Overcoming Barriers for Ubiquitous User-Centric Healthcare Services

Alex Palesandro
Orange Labs

Chirine Ghedira Guegan
Université de Lyon

Marc Lacoste
Orange Labs

Nadia Bennani
Université de Lyon

The Orchestration for Beyond Intercloud Security (Orbits) architecture enables flexible and legacy intercloud application deployment for mobile remote healing, while providing a homogeneous service abstraction across multiple clouds.

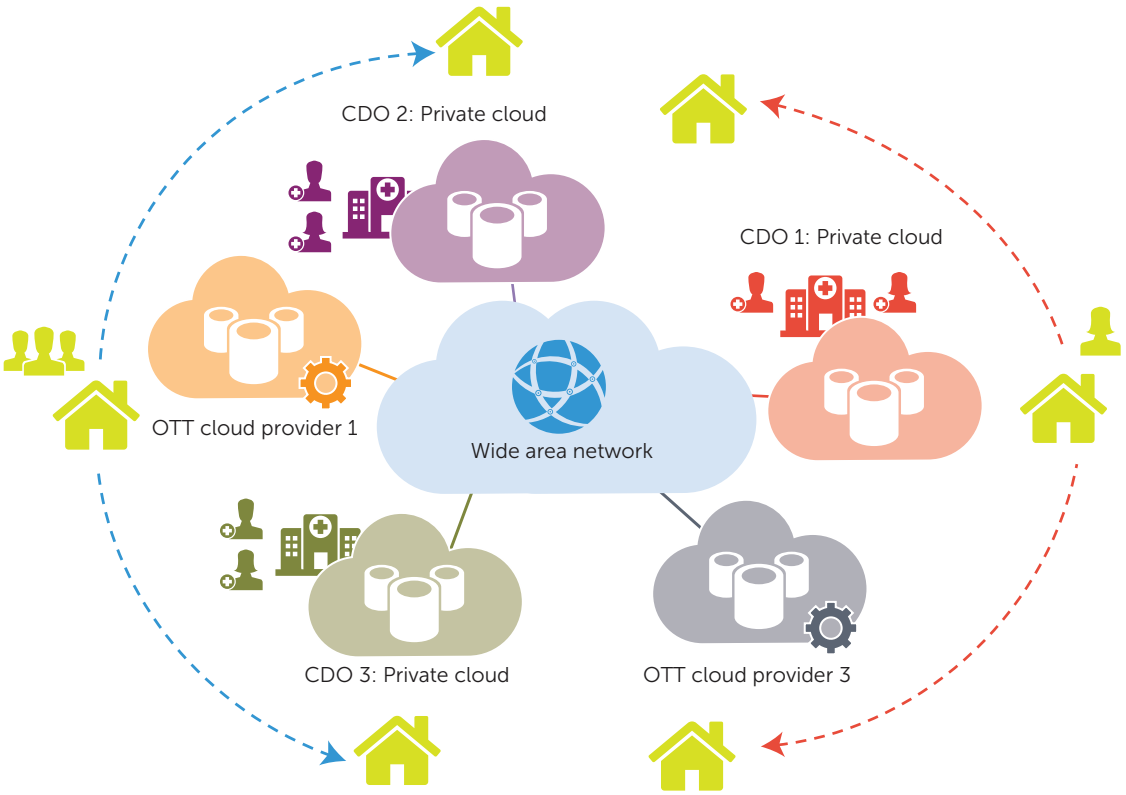

FIGURE 1. Follow-me use case. Actors in this scenario include care delivery organization (CDO), private cloud, and over-the-top cloud (OTT) providers.

Cloud home healthcare systems represent a widely investigated research area.[1] These systems are designed for a wide spectrum of healthcare applications, from simple electronic health record (EHR) consultation to remote monitoring and assisted surgery. Key requirements for such applications are geographical restrictions on the hosting of applications and data, usually imposed by laws; stringent high-availability and QoS constraints (99.99 or 99.999 percent of availability time per year); and dependency on a homogeneous set of system security services from different cloud public providers. In other words, applications should be accessible anywhere, anytime, with acceptable performance and security.

Current home-based scenarios are limited to patients who might leverage the service on premise, relying on the same practitioner or care delivery organization (CDO). Moreover, current systems don't support "follow-me" scenarios, where traveling patients might require treatment away from their usual residence, potentially relying on new practitioners and CDOs (see Figure 1).

Single provider clouds can't meet these challenges. First, data processing has strict requirements in terms of location awareness. In addition, single-provider availability guarantees might not be sufficient in medical environments. Quality of service (QoS) is also impacted by latency, increasing with distance between service users (such as patients and doctors) and the datacenter. Finally, cloud providers must be trustworthy given the privacy issues related to medical data. To overcome such limitations, healthcare services should rely on multiple cloud providers. A multiprovider approach brings both benefits, in terms of geolocation, availability, and QoS; and challenges, such as the need for consistent quality of protection

RELATED WORK IN E-HEALTH AND INTERCLOUD ARCHITECTURES

E-health cloud opportunities and corresponding challenges are widely discussed in literature. Assad Abbas and Samee Khan[1] and Eman AbuKhuosa and his colleagues[2] discuss privacy issues in treating sensitive healthcare data in public cloud infrastructures, including threats, corresponding requirements, and different proposals for secure and private data treatment. Orbits is orthogonal to these proposals, because it facilitates multiprovider adoption and supports adding recurring components (such as encryption proxies) to the overcloud service model and deploying them on multiple providers. Other work leverages the multicloud as a secure and resilient infrastructure for performing multiparty computation and offloading mobile healthcare applications.[3,4] Both use cases are compatible with Orbits. In particular, the overall visibility of application orchestration logic could simplify the deployment of sophisticated policies in job distribution across multiple providers.

