Deployment and migration of business services in the cloud

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

Autor: Bartosz Ignaczewski
Director: Jose Luis Vazquez-Poletti

Erasmus programme 2016-2017

Table of contents

Chapter 1
Introduction

Cloud solutions in business are nowadays getting more and more popular. Many companies decide to deploy their applications in the cloud or migrate them from their non cloud based solutions. It allows them to focus on functionalities and turn over the work connected with software distributed architecture and scalability issues to cloud providers.

1.1 Issue of the work

One of the most popular cloud providers are Amazon [1], Google [2] and Microsoft [3]. They allow clients to set up an account from which they can manage their infrastructures. Of course there are many ways to do it: for instance via their web-based online management tools, CLIs (Command Line Interfaces), APIs (Application Programming Interfaces). On top of them the companies and different communities built many SDKs (Software Development Kits) and libraries to speed up the process of administrating clients' resources.

As an example of those could be SDKs built by Amazon for the most popular languages and systems for the time being of writing this work [4]. However, the communities always build libraries which goals are to automate processes as far as they possibly can. So that, solely looking at the Ruby on Rails framework [5] to narrow the outcome, a couple gems (Ruby language [6] libraries) can be listed [7, 8, 9] to manage the AWS (Amazon Web Services) infrastructure, along with an application deployment, via special config files and CLIs.

However, the competitive analysis has shown that there are no well-known possibilities of managing an application infrastructure together with a deployment and migration of business services via the application own administration panel. A user who wants to communicate with an AWS has to either have an terminal access or control everything through the Amazon website. Yet, such a solution is not well-centralized and

if a user wants to use multiple cloud providers, for instance because of better pricing that it has at some other provider for one of the modules he needs to use, he has to go to every provider's website and make changes manually. Apart from being time consuming, which is costly because of both a time a worker has to spend to configure everything and passing time of using the unwanted cloud infrastructure, it is also error prone as such tasks are not automated, but reliable on a human.

The solution to these issues is to create software that allows an administrator of an application to deploy and migrate business services for multiple cloud providers from only one panel in a simplified way. That brings another advantage: because of using an user-friendly UI (User Interface) and high level of automation instead of command line interface step-by-step approach, the actions can be controlled with a lower level of knowledge and training how specific building blocks are connected and working with each other. It directly translates to lowering labor costs that comes together with the ones mentioned earlier.

This work presents a possible solution to the indicated problems. It is a web application written in Ruby on Rails framework with an administration panel that allows to deploy and migrate the system itself in the Amazon Web Services domain (along with creating a proper infrastructure). It is a prototype that in the future can be expanded to support more cloud providers and where more functionalities of a cloud base management can be added.

1.2 Scope and objective of the work

The scope of this work is a field of software engineering, which is the design and implementation of a web application with the use of available programming framework and libraries.

The objective of this work is to design and implement a web application that allows to deploy and migrate its parts to the cloud to achieve high availability and scalability of the system using the Ruby on Rails framework.

1.3 Structure of the work

The first section of this work provides a brief introduction to the subject of the cloud along with a brief presentation of the needs of its existence. It includes the scope and objective of the work.

The second chapter includes a general overview of the cloud along with application deployment and migration solutions. It presents selected solutions and points out their main advantages and disadvantages.

The third chapter presents an accurate description of the created web application. It consists of a delineation of the functional requirements of the system and what tools and technologies were used to build it.

The fourth chapter contains an extensive user documentation. It is a guide, introducing user, how it can use the built website.

The fifth chapter is a summary of all the work.

Chapter 2
Deployment and migration of business services in the cloud

The Software Development Process Life Cycle consists of many steps at the end which we have phases of deployment and maintanence, which includes migration [10]. As it is extremely crucial to focus on building great, reusable software it is also essential to remember the importance of these two last aspects, on which the cloud computing focuses.

2.1 Cloud computing overview

2.1.1 Cloud computing definition

There are many definitions of what exactly cloud computing is. The National Institute of Standards and Technology definition [11] is as follows:

„Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.”.

On the other hand, the Amazon definition [12] is described as:

„Cloud computing is the on-demand delivery of compute power, database storage, applications, and other IT resources through a cloud services platform via the internet

with pay-as-you-go pricing.”.

Looking at above it can be said that cloud computing is the on-demand service that allows management of various IT resources via the Internet which has certain characteristics and models.

As the characteristics and models are described in detail in the following sections, first a question „why do we need cloud computing?” should be answered.

2.1.2 Why do we need cloud computing?

Cloud computing, as a solution, is a response to some problems that occur during the development process in the (especially) last years. Its advantages are the best seen in large project, which require for instance high availability, performance and scalability.

As a software is being created and deployed for the first time it is usually done on a simple server that hosts the application (actually, there shall be at least two to avoid a single point of failure). The cost of usage does not change and monitoring can be done easily. As the project grows, gains more and more users and manages lots of data (getting to sizes of big data) the administrator of the server starts to spend an increasing amount of time on maintenance, monitoring. Assuming it was not done at the beginning, at some point there is a need to scale as the high performance and availability start to be endangered. There are two ways an administrator can go to scale:

1. Horizontally - add more servers to the existing ones and connect them, so that processing the workload is now distributed among all of them
2. Vertically - add more resources within the existing server, which basically means increasing the RAM amount or CPU capabilities; improving hardware specification in general

The main next things that are going to happen are (at least):

1. Setting up load balancers (if the administrator scales horizontally), which allow handling incoming traffic and redirecting to a proper server with specified algorithms
2. Adding databases instances, focusing on synchronous replication, back ups
3. Removing single points of failure
4. Solving availability and delay issues for clients from different continents

All in all, the infrastructure grows, the hardware and bandwitch costs grow, the labor costs grow, not mentioning very hard to maintain high availability and performance. One of the worst things is improper sizing, which usually cannot be well predicted.

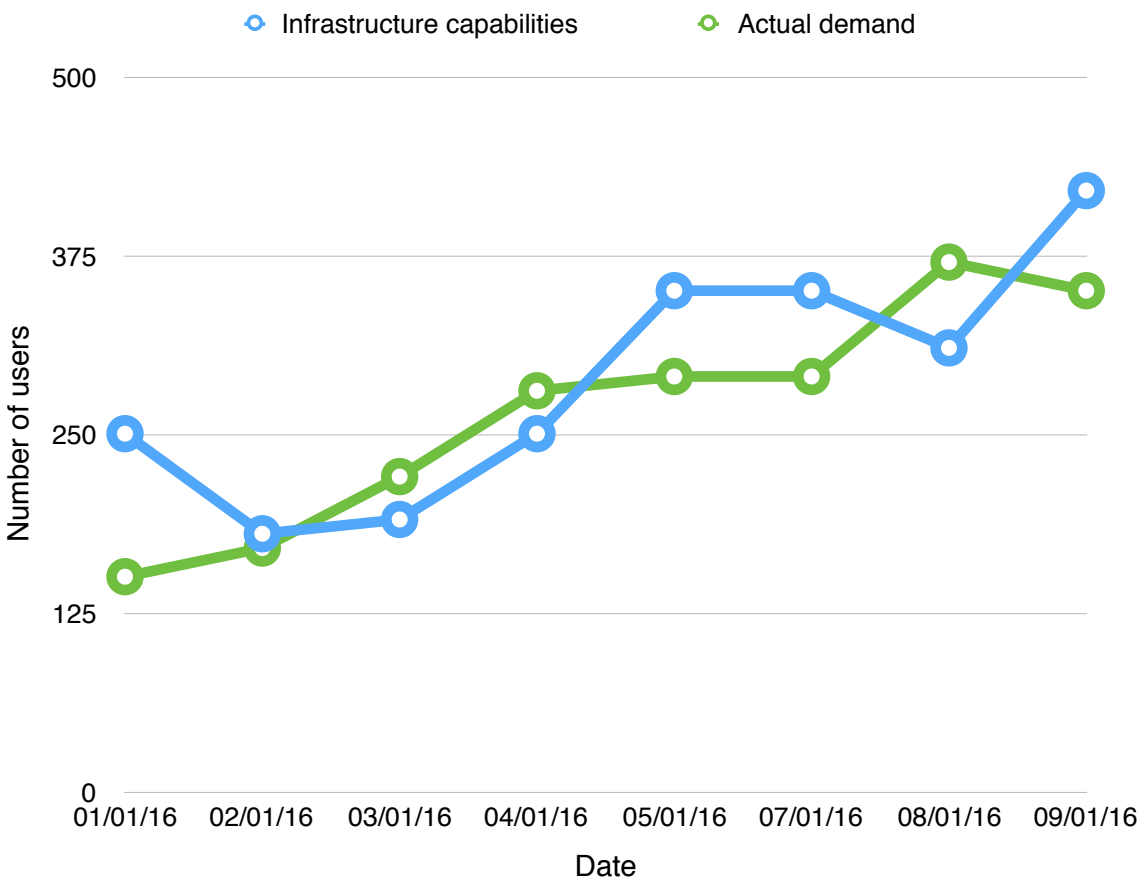

Figure 1: The infrastructure capabilities versus the actual demand

As the figure 1 presents the general idea of improper sizing based on the best administrator (or analyst) guess, it shows how this approach is cost-ineffective. It is almost impossible to foresee the actual demand, and in reality the infrastructure always

provides more computing capabilities than it should (which is a waste of money), or less than it has to (which concludes with worse performance and users' annoyance).

As time passes the costs of built data centers increase. Administrators (now many of them due to the need of supporting the whole infrastructure properly) have to upgrade hardware once in a while, apply software updates, take care about different server locations due to possible regional disasters, etc. This involves of course recruitment costs, staff training costs and many more.

Cloud computing overcomes the mentioned issues which occur during building an own data center.

2.1.3 Cloud computing characteristics

According to the National Institute of Standards and Technology's definition that was given in the previous section, it can be described with five major characteristics [11]:

1. On-demand self-service: cloud providers allow to manage IT resources, either increase or decrease computing capabilities automatically and without a need to directly contact a cloud provider
2. Broad network access: services are widely accessible throught the network and can be managable with thin or thick client platforms (for instance mobile phones, laptops)
3. Resource pooling: IT resources (physical and virtual) are shared with multiple clients (multi-tenant model) and are (re)assigned depending on the actual demand. Clients do not exacly know nor control how it is done, but may be able to choose for instance the location on a very high level of abstraction (region, country, continent)
4. Rapid elasticity: clients can request and release resources as they need to scale, which sometimes can be done automatically. Clients can request as many resources as they want, what gives a sense of unlimited capabilities

5. Measured service: clients are given exact measures for how long or to what extend they were using specified services (for example storage or bandwitch), which provides transparency about the billing.

For the last point, Amazon's definition even mentions „pay-as-you-go princing", which means that a client pays exactly for the computing capabilities that were used without any long-term commitments or prepaid expenses. Of course there are also other types of pricing models that can be chosen by a client (for instance by bidding, reserving resources or buying a dedicated host) which lower down the expenses in comparison to the on-demand model, but describing them is out of scope of this work.

Putting the major five characteristics into perspective with the given example of the own built data center, the advantages can be described as follows:

1. Cloud providers are responsible for maintaining cloud computing resources, so there is no need of hiring many administrators who take care of the data center and maintain its state, both hardware and software wise
2. The labor costs are cut with the recruitment and staff training
3. The on-demand „pay-as-you-go pricing" model allows to avoid improper sizing with not matching users demand - infrastructure capabilities amounts that are shown on figure 1
4. There are no prepaid expenses
5. Scaling can be done automatically
6. Resources are loosely coupled and located in different regions and continents (availability), so that they are less exposed to natural disasters or other failures (reliability) and conclude in better performance (lower delay in requests and responses per location)
7. Resources are easily accessible and maintanable via the network and they can be configured to suit our needs in minutes, not in hours or days

However, cloud computing also has some flaws or disadvantages. For instance, as it always happens when using solutions provided by other companies, we have less control over it and it cannot be fully customizable. In addition, cloud computing might

be not available in some regions that are cared about and because of that the provided application is blocked in a specific country. This is what happened to LinkedIn in Russia - they did not store users' data in the country, so they got blocked [13]. There might be also other legal issues that disallow from using some cloud providers and they may lack some standards.

