An Architecture for Overlaying Private Clouds on Public Providers

Mark Shtern, Bradley Simmons, Michael Smit, Marin Litoiu
York University, Canada
{mshtern,bsimmons,msmit,mlitoiu}@yorku.ca

***Abstract*—Organizations shifting to a public cloud infrastructure face potential hurdles regarding control and security, and must acquire a new set of best practices regarding developing and deploying to a cloud infrastructure. We propose a reference architecture for a virtual private cloud built on cross-provider on-demand compute instances, with a set of components, services, and algorithms to produce a managed platform that reduces the level of trust required for infrastructure-as-a-service (IaaS) providers, increases control and isolation, improves security and data protection, and allows architects, developers, and operations staff to deploy applications to the cloud using their existing body of knowledge and best practices. Two concrete architectures based on this reference are presented, and a prototype implementation is described and tested.**

***Keywords*-public cloud, nested virtualization**

I. Introduction

Enterprises are adopting cloud computing – particularly utility, on-demand computing resources – to attain perceived benefits such as increased ability to scale, lower total cost of ownership (TCO), and the potential to improve availability. These benefits come at the price of negative properties such as requiring trust in the provider, reducing control and isolation, impacting security and data protection, and constraining applications to be developed or migrated with deference to the unique properties of cloud computing [1], [2], [3].

Our vision is virtual infrastructure that combines the virtues of cloud computing with the environment and desirable properties we expect of private infrastructure. In total, the requirements include security by default (including mitigating attacks on confidentiality, integrity, and availability), scalability and elasticity out of the box, low TCO, functionality whether the provider is trusted or not, increased control over our instances including network configuration, freedom to choose and move among providers, and a simplified process for deploying legacy applications to the cloud without re-training developers with cloud- or provider-specific knowledge. The open research question is if these requirements can be met using the unmodified public cloud currently available, effectively overlaying a private cloud on public resources. Tools that address one or two of these requirements exist, but tend to trade-off another requirement. For example, virtual private clouds with dedicated hardware can address some of the multi-tenancy security and data protection concerns, but increase the TCO. Approaches to ensuring trust between the cloud provider and consumer require hardware modification (e.g. [4]). We provide an integrated solution.

This paper describes a reference architecture and introduces architectures and an implementation that realize this vision. The **AERIE**[1] **reference architecture** describes a set of components, services, and algorithms to produce a managed platform that reduces the level of trust required for infrastructure-as-a-service (IaaS) providers, increases control and isolation, improves security and data protection, and allows architects, developers, and operations staff to deploy applications to the cloud using their existing best practices. The architecture can be deployed on top of existing public cloud providers, but could in the future be better supported by providers.

The contributions of this paper include a novel reference architecture that when implemented addresses known shortcomings with the public cloud without compromising the advantages, while offering a separation of development and operations concerns. Included in this architecture is a key exchange algorithm for verifying an instance on a public cloud has not been tampered with before trusting it with keys and certificates required to access a trusted architecture. We present two concrete architectures based on this reference architecture, and describe a functioning implementation capable of automatically deploying and scaling a three-tier web application to the Amazon public cloud on this virtual infrastructure.

This paper is organized as follows. Section II describes approaches that partially solve the shortcomings of the public cloud. Section III describes our reference architecture in more detail; the advantages and non-functional attributes of this architecture are described in Section IV. We describe two concrete architectures based on our reference architecture (Section V), then describe a functioning prototype that runs on Amazon EC2 in Section VI, then conclude in Section VII.

II. Background

Existing work address some of the concerns with moving to the cloud: migrating applications, assuring trust in the provider, and increasing isolation using nested virtualization. These individual tools do not offer a complete solution.

General cloud migration tasks are covered at length in the software maintenance community, online how-to manuals, and cloud-specific academic conferences. For example, Frey et al. [5] describe an approach for identifying the ways existing software would violate the constraints of a cloud service if deployed to it. Constraint violations are detected in the model and reported to the developer for repair prior to cloud deployment. Babar et al. [6] describe a process for migrating that requires architectural changes, and offer general observations on desirable properties of migrated application. REMICS [7] is an EU FP7 research program focused on

[1]AERIE stands for Automated Encrypted Runtime Image Environments; in English, an aerie is a large nest built high on a cliff, or: a nest in the clouds.

