DEVELOPMENT OF THE ANCOR DASHBOARD

by

BRIAN CAIN

B.S., Kansas State University, Manhattan KS, 2012

A REPORT

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2014

Approved by:

Major Professor
Dr. Xinming (Simon) Ou

Copyright

Brian Cain

2014

Abstract

In this paper I present ANCOR Dashboard, a front end web framework that interfaces with ANCOR. ANCOR Dashboard aims to provide ANCORs users with an easy to use front end interface for accomplishing various use-cases against the ANCOR framework. ANCOR Dashboard was developed mainly in AngularJS, a lightweight JavaScript framework. This dashboard is able to accomplish everything that the ANCOR Command Line Interface, or ANCOR-CLI, is able to do. This framework also needed to provide some information about the state of the system through various data gathered from ANCOR in a human readable format. Not only should it be able to inform the user about the state of ANCOR but it needs to be able to perform operations against ANCOR just like the command line interface can do.

This report documents the design and implementation of ANCOR Dashboard. It will detail the necessary background of the project, and overview of the framework, and discussion of implementation and component breakdown. I will also provide an evaluation of the dashboard and a discussion about future work with ANCOR Dashboard.

Table of Contents

List of Figures

Acknowledgments

I would like to express my appreciation to Dr. Simon Ou, Dr. Scott DeLoach, and Dr. Eugene Vasserman for being on my masters committee. I would also like to thank Ian Unruh and Alexandru Bardas for their help and collaboration on the ANCOR project. Finally, I would like to thank the Argus Lab group for giving me valuable feedback on my presentation of ANCOR Dashboard.

The ANCOR project is supported by the Air Force Office of Scientific Research awards FA9550-12-1-0106 and FA2386-13-1-3021, and the U.S. National Science Foundation awards 1038366 and 1018703.

Dedication

I would like to dedicate this report to all of my family and friends who have been with me along my graduate school journey.

Chapter 1

Background

In this chapter, I hope to provide some necessary background information to explain the problem I am solving with ANCOR Dashboard. I will also provide some basic information about the ANCOR framework to interface with. This chapter will also describe some of the potential users of ANCOR Dashboard.

1.1 Problem Description

Projects in software engineering always provide a way to interact with their application. Traditionally these tools have been in the form of command line interfaces (CLI) or graphical user interfaces (GUI) that run as an application on the users computer or a remote server. CLIs provide a simple and quick way to perform various commands with the provided software, whereas GUIs that run on the users desktop will provide a more visual point-and-click approach to interact with the software.

However within the past few years there has been a trend where developers are instead providing the user with a web interface, or dashboard[1], for them to interact with the program. This dashboard interface provides a cross-platform experience where developers no longer have to worry about using a GUI framework that will be supported across all major

operating systems. Now, developers can create a simple to use frontend framework through the web where the only major differences they have to worry about is between the major browsers. Not only that, but with projects like Bootstrap, cross browser support has never been easier to accomplish as a web application developer. With the creation of these frontend dashboards, users can now interface with programs through their favorite browser without having to install anything extra on their computer.

ANCOR Dashboard hopes to continue this trend by providing a front end web dashboard framework that interfaces with ANCOR. Through modern web technologies like AngularJS. Bootstrap 3, and D3JS, ANCOR Dashboard gives users a simple solution to interact with ANCOR.

1.2 ANCOR Framework

To understand the need for ANCOR Dashboard, it will be helpful to explain what ANCOR[2] is and what problem it attempts to solve. This project was developed by the Argus Lab group[3] separately as a component of the Moving-Target Defense project.

ANCOR stands for *Automated eNterprise network COmpileR*. ANCOR is a cloud automation framework that abstracts the various elements of an enterprise network as defined by a user. This project helps solve the problem when users are interested in using a cloud-based IT system but might not be completely familiar with the lower level details that would be required to set up and deploy. While there are other solutions currently such as IaaS, SaaS or PaaS, these services are not as flexible and depend on what the vendor is interested in providing to their customers. With ANCOR, users can define a higher level abstraction to construct and manage their cloud-based IT system.

These abstractions are what defines what is called a requirement model. Once this requirement model has been defined, ANCOR takes those requirements and compiles an enterprise network. Through this technique, ANCOR is able to model dependencies between

the different layers in a given application stack. ANCOR uses technologies like OpenStack, Puppet, and MCollective to accomplish this.

When a given enterprise network configuration called ANCOR Requirement Model Language (or ARML)[2] is given to ANCOR, it will generate all of the required services and define them as instances. In ANCOR Dashboard's case, I will refer to this as a configuration file or configuration specification file. These instances are defined in the specification as roles. A role generally can be defined as something like a webapp, a worker instance, a database master or slave, and so on. These roles will be defined by the user with various attributes for ANCOR to deal with. Overall, these roles are all related by a goal. The specification has a goal attribute which groups these roles together. An example goal might be an eCommerce website with example roles being a few load balancers for web, some web applications, database master and slaves, and worker nodes.

1.3 Users of ANCOR Dashboard

This project was developed with a couple users in mind. Both of these users will be ones who are wanting to interact with ANCOR, but each user may have different preferences when it comes to interacting with ANCOR.

Our first example of a user is users who may not be familiar with using the console or operating programs through console commands. In this case, they are not interested in using the ANCOR-CLI to interface with ANCOR. Because of this, they will be more interested in the ANCOR Dashboard. ANCOR Dashboard offers an alternate solution for these users. This solution is very visual, and gives them a point-and-click experience. The only real typing they might have to do is when they want to deploy a new environment. Even in this case, this configuration might already be defined and they can just paste the configuration file into the ANCOR configuration editor.

Another example of a potential ANCOR Dashboard user is someone who want a more

visual experience when interacting with ANCOR, despite having the skills required to be comfortable in command line. This situation may become more important as scenarios for ANCOR become more complicated. The ANCOR-CLI may end up returning more information than the average person can decypher at once. Using ANCOR Dashboard will give users the ability to analyze a large amount of information through graphs and statistics that may be difficult to represent in terminal.

Chapter 2

ANCOR Dashboard Overview

In this chapter, I will give a brief overview of ANCOR Dashboard. I will go into some of the enabling technologies and how they work. I will go over the basic planned architecture of the project, the use-cases that were defined while developing the project, and the interaction diagrams created for the project.

2.1 Description

ANCOR Dashboard is a front end web dashboard developed to interact with the ANCOR[2] project. It offers its users a visual experience as opposed to interacting with the ANCOR-CLI through a terminal. It gives the user various statistics about the state of ANCOR, graphs detailing what the environment looks like and how instances are related, and allows users to perform different operations on instances, the entire deployment environment, as well as configuring ANCOR through a configuration file for deploying new environments.

2.2 Enabling Technologies

The web framework has been built on top of AngularJS, but is also comprised of several other technologies. These technologies along with the basic ANCOR Dashboard architecture are detailed in this section. To make this project possible, I take advantage of some workflow applications that help construct and build ANCOR Dashboard.

2.2.1 Main Workflow Components

Yeoman[4] is a scaffolding tool that offers several different official and third party web application generators. Scaffolding is a technique popularized by Ruby on Rails that allows developers to quickly generate components of an application with a few commands. Normally without scaffolding, developers would have to manually create routes, views, controllers, and models by hand. Since this operation is common, web application frameworks like Ruby on Rails and Yeoman have automated this process.

Yeoman provides scaffolding generators to help developers get a jumpstart on their project. These generators offer some basic functionality to generate things like routes, controllers, views, and other various components depending on which generator is used. This project takes advantage of the generator for AngularJS projects.

One example of why Yeoman is important is the ability to quickly add a new route. A route is a new endpoint that a user can visit within a web application, like *myproject.com/mynewroute*. To add a new route, simply type the command found in figure 2.1 the root of the project directory. When we break down this command it has a couple different components that tell Yeoman what to do. First, *yo*, is the command used to invoke the Yeoman program. Next *angular:route* tells Yeoman that you wish to use the AngularJS route generator. The final part of the command is the name you wish to give the route. Since Yeoman is a scaffolding tool, it will also generate the appropriate view and controller that goes along with the new route. The controller and view will be blank however, and it

is left up to the developer to add in all of the styling for the views, and functions and model variables for within the controller script.

```
1    yo angular:route mynewroute
```

Figure 2.1: *Yeoman AngularJS route generation command.*

Bower[5] is a dependency management tool that serves as a package manager for scripts used in the project. This tool is used to install and inject various JavaScript scripts that ANCOR Dashboard uses like angular-ui, the ACE Editor, and several other important dependencies. Using Bower makes it simple to add a new dependency, remove a dependency, or update a dependency related to the project. It also provides an easy way for developers who are wishing to contribute to the project to grab and install all of the required dependences at once when setting up the project.

The final piece for the web application workflow is a build tool called Grunt[6]. Grunt is a Javascript taskrunner that helps automate the process of deployment, running tests, and previewing a project an easier process. Grunt is what builds ANCOR Dashboard along with building the necessary assets required to run the project.

2.2.2 AngularJS

The main technology used for the web framework on this project is called AngularJS[7]. It's a Javascript MVC (Model View Controller) framework developed by Google that is fairly light weight. There will be more on how the components of AngularJS relate to ANCOR Dashboard under the implementation section of this paper. For now, lets talk about some of the basic components that make up the AngularJS MVC framework.

For every AngularJS app, there will be a module name. You can think of the module of the app as a namespace for the project. This is what helps tell all of the components of the

project what pieces will fit together under the project namespace. For example, ANCOR Dashboards namespace is *ancorDashboardApp*.

AngularJS has a feature known as *data binding*. This means that between the model and the view of a given webapp, variables between the two components will always be at the same state. For example, if a page has a form textbox with a model variable hooked up to the input, the model variable will be updated at the exact same time as a user is typing. This is much different from other web applications where it is required to get the object ID of the textbox and obtain the value when a user hits a submit button.

Data binding also offers some useful features when it comes to filtering out data within the view. This feature is mostly taken advantage of when a developer wants to filter data from a search box and a list of results or if a developer wishes to filter out how data is formated.

