3.4 Heroku.

Heroku (2013) is a PaaS provider that offers support for applications developed in several programming languages (e.g., Java, Scala, Node.js, Python, etc.). Heroku targets developers that want to run and scale their applications in its own cloud infrastructure, which in turn is build on top of Amazon EC2. For experienced developers this can be an interesting solution as it supports several popular development tools, such as application frameworks (e.g., Spring MVC and Hibernate) and source code versioning tools (e.g., Git). Using Heroku to deploy and manage applications requires advanced technical skills from its users, who must be able to write and understand shell scripts to deploy and configure their applications.

REQ01: User-friendliness. Heroku focuses on providing support for software developers interested in deploying and managing their applications in the Heroku cloud. Users with less expertise in source code versioning tools and scripting languages might have difficulty in using Heroku.

REQ02: Fully automated support. Heroku fully automates the deployment process for several kinds of applications and development environments.
REQ03: Database support. Heroku supports the PostgreSQL and MySQL relational databases. The tool requires the use of a series of scripts in order to migrate an existing database to the Heroku cloud.

REQ04: Multicloud support. Heroku operates on its own cloud, which is built on top of Amazon EC2. Users are not able to deploy applications in other cloud providers with Heroku.

3.5 CLAUDIA.
Claudia (2013) (Rodero-Merino et al., 2010) is a platform that supports application deployment and auto-scaling in IaaS clouds. Claudia users can specify a set of elasticity rules with triggers and actions that enable the platform to manage different types of applications. Claudia works with different IaaS providers and uses the Open Virtual Format (OVF) standard (OVF, 2013) to package the application components, configuration parameters and scalability rules.

REQ01: User-friendliness. Claudia users need to have a deep knowledge of OVF in order to provide the virtual application specification and the scalability rules.

REQ02: Fully automated support. Claudia fully automates the deployment process given an OVF specification provided as input.

REQ03: Database support. Users can deploy any type of application with Claudia as long as they provide the proper OVF specification and database images, which requires a good technical knowledge.

REQ04: Multicloud support. Claudia supports multiple cloud providers.

3.6 SUMMARY AND OTHER APPROACHES.
There are a number of other approaches that, like most of the tools investigated in the previous section, also support the deployment of any kind of application by resorting to more general deployment mechanisms. TOSCA (Binz, 2012), for example, works in a similar fashion as Claudia. The applications that will be deployed in the cloud are described in an XML-like template where the user describes the software components to be used, their relationships and configuration properties. Moreover, the user also specifies workflows (called Pans) that describe the management flow of the application. A similar scripting-based approach is offered by CloudFoundry (2013) and Chef (2013). The main shortcoming of these approaches is that they require a deeper knowledge of system administration and scripting languages, thus restricting their use to more advanced technical users.

The approach described in (Wettinger, 2013) defines a methodology that focuses on deploying the application and the middleware (e.g., application servers, database, Enterprise Service Bus) based on a set of script-based deployment plans. There is no tool support yet for the approach and its applicability in practice was demonstrated using Chef. Besides, the approach currently does not support dynamic wiring between components which is crucial for automating cloud applications as some properties are only known during deployment (e.g., IP address).

<table>
<thead>
<tr>
<th>Approach</th>
<th>REQ01</th>
<th>REQ02</th>
<th>REQ03</th>
<th>REQ04</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appzero</td>
<td>❌</td>
<td>⬤</td>
<td>❌</td>
<td>⬤</td>
</tr>
<tr>
<td>EB</td>
<td>⬤</td>
<td>❌</td>
<td>⬤</td>
<td>❌</td>
</tr>
<tr>
<td>Rightscale</td>
<td>❌</td>
<td>⬤</td>
<td>❌</td>
<td>⬤</td>
</tr>
<tr>
<td>Heroku</td>
<td>⬤</td>
<td>⬤</td>
<td>⬤</td>
<td>⬤</td>
</tr>
<tr>
<td>Claudia</td>
<td>❌</td>
<td>⬤</td>
<td>❌</td>
<td>⬤</td>
</tr>
</tbody>
</table>

Table 1. Analysis of five deployment approaches

The work by (Kang, 2011) focuses on redeploying cloud applications based on the user experienced performance (measured by latency). The idea is to dynamically redeploy the application in machines that provide optimal performance to the users. This work was further improved in (Kang, 2012) to consider not only distribution of users but also how services affect the performance of others. To attack this challenge, they employ cross service information as well as user locations to build a new model based on integer programming. The works (Fan, 2011) (Fan, 2012) also focus on selecting the best deployment strategy from the performance perspective. They propose a clustering based method for deploying communication intensive applications to the cloud environment. All these works focus on a different aspect (performance optimization) of deployment and are complementary to ours.

