DevOps as a Service: Pushing the Boundaries of Microservice Adoption

Taking the Pulse of DevOps in the Cloud

Demetris Trihinas
University of Cyprus

Athanasios Tryfonos
University of Cyprus

Marios D. Dikaiakos
University of Cyprus

George Pallis
University of Cyprus

Software teams of all sizes are embracing the DevOps philosophy to rapidly deliver applications by adopting the microservice paradigm. Industry trends show clearly that microservice adoption is expanding rapidly. However, microservices are not without challenges. Deployment at scale calls for implementing autonomic principles and solutions, which lead to higher complexity and increased failure risk.

The evolvement of software development paradigms is following the industry needs for applications that adhere to the notions of *modularity*, *distribution*, *elasticity* and *robustness*.[1] The birth of the microservice paradigm has emerged from common (best) practices adopted by leading companies such as Netflix, Amazon, and Uber, which embraced DevOps concepts into their software development lifecycles and deployed their coded artifacts seamlessly across cloud platforms, almost exclusively through containerized execution environments.

Microservices can be seen as the resulting set that arises from the decomposition of an application into smaller pieces (services), which tend to run as independent processes and have the ability to intercommunicate using lightweight communication mechanisms.[2] Applications embracing this paradigm are sliced into services organized around discrete business capabilities with the boundaries between these units usually comprised of platform-agnostic APIs that expose the core capabilities of each service. Large systems are then composed of many (micro) services, whereby communication between services is a central ingredient.

Although not a necessity, microservices are packaged and deployed through containerized runtimes offering portability across private and public cloud deployments. This setting is ideal for data harvesting and manipulation, content streaming, recommendation services and the Internet of Things, where software is compiled and offered in the cloud as a service from a set of smaller,

Published by the IEEE Computer Society
1089-7801/18/$33.00 ©2018 IEEE

loosely-coupled and independent features. Microservice adoption is surging with RedHat's 2017 State of Microservices[3] report indicating that all but 30% of respondents are embracing the microservice paradigm to re-architect existing applications and for new projects.

In the cloud era, as applications grow by adding more services, security enforcement and dynamic resource allocation become significant challenges. At scale, these challenges can be addressed with autonomicity. Through automation, microservices are equipped with the ability to continuously control the underlying infrastructure, thus turning into services that can be harnessed programmatically at runtime. However, traditional monitoring is ineffective for ephemeral, decomposed and highly dynamic microservices deployed over shared execution environments. On the other hand, finer service-granularity means more moving parts and hence an increased complexity of auto-scaling, potentially more points of failure, and more possibilities for serious security violations and privacy leaks. For example, if you are a startup comprised of a handful of developers, you may quickly release a prototype but once the hype is out, if security is not properly dealt with, you are a few clicks away from loosing your user base. In turn, if you are in the content streaming business, either this may be music, video or even micro-blogging, a few seconds of unanticipated content buffering are enough to dreadfully impact your subscription-based revenue stream. This challenge set creates the need for devising new solutions with the ability to design, run and monitor microservices at scale while also achieving the anticipated autonomicity and security of the application runtime on top of the underlying programmable cloud infrastructure.

MICROSERVICES

Microservices are now being studied as part of the cloud infrastructure. But, *why are microservices so important to the future of DevOps*? Unlike software-heavy VMs, microservices can share the core of the underlying OS, which enables faster deployments in the cloud without diminishing performance. Thus, instead of all application services being part of one enormous monolith, business capabilities are self-contained with well-defined interfaces that avoid synchronous and blocking-calls whenever possible. By adopting the DevOps "ideology," separate software teams are each responsible for different aspects of the end application allowing both the team and software core to *develop*, *test*, *handle failures* and *scale independently*.