In each controller for AngularJS, there will be a $scope variable. The $scope variable can be seen as the glue that relates the controller, model and the view together. $scope can contain not only all of the models used, but it can also be functions that are defined within the controller that are accessible from the view as well.

Views in AngularJS rely on what are called templates. Using templates makes developing dynamically generated content in a web application much easier. Through view templating, a developer can dynamically populate a table without having to enter in every piece of data manually. These templates do not get compiled until either a developer deploys the web application or runs it on a local server. An example of a component in a template HTML view can be seen in figure 2.3.

2.2.3 Bootstrap

For styling and various web UI components, ANCOR Dashboard uses Bootstrap 3[8]. Bootstrap provides a lot of HTML, JavaScript, and CSS based templates to help developers create decent looking web pages. Originally developed by two developers from Twitter, it

is now a stand alone project independent from Twitter.

Over the past few years, Bootstrap has become one of the largest and most popular web technologies for software developers. Currently at the moment it is the most popular of any projects on Github, with over 66,000 stars/favorites and almost 25,000 forks/clones[9].

Because of how simple and powerful Bootstrap is for web developers, many websites are now switching over to the highly customizable framework. Since Bootstrap provides such an easy way to make a website look nice, many companies no longer have to roll their own CSS or JavaScript from scratch. Some of the companies that use Bootstrap include Newsweek, Github, GruntJS, Twitter, NASA, MSNBC's Breaking News, and more[8].

2.2.4 Other Components

ANCOR Dashboard uses the ACE Editor[10] for submitting and writing configuration files to send to ANCOR. ACE is the same editor used at Github when users want to edit files in a project repository directly through the website. ACE is a highly customizable JavaScript editor that offers several different features like theme selection, Vi shortcuts, a full undo and redo stack, line wrapping, code folding, syntax highlighting, and more. The ACE Editor is a perfect choice for editing configuration files because the syntax matches YAML syntax. This means users can edit configuration files with syntax highlighting directly within ANCOR Dashboard.

D3.js[11] is an advanced JavaScript library for manipulating HTML, SVG, and CSS. Through D3.js, a developer is able to dynamically generate interactive graphs or diagrams. Developed by a software engineer from the New York Times, D3.js was mainly used as a way to display data to the user with an interactive experience.

2.3 Architecture

ANCOR Dashboard follows the Model View Controller design pattern. This is a typical architecture for most web applications. For communicating with ANCOR, it uses REST over HTTP to send and receive data.

2.3.1 Model View Controller

ANCOR Dashboard follows the traditional Model View Controller (MVC) approach for web applications. MVC is a typical architecture for web applications. Each part of the dashboard is broken up into 3 distinct components that map to a model, view, and controller as seen in figure 2.2.

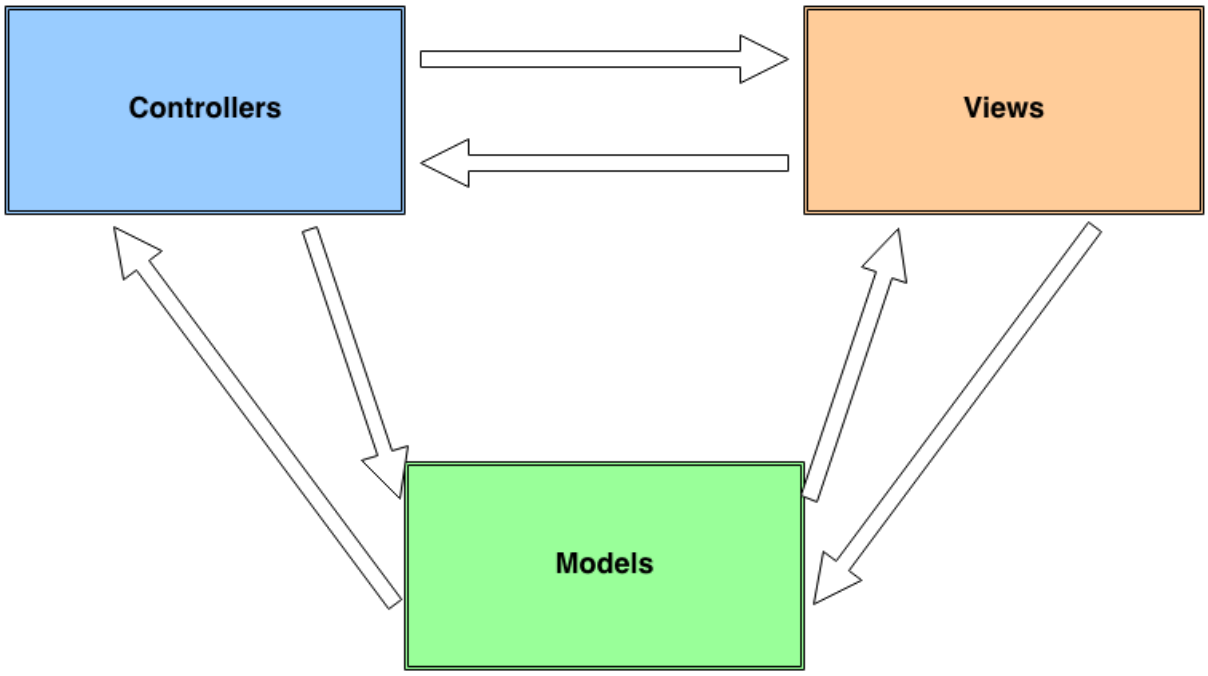

Figure 2.2: *A generic Model-View-Controller architecture.*

Controllers in AngularJS are stored as *controllerName.js*. A controller in MVC is something that stores business logic that controls the web application. In AngularJS, any thing within the controller that is not a function will execute the moment that specific route is

loaded. This includes any sort of models or JavaScript logic not wrapped in a function. Otherwise, the code within a function will not happen until that specific method is invoked. For example, all of the REST HTTP calls will happen right away when the controller is loaded.

Models in AngularJS are defined through the $scope variable and are related to the variables that reside within the controller. The models are what contains data that is shown and also modified by the user. In the case of ANCOR Dashboard, these model data variables are usually data that ANCOR Dashboard queries from ANCOR. An example of some of these data variables include instances, roles, and the ANCOR version. AngularJS stores this data to make it easily accessible to the view.

Views in AngularJS are still in HTML, however developers are able to modify the DOM tags to contain Angular code that makes working with HTML a bit easier. The code segment in 2.3 shows off an example of how AngularJS makes writing views easier. It is a code segment from ANCOR Dashboard where it builds a simple dropdown that contains all of the roles from ANCOR. It gets the roles variable from the MainCtrl model and uses an AngularJS style for each to loop over all of the elements on roles to create a list of roles for the dropdown.

```
1  <ul class="dropdown-menu" role="menu" >
2    <li ng-repeat="role in roles">
3      <a ng-click="addNewRole(role)">
4        {{role}}
5      </a>
6    </li>
7  </ul>
```

Figure 2.3: *AngularJS generating a dropdown menu.*

2.3.2 Interacting with ANCOR Through REST

When ANCOR Dashboard needs to directly interact with ANCOR, it communicates through a REST API, or Representational state transfer[12] API. Developed in 2000 by Roy Thomas Fielding, REST is a popular solution developers use when projects need to communicate to each other over HTTP.

By communicating over HTTP with REST, ANCOR is able to expose a few functions that help outside users control the project. To start this process, you make an HTTP operation call to a specified URL. These operations are known as *GET, POST, PUT,* and *DELETE*. Since ANCOR was designed to communicate to outside projects through REST, ANCOR Dashboard uses this API to send and receive data directly to ANCOR. The following is a description of each component of the REST API basic functionalities with examples on how ANCOR Dashboard uses them.

GET is a RESTful operation that when invoked, will return information depending on what endpoint has been queried. For example, if ANCOR Dashboard wants to get a detailed list of all the instances currently on ANCOR, it can query the *http://ancorIP/v1/instances* endpoint and get a json-formatted chunk of data that details all of the instances currently deployed.

POST is a RESTful operation that allows information to be *posted* to the remote server. This operation is often used when a website has form data that needs to be submitted to the remote server. In ANCOR Dashboard, *POST* is used when a user is ready to submit a configuration file to deploy a new environment. In this case, we are not posted json formatted data to ANCOR. In the header we put that the data is yaml format so ANCOR knows what format to expect.

PUT is a RESTful operation that implies that you are *putting* or modifying a resource. This means you will replace whatever was there before with the new call. In ANCOR Dashboard's case, this operation is used when we want to tell ANCOR to commit the newly *POSTED* environment.

DELETE is a RESTful operation that will *delete* a given resource depending on what endpoint is queried in ANCOR. For example when a user wants to delete a given instance or an entire environment, they are able to make a *DELETE* call to remove that given instance or environment.

2.4 Use-Cases

In this section, I will describe the use-cases that this dashboard covers. These use-cases mirror the functionality of ANCOR-CLI, but give a more visual interactive experience for the user instead of the terminal based experience ANCOR-CLI gives.

2.4.1 Viewing Instances

The main use-case that this dashboard was developed for was seeing the state of instances from ANCOR. This is also the main view of ANCOR Dashboard when you first visit the project in your browser. It displays how many instances ANCOR currently has, as well as what current state each instance is in. If you scroll down, you can see the D3.js generated network graph. This graph displays the layout of ANCOR and shows how each instance is related. This can be seen on figure A.1. A closer look at the dynamically generated network graph can be seen in figure A.2.

If a user wishes to see all of the instances of ANCOR in more detail, they can scroll down to the table of instances near the bottom of the main page as seen in figure A.4. In this case, the table shows the user the instance name, the interface it's on (an ip address), the current stage, and the planned stage. Clicking the more info button will display all of the relevant information that ANCOR provides about each instance (figure A.5). At the moment, there is no ANCOR endpoint to query for any *advanced* details, however there is a space left in case that feature is added in the future. The final column contains various operations that a user can perform on the instance as well.