978-3-901882-48-7

migrating legacy applications to the cloud. Their heavyweight model-driven approach is based on the OMG standard Architecture Driven Modernization. Model-driven engineering is used to produce a new system, and this system is validated against the existing system. At present they have not reported an implementation or evaluation of their approach, but expect to ease the process of large and complex migrations. Tran et al. [1] developed a model for estimating migration effort, validated both theoretically and empirically, showing changes were required for 6-10% of the total lines of code for simple three-tier web applications.

There is work on trusted computing platforms for the cloud. The myTrustedCloud (MTC) project [8] proposed a method for attesting the integrity of a VM, the hypervisor, and the storage controller. Terra [9] provides a trusted platform including hardware, the hypervisor, and VMs. Jin et al. [4] describe a hardware component that uses nested memory tables to prevent unapproved access to the physical memory used by a virtual machine, even by the hypervisor. All three approaches require specialized hardware modifications, and MTC requires BIOS modification: in short, full cooperation of the provider is required. Rasmusson et al. [10] propose using such a trusted computing platform but permitting only well-defined, approved probes to be inserted into the machine code of the client to detect malicious behavior. While this may encourage adoption of such platforms, hardware modification and provider cooperation is required.

Nested virtualization refers to the use of virtualization by an already virtualized resources. With two virtualization layers, a hypervisor running on bare-metal manages virtual machines; these virtual machines themselves run hypervisors with their own set of virtual machines. This is particularly useful when the layer 0 hypervisor is controlled by a third-party; a second level of virtualization offers control, isolation, and homogeneity [11], [12]. One example is the Turtles project [13], which modifies the hypervisor to allow multiple hypervisors to run on top of the lowest-level hypervisor. Zhang et al. [14] developed CloudVisor to operate below the hypervisor to improve security and isolation by separating the management of security-sensitive data structures from the hypervisor. Both of these require provider adoption. Kauer et al. [15] propose using nested virtualization to provide defense in depth with multiple security layers. Xen-Blanket [16] allows nested virtualization on public cloud providers running Xen as a hypervisor (which includes Amazon EC2 and Rackspace Cloud), and is intended to enable migration of virtual machines to other providers.

Nested virtualization imposes some overhead depending on the operation. The Turtles project [13] showed overhead of 10.3% and 6.3% on benchmarks that tested CPU, memory, and disk performance on real tasks. Xen-Blanket reports CPU overhead as low as 3% but disk IO overhead as high as 30% [16]. When acquiring resources from a third party, the impact can be mitigated by leasing a very large virtual machine and running multiple VMs on it; the economy of scale allows for the multiple VMs to be over-provisioned with the total cost comparable to leasing multiple smaller VMs.

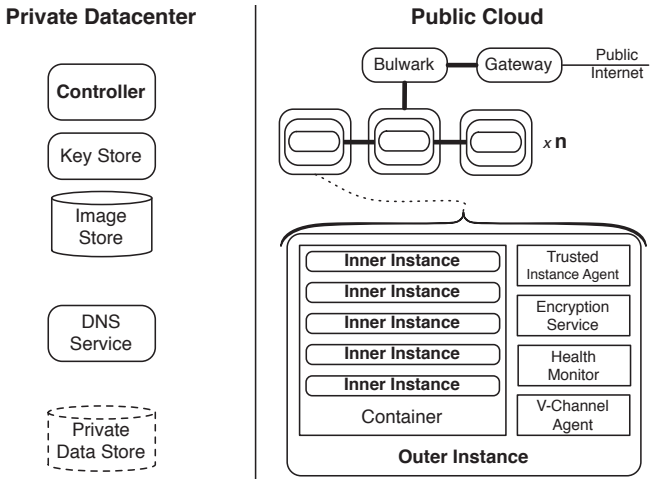

Fig. 1: The logical components of the AERIE reference architecture.

In general, existing approaches to address the challenges of cloud computing are heavy-weight, require provider cooperation, and are one-dimensional, addressing only fragments of our overall vision for a virtual data center.

III. AERIE Reference Architecture

The AERIE reference architecture provides templates, a set of possible components, and shared terminology for more concrete architectures that realize the vision of a virtual datacenter that combines the advantages of the public cloud with the desirable properties of a private data center.