Interconnection of multiple provider resources promises important benefits compared to single clouds. These benefits include finer-grained distribution of resources across multiple countries, improving quality of service; unified abstraction for resource access; and cost savings, optimizing expenditures through dynamic price comparisons between providers (for example, for Amazon Web Services Spot instances). Several surveys on interconnected clouds identify two main types of architectures.[5]

Provider-Centric Architectures

In this federation-oriented approach, providers mutualize their resources, agreeing on a common standard to cooperate.[6] Resource federation enables single providers to better support peak demand or maintenance operations. This approach presents two limitations: providers are typically competitors, and often aren't interested in cooperating; and different technological choices on their infrastructure may dramatically reduce interoperability among them.[7] However, the evolution of the cloud market, where a few major players control the largest part of market share shows that it's difficult for customers to cross provider barriers.

Client-Centric Architectures

Client-centric approaches require limited provider intervention.[7,8] The client-centric model breaks the general limitation of absence of a standard, since the burden of the interoperability is moved from provider to customer/third party. The architectures typically used are either brokering or infrastructure as a service

(QoP) across providers. The multiple provider model also adds significant complexity. The impossibility of simply and practically leveraging multicloud benefits prevents many applications from relying on multiprovider infrastructure-as-a-service (IaaS) models.

Therefore, a multiprovider system must provide for flexible provisioning, where the application logic influences resource allocation in the multicloud; and must support interoperability. The multicloud should provide infrastructure homogeneity from security and resource abstraction standpoints across multiple sites. Infrastructure homogeneity allows each provider to use the same security services to protect application execution.

The Orchestration for Beyond Intercloud Security (Orbits) architecture addresses these needs, providing simultaneous and flexible application provisioning across multiple providers, as well as a homogeneous service abstraction across multiple clouds enforced at the IaaS level.

Orbits Multicloud Architecture

Healthcare use cases typically embrace a wide range of actors (patients, pharmacists, CDO administrators, doctors, and so on) and different classes of devices. In addition, service developers and operators, who are responsible for building applications and delivering services, represent technical actors in our scenarios.

Hence, we consider two classes of service. The first is applications deployed by CDOs and other institutions that are typically shared across multiple actors and hosted inside private clouds or scaled

(IaaS) compatibility layers.[7,8] Brokering approaches offload multiprovider orchestration, agreeing with a broker on the desired service-level agreements (SLAs) and associated costs. Compatibility layers typically rely on a client-controlled virtualization layer to escape vendor lock-in through an interoperable layer. Such techniques showed fair performance and consolidation improvements compared to traditional cloud deployment. However, how they'll handle flexible provisioning of applications is unclear. Orbits implements an IaaS compatibility layer-based approach, in addition to providing multicloud flexible-provisioning mechanisms.

Meanwhile, application architectures have evolved toward more modularity in deployment, reducing time between development and delivery.

Microservices Frameworks

The rise of lightweight virtualization (such as Docker containers) is changing the way cloud applications are developed and deployed. Revisiting the service-oriented architecture (SOA) paradigm, monolithic applications are componentized into cooperating microservices run inside lightweight containers (for example, Google Kubernetes and Apache Marathon). However, with multiple providers, such frameworks don't consider the homogeneity of the infrastructure services they're leveraging (for example, intrusion detection systems or firewall-as-a-service for security).

References

1. A. Abbas and S.U. Khan, "A Review on the State-of-the-Art Privacy-Preserving Approaches in the E-Health Clouds," *IEEE J. Biomedical and Health Informatics*, vol. 18, no. 4, 2014, pp. 1431–1441.
2. 2. E. AbuKhousa, N. Mohamed, and J. Al-Jaroodi, "E-Health Cloud: Opportunities and Challenges," *Future Internet*, vol. 4, no. 3, 2012, pp. 621–645.
3. H. Wu, Q. Wang, and K. Wolter, "Mobile Healthcare Systems with Multi-Cloud Offloading," *Proc. IEEE 14th Int'l Conf. Mobile Data Management*, vol. 2, 2013, pp. 188–193.
4. T. Ermakova and B. Fabian, "Secret Sharing for Health Data in Multi-Provider Clouds," *Proc. IEEE 15th Conf. Business Informatics*, 2013, pp. 93–100.
5. N. Grozev and R. Buyya, "Inter-cloud Architectures and Application Brokering: Taxonomy and Survey," *Software: Practice and Experience*, vol. 44, no. 3, 2014, pp 369–390.
6. R. Buyya, R. Ranjan, and R. Calheiros, "InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services," *Algorithms and Architectures for Parallel Processing*, LNCS 6081, Springer, 2010, pp. 13–31.
7. D. Williams et al., "The Xen-Blanket: Virtualize Once, Run Everywhere," *Proc. 7th ACM European Conf. Computer Systems* (EuroSys12), 2012, pp. 113–126.
8. K. Razavi et al., "Kangaroo: A Tenant-Centric Software-Defined Cloud Infrastructure," *Proc. IEEE Int'l Conf. Cloud Eng.*, 2015, pp. 106–115.