If a user wants to search for something specific within the table, they are able to use the filter search box. As seen in figure A.3, when typing something in this box, the table will shrink down to match the search parameters.

2.4.2 Viewing Tasks

Visiting Tasks (figure A.6) will give you the state of all current tasks. They are sorted by tasks that were updated most recently and give a user an idea of what's currently happening on ANCOR. For example, when a user deploys a new configuration, they are able to visit the tasks page to watch what tasks are currently in progress, what tasks have completed, or what tasks are suspended. They are also able to filter tasks if they are searching for something specific. The filter will work for any column in Tasks.

Like in the Main view, Tasks also has a filter search box to allow users to look through all Tasks. Typing anything in the search box will start the process of filtering through all of the Tasks data as seen in figure A.7.

2.4.3 Deploying New Roles/Instances

All roles can be discovered by selecting the dropdown link above the Instance overview table. Each element in the dropdown for adding a new role is the given role slug provided by ANCOR. This dropdown can be seen in figure A.8.

If a user wants to deploy a new role to ANCOR, they can select one of the role slugs from the dropdown menu. ANCOR Dashboard will then make an HTTP POST call to ANCOR with the selected role slug to deploy a new specified instance.

2.4.4 Importing Configuration Specification Files

ANCOR Dashboard provides an interactive method for deploying and importing configuration specifications. Visiting the Deploy endpoint for ANCOR Dashboard will load up the

ACE embedded cloud editor (figure A.9). This editor is fully featured and supports yaml syntax highlighting.

For the users convenience, ANCOR Dashboard provides two templates that can be inserted into the arml specification while they develop their configuration file (figure 3.5). The first one is the basic goal template that a user will have to make when starting a new configuration. The other template provided is a role template, which a user can insert after they have created a goal within the configuration specification. These templates provide some basic structure that a user can modify while developing their configuration specification.

If a user is confused on the structure of an configuration specification, they can select the help button on top of the ACE Editor (figure A.10), which will bring in a modal view explaining the structure as well as showing an example that they can use.

2.4.5 Replacing and Deleting Instances

For replacing an instance, ANCOR Dashboard provides a button on the main Instance table view that a user can select to perform the replace action (figure A.11). When the user selects replace, the view will invoke the replaceInstance method in the main controller with the given instance id. ANCOR Dashboard then does an HTTP POST call to ANCOR against the id passed in from the view. At the moment, ANCOR does not support replacing an instance. However the code is in place once this feature has been completed, and the only change required would be uncommenting out some code within the main controller.

Similar to replacement, ANCOR Dashboard provides a way for a user to delete instances as well. Selecting the delete button from the instance table list will invoke a function within the Main controller to do an HTTP DELETE call to ANCOR with the provided id.

2.4.6 Viewing Environments

If a user would like to see if the current environment is locked, or if they are interested in the name of the current deployments environment, they can visit the Environments page

of the dashboard (figure A.12). This view will also let the user know if the environment is currently locked. When an environment is locked, that means ANCOR is currently working on a job and no other operations can occur.

2.4.7 Deleting Environments

When an ANCOR user wants to completely remove an environment, they can visit the Environments page of the dashboard. This view provides a button that will completely remove the current deployment. After invoking this operation, the Tasks page will show the progress of removing the specific environment. The delete button will become disabled if the environment is locked.

2.5 Interaction Diagrams

Interaction diagrams give a general overview of how a user might interact with the software the diagrams were based off of. In this section, I introduce ANCOR Dashboards interaction diagrams based off of the use cases referenced in 2.4.

In figure 2.4, we have the interaction diagram for the Main, Env, and Task endpoints. The first event that happens is when a user visits one of the three endpoints on the dashboard. This will kick off a few HTTP GET calls which will interact with ANCOR's REST API. These HTTP GET calls are different depending on which endpoint we are looking at (this will be explained in the next chapter). Once the controller receives a response from ANCOR, if successful, it will take the json data it received and store it in a model variable to be used for later. This model variable will also be displayed on the view for the user to see and interact with. If there are any updates made by the user through the view, those updates will also be made in the model because of AngularJS data binding. The views also have various functions that can interact with the controller. For example in the Main view, there are functions for creating and deleting instances. These functions are invoked from

the controller, which then make the appropriate HTTP POST or HTTP DELETE calls to ANCOR.

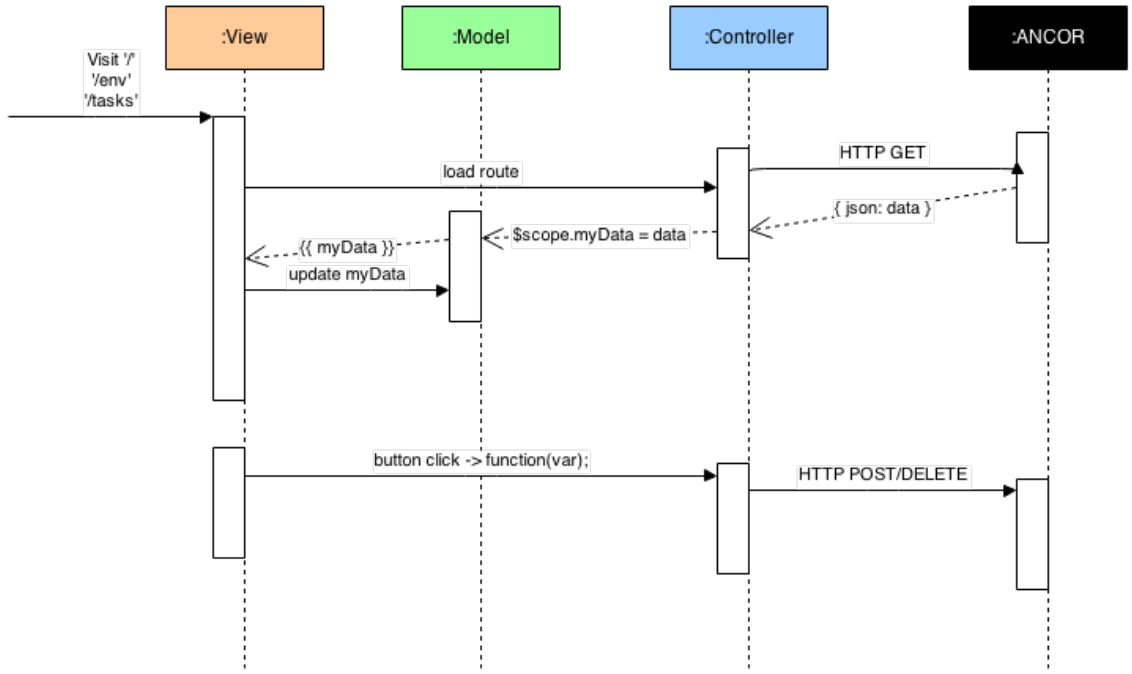

Figure 2.4: *Interaction diagram for Main, Environment, and Tasks endpoints.*

In figure 2.5, we have the interaction diagram for the Deploy endpoint. When this endpoint is first loaded up, it goes out and fetches environments from ANCOR. This is so the user can be informed as to how many environments are currently deployed in ANCOR. Currently, this is important because the dashboard only supports a single deployed environment. Next, the ACE editor is loaded up for the user to interact with. The user is able to have a full editor where they can write their configuration configuration file. As the user writes out their configuration, the View makes sure to update the model as to what is within ACE. Once the user clicks the deploy button, ANCOR Dashboard will take all of the text within the ACE Editor to send off to ANCOR. It will first to an HTTP POST that contains the configuration data in yaml format. Once that call has been successful, it will do an HTTP PUT to let ANCOR know to commit the changes and start the deployment.

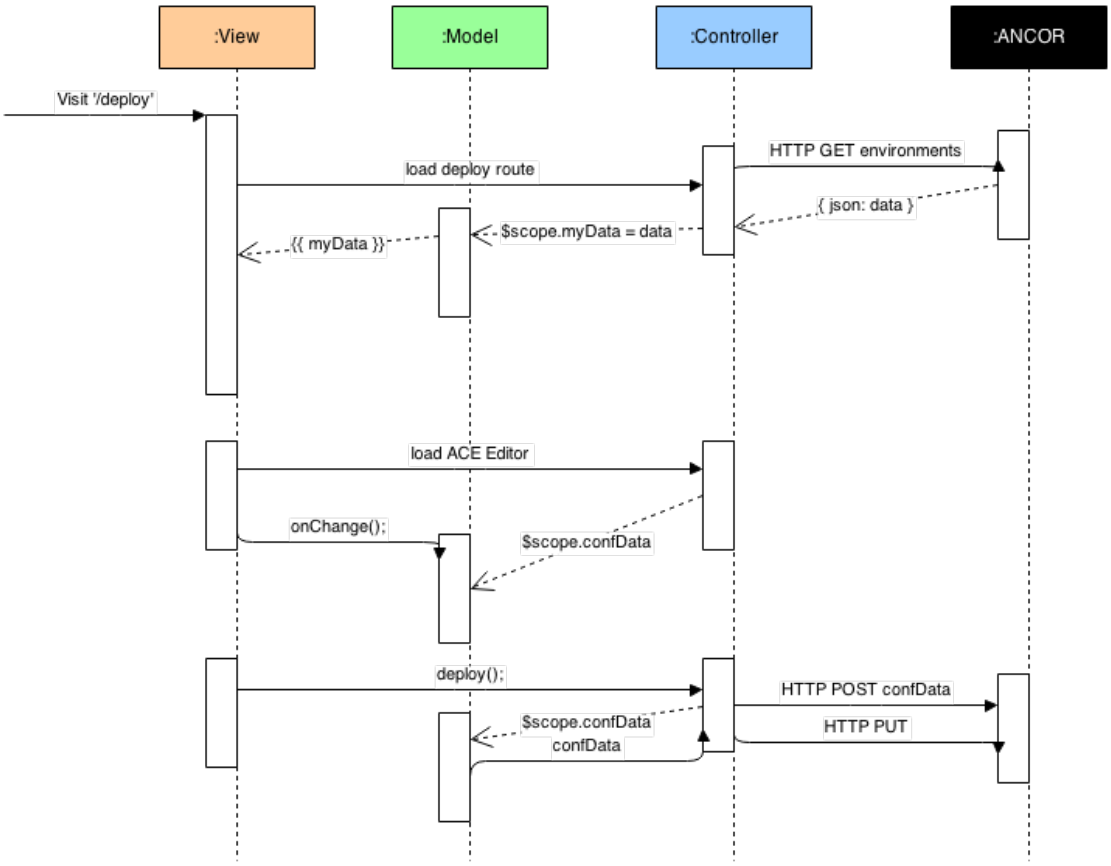

Figure 2.5: *Interaction diagram for Deploy endpoint.*

Chapter 3

ANCOR Dashboard Implementation

In this chapter, I will go over the main implementation details for the project. This will include the final architecture design, the original development process, a discussion on how some of the code has been implemented, and then a component breakdown of the entire project.