There are also some other interesting works that focus on different problems related to deployment. (Ruehl, 2013) and (Ruehl, 2012) describe an approach for deploying application components (services) in SaaS that allows the customer to select (based on a set of requirements) with whom they want to share infrastructure. The idea is to avoid sharing a multitenant infrastructure with possible competitors or untrusted partners. Another work (Chen, 2012) offers a framework for deploying virtual machines in a way that maximizes resource utilization and minimizes the costs for provisioning the VMs in a services provider infrastructure.

Neither the approaches previously described nor the five deployment tools shown in Table 1 fully satisfy all our requirements.

4. TREXCLOUD
TREXCDM aims at providing easy to use support for deploying web applications in IaaS clouds. The tool currently supports the deployment of Java EE applications in two well-known commercial cloud providers: Amazon and Rackspace. The goal of the tool is to free the user from having any specific knowledge about proprietary
technologies of IaaS clouds such as those required for creating and configuring virtual machine images. Moreover, the user does not need to have any knowledge about “low-level” scripting languages, so that s/he can deploy a complete application in the cloud with a few mouse clicks. Figure 1 shows the tool’s architecture, which contains three main components, briefly described below.

Figure 1. TREXCLOUD architecture

User Interface (UI). This component implements the tool’s web-based graphical user interface. It is responsible for reading in the user-provided input files and configuration parameters necessary for setting up and deploying the application in the cloud.

Communication. This component is responsible for invoking the specific deployment operations provided by each IaaS cloud provider supported by the tool. It currently offers support for the Amazon and Rackspace clouds, and can be easily extended to support other providers, as we will explain later on in the paper.

Pre-built Images. This component provides a set of pre-configured images (e.g., application server, database server, etc.) for each IaaS cloud supported. The tool is able to automatically launch and configure those images with the appropriate set of configuration parameters during the deployment process.

The following subsections provide more details on the design and implementation of each of these three components.

4.1 USER INTERFACE.

One pre-requisite for using the TREXCLOUD is that before performing any new deployment the user needs to have created an account with one of the supported IaaS providers. Once the user has created such an account, s/he has to register her/his cloud credentials with the tool, as shown in Figure 2.

After this step, the user is apt to perform a deployment with the registered clouds. To do this, the user must provide the tool with following information (see Figure 3):

- **Cloud Provider**: the name of one of the supported IaaS cloud providers.
- **Name**: the name given for the current deployment.
- **File**: the location of the .WAR file that contains the packed application logic that will be uploaded to the cloud.
- **Context**: the context file of the Java EE application.
- **Database**: the name of one of the supported database servers, along with the username, password, name and backup file created for the application database.

Figure 2. Registering cloud credentials

Figure 3. Deploying a web application

4.2 COMMUNICATION.

This is the core component of the tool. It offers a generic interface, named TRexCloudService (see Figure 4), through which the UI component invokes the services of different cloud providers. In this way, the UI component is completely decoupled from any particular IaaS provider API. The TRexCloudService interface defines a set of deployment operations that need to be implemented for each specific provider. In the case of the Amazon EC2 cloud, for instance, the implementation for that interface (AWSServiceImpl) was based on the EC2 Java development API. Similarly, in the case of the Rackspace cloud, the implementation (RakspaceServiceImpl) was based on that provider’s RESTful API. In order to extend the TREXCLOUD for another cloud provider, one has to
implement the operations defined by the TRexCloudService interface, along other requirements, as we explain in Section 4.4.

4.3 PRE-BUILT IMAGES.

In TREXCL OUD, it is up to the tool developer (instead of the tool user, as with other deployment approaches) to create all the images and configuration scripts required to automate the deployment and configuration process. The way this is supported in the current version of the tool is as follows. The TREXCL OUD developer creates virtual machine images containing the required components for each IaaS deployment, as described in Section 2.1. For example, the tool currently has pre-built images for the application server (Tomcat) and the database server (Postgres) supported by both Amazon and Rackspace. Each component image contains an embedded configuration script that runs automatically once the virtual machine is booted and is responsible for setting up the configuration properties passed by the user in the UI component.

