Virtual Environments: Easy Modeling of Interdependent Virtual Appliances in the Cloud

Xabriel J. Collazo-Mojica
S. Masoud Sadjadi
Florida International University
{xcoll001,sadjadi}@cs.fiu.edu

Fabio Kon
Universidade de São Paulo
fabio.kon@ime.usp.br

Dilma Da Silva
IBM T.J. Watson Research Center
dilmasilva@us.ibm.com

Abstract

We present our ideas for modeling groups of interdependent virtual machines in the cloud. We call these models *virtual environments*. This abstraction is built on top of virtual appliances and the services they provide. We discuss previous attempts in this domain and present our motivations for working on an uncomplicated model for non-expert users of cloud computing such as Web developers and CS students. Visual and internal representations of the model are presented. Early work on a prototype implementation is described. We argue that easier-to-use models such as ours are needed for today's and tomorrow's distributed applications.

Keywords virtual environment, virtual appliance, flexible modeling, cloud computing.

1. Introduction

In this position paper, we present our ideas for modeling groups of interdependent virtual machines in the cloud. Designing these compositions require a good understanding of the underlying details such as the software installation or the network configuration. They are typically deployed in a cloud layer called Infrastructure as a Service (IaaS). Each IaaS provider has different APIs to configure the virtual machines and their connections, requiring the user to learn the details of a new API if they would like to migrate. As of now, users such as Web developers and CS students have to deal with these low-level details, which may entail configurations that depend on various machines in the environment. Research has pointed the need for better tools for composition in the cloud [16]. Clearly, an easier to understand model can help non-experts in cloud computing to develop solutions in this domain.

Sapuntzakis et al. presented in 2003 the idea of a virtual appliance, effectively treating full stacks of software applications and OS as updatable image files [15]. These virtual appliance files could then be cloned in bare metal computers, or instantiated in virtual machines. Still, when the time comes to compose multi appliance systems with interdependencies, Sapuntzaki's implementation relied on defaults from software vendors for most of the configurations. This may not be the case for software in Web development (e.g., when trying to configure a database server and its clients). Similarly, the configuration of the network connection between appliances had to be done by hand. In addition, various recent attempts at automatically configuring virtual appliances and their network dependencies have been presented [10, 11, 13], but they all rely on having experts on appliance configurations, or on specific IaaS providers.

We propose that with proper modeling of these kind of scenarios, the configuration problems can be abstracted away for non-experts. We call these models *virtual environments*. By designing virtual environments, non-expert users can easily architect interdependent virtual appliances. The key idea is to model each appliance with the services they provide. We model these services with *service endpoint* connections. How many service endpoints an appliance has depends on the installed applications. These endpoint connections allow an appliance to be seen as providing and/or consuming other appliance's services. This way, users can compose appliance to appliance dependencies by matching service consumers with service providers. No notion of the underlying network nor software provisioning needs to be done. Other interesting detail of this abstraction is the separation of the model from the deployment, effectively decoupling the virtual environment from the IaaS provider.

Although this model simplifies composition for end users, it also presents various technical challenges. The principal ones are: 1) network and software configurations need to be inferred from the specified service endpoint connections, 2) changes in the environment model need to be reflected on running deployments, and 3) virtual environment models should be deployable on different IaaS providers. These are very interesting problems in the areas of autonomic computing, model transformation, and systems. Our research group has worked on these topics before and we intend to leverage our experience to tackle these problems.

In the rest of this paper, we focus on the modeling aspect of virtual environments. Specifically, we present the details of our flexible modeling approach and we discuss a prototype system in which non-expert users can design these environments. We argue that our approach is a natural way to specify appliance to appliance dependencies.

2. Motivation and Related Work

Our research is driven by the needs of non-expert users of cloud computing like Web Developers and CS students. At Florida International University, we have a new IT curricula where students pick up the skills necessary to run today's datacenters [4]. We simulate real scenarios by the use of virtualization, and students have to configure everything from scratch. Nonetheless, we also want to push forward virtual infrastructure management. As most CS students today are novice users of virtualization and cloud computing services, we believe that they are good candidates to prove the usefulness of new modeling ideas. We have identified the

need of a flexible tool to allow students to deploy working systems from day one of usage.

