Orquestación de servicios para el desarrollo de aplicaciones para big data

Maria Murazzo[1], Miguel Guevara[1], Martin Tello[1], Nelson Rodriguez[1], Fabiana Piccoli[2], Mónica Gimenez[3]

[1]*Departamento de Informática, F. C. E. F. y N, Universidad Nacional de San Juan*
marite@unsj-cuim.edu.ar, migueljoseguevaratencio@gmail.com, martinl.tello@gmail.com, nelson@iinfo.unsj.edu.ar

[2] *Departamento de Informática - F.C.F.M. y N, Universidad Nacional de San Luis*
mpiccoli@unsl.edu.ar

[3]*Facultad de Ciencias Exactas Físicas y Naturales – Universidad Nacional de La Rioja*
monik.gimenez@gmail.com

Abstract

Los avances tecnológicos han permitido que se generen grandes cantidades de datos, los cuales necesitan ser almacenados y procesados de manera eficiente. Surge así el paradigma Big Data, donde el principal requerimiento no solo es la capacidad de cómputo, sino el manejo en un tiempo razonable de ingentes cantidades de datos. En este contexto, las aplicaciones para big data necesitan ser escalables, livianas, autocontenidas, distribuidas y replicadas con el objetivo de lograr la mejor performance frente a variaciones del volumen de datos. Para lograr esto, este trabajo propone ajustar la construcción de aplicaciones a una arquitectura basada en microservicios los cuales puedan ser implementados con contenedores. La replicación y distribución para lograr altos niveles de escalabilidad se plantea mediante la orquestación de contenedores sobre una arquitectura distribuida virtualizada.

Keywords: Big Data, Orquestación, Microservicios, Contenedores, Docker

1. INTRODUCCIÓN

La integración de Cloud Computing (CC) [1] con Internet of Thing (IoT) [2] representa el próximo gran paso en las TI del futuro. Las aplicaciones que surjan de esta integración abrirán nuevas perspectivas de negocio y oportunidades de investigación. Gracias al paradigma CloudIoT [3], la vida cotidiana mejorará, por ejemplo, en las Smart city [4], permitirán servicios públicos más eficientes través de aplicaciones ubicuas que mejorarán la calidad de vida de los ciudadanos. El futuro de este paradigma pasa por la convergencia hacia una plataforma de servicio común que supere los desafíos planteados por la heterogeneidad de los dispositivos y las tecnologías involucradas. Esto conlleva tener en cuenta la ubicuidad y la omnipresencia de los dispositivos, lo cual, requieren plataformas de computación escalables.

Esta convergencia ha provocado que se creen datos a una tasa exponencial, dando lugar al paradigma Big Data [5]. Big data se caracteriza por tres aspectos: a) el volumen de los datos, b) los datos no se pueden almacenar de forma tradicional y c) los datos are generados, capturados y procesados con rapidez. Estas características hacen que los sistemas de cómputo convencionales sean inapropiados para lograr una gestión adecuada del big data [6].

Las arquitecturas de software tradicionales no permiten almacenar y gestionar grandes volúmenes de datos con el fin de extraer un valor estratégico. El reto del big data es convertir grandes volúmenes de datos en inteligencia de negocio, lo cual implica una gestión de datos distinta, en la que se incorpora el análisis en tiempo real para orientar la toma de decisiones. No se trata de recopilar datos y hacer análisis detallados, sino más bien en hacer interpretaciones rápidas que orienten en la toma de decisiones a partir de datos de cualquier fuente [7].

Construir aplicaciones para big data tiene como principal problema la necesidad de administrar recursos, lo cual implica analizar y planificar la forma en la cual la aplicación debe escalar.

Tradicionalmente, la construcción de aplicaciones ha seguido un paradigma de escalado vertical mediante el cual la solución a los problemas de cuellos de botellas se resuelve mediante el

aprovisionamiento de mayor cantidad de recursos. Pero cuando se debe procesar y analizar grandes cantidades de datos, las aplicaciones deben pensarse y construirse de forma distribuida; obligando a contar con mecanismos de *escalabilidad horizontal*, que permita distribuir la carga de trabajo dinámicamente y paralelizar las tareas. La escalabilidad horizontal implica balanceo de carga, replicación de recursos, reinstanciación de recursos en tiempo real y optimización en su uso, de forma que el usuario no perciba degradación de performance.

El presente trabajo tiene como objetivo definir una manera de construir aplicaciones para big data que escalen de forma horizontal, lo cual asegurará que los datos sean almacenados, procesados y visualizados en forma satisfactoria.

