Marvin
Deployment system for CoreData

Fall 2014

Gadidjah Margrét Ögmundsdóttir
Marinó Fannar Pálsson
Sigurður Rúnar Helgason

B.Sc in Computer Science

Instructor: Hlynur Sigurþórsson
Examiner: Elín Elísabet Torfadóttir

T-404-LOKA
School of Computer Science

Abstract

In this report we discuss a B.Sc project entitled "Marvin: Deployment system for Coredata". This work was conducted at Reykjavík University in the fall of 2014 in collaboration with Azazo. In this work we introduce a new deployment system, which we call Marvin, that will eventually be the main deployment tool for Azazo's CoreData.

Contents

1 Introduction

Software deployment for large scale software applications with many interconnected components can be hard; it can even grow more difficult when managing different versions of a given software running for a variety of customers with different demands on updates.

Managing such environments and deployments without sufficient overview can be troublesome. Many questions might arise such as which clients need to be updated, have any clients been neglected and are specific updates failing for many different clients?

Azazo[1] is a knowledge and software company whose expertise include project management, quality assurance, management of information, meetings and records. Azazo utilizes its own creation, the cloud-based CoreData ECM software, to supply customers with solutions to the aforementioned management objectives.

The deployment process at Azazo is time consuming and complicated in the sense that many components have to be configured and properly set up during deployment, depending on the software version a customer is running at the time. Inadequately prompt overview of current version status and information between clients results in an extra preparation process preceding deployments.

Azazo has a large client base with different setups and update frequency requirements for each individual client. Managing deployments at Azazo is handled by two software systems, SAL and Revivhal, each managing its respective major release of CoreData. SAL handles deployments for the older release and will eventually become unnecessary once all customers have been migrated to the latest version.

SAL and Revivhal each provide an overview of hosts currently running CoreData and information about the hosts' respective clients and version information. Accessing this information is on a host-by-host basis and requires polling remote servers to pull necessary information each time. Both deployment systems lack sufficient overview of version status for all of their respective hosts. Due to the large number of CoreData hosts at any given point in time, it is not explicit which versions all the hosts are running. This can result in confusion which hosts need to be updated and if any have accidentally been neglected.

Manually keeping track of this information relies on using scripts to connect to the CoreData hosts to extract the necessary information. This poses a set of potential complications, such as whether the scripts currently have the correct list of hosts. Using these scripts is also fairly technical and requires all the users to have the know-how to be able to access this important information. This lack of immediate overview is the main reason for this project and the focus point of the project is to make the relevant information more readily available in one location so the status of versions and deployments is always at hand.

In this project we introduce a new deployment system that we have named Marvin, based on Revivhal and SAL, providing a better overview of the CoreData servers and deployments

[1]`http://www.azazo.com`

as a whole and enhancing everyday user experience. Marvin's host overview promptly presents its users with more detailed information about all of the hosts simultaneously, instead of on a host-by-host basis.

We aim to address the lack of overview in previous systems by storing statistical data from deployments concerning how long deployments take, whether they fail or succeed, how many and which clients are on a respective version and so on. This statistical data is collected over each host's lifetime and deployment history. To provide even further overview of the current status at any given point, the data is also graphically presented and can be queried for a specific time period or for other relevant parameters. These graphs are accessible by either the everyday developer interface or by a specific automated view that was created for independent overview displays.

It was a design criteria that the user interface of Marvin would be more user friendly than Marvin's predecessor, Revivhal, as well as being a user friendly system in general. To measure this we conducted a user evaluation in which five Azazo developers participated by performing predefined tasks preceding an interview. The result of this research were that all participants found Marvin more user friendly than Revivhal as well as a generally user friendly system.

This report is structured as follows: We begin in section two by introducing the collaborating company Azazo, their main software solution CoreData and the two deployment systems this project is based on. In section three we further discuss the motivation behind Marvin, its structure, architecture and functionality. In section four we describe a few typical use cases of Marvin as step-by-step scenarios. In section five we describe the research conducted to prove the hypothesis that Marvin is a user friendly system to its target group. In section six we discuss the progress of this project, our chosen development framework and describe the initial plan for the project's development. In section seven we discuss potential enhancements of Marvin that we would have liked to implement given more time and will hopefully be in Marvin's future. At last we discuss the conclusion of our experience of the project and its development.

2 Background

In this section we introduce Azazo and discuss the in-house deployment systems that they use in daily operation. The goal is to introduce the reader to the collaborating company of this project and briefly discuss their main product, CoreData. Also, we examine SAL and Revivhal in more depth as Marvin is based on these two applications.

2.1 Azazo

Azazo is a fairly young knowledge and software start-up founded in 2007 by entrepreneur Brynja Guðmundsdóttir. Originally the company was named Gagnavarslan but in preparation to market the company on an international ground, the name was changed to Azazo. In addition to software development, Azazo also runs a high-security storage facility on the old military base in Keflavik that houses documents, artifacts, artwork and etc. Azazo's expertise include project management, quality assurance, management of information, meetings and records. Azazo utilizes its own creation, the cloud-based CoreData ECM (Enterprise Content Management) software, to supply customers with solutions to the aforementioned management objectives.

CoreData ECM is a cloud based project- and information management system accessible to users over the internet. With CoreData, project managers can manage projects with simple and powerful user permissions, track project tasks and store documents related to projects. CoreData facilitates many great features for project- and document management such as electronic signatures with mobile devices, virtual file system disk and fine grained record management to name a few.

Cloud services often have complex setup processes where many inter-connected components have to be configured, as is the case with CoreData. The CoreData software implements multi-tenancy[2] architecture. In such structure, a single instance of a software runs on a server and is capable of serving multiple groups of users, called tenants. To manage this type of architecture, during the deployment process many components need to be connected and configured precisely in order for the software to serve its tenants correctly. Azazo owns two independent softwares that manage deployments today, named SAL and Revivhal. Each of the two manage their respective major releases of CoreData.