Similarly, we envision that different IaaS APIs and providers will continue to proliferate. Web developers should not be bothered by the configuration details of interdependent virtual machines, and they should be able to deploy working solutions by themselves and without the help from virtualization and IaaS experts. That is, we have identified the need for better abstractions from the current IaaS implementations provided by vendors such as Amazon [2] or GoGrid [6]. Thus, we propose a modeling approach that is abstract enough to allow these interdependent systems to be easily designed, is fast to deploy, and limits the effects of IaaS vendor lock-in.

Teaching computer networks using virtual machines and virtual networks has been researched before by the GINI project [14]. They provide the user with 'lightweight virtual elements', which can be instantiated on laptop and desktop computers. However, they only support 'user-mode Linux' instances, which are not sufficient for many of todays IaaS providers.

Commercial applications implementing a similar modeling approach are available [1, 3]. They only offer closed-source implementations and only work on their proprietary cloud platforms. IBM has worked on a similar project, but their implementation assumes that users are experts in the domains of virtual image provisioning, image composition, and composition deployment [13]. While they target enterprise customers, we target non-expert cloud users.

Platform as a Service (PaaS) providers, such as Google AppEngine [7], abstract away the underpinnings of a fully working web application. Of course, this means that the user has to learn the vendor's API, and that migrating the application to other PaaS provider implies changing most of the implementation. Our work envisions models that once specified do not need to be changed because of a vendor switch.

3. Virtual Environments

A *virtual environment* is a model of a group of interdependent virtual appliances. The life-cycle of these environments is presented in Figure 1. First, the user should *design* the environment. Designing an environment entails specifying virtual appliances and defining properties between them. Once designed, virtual environments can be *modified* or *deployed*. Modifying an environment is changing some specification of an appliance or a relation between appliances. Deploying an environment is enacting the model in an IaaS provider. Once deployed, a virtual environment could suffer a change to any of these states: *terminated*, *dynamic modify* or *static modify*. Terminating a virtual environment is simply shutting down all the appliances of the running model. Dynamic modification entails making a change to the model (e.g., adding appliances or modifying properties between appliances) while it is in the deployed stage. Static modification is similar to dynamic modification, but the environment is shutdown temporally before the changes, and re-deployed after the modifications.

From the point of view of the end-user, virtual environments should have the following properties:

1. **Easy to understand**: views for the design, deployment, change management, and monitoring of these environments should only presents what is strictly necessary to realize them. Advanced options should be available but normally hidden.
2. **Self-configurable**: once the user has specified a basic description of what he/she needs, the model should be able to instantiate and configure all the details automatically by following some general policies.
3. **Present deployment choices**: modeling should be abstract enough to allow for an implementation to present deployment choices. Given the variability of current IaaS APIs, this is the main challenge of our approach.

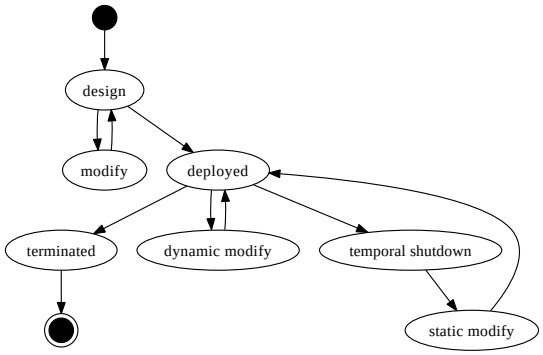

Figure 1. Life-cycle of a virtual environment.

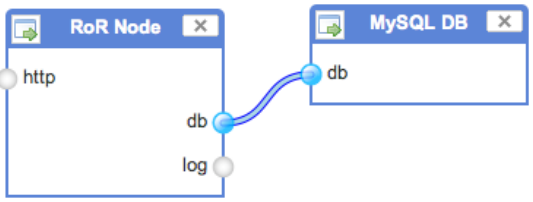

Figure 2. Service endpoint connection between two appliances.