3.1 Final Architecture and Design

For the final design of ANCOR Dashboard I choose the traditional Model-View-Controller architecture for a web application.

Following figure 3.1, we can see that we have ANCOR as a black box that the ANCOR Dashboard communicates with. When ANCOR Dashboard needs to retrieve or send data to ANCOR, it will send an HTTP GET, PUT, POST, or DELETE call to invoke a REST action. This is represented in the figure by the two arrows between the ANCOR black box and the controller.

Within the dashed rectangle with rounded edges, I have represented ANCOR Dashboard. It follows the Model-View-Controller architecture that most web applications conform to. The controller is where all of the methods and HTTP REST functions are located. It is

also the component in charge of defining the models. After a successful HTTP call from ANCOR, the controller will save the relevant data into the model. On the view, if any of the model variables are needed, they can be accessed and displayed to the user. Also, if there are any functions that need to be called from the controller, the view contains buttons to invoke these functions.

Finally on the very right of figure 3.1, we have the user browser. This is where the user interacts with ANCOR Dashboard. Normally the user will retrieve the compiled template view that is ready to be displayed in the browser. However if the user wishes to add a new instance, delete an instance, deploy a new configuration file, etc, then they will invoke that action through the view that will then call a method within the controller.

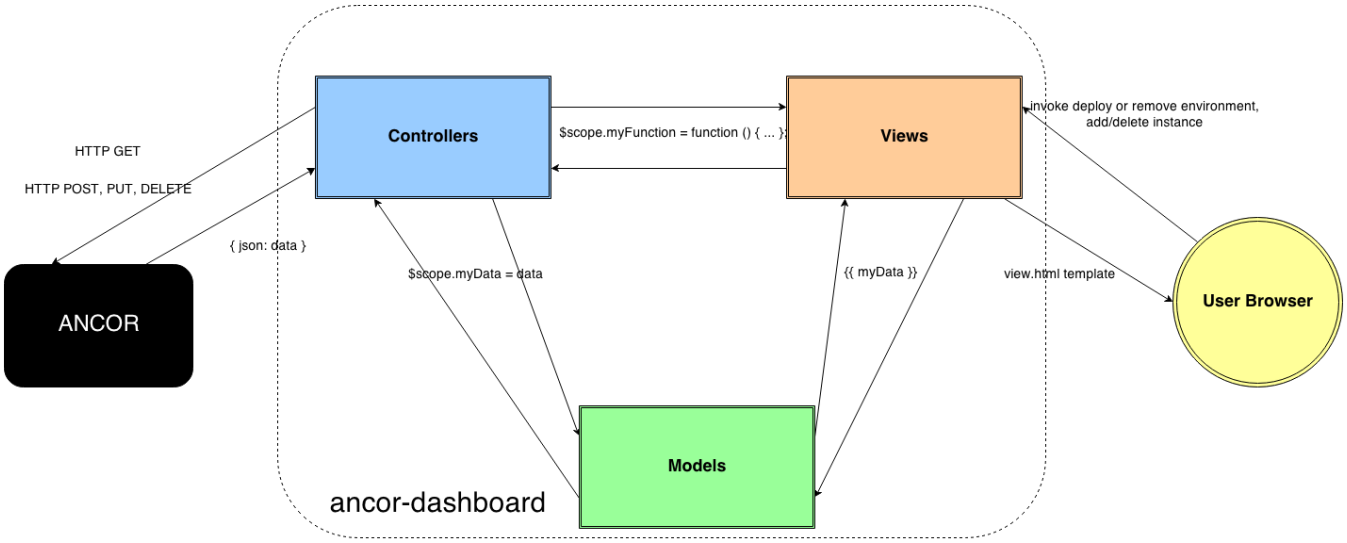

Figure 3.1: *Full caption to appear below the Figure*

3.2 Prototyping

Initially, this project was developed with Ruby on Rails. After a few months of development, it became clear that Ruby on Rails was a heavy framework to use for a project like this. Not only that, but because of the way Ruby on Rails works, it added an extra step to interact

with ANCOR. With Ruby on Rails, if a user wanted to make an API call to ANCOR (like deploying a configuration file or viewing tasks), the user would have the browser talk to the Ruby on Rails server, and then that server would talk to ANCOR. This added an extra step in the interaction between ANCOR Dashboard and ANCOR.

To resolve this issue, I redid the framework in AngularJS. With AngularJS, the software is compiled to minified CSS and JavaScript and ran on the users machine directly. This compiled code will often be placed in an Apache or Nginx server to be served out to the user. In this case, there is no longer that extra medium required to talk to ANCOR. Now the user can directly communicate with ANCOR through their browser without needing to query a Ruby on Rails server first.

3.3 ANCOR Dashboard Code Discussion

In this section, I will break down some of the major functionalities of ANCOR Dashboard and talk about how the code accomplishes those functionalities.

3.3.1 REST over HTTP with AngularJS

Earlier in chapter 2, I mentioned the idea of how ANCOR Dashboard communicates with ANCOR with a concept called REST over HTTP. As mentioned before, REST has several different protocols to interface with: *GET, PUT, POST, and DELETE*. But how exactly does that work within this project? Let's take a look at the Main controller to get a better idea of how this interaction works.

```
1  $http.get($rootScope.ancorIPAddress+'/v1').success(function(data) {
2    $scope.version = data.version;
3  });
```

Figure 3.2: *ANCOR Dashboard using REST to ask ANCOR what version it is.*

```
1  <title>ANCOR {{version}} Dashboard</title>
```

Figure 3.3: *Displaying a model variable within a view.*

In figure 3.2, we have a simple HTTP GET call to ANCOR that is retrieving the *version* to display to the user on the dashboard. It queries the version number through a URL, and then ANCOR responds with a json formatted data set. In this case, ANCOR responds with the version number. In the controller we set that version to a model variable to be displayed on the view as seen in figure 3.3. The double brackets in AngularJS represent an evaluation. Because the variable version exists within the $scope model, the view is able to display that value within HTML to the user. If no data is returned from ANCOR in the controller, the *version* model variable will be empty and the evaluation will be left blank when the view is compiled.

```
1  $scope.addNewRole = function (roleSlug) {
2    var url = $rootScope.ancorIPAddress+'/v1/instances',
3        newRole = { 'role': roleSlug };
4    $window.alert('New role ' + roleSlug + ' added!');
5    $http.post(url, newRole);
6    $route.reload();
7  };
```

Figure 3.4: *Code from the Main controller to add a new role to ANCOR with an HTTP POST call.*

In figure 3.4, we have an example of an HTTP POST call to ANCOR. In this case, this function is invoked when a user is interested in adding a new role through the role dropdown generated on the main page. First it builds the correct URL with a json data key value structure. It alerts the user that the invocation has started, and then sends the json data to the specified URL. ANCOR will then take over from there, and the page is reloaded.

3.3.2 D3 Network Graph

On the main view of the dashboard, we have a dynamically generated network graph that uses D3js. This can be seen in figure A.2. The initial graph drawing code was modified from one of the numerous D3js examples on the main website. However the code has been modified to work within the project itself. For example, to generate the nodes on the network, it must take a specially formed set of data from the Main controller. The main controller first does an HTTP GET call to retrieve all of the instances. Within this json on each instance, there is a *depends_on* attribute that is used to get what each node depends on and interacts with. A json object is then created with the source name, each sources

dependency (or target in this case), the type of dependency (this will determine the color and style of the line with css), and the source ID. This data set is then passed into the method located in *forced-graph.js* that generates the entire network graph.

The network graph code also defines how large each node circle will be, the length between connections, the color and size of the text used, and so on. Everything about the network graph is customizable here. Another modification to the network graph script was preventing the graph from jumping around when first loaded. Many D3js examples have very active elements when first loaded, almost like bouncy balls being released into a gymnasium. There is a simple method near the bottom that skips this activity so the graph looks more static on first load. This does not however prevent the user from dragging nodes around and moving the network graph on the page.

It has an event method for when a user selects one of the nodes. At the moment, this method only prints the node name and id to the JavaScript console. However in the future this method will be used for more than just that. See the future work section for more information.

3.3.3 ACE Editor

The ACE Editor is an important part of the configuration writing process with ANCOR Dashboard. Thanks to the angular-ui library, integrating the ACE Editor is extremely easy. In figure 3.6, we can see that adding a new editor is as easy as making a simple div tag with a few options defined. We give this div a specific id so that the controller knows which div to apply to.