El resto del trabajo se organiza de la siguiente manera: en la próxima sección se explican las generalidades de las arquitecturas de soporte tanto de hardware como de software. En la Sección 3 se explican las generalidades de la orquestación de servicios. En la Sección 4 se realiza una breve descripción de la herramienta seleccionada. En la Sección 5 se describe el caso de estudio abordado. En la sección final se abordan las conclusiones y futuros trabajos.

2. ARQUITECTURA DE SOPORTE

Como se mencionó en la sección anterior, las tecnologías, arquitecturas, lenguajes y metodologías tradicionales son inapropiadas para realizar el tratamiento de datos masivos. Es por ello que una alternativa es usar una arquitectura distribuida [8] con el fin de proveer tolerancia a fallos, distribución y replicación.

La computación distribuida es la evolución de los sistemas de cómputo convencional, los cuales permiten realizar operaciones de cómputo intensivo y mejorar la velocidad de procesamiento; involucrando tecnologías tal como cluster y cloud [9].

Los entornos distribuidos son ideales para ejecutar aplicaciones computacionalmente costosas con manejo de grandes cantidades de datos, a fin de lograr resultados en menor tiempo.

La conjunción de big data y la computación distribuida, se enfoca en la paralelización del problema mediante la distribución de los datos y la delegación del cómputo en los nodos con capacidad de procesamiento [10].

Construir aplicaciones capaces de almacenar, procesar y visualizar grandes volúmenes de datos sobre una arquitectura distribuida implica la definición de los métodos de distribución y replicación a fin de lograr la escalabilidad deseada [11].

2.1. ARQUITECTURA DE SOFTWARE

Para construir aplicaciones con las características de escalado mencionadas en el apartado anterior, es necesario que la arquitectura de software se alinee con estas características.

En tal sentido, la arquitectura de microservicios [12] permite construir aplicaciones distribuidas como un conjunto de pequeños servicios desacoplados, desplegados de manera independiente y que trabajen coordinadamente.

De la misma manera que en una aplicación monolítica, los microservicios pueden acceder a datos a través de capas de persistencia, drivers o configuración específica para cada caso, exponiendo sus funcionalidades a través de API's REST y comunicándose entre sí con mecanismos ligeros tales como API de HTTP o mensajería. Una de las ventajas más importantes, es que los microservicios deben ser creados, mantenidos, ejecutados y distribuidos de forma totalmente independiente [13].

Una de las características más importantes que provee los microservicios es la escalabilidad [14]. Debido a que los microservicios se desarrollan en forma independiente uno de otros, también escalan en forma independiente, esto es muy importante cuando se trabaja con grandes volúmenes de datos pues permite la implementación de carga compartida de trabajo.

Los microservicios dependen no solo de la tecnología que se está configurando, sino también de una organización que tenga la cultura, el conocimiento y las estructuras establecidas para que los equipos de desarrollo puedan adoptar este modelo. Los microservicios forman parte de un cambio más amplio en los departamentos de TI hacia una cultura DevOps [15], en la que los equipos de desarrollo y operaciones trabajan estrechamente para respaldar una aplicación a lo largo de su ciclo de vida y pasan por un ciclo de publicación rápido o incluso continuo en lugar de un ciclo tradicional largo.

Un enfoque de microservicios también puede facilitar que los desarrolladores de aplicaciones ofrezcan interfaces alternativas a sus aplicaciones. Cuando todo es una API, las comunicaciones entre los componentes de la aplicación se estandarizan. Todo lo que tiene que hacer un componente para hacer uso de su aplicación y sus datos es poder autenticarse y comunicarse a través de esas API estándar [13]. Esto permite que tanto los que están dentro como, cuando corresponda, los que estén fuera de su organización, desarrollen fácilmente nuevas formas de utilizar los datos y servicios de su aplicación.

2.1. 1. CONTENERIZACIÓN

El uso de microservicios implica la construcción de aplicaciones a partir de múltiples componentes autosuficientes. Estos componentes pueden ser implementados con contenedores [16].

Un contenedor, es un proceso que, internamente, contiene la aplicación que se quiere ejecutar y todas sus dependencias usando indirectamente el kernel del sistema operativo que se esté usando. Los contenedores permiten implementar aplicaciones distribuidas debido a dos características: portabilidad, pues los contenedores se pueden ejecutar de forma independiente al hardware y al software donde se crearon; y baja sobrecarga, pues son livianos e introducen menos overhead que las VM (Virtual Machine) [17].

Realizando un paralelismo entre un contenedor y una VM (Virtual Machine) [18] [19], ambos son sistemas autocontenidos que tienen como principal diferencia que, una VM necesita contener todo el sistema operativo mientras un contenedor aprovecha el sistema operativo en el que se ejecute.