Figure 1 shows an overview of the reference architecture. Several supporting components run in a private datacenter, but the bulk of the architecture is overlaid on public IaaS compute instances using nested virtualization, to preserve the feature of utility-based pricing with low upfront costs. An *outer instance*, based on an *outer image*, includes a *container* that runs one or more *inner* instances. The container may be the Xen hypervisor, Qemu, Linux Containers (LXC)[2], or any other method of isolating the inner instances from the outer instances. The inner instance is started from an encrypted *inner image* stored on the outer instance. Together, the outer and inner instances form a *nested instance*.

The outer instance runs an agent to ensure it has not been modified or compromised. This agent establishes contact with a central controller tier using a novel key exchange algorithm that establishes a secure management channel; these channels are the only links between the private data center and the public cloud. The integrity of the nested instance is maintained by deploying standard security applications to protect the outer instance; if the integrity can no longer be assured, the secure channels are closed and any keys and certificates revoked.

The reference architecture specifies that the inner instances be connected together using virtual channels in whatever

[2]Respectively, http://www.xen.org/, http://wiki.qemu.org/Main_Page, http://lxc.sourceforge.net/

network topologies are required for the applications being deployed (although a particular topology is shown in the figure for illustration, no specific topology is required). Traffic from the public internet is introduced to the virtual channels only after passing through security bulwark. The inner instances accept connections only from the virtual channel, and only from instances they must communicate with per the specified topology. The isolation and encryption of network traffic allows a topology of nested instances to be deployed over multiple availability zones, data centers, or even providers. A load-balancing DNS service capable of detecting inaccessible hosts forms part of an availability solution.

We describe each component of this architecture in more detail, considering two major classes of components: those that are included on the **outer instance** (§III-A) and the **supporting components** that manage the outer instances and the supporting network nodes (§III-B).

A. Outer Instance

The outer instance is launched by the controller from an existing image using the API provided by the cloud provider. The image includes an encrypted image[3] that will be used by the container to launch the inner instances. Once booted, a set of components provide services for protecting and managing the inner instances. Each component is described below.

Trusted Instance Agent (TIA): The TIA manages the outer instance roles in establishing and maintaining a two-party trust relationship. The TIA starts early in the boot sequence of the outer instance and conducts a key exchange with the controller to establish a trusted channel using the following novel algorithm[4]. When an instance is launched on a public cloud, the provider establishes a channel for communication between the requester and in the instance. This channel is secured using shared information, for example a root password (Rackspace) or a public-private keypair (Amazon)[5]. This channel is considered untrusted. The TIA connects to the controller using an HTTPS connection, verifying the controller's certificate matches the one stored in the image. It acquires an executable program from the controller to oversee a challenge-response authentication that verifies the files on the instance are unmodified (using standard techniques like file signatures), then using the shared information intended to provide a secure channel to instead sign the response sent to the controller. Once the controller verifies the integrity of the instance, it sends any keys or certificates required decrypt the inner image and join the topology of nested instances. The TIA stores these in memory, providing them to other components as required. The keys are used to establish a secure management channel between the controller and the instance.

To maintain the integrity of the outer instance, the TIA employs a selection of sub-components to prevent and detect threats to integrity, for example anti-virus applications, firewalls, and host intrusion detection systems (HIDS). The TIA also detects any separation from the controller. If violations are detected and not mitigated, the TIA must terminate the virtual channels, unmount the encrypted partition, stop the V-Channel Agent and Encryption Service, and delete any keys in memory. The TIA may attempt to re-connect with the Controller and re-establish the trust protocol. The Controller may decide if it needs or wants to re-establish the relationship or simply start a new instance in its place and revoke the keys and certificates issued to the now-untrusted instance.

Encryption Service: The encryption service[6] uses keys acquired from the TIA to mount the encrypted inner image. Files accessed on this encrypted partition are decrypted on-demand as they are accessed; files written to this partition are encrypted before being written to disk. This encryption/decryption is completely hidden from the application reading/writing the files. The decrypted data and the keys are stored only in memory[7]. TrueCrypt[8] is capable of providing this service.