Despite that, one of the most discussed issue of cloud computing is privacy and security. As was stated above, cloud computing uses multi-tenant model of sharing resources (both physical and virtual). Basically it means that users share the same hardware on top of which there are virtually designated spaces (operating system and virtual hardware resources) for for them, which are managed by a hypervisor (figure 2).

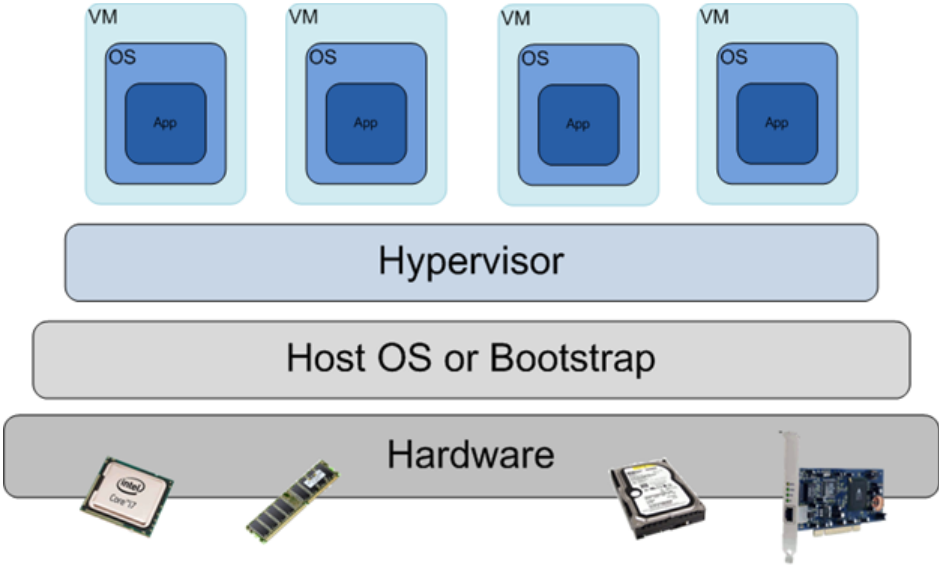

Figure 2: How virtualization works [14]

Not only the security and privacy relies on the hypervisor which has access to every operating system and application users use (so if there is a bug one user can access data of the other user), but in addition cloud providers and third parties have access to the hardware and software that are being used. It means that cloud clients completely rely on policies of providers and governments may access data easily. There are also concerns connected to erasing data and its loss. What happens if a user is assigned a storage space that belonged to a different user? Will the data be fully erased before it can access it?

Cloud providers of course describe security and privacy issues extensively [15], recommending using reliable solutions like encryption, setting up security groups and roles, multi-factor authentication, firewalls, monitoring audit logs, but there are also different deployment models which address these concerns and can be used depending on clients' needs.

2.1.4 Cloud computing deployment models

According to the National Institute of Standards and Technology definition, there are four different cloud computing deployment models: public cloud, private cloud, hybrid cloud and community cloud. Figure 3 presents the visual representation of the first three mentioned models.

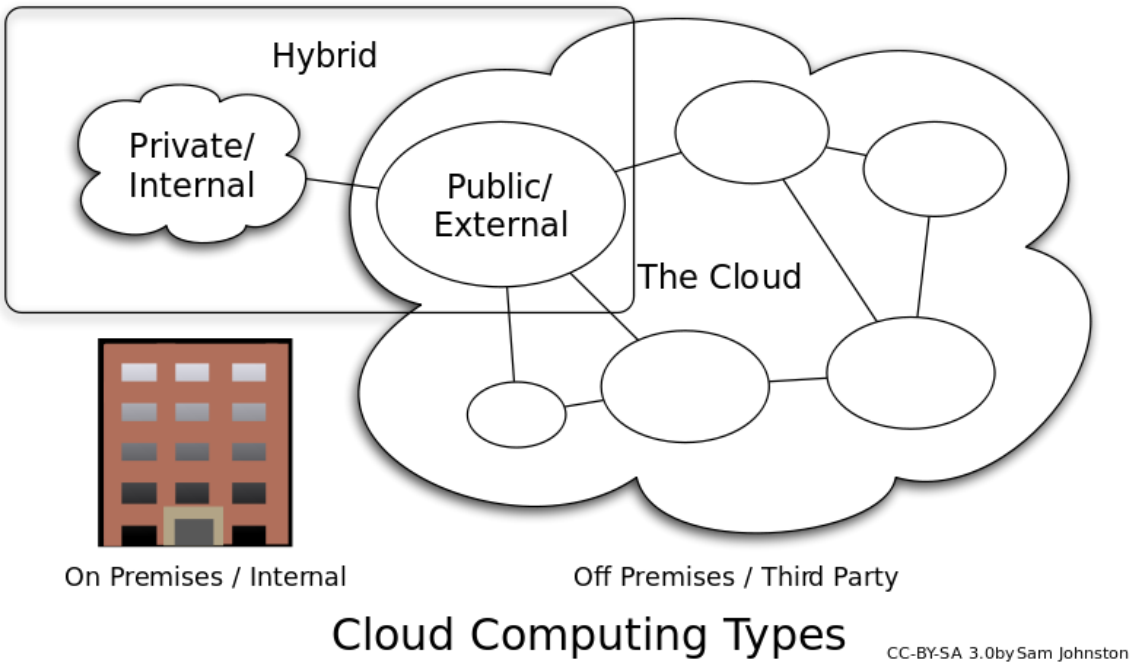

Figure 3: Visual representation of cloud computing models [16]

2.1.4.1 Public Cloud

This is the type of cloud that was characterized in the previous sections. Data centers are at locations of cloud providers (for instance Amazon, Google,

Microsoft), multiple clients share its infrastructure and it has all advantages and disadvantages that were mentioned before. All in all, it is the cheapest but the least secure type of cloud.

2.1.4.2 Private Cloud

This type of cloud is a contradiction to what public cloud represents. First of all, data centers are located either on premise or at cloud provider's location. It supports only one client and the infrastructure is not shared with anyone else. It is the most expensive cloud solution, but at the same time it is the most date-secure type of cloud and grants full control over hardware and software. It is believed that actually this model does not use the potential and of the cloud and it does not present its' major characteristics. It has many disadvantages of building own data center (which it is), including limited scalability, labor and staff training additional costs, non-elastic pricing and it is extremely hard to build multiple connected data centers around the globe.

2.1.4.3 Community Cloud

It is an „expanded" type of a private cloud: it has the same characteristics and requirements but the data center is shared between few organizations. It has better security than a public cloud type (which is open to everyone) and less cost than private cloud (because it is shared between multiple companies). It is especially helpful when businesses share the same security concerns or others. On the other hand, that requires special set of skills.

2.1.4.4 Hybrid Cloud

It is a type that is a mix of those presented above (two or more). It is very flexible and allows businesses to decide which components or parts of their systems and data should be kept on premise or remotely at cloud providers' locations.

However, businesses have to take care of a reliable and fast connections between data centers security policies compatibility.

In conclusion, different types of cloud allow businesses to choose which of them suits the best their needs. They provide incrementally more and more scalability, reliability, cost-efficiency and availability in trade of privacy and data protection in comparison to a „classic" own data center approach. Figure 4 presents their popularity.

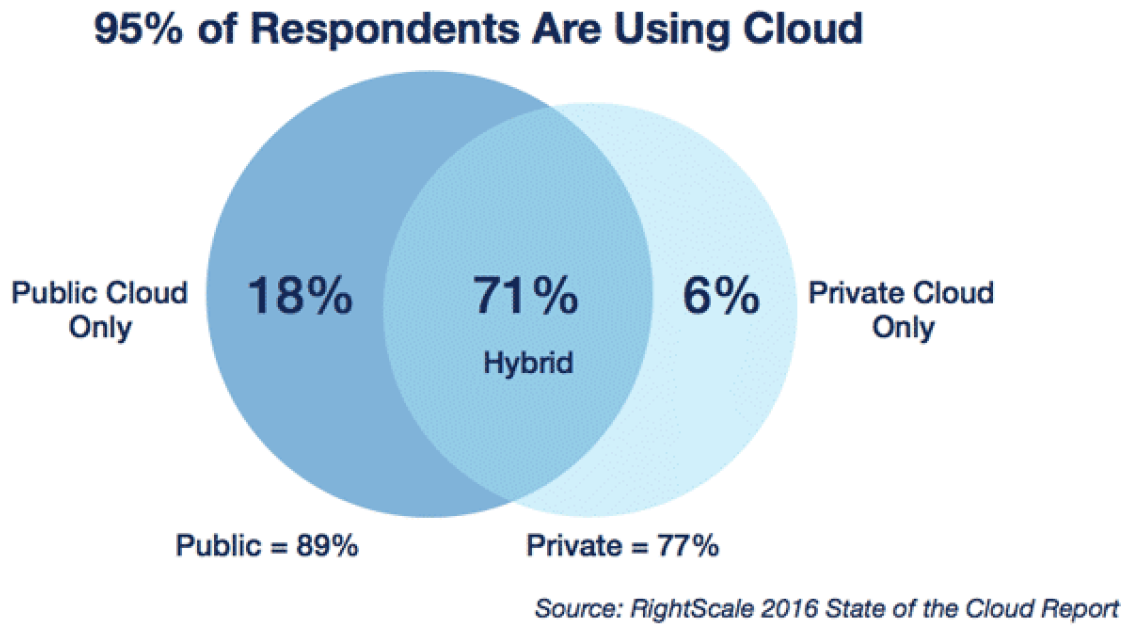

Figure 4: Cloud computing trends in 2016 [18]

Apart from virtualization, from which the discussion about privacy and security was started, cloud computing also relies heavily on service-oriented architecture (SOA). It means that cloud providers offer services which clients can utilize. However, there are different models, which are described in the following section.

2.1.5 Cloud computing service models

According to the National Institute of Standards and Technology definition, there are three different cloud computing service models: Infrastructure as a Service (Iaas), Platform as a Service (Paas) and Software as a Service (Saas). They are reffered

as the „cloud stack", because they are increasingly providing more comprehensive levels of abstraction. Figure 5 is their visual representation.

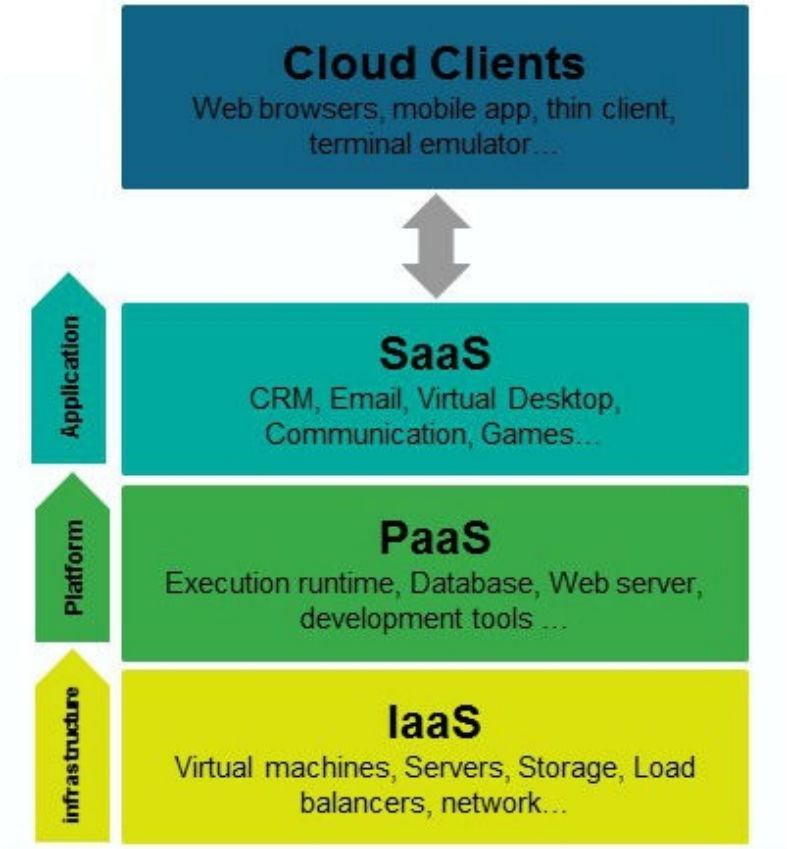

Figure 5: Visual representation of the cloud stack [19]

2.1.5.1 Infrastructure as a Service

Infrastructure as a Service, abbreviated as IaaS, is the least abstract layer of the cloud computing stack. It is mostly used by administrators as it conceptually allows to manage IT resources (infrastructure: for instance servers, storage, network components, operating systems and load balancers) with a great level of control. It directly uses virtualization to emulate proper environments which have to be handled manually. This is a model that was referenced to the most through previous chapters and sections. Practically, it is designed to be used by administrators responsible for building the whole system infrastructure from the start, perform software updates, etc.