4.4 EXTENDING TREXCL OUD.

In order to provide support for a specific cloud provider, the developer of TREXCL OUD has to extend the tool in basically two steps. The first step would be to provide a concrete implementation for the tool's communication interface implementing all its methods based on the APIs of the target cloud provider. The second step would be to create the specific images that will contain the software components (e.g., application server, database, etc.) that will be later configured when the user application is deployed. To illustrate these two steps, we now describe how the tool was extended to support deployment of Java EE applications in the Amazon EC2 Cloud.

The first step was to create the class AWSServiceImpl (Figure 4) that implements all the methods of the TRexCloudService interface using the Amazon AWS Java SDK. Figure 5 shows a sequence diagram illustrating the implementation of those methods. Basically, those methods implement the steps required to gather the input files provided by the TREXCL OUD user (e.g., .WAR files, database backup files and other configuration parameters), upload those files to a storage service in the cloud (e.g., Amazon S3), and then configure and start the virtual machine images containing the application software stack.

The second step involves building the images that will contain the application stack and a configuration script to configure and start those images in the cloud. This script is stored inside the main image of the application and executed during its initialization. Any configuration information required by the script is passed on to its respective virtual machine instance via cloud API calls issued by the TRexCloudService methods implementation, as shown in the sequence diagram of Figure 5. The configuration script is specific for a given application profile. For example, in order to configure Java web applications based on a relational database, the script would need to:

- Create a database user;
- Upload a database backup file to the cloud storage service;
- Restore a new database from the backup file;
- Start up the database;
- Copy the application context file to the application server image;
- Copy the application logic code to the appropriate path;
- Startup the application server (ex.: Tomcat) and configure it to access the database.

It is important to mention that the previous two steps are implemented by the developer of TREXCL OUD, and not by the users of the tool. The next section shows how the users interact with TREXCL OUD.

4.5 AN ILLUSTRATIVE EXAMPLE.

To illustrate how the whole deployment process works, we will use a simple application as an example. Suppose the developer of a Business Intelligence (BI) application, called MyDashBoard, wants to deploy it in the Amazon cloud. If s/he decides to use the TREXCL OUD tool (from now on we call this person the user) the user needs first to create an account in Amazon (is s/he does not have one already) and then to register her/his Amazon credentials with the tool (Figure 2).

The user then needs to provide a set of configuration properties and some input files, as shown in Figure 3. At this point, when the user hits the “send” button what happens “behind the scenes” is that all those properties are saved into a file that will be uploaded and stored in the Amazon simple storage service (S3). Moreover, the files containing the application logic (.WAR) and the database backup (.TAR) are also uploaded from the user’s computer into S3.

Once the user hits the “send” button the whole application is configured and started up in the cloud. The Amazon-specific implementation for the tool’s communication interface (Figure 5) will call the required APIs for starting up the VM instances of all application components in the correct order.
For example, it will start the database instance first and then, when that instance is booting, the script embedded in it will download the properties file (containing the database username, password and name) and database backup (.TAR) file and perform the complete configuration of the database. After the database is configured, the tool starts up the VM instance (based on the Tomcat pre-built image) of the application server and its embedded script downloads the application logic files (.WAR) and the correct configuration of the database IP address. What is important to highlight is that the user of the tool is completely unaware of all these configuration steps and only has to provide a bunch of configuration properties and files. This is one of the main contributions of our approach since it makes deployment of a complete application a matter of a few mouse clicks.

5. EMPIRICAL ASSESSEMENT
To evaluate the effectiveness of TREXCLOUD, we conducted an empirical study where we asked nine different users to use and assess three different approaches to deploy two different web applications in the Amazon cloud. Our goal was to investigate how TREXCLOUD compares, in terms of user effort, with other approaches commonly used for deploying web applications in the Amazon cloud. More specifically, we wanted to investigate the following research questions:

(RQ1) How does our approach compares, in terms of effort (measured as the time spent to complete a deployment), with other approaches commonly used by Amazon customers?

(RQ2) Which approach is more suitable for which category of user (considering users with varying levels of expertise in Java and cloud related technologies)?