Container: The container component launches the inner instances from the encrypted partition on virtualized hardware; the level of isolation required will depend on the implementer. Adding a second virtualization level on top of the virtualization already employed by the public cloud provider is not a trivial step, unless the provider explicitly supports this (as suggested in [13]). We have tested this approach on Amazon EC2 with three containers: the Xen hypervisor using Xen-Blanket [16], Qemu, and Linux Containers (LXC).

Virtual Channel (V-Channel) Agent: This agent establishes a secure virtual channel connecting the inner instances together in a topology matching the deployment pattern specified by the deployer. The virtual channel must ensure that unencrypted traffic does not traverse any network external to the outer instance, and that only authenticated inner instances that need to communicate with each other are connected. For encryption, it may use keys or certificates acquired from the TIA. Arbitrarily complex networks can be created, for example connecting multiple network interfaces within the inner instance to different virtual channels. A V-Channel can be implemented using a virtual private network (VPN) built using OpenVPN[9] or other VPN implementations.

Health Monitor: This monitor is responsible for assessing the health of the inner instances. Health reports are signed with the keys obtained from the TIA and ultimately sent to the controller, perhaps with intermediate components implementing (for example) a hierarchical distributed monitoring system, or one already used within the deploying organization.

B. Supporting Components

Image Store: The image store holds outer images for deployment to public cloud providers. Optionally the image store

[3]Alternatively, it could include instructions for obtaining such an image at run-time; for example, when running on Amazon EC2, it could mount an EBS volume with such an image.

[4]This algorithm could also be used independently.

[5]To avoid replay attacks, it is important that these passwords and keys be used only once.

[6]More accurately an encryption/decryption service.

[7]Unencrypted swap space is a threat; swap should be turned off or managed by the encryption service for both the outer instance and inner instance.

[8]http://www.truecrypt.org/

[9]http://openvpn.net/

can provide image signature services to assist the controller and TIA authenticate outer instances. OpenStack Glance[10] is an application capable of providing this service at scale.

Controller: The controller is responsible for the overall management of the components. Automation is specified to enable the deployment and adaptive management of applications to a topology (or multiple topologies) of nested instances without extra effort. The controller bears primary responsibility for ensuring the advantages of nested instances without compromising the advantages provided by cloud computing.

Optionally, a cloud broker [17] or a cloud metadata service [18] may be used to identify the appropriate public cloud provider. It contacts the image store to either commence uploading the required images to the public cloud provider or obtain the identifiers of already uploaded images. It launches outer instances then waits to be contacted by the TIA (as described in §III-A) to verify the integrity of the deployed instance. Once the outer instance is verified, the inner instance is launched by the container and the Controller tells the V-Channel Agent what connections to establish and what routing is required to add this instance to the topology. The controller may be capable of automating the deployment of the components required for this architecture and managing the elasticity policy of the topology in line with business objectives [19]. Actions available include adding and removing inner instances and outer instances, configuration changes, and migrating inner instances from one outer instance to another.

Key Store: The Key Store is a logical component used by the the Controller to store and organize the keys and certificates used to authenticate the deployed instances and connect to them on the secure management channel. The Key Store could be hosted on the same machine as the Controller in a single-Controller architecture.

Gateway: The Gateway (or gateways) is the only path public internet traffic can take to reach the application. It accepts incoming traffic and passes it to the application over the virtual channel, acting as a reverse proxy service. (Traffic will be monitored by the Bulwark before it arrives at the application). Apache[11] can be used to implement this component.

Area Bulwark: The Bulwark component inspects traffic on the virtual channel, discarding malicious traffic and preventing attacks. Monitoring can occur at both the network level and the application level. A standard application like Snort[12] can be used at the network level. The application level can be monitored using ModSecurity[13] and similar or an application-level distributed denial-of-service detection [20].

DNS Service: This component is a load-balanced domain name service capable of responding to resolution requests with the IP address of multiple Gateways (round-robin), choosing among the available Gateways based on a) their availability and b) their load. The Controller is responsible for maintaining the list of available Gateways. The presence of this service allows the Controller to launch nested instances on multiple public clouds or in multiple availability zones while evenly distributing requests among them. If an availability zone or an entire data center become unreachable, the DNS service will redirect. This service can be in a private data center for complete control, but could also be managed by a service provider. Such a service has been implemented successfully [21].