La principal ventaja [20] de utilizar contenedores es que no depende de las instalaciones del sistema anfitrión, es decir es posible realizar las instalaciones necesarias para desplegar aplicaciones dentro de un contenedor. Otra ventaja es que permite la portabilidad en las aplicaciones, esto quiere decir que además de crear contenedores, es posible definir repositorios que eventualmente alojarán a los contenedores con el objeto que otros usuarios los descarguen y los consuman como servicio.

Quizás la características más relevante de los contenedores es que permiten desplegar aplicaciones de forma rápida y escalables, ya que permite replicar ambientes de desarrollo, prueba y producción en poco tiempo, pudiendo aumentar o disminuir la cantidad de contenedores a medida que sean necesitados (escalabilidad elástica) [21].

El uso conjunto de contenedores y microservicios [22] mejora las capacidades de escalado, ya que el microservicio es portable y reutilizable; mientras que los contenedores proporcionan recursos eficientes, principalmente el empaquetado de aplicaciones y sus dependencias en un contenedor virtual que puede ejecutarse en cualquier servidor.

3. ORQUESTACIÓN DE SERVICIOS

Se ha dicho que cuando se trabaja con datos masivos es necesario escalar horizontalmente las aplicaciones, esto se puede lograr mediante el uso de múltiples nodos (cluster o cluster as a service). Para lograr esta forma de distribución es fundamental abstraerse de la complejidad de la plataforma subyacente, lo cual se logra mediante el uso de clusters de contenedor [23] como se muestra en la figura 1.

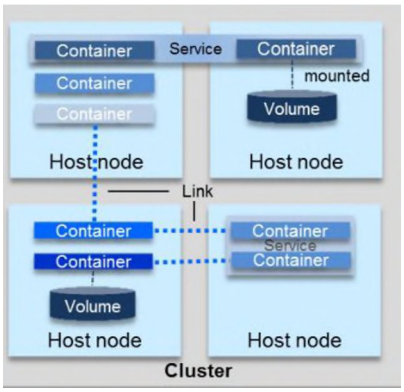

Figura 1: Arquitectura de un cluster de contenedores

Cada nodo en el cluster es un servidor virtual y puede tener múltiples contenedores con una semántica común, a su vez, una aplicación puede estar formada por un grupo lógico de varios contenedores, la cual puede escalar a través de múltiples nodos.

Esta forma de trabajar necesita administrar y coordinar los contenedores (microservicios), para ello es posible realizar orquestación o coreografía [24].

Orquestar significa que hay una entidad central (idealmente un microservicio en sí mismo) que coordina la comunicación entre los microservicios. Mientras que para microservicios coreografiados, cada servicio implicado en dicha coreografía "conoce" exactamente cuándo ejecutar sus operaciones y con quién debe interactuar, lo cual le quita transparencia e independencia a los servicios. Los servicios orquestados no "conocen" (y no necesitan conocer) que están implicados en un proceso de composición y que forman parte de un proceso de negocio de nivel más alto. Solamente el coordinador central de la orquestación es "consciente" de la meta a conseguir, por lo que la orquestación se centraliza mediante definiciones explícitas de las operaciones y del orden en el que se deben invocar los servicios (véase figura 2).

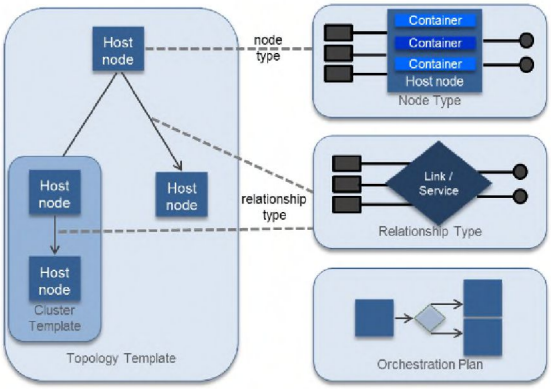

Figura 2: Orquestación en clúster de contenedores

En este trabajo se ha decidido usar orquestación, debido a las siguientes razones:

- Los microservicios no tienen que conocer el proceso de negocio.
- Existe una pieza central a la que se le puede preguntar por el estado de los procesos, estadísticas, etc.
- Permite la migración de versiones de procesos comenzados.