Next with angular-ui, we pass in different configuration options to customize ACE. For example, the configuration files that ANCOR uses are of the yaml syntax, thus we tell ACE to use yaml syntax highlighting for anything within the editor. useWrapMode is a standard text editing feature that prevents users from having to scroll side to side while typing up a document. Finally the two functions onLoad and onChange are functions defined by angular-

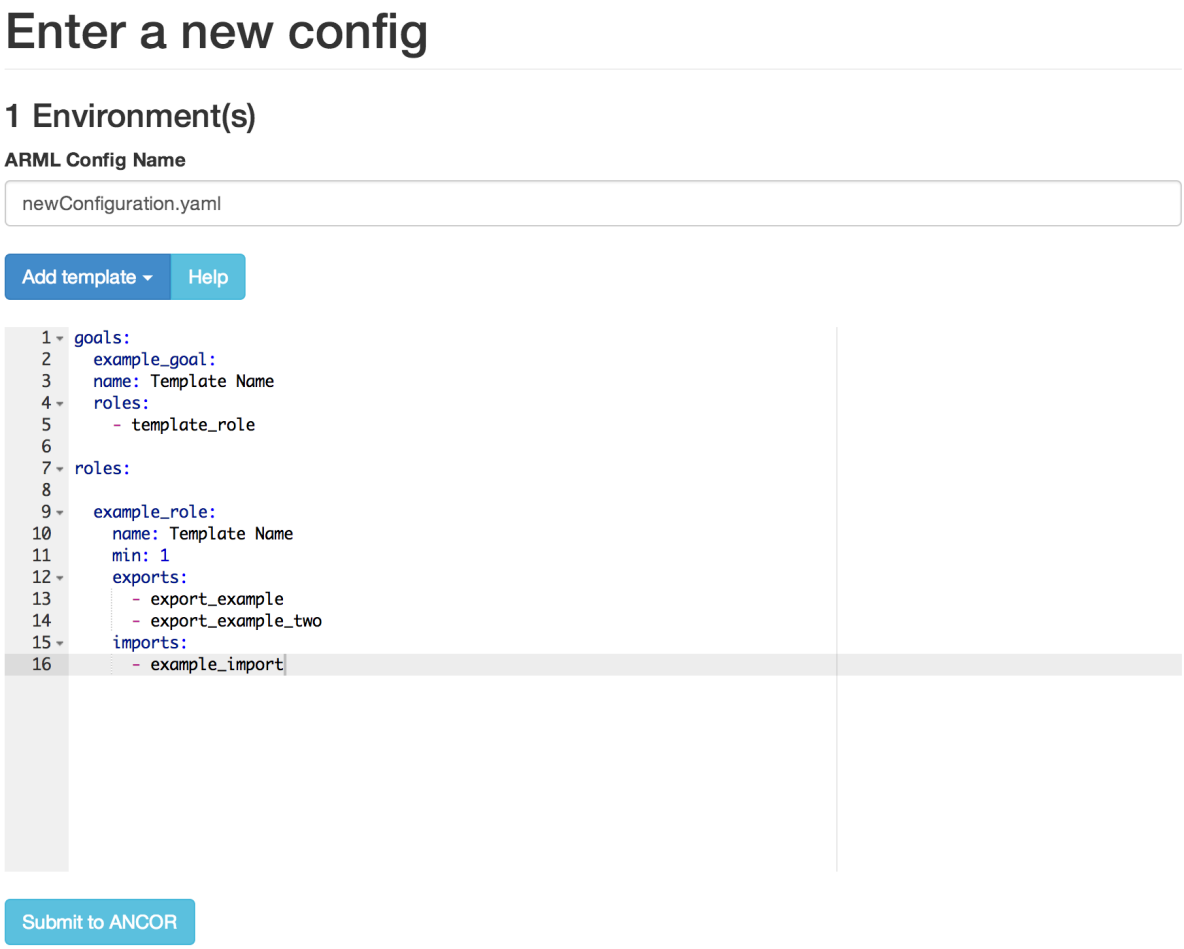

Figure 3.5: *An example of what the ACE Editor looks like with a sample config file.*

ui's ACE package for when the editor first loads and when someone makes a change within the editor. These options are just the names of the functions from within the controller. An example of how the onChange and onLoad functions work can be seen in figure 3.7.

```
<div id="editor" ui-ace="{
  mode: 'yaml',
  useWrapMode: true,
  onLoad: loadConf,
  onChange: confChange
  }">
</div>
```

Figure 3.6: *Initializing the ACE Editor with angular-ui.*

```
$scope.loadConf = function(_editor) {
  var _session = _editor.getSession();
  _session.setUndoManager(new ace.UndoManager());
};

$scope.confChange = function(e, _editor) {
  $scope.submitData = _editor.getValue();
};
```

Figure 3.7: *Two examples that help operate the ACE Editor. loadConf is what initializes the editor, while confChange keeps the configuration model variable up to date as a user writes their configuration file.*

3.3.4 Table Filtering with AngularJS

One of the advantages of using AngularJS is how simple they have made implementing complex features in HTML and JavaScript. One example of this is the search box that exists on the Main and Tasks view for filtering the large tables of data from ANCOR. This consists of only a few components in the HTML template. The first one is the textbox. From figure 3.8, you can see the input text box tag with an ng-model attribute. This model variable is what keeps track of the input from the user.

The only other element to this powerful search function is the filter attribute when AngularJS builds the Task or Instance rows within the table. Looking at line 5 of figure 3.8, the final element in ng-repeat is *filter:searchText*. This means if a user starts typing and modifying the model searchText, it will filter the results in the table automatically. This is a very powerful feature that only required a few lines of code thanks to AngularJS.

```
1  . . .
2  <input type="text" placeholder="Search tasks..."
3    class="form-control" ng-model="searchText" />
4  . . .
5  <tr ng-repeat="task in tasks | orderBy:predicate:reverse | filter:searchText">
6  . . .
```

Figure 3.8: *An example of how AngularJS easily creates a filtering search box within a template view.*

3.4 Component Breakdown

In this section, I will talk about the different components of ANCOR Dashboard. I will cover everything that exists in the controller as well as the viewmodel for each component.

3.4.1 Main

The main component of ANCOR Dashboard is the root route of the project. This means when a user first visits the main URL for ANCOR Dashboard, they will be presented with this view. This is the view in charge of displaying information about instances from ANCOR. Once a user visits Main, the controller will then query ANCOR through REST for its current version, all of the goals, all of the roles, and all of the instances.

These HTTP GET calls will then save the relevant information in the $scope model for the view to use. Querying goals will get the name of the current deployment. This name is saved and shown as a title on the front page. Roles are queried for the add instance dropdown. Each role slug is saved so a user can select one from a dynamically generated dropdown.

With the call against ANCOR instances, there is a bit of extra logic to keep track of each instances current stage and planned stage. The current stage is what is displayed to the user on the front page. It also shows how many total instances there are. Finally, data is saved and sent to the network graph generation script. This information is a set of data that shows how all of the instances are related to each other.

Two functions exist in this controller to replace and delete instances. They both are given an instance id, form the correct URL, and then make the appropriate REST HTTP call (either POST or DELETE) to ANCOR.

There is a helper function to determine which label to apply to an instances state from the Instance table. This is needed so that the HTML template file can stay as simple as possible and let the controller do all of the logic for which class label to apply.

Finally, there are several functions in charge of showing the modal popup when a user is interested in viewing more information about each instance. These modal functions are mostly used by angular-ui's Bootstrap package.

3.4.2 Env

The Environment component is used to show the current deployment from ANCOR. When this route is first loaded, the controller queries ANCOR for its version and all of the current environments. At the moment, ANCOR Dashboard only supports one deployment, so we grab the first environment in the data returned.

The main function in this controller is the ability to delete the given environment. When the user clicks the delete button from the view, it passes the environment id to the controllers *deleteEnv* function. This function then builds the correct URL with the id appended onto the end, and makes the HTTP DELETE call to ANCOR.

Environment also has a helper function to determine if the delete button should be disabled or not. If the environment is locked, the button will become greyed out so the user cannot click on it. Otherwise, the button will be enabled.

3.4.3 Tasks

The Tasks view is what is in charge of displaying all current ANCOR Tasks. When the Tasks view is first loaded up by the user, the controller then queries ANCOR for its current version and all of the tasks. The HTTP GET for Tasks is different from other controllers however because it needed to be able to be refreshed constantly so a user could see the tasks progress. For this to happen, the HTTP GET call was wrapped up in a function within the controller. However since there is a call to that function $scope.getData(), it is invoked initially when the route is loaded. Another way to update the task list is to select the *Refresh* button located right above the filtering search box. Finally like the other endpoints, Tasks has a helper function to determine the label for each Tasks state.

3.4.4 Deploy

The Deploy view is where the user can write their own configuration file and deploy it directly to ANCOR. When the Deploy route is loaded, it will query ANCOR for its version and the current deployed environment. The environment is required just so the user can know if any environments are currently deployed to prevent any overlap.

roleTemplate and goalTemplate are what generates the template text for the configuration within the ACE Editor. It simply uses an already made string template and inserts it where the cursor is at within ACE.

There are two functions that relate directly to the ACE editor API. The first one is loadConf, which is in charge of initializing the editor. The next one is called confChange. This function is in charge of keeping the configuration data model up to date within Deploys model.

The deploy function takes the current data from the ACE editor and constructs a message to be sent off to ANCOR. Before this happens, it builds two urls: A URL for planning the new deployment which will contain all of the configuration file data, and a URL for telling ANCOR to begin the deployment. Once these URLS are created and the configuration data is saved, the deploy function makes these two HTTP PUT and POST calls (in that order) to ANCOR.

The modal for Deploy uses all of the angular-ui conventions for displaying a modal to the users. It displays an example configuration, and gives some help text to help the user understand how to write a configuration file.

Chapter 4

Evaulation of Dashboard

4.1 ANCOR Dashboard Evaluation

With the implementation of ANCOR Dashboard complete, there needed to be some testing to ensure that the dashboard was easy to use and as helpful as the ANCOR-CLI.

4.1.1 Speed

ANCOR Dashboard is very fast and responsive in most modern web browsers. Because of the Bootstrap framework, it can work in the majority of what users will choose as their favorite browser. Since AngularJS compiles all of its HTML, CSS, and JavaScript down to a minified version, loading up the dashboard will be extremely fast when deployed. Because of how AngularJS is deployed, it will also be faster to respond compared to other popular MVC frameworks like Ruby on Rails or .NET as explained in section 3.2, Prototyping.