Azazo has a large client base with different setups and update frequency requirements for each individual client. This results in CoreData hosts running various versions at any given point in time. Keeping track of each host, its version, tenants and configurations at all times is an important task.

[2]http://people.apache.org/~hemapani/research/papers/ode-multi-tenancy.pdf

2.2 SAL

SAL, the predecessor of Revivhal, is a deployment system that used to manage all deployments for Azazo but now only manages a specific group of customers. The system provides an overview of the respective hosts it manages and their current status, displays the history of updates for each host and each update's resulting status. Users can also access the hosts through SAL and modify service configurations, view logs from previous and current updates and etc. SAL uses process workers to execute tasks through an SSH tunnel to the individual remote hosts.

At a certain point, Azazo remade one of its back-end services which resulted in a new major release of CoreData. However due to the substantial amount of data behind many of its customers, migrations from the old major release to the latest version of CoreData take a long time. Until those respective customers have been migrated, SAL continues to support and manage version upgrades for those specific customers.

2.3 Revivhal

Revivhal, the successor to SAL, is a web interface on top of a command line interface used to manage the post back-end remake release of CoreData. Revivhal currently handles all deployments for new customers as well as all customers that have been migrated from the previous major release of CoreData. Revivhal offers most of the same functionality as SAL with a few exceptions, including persistent logging of updates and deployment history for hosts.

3 Marvin

In this section we will further discuss the motivation behind Marvin. Why is it needed? We will also look into the architecture of Marvin and what design decisions were made when creating the system. Finally we look into the functionality of the end product and discuss the different viewpoints on the user interface.

3.1 Motivation

As previously stated; The deployment process at Azazo can be time consuming and complicated in the sense that many components have to be configured and properly set up during deployment, depending on the software version a customer is running at that time. On top of that, Azazo has a large customer base with different update requirements. Insufficient overview of update status for the CoreData hosts can introduce unnecessary confusion and complication in the update process. The current deployment software solutions at Azazo lack this overview, making the update process time-inefficient and open to oversight in version control.

Marvin is a new deployment system designed for use by the developers at Azazo that handle deployments to CoreData servers. This project is based on Azazo's current deployment systems, while adding a more detailed overview of the status of CoreData servers at any given time. Marvin provides this enhanced overview by displaying more detailed information of hosts as a whole instead of singularly on a host-by-host basis. These detailed information can also be searched for any relevant parameters, further enhancing the ability to quickly determine the current status of all CoreData hosts. As well as providing this instant feedback of host status as a whole, much of the relevant information can also be viewed graphically. Graphically displaying the data makes the current status even clearer to its users.

3.2 Structure

Coding criteria from Azazo were that the project back-end would be written in Python[3] and the front-end in Python, HTML5[4] as well as AngularJS[5].

Django[6] is a Python web framework that is the foundation of the current deployment system Revivhal, which this project is based on. The Django layer from Revivhal was used as a base for this project with some modifications and additions. For example, the Django

[3] https://www.python.org/
[4] http://www.w3schools.com/html/html5_intro.asp
[5] https://angularjs.org/
[6] https://www.djangoproject.com/

database now houses more detailed information and for that, the Django models needed to be modified and more models added.

SQLite3[7] is the SQL database engine of choice. Marvin's predecessor, Revivhal, made use of SQLite3 to store very limited amount of data. For Marvin, a lot more information is stored and the original idea was to work with SQLite3 until the project was nearing its final stages and then transition over to PostgreSQL[8]. The reason for that is because PostgreSQL is simply faster, more reliant and robust. However due to lack of time towards the end, SQLite3 will stay as the database choice for this project for now.

AngularJS was used in Revivhal as well. In Marvin the AngularJS layer was written from scratch, Revivhals existing functionality was used as a guideline and if possible, existing code was reused. By rewriting the AngularJS layer new libraries and AngularJS functionalities could be introduced in Marvin, such as AngularUI Router[9]. The rewriting had its consequences which will be further discussed in the conclusion chapter of this paper.

D3[10] is a JavaScript library which was used to build charts in Marvin. D3 was integrated in the AngularJS layer.

Bootstrap[11] was used to aid in making of the User interface.

3.3 Architecture

Azazo's primal software application, CoreData, is a cloud-based service where multi-tenancy architecture is implemented on each individual server. A cloud-based service is fundamentally a location independent group of remote servers accessible via the Internet. In a multi-tenancy architecture, a single instance of a software runs on a server, capable of serving multiple groups of users called tenants. Users of each tenant share the same view on the software and each tenant is served its own data, configuration, user management, individual functionality, business rules and etc. In Azazo's case, a tenant represents a single customer (company) and it's users (staff) all have access to the same instance of that customer's CoreData application. Azazo's servers then run a tenant catalogue service to manage the tenants on that specific server.

Marvin has an Angular web application that contains HTML templates, Javascript functions and services to display and control functionality in the user interface. The application feeds requests from the user down to a Django web server and returns the responses from Django back to the user interface.

The Django application is a Python web server that manages the functionality provided by Marvin. The server listens for requests through an Application Programming Inter-

[7] `http://www.sqlite.org/`
[8] `http://www.postgresql.org/`
[9] `https://github.com/angular-ui/ui-router`
[10] `http://d3js.org/`
[11] `http://getbootstrap.com/`

face(API), handles user authorization, user sessions and manages tasks to be performed on remote CoreData servers. A task is simply a sequence of actions to be performed on a specified remote server. Tasks can be as simple as updating the remote server's version control repository or as complicated as running large scripts that take several minutes. The Django server uses a message queue service called RabbitMQ to handle multiple concurrent task requests. RabbitMQ stores the task requests in a First-In-First-Out(FIFO) queue until the tasks have been performed. To carry out the tasks, the server also runs a task worker service called Celery. When tasks in the RabbitMQ queue are to be performed, the next available Celery worker process handles that task from start to finish. The tasks to be performed on remote servers are defined in a separate layer below the Django application known as X.