2.1.5.2 Platform as a Service

Platform as a Service (Paas) provides an incremented level of abstraction in comparison to the IaaS model and it is designed to help developers utilize cloud by serving „out-of-the-box" infrastructure, operating system and various technology-centered execution environments so that web applications can be deployed in minutes without (almost) any additional work. It also takes care of other phases of the software development process. For instance, a developer can set up a continuous integration service for testing purposes. Theoretically in this model a loss of control in comparison to the IaaS type can be seen. However, many cloud providers allow managing the infrastructure despite using the PaaS model, as it is a „higher" layer on the stack.

The major issue with this model is a possibility of a vendor lock-in. It is one of the possible disadvantages of the cloud that was not mentioned before. Generally, when a client uses services from one cloud provider it may be very difficult to change to services offered by another one. This might be caused by for example different technologies usage or various incompatible implementations of core functionalities.

2.1.5.3 Software as a Service

Software as a Service (SaaS) is a model designed to be utilized by end users, that do not have any technical skills. Google Apps, iCloud could be shown as examples. The real control is very narrow and users are usually presented a very limited set of settings, but do not have to take care of software updates and similar tasks (for instance scaling) at all. Usually, SaaS offers subscription-based services in comparison to a „on-demand" „pay-per-use" IaaS model. There are APIs that allow integration between different systems. Vendor lock-in is here very easily seen.

Apart from the three models mentioned in the National Institute of Standards and Technology definition there are also others, for instance Network as a Service, Identity as a Service, Mobile Backend as a Service, but describing them is out of scope of this work.

This work focuses entirely on a public cloud model due to little or no expenses required to do so as well as Amazon as a provider because of the highest popularity and great community support (figure 6).

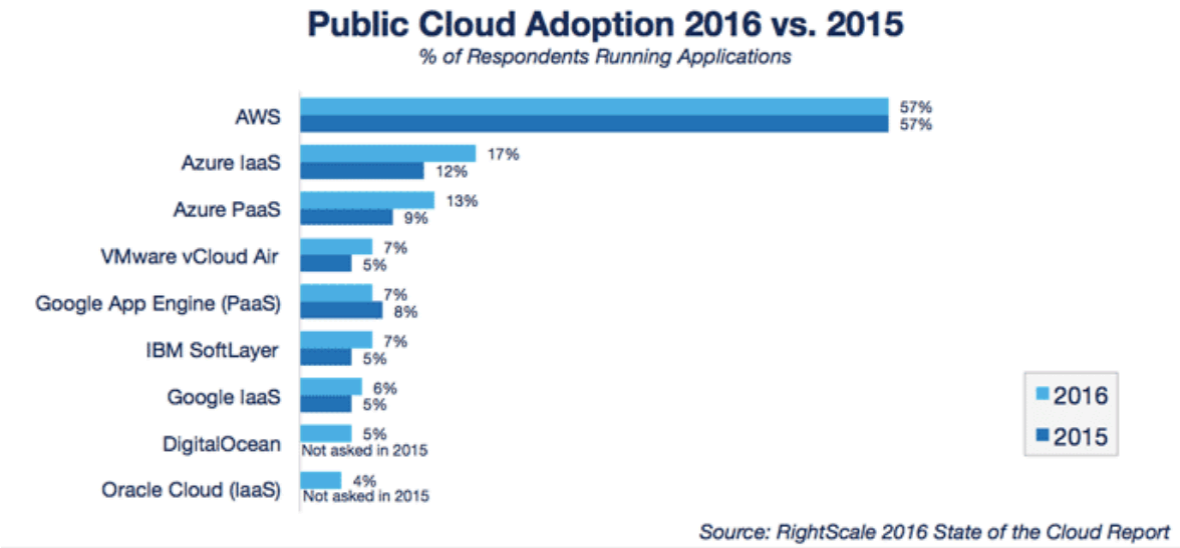

Figure 6: Public Cloud Adoption 2016 vs 2015 [18]

2.2 Overview of existing solutions

Deployment and migration in the cloud can be done in multiple ways. Not only Amazon provides tools to do so (like a web interface, command line interface, SDKs), but there are also many libraries and other solutions managed by a vast open-source community and enterprises.

2.2.1 Deployment

When focusing on a deployment it is very important to answer which service model we are going to use. As it may differ depending which one we choose, the following sections cover both Infrastructure as a Service and Platform as a Service.

2.2.1.1 Infrastructure as a Service

Amazon offers (almost) countless services to support its cloud computing solutions. However, their basic resource that is being used in the infrastructure is Amazon Elastic Compute Cloud (EC2) [20] which presents itself as a virtual server. To allow quick deployment and usage it offers predefined templates of operating systems and environments that we can choose to be automatically installed on our instances. These templates are known as Amazon Machine Images (AMIs) [21]. They can be easily customized (and then copied to other EC2 instances) and shared with other people. For example, if we would like to have support for a very customized software stack, we can look for it and there is a great chance that either Amazon or the community has already published one. If there is none, we could prepare it once and then share it with others. It greatly improves the time we need to spend on different configurations.

There are many solutions to support more automatic and easier deployment of EC2 instances. Amazon by themselves offers Amazon OpsWorks [22] which allows to use Chef [23] in order to build infrastructure by specifing everything with the code. An alternative to this is using Puppet [24], which also grants a possibility to „code" a cloud infrastructure and then run scripts that build it. The mentioned tools could be used in more ways (like a continuous integration with Chef), but it is outsite of scope of this work to present them all. An example of a Puppet code (code snippet 1):

```
ec2_instance { 'name-of-instance':
  ensure           => present,
  region           => 'us-east-1',
  availability_zone => 'us-east-1a',
  image_id         => 'ami-123456',
  instance_type    => 't1.micro',
  monitoring       => true,
  key_name         => 'name-of-existing-key',
  security_groups  => ['name-of-security-group'],
  user_data        => template('module/file-path.sh.erb'),
  tags             => {
    tag_name => 'value',
  },
}
```

Code snippet 1: Puppet code example to deploy an EC2 instance [25]

Focusing more on a Ruby on Rails framework which will be used as a main framework to build a website that will be deployed on the Amazon cloud computing platform, an equivalent solution to the presented ones could be a rubber gem [26].

EC2 also supports the more „classical" approach, so that if we have a project with a configured version control system like Git [27], we can access the server via a terminal, pull the changes and restart the server. Another way is to use special deployments gems that work the same as on a non-cloud based machines [28].

These solutions are great, but as stated earlier, Infrastructure as a Service model requires a lot of work and knowledge to set up and it is designed mostly to be used by administrators. Code changes that are required to be made by Chef or Puppet are enormous and take time, the „classical" approach does not build an infrastructure (which has to be done manually on the AWS website for instance) and in the end it does not automate the process of deployment. Not mentioning not having a proper user interface so that could be done inside a web application.

However, Infrastructure as a Service is not the only model that is used in cloud computing.

2.2.1.2 Platform as a Service

As this work focuses on building a web application, Amazon recommend using its Platform as a Service service AWS Elastic Beanstalk [29]. Its advantages are enormous. As it is designed to be mostly used by developers, it automates building the infrastructure to the level that only application deployment is necessary and the rest is handled by Amazon. It automatically adds load balancers and other components required to properly run the project. Moreover, a user still has full control of every resource it is used, as AWS Elastic Beanstalk is built on top of EC2 instances and other infrastructure blocks known from the Infrastructure as a Service model, with its advantages.

Time needed to deploy a fully working application is greatly reduced. It can be done in multiple ways: by uploading a .zip file, using command line interface or a special one specifically created to be used with Elastic Beanstalk [30]. There are even Ruby libraries to make things easier framework-wise [31, 32].

However, this solutions also do not have a user interface which could simplify things even more. A user has to be a skilled technical person to configure and run specified commands nonetheless.

In addition, all presented ways of deployment are only focusing on AWS. The application built in this work concentrates on being expanded to have a possibility to support many cloud providers. That being said, as it can use tools provided by specific providers in its underlying implementation, it cannot be fully relying on them.

2.2.2 Migration

Migration, as deployment, can be done in multiple ways. Practically, it comes down to a database creation, applying a schema that is present on a one to be replicated and copying data. As creation could be done during the deployment phase, this section focuses mostly on data replication.

Data can be migrated either with Amazon specific interfaces or using general solutions. The latter could be using rsync [33] (which Amazon recommends) that allows to transfer files or, for instance, dumping the whole database to run scripts on the other one. Although it is possible, when there is a huge data center and databases have to be available all the time or for the sake of convenience we would like to use a direct solution, Amazon provides many services do to so. Depending on what exacly we need, we could either transfer petabytes of data, connect directly to a reginal datacenter, use a Schema Conversion Tool (if we migrate from one type of database to another) or run some commands from the command line interface [34, 35].

The topic of data migration is extremely complicated and the possibilities are so many, mostly not free, that describing all of them is well outside of the scope of this work.

However, the important disadvantage of all of them is that presented solutions do not provide an easy way of data migration without a specific vendor lock-in (in this case, Amazon). There is not a well-known service that could integrate with Amazon to migrate for free the existing data within the web application’s user interface.

Because of the specified lacks of functionalities that were spotted in the existing solutions, mainly either vendor lock-in, not a free service or high complexity without a proper user interface inside an own web application to manage resources, this work presents a prototype project that integrates with Amazon Web Services and that allows easy deployment and migration. Ideally, it is designed to support multiple cloud providers and many advanced options, which could be added in the future.

Chapter 3
Project *CloudM*

Project *CloudM* was built with an idea that an application administrator, even with almost no knowledge of cloud computing or computer science in general, could use it after a short introduction of the action flow in the settings panel. It is a prototype that integrates with Amazon Web Services and allows extremely simplified deployment and migration of business services and data.

3.1 Project overview

The project consists of two main panels: end users and administrator panel. The first one is designed to be used only by „real" application users, whereas an administrator can change application settings, connect with the cloud and create an infrastructure to support it.

This section presents main functionalities and requirements used in the prototype in a high-level description. The more technical approach will be shown in the next part of this work as well as a user guide with corresponding screenshots.

3.1.1 End users panel

The end users panel is an example of an application that could be used in a real world. It is a microblogging tool with a possibility to CRUD (Create, Read, Update, Delete) posts that consist of title and body.

When a post is being created or updated, a user has to provide these two fields as the validator checks if they are not blank, otherwise displaying a message that there are errors in the form and the post cannot be saved.

The index page presents a list of posts that are currently in a database. To ease the navigation there is a pagination with a default 10 posts per page setting set.

Before any post deletion there is an additional pop up window asking if a user is sure about this decision.

As this panel is only an example where the main goal is to provide data that could be migrated to a cloud database later on, there is no need to implement more advanced functionalities like filtering posts or linking to other models. However, it is important to mention that the application is built in a way that can be easily developed in any direction in the future.

The most important part of this prototype is directly connected to the cloud infrastructure management, and the administrator panel allows it.

3.1.2 Administrator panel

Before using actions linked with changing any settings of the application a user has to log in to have privilleges to do it (posts CRUD actions do not require that). However, there is one method that can be triggered without a session: a superuser creation. The idea why it is done like this is very simple: when we create a new database in a cloud and connect to it, there is no data. To migrate data we need to go to the settings page, which is inaccessible without being logged in. As we do not have any user account created in the new database, we can do it in multiple ways: within the project interactive shell, or the database console which can be directly accessed via the SSH, but the idea of this prototype is to be as simple as possible and do things as quickly as possible. Because of that, there is a button that allows a superuser creation so that it is managable to log in within seconds after connecting to a database.