5.1 SETUP AND PLANNING.
To carry out the evaluation, we selected nine users with the following profiles (three users from each profile):

Naïve. Users that have never performed a deployment of a Java EE application before and that have never used an IaaS cloud.
Intermediate. Users that have some previous experience with the deployment of a complete Java EE application (including application server and database), but no experience with IaaS clouds.

Advanced. Users that have the same skills as the users of the Intermediate profile plus some previous experience with IaaS cloud deployment.

Each user was asked to perform six deployments in the Amazon cloud, three deployments per application. In each deployment, the user selected one out of the three following approaches to deploy the application:

Amazon Elastic Compute Cloud (EC2): In this approach, the user performs the deployment using only the Amazon APIs and Dashboard tool, which provides a set of basic operations for deploying and managing VM instances in the Amazon EC2 cloud. Therefore, with this approach most of the deployment steps described in Section 2 are performed manually by the cloud user. This is the approach most commonly used by current IaaS cloud users.

Amazon Elastic Beanstalk (EB): In this approach, the user performs the deployment using Amazon Elastic Beanstalk, a web-based tool offered by Amazon to deploy web applications in its EC2 cloud (Beanstalk, 2013). One important limitation of this tool is that it does not support deploying the database only the application logic.

TREXCLoud (TREX): In this approach, the user performs the deployment using TREXCLoud, the tool described in this paper.

The two applications selected for the study were Calendar and MyDashBoard. Calendar is a simple web-based calendar application written in Java and obtained from Google. One interesting characteristic of this application is that it does not have a database, only application logic deployed as a .WAR file. MyDashBoard is a web-based Java Business Intelligence tool developed by E-NOVAR, the company behind TREXCLoud. MyDashBoard contains both application logic and a database. The application logic is packed in a .WAR and context file while the database is packed in a backup file (.TAR) for the Postgres database.

During the study, the nine users were instructed to perform the deployments in random order, to avoid biasing one approach in favor of others. Moreover, they were required to complete all six deployments. A deployment was considered complete when the user: (i) had the application up and running in the Amazon cloud, which was verified by successfully accessing its URL from a web browser; or (ii) explicitly assumed that s/he had failed to perform the deployment.

Before conducting the experiments, the users were given an overview of the three deployment approaches investigated. They also received a package containing all files required for the deployment of the two applications (e.g., .WAR, .TAR, context file) as well as the necessary credentials to access the Amazon cloud.

To provide a familiar computing environment for the experiments, the users were allowed to perform the deployments using their own computers. However, to make it easier to compare the users’ deployment efforts, we configured all computers used in the study to record a video of their screen during each deployment session. These videos were then subsequently analyzed to calculate precisely the time taken by each user to complete each deployment.

In addition to the quantitative analysis of the users’ deployment effort, we also conducted a qualitative analysis to investigate the users’ impression with each of the three approaches. For the qualitative analysis, we asked the users to fill in a simple questionnaire to provide us with some qualitative data regarding each deployment. The questionnaire contained the following items:

- Assign a grade to each deployment in a scale from 0 to 5, where 0 represents that you failed to perform the deployment and \{1, 2, 3, 4, 5\} represent, respectively, that the deployment was very difficult, difficult, average, easy, very easy to perform.
- Suggest some improvements on the functionality of each of the tools used.
- Comment about the advantages of using each tool in the context of each deployment.

5.2 Quantitative Results

<table>
<thead>
<tr>
<th>User</th>
<th>Profile</th>
<th>Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>EC2</td>
</tr>
<tr>
<td>User 1</td>
<td>Naive</td>
<td>0:22:26</td>
</tr>
<tr>
<td>User 2</td>
<td>Naive</td>
<td>0:03:51</td>
</tr>
<tr>
<td>User 3</td>
<td>Naive</td>
<td>0:24:34</td>
</tr>
<tr>
<td>User 4</td>
<td>Intermediate</td>
<td>0:35:10</td>
</tr>
<tr>
<td>User 5</td>
<td>Intermediate</td>
<td>0:51:05</td>
</tr>
<tr>
<td>User 6</td>
<td>Intermediate</td>
<td>0:56:12</td>
</tr>
<tr>
<td>User 7</td>
<td>Advanced</td>
<td>0:29:08</td>
</tr>
<tr>
<td>User 8</td>
<td>Advanced</td>
<td>0:27:55</td>
</tr>
<tr>
<td>User 9</td>
<td>Advanced</td>
<td>0:30:43</td>
</tr>
</tbody>
</table>