X is a Python module that handles all Secure Shell(SSH) connections from Marvin to the remote CoreData servers. The main purpose of X is to define all the relevant tasks that need to be executed on the remote servers to manage deployments of the CoreData software. The tasks that X runs on the remote servers are in-house shell scripts designed by the CoreData developers.

Following is a graphical illustration of the architecture described above, where each component's purpose and disposition is further put into perspective.

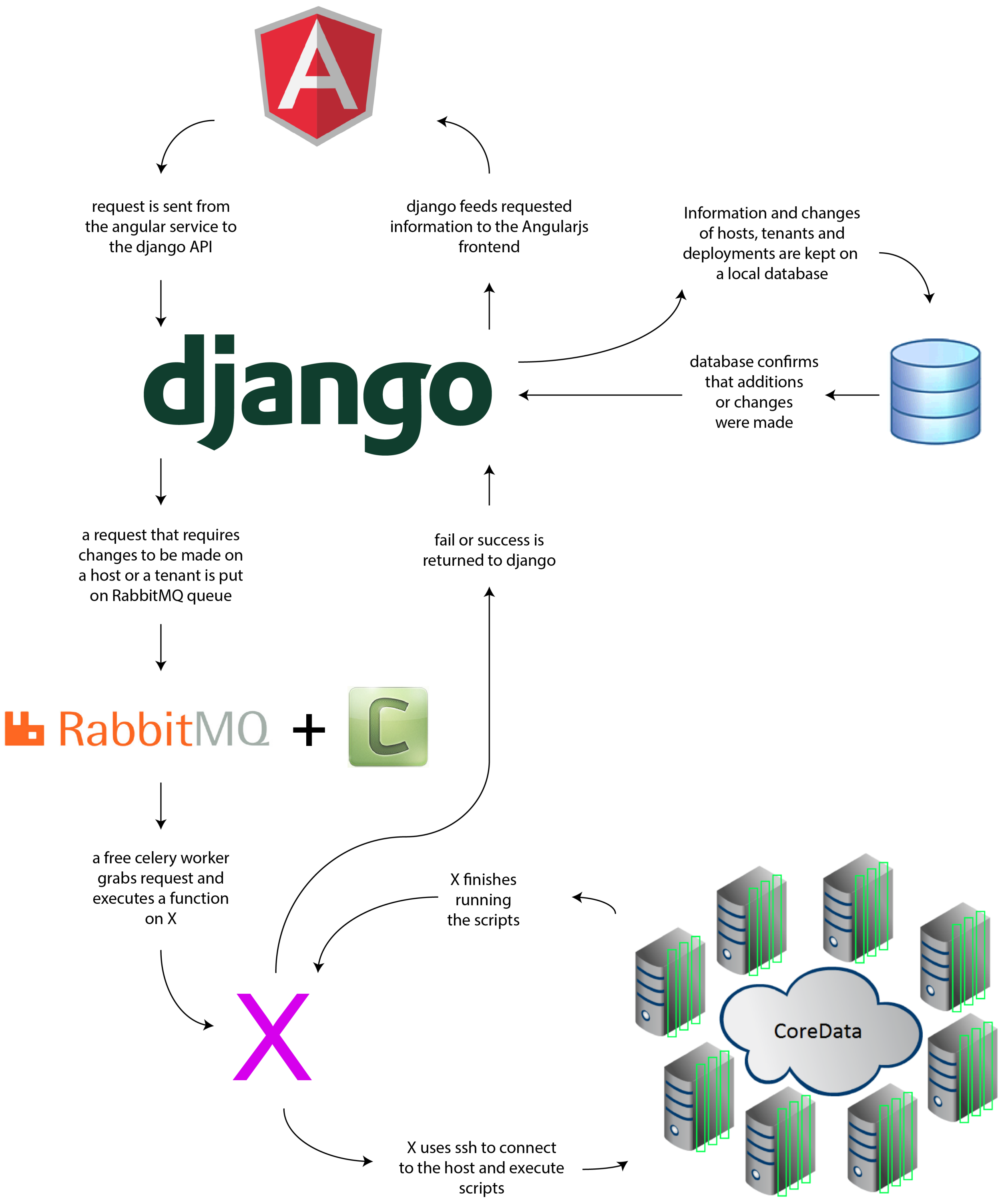

request is sent from the angular service to the django API
django feeds requested information to the Angularjs frontend
Information and changes of hosts, tenants and deployments are kept on a local database
database confirms that additions or changes were made
a request that requires changes to be made on a host or a tenant is put on RabbitMQ queue
fail or success is returned to django
RabbitMQ +
a free celery worker grabs request and executes a function on X
X finishes running the scripts
X
CoreData
X uses ssh to connect to the host and execute scripts

Figure 1: Architecture overview of Marvin

3.4 Functionality

The Marvin user interface is split up into different views, each of which has its own respective focus point and different functionality. There are four different facets to view the system. Below is a list of the four views in no particular order.

First is a host view which focuses on the CoreData hosts both independently and as a whole, providing functionality specific to the respective hosts. The view displays a list of CoreData hosts with basic information about each host, including the version of CoreData running on that host and the current status of that host. Some more detailed information can be seen in a more detailed view for each host, such as the tenants currently on that host, when it was last updated and etc. There are several different actions that can be performed on each host, depending on the respective host's current status. Notable actions are to update the host, view the host's update history and so on.

The tenant view is similar to the host view in the sense that it displays a list of tenants with detailed information regarding each tenant, such as the name of its host, on which domain it is located, whether it is active or inactive and etc. As with the host view, further details regarding a tenant are displayed by clicking on a tenant in the list. As with the host view, there are specific actions for tenants that are accessible in this view. Notable actions are editing a tenant, provisioning, activating and deactivating a tenant.