Private Data Store: This optional component allows sensitive data to be stored in a private data center while the application runs in the public cloud, communicating securely over the virtual channel to read or write data.

[10]http://openstack.org/projects/image-service/
[11]http://httpd.apache.org/
[12]http://www.snort.org
[13]http://www.modsecurity.org/

IV. AERIE Characteristics

The AERIE architecture described in §III allows the deployment of an application to a topology of nested instances, but more importantly it achieves several non-functional qualities, as described in the following sections.

A. Mitigating Security Concerns

Deploying to a public cloud introduces security concerns. Van Dijk et al. [3] assert that confidentiality cannot be guaranteed on an architecture like cloud computing where the hardware is not controlled by the user. We recognize this limitation; while we cannot guarantee security, the reference architecture improves the mitigation of security threats.

Attacks on the inner instance are more challenging for outside attackers, co-located instances, and even attackers that compromise the physical host machine [13], [14]. We describe how the architecture improves isolation, network security, confidentiality, integrity, and availability.

a) Isolation: A key principle of AERIE is improved isolation. Better isolation improves the security of an overall solution; previous work [8], [12] showed security improvement and other benefits from using nested virtualization. To increase the isolation between the outer and inner instances, the inner instance could run a different operating system. In addition to providing additional obfuscation, this limits the risk posed by vulnerabilities in a single operating system.

b) Network Security: Connecting each nested instance over a virtual channel allows more complex network models, and enables both automatically routing all traffic through the Bulwark and encrypting all traffic between the application and the database that might not have been encrypted when deployed to a private data center. Configuring the Firewall to allow only traffic on the virtual channel to reach the inner instance on defined ports ensures that services the application developer accidentally leaves running are not accessible or exploitable, reducing the attackable surface of the application. Using a separate Bulwark reduces the reliance on the provider; even without application-specific configuration, common security threats can be mitigated.

The clear separation between the infrastructure layer and the application layer simplifies configuration of the HIDS and the Firewall; rather than managing all of the running processes and open ports required by a large application, the HIDS can block all non-system processes except the short list of architecture components and allow only the virtual channel connection.

c) Confidentiality: The threat to data confidentiality from public cloud deployments can be mitigated by storing sensitive data in a private cloud, accessible only by the virtual channel already in place for the deployed application. Slightly less sensitive data may be stored on an inner instance, as they are launched from an encrypted partition. However, in-memory caching by the DBMS could still pose a threat if an attacker is able to dump the memory of the physical host. The threat of exposing information in log files or core dumps is mitigated because these files are created on the encrypted partition.

d) Integrity: The images used for the outer instances are created in a trusted environment and uploaded to the cloud providers. The TIA is included on all images to verify the integrity of a started instance. Our approach to establishing trust between an instance and the controller ensures that only an instance running an unmodified image is given the keys and certificates required to join the topology. This trust-creation sequence, and the service that deletes keys if it detects the instance is interfered with, helps protect the outer instance.

The Controller uses SSL for platform management and monitoring. To avoid a possible Man-in-the-Middle attack, certificates must be verified. All outer instances are issued unique certificates; when the Controller loses trust in an outer instance, that certificate is revoked. The *Controller* verifies the host fingerprint before establishing an SSL connection.

e) Availability: We consider both availability of applications deployed to the infrastructure, and of the infrastructure itself. Application availability in the event of provider failure is enabled by deploying to multiple public cloud datacentres and using round-robin DNS to load balance. The Bulwark component mitigates availability problems in the event of a denial of service attack. The availability of the architecture components themselves is assured by protecting the in-cloud supporting components the same the application itself is protected, and deploying the off-cloud supporting components to high availability clusters that are distributed geographically with dual independent links to the public internet.

B. Mitigating Migration and Control Concerns

a) Development effort: Rather than migrating an application to the cloud, we provide an infrastructure that allows an application to be deployed almost as-is. By addressing security and other non-functional requirements in infrastructure, AERIE enables applications to be deployed without undergoing cloud security updates and audits, while allowing applications to leverage public cloud features like availability and utility-based resources.