After a user logs in and goes to the settings page there are new options available. There are two sections that provide actions connected with the application deployment and migration: the Amazon RDS (Relational Database Service) [36] section and the AWS Elastic Beanstalk section. The first one is responsible for actions linked with database creation, deletion and migration, the latter with application deployment and deletion. As it has been already described in the previous chapters, AWS Elastic Beanstalk is a PaaS solution recommended to developers that allows cloud infrastructure management on a higher level than creating each component separately.

The logs from an environment creation (which is done with a single command) are one of the best ways to show how many components (along with a load balancer for instance) are set up in minutes (Figure 7).

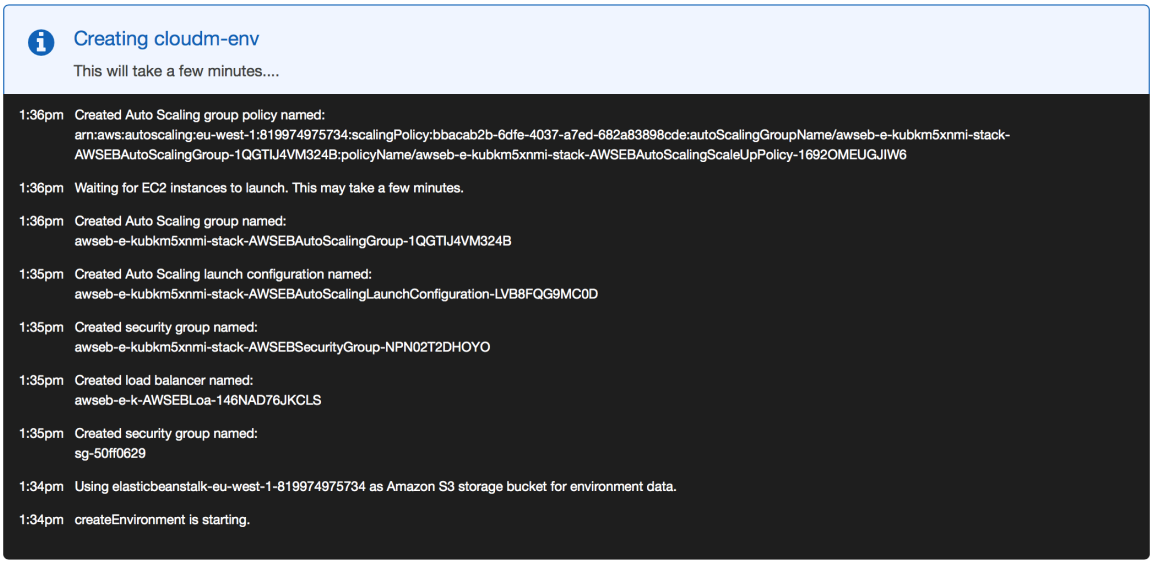

Figure 7: AWS Elastic Beanstalk logs from an environment creation

The two mentioned sections are strictly connected to each other as the application requires to have a database before it could be deployed. That being said, the requirements can be described as follows:

I. A user can create an AWS database if there is none created. When the database is being created, the corresponding information is shown: the status, possible endpoint (it is not visible for a while), to which database we are connected (local by default). After the database was created, the next actions appear to be possible to trigger:
    1. Prepare data migrations: a user can prepare data that will be migrated after it connects to the AWS database. The action takes into account all posts that are currently present in the local database and overwrites the existing migration data (if present). It is not possible to prepare a migration data without being connected to the local database.
    2. If the created database has an „available” status and an endpoint a user can connect to the newly created database.

3. If a user does not have the application deployed on the AWS Elastic Beanstalk and the AWS database has the „available” status, it can deploy one. It is important to mention that the deployed application is automatically connected to the AWS database and it cannot be changed to the local one. The deployment phase consists of two steps:
    - First, a user has to create an application in the AWS that holds the current application version.
    - Then, it is possible to create an environment which has an application endpoint (under which a user can get the fully functional prototype), status and health information.
4. If the application is deployed, a user can delete it in two steps, opposite to the previous operation:
    - A user has to terminate the environment.
    - When the environment has the status „Terminated” it is possible to terminate the application.
5. If a user connects to the AWS database it is possible to migrate the previously prepared data. After a page refresh:
    - A user sees updated messages about his current database connection
    - A data migration action is now accessible (it populates the database with a previously prepared data)
    - A „Connect back to your local database” action is available
6. If there is no environment created, the AWS database is available and a user is connected to the local database, it is possible to delete the AWS database.

In addition:

- Each action triggered by a user shows a notification as a confirmation of the action result.
- As all actions are synchronous, to see the new status of the infrastructure components (if available), a user has to refresh the page.

- Bare in mind that it is possible that a localhost application connects to the database that is in use of the AWS Elastic Beanstalk application. It this case they can both make changes in data which will be globally visible.

3.1.3 Sample use cases

As it is now clear what is possible to achieve within the application, some use cases of a real-world usage could be presented to show how useful it is.

First, let's assume that we have a project that is deployed on Google Cloud Platform (as it has multiple advantages in comparison to other cloud providers that we care about). Technologically there are no issues, but Google had to increase the prices of using its databases. The application we have serves million of users, holds a lot of data and because of that the total costs increase significantly.

One of the solutions to reduce the higher costs could be scaling down (both vertically and horizontally). As this makes users suffer and wait longer, it is far from ideal. The other option could be migration to another cloud provider, which has better pricing model. However, if we are using a PaaS solution there is a possibility of a vendor lock-in, as discussed earlier. In this case we either need a tool that is integrated with this new cloud provider (for instance Amazon) to build our infrastructure there and replicate the data we have right now, or do it manually. Not only the second option is prone to errors but also it is highly likely that it will take a lot of time (and money). The first option seems to be a very good one and this is exactly when a company could use the built prototype - to create an AWS database and migrate its data there. It can be done almost instantly and has no costs connected to it.

The other real-world use case could be linked with testing different cloud providers components' performance, availability and scalability capabilities before deciding to use a specific one.

Assuming that there is a sample application that we would like to test heavily with a lot of data and requests, we could use a tool that integrates different cloud providers solutions to set it up almost in no time (costs reduction). Normally, that would require a lot of manpower, knowledge and hours spent on configuring multiple components (for instance security groups, load balancers). However, with a built prototype

(*CloudM*) it is possible to deploy and migrate the existing sample project to a desired cloud provider (Amazon in this case) within minutes. Ideally, it could support multiple cloud providers so with a single click it would be possible to deploy the project to all of them. Then, with a special tests prepared (which also could be added as an option to the *CloudM* application), it would be viable to perform them from the main configuration panel, seeing the gathered and visualised results all together.

The described use cases are only two from many that could be taken from the real-world needs. Nonetheless, they present the potential of the built prototype, how it can be developed further and how useful (for example in terms of cost reduction) it is and could be even more.

3.2 Tools, solutions and technologies used in the project

3.2.1 Overview

Apart from using Amazon RDS and AWS Elastic Beanstalk solutions from Amazon, there are other tools and technologies that were used to build the prototype. One of the most important one is the Ruby on Rails framework [37], which uses the Ruby language [38].

3.2.1.1 Ruby on Rails framework

Ruby on Rails framework was released in 2005 and it was used to build a web application called Basecamp [39]. It is perfect for building prototypes and proofs of concepts: it allows extremely fast and easy (in comparison to other frameworks) development. For instance, in comparison to Java and its long configuration files in XML, a database connection is as simple as presented on the Code snippet 2.

However, it is not only the configuration files that makes it so good. For instance, creating CRUD controller actions for a new model with a database migration, views,

required routes and much more (like a JSON [41] support) is done with one single, simple command (Code snippet 3). It is called *scaffolding*.

```
production:
    <<: *default
    adapter: postgresql
    encoding: unicode
    database: <%= ENV['RDS_DB_NAME'] %>
    username: <%= ENV['RDS_USERNAME'] %>
    password: <%= ENV['RDS_PASSWORD'] %>
    host: <%= ENV['RDS_HOSTNAME'] %>
    port: <%= ENV['RDS_PORT'] %>
```

Code snippet 2: Ruby on Rails database connection configuration code

```
Bartek@MacBook-Pro-Bartosz ~/D/maslo> rails generate scaffold post title:string body:text
Running via Spring preloader in process 84553
      invoke  active_record
      create    db/migrate/20170317115726_create_posts.rb
      create    app/models/post.rb
      invoke    test_unit
      create      test/models/post_test.rb
      create      test/fixtures/posts.yml
      invoke  resource_route
       route    resources :posts
      invoke  scaffold_controller
      create    app/controllers/posts_controller.rb
      invoke    erb
      create      app/views/posts
      create      app/views/posts/index.html.erb
      create      app/views/posts/edit.html.erb
      create      app/views/posts/show.html.erb
      create      app/views/posts/new.html.erb
      create      app/views/posts/_form.html.erb
      invoke    test_unit
      create      test/controllers/posts_controller_test.rb
      invoke    helper
      create      app/helpers/posts_helper.rb
      invoke      test_unit
      invoke    jbuilder
      create      app/views/posts/index.json.jbuilder
      create      app/views/posts/show.json.jbuilder
      create      app/views/posts/_post.json.jbuilder
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/posts.coffee
      invoke    scss
      create      app/assets/stylesheets/posts.scss
      invoke  scss
      create    app/assets/stylesheets/scaffolds.scss
```

Code snippet 3: Ruby on Rails scaffold generation

What is more, Ruby on Rails uses the *Convention Over Configuration* rule [40], which adds a very high usage of conventions (for instance naming) to avoid unnecessary configurations (in addition to their simplicity). For example, the *app/views/posts/** files are directly connected to the CRUD controller actions with the same names (except of *app/vies/posts/_form.html.erb,* which is used as a partial in other views).

In addition, Ruby on Rails libraries and possibilites can be easily extended with many plugins (*gems*) that are built every day by an extremely enthusiastic and active community. They are open-source and adding them to a project takes basically no time. They are very different: from supporting images uploading and storing, through views bootstrapping and user logging to integrating other systems almost automatically.

On top of that there is a *DRY* rule (Don't Repeat Yourself) [42] which stands for not creating the same work in many places - instead, create something once and then reuse it, which Ruby on Rails uses heavily.

There are also some other advantages which description is out of scope of this work, but it would be impossible not to mention the Ruby language on top of which the Ruby on Rails framework was built. In general, its most important characteristis relevant for this work are: it is fully object-oriented, dynamic, scripting language with an interactive shell that can use it. What is crucial, the amount of code needed to create a specific functionality is very small in comparison to other languages like Java. For instance, to get the current date and time in Ruby, all we have to do is executing a „*Time.now*" command in the interactive shell, whereas in Java we need to write a lot more (Code snippet 4) and it has to be compiled first.

```
DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
Calendar cal = Calendar.getInstance();
System.out.println(dateFormat.format(cal)); //2016/11/16 12:08:43
```

Code snippet 4: Getting the current date and time in Java [43]

All these advantages made me choose the Ruby on Rails framework as the main technology to build the *CloudM* project - as it is a prototype which had to be developed as quickly as possible. There are also other points why I decided to use it and they are mentioned in the next sections. However, as important as framework selection was setting up the Amazon Web Services platform and choosing how to use it.

3.2.1.2 Communication with Amazon Web Services

There are many ways to manage Amazon Web Services components. It can be done for instance via the command line interface, directly through their website, mobile application or the REST API which can be used in the Ruby on Rails framework project by adding the AWS SDK gem [44]. This basically gives access to all possible services and the AWS SDK documentation profoundly describes them.

In the *CloudM* prototype I decided to use the REST API, as it is the most common way to integrate with other systems and it is officially supported by Amazon.