Table 2. Deployment Effort for Calendar

Table 2 shows the quantitative results obtained representing the time taken by each user to deploy the Calendar application. As we can see, for all users (except for User 2 by a few seconds) TREX outperformed the other two approaches by a large margin. We can also see that none of the naive users were able to deploy the application with EC2. In contrast, all intermediate and advanced users were able to successfully complete the deployment with that approach; however, those deployments all took a considerable amount of time - approximately 28 minutes in the fastest case (for User 8). The greater effort required by EC2 can be explained by the fact that this is a mostly manual approach.
Another interesting result is the deployment times obtained with EB. Since the Calendar application does not include a database, technically deploying it with this approach should be of a similar level of difficulty of doing it with TREX. This seemed to be true for Users 2, 5 and 9, who were able to complete the task with EB in less than 6 minutes. However, we can see that most users took much longer to deploy the application with EB than with TREX.

In fact, deploying the application with TREX was always very fast, taking from less than two minutes (for Users 4, 6 and 8) up to a maximum of four minutes and twenty-nine seconds (for User 3). Most of this time is spent in the upload of the configuration files (mentioned in Section 4) and virtual machine start up at Amazon rather than with user interaction with TREX.

Table 3 shows the results representing the time taken by each user to deploy the MyDashboard application. Again, only the intermediate and advanced users were able to complete the task with EC2, although it took them a very long time to do it (approximately two hours on average). In the case of EB, this time only one user (User 5) was able to successfully perform the deployment, taking her/him approximately one hour to complete the task. The reason for the high failure rate observed with this approach was the difficulty to deploy the application database. When analyzing the videos recorded during the deployment sessions performed with EB, we noticed that the users somehow got confused about the proper way to configure the application since that tool does not explicitly support database deployment.

Table 4 shows a consolidate view of the effort reduction achieved by TREX when compared to EC2 and EB, with respect to the three user profiles and the two web applications. For Calendar, considering only successful deployments, TREX achieved a reduction of 94% and 83%, on average, in the application deployment time when compared with EC2 and EB, respectively. For MyDashboard, in turn, the results show an average reduction of 94% and 91% in the deployment time when compared to EC2 and EB, respectively (again, considering only successful deployments).

5.3 Qualitative Results

Figure 6 shows the results gathered in the questionnaires with the grades assigned by all users for the Calendar application. Users 1, 2 and 3 (naïve users) could not complete the task with EC2 (grade = 0) and gave either grade average (Users 1 and 3) or very easy (User 2) to EB. All naïve users found TREX very easy (grade = 5) to use. For intermediate and advanced users (except for User 6) they found that EC2 is the most difficult to use followed by EB and then TREX, which almost all of them considered very easy. This feedback helps to back up our claim that independently of user profile TREX provides an easy solution for Java Enterprise deployment.

Table 4. Effort Reduction (Relative to EC2 and EB)

![Table 4](http://hipore.com/ijcc)

Figure 7 shows the grades for MyDashboard. Again, we can see again that all naïve users were unable to complete the task with EC2, which was best evaluated as being of average difficulty (Users 6 and 9). The interesting results here are the grades assigned to EB. Only User 5 was able to complete the deployment with EB and recognized it as
being very difficult (grade = 1). As explained before, this was due to the inclusion of the database as part of the deployment, with most users being unable to complete the process with the EB tool. Regarding TREX, all users found it either easy or very easy to deploy the MyDashBoard application, and had no problems in configuring the database.

(RQ2) Which approach is more suitable for which category of user (considering users with varying levels of expertise in Java and cloud related technologies)?

For all three user profiles investigated, deploying the two applications with TREX was much easier and faster than with other common approaches. In the case of naïve users, the tool proved essential as they failed to perform the deployments using the other approaches. This was especially true when a database deployment was required.

5.5 LIMITATIONS

The study reported in this paper shares some of the limitations commonly found in any experimental studies involving software (Easterbrook, 2007), such as to depend on people with varied level of skills to perform the required tasks (i.e., cloud deployments). For our study purposes, it was better to have users with a more diverse background since we wanted to investigate how the different categories of users influenced the results.