As stated before; Marvin stores statistical data from deployments which is graphically presented. The last two views are different ways to view these graphs. One of the views is a typical day-to-day view accessible to any standard user of Marvin. In that view, the user has some control over the data that's being presented. The view has a list of graphs and the user clicks on a graph to view it. The user can choose from the graphs: version history, tenants/hosts per version, average update duration of a version, number of updates, update failure and tenant/host turnover. The aforementioned control over the graphs is mainly that the user can view a graph from a tenant or a host perspective, by major or minor versions of CoreData and/or by a specific time period chosen by the user. Finally the last view is a view that cycles the same graphs as mentioned above. This is designed to run independently and automatically on an overview display to provide Azazo with the latest year-to-date information at any time.

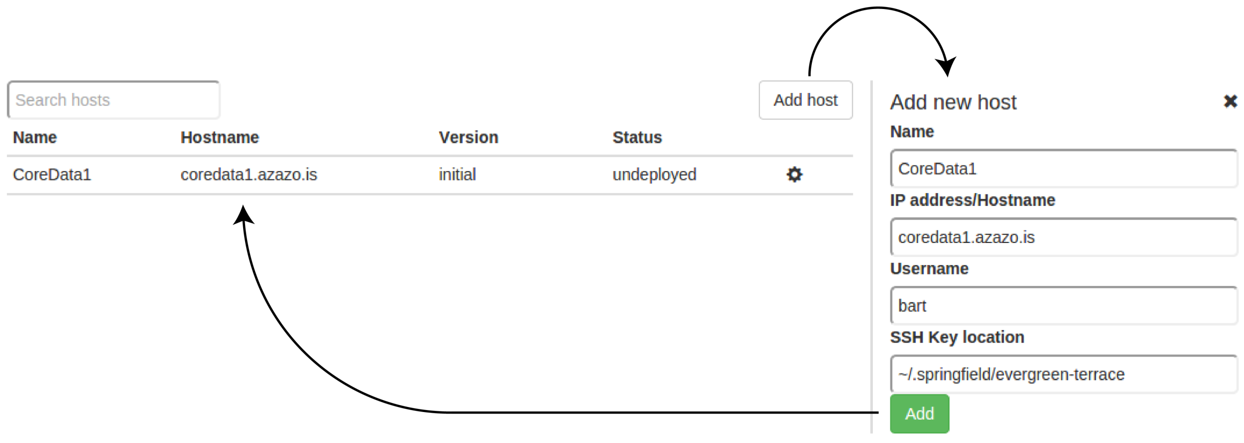
4 Scenarios

In the following section we will go through four possible scenarios in Marvin. How to add a host, how to deploy a host, how to add a tenant and how to provision a tenant.

4.1 Adding a host

Before a host can be added to Marvin, a server needs to be set up and the user adding the host needs to know information like ip-address/hostname, username for ssh and where the ssh-key is stored on the machine.

To add a host, click the “Add host” button above the list of CoreData hosts. A sideview appears requesting information about the new server. After filling out the information, press “Add” at the bottom of the sideview and the host should appear in the list of hosts to the left.



Name	Hostname	Version	Status
CoreData1	coredata1.azazo.is	initial	undeployed

Name	CoreData1
IP address/Hostname	coredata1.azazo.is
Username	bart
SSH Key location	~/.springfield/evergreen-terrace

Figure 2: Adding a host

The host information is now stored in the local Marvin database and needs to be deployed to be a functional host with the CoreData software.

4.2 Deploying a host

To deploy a host, click the settings icon(see figure 3) for the host to get a list of all available actions. Click “Deploy”. A sideview appears requesting information about the server deployment. Fill out all the information required. If a specified version/tag is not available in the dropdown list, click the refresh button to the right of the dropdown menu to update the host’s repositories.

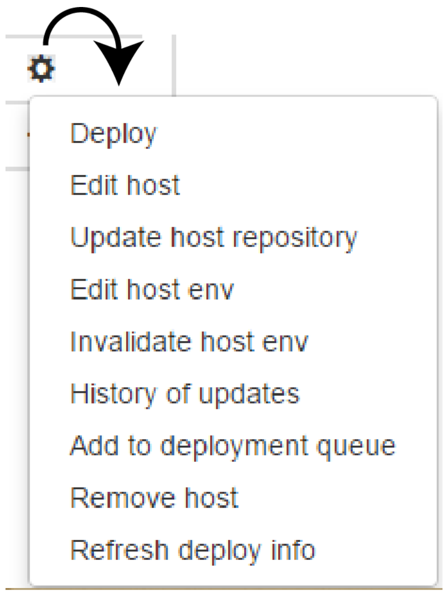
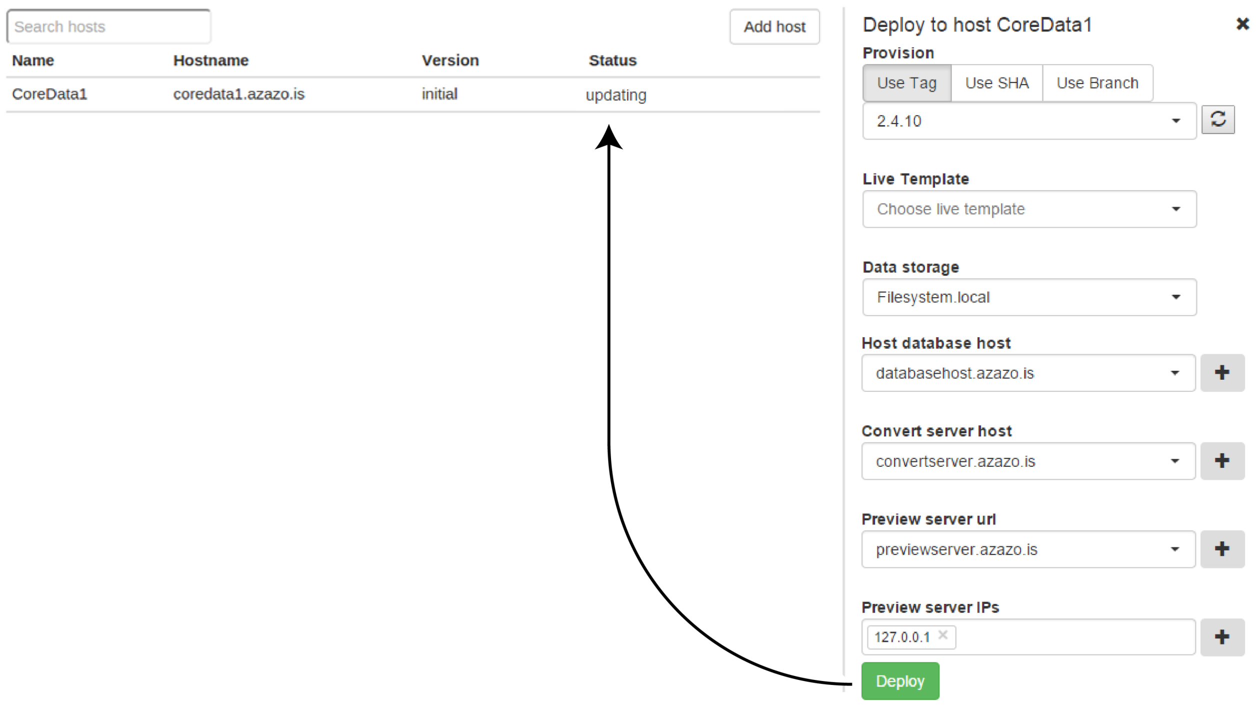