out to public clouds.[1,2] This class includes EHR consultation for patients and prescription management for doctors or institutions. The other class of services is patient-oriented applications, which typically produce or analyze personal health records (for example, drug therapy self-assessment questionnaires, periodic self-treatments, and epidemiological studies). Such patient-oriented applications might require downloading and uploading data to CDOs or designing complex interconnections among services.[3] Deployed services usually leverage a three-tier application structure with SQL/NoSQL databases, application servers, and front-ends on top of infrastructure abstractions (virtual machines [VMs], object/block storage, and virtual networking) supported by the cloud provider. Given the heterogeneity of actors and applications, each tier is usually split into cooperating subcomponents (microservices) and services, following the service-oriented architecture (SOA) approach.

We consider a simpler scenario in which patients move among locations and thus need to perform telemedicine operations (such as remote treatment, periodic self-treatment and monitoring, and (EHR) access) using mobile devices, while CDO services are geographically fixed in the CDOs' private clouds. The application orchestration logic can retrieve the actual geolocations of the services and patients through the front-end application and device capabilities (such as GPS).

Cloud customers rely on cloud providers not only for low-level resources (compute, networking, and

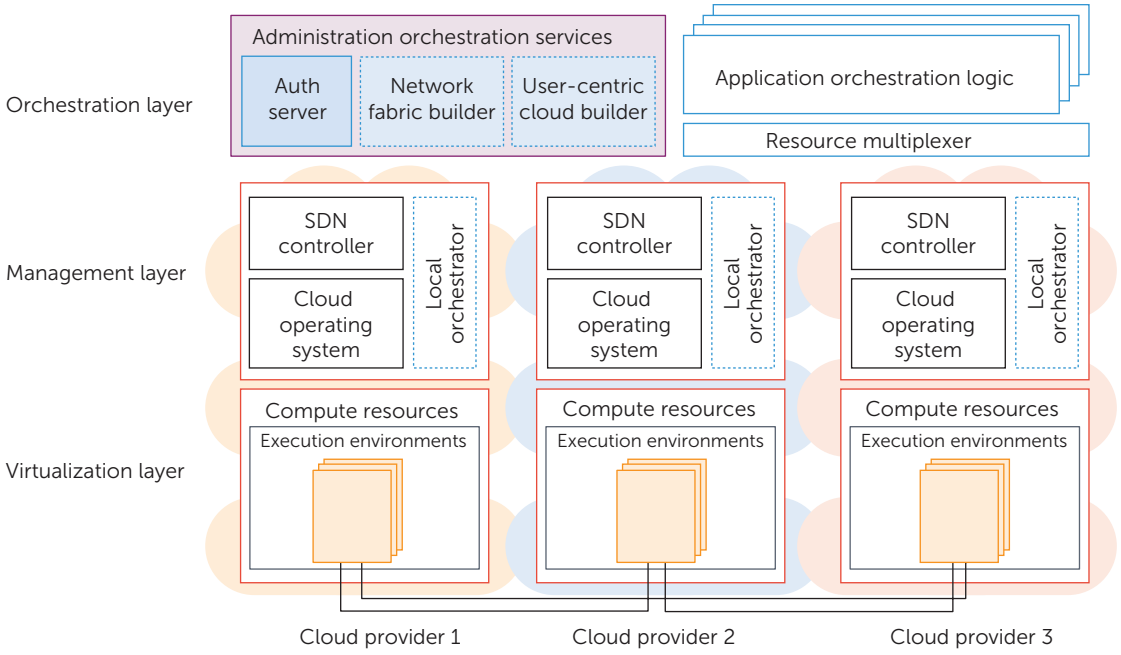

FIGURE 2. The Orbits architecture. Management and virtualization instances are replicated through different providers, creating the "overclouds."

storage) but also for high-level services, such as database management systems as a service (DBMSaaS) or load balancer as a service (LBaaS), offloading operational complexity from developers. However, the use of these complementary services creates a de facto lock-in that introduces a strict dependency between cloud customer and provider. In addition, similar services might encourage customers to remain inside the provider realm since transferring data inside the same provider region is free or inexpensive. Therefore, interoperability at the IaaS level can hide the complexity of compatibility layers on different providers. However, the orchestration logic could effectively deploy multiprovider applications, since all requirements can be handled with a precise knowledge of subcomponent interactions, which is possible at the application orchestration level. Obviously, IaaS interoperability might not solve incompatibility issues at the application layer, but they could simplify interoperability by enhancing the orchestration expressiveness of this layer while hiding the underlying complexity.

To address these challenges, Orbits offers both flexible provisioning of microservices-based applications, handling placement, elasticity, and availability; and infrastructure homogeneity so customers can completely control their security appliances. Orbits enables infrastructure deployment to support application requirements (such as peak usage or CSP breach) when and where they occur.