However, to use the SDK we need to set up a new account on Amazon, create special keys that will be used for authentication and pass them in the requests. As in the requirements for the prototype there is no action connected with giving the API keys (because storing passwords is a whole different topic to address and it is not easy to do), Amazon's documentation [44] actually mentions that the best way to do it is to create a secret file that contains them and is ignored by our source control system (Git [45] in this case). As the administrator panel uses only one account and there are no more accounts, it is a perfect way to add the API keys to be authenticated by Amazon.

Using Git as a source control has many advantages which describing is out of scope of this work. Despite that, there is one special favor linked to it which will be presented in one of the next sections.

As it is clear what is the main framework and a way of communication with Amazon Web Services in the *CloudM* prototype, there is a need to focus on more specific tools and solutions that are implemented. The description is divided into two parts: one for the end users panel and the other one for the administrator panel, which itself has two more sections.

3.2.2 End users panel

End users panel presents CRUD actions along with corresponding views. It was built with a modification of a default scaffold template. First, Bootstrap [46] was used to

gain interactive, reusable components to create the user interface. Then, some other gems (Ruby libraries) were put into the prototype:

- gem 'bootstrap-sass' [47] : to add Bootstrap
- gem 'haml-rails' [48] : to support HAML markup language [49] instead of the default one
- gem 'simple_form' [50]: a Bootstrap integration with forms in views
- gem 'bootstrap-generators' [51]: a Bootstrap integration with scaffolds
- gem 'kaminari' [52]: allows adding pagination in views for models
- gem 'bootstrap-kaminari-views' [53]: a Bootstrap integration with Kaminari

On top of that, the whole project uses gem 'growlyflash' [54] which is responsible for notification (flash) messages.

3.2.3 Administrator panel

Apart from Bootstrap, Simple Form, Growlyflash and HAML that are used in the project globally, administrator panel also implements mentioned gem 'aws-sdk-rails' (for AWS REST API connections), gem 'devise' (for authentication) [55] and a gem 'devise-bootstrap-views' (for Bootstrap integration with Devise) [56].

As the administrator panel consists of two parts, each of them has its own solutions and additional gems that were used to build it.

3.2.3.1 Amazon Relational Database Service section

This section contains information and actions linked to a database connection, data migration and the cloud infrastructure management.

Database creation is based on sending specific parameters to Amazon Web Services: the engine is postgres [57] and, what is important when using a free tier, the instance class is 'db.t2.micro'. Other key - value pairs contain information about istance identifier, name, user name and password.

After the database was created there is a need to run schema migrations. This is the time when the Ruby on Rails framework is in favor: to do it, all what has to be done is invoking a 'db:migrate' command.

It looks similar when it goes to preparing and running data migrations. Ruby on Rails has special files and tasks to support it. As for the data preparation a gem 'seed_dump' [58] was used to dump currently held data to a file, whereas invoking migrations is as simple as running a 'db:seed' task (Code snippet 5).

```
def invoke_data_migration
  Post.delete_all
  Rails.application.load_tasks
  Rake::Task['db:seed'].invoke
  redirect_to app_settings_path, notice: 'Data migrated'
end
```

Code snippet 5: Invoke data migration method

Other methods, for instance connecting to a database (local and Amazon RDS), its deletion and getting its information are implementations that use either AWS REST API or Ruby on Rails methods.

3.2.3.2 AWS Elastic Beanstalk section

When creating and deploying the project to the cloud it is important to upload a current version of the application. This is done with a help of two solutions (apart from the REST API): Git and AWS CodeCommit [59], which allows to store the Git repository.

In the first part of the project deployment (application version creation), we need to specify how we are going to upload the project. It could be either a *.zip file (which can be also generated with Git [$ git archive --format=zip HEAD]), or an AWS CodeCommit repository. Since generating a *.zip file connects with uploading it to some server what can take a lot of time, I decided to use the AWS CodeComit service. Creating an application version in this case is straightforward and it is based on a created repository (Code snippet 6).

```ruby
def self.create_application_version
  latest_commit_id = get_latest_commid_id
  EB_CLIENT.create_application_version({
                                         application_name: APP_NAME,
                                         version_label: latest_commit_id,
                                         description: 'UCM final project app',
                                         source_build_information: {
                                             source_type: 'Git',
                                             source_repository: 'CodeCommit',
                                             source_location: 'cloudm/' + latest_commit_id,
                                         },
                                         auto_create_application: true,
                                         process: true
                                       })
end
```

Code snippet 6: Create application version method

As the application and its version are created, now it is time to create the environment which will have an endpoint that will allow to access it.

It is extremely impotant to remember that in the production environment we are using environment variables (Code snippet 2) that are referencing information about the database connection details. Without them (and an existing database) some scripts will not work (for instance those responsible for schema migrations) and the whole deployment will fail. To generate them, we need to create special options (Code snippet 7).

```ruby
def self.generate_db_opts(db_hostname = get_db_endpoint_info[1], db_name = DB_INSTANCE_IDENTIFIER,
    db_username = DB_USER_NAME, db_user_pass = DB_USER_PASS, db_port = DB_PORT)
  [{
       namespace: 'aws:elasticbeanstalk:application:environment',
       option_name: 'RDS_DB_NAME',
       value: db_name
   },
   {
       namespace: 'aws:elasticbeanstalk:application:environment',
       option_name: 'RDS_USERNAME',
       value: db_username
   },
   {
       namespace: 'aws:elasticbeanstalk:application:environment',
       option_name: 'RDS_PASSWORD',
       value: db_user_pass
   },
   {
       namespace: 'aws:elasticbeanstalk:application:environment',
       option_name: 'RDS_HOSTNAME',
       value: db_hostname
   },
   {
       namespace: 'aws:elasticbeanstalk:application:environment',
       option_name: 'RDS_PORT',
       value: db_port
   }]
end
```

Code snippet 7: Generate database environment variables method

There are also other parameters that are being sent to Amazon when making a request, for example the solution stack name which tells what type of system and preinstalled software we would like to have on a server (as AWS Elastic Beanstalk is a PaaS solution) or the instance type (Code snippet 8). Notice how we use the *generate_db_opts* method that is presented earlier (Code snippet 7).

```
({
    application_name: APP_NAME,
    environment_name: env_name,
    version_label: get_latest_commid_id,
    solution_stack_name: '64bit Amazon Linux 2016.09 v2.3.2 running Ruby 2.3 (Puma)',
    option_settings: [
        {
            namespace: 'aws:elasticbeanstalk:application:environment',
            option_name: 'SECRET_KEY_BASE',
            value: SecureRandom.hex(64)
        },
        {
            namespace: 'aws:autoscaling:launchconfiguration',
            option_name: 'InstanceType',
            value: 't2.micro'
        }
    ] + generate_db_opts(db_hostname, db_name, db_username, db_pass, db_port)
})
```

Code snippet 8: Environment creation parameters

Chapter 4

Project *CloudM* from the point of view of a user

As the application is designed to be used by two different types of users, this chapter is divided into two sections: one that is designed to be read by end users and the other one is meant to be for administrators.

Ideally, the prototype would already run in the cloud (with a provider different than Amazon), but for the purpose of this work it is running locally.

4.1 End users panel

When a user enters the website, the main *Microblog* application is presented (Figure 8).

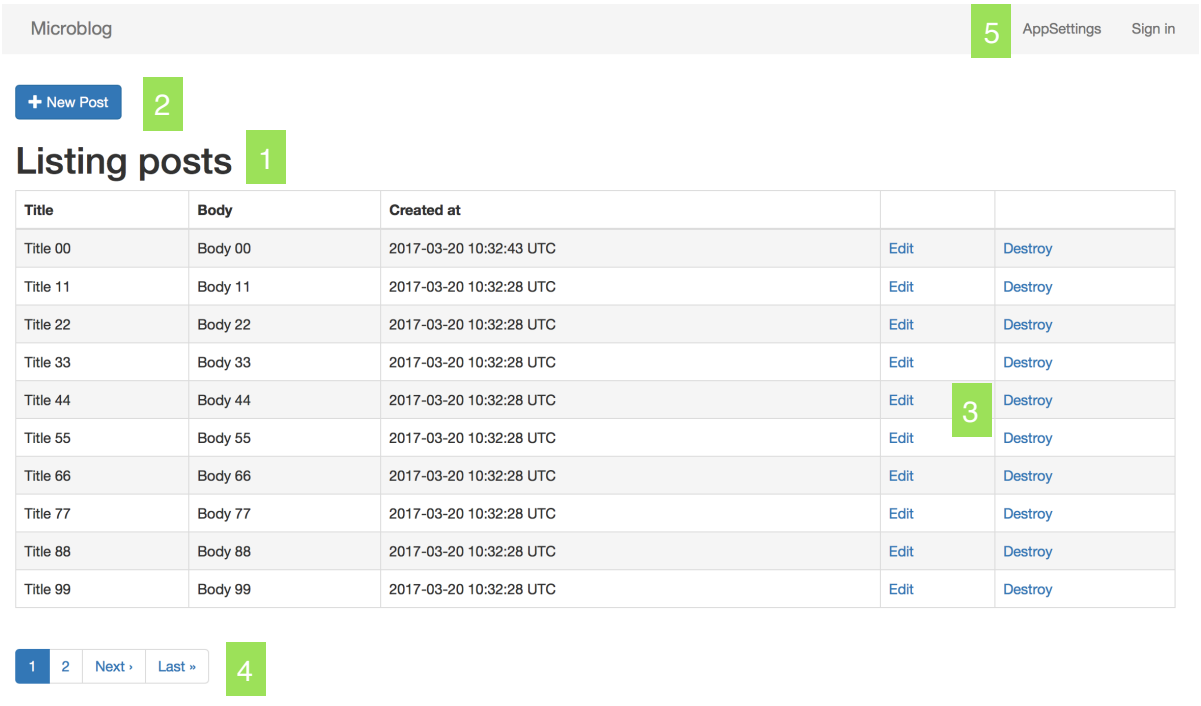

Figure 8: The *Microblog* application main page

It consists of posts listed by descending *Created at* attribute (1). There are also buttons and links to Create (2), Update and Delete (3) actions. On the bottom left hand corner there are pagination links (4). By default, there are ten posts listed per page.

As the user wants to create a new posts, it clicks a *New Post* button (2) and it is redirected to another page (Figure 9).

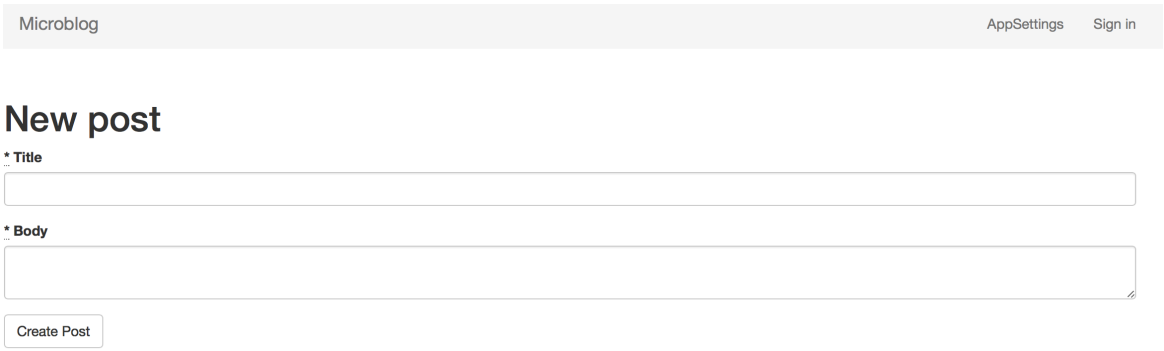

Figure 9: The *New post* page

To create a new post it is required to fill in the two text fields: *Title* and *Body*. If there is no content in any of them, there is an error as there is a validation on a *Post* model (Figure 10).