Figure 3: Host settings

After filling out the information, click "Deploy" on the bottom of the sideview. The host's status in the hostlist should change to "updating" and will go to "online" when the deployment is finished.



Name	Hostname	Version	Status
CoreData1	coredata1.azazo.is	initial	updating

Deploy to host CoreData1

Provision: Use Tag / Use SHA / Use Branch — 2.4.10

Live Template: Choose live template

Data storage: Filesystem.local

Host database host: databasehost.azazo.is

Convert server host: convertserver.azazo.is

Preview server url: previewserver.azazo.is

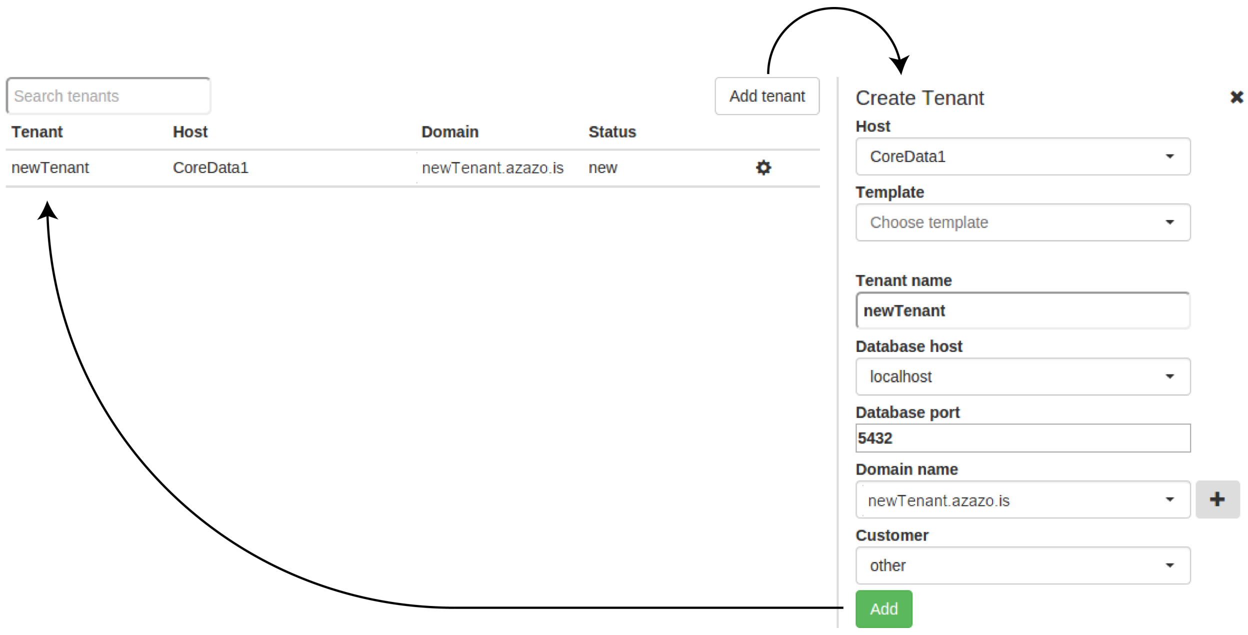
Preview server IPs: 127.0.0.1

Figure 4: Deploying a host

4.3 Adding a tenant

Since tenants are added to hosts, a host with the status online must be available before adding a tenant.

Go to Marvin's tenant view. Click on "Add tenant" above the list of tenants. A sideview appears requesting information about the new tenant. After filling out the information, press "Add" at the bottom of the sideview and the tenant should appear in the list of tenants to the left.