4.1.2 Usability of ANCOR Dashboard vs ANCOR-CLI

This project originally set out to cover all of the use-cases (section 2.4) defined by the developers of ANCOR-CLI. Because of this, ANCOR Dashboard is able to do everything

that the ANCOR-CLI can do. Since ANCOR Dashboard is a web application, it has an easier time displaying relevant information to the user compared to the ANCOR-CLI. In ANCOR-CLI's case, the best it can do is display the json data in formatted tables within a terminal. This is just the nature and limitation of console applications compared to web application frameworks with rich visuals and point-and-click functionality.

4.1.3 Using ANCOR Dashboard to Control ANCOR

Finally, ANCOR Dashboard should be able to control ANCOR in such a way that it takes complete advantage of the REST API it has provided to interact with. With the addition of the powerful ACE Editor, AngularJS components like HTML templating to display relevant data, filtering to search through large data sets, dynamically generated network graphs, and other usability operations against ANCOR, a user is able to do whatever they could need to accomplish with ANCOR Dashboard.

4.2 Future Work

In this section, I will briefly go over some future work that could be done to the dashboard that would improve the user experience.

Eventually ANCOR is planning on creating authentication for their REST API. When this happens, it will be important for ANCOR Dashboard to also have a way to authenticate users.

At the moment, if communication goes wrong with ANCOR, there is not an intuitive way to let the user know. Errors will be displayed within the JavaScript console of a browser, however the average user will not think to look there if something unusual happens. It would be nice to give more feedback to the user when RESTful operation errors occur.

The instance network graph can also be improved to make it more helpful to the user. One improvement that could help is making the node links toggle between hidden and

visible. Currently, there is no logic within the D3js script that keeps track of each nodes links once they are drawn. They all belong the the same CSS class. An improvement might be to separate each nodes links into different CSS classes, and then when a user clicks on a node toggle the CSS of its links between hidden and visible. This might make analyzing the network graph a little easier.

A usability improvement could be made to the tables in the Main and Tasks view. Adding a sortable toggle between each column attribute would improve how a user might go about looking at the data from ANCOR.

Chapter 5

Conclusion

In this report, I have presented ANCOR Dashboard, a front end web application dashboard that interacts with the ANCOR project. Built with the latest web technologies, ANCOR Dashboard is a powerfully light tool that makes using ANCOR easier. Designed with the intention of replicating all of the functionalities that the ANCOR-CLI provides, combined with the visual advantages that a dashboard provides, ANCOR Dashboard helps ANCOR users be more productive and gives them a new experience when looking at the state of their deployed IT system.

5.1 Source Code

The complete source code is currently located on a public repository on Github. The URL can be found below:

- https://github.com/arguslab/ancor-dashboard

The repository offers a quick start guide for those wishing to contribute or test out ANCOR Dashboard. It also provides the binary files required for users to deploy within an Apache or Nginx deployment. This is provided so users will not have to install all of the dependencies needed to run the project locally.

Bibliography

[1] S. Few. *Information Dashboard Design*, volume 1st Edition. O'Reilly Media, Inc., 2006.

[2] I. Unruh, A. G. Bardas, R. Zhuang, X. Ou, and S. A. DeLoach. Compiling abstract specifications into concrete systems bringing order to the cloud. Kansas State University, Manhattan, KS, 2014. (Unpublished).

[3] Argus lab moving-target defense, April 2014. URL http://www.arguslab.org/mtd.html.

[4] Yeoman - modern workflows for modern webapps, March 2014. URL http://yeoman.io/index.html.

[5] Bower - a package manager for the web, March 2014. URL http://bower.io/.

[6] Grunt: The javascript task runner, March 2014. URL http://gruntjs.com/.

[7] Angularjs - superheroic javascript mvw framework, March 2014. URL http://www.angularjs.org/.

[8] Bootstrap, March 2014. URL http://getbootstrap.com/.

[9] Top starred projects on github, April 2014. URL https://github.com/search?q=stars%3A%3E1000&ref=cmdform.

[10] Ace - the high performance code editor for the web, March 2014. URL http://ace.c9.io/.

[11] D3.js - data driven documents, March 2014. URL http://d3js.org/.

[12] R. T. Fielding. Architectural styles and the design of network-based software architectures. University Of California, Irvine, 2000. (Dissertation).

