Overcast: Eclipsing High Profile Open Source Cloud Initiatives

Chris Matthews[1], Stephen Neville[2], Yvonne Coady[1], Jeff McAffer[3], and Ian Bull[3]

[1] University of Victoria {cmatthew,ycoady}@cs.uvic.ca
[2] University of Victoria sneville@ece.uvic.ca
[3] EclipseSource{irbull,jeff}@eclipsesource.com

Abstract. Can Cloud Computing be used without the traditional clustered, high-availability and highly complicated software stacks from big vendors or high profile open source initiatives? This paper presents the position that by leveraging the virtualization that is inherent in the cloud, we can recompose cloud systems out of more simple building blocks, breaking the dependancy on complicated software stacks and creating more secure and reliable systems than at present.

1 Introduction

One of the challenges of cloud computing is the complexity of composing services in ways that will ultimately be not only be efficient, secure, and robust, but also scalable and sustainable in the face of system evolution. Virtualization has provided critical leverage within this domain. However, current coarse grained virtualization is arguably a mismatch in terms of cloud service compositions. Cloud infrastructure as a service (IaaS) providers all support the same granularity for system decomposition, consisting of a full operating system and a complete software stack.

In this position paper we discuss the possibility of composing cloud based systems out of a collection of smaller components. The ultimate goal being more maintainable, scalable systems, which break our dependence on the current large brittle software stacks used in IaaS and ultimately make IaaS a bit more like platfrom as a service (PaaS) computing.

In [1] we presented *MacroComponents*, designed specifically to combine the power of the isolation currently provided by virtualization with software engineering qualities borrowed from component systems for scalability and sustainability in cloud environments. In systems composed of MacroComponents, components run in isolation from the rest of the system, but without the full foundations of their more traditionally virtualized counterparts.

In this paper we discuss how MacroComponents would change the way cloud software is built, and what differences we could expect to see in systems built out of MacroComponents. We start with a discussion of current techniques for decomposing cloud systems in Section 2, then in Sections 3 and 4 we introduce

some technologies of interest: OSGi, P2 and CloudClipse, then in Section 6 we describe how these technologies might be used to make MacroComponents more potent in the cloud. In Section 7 we detail how MacroComponents might make cloud software systems better, and in Section 8 we describe some of the problems that could be encountered.

2 Decomposition and Virtual Appliances

Outside of the cloud, there are several reasons to split up a full software stack across separate virtual machines. The isolation imposed by virtualization provides a mechanism with which a system's engineer can decompose the system. This decomposition could be done to help bring the system closer to the conceptual model it is based on. It could also be done to reduce the cost of building or maintaining the system [2, 3].

Concretely, prior to the acceptance of virtualization, to ensure good hardware and utilization a company might have installed a web server and e-mail server on the same physical host. In a virtualized setup, that company might put their e-mail server in one virtual machine and their web server in a different virtual machine on the same physical host. This could have been motivated by several different reasons: first, to match the conceptual understanding of the system, second to separate the stakeholders of the system, third to separate proprietary modules from the rest of the system.

Matching the conceptual understanding of the system is something that is done to enhance the mental modeling of the system. Since e-mail and web servers are separate services, having them running on separate containers makes sense from that perspective. When the maintainers of the system are thinking of the services the system provides, it would be nice for them not to have to think about where the services are instantiated. For example, if the maintainers of the system wanted to move the e-mail server behind a firewall or to a new physical location, but keep the web server in its currently is, the conceptual understanding of the system is that the e-mail server can just be moved; however, that is not the case if the e-mail server and the web server are running on one operating system. In this particular example, if the web server and e-mail server were running on a virtualized platform in separate virtual machines there would be no dependency between the two, and the e-mail server could be moved without any consideration for the web server.

Sometimes it makes sense to decompose the system to separate the stakeholders involved in the system. For example, the Microsoft documentation warns strongly against running a domain controller and exchange e-mail server on the same Windows 2003 instance [4]. Statements like this can be motivated by several different issues of compatibility:

- legal: the license that software is written under,
- technical: two software products do not interact well after installed on the same system, based on several isolation properties:

  - Configuration Isolation: Configuration of one entity effects the other,
  - Temporal Isolation: One entity steals execution time from another,
  - Spacial Isolation: One entity interferes with another's memory or cache,
- security: software does not trust other software products on the system and
- quality of service: software cannot guarantee it will work well with other software installed.