The architecture itself was designed to leverage existing tools as components of an overall solution. The TIA and the Controller are the two components requiring development effort; we provide reference open-source implementations.

b) Separation of concerns: By isolating the inner instance (for the application) from the outer instance (for the infrastructure/platform) using nested virtualization, we separate two traditionally separate areas, application development and IT operations. The AERIE architecture neatly bisects the development / operations roles, placing responsibility for the inner image with developers and the outer image with IT operations staff. While communication is still necessary, operations decisions will not impact developers.

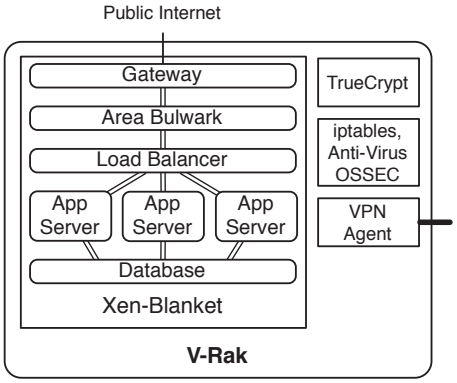

Fig. 2: V-Rak is a concrete architecture based on the AERIE reference architecture.

c) Provider independence: AERIE architectures provide an abstraction layer between the application and the cloud provider, reducing provider lock-in. Depending on the requirements of the container used, inner images once created can be deployed to outer instances on other cloud providers without modification (a task described in more detail in [16]). To achieve this, the controller should be implemented using a cloud broker (e.g. [17]) or abstraction layer (e.g. Deltacloud[14]).

d) Scalability: An application can be scaled out by launching more nested instances and updating the Gateway and Virtual Channels to add the new node to the list of nodes that receive inbound traffic. Application-specific configuration required to scale up may be specified in the topology description file. The Controller component can implement adaptive management via pluggable elasticity policies, as in our implementation. The architecture itself scales from one managed inner instance to thousands. Multiple controller nodes can operate together, splitting management tasks by zone. The existing tools used in the implementation are designed to scale.

V. Architectures Based on AERIE

In this section, we describe two concrete architectures demonstrating different realizations of AERIE.

The **V-Rak** architecture treats an outer instance as a *virtual rack*: the inner instances are virtual machines mounted in this V-Rak. By using two of the inner instances to provide the network supporting services (Gateway and Bulwark), and the remainder to deploy a three-tiered web application, we create a self-contained virtual-appliance that can be easily started on a public cloud provider[15]. If a single V-Rak is sufficient to meet demand, there is no need for a controller, trust agent, key store, or CA as only a single instance is launched. The outer instance is secured using a firewall (iptables) and a

[14] http://deltacloud.apache.org/

[15] A very large instance is required.

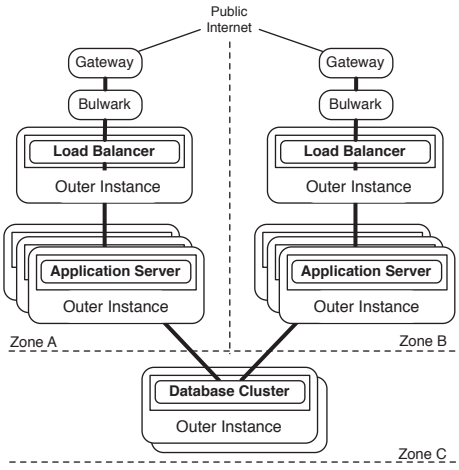

Fig. 3: Wrapt is a concrete architecture based on the AERIE reference architecture.

host intrusion detection application (e.g., OSSEC[16]). All public internet traffic enters through the Gateway; all traffic among the inner instances transits over virtual interfaces entirely isolated from the external network, so no virtual channels are required to communicate. A VPN agent allows connections with other virtual appliances if needed. The App Server inner instances run from encrypted partitions, managed by TrueCrypt. Availability can be achieved by launching multiple virtual appliances on different providers or availability zones and using load-balanced DNS to balance traffic among them.