Existing approaches partially meet these requirements (see the related work sidebar). Indeed, overlay-based approaches give users an important degree of control (such as a virtualization layer and security appliances),[4] but lack effective multiprovider orchestration tools. However, brokering-based approaches (for example, RightScale and jClouds) optimize provisioning of application resources without giving users more control over the infrastructure.

To sum up, we model a use-case where a healthcare service is described by

- a microservice-based application with related orchestration logic,
- a minimal threshold of N distinct providers and M regions that they require a priori (such as for availability),
- the set of security services and configurations they want to deploy for QoP requirements, and
- a list of static provider constraints to address geolocation (such as legal country and per-provider minimum availability).

Developers and operators of the healthcare service might consider a cloud service provider (CSP) as trusted or untrusted, adopting an adversary model to deal with security and privacy.

Figure 2 gives an overview of the Orbits architecture's three layered-design:

- The *virtualization layer* executes scheduled jobs, with tradeoffs between performance and isolation among workloads, using security services specified by operators at build time. It provides a homogeneous view of security services to upper layers to meet the QoP requirement.
- The *management layer* oversees resource provisioning on each overlay provider, managing the virtualization layer and the creation of new execution environments. This layer also meets the QoP requirement, focusing not only on application execution, but also on access to resources.
- The *orchestration layer* ensures flexible provisioning across multiple providers required by the use cases. It gives an overall view of the available providers and coordinates application orchestration between provider instances.

Management and virtualization layer services are deployed on each provider inside the multicloud. We refer to those instances as *overclouds*, as they're overlay instances that provide a homogeneous view of resources to the orchestration layer.

Virtualization Layer

The Orbits virtualization layer runs microservices using a provider-agnostic approach. Virtualization is a widely adopted approach to obtain isolated and transparent hardware resource sharing between competing software or systems. Several technologies can be adopted to deploy and run execution environments that generally aren't interoperable.[5]

The virtualization layer should realize interoperability among isolated execution environments across different providers, hiding provider heterogeneity. Technological heterogeneity makes this impossible at the underlay level. The virtualization layer should also be customizable, allowing each operator to deploy its chosen security services and to impose minimal performance overheads.

Two main technological alternatives are available for the virtualization layer.

Nested virtualization is a system architecture in which the guest operating system virtualizes a nested guest.[6] This extra level of virtualization can be executed through nested hardware-assisted full virtualization[6] or paravirtualization over hardware-assisted virtualization.[5] Performance has always been an impeding factor for massive adoption of such techniques. However, some recent work shows more acceptable overhead.[5,6]

Containers are user-space environments on an operating system providing isolation between them and host resources.[7] Resource isolation is achieved using new kernel functionalities (for example, cgroups and Linux namespaces). However, containers still suffer from major isolation concerns due to Linux kernel sharing and achieve weaker isolation than VMs. Recent work has also shown that overlay containers don't significantly degrade performance.[4]

In both cases, microservices composing a complex application will be run inside execution environments provided by the virtualization layer. Nested virtualization and containers offer different tradeoffs in terms of isolation and performance. Stateful applications might need to be migrated without loss of state through live migrations, which is simpler with VMs. With stateless services, a simple respawn on a new infrastructure is better addressed by lightweight containers, which can enhance rescheduling time on new infrastructures when detecting that a patient is moving and requesting service from another location. VMs achieve better isolation and resilience than containers, but have slower performance, and might be a better tradeoff for critical components in terms of service availability.

Thus, developers and/or operators might adapt virtualization to workloads, selectively isolating or aggregating diverse application components. This can be achieved through the management layer API.

Management Layer

For infrastructure homogeneity, Orbits aims not only at virtualization interoperability but also homogeneous resource management across multiple clouds. This implies uniform APIs across providers. Indeed, complete interoperability issues arising from the infrastructure's multiprovider nature could be prevented by security services provided as a service by cloud providers (for example, anti-DDoS and firewalls). Different APIs might require per-provider adaptation; thus, homogeneous resource management is critical to guaranteeing QoP in our use case.

We distinguish two classes of management services for Orbits overclouds.

In *local resource provisioning*, the local cloud operating system and software-defined networking (SDN) controller components are typically in charge of compute, storage, and networking management.

In *relation with orchestration logic* services, the local orchestrator, or *Stratopause* component, is the link between local resource provisioning and application dispatching. It regularly informs the application orchestration framework about available overclouds, for example, resources and cloud attributes (provider, region, and virtualization technologies). When the application orchestration logic schedules a job on a certain Stratopause instance,

the Stratopause communicates with the cloud operating system service to trigger resource allocation to satisfy the allocation requirements demanded by the orchestration layer. The global orchestration logic collects updates from Stratopause instances to reach placement decisions. This instance also collects microservices that dispatch commands to local overlays, which are transmitted to the local cloud operating system to provision resources according to expressed requirements.

The management layer enables the use of equivalent security services on different providers, for example, to fulfill EHR systems security requirements. However, this layer doesn't have the overall vision of all deployed overclouds.

Orchestration Layer

Orchestration is performed at both the infrastructure and application levels.

Infrastructure orchestration. Following the "infrastructure as code" paradigm, a cloud template text description for the overlay infrastructure defines which services are deployed and where. Orchestration covers

- deploying management and virtualization layers on selected providers,
- providing on-demand interconnection between providers, and
- managing identity and access across overlay instances.