Gracias a la orquestación de contenedores [25] se puede trabajar de forma transparente frente al aumento de carga de trabajo, falla de algún nodo, creación de nuevos contenedores, etc. El orquestador decide cuándo adaptar el sistema (por ejemplo, para iniciar la ejecución de un contenedor) y donde la adaptación tendrá lugar (por ejemplo, en qué nodo se ejecutarán los contenedores).

Orquestar contenedores, aísla al usuario de los detalles de implementación, lo cual provee transparencia en el acceso y ejecución de los contenedores. Este aspecto es importante cuando se desea implementar replicación y distribución de datos masivos con el objetivo de lograr balanceo de carga y auto escalado horizontal, dependiente de la sobrecarga del volumen de información. Además, la orquestación, permitirá un aprovisionamiento de recursos bajo demanda y un escalado horizontal de la aplicación desarrollada.

En resumen, se puede decir que un orquestador de contenedores es la herramienta, que permite crear un clúster de alta disponibilidad de virtualización de contenedores, y lo dota de un sistema de orquestación dinámica de servicios a nivel de múltiples servidores.

Existen varios productos que permiten realizar la orquestación de contenedores, entre los cuales se puede mencionar: Kubernetes [26], Marathon Mesos [27], Conductor [28] y Docker Swarm [29]

4. Herramienta Seleccionada

En este trabajo de investigación se ha decidido trabajar con las tecnologías de contenerización provistas por el ecosistema Docker [30], cuya plataforma se muestra en la figura 3.

Figura 3: Plataforma Docker

Docker es un proyecto "open source" que permite crear aplicaciones en contenedores de software que son ligeros, portables y autocontenidos.

La implementación de la orquestación se realizó con Docker Swarm para orquestar los contenedores construidos con Docker. De esta manera, es posible convertir un conjunto de nodos (clúster físico) en un nodo único virtual para aprovechar los recursos de cada una de las máquinas.

El reparto de contenedores entre los nodos esta implementado por Docker y para la comunicación con el clúster se utiliza la API estándar de Docker, por lo que cualquier aplicación o herramienta que se comunicaba con un proceso Docker, puede comunicarse con Docker Swarm para escalar a varios nodos.

Este orquestador, posee una arquitectura maestro-esclavo (véase figura 4). Cuando se tienen que distribuir tareas en el swarm (enjambre), los usuarios transfieren los servicios al clúster Manager (CM), que actúa de maestro en el clúster.

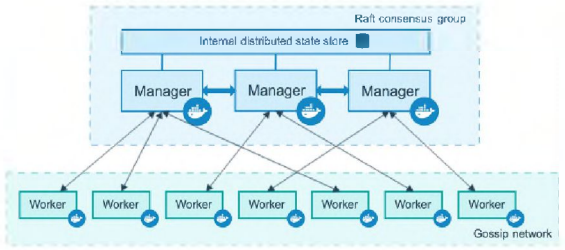

Figura 4: Arquitectura de Docker Swarm

El CM es responsable de la planificación de los contenedores en el clúster y actúa de interfaz primaria a la hora de acceder a recursos de Swarm. La tarea más importante del CM es enviar cada unidad de trabajo (tasks) a los esclavos subordinados (worker nodes).

En cada esclavo funciona un agente el cual, recibe las tareas y entrega al nodo maestro los informes de avance.

Para crear el clúster se debe instalar Docker en todos los hosts y abrir un puerto TCP en cada nodo para la comunicación con el gestor de Swarm. Tras este paso, hay que configurar el contenedor de la imagen Swarm con el rol de agente o gestor.

Una característica muy importante de Swarm es la posibilidad de modificar el número de nodos durante la ejecución gracias a los servicios de descubrimiento.

5. Caso de Estudio

Una de las características más restrictiva de las aplicaciones para trabajar con datos masivos, es la necesidad de procesar ingentes cantidades de datos en un tiempo razonable. Más allá de las técnicas usadas para el procesamiento: paralelas, distribuidas o hibridas, es necesario minimizar los tiempos de acceso a los datos. Para ello, una solución es la replicación, gracias a la cual se puede lograr balanceo de carga, procesamiento concurrente y paralelización

de consultas. Para lograr esto, es necesario contar con una plataforma de hardware que permita orquestar los contenedores con el fin de lograr los objetivos planteados.

En esta investigación se ha trabajado con orquestación de contenedores para el manejo de réplicas en un clúster de nodos virtuales. La idea principal, es realizar la replicación de una aplicación con el objeto de acceder a datos distribuidos en el clúster virtual. De esta manera se podrá acceder desde cualquiera de las réplicas a los datos distribuidos logrando balancear la carga en las consultas y disminuir los tiempos de respuesta.

Para lograr esto, se definió un clúster de máquinas virtuales como primer escenario de trabajo.