Tenant	Host	Domain	Status
newTenant	CoreData1	newTenant.azazo.is	new

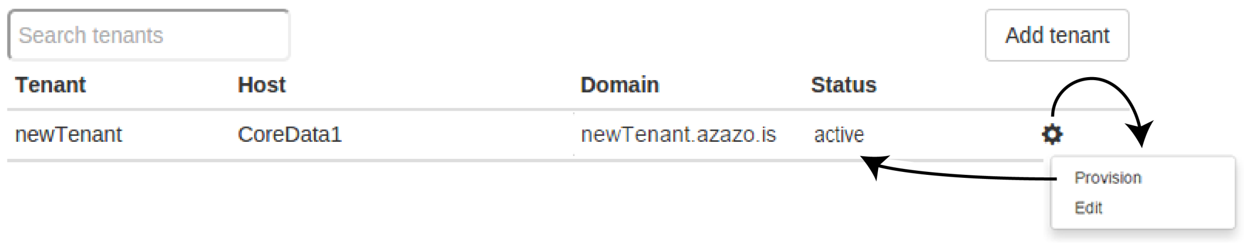
Host: CoreData1
Template: Choose template
Tenant name: newTenant
Database host: localhost
Database port: 5432
Domain name: newTenant.azazo.is
Customer: other

Figure 5: Adding a tenant

At this time, the host has information about the new tenant but it needs to be provisioned (set up) before it is usable.

4.4 Provisioning a tenant

In Marvin's tenant view, click the settings icon(see figure 6) for the tenant to get a list of all available actions. Click "provision". When the provisioning is finished, the tenants status changes from "new" to "active".



Tenant	Host	Domain	Status
newTenant	CoreData1	newTenant.azazo.is	active

Figure 6: Provisioning a tenant

5 User evaluation

User evaluation was conducted to prove the hypothesis that Marvin provides sufficient overview of deployments and version status at Azazo as well as being a user friendly system to its target group of Azazo developers.

Our goal was to make Marvin more user friendly than its predecessor Revivhal. In order to draw a conclusion we decided to have users with experience of Revivhal evaluate Marvin.

5.1 Method

As Marvin is a complicated system that requires inside knowledge of Azazo's software solutions, our potential user group for this research only includes developers at Azazo that are familiar with Revivhal. We asked five users to take part in our evaluation. All participants had used Revivhal before but aren't necessarily using it today. One participant uses Revivhal at least three times a day, while another hadn't used it in many months.

Participants were asked to perform a series of tasks to determine the usability and intuitiveness of Marvin as well as to introduce new functionality and overview of deployments and hosts/tenants. Following is a list of tasks the participants were asked to perform.

- Find all hosts on version 2.4.2
- Find the host that has a tenant named "tenant40"
- Find a tenant that has "azazoOnline4" as its host
- Add tenant on host "lokaverkefni" with a name of your choice
- Schedule an update on host "lokaverkefni" 2 mins. in advance to the newest tag
- Find scheduled update in updatelist
- Discover history of version 2.3
- Find how many hosts are on version 2.4.2
- Find out the average update time of the last 2 minor versions
- Find how many updates occured in the year 2014
- Find which hosts failed an update to version 2.4.1
- Look at new hosts grouped by months in a time period of your choice
- Look at the log of the performed update on host "lokaverkefni"
- Find out what time the update finished
- Join the Marvin chat room on Hipchat

- Look at update history of host “lokaverkefni”

The research method was performed in the following manner; The participant was asked a couple of background questions regarding their experience with Revivhal, if they had used Revivhal before or taken part in developing Revivhal at any point. The latter of the two inquiries was to be able to take into account any bias from the participant against Marvin, being a developer of the preceding system. Following that, Marvin’s user interface was briefly presented to the participant after which he was asked to perform the aforementioned tasks. During this process, everything concerning the participants’ experience when performing the tasks was duly noted. At last the participant was interviewed on his user experience of Marvin.

5.2 Results

In the interview following the usability tasks, the participant was asked to rate how user friendly Marvin is, on a scale of 1 to 10; 1 being not user friendly at all and 10 being perfectly user friendly. The resulting grades from our participants were as follows: 8.2, 9, 8, 9, 8.5 with an average of 8.5.

The participants were then asked to rate how user friendly Marvin is compared to its predecessor, Revihal. The scale was again from 1 to 10; however in this case 5 signified exactly as user friendly as Revivhal, lower than 5 meant it’s less user friendly and greater than 5 meant Marvin is more user friendly than its predecessor. The resulting grades from our participants were as follows: 9, 9, 9, 10, 10 with an average of 9.4.

Participants unanimously agreed that Marvin provides excellent overview of deployments with its graphical view as well as good overview of hosts and tenants with the more detailed listings of their respective views.

5.3 Discussion

From the results of Marvin scoring an 8.5 we can assume that Marvin is user friendly according to its target group, Azazo developers. From the result of Marvin scoring 9.2 when compared to Revivhal we assume that we have succeeded in making Marvin a more user friendly system than its predecessor.

In the interview, participants admitted that they found it hard to give Marvin the score 10, the reason being that there is always something that can be done better. Participants only had positive comments about Marvin, even exceeding the expectations of some by leaps and bounds. Participants found new functionality introduced in Marvin beneficial for everyday use in Azazo’s deployment process and when asked which features they found to be the highlights, the most common answers concerned the ability to search for any connected variable to a host and/or tenant, the multiple host deployment queue for batch updates, the ability to view logs of previous updates and the overview provided by the graphs; in particular the graph that shows the lifeline of all versions.

6 Project Progress

In the beginning, the group members planned how much minimum work would be put into Marvin for every week of the project. The estimated time plan was split up into two parts to account for the workload in other courses during the first ten out of thirteen weeks of this project. For the first ten weeks there would be meet ups three times a week yielding a total of 130 hours per team member. For the last three weeks, other courses would be finished and so the allotted time would be increased to 10 hours per weekday, for a total of 150 hours per member. For the group as a whole, a minimum of 840 hours of work were planned for those thirteen weeks, excluding 70 hours at the start of the semester spent on scoping the project.

The team decided to use the Scrum method to organize and plan the work with daily stand up, a retrospective meeting and a planning meeting for each sprint. The first six sprints were scheduled as two weeks each and the final sprint would span the last week of the project. Sprints six and seven would have higher capacity because of higher team member availability the last three weeks as other courses would be finished.