3.1 Visual Model

Research has shown the benefits of visual aids when designing system architectures, claiming a gain of over 60% in comprehension [12]. Since our target users are supposed to be non-experts in configurations for the cloud, a graphical representation is desirable. We model virtual appliances with boxes with *service endpoints*. The boxes represent all the necessary software, the OS and the configurations necessary to support the services provided or consumed by the endpoints. Figure 2 presents an example of two virtual appliances. The box entitled 'RoR Node' is a representation of an appliance provisioned with the Ruby on Rails web framework (and all other needed software). Similarly, the box entitled 'MySQL DB' is an appliance provisioned with a MySQL database. These appliances have been interconnected with a 'db' link by joining the corresponding endpoints. In our model, there is no need to configure IP addresses, ports or configuration files. Endpoints have type and are inspected with a dictionary of valid connections. If any is found to be invalid, a visual cue will be provided to the user.

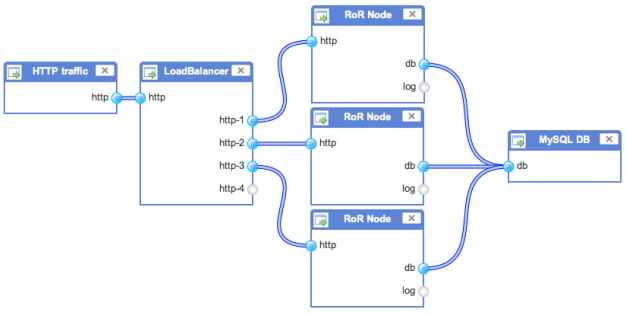

Figure 3. A complete virtual environment model.

Using our visual model, users will be able to design full appliance-based architectures. On Figure 3, we present a full representation of a typical dynamic web deployment. Building from Figure 2, we have added two more Ruby on Rails nodes, a load balancer, and a special node to enable HTTP traffic in the environment. Note that all connections are modeled by the services that each appliance consumes or provides.

3.2 Internal Model

Once the user has designed a virtual environment, it would be persisted in a XML representation. Figure 4 shows a simplified version of the XML that is generated from the example environment in Figure 2. This representation includes a list of the software packages that each appliance needs (lines 9-11 and 20-21). This information is used for the provisioning of each appliance. A user can choose to deploy an instance of a virtual environment, to later make modifications to the model, and then deploy another instance of the model. Because we intend to provide the ability for runtime modifications, we need to keep track of the model of each separately instantiated environment. Therefore, we include revision data for both full environments and individual appliances (lines 2,5, and 16).

Because endpoint connections have type, we can also include special properties in the connections (lines 29-35). These do not need to be specified, but are allowed. Figure 4 includes a 'MaxAllowedConn' property in the database connection between the appliances (line 31). As all service endpoint connections are explicitly stated on the model, security measures can be taken. Note that the only service endpoint specified in the XML example is the database connection. Therefore, all other ports in both appliances can be safely blocked when the environment is deployed. This could be realized, for example, by utilizing the IaaS provider's security features (e.g., Amazon's EC2 security groups [2]).

3.3 Trade-Offs

Administrator users manage the availability of service interfaces, which are bundles of needed endpoints and software dependencies for specific software packages. Going back to Figure 2, the service interface for the Ruby on Rails service is the bundle of endpoints 'http', 'db', and 'log'. The user composes appliances by selecting which service interfaces he/she would like to have in an appliance.