2.1 Making it Easy

These are all motivating factors for the notion of the virtual appliance. A *virtual appliance* is a virtual machine with software pre-packaged and pre-configured within it. All the user of the virtual appliance has to do is download the virtual appliance and start it. Virtual appliances also tend to carry a full OS in their virtual machine, although often they have their non-essential services and applications removed. Libra [5] is an attempt to build a minimal library OS to support even smaller virtual machines. Libra provides a composable operating system that was shown to run a Java Virtual Machine (JVM). MinOS a minimal OS that is included with Xen [6]. It lacks some of facilities that are found in a typical OS such as a scheduler, or network stack. MinOS has been used to build small virtual machines that are built for a specific purpose, for example, running a simple C program, running an OCaml interpreter, or even providing an isolated domain builder for Xen [7]. These are all examples of efforts to reduce the size and complexity of virtual machines to make the system as a whole better in some way.

In Amazon's EC2 [8], users are presented with a large variety of free and paid preconfigured Amazon Machine Images (AMIs). Many of these could be considered virtual appliances, with common prebuilt software stacks in them. These images reduce the time it takes for clients to get their applications into the cloud, effectively reducing the setup to just application specific setup.

2.2 Building Monolithic Systems

Virtual appliances tend to be monolithic systems, a large collection of software all tuned to work together to perform the virtual appliance's task. But prefabricated virtual appliances have limits to their customizability.

The need for custom software stacks has been made evident by the success of products like rPath's rBuilder [9] or openQRM [10]. Both create customized images for virtual machines and the cloud. All the user has to do is provide the application, and these systems prepare the image for you.

One thing that is interesting about these systems is the interface that they present to their users. The interfaces make the virtual machine appear as though it is a collection of components. The user selects a kernel, some software packages, and perhaps a web server, and these systems compose them for you. At the end you are presented with a software system ready for the cloud. In this case these systems are building a monolithic software system on your behalf.

3 OSGi and P2

OSGi is a component model that has gained some popularity in the Java programming world [11]. Although not specified in Java, all the current implementations of the OSGi framework are for Java. OSGi provides a model in which components (bundles) and services can be defined and accessed along with some associated component lifecycle and composition operations.

Equinox is the reference implementation of the OSGi model. Equinox is part of the Eclipse project, and provides the implementation of OSGi that the Eclipse IDE uses. OSGi can even be used in distributed systems. R-OSGi [12] can be used to have OSGi components communicate with each other even when they are on different physical or virtual machines.

Because of the dynamic nature of OSGi components, and the ease with which modules in the system can be added, removed and upgraded, a natural addition to the OSGi world is P2, a provisioning platform [13]. P2 is the mechanism behind the newest version of the Eclipse IDE's update system, and provides a general purpose means for installing and updating an OSGi system. P2 was conceived out of the need to keep complex systems of versioned component which come from different places. P2 is designed to be extensible and flexible, and can do more than just adding and removing components from OSGi systems. P2 comes with a set of native operations that can perform many different common tasks, but P2 also provides the ability to extend itself as it is running with new operations.

4 CloudClipse

CloudClipse is a framework for helping get software into the cloud. It was a prototype system created as a Google summer of code project for the Eclipse Foundation. The initial version of CloudClipse provided an extra set of P2 operations which allowed an installer to create virtual machine image that could then have software installed into it. The first version provided support for fetching a compressed virtual machine image, mounting a virtual machine image, doing a RPM install into an image, and performing arbitrary commands inside the image via Linux's chroot facility. Currently, CloudClipse downloads one of several small linux distributions as the base to work from. Then that image can be customized by the P2 installer. Once the image is present on the local machine, P2 can install software from local RPMs or from another P2 repository. P2 can also configure the image by doing things such as setting up services to run on boot, changing the default root password, and installing ssh keys.