The **Wrapt** architecture wraps each single inner instance in an outer instance running Qemu. Each outer instance runs the supporting services as listed in the AERIE reference. It is designed for J2EE applications; a set of web clusters are deployed across different zones (which can be different providers, different data centers, etc.). Requests from the public Internet are routed among the web clusters using a load-balancing DNS service. Requests are received by a Gateway and forwarded to one of a set of application servers over a VPN-based virtual channel, passing through a Bulwark including intrusion detection systems and application firewalls. The database runs in a separate zone, or potentially a private data center. This architecture can be scaled at any tier: gateway, load balancer, application server, or database. The full suite of supporting services – Key Store, Controller, DNS server – is required. The inner instance uses all of the outer image resources that are available, i.e. that are not needed to run the supporting services or the container. Availability can be achieved by running outer instances in different zones or with different providers.

VI. Implementation Details

As a proof-of-concept, we implemented the Wrapt architecture (Fig.3). We deployed a J2EE application we have worked with previously [20] to Amazon EC2, distributed across availability zones. The supporting services and a database were deployed to a private domain while deploying two web clusters to separate availability zones. The web cluster Gateway was implemented using a front-end reverse proxy; incoming network traffic is inspected by a Bulwark (realized using Snort and ModSecurity) before reaching a back-end load balancer (an Apache HTTP Server). The load balancer distributes requests among application server (Tomcat) instances; the application tier scales adaptively in response to workload.

To facilitate this application topology deployment, an Amazon Machine Image was created (Ubuntu 11.10 64 bit). The following software packages were used for the various tools required to provide our solution: QEMU emulator version 0.14.1, TrueCrypt 7.1a, OpenVPN 2.2.0, OSSEC version 2.6, lbcd version 3.30, Snort version 2.8.5.2, and Apache modules mod-security 2.6.0 (using default/standard community rules) and mod-evasive 1.10.1.

A guest QEMU image was created and stored on an encrypted partition. The operating system for the guest image was Ubuntu 11.10 64 bit and the application-specific packages were installed on it: Apache version 2.2.20, Tomcat version 6.32 and mysql 5.1.61.

To establish trust when initializing instances we used the SSH public key of the outer instance and RSA. All inter-node communications used VPN channels. The set of supporting services (e.g., Certificate Authority, https, HIDS, DNS) were deployed to the private domain. An automation solution [22] provided the ability to both deploy and manage the application. The controller extends a manager component of this framework augmented to handle the complexities of deploying a nested instance environment. Management of the application's elasticity policy was accomplished through use of a model. Availability across the set of zones was achieved by adopting an existing DNS-based solution [21].

A series of m1.medium Amazon instances were deployed and the infrastructure initiated. Tests were performed to verify the functional correctness and availability of the application, the manageability of the infrastructure and correct functioning of the deployed security measures.

VII. Conclusion

The public cloud offers great potential in terms of scalability, economies of scale and elasticity; however, there are limitations with current approaches in terms of security vulnerabilities and the need to migrate to a cloud programming model. While many providers offer cloud services, each adopts their own programming model and best practices. One approach does not fit all when it comes to computing infrastructure. The AERIE reference architecture enables the creation of virtual infrastructure that can be molded to fit the requirements of an enterprise, built on top of the public cloud with all its benefits. The collective solution leverages existing tools and algorithms in a particular configuration to restore the control, overlaying a private cloud on public cloud resources. We have realized this reference architecture through the design of two concrete architectures, V-Rak and Wrapt, and presented a prototype implementation of the latter.

[16] http://www.ossec.net

ACKNOWLEDGMENT

This research was supported by IBM Centres for Advanced Studies (CAS), the Natural Sciences and Engineering Research Council of Canada (NSERC) including through the SAVI project, the Ontario Centres of Excellence (OCE), and Amazon Web Services (AWS). Special thanks to Cornel Barna for use of his management framework.

REFERENCES

[1] V. T. K. Tran, K. Lee, A. Fekete, A. Liu, and J. Keung, "Size estimation of cloud migration projects with cloud migration point (CMP)," in *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*. Washington, DC, USA: IEEE Computer Society, 2011, pp. 265–274.

[2] W. A. Jansen, "Cloud hooks: Security and privacy issues in cloud computing," in *2011 44th Hawaii International Conference on System Sciences (HICSS)*, 2011, pp. 1–10.

[3] M. Van Dijk and A. Juels, "On the impossibility of cryptography alone for privacy-preserving cloud computing," in *Proceedings of the 5th USENIX conference on Hot topics in security*. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–8.