This constrain limits the possibilities of software packages. That is, the user will not be able to design appliances with software that is not known by the admin. This strict control

```
1  <?xml ver="1.0" encoding="UTF-8"?>
2  <VirtualEnvironment rev="1">
3    <VENodeList>
4      <VENode>
5        <Appliance rev="1">
6          <name>RoR Node</name>
7          <guestOS ver="10.04">Ubuntu</guestOS>
8          <dependencies>
9            <dep ver=">2.2.16">Apache HTTPD</dep>
10           <dep ver="2.3.8">Ruby on Rails</dep>
11           <dep ver="1.2.3">...etc...</dep>
12         </dependencies>
13       </Appliance>
14     </VENode>
15     <VENode>
16       <Appliance rev="2">
17         <name>MySQL DB</name>
18         <guestOS ver="10.04">Ubuntu</guestOS>
19         <dependencies>
20           <dep ver="5.1.49">MySQL</dep>
21           <dep ver="1.2.3">...etc...</dep>
22         </dependencies>
23       </Appliance>
24     </VENode>
25   </VENodeList>
26   <VENodeRelationList>
27     <VENodeRelation>
28       <n1Name>RoR Appliance</n1Name>
29       <n1Int type="database-consumer">db</n1Int>
30       <relationProperties>
31         <prop key="MaxAllowedConn">5</prop>
32         <prop key="Etc">...</prop>
33       </relationProperties>
34       <n2Name>MySQL DB</n2Name>
35       <n2Int type="database-provider">db</n2Int>
36     </VENodeRelation>
37   </VENodeRelationList>
</VirtualEnvironment>
```

Figure 4. Example XML of a virtual environment.

helps to have a manageable number of configuration alternatives. This is a needed trade-off to keep the administration burden low.

4. Prototype

We have developed a limited prototype that allows users to specify virtual appliances, virtual environments, and can also schedule environments for instantiation and deployment. The current implementation is built on top of the Ruby on Rails platform [8]. Only the user interface and the internal representation of the model are functional as of now. The latest version of the project can be accessed at [5]. It is licensed under GPLv3.

In the prototype, a user can first choose which broad software packages are needed in each appliance. The system can then infer which endpoints are needed according to the chosen packages. To design virtual environments, we developed a visual modeling view, presented in Figure 5. The current implementation is based on the WireIt Library [9]. It provides a drag-and-drop interface, where users compose the desired environment by dragging the already defined appliances and connecting the service endpoints with wires. Once the user is done with the design, he/she can click on the 'deploy' button, where a wizard will be presented as to enact the environment. We are currently working on this feature.

The prototype also presents an admin view where new service interfaces can be defined. Recall that service interfaces are sets of endpoint connections. Another future admin view would provide the ability to manage the available IaaS providers.

5. Concluding Remarks and Future Work

In this position paper, we presented our ideas for modeling groups of interdependent virtual machines in the cloud. First, we introduced the problem by referring to previous