Appendix A

ANCOR Dashboard Use-Case Screenshots

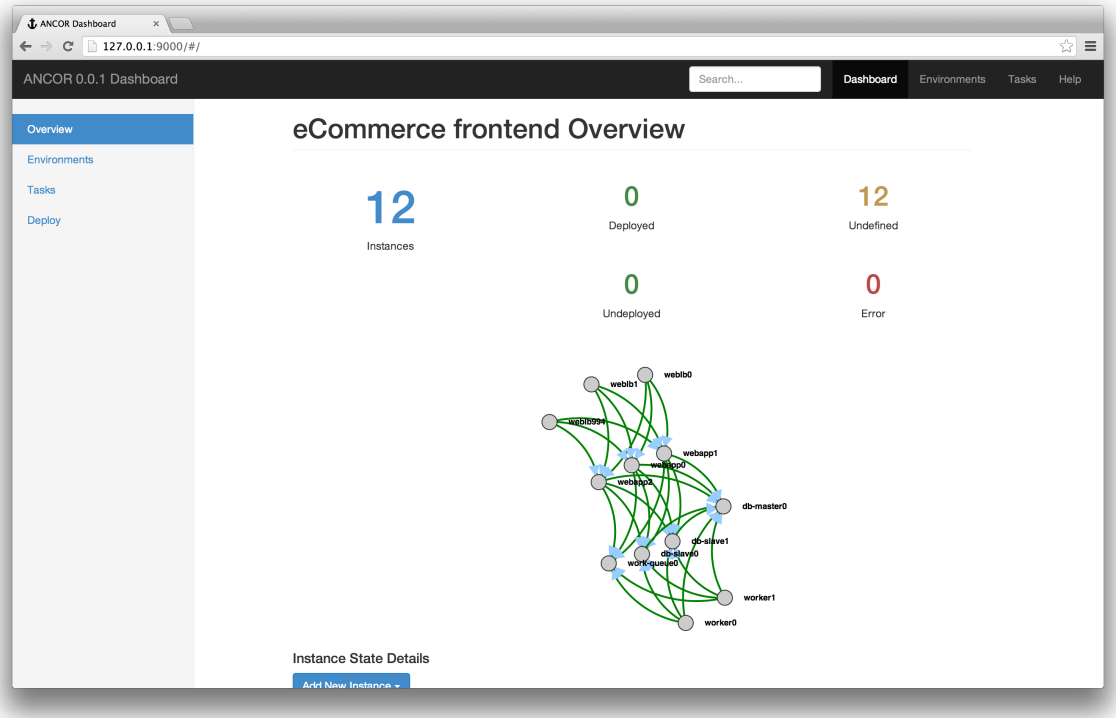

Figure A.1: *The main view of a deployed ANCOR system.*

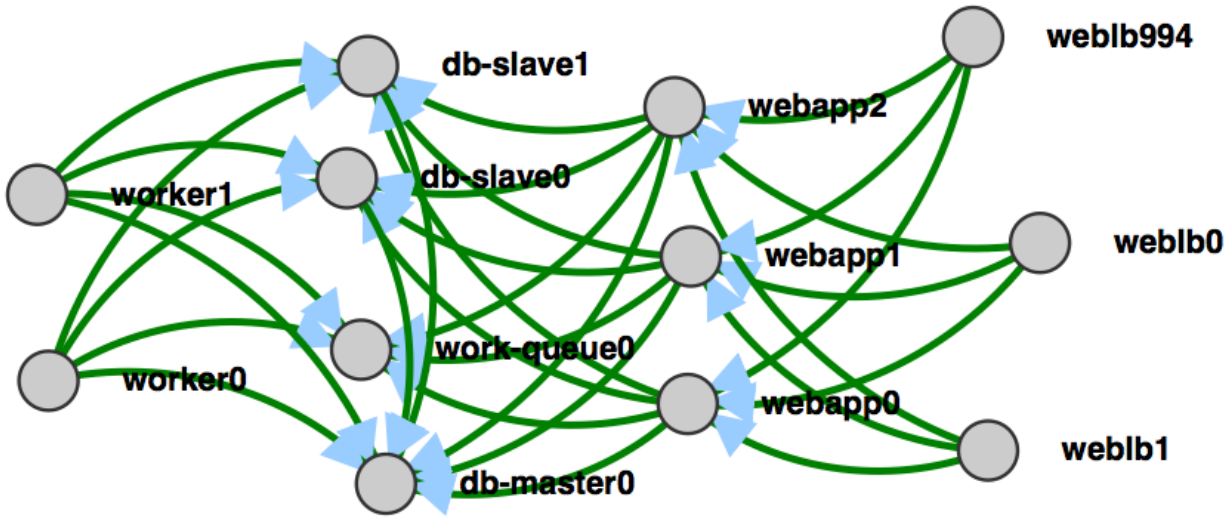

Figure A.2: *A dynamically generated network graph.*

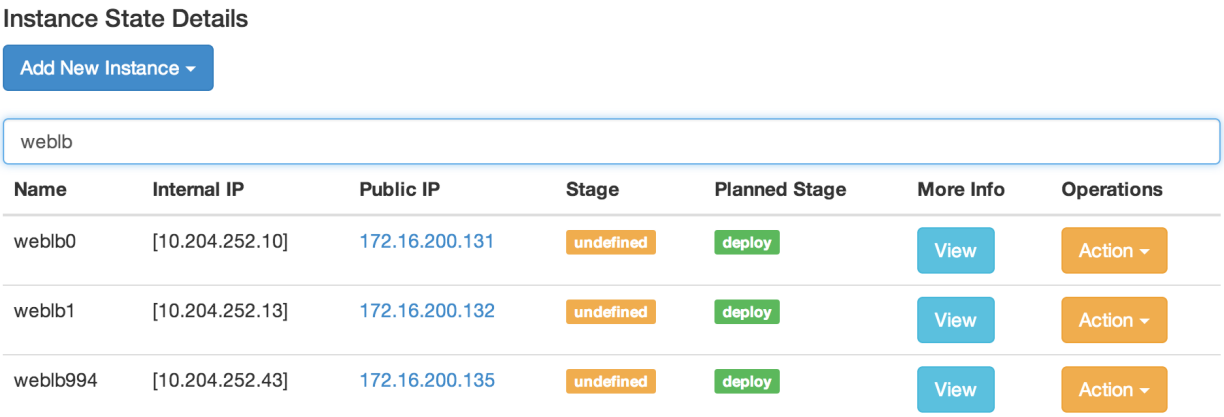

Figure A.3: *A user searching for weblb from the instance table.*

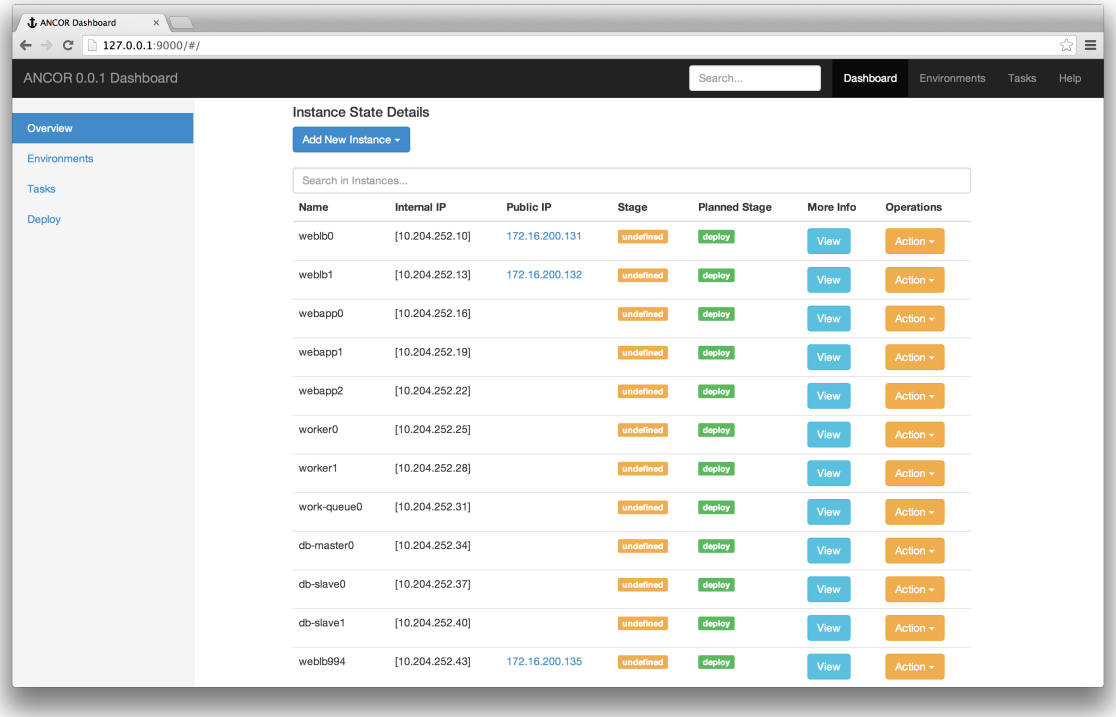

Figure A.4: *The instance table shown on the main view.*

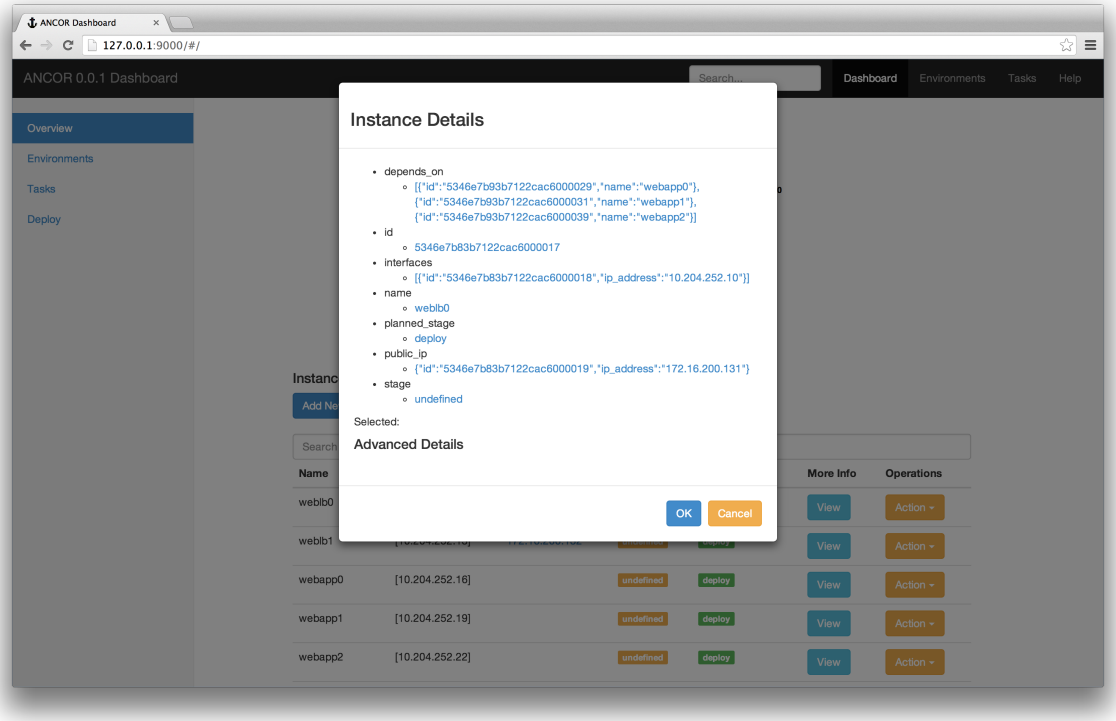

Figure A.5: *The instance modal view shown when a user wants more information.*

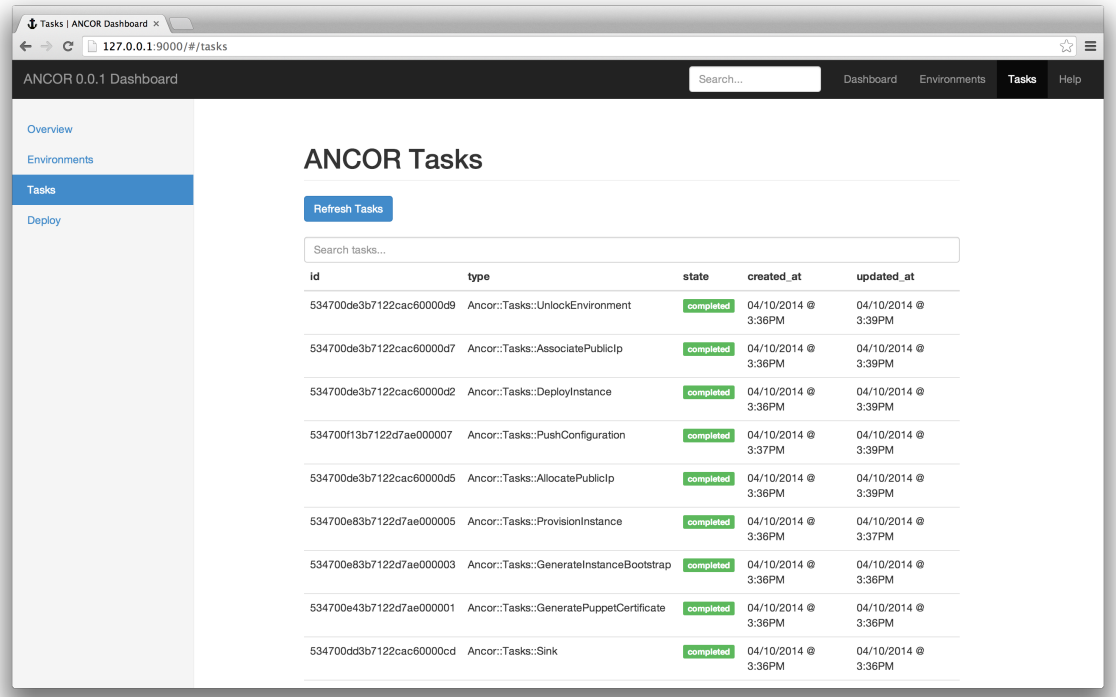

Figure A.6: *A list of all current Tasks.*

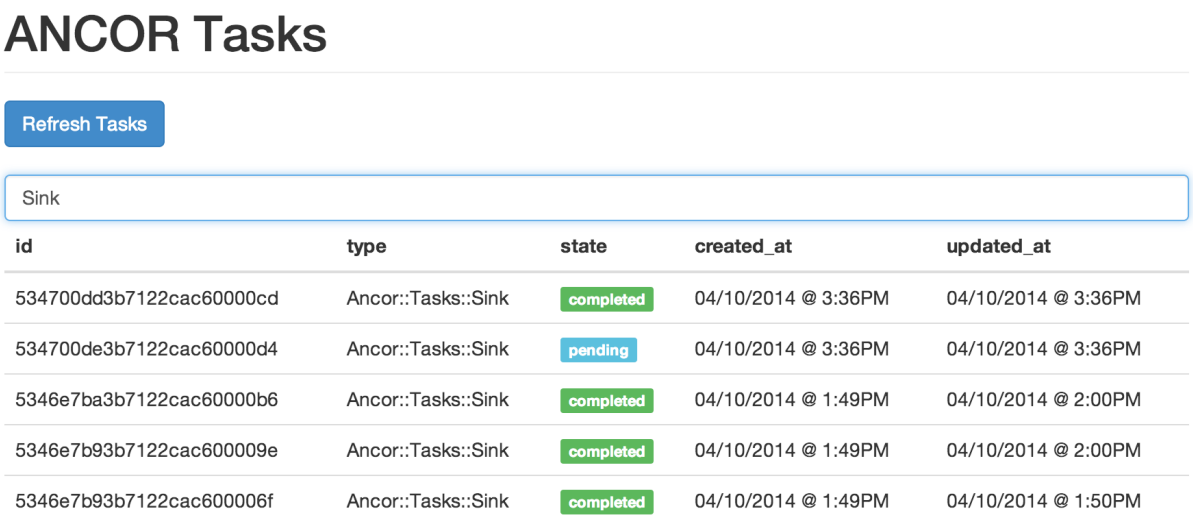

Figure A.7: *A filtered list of all current Tasks related to Sink.*

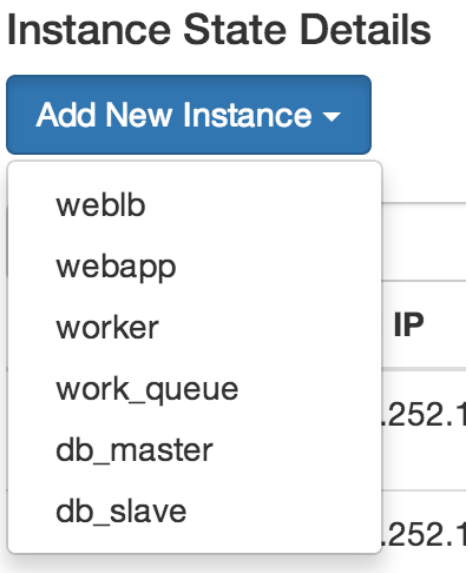

Figure A.8: *A dropdown of all the current instances/roles to add to the deployment.*