What is more, *continuous integration* and *continuous delivery* of frequent and incremental changes to the codebase, a task almost impossible to achieve for monoliths, is possible as small units are easier to develop, deploy and manage, while a software bug only affects a single service and not the entire application. In turn, each service dealing with a specific feature may utilize its own data back-end to optimize storage, processing and acquisition to fill its needs. Thus, *for each (micro-) service one can understand, alter and write new code without knowing anything about the internals of its peers*, because services and their peers interact strictly through APIs and hence, there is no need for sharing or exposing data structures, schemata, or other internal object representations.

It becomes evident that to support small and independently deployable services, infrastructure-wise lightweight mechanisms, scalability and portability are of essence. These requirements can be met by using containers. Containerization allows applications to share a single host OS in a portable manner. Because containers do not have the overhead of an entire guest OS, something absolutely required by VMs, their size is significantly smaller which makes them *easier to migrate, faster to boot and less demanding on memory*. As a result, it is possible to run many more containers on the same infrastructure rather than VMs.[4] Development with the use of containers is perfect for microservices, as complex applications are split into discrete and modular units where e.g., a database backend might run in one container while the front-end runs in a separate one. Hence, containers reduce the management complexity because any potential problem or change related to one service does not require an overhaul of the overall application.

Containers are not a new technology. They have been part of the Linux ecosystem for more than a decade with the goal to offer resource isolation at the OS level through kernel namespaces and control groups (cgroups). In brief, namespaces deal with resource isolation for a single process, while cgroups manage access for groups of processes. Nonetheless, for a containerized application to run, specialized software must coordinate and streamline the process on top of the OS. The most popular engine to create, ship and run Linux containers is by far Docker with RightScale's 2017

State of the Cloud[5] report declaring that Docker is the golden standard for containerized technologies. Docker automates the packaging of an application and its dependencies within a portable container that can run on any Linux OS. This enables flexibility for the packaged application which can run without customization on laptops, virtual machines, bare-metal and the cloud. To advance container standardization, Docker has even contributed its container format and runtime environment to the Open Container Initiative, which is operated by the Linux Foundation.

CHALLENGES

The plethora of toolsets entering the evolving microservice landscape, although overwhelming, are lowering the innovation barriers towards microservice adoption for software teams of all sizes. However, there are still a number of challenges that must be overcome to make this adoption simpler and faster. Most importantly, moving the microservice model from evaluation to production and then scaling to meet demand, is a whole different picture.

In particular:

- **Monitoring and Diagnostics**: Most traditional monitoring tools, even in the cloud era, are designed for slowly evolving execution environments with application instances resembling one another and residing on VMs. While containers ease application deployment, monitoring them is an open challenge as there is no guest OS to deploy a monitoring agent alongside running services. An agent must be run through the container engine or be part of the application itself, which means that monitoring should be an integral part of application design and cannot be decided after deployment. Granularly slicing an application into services inherently introduces heterogeneity which requires full customization of the monitoring process to perform diagnostics and receive helpful insights. However, customization must be automated and become part of the continuous delivery process for immutable container environments. In turn, there are significant costs and actual runtime overheads[6] when monitoring ephemeral, decomposed and highly dynamic applications over virtualized and shared execution environments.
- **Auto-Scaling and Optimization**: Scaling to meet demand is a challenge for most applications and microservices are no exception. Although microservices and containers are inherently easier to scale by simply creating more copies of the services overwhelmed by demand (horizontal scaling), significant profiling is required to optimize performance, cost and quality as one must identify what should be monitored, when to scale, and even, how to scale.[7] If this is not already a daunting challenge, then also consider the difficulty in determining what is the best optimization strategy to follow and how to investigate, in a distributed and granular deployment, if one service endpoint is currently affecting the performance of another service endpoint.
- **Orchestration in Hybrid Cloud Deployments**: Software teams are increasingly choosing to work with multiple cloud offerings and/or cloud providers so as to meet their goals. Although containers indeed provide both flexibility and portability by being able to run—ideally—anywhere, this does not mean a deployment can span across geographic regions, cloud availability zones and different cloud sites.[8] This challenging task requires devising complex placement strategies for determining where each service should run and provision the appropriate infrastructural resources and also requires constructing and managing a cross site overlay network. This introduces severe propagation delays and other network performance issues for service communication especially when network traffic must pass across datacenter boundaries.
- **Security Enforcement and Privacy Protection:** Decomposing an application into services with each service inter-communicating over the network raises significant security risks. This can lead to severe privacy leaks with code vulnerabilities lurking in the diversified application stack in both in-house written code and third-party libraries. This requires dealing at scale with the security of numerous self-contained services, while also having to maintain identity and access management across the entire deployment instead of just a single monolith. However, mitigating new sets of security rules at runtime without service disruption is another challenge, if one is not to shut down thousands of instances for alteration.