Table 1: Capacity planned and logged hours

	Estimate	**Logged hours**
Sprint 0	70	74
Sprint 1	78	86
Sprint 2	78	48
Sprint 3	78	92
Sprint 4	78	153
Sprint 5	78	133
Sprint 6	300	346
Sprint 7	150	159
Total	**910**	**1091**

The group created the product backlog and stories in cooperation with the product owner from Azazo. The backlog consists of 50 stories which we estimated would total 207 story points (SP), 43 stories of which were priority A (172 SP) and 7 stories that were priority B (35 SP).

A risk analysis was conducted to be prepared for severe problems arising that could potentially stop our work. We realized some potential risks that could have a big impact on our progress, so all of those points were handled as soon as possible. For that reason, whenever an issue came up related to these risks, we had already taken precautions and were not affected.

In the beginning, a couple of trivial stories were divided into tasks to get an initial velocity. The sprint velocity varied quite a bit throughout the project (7 to 24.7 SPs). There are multiple reasons for the variations, but the group consensus is the two biggest causes being inexperience in estimating story points and working seperately instead of together. Additionally in sprint two, the group didn't expect the high workload in other courses rendering half the sprint fruitless.

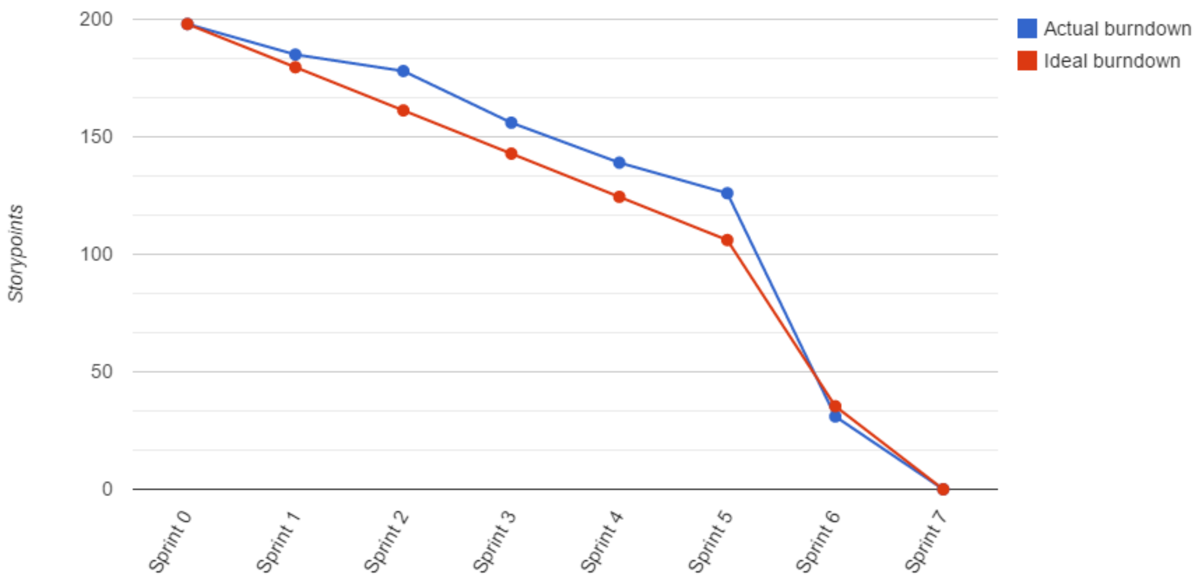


Figure 7: Release burndown chart. Difference in the estimated burndown line being because of groups higher capacity in sprint 6 and 7.

The group managed to finish 45 of the initial 50 stories. The 5 remaining stories were cancelled in consultation with the product owner due to various reasons. The group members didn't deviate much from the original plan of meeting three times per week for the first weeks, and then joining forces every day once other courses were finished. Every team member put more work into the project than originally estimated (181 hours total, see table 1 and figure 8. Main reason for that is underestimation of how much work had to be put into programming the whole AngularJS front end from scratch. A new version of AngularJS changed alot of functionality causing us to be unable to reuse a big portion of the code we expected to reuse.