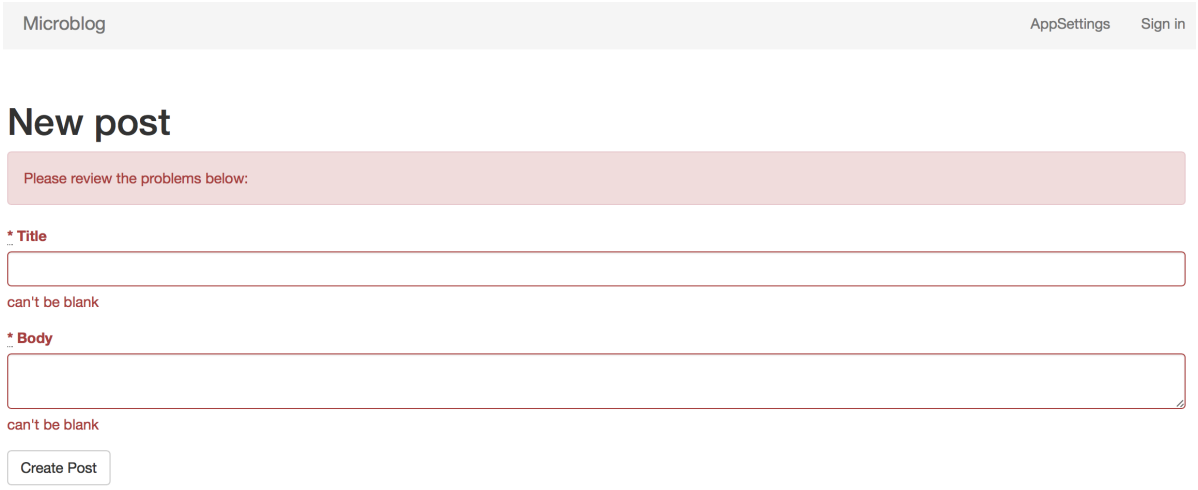

Figure 10: The *New post* page validation errors: *can't be blank* for the *Title* and *Body* fields

After a user enters values into both of the text fields, it is redirected to a main page with a notification (flash message) in the top right hand corner (1) about the successful action (Figure 11). Notice the new post listed as a first one (2).

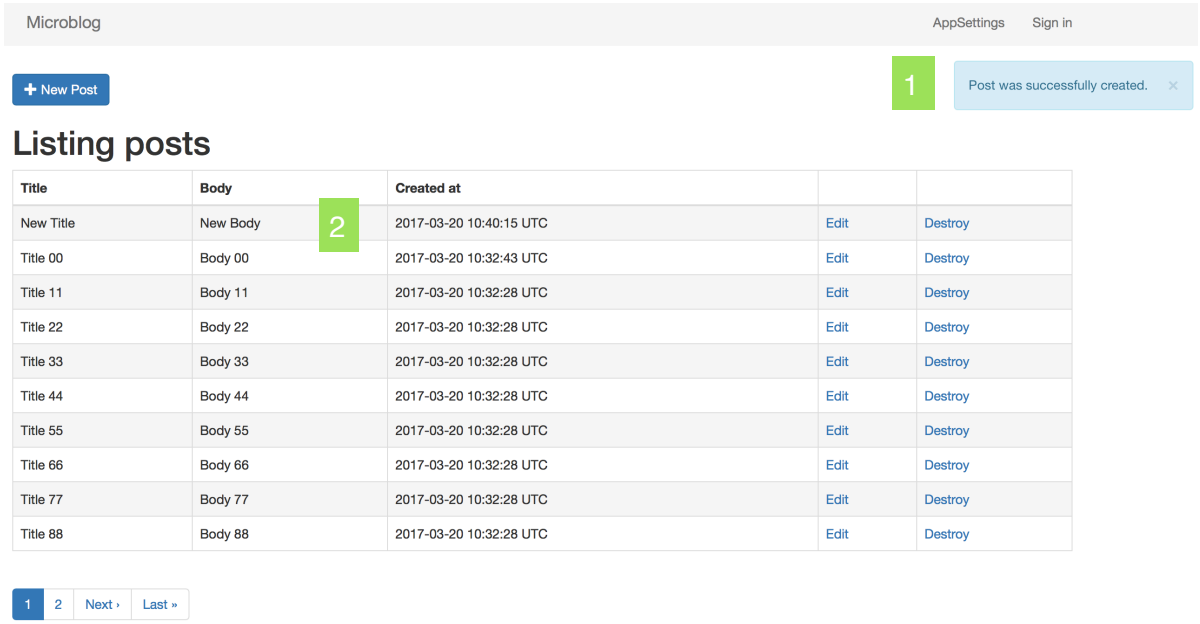

Figure 11: New post was created

The *Edit* action redirects to a *Post* form shown on Figure 9, but the difference is that the values of the current post are already filled in the text fields.

The *Destroy* action performs a post deletion, but before that a user is asked if it is sure that it wants to remove it (Figure 12).

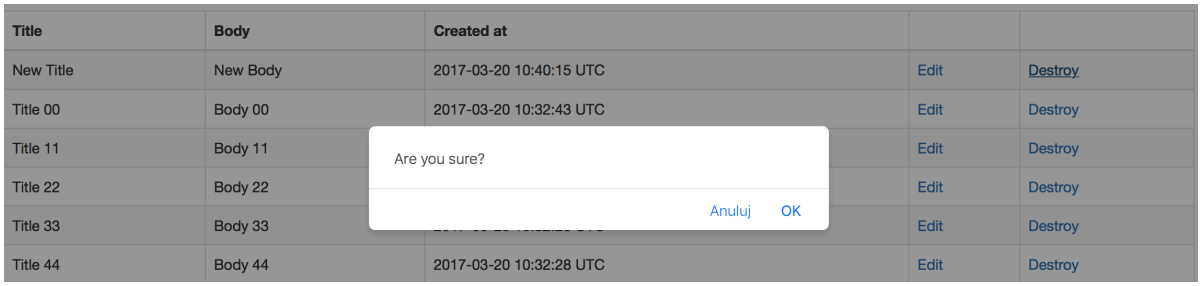

Figure 12: The confirmation dialog for a post deletion

4.2 Administrator panel

There are two additional links in the navigation bar present on Figure 8 (5): *AppSettings* and *Sign in*, which are designed to be used by an administrator.

To manage the settings and configuration of the cloud infrastructure and the application itself, an administrator has to log in first. After clicking *Sign in,* it is

redirected to a form where it can do that (Figure 13). Additionally, it can check the *Remember me* checkbox (1) to hold a session.

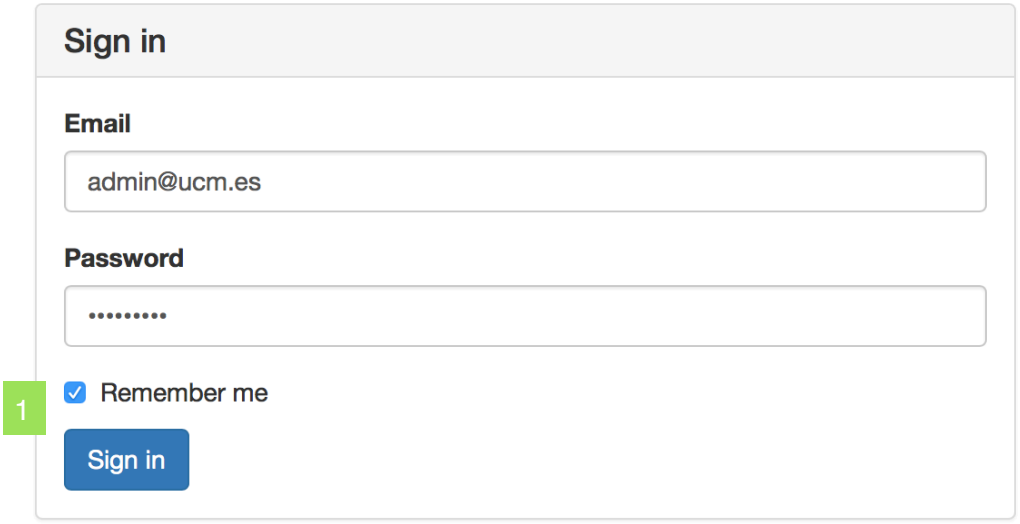

Figure 13: The administrator log in form

Of course, if a user name and password do not match, there is a flash message informing about it.

When an administrator is logged in, it has a full control over the settings. Assuming that the application was never deployed to the cloud, basic actions are available (Figure 14).

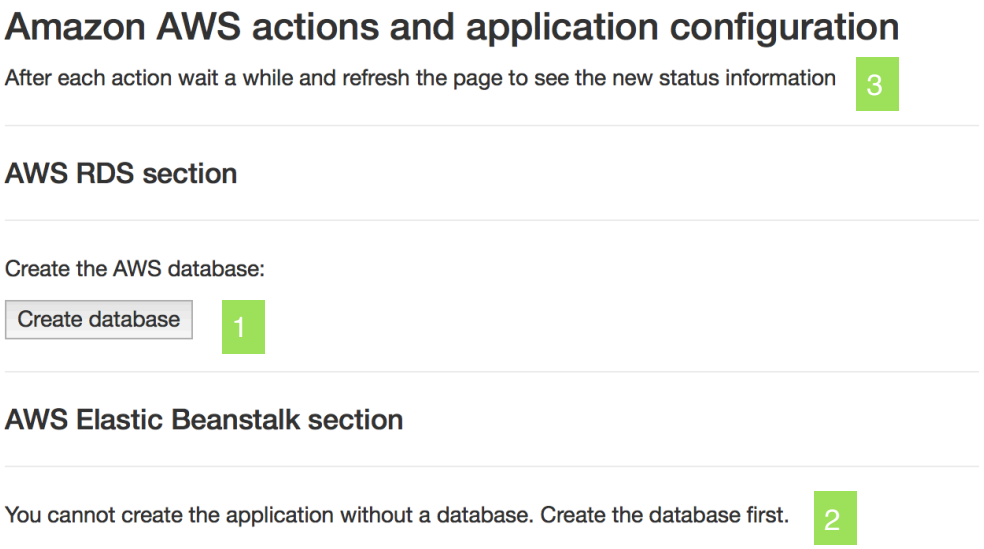

Figure 14: The administrator's *AppSettings* page

Apart from the *Create database* button (1), there are some information how the application has to be used. For instance, to deploy the application a database has to be created first (2) and to get a new status information the page has to be refreshed (3).

After the *Create database* action was triggered, new options became available (Figure 15).

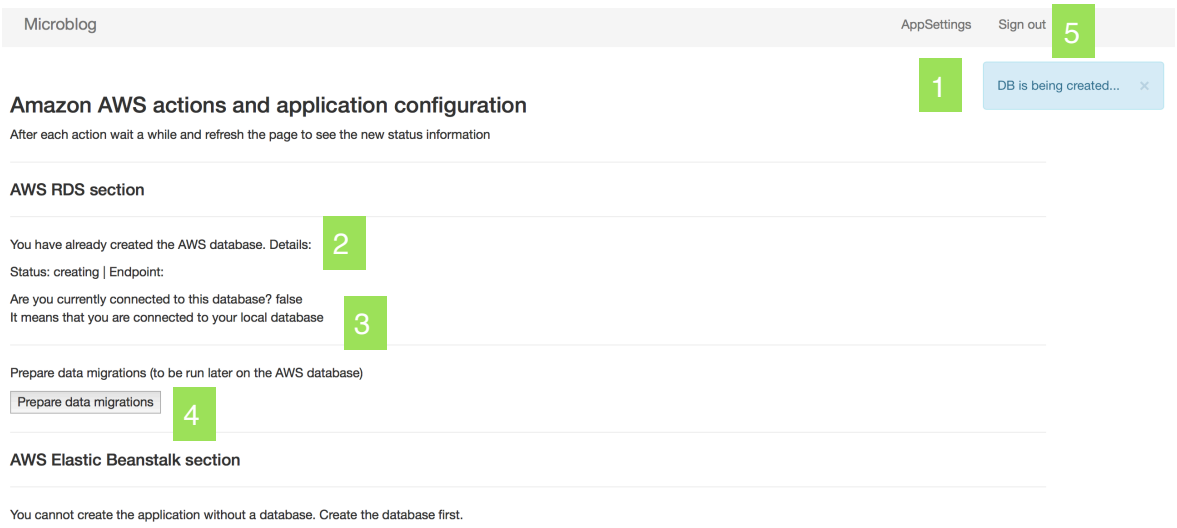

Figure 15: The *AppSettings* page after clicking the *Create database* button

First, there is a notification about the action taken (1). Then there is new information about the database: its status and an endpoint (as Amazon did not set it up yet, it is blank) (2). The administrator can see to which database the application is currently connected (3) and it can *Prepare data migrations* (4) that can be used later on to migrate current data to the AWS RDS. Notice that the *Sign in* link is now replaced by *Sign out* one (5): it is present since the administrator logged in.