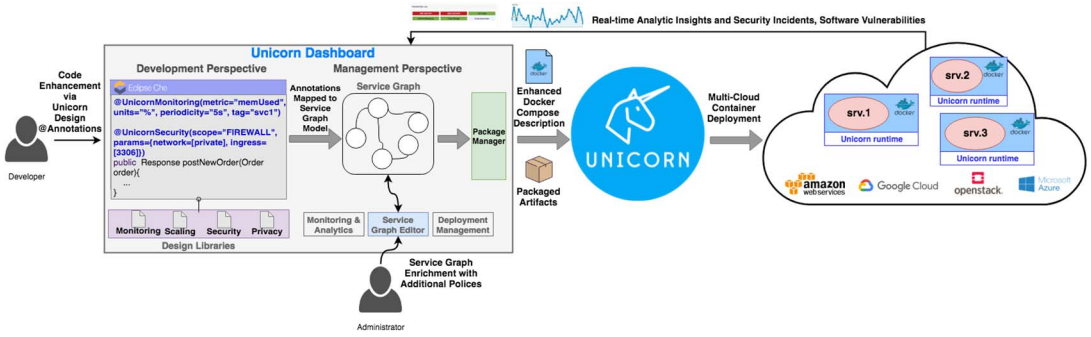

Figure 1. High-level overview of the Unicorn framework.

ENTER THE UNICORN FRAMEWORK

To overcome these challenges and take full advantage of the rapidly evolving microservices landscape, the newly released Unicorn Framework (http://unicorn-project.eu) aims to provide software teams of all sizes with a powerful toolset to simplify the design, development and management of scalable and secure applications over multi-cloud containerized execution environments. To achieve this, its novel offerings are available to software teams as a unified DevOps-as-a-Service platform (see Figure 1). In respect to microservices, Unicorn facilitates microservice adoption by providing through its DevOps offerings both a cloud IDE plugin for application development and packaging and a dashboard for runtime management. Going beyond the offerings of existing toolsets, Unicorn puts particular emphasis on both security and elastic scaling enabled through policy and constraint definition, as well as, through continuous risk and vulnerability assessment, and complements its solution with advanced orchestration and monitoring capabilities.

The Unicorn Dashboard

Microservices are an integral part of the shift in ICT towards a DevOps culture, in which DevOps teams work closely together to manage an application over its lifecycle, and go through rapid and continuous releases. The Unicorn framework in its philosophy is not different. Through the Unicorn Dashboard -either one is a developer, product engineer or an administrator- collaborative development and management of the application lifecycle is conducted via two perspectives.

The *Development Perspective* supports software teams to develop secure and scalable microservices using Design Libraries to annotate their source code with policy and constraint definitions. This allows for minimal code intrusion while hiding the complexity behind the operation of each code annotation in the background to free developers by not affecting code development, debugging and software releases. Most importantly, developers are not constantly derailed and can focus on core application development leaving security enforcement, privacy restrictions, monitoring and elastic scaling to the handlers behind the design libraries and the Unicorn platform.

Although the design libraries can be downloaded and used without any further tools required, these libraries along with the Unicorn packaging and deployment toolset are available through the newly released Unicorn plugin for the popular and open-source Eclipse Che cloud IDE. This provides *one collaborative and unified environment in the same place where developers write their code, allowing them to share workspaces, ship applications to the cloud, and then, manage the lifespan of their deployments*. In addition, the Unicorn plugin provides developers with the capability to search for OS and runtime libraries that will extend their container environments, a tool currently absent from the fast-evolving container community.