Therefore, to address the deployment of overlays on different providers, the user-centric cloud builder component, Mantus customizes the cloud template according to tenant-requested security services, which might include network and system control, management services, and virtualization; selects a subset of cloud providers, compatible with policies expressed by the tenant needs; and instantiates overlay clouds on multiple providers.

Moreover, hosting cloud providers create virtual networks inside each overlay cloud. To create multiprovider connections, a *network fabric builder* component extends local virtual networks across provider barriers. Finally, an overall *authentication and authorization service* transparently manages identity and access across deployed overclouds, for example, by coordinating different authentication services.

The Mantus orchestration component communicates with orchestration providers' APIs (such as OpenStack Heat and Amazon CloudFormation), deploying the overclouds template, which consists of a text-based description of the topology and configuration of hardware resources and software components. Some legislation, such as the General Data Protection Regulation (GDPR), might require technological and organizational settings to protect sensitive data and its processing. The infrastructure-as-code-based security enrichment approach leveraged by Mantus reduces the effort required to provide the same infrastructure security and privacy services across multiple cloud providers.

Application-level orchestration. Whereas the role of infrastructure services is building and maintaining the Orbits multicloud, the application orchestration logic is responsible for flexible provisioning across clouds, which it typically achieves by placing application microservices across providers.

Orchestration frameworks are usually composed of application frameworks and a resource multiplexer (for example, Apache Mesos). Application frameworks are responsible for application deployment on available resources, following developer/operator specifications. The resource multiplexer guarantees fair sharing between frameworks on a pool of resources. In Orbits, we enhance the placement logic of application frameworks, introducing multiprovider awareness of overclouds deployed by Mantus. The overcloud-aware placement leverages Stratopause instances to receive updates about overcloud instance availability and dispatch selected jobs on a given provider.

Essential requirements of healthcare applications, such as confidentiality, data integrity, and anonymity, might leverage the single point of orchestration to effectively decide where to deploy different instances of services, relying on the infrastructure's homogeneity.[8] This runtime control could also allow service operators to easily comply with legislation in terms of data protection and geolocalization.

Experimental Results

We built a proof-of-concept prototype of the basic overlay template cloud based on OpenStack and Mesos (see Figure 3). We leveraged Xen, Linux Containers (LXC), and the Kernel-based Virtual Machine (KVM) as basic virtualization technologies. The management layer is based on OpenStack, which supports those virtualization technologies OpenStack is integrated with an overlay OpenDaylight as the SDN controller. We realized a first implementation of Mantus and Stratopause in a simpler scenario, where a developer can trigger deployment of Orbits on a select number of providers without considering the patient's location; instead, the focus is

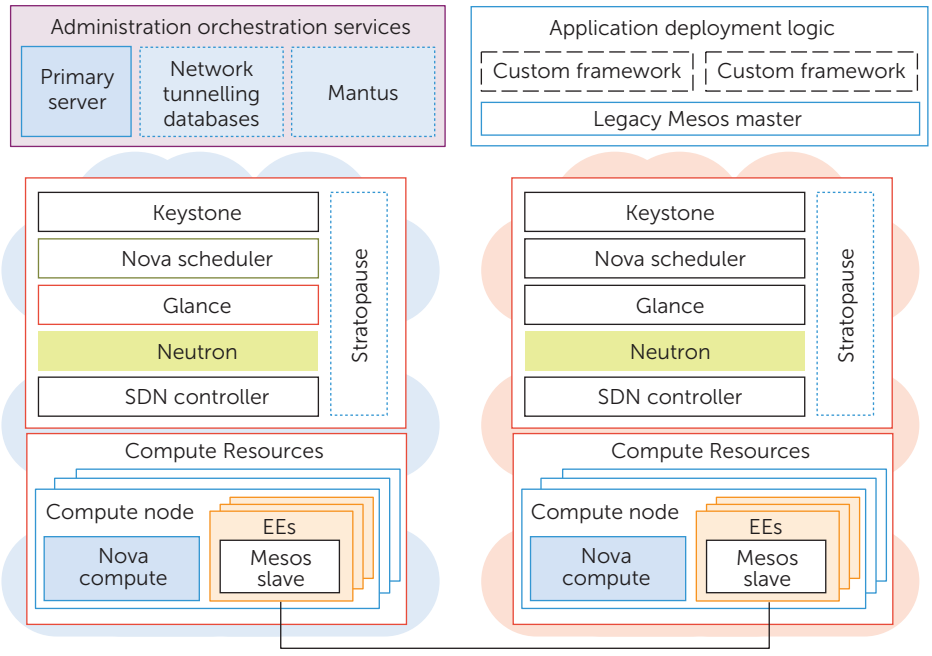

FIGURE 3. Orbits prototype components. Dashed borders indicates newly introduced components, among legacy open-source.

on enriching security services and deploying a uniform infrastructure layer.

The Mantus orchestration workflow proceeds as follows.

In the first step, *service definition*, Mantus uses a code description to automate infrastructure resource provisioning and configuration, which provides benefits in terms of reproducibility and maintenance. Such a description concerns services from management and virtualization layers (such as cloud operating system services, SDN controller, and virtualization nodes).