After a while the page can be refreshed and it can be seen new status, endpoint information (1), possibility to connect to the AWS database (2) and to delete it (3) (Figure 16).

When the administrator connects to the newly created database, it cannot log in because there is no data on it. To create the administrator account it can go to *AppSettings*, which has an action to do it (Figure 17).

If the administrator wants to see on AWS RDS website the information about the newly created database, it can do it (Figure 18). Notice the endpoint address (1), one current connection (2), the instance class (3) and the status (4).

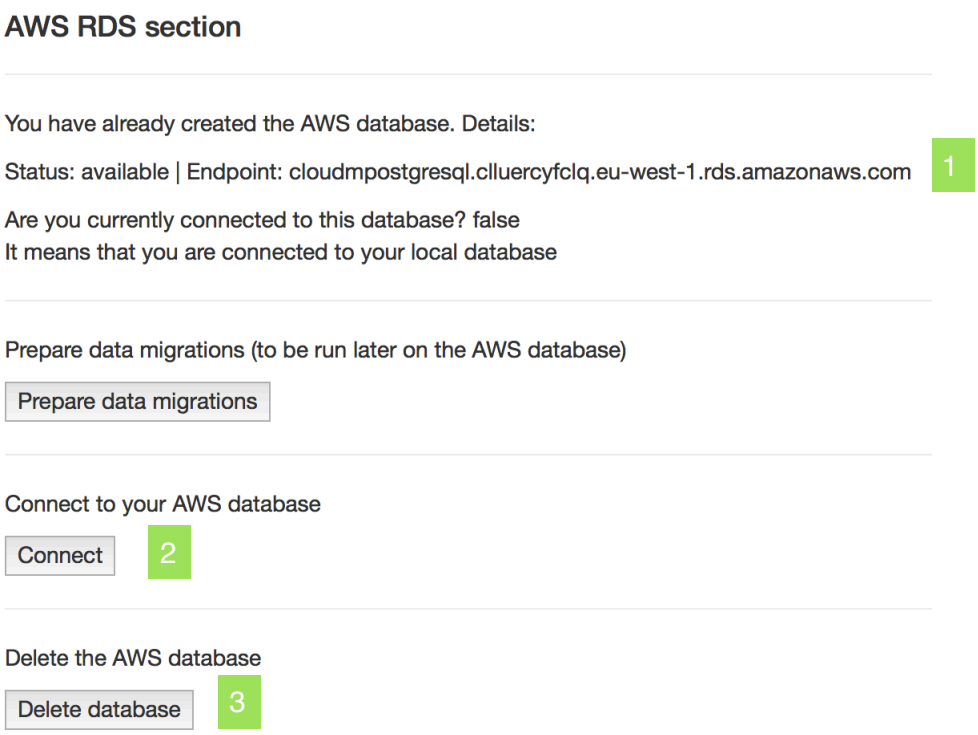

Figure 16: Visible options after the database creation (with the *available* status) on the *AppSettings* page

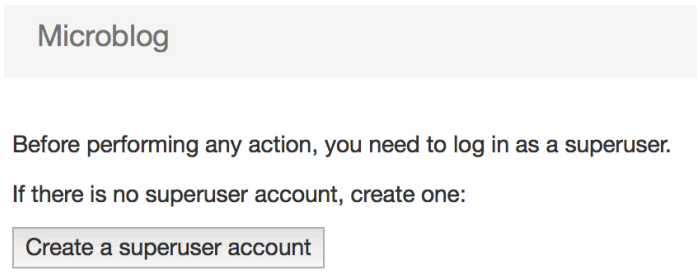

Figure 17: The *AppSettings Create a superuser account* button and the information about it

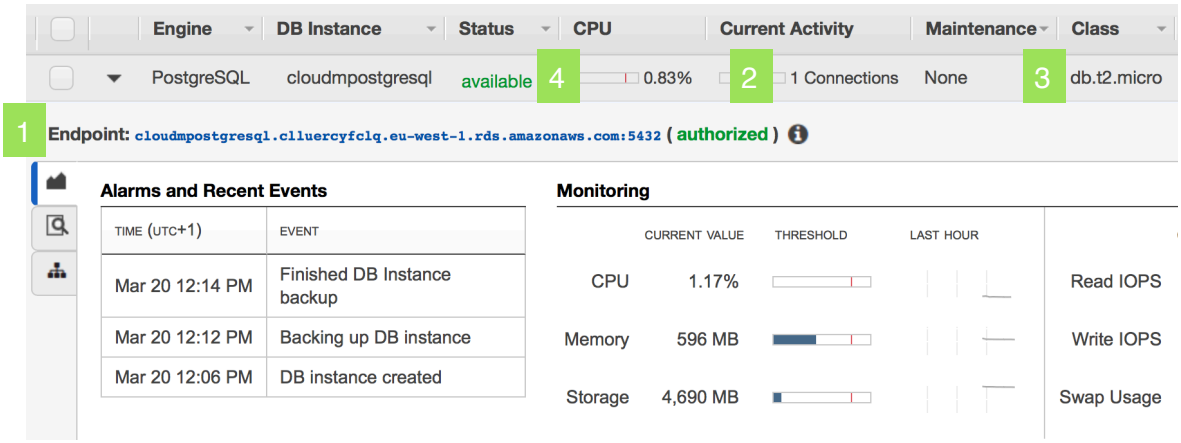

Figure 18: The AWS RDS database information

After the account creation and logging in, the information about the current database connection is updated (1), the administrator has a possibility migrate previously prepared data (2) and reconnect to the local database (3) (Figure 19).

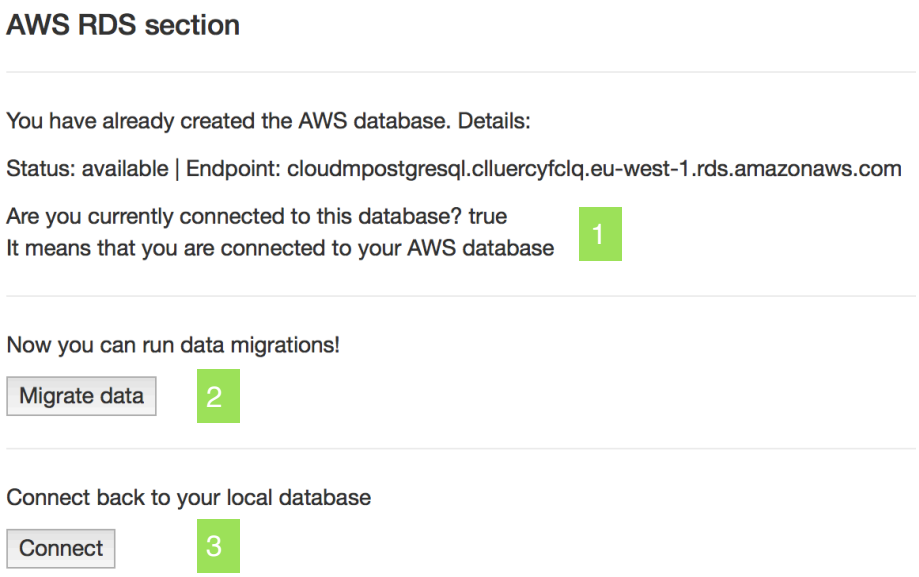

Figure 19: The *AppSettings* page after connecting to the AWS database

After the database has an *available* status, the AWS Elastic Beanstalk section has a *Create application* action available (Figure 20).

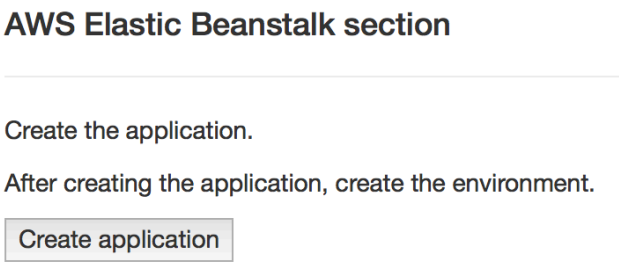

Figure 20: The *Create application* action button (after the full database creation process)

After the administrator triggered a *Create application* action, it can create the environment to fully deploy it (Figure 21).

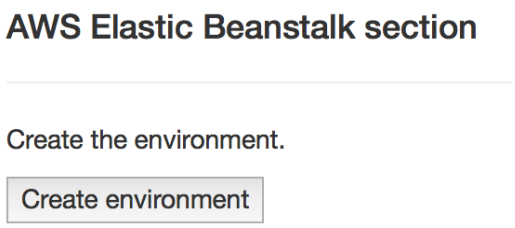

Figure 21: The *Create environment* action button

After the environment has been successfully created, new information is present on the *AppSettings* page (Figure 22).

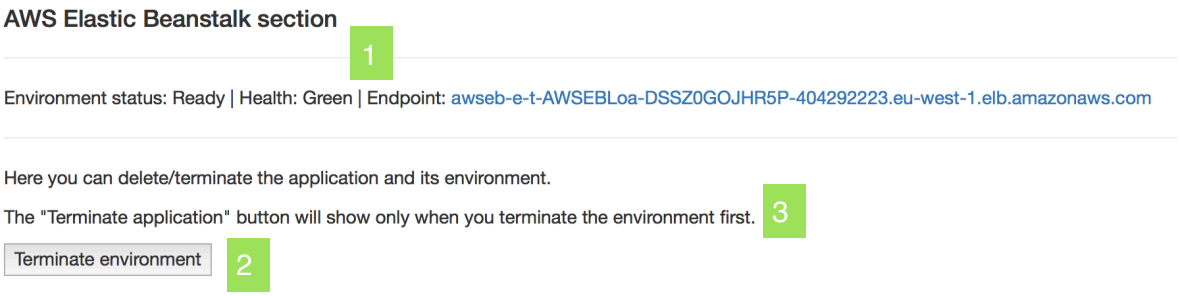

Figure 22: The AWS Elastic Beanstalk section after full application deployment

Notice the status, health and endpoint (1) as well as the action linked with termination the environment (2). To terminate the application, first terminate the environment (3).

The AWS Elastic Beanstalk console shows almost the same information (Figure 23): endpoint (1), health (2), configuration (3).

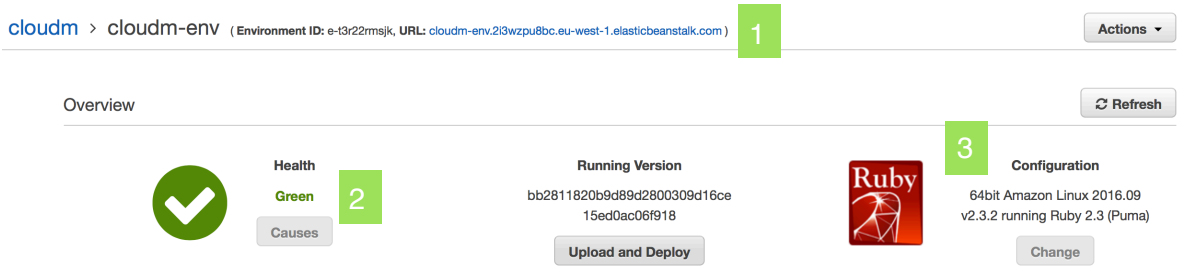

Figure 23: The AWS Elastic Beanstalk console information

When the administrator clicks the endpoint URL, it is taken to the fully deployed application website (Figure 24). Notice the URL (1).