[4] S. Jin, J. Ahn, S. Cha, and J. Huh, "Architectural support for secure virtualization under a vulnerable hypervisor," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44 '11. New York, NY, USA: ACM, 2011, pp. 272–283.

[5] S. Frey and W. Hasselbring, "An extensible architecture for detecting violations of a cloud environment's constraints during legacy software system migration," in *15th European Conference on Software Maintenance and Reengineering (CSMR)*, 2011, pp. 269–278.

[6] M. A. Babar and M. A. Chauhan, "A tale of migration to cloud computing for sharing experiences and observations," in *The 2nd International Workshop on Software Engineering for Cloud Computing*, ser. SECLOUD '11. New York, NY, USA: ACM, 2011, pp. 50–56.

[7] P. Mohagheghi and T. Sæther, "Software engineering challenges for migration to the service cloud paradigm: Ongoing work in the REMICS project," in *World Congress on Services (SERVICES)*, 2011, pp. 507–514.

[8] D. Wallom, M. Turilli, A. Martin, A. Raun, G. Taylor, N. Hargreaves, and A. McMoran, "myTrustedCloud: Trusted cloud infrastructure for security-critical computation and data managment," in *IEEE International Conference on Cloud Computing Technology and Science*. Los Alamitos, CA, USA: IEEE Computer Society, 2011, pp. 247–254.

[9] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: a virtual machine-based platform for trusted computing," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 193–206, 2003.

[10] L. Rasmusson and M. Aslam, "Protecting private data in the cloud," in *2nd International Conference on Cloud Computing and Services Science (CLOSER)*, Porto, Portugal, 2012, pp. 18–21 April.

[11] D. Williams, E. Elnikety, M. Eldehiry, H. Jamjoom, H. Huang, and H. Weatherspoon, "Unshackle the cloud!" in *3rd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)s*, Portland, OR, June 2011.

[12] Z. Pan, Q. He, W. Jiang, Y. Chen, and Y. Dong, "Nestcloud: Towards practical nested virtualization," in *Proc. Int Cloud and Service Computing (CSC) Conf*, 2011, pp. 321–329.

[13] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour, "The turtles project: design and implementation of nested virtualization," in *The 9th USENIX conference on Operating systems design and implementation*. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6.

[14] F. Zhang, J. Chen, H. Chen, and B. Zang, "Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization," in *Twenty-Third ACM Symposium on Operating Systems Principles*. New York, NY, USA: ACM, 2011, pp. 203–216.

[15] B. Kauer, P. Verissimo, and A. Bessani, "Recursive virtual machines for advanced security mechanisms," in *41st International Conference on Dependable Systems and Networks Workshops*. Washington, DC, USA: IEEE Computer Society, 2011, pp. 117–122.

[16] D. Williams, H. Jamjoom, and H. Weatherspoon, "The Xen-Blanket: Virtualize once, run everywhere," in *7th ACM European Conference on Computer Systems (Eurosys)*, Bern, Switzerland, April 2012.

[17] P. Pawluk, B. Simmons, M. Smit, M. Litoiu, and S. Mankovski, "Introducing STRATOS: A cloud broker service," in *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, 2012, pp. 891 –898.

[18] M. Smit, P. Pawluk, B. Simmons, and M. Litoiu, "A web service for cloud metadata," in *IEEE Congress on Services*. Los Alamitos, CA, USA: IEEE Computer Society, 2012, pp. 24–29.

[19] H. Ghanbari, B. Simmons, M. Litoiu, and G. Iszlai, "Exploring alternative approaches to implement an elasticity policy," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, 2011, pp. 716–723.

[20] C. Barna, M. Shtern, M. Smit, V. Tzerpos, and M. Litoiu, "Model-based adaptive DoS attack mitigation," in *7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2012.

[21] R. J. Schemers, III, "lbnamed: A load balancing name server in Perl," in *Proceedings of the 9th USENIX conference on System administration*. Berkeley, CA, USA: USENIX Association, 1995, pp. 1–12.

[22] C. Barna, B. Simmons, M. Litoiu, and G. Iszlai, "Multi-model adaptive cloud environments (MACE)," November 2011, http://www.ceraslabs.com/projects/management-services-for-cloud-computing.