Concurrently, but in the background, the Unicorn IDE plugin enriches a topology-aware descriptive model denoted as the *service graph* which is automatically compiled and maintained as the developer describes her application. This topology-aware description is not based on yet-another domain specific language but in contrary it adheres to the Topology and Orchestration Specifica-

tion for Cloud Applications[9] (TOSCA) maintained by OASIS and is compiled as a Docker Compose file which holds the configuration for multi-container Docker applications. This Unicorn enriched Compose file can still be used in any other Docker environment where Unicorn-related constraints and policies, along with their benefits, will be ignored. This means Unicorn enriched Docker Compose files still adhere to the container paradigm requirement that a description must be portable and run anywhere.

Through the *Management Perspective* administrators and product managers can modify and define additional policies and constraints that will govern the operation of deployed applications through the graphical service graph editor. Administrators can also view in an intuitive graphical manner collected metrics, potential security incidents and vulnerabilities, and manage the lifecycle of their deployments.

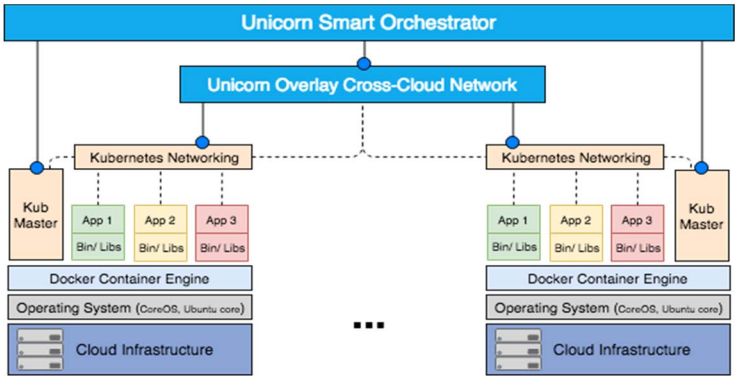

Figure 2. Unicorn orchestration.

The Unicorn Platform

The Unicorn Platform acts as link between the Unicorn Cloud IDE Plugin and the Multi-Cloud Execution Environment and is the layer where the business logic of Unicorn is applied. Its main tasks include:

- The validation of service graph submitted for deployment to detect potential problems such as antagonizing policy restrictions and circular dependencies.
- The interpretation and binding of the annotated source code.
- The enforcement of privacy, security and elastic scaling policies at runtime and compile-time based on the annotated policies and constraints.
- The application lifecycle management of deployed applications
- The orchestration, monitoring and management of both the underlying programmable infrastructure, network fabric and multi-cloud containerized execution environment.

When embracing Unicorn, there is no requirement to learn new runtimes or proprietary technologies, which is always a challenging task for software teams and requires investing in training and expertise. Unicorn is not another runtime, container or language but a service built on top of popular and open-source technologies. In particular, as the service graph description is translated into a Unicorn-enhanced Docker Compose file, Unicorn makes use of the Docker Engine to run microservices in containers. However, while the Docker Engine is sufficient for small deployments, it is limited to a single host environment.

To support the orchestration of large-scale distributed containerized deployments spanning across multiple hosts, Unicorn makes use of Kubernetes, a popular and open-source orchestration tool for containers running on a cluster of machines (see Figure 2). This tool provides the ability to automatically provision and de-provision containerized applications running on a single cluster but it has two important limitations. Specifically, Kubernetes lacks the ability to (de-)provision infrastructure resources and does not support deployments spanning across multiple cloud sites. To address these limitations, the Unicorn Platform extends Kubernetes with cloud adaptors to

probe and program the underlying infrastructure. Most importantly, Unicorn introduces cross-cloud network overlay management along with low-cost and self-adaptive monitoring to reliably handle over Software Defined Networks network accessibility and reduce the monitoring intensity along with traffic propagation.