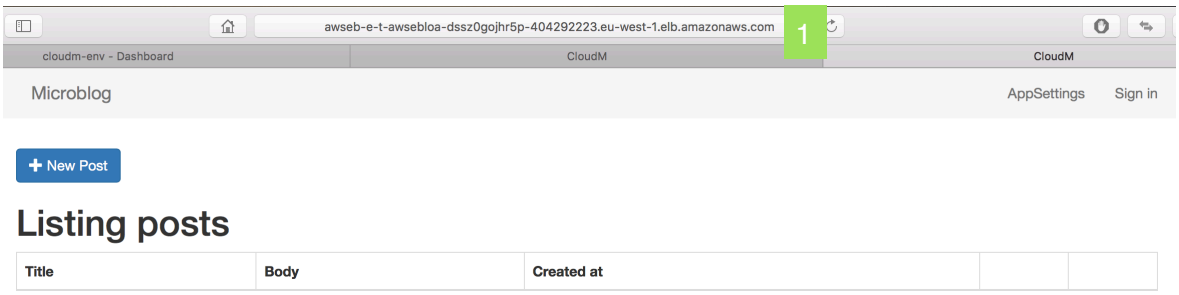

Figure 24: Fully deployed application is available at the endpoint URL

The *Delete database* button action was present on the Figure 16 (3), but it is missing now. The application is fully deployed and it uses the created database - currently it is not possible to terminate it (even if the application is connected to a local database): the administrator has to terminate the environment and the application first (Figure 25).

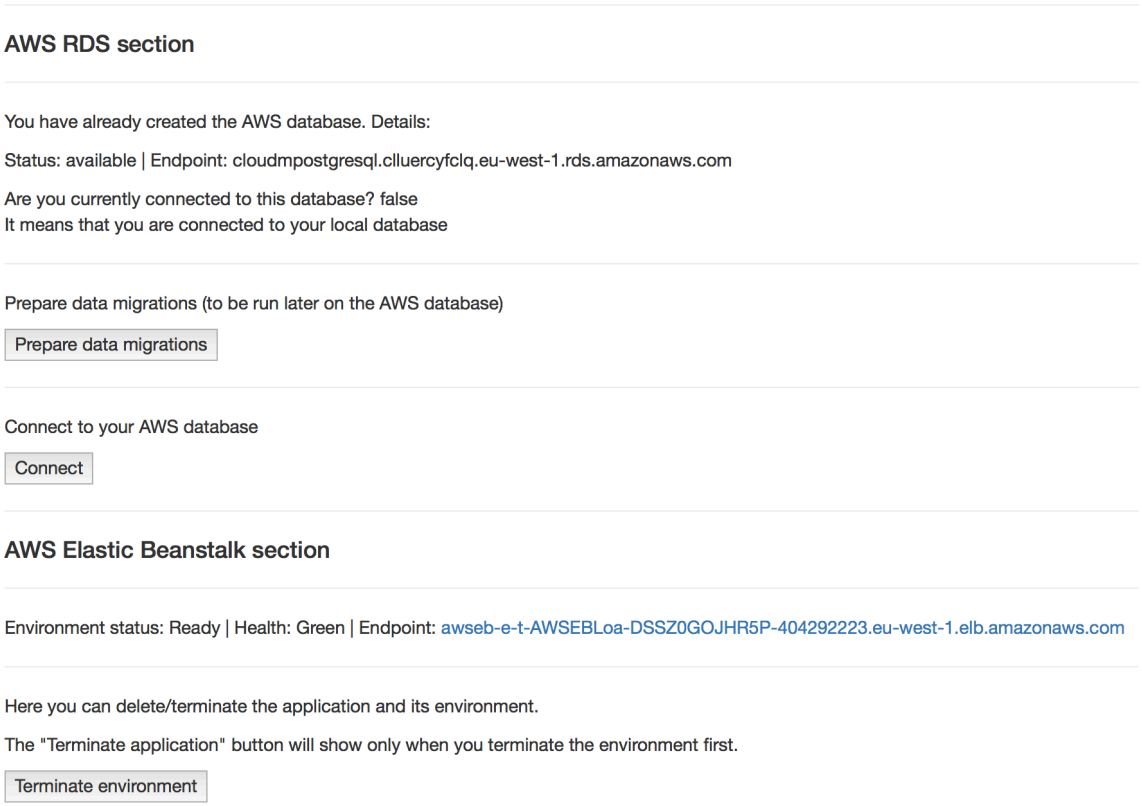

Figure 25: No *Delete database* action button when the application is fully deployed on the *AppSettings* page

After the *Terminatie environment* action is triggered, the application can be deleted only when the environment is fully removed (Figure 26).

Figure 26: The *Terminate application* button shows up only when the environment status is set to *Terminated*

Chapter 5
Conclusion

Cloud computing is a very broad and advanced topic that consists of a lot of fields in computer science. Due to Internet accessibility and speed it is very convenient (and affordable) to use it. Many people do not even know that they do it on daily basis: by storing images in their iCloud accounts, sending emails through Inbox by Google and so on.

In this work there are presented many aspects of cloud computing. Some of them were deployment and migration of business services and data. They are crucial, as they are the first steps to take to build the whole infrastructure before end users can operate. Many businesses struggle with them as it is not obvious either how to make them or how to change from one cloud provider to some other one. They might want to do it for many reasons: the technologies that are implemented are not reliable or not updated, the pricing models may be undesirable for the possible gain, the security methods are insufficient to protect company's data.

Because of that the *CloudM* prototype was created: to present users quick and simple way to choose and use given cloud solutions. It is an integrator, which is only responsible for communication with cloud provider(s). The current version supports database creation and deletion, data migration, application deployment and termination in the Amazon Web Services platform. It also contains a sample *Microblog* application that allows posts create, update, delete and read actions to test how well the mentioned integration was done.

The project can be easily extended to add more providers, more sophisticated options (for the REST API parameters to be set), asynchronous calls with an additional server that is designed to use queues and background jobs and so forth.

All this work was done within the Ruby on Rails framework, which is designed to build prototypes and proofs of concepts (which the *CloudM* project is) as quickly as possible, focusing the most on what is needed to be build, not how to do it on a very

detailed level. There are many advantages of this technology, especially in comparison to some existing solutions.

The Ruby on Rails framework would not exist if it was not for the Ruby language, which allows extremely fast functionalities creation mainly due to the fact that it requires very little code (in comparison to Java for instance) to implement them.

Thanks to the mentioned technologies the *CloudM* prototype was created and it consists of many operations that allow the management of the cloud infrastructure and data migrations.

List of figures

List of code snippets

Bibliography

[1] Amazon AWS main website https://aws.amazon.com, 09.02.2017

[2] Google Cloud Platform main website https://cloud.google.com, 09.02.2017

[3] Microsoft Azure https://azure.microsoft.com, 09.02.2017

[4] Amazon tools for developing and managing AWS applications https://aws.amazon.com/tools/?nc1=f_dr, 09.02.2017

[5] Ruby on Rails framework website http://rubyonrails.org, 09.02.2017

[6] Ruby language website https://www.ruby-lang.org, 09.02.2017

[7] Rubber gem github webpage https://github.com/rubber/rubber/wiki, 09.02.2017

[8] Puppetlabs gem github page https://github.com/puppetlabs/puppetlabs-aws, 09.02.2017

[9] Elastic Beanstalk gem github page https://github.com/alienfast/elastic-beanstalk, 09.02.2017

[10] Software Development Process https://en.wikipedia.org/wiki/Software_development_process, 09.02.2017

[11] The National Institute of Standards and Technology Definition of Cloud Computing, http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf, 09.02.2017

[12] AWS Cloud Computing intruduction, https://aws.amazon.com/what-is-cloud-computing/, 09.02.2017

[13] LinkedIn blocked in Russia news, https://www.theguardian.com/world/2016/nov/17/russia-blocks-access-to-linkedin-over-foreign-held-data, 10.02.2017

[14] The difference between virtualization and cloud computing, https://blog.cloudpassage.com/2011/07/22/the-difference-between-virtualization-and-cloud-computing/, 10.02.2017

[15] AWS security resources, https://aws.amazon.com/security/security-resources/, 10.02.2017

[16] Information about cloud computing, https://en.wikipedia.org/wiki/Cloud_computing, 10.02.2017

[17] Information about cloud deployments models, http://www.slideshare.net/melsatar/cloud-deployments-models, 10.02.2017

[18] Information about cloud computing statistics, http://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2016-state-cloud-survey, 10.02.2017

[19] Information about cloud stack, https://web.archive.org/web/20151121220939/http://www.clouderpc.com/making-sense-of-the-cloud-saas-paas-and-iaas-explained#.WJ79vSTiTjA, 11.02.2017

[20] Amazon EC2 website, http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html, 12.02.2017

[21] Amazon Machine Image website, http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html, 12.02.2017

[22] Amazon OpsWorks website, https://aws.amazon.com/opsworks/, 12.02.2017

[23] Chef website, https://www.chef.io, 12.02.2017

[24] Puppet website, https://puppet.com, 12.02.2017

[25] Puppet Rails gem website, https://github.com/puppetlabs/puppetlabs-aws, 12.02.2017

[26] Rubber gem website, https://github.com/rubber/rubber/wiki, 12.02.2017

[27] Git website, https://git-scm.com,12.02.2017

[28] How to deploy Ruby on Rails application to AWS website, https://www.sitepoint.com/deploy-your-rails-app-to-aws/, 12.02.2017

[29] AWS Elastic Beanstalk documentation website, https://aws.amazon.com/documentation/elastic-beanstalk/, 12.02.2017

[30] Deploying a Ruby on Rails application on AWS Elastic Beanstalk website, http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_Ruby_rails.html, 12.02.2017

[31] Elastic Beanstalk Ruby gem website, https://github.com/alienfast/elastic-beanstalk, 12.02.2017

[32] Elastic Beanstalk deployer Ruby gem website, https://github.com/ThoughtWorksStudios/eb_deployer, 12.02.2017

[33] rsync website, https://rsync.samba.org, 12.02.2017

[34] AWS Cloud Data Migration website, https://aws.amazon.com/cloud-data-migration/, 12.02.2017

[35] AWS Database Migration Service website, https://aws.amazon.com/dms/?nc1=h_ls, 12.02.2017

[36] Amazon RDS docs website, http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html, 16.03.2017

[37] Ruby on Rails framework website, http://rubyonrails.org, 17.03.2017

[38] Ruby language website, https://www.ruby-lang.org/en/, 17.03.2017

[39] Basecamp website, https://basecamp.com, 17.03.2017

[40] Convention Over Configuration, https://en.wikipedia.org/wiki/Convention_over_configuration, 17.03.2017

[41] JSON documentation, https://www.w3schools.com/js/js_json_intro.asp, 17.03.2017

[42] Don't Repeat Yourself, https://en.wikipedia.org/wiki/Don%27t_repeat_yourself, 17.03.2017

[43] How to get the current date and time in Java language, http://www.mkyong.com/java/java-how-to-get-current-date-time-date-and-calender/, 17.03.2017

[44] AWS SDK for Ruby docs, http://docs.aws.amazon.com/sdkforruby/api/index.html, 17.03.2017

[45] Git website, https://git-scm.com, 17.03.2017

[46] Bootstrap website, http://getbootstrap.com, 17.03.2017

[47] Bootstrap-saas gem website, https://github.com/twbs/bootstrap-sass, 17.03.2017

[48] Haml-rails gem website, https://github.com/indirect/haml-rails, 17.03.2017

[49] HAML markup language website, http://haml.info, 17.03.2017

[50] Simple_form gem website, https://github.com/plataformatec/simple_form, 17.03.2017

[51] Bootstrap-generators gem website, https://github.com/decioferreira/bootstrap-generators, 17.03.2017

[52] Kaminari gem website, https://github.com/kaminari/kaminari, 17.03.2017

[53] Bootstrap-kaminari-views gem website, https://github.com/matenia/bootstrap-kaminari-views, 17.03.2017

[54] Growlyflash gem website, https://github.com/estum/growlyflash, 17.03.2017

[55] Devise gem website, https://github.com/plataformatec/devise, 17.03.2017

[56] Devise-bootstrap-views gem website, https://github.com/hisea/devise-bootstrap-views, 17.03.2017

[57] Postgres database website, https://www.postgresql.org, 18.03.2017

[58] Seed_dump gem website, https://github.com/rroblak/seed_dump, 18.03.2017

[59] AWS CodeCommit website, http://docs.aws.amazon.com/codecommit/latest/userguide/welcome.html, 18.03.2017