Next, in the *service enrichment* step, Mantus extends the abstract service description with the list of security services provided as input (see Figure 4). The initial description is then enriched by the addition of selected services from providers (such as access control framework, hardening services, hypervisor appliances, and network middleboxes).

Access control and hardening services could be introduced as new services in the provider-agnostic description. The infrastructure should have network connectivity with control services. Thus, network applications can be described as configuration files to be deployed inside the SDN controller. Similarly, hypervisor appliances can be added to compute nodes. Finally, network middleboxes (for example, firewalls, intrusion detection services, and HTTP accelerators) can be described as extra services, chained together by traffic steering flows.

In parallel to the first two steps, Mantus retrieves a list of available providers and applies a simple *filter and weight* algorithm. We assume that Mantus retrieves a list of provider datacenter regions with predefined and comparable service-level agreements (SLAs), such as minimal availability and location of specific regions.

The next step is *instantiation*. When providers are selected, the provider-agnostic description of services is converted into the provider-specific orchestration language[3] of the selected cloud providers. In the Mantus workflow, provider-agnostic Topology and Orchestration Specification for Cloud Applications (Tosca, www.oasis-open.org/committees/tosca) service descriptions are mapped to per-provider descriptions, such as OpenStack Heat Orchestration Template (HOT, http://docs.openstack.org/developer/heat/template_guide/hot_guide.html) and, in the future, Amazon Web Services CloudFormation (https://aws.amazon.com/cloudformation).

Modeling the base cloud services resulted in 1,103 lines of code (601 lines of Tosca YAML (Yet Another Markup Language) and 502 of BASH [Bourne-Again Shell] configuring scripts). The translation of Tosca to OpenStack Heat plus the instantiation logic for Heat APIs required 868 lines of Python, which represent the specific OpenStack driver code required to port Mantus to a new provider. Supporting OpenStack enables Orbits to support not only private clouds but also several public

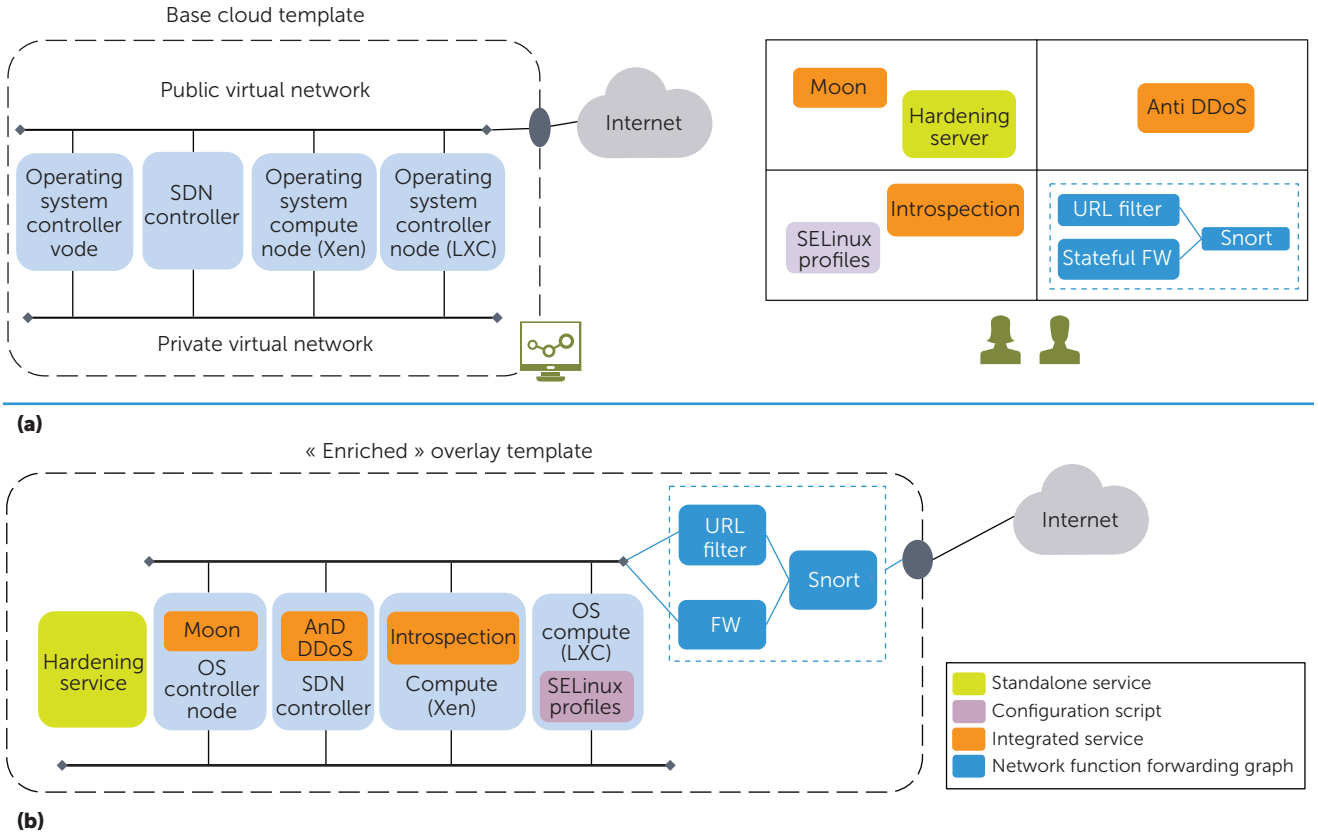

FIGURE 4. Mantus orchestration templates: (a) initial sample overlay template, and (b) services after enrichment process.