In regard to elastic scaling, Unicorn provides DevOps teams with an extensible policy-based definition toolset to tap into the auto-scaling mechanisms offered by the underlying cloud offerings in order to estimate and assess the elasticity behavior and scaling effects of their deployed applications. Finally, Unicorn adopts the lightweight and library-based CoreOS as its underlying OS to provide policy-based and secure out-of-the-box support for the Docker runtime engine.

CONCLUSION

The adoption of microservices enables teams to rapidly develop and deploy modern, distributed, cloud-based applications. The use of microservices and containers is a new standard for the cloud computing industry which is becoming a trend in IoT as well. The technology shift that has been observed to microservices and containers is expected to continue and increase. As many organizations adopt the microservice paradigm and migrate their apps to the cloud, new programming tools must be developed. In this context, the Unicorn framework adopts and extends popular and open-source technologies to provide software teams with a unified DevOps-as-a-Service platform that simplifies the design, deployment and management of scalable and secure applications over multi-cloud containerized execution environments.

ACKNOWLEGMENTS

This work is partially supported by the EU Commission in terms of the Unicorn 731846 project (H2020-ICT-2016-1).

REFERENCES

1. J. Thones, “Microservices,” *IEEE Software*, vol. 32, no. 1, 2015, p. 116.
2. S. Newman, *Building Microservices*, O'Reilly Media, 2015.
3. “The 2017 State of Microservices,” blog, RedHat; https://developers.redhat.com/blog/2017/12/05/state-microservices-survey-2017-eight-trends-need-know/.
4. R. Morabito et al., “Consolidate IoT Edge Computing with Lightweight Virtualization,” *IEEE Network*, vol. 32, no. 1, 2018, pp. 102–111.
5. “The 2017 State of the Cloud Survey,” RightScale; https://www.rightscale.com/lp/2017-state-of-the-cloud-report.
6. D. Trihinas, G. Pallis, and M. Dikaiakos, “ADMin: Adaptive monitoring dissemination for the Internet of Things,” *IEEE Conference on Computer Communications* (INFOCOM 17), 2017, pp. 1–9.
7. M.R. López and J. Spillner, “Towards Quantifiable Boundaries for Elastic Horizontal Scaling of Microservices,” *Proceedings of the10th International Conference on Utility and Cloud Computing* (UCC 17), 2017, pp. 35–40.
8. M. Villari et al., “Osmotic Computing: A New Paradigm for Edge/Cloud Integration,” *IEEE Cloud Computing*, vol. 3, no. 6, 2016, pp. 76–83.
9. N. Loulloudes et al., “Enabling Interoperable Cloud Application Management through an Open Source Ecosystem,” *IEEE Internet Computing*, vol. 19, no. 3, 2015, pp. 54–59.

ABOUT THE AUTHORS

Demetris Trihinas is a Senior Researcher at the Department of Computer Science, University of Cyprus. His research interests include Distributed and Internet Computing with particular focus on Cloud Monitoring and Elasticity, and the Internet of Things. He has a PhD in Computer Science from the University of Cyprus. Contact him at trihinas@cs.ucy.ac.cy.

Athanasios Tryfonos is a Researcher at the Department of Computer Science, University of Cyprus. His research interests include Cloud Computing, Microservices, Containerization and Virtualization technologies. He has a BSc in Computer Science from the University of Cyprus. Contact him at a.tryfonos@cs.ucy.ac.cy.

Marios Dikaiakos is professor of computer science at the University of Cyprus, Nicosia, and director of the University's Centre for Entrepreneurship. His research interests are in the area of Internet Computing, focusing on the design and implementation large-scale federated computing systems and tools, online social networks and big data. He has a PhD in computer science from Princeton University. Contact him at mdd@cs.ucy.ac.cy.

George Pallis is an assistant professor of computer science at the University of Cyprus, Nicosia. His research interests focus on distributed systems, cloud computing and Big Data analytics. Pallis has a PhD in computer science from the Aristotle University of Thessaloniki. Contact him at gpallis@cs.ucy.ac.cy.