As mentioned before, we tried to make sure users performed deployments in different orders (varying the tool and application) so that this could avoid putting one tool in favor of another as the same user normally learns between deployments. Moreover, during the studies each user performed the tasks on their own and did not communicate among them.

The first author of this paper was also available during the experiments to respond to any doubts about the task to be done but not about how to do it or how to use a specific tool. In addition, to avoid any bias in the results, all the videos for each deployment session were analyzed to make sure that only the time devoted to the deployment being performed (e.g., the user could potentially start browsing the web for another purpose) was actually computed as part of the overall deployment time associated with a given deployment task.

6. DISCUSSION

After presenting the TREX tool implementation and its evaluation, we now revisit our requirements defined in Section 2 and discuss if (and how) the tool satisfies them.

REQ01: User-friendliness. The users of TREX have a very easy to use way of deploying their applications, as shown in Section 5. All they are required is to provide a set of input files in a web-based user interface without any need to write any scripts or XML specifications.

REQ02: Fully automated support. TREX fully automates the deployment and configuration of the application in the cloud once the required input files are provided. The user does not need to perform any manual configuration step.

REQ03: Database support. TREX supports automated creation of relational databases in the cloud, enabling users also to move an existing database to the
cloud by providing a backup file and some configuration parameters (e.g., database name, password, etc.). The application is automatically configured to connect to the provided database without any action required by the user. This was a major facilitator for users as pointed out by the study presented in Section 5.

REQ04: Multicloud support. TREXCLOUD currently supports deployments on Amazon and Rackspace and can be extended for other providers, as explained in Section 4.4.

To sum up, we can conclude that TREXCLOUD is the only approach that completely fulfills our requirements when compared to other existing deployment approaches (see Table 1).

7. CONCLUSIONS AND FUTURE WORK

One of the fundamental aspects of deploying or moving applications to the cloud is to be able accomplish this task in the easiest and effortless manner. In this work, we presented TREXCLOUD, a tool created for easy deployment of web applications in IaaS clouds. TREXCLOUD enables users to quickly deploy a web application requiring, as input, a small set of files (e.g., WAR, context and database backup files) that are automatically uploaded to the cloud and used by TREXCLOUD to set up and configure all the virtual machines that represent the complete application stack.

Our empirical results show that users of varying levels of expertise in cloud technologies (from novice to more experienced users) were able to deploy a Java web application in the Amazon cloud much faster with TREXCLOUD than with the two most common approaches used for this same purpose. TREXCLOUD outperformed the other deployment approaches with a reduction in deployment effort of up to 94%.

Our future work will focus on supporting other development platforms (e.g., .NET, PHP, Ruby on Rails) as well other database systems (e.g., MySQL, NOSQL). We also plan to incorporate self-scaling mechanisms and provide support for other IaaS clouds.

8. ACKNOWLEDGMENT

The Authors wish to thank Financiadora de Estudos e Projetos (FINEP) for supporting this research under the umbrella of the Subvenção Econômica 2009 Program. Nabor C. Mendonça is partially supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), under grants nos. 311617/2011-5 and 487174/2012-7.

9. REFERENCES


Authors

Américo Sampaio holds a B.Sc. in Computer Science from Federal University of Ceará, Brazil; an M.Sc. in Computer Science from Federal University of Pernambuco, Brazil; a Ph.D. in Computing from Lancaster University, U.K and a Post-Doc from SAP Research United Kingdom. Currently he works as Assistant Professor at University of Fortaleza and collaborates in several cloud computing projects with companies and other universities.

Roberto Costa Jr holds a B.Sc. in Computer Science from Federal University of Ceará, Brazil; and an M.Sc. in Computer Science from University of Fortaleza, Brazil. Roberto has over 10 years of experience as a software engineer and is the director of E-Novar.

Nabor Mendonça holds a B.Sc. in Data Processing from Federal University of Amazonas, Brazil; an M.Sc. in Computer Science from State University of Campinas, Brazil; and a Ph.D. in Computing from Imperial College London, U.K. Currently, he’s a Titular Professor at University of Fortaleza, Brazil, where he conducts several research projects on the use and evaluation of cloud computing technologies. Prof. Mendonça is a recipient of a Microsoft Research 2013 Software Engineering Innovation Foundation (SEIF) Award.

Ricardo Hollanda Filho holds a B.Sc. in Computer Science from University of Fortaleza and has more than 5 years of experience as a software engineer at E-Novar.