Table 1. Meeting healthcare requirements with Orbits.

Healthcare requirements	Orbits component	Feature
Geolocation awareness	Mantus Stratopause	Location and service level agreement (SLA)-based provider prefiltering
High-availability/quality of service	Stratopause	Application-driven flexible orchestration over multiple clouds
Homogeneous quality of performance	Mantus	Homogenous description-based security services deployed across multiple clouds

CSPs leveraging this open source cloud management system.

Table 1 summarizes how Orbits addresses healthcare requirements. The geolocation requirement is addressed through Mantus, which selects acceptable providers according to service SLAs requirements; and through Stratopause, which instructs the application logic with IaaS provider details. For the QoS requirement, Stratopause notifies the application orchestration logic to satisfy desired availability through replication on different infrastructures. The QoP requirement over multiple clouds is guaranteed by the description-based model elaborated by Mantus.

To assess overhead when using nested virtualization, we evaluated our Orbits prototype in terms of both performance and scalability. To this end, network latency and bandwidth represent important parameters to influence the execution performance of healthcare applications as analyzed earlier. Figures 5a and 5b compare nested virtualized execution environments (VM plus containers), single-layer VMs, and a bare-metal system. Degradations are concentrated in the nested KVM setting, where overhead

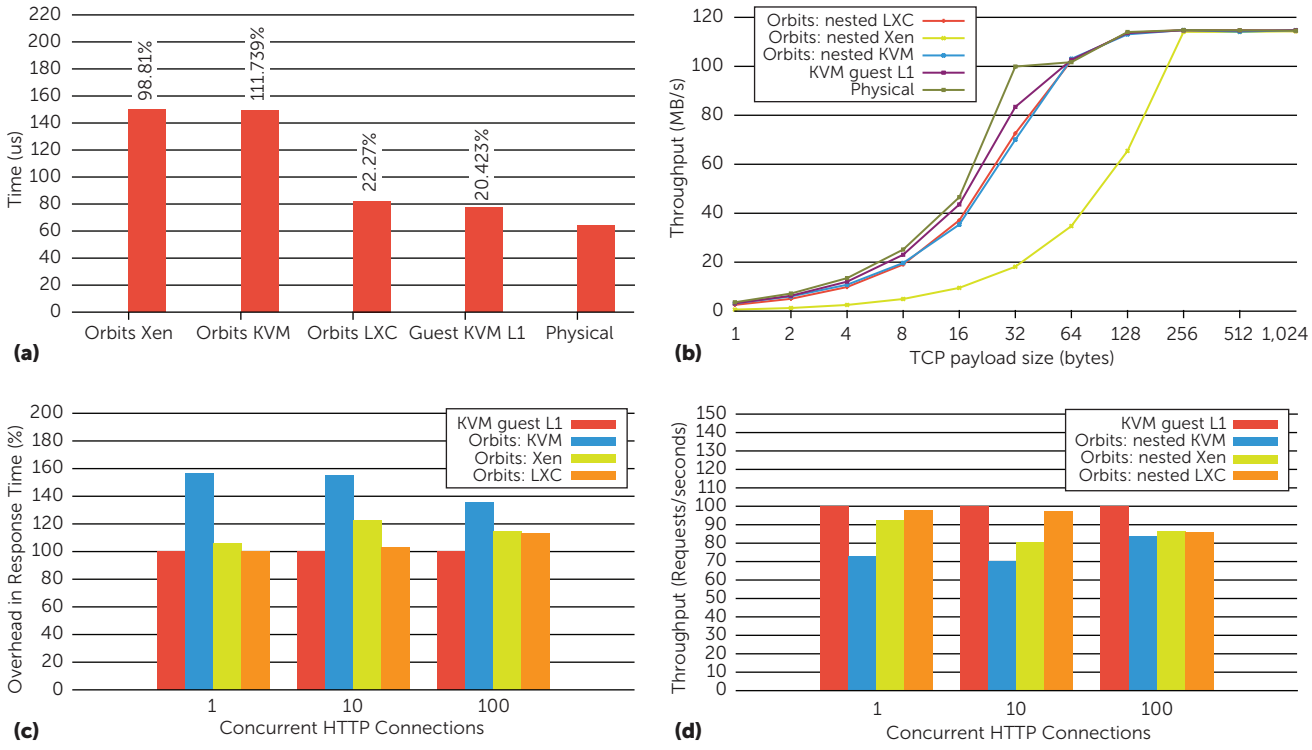

FIGURE 5. We ran performance and scalability tests using an Intel Xeon E5-2650 Haswell at 2.60 GHz with 64 Gbytes of RAM and Centos 7 as a bare-metal operating system. The base software platform is an OpenStack over Linux KVM executing Ubuntu 16.04 guests VMs, with a paravirtualized VirtIO drivers network card and disk. (a) Average TCP latency (less is better). (b) Average TCP throughput. (c) Request service response time. (d) Request throughputs per second.

often exceeds 50 percent compared to the baseline. LXC performs quite well and can be considered a viable solution to introduce a user-controlled virtualization layer.