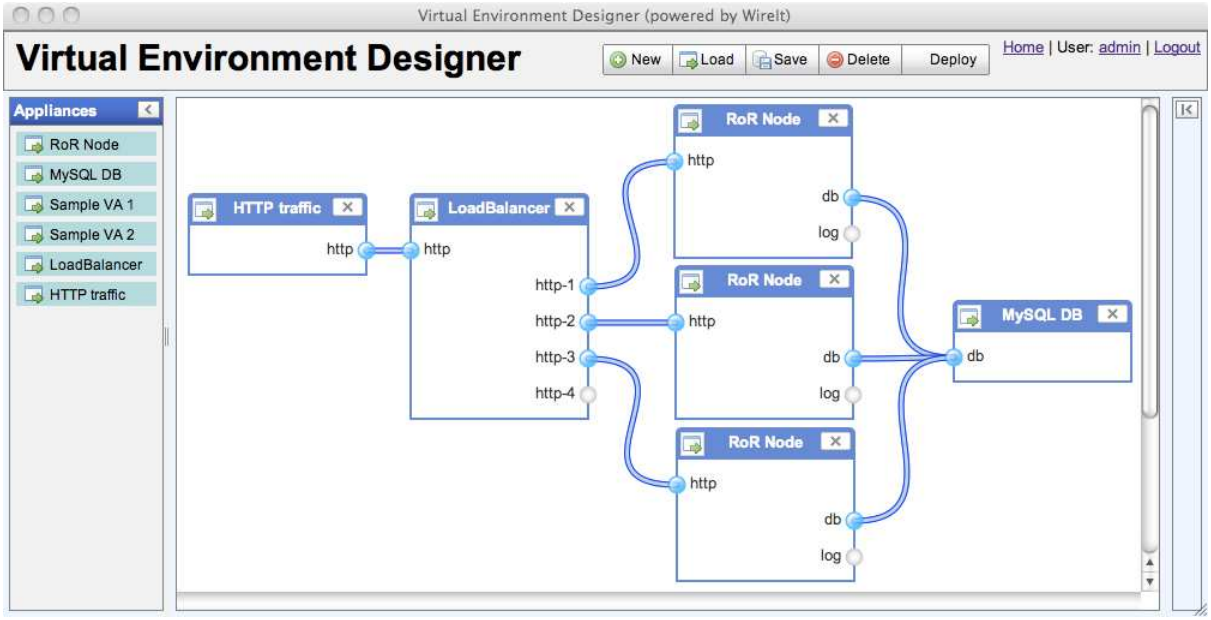

Figure 5. Designing a virtual environment.

work in the area. Although there is recent related work which try to address similar problems, none of these solutions present abstractions appropriate for non-experts. We defined our virtual environment model by presenting its life-cycle and properties. A visual and internal representation for these environments was discussed, including the main trade-off of the approach. Finally, an early prototype with limited functionality was introduced.

We envision that our idea of abstracting away unnecessary configuration details from non-expert users will enable fast deployment of working systems through an automatic configuration process. For future work, in short term, we hope to have a prototype that can deploy fully working virtual environments with dynamic change support. In long term, we will put our efforts on providing various IaaS deployment choices.

Acknowledgments

We appreciate the discussions held with David Villegas. This work was supported in part by the National Science Foundation under OISE-0730065, by the US Department of Education under P200A090061, and by IBM.

References

[1] 3Tera Inc., Aug 2010. URL `http://www.3tera.com/`.

[2] Amazon Elastic Compute Cloud, Aug 2010. URL `http://aws.amazon.com/ec2/`.

[3] Elastra Corporation, Aug 2010. URL `http://www.elastra.com/`.

[4] FIU's IT Automation Course, Aug 2010. URL `http://users.cis.fiu.edu/~sadjadi/Teaching/IT%20Automation/Spring%202010/Syllabus.html`.

[5] Virtual Environments Designer Portal, Aug 2010. URL `http://acrl.cis.fiu.edu/ve_designer`.

[6] GoGrid Cloud Hosting, Aug 2010. URL `http://www.gogrid.com/cloud-hosting/cloud-servers.php`.

[7] Google App Engine, Aug 2010. URL `http://code.google.com/appengine/`.

[8] Ruby on Rails Web Framework, Aug 2010. URL `http://rubyonrails.org/`.

[9] WireIt - A Javascript Wiring Library, Aug 2010. URL `http://javascript.neyric.com/wireit/`.

[10] T. Chen, Y. Wang, Y. Ren, C. Luo, D. Qian, and Z. Luan. R-ECS: reliable elastic computing services for building virtual computing environment. *ICIS '09: Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, Nov 2009.

[11] K. Keahey and T. Freeman. Contextualization: Providing One-Click Virtual Clusters. *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, pages 301–308, 2008. doi: 10.1109/eScience.2008.82.

[12] J. Knodel, D. Muthig, and M. Naab. Understanding software architectures by visualization–an experiment with graphical elements. *Reverse Engineering, 2006. WCRE '06. 13th Working Conference*, pages 39–50, 2006.

[13] A. Konstantinou, T. Eilam, M. Kalantar, A. Totok, W. Arnold, and E. Snible. An architecture for virtual solution composition and deployment in infrastructure clouds. *VTDC '09: Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*, Jun 2009.

[14] M. Maheswaran, A. Malozemoff, D. Ng, S. Liao, S. Gu, B. Maniymaran, J. Raymond, R. Shaikh, and Y. Gao. GINI: a user-level toolkit for creating micro internets for teaching & learning computer networking. *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, Mar 2009.

[15] C. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. S. Lam, and M. Rosenblum. Virtual appliances for deploying and maintaining software. *LISA '03: Proceedings of the 17th USENIX Large Installation Systems Administration Conference*, pages 181–194, Aug 2003.

[16] K. Sripanidkulchai, S. Sahu, Y. Ruan, A. Shaikh, and C. Dorai. Are clouds ready for large distributed applications? *SIGOPS Operating Systems Review*, 44(2), Apr 2010.