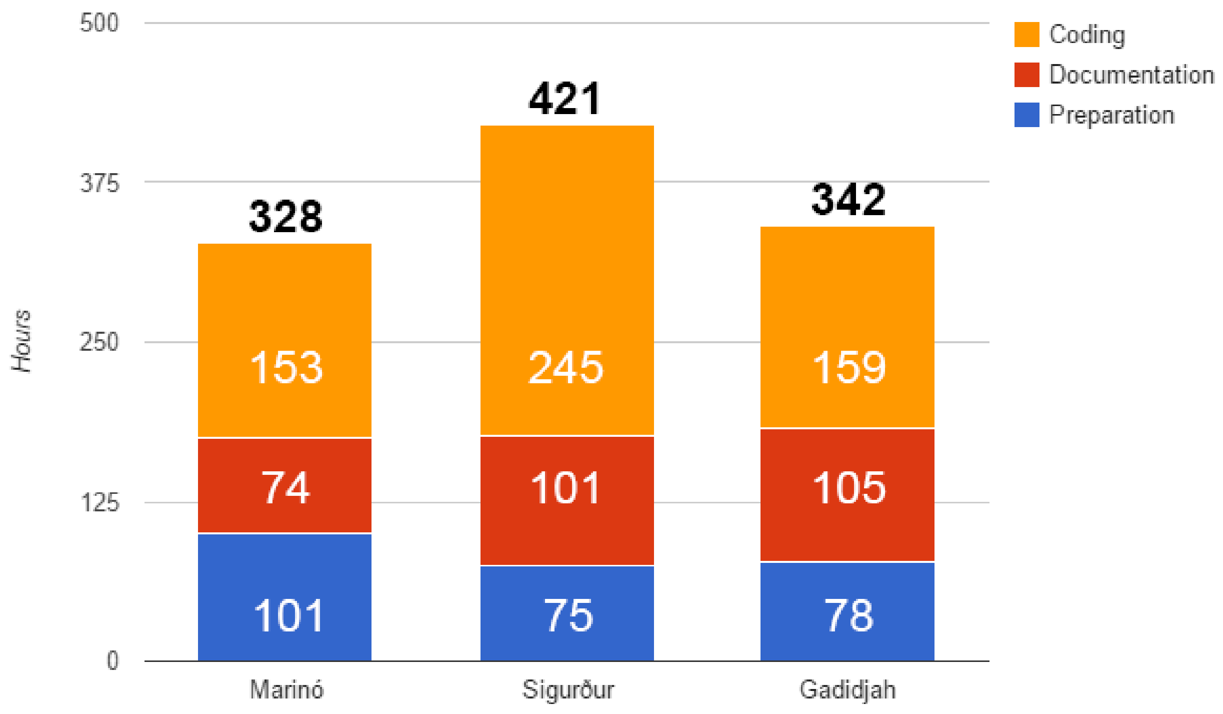


Figure 8: Shows how much work each group member put into Marvin and how it was split between categories

7 Future works

This project entails a fully functioning deployment system ready for use by Azazo. During development, many enhancement ideas came up and were discussed. A select few were implemented as lower priority requirements and others were left as topics of discussion for this chapter due to time constraints. The following are all features we see as valuable and would've liked to have implemented, given more time to work on the project.

In the development of Marvin, some steps were taken to separate hosts and tenants into individual units with their own respectful viewpoints. We feel it's important to be able to view a host and its tenants as individual entities and not necessarily a singular connected component. The next steps could then be to move the tenant catalogue of each CoreData host over to one common tenant catalogue that's run by Marvin. This would further separate hosts and tenants and as a result, in the event of a host update, the communal tenant catalogue could then swap the host's tenants over to an already updated CoreData host to reduce downtime. Moving further in this direction it would be very useful to be able to view a host's health status in the host specific view. Some criteria could be constructed to determine if a host is of good health; regarding memory usage, available disk space and etc.

To make the graphical presentation of deployment data even more useful, it could be interesting to make the graphs dynamic. For instance, if a user wants to see a specific trend that is not currently available, that user could change the axis parameters to show the desired trend. If we had more time we would have liked to be able to extract the statistical data from the graphs into printable report documents.

It was unanimously agreed by all users of our user evaluations that persistent logging for deployments is an important feature of the system. That said, we would have loved to be able to also view real-time logs of deployments while they are in progress.

8 Conclusion

In this report we have introduced our deployment system Marvin. We gave insight into the collaborating company Azazo and touched on CoreData's architecture and its deployment process. We then looked closer into the motivation behind Marvin, the development criteria required by Azazo, the system's architecture and it's typical use cases. We finally discussed the progress throughout the project and potential enhancements for Marvin that might be interesting for developers to implement in the future.

The development of Marvin was interesting to say the least. The idea behind the project was initially to add functionality in Revivhal to provide overview of deployments with a graphical presentation as well as providing better overview of CoreData hosts. The way this would be achieved was undecided and it was left to us as a part of this project to further define its scope. The initial scope was wide open to begin with which meant we had free reign, so to speak, over the project's direction. However this also required good planning and many brainstorming sessions, with and without the product owner, to construct the product backlog.

Two big decisions played a major role in slowing our progress significantly in the beginning stages of development. One of which was to rewrite Revivhal's AngularJS front-end and thus creating a complete new front-end web application. This can be attributed to our inexperience in working together and a blatant underestimation on our part of how complicated Revivhal is in reality, even if we thought it's user interface seemed like there wasn't a whole lot to cover. The second factor was inexperience with the latest and greatest in AngularJS functionalities, such as UI-Router and UI-Select. This resulted in a bottleneck forming time and time again, causing all of us to focus on the same task until it was completed.

Another thing we didn't account for was the heavy workload from other courses at school resulting in more time spent apart than together to work on the project. Realizing that this had a bad effect on the project's development, an effort was made to meet up more frequently.

Learning from our mistakes, we actively focused on working together but at the same time dividing our attention between the layers of development. This, along with everyone being able to work in their personally preferred layer, resulted in a much better throughput and overall progress of the project. In the end it's safe to say that we are immensely proud of Marvin and excited to hand it over to Azazo.

Our main lessons learned are to respect previously written code. In the future we will be sure to research a system more before taking the decision to rewrite it from scratch. This is our first time maintaining such a big project and it has taught us the value of methods like Scrum to maintain organization.

We would like to thank Azazo for collaborating with us on this project. The facilities at Bæjarhraun were greatly appreciated and the almost-constant availability of their employees, who were always ready to help us out and advice or discuss any questions we had, was amazing. We also want to thank Reykjavík University for the opportunity to develop such a sizable and great project as Marvin.

Finally we would like to extend our sincerest gratitude to our project owner Skúli Arnlaugsson, our instructor Hlynur Sigurþórsson and examiner Elín Elísabet Torfadóttir, for their endless support and constructive criticism and advice throughout the project.

9 Review from Azazo

This fall we've had the opportunity to work with a group of talented young people on Azazo's in-house deployment tool, Revivhal, or Marvin. The team has been enthusiastic and willing to learn a great deal of business requirements and logic to be able to contribute valuable work to making our tools better. It has been a pleasure and the work is very promising.

Skúli Arnlaugsson, Product owner
Test developer, Azazo

Reykjavík, December 15, 2014

Gadidjah Margrét Ögmundsdóttir
250494-3439

Marinó Fannar Pálsson
200881-4519

Sigurður Rúnar Helgason
041185-3179