4.1 Future of CloudClipse

The ease with which P2 was adapted to make virtual machines images indicates it might be a good candidate to dynamically configure and install software in the cloud. We think that P2 is particularly well suited for installing and configuring

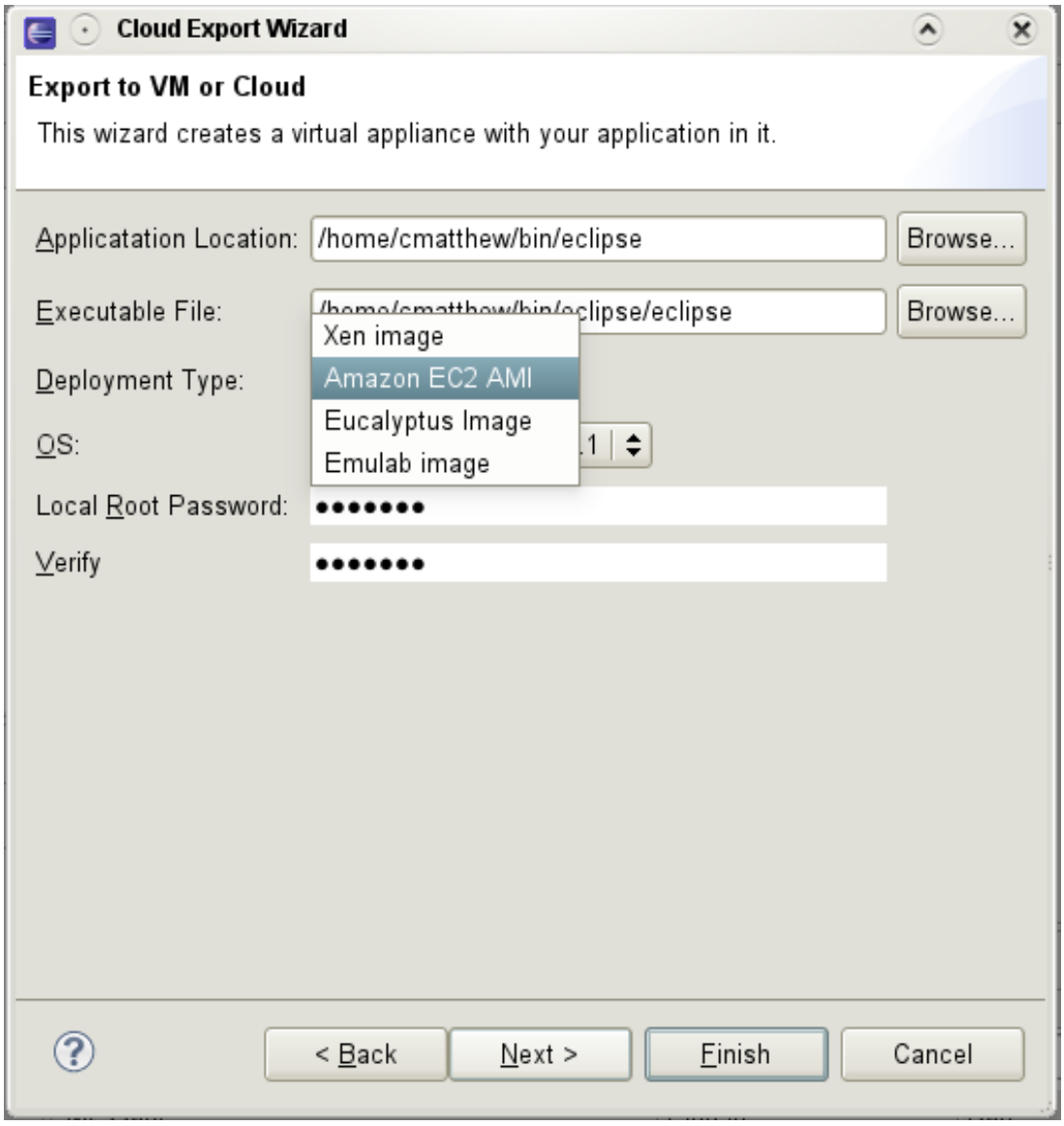

Fig. 1. Creating a new virtual machine with the Eclipse CloudClipse plugin.

the nodes from inside the cloud. CloudClipse could be used to set up a simple P2 system inside a cloud image; then, dynamically, P2 could find the software it needs from repositories inside the cloud and install them. CloudClipse provides a simple way of producing virtual machines with a small installation of base software to get the rest of the P2 installer going. CloudClipse can configure the operating system and kernel versions of installed libraries, then, once in the cloud P2 would be able to dynamically configure and install the rest of the software in the system. Ideally, P2 actually finds the software from within the cloud.