As Figures 5c and 5d show, we tested the scalability of nested execution environments when increasing load in a WordPress application. A WordPress application, like many healthcare applications,[3] relies on a Web front end, a server-side application logic, and access to a database, and could be used as a generic and representative benchmark. From the perspective of both throughput and elapsed time, Xen and LXC perform well, keeping overhead below 20 percent. In addition, from a scalability viewpoint, control of a nested virtualization layer on a public cloud makes physical collocalization possible,[4,5] which might enable better performance regardless of the underlying provider, in the context of applications using multiple execution environments.

To sum up, experimental results show that the performance and scalability loss of the Orbits architecture due to the adoption of an extra virtualization layer might be affordable. The cost to adopt a new provider isn't huge in terms of code development, so supporting new providers would require adding only their orchestration service to the appropriate Mantus driver.

We plan to extend the Orbits architecture with additional features, such as the ability to model security services (Tosca) and weave them into the functional infrastructure, and to integrate SLAs. We also intend to benchmark the Mantus and Stratopause components and overall Orbits framework using sample healthcare applications to further validate multicloud-aware placement and follow-me types of ubiquitous healthcare scenarios, as well as other classes of applications to evaluate the genericity of the architecture in a variety of use cases. We'll also address the additional management complexity introduced by multiple overlays, exploring existing frameworks (such as the Virtual Environment Self-Protecting Architecture)[9] to enrich Stratopause and Mantus with self-management features for typical administration tasks, or detection of and reaction to unusual events such as failures.

Acknowledgments

This work is partially supported by the European Union SUPERCLOUD Project (Horizon 2020 Research and Innovation Program, grant 644962) and by the Swiss Secretariat for Education, Research and Innovation (contract 15.0091).

References

1. Z. Jin and Y. Chen, "Telemedicine in the Cloud Era: Prospects and Challenges," *IEEE Pervasive Computing*, vol. 14, no. 1, 2015, pp. 54–61.
2. S. Biswas et al., "Cloud Based Healthcare Application Architecture and Electronic Medical Record Mining: An Integrated Approach to Improve Healthcare System," *Proc. 17th Int'l Conf. Computer and Information Technology* (ICCIT 14), 2014, pp. 286–291.
3. M. Deng et al., "A Home Healthcare System in the Cloud: Addressing Security and Privacy Challenges," *Proc. IEEE 4th Int'l Conf. Cloud Computing* (Cloud 11), 2011, pp. 549–556.
4. K. Razavi et al., "Kangaroo: A Tenant-Centric Software-Defined Cloud Infrastructure," *Proc. IEEE Int'l Conf. Cloud Eng.*, 2015, pp. 106–115.
5. D. Williams et al., "The Xen-Blanket: Virtualize Once, Run Everywhere," *Proc. 7th ACM European Conf. Computer Systems* (EuroSys12), 2012, pp. 113–126.
6. M. Ben-Yehuda et al., "The Turtles Project: Design and Implementation of Nested Virtualization," *Proc. Operating System Design and Implementation* (OSDI) 10, 2010, pp. 423–436.
7. S. Soltesz et al., "Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors," *ACM SIGOPS Operating Systems Rev.*, vol. 41, no. 3, 2007, pp. 275–287.
8. A. Abbas and S.U. Khan, "A Review on the State-of-the-Art Privacy-Preserving Approaches in the E-Health Clouds," *IEEE J. Biomedical and Health Informatics*, vol. 18, no. 4, 2014, pp. 1431–1441.
9. A. Wailly, M. Lacoste, and H. Debar, "Vespa: Multi-Layered Self-Protection for Cloud Resources," *Proc. 9th Int'l Conf. Autonomic Computing*, 2012, pp. 155–160.

ALEX PALESANDRO *is a PhD student at Orange Labs and the University of Lyon III. His research interests include cloud computing technologies, with a focus on virtualization and hypervisor security. Palesandro has a master's degree in computer engineering as part of double degree program between the Politecnico di Torino, Italy, and the École Nationale Supérieure d'Informatique et Mathématiques Appliquées of Grenoble, France. Contact him at alex.palesandro@orange.com.*

CHIRINE GHEDIRA GUEGAN *is a full professor of computer sciences and co-head of the service-oriented computing research team at the Lyon Research Center for Images and Intelligent Information Systems associated with the French National Center for Scientific Research (CNRS) in Lyon, France. Her research interests include service-oriented architectures and computing; interoperability; complex, autonomic, and adaptive systems; context-aware computing; data services; privacy; and cloud computing. Guegan has a research habilitation in computer science from Université de Lyon I. Contact her at chirine.ghedira-guegan@univ-lyon3.fr.*

MARC LACOSTE *is a senior research scientist in the Security Department of Orange Labs. His research interests include security architecture, cloud computing security, self-protecting systems, and open security kernels. Lacoste has a PhD in computer science from the University of Grenoble, France. Contact him at marc.lacoste@orange.com.*

NADIA BENNANI *is an associate professor at the Institut National des Sciences Appliquées de Lyon. Her research interests include security, privacy, and data management in clouds and mobile networks. Bennani has a PhD in computer science from the University of Lille France. Contact her at nadia.bennani@insa-lyon.fr.*