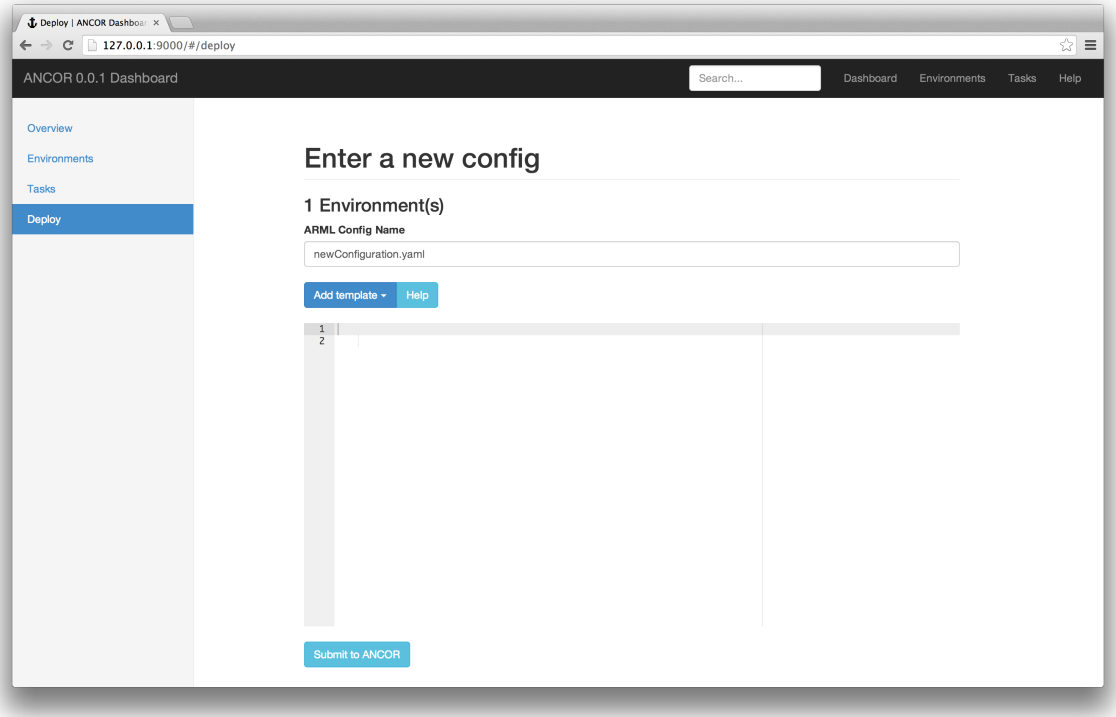

Figure A.9: *The deploy view, where users can write their own configuration file.*

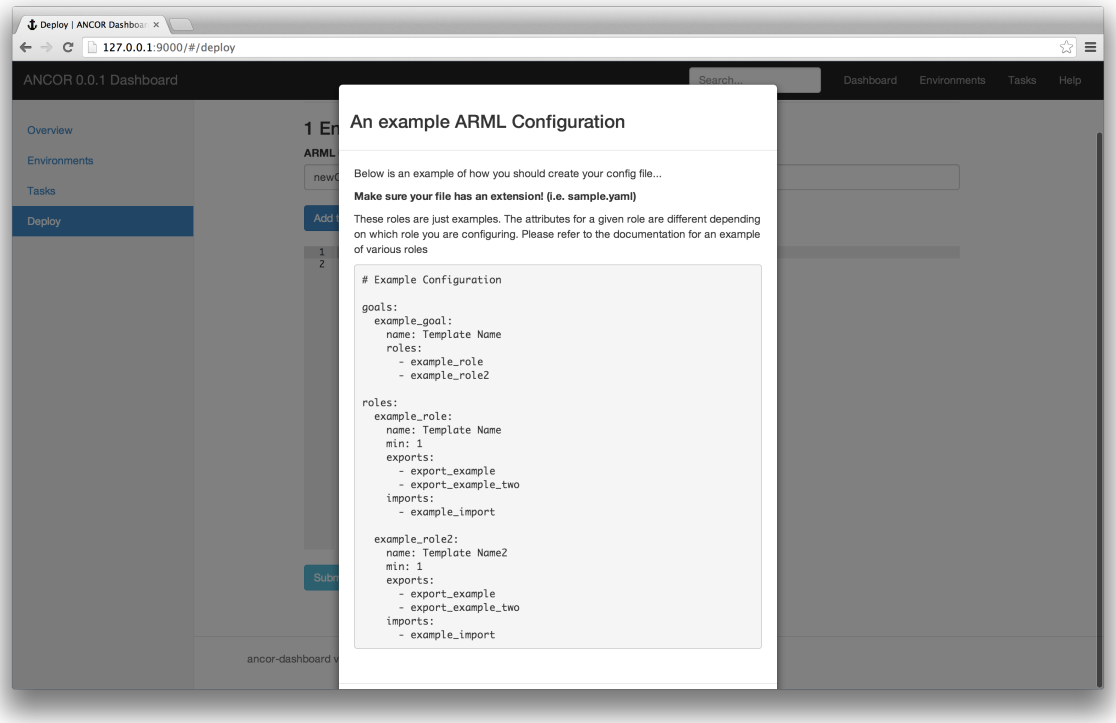

Figure A.10: *The deploy help modal view.*

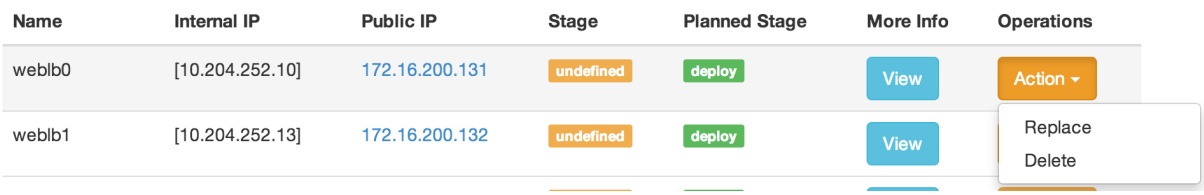

Figure A.11: *Replacing or deleting an instance.*

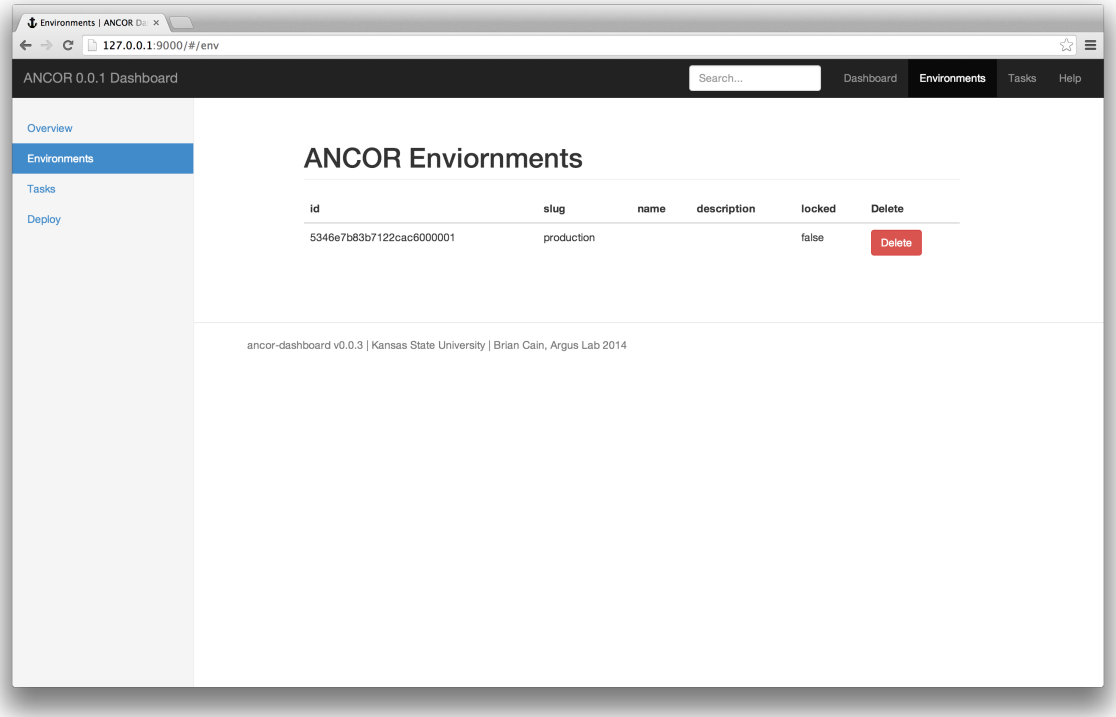

Figure A.12: *The environments view.*