The ease with which CloudClipse, and other tools like openQRM and rBuilder build base images for the cloud, makes it almost a mechanical process.

5 MacroComponents

MacroComponents is a proposed approach to software compostion that aims to alleviate the burden of creating lightweight virtual machines [1]. With lightweight virtual machines, there is no need to keep an entire monolithic software stack in each virtual machine. The goal of MacroComponents is to decrease the size of virtualizable entities in the system to the point that small software components can be run in isolated virtual machines. Each virtual machine is merely a container for a single software component of the system. Because of the decrease in complexity of the software in each virtual machine, much of the traditional supporting software stack can be removed from the virtual machine as well.

To support the fine grained decomposition of a software system across virtual machines, MacroCompents would have to provide several facilities. Some of the most important of which are:

- programatic creation and configuration of virtual machines,
- fast communication between virtual machines,
- scalable, simple composition of virtual machines,
- programatic control over other virtual machines.

We believe with all these and other properties satisfied, software systems could be distributed as components across virtual machines.

Cloud computing heavily utilizes virtualization; so the question remains, would fine grained decomposition of cloud software into virtual machines be beneficial? We investigate this question in the following sections.

6 Leveraging Current Component Systems

OSGi could lend an interesting compent model to MacroComponents. There is some effort to use OSGi in the cloud already [12]. Its model for dynamic service binding maps well to cloud software, but also could map well to MacroComponents. In systems built with OSGi, components can be added and removed from the running system. Using R-OSGi, it was shown that the failure of distributed

communication can map directly onto the OSGi services failure modes [14]. Extensive tooling already exists for OSGi; for example, the Eclipse PDE is designed for OSGi development [15]. Furthermore, there are several popular implementations of the OSGi runtime which already exist.

P2 and CloudClipse also could provide mechanisims to assist in the complex creation of virtual machines. P2 could prepare a MacroComponent's virtual machine images, and prestock them with the simple runtimes they would need. Furthermore, an instance of P2 could be used inside the virtual machine, to load that virtual machine's component(s) dynamically. This alleviates some of the per-MacroComponent setup that would need to be done.

7 The Case for Fine-grained Cloud Compositions

Section 2 listed many reasons to decompose a software stack, and MacroComponents as described in Section 5 allow us to decompose a system in a more fine grained manner. If this decomposion took place, we could expect to see many benefits to the software system as a whole.

The dynamic nature of OSGi systems makes it easy to swap one service for another similar service. This makes the system easier to evolve. Componet version dependencies can be explicitly noted, live upgrades are possible and maintance can replace smaller parts of the system.

Because each component is isolated in its own virtual machine, the component has explicit configuration isolation, temporal isolation and spacial isolation. It should be nearly impossible for other elements of the system to interfere with the expected operation of a component.

In an OSGi service based architecture, components bind dynamically and only temporarily, so redundancy is easily inserted into the system. Multiple instances of important componets can be used, and failover can be automatic. There is a lower cost to keeping hot standbys for important components because a smaller part of the system (just the component) is all that has to be kept ready. Furthermore, new instances of components can be brought up much faster than starting a whole new version of the software stack. Both of these properties have good implications for the robustness and scalability of the system. The need for traditional high availability software mechanisms is reduced.

In the event of a security compromise, only one component will initially be exposed to the attacker. If the attacker wants to compromise more of the system they have to also find a vulnerability in the hypervisor or the inter virtual machine communication.

The hypervisor provides a good generic base runtime. They allow many different sizes and types of operating systems and runtimes to be used. Almost anything could be used inside each macro component. It is also transparent to swap one component runtime for another. This breaks the dependence on large standard software stacks. Each component can run the base system that best suits it, with no worry for what other parts of the system need. Developers are not locked into one OS, but rather can select based on the needs of a component.

8 The Challenges of Fine-grained Cloud Compositions

There are also significant barriers to fine grained composition. The first barrier is the decomposition of systems themselves. It would be nieve to think that systems engineers would rewrite their software to fit this model. Existing software system are complex, and decomposing them as a whole is a tremendous task. However, new systems are built, which could be designed with a component model in mind. Further more, MacroComponents are not an exclusive choice, the parts of a system that could most benefit from the beneficial properties of MacroComponents could be extracted, and the rest of the system could run along side in typical monolithic system. It remains to be seen if the engineering effort is worth the benefits to existing systems.

One issue that is already pertentant in the cloud is that of information security and privacy [16]. Increasing the number of virtual machines and that amount of communication and coordination in cloud systems surely will not help these problems. We hope that general solutions proposed in this space will apply to MacroComponent type systems as well. Also, because of the systems decomposed state, things that might not have otherwise need to be transmitted between virtual machines may have to be. One other practical issue of system decomposition is the overheads of communication between modules and in the creation of components (and their resident virtual machines). To produce a useful system, both these overheads have to be addressed.

Finally, compostion of components in large systems may become a complex task. Some of the features of OSGi like explicit versioning and dependancy declaration help with this, but it is still a problem suffered by all complex systems.

9 Conclusion

Fine grained virtual components show promise as different way of decomposing cloud software systems. There are a miryaid of benefits and disadvantages to building software with a model like this; but, current technologies like OSGi and P2 could help make this approach more feasible. In the future we intend to build a MacroComponent based system and test it in the cloud.

References

1. Matthews, C., Coady, Y.: Virtualized recomposition: Cloudy or clear? In: CLOUD '09: Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, Washington, DC, USA, IEEE Computer Society (2009) 38–43
2. Collier, G., Plassman, D., Pegah, M.: Virtualization's next frontier: security. (2007) 34–36
3. Bellino, J., Hans, C.: Virtual machine or virtual operating system? In: Proceedings of the workshop on virtual computer systems, New York, NY, USA, ACM (1973) 20–29
4. : XADM: How to configure exchange server 5.5 to run on a domain controller that is a global catalog (2008) `http://support.microsoft.com/kb/275127`.

5. Ammons, G., Appavoo, J., Butrico, M., Silva, D.D., Grove, D., Kawachiya, K., Krieger, O., Rosenburg, B., Hensbergen, E.V., Wisniewski, R.W.: Libra: a library operating system for a JVM in a virtualized execution environment. In: VEE '07: Proceedings of the 3rd international conference on Virtual execution environments, New York, NY, USA, ACM (2007) 44–54
6. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauery, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP'03. (2003) 1–14
7. Murray, D.G., Milos, G., Hand, S.: Improving Xen security through disaggregation. In: VEE '08: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, New York, NY, USA, ACM (2008) 151–160
8. Amazon.com Inc: Amazon elastic compute cloud. Website `http://aws.amazon.com/ec2/`.
9. rPath: rBuilder Online. Website (2009) `http://www.rpath.org`.
10. openQRM: openQRM: Open source data-center-management-platform. Website (2009) `http://www.openqrm.com`.
11. : Open Service Gateway Initiative. Website (2009) `http://www.osgi.org`.
12. Rellermeyer, J.S., Duller, M., Alonso, G.: Engineering the cloud from software modules. In: CLOUD '09: Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, Washington, DC, USA, IEEE Computer Society (2009) 32–37
13. : Equinox P2. Website `http://wiki.eclipse.org/Equinox/p2`.
14. Rellermeyer, J.S., Alonso, G., Roscoe, T.: R-osgi: distributed applications through software modularization. In: Middleware '07: Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware, New York, NY, USA, Springer-Verlag New York, Inc. (2007) 1–20
15. : Eclipse PDE. Website (2009) `http://www.eclipse.org/pde/`.
16. Pearson, S.: Taking account of privacy when designing cloud computing services. In: CLOUD '09: Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, Washington, DC, USA, IEEE Computer Society (2009) 44–52