La aplicación cuenta con tres pilas de servicios:

- getstartedlab_web (5 réplicas)
- getstartedlab_visualizer (1 replica)
- getstartedlab_redis (1 replica)

Se ha definido una replicación con cardinalidad cinco para el servicio *getstartedlab_web*. Para el acceso a los datos se ha construido una aplicación web escrita en python, el framework flask (incorpora un servidor web) y una base de datos redis. En *getstartedlab_visualizer* se muestran los servicios Docker que se ejecutan en el swarm dentro del cluster, y en *getstartedlab_ redis* se agrega una base de datos redes para almacenar datos de la aplicación.

Toda la configuración específica de cada uno de los contenedores se plasmó en un Dockerfile como lo muestra la figura 5. Con este archivo, se pueden indicar las versiones de la imagen de cada contenedor que el servicio necesita, agregar las librerías faltantes a la imagen, etc.

```
FROM webdevops/php-apache

ENV TZ=America/Argentina/Buenos_Aires

RUN apt-get update \
    && apt-get install php-pgsql -y \
    && echo display_errors=Off >> /opt/docker/etc/php/php.ini \
    && echo short_open_tag=On >> /opt/docker/etc/php/php.ini \
    && echo extension=php_pgsql.so >> /opt/docker/etc/php/php.ini

EXPOSE 80 443
```

Figura 5: Archivo Dockerfile

Para la configuración de los microservicios se usó Docker Compose, el cual es una herramienta para definir y correr aplicaciones que utilicen múltiples contenedores. Las configuraciones de los servicios que la aplicación necesite se especifican en un archivo .yaml. Una característica importanre es que Docker Compose puede trabajar en todos los ambientes ya sea producción, desarrollo o testing. Dado que en este caso de estudio se necesitaba de múltiples servicios, resultó ser de gran ayuda para determinar los puertos de cada microservicio, como así también los parámetros de subred, niveles de replicación, volúmenes espejados, como también la relación de dependencia entre microservicios.

Una vez creados y configurados los contenedores, se crea el clúster de máquinas virtuales, como se muestra a continuación:

```
cker-machine create --driver virtualbox myvm1
cker-machine create --driver virtualbox myvm2
```

Para orquestar los contenedores en los nodos virtuales se selecciona el nodo destinado a ser el manager y se ejecuta el comando que se muestra en la figura 6:

Figura 6: Creación del Manager en Swarm

Posteriormente se levantan las réplicas, como se muestra en la figura 7.

Figura 7: Creación de réplicas

De esta manera queda configura un cluster Swarm con un manager y cuatro worker. A partir de esta configuración es posible acceder a las réplicas desde un front end web.

Es importante aclarar que esta tarea de configuración solo define el contenedor de la aplicación, la cual es la que implemente el almacenamiento, procesamiento o visualización de los datos. En tal sentido, con Docker Compose, se puede armar una aplicación con tantos componentes como se necesite y, posteriormente orquestar la replicación en el clúster.

6. CONCLUSIONES

El exponencial aumento en la cantidad de los datos generados ha exigido el desarrollo de soluciones computacionales que permitan afrontar las necesidades y retos que trae consigo el big data, en cuanto a su volumen, variedad de fuentes y velocidad con que se generan. Con este objetivo es necesario construir aplicaciones escalables horizontalmente, con la capacidad de abstraerse de la complejidad de la plataforma subyacente.

Para lograr este objetivo una alternativa en la construcción de aplicaciones para datos masivos, es migrarse a la arquitectura de microservicios implementados con contenedores. El escalado horizontal, se debe cumplir tanto a nivel de la plataforma como de la arquitectura de hardware, por lo que usar una arquitectura distribuida permite crecer

en la medida que la aplicación lo necesite. Es por ello que la conjunción de microservicios, contenedores y arquitecturas distribuidas es una solución para lograr que el almacenamiento, procesamiento y visualización de grandes volúmenes de datos se pueda realizar en forma eficiente, sin degradar la performance de la aplicación.

En función de estas consideraciones, se puede inferir que trabajar en el desarrollo de aplicaciones para big data de la forma descripta en este trabajo, permite lograr escalabilidad sobre todo en el aprovisionamiento de nuevas réplicas de la aplicación. Esto es fundamental sobre todo cuando se trabaja con aplicaciones que demandan restricciones de tiempo pues permite aumentar o disminuir en forma elástica los contenedores mediante la orquestación.

Por otro lado, es importante destacar que el uso de contenedores permite la interoperabilidad entre todos los microservicios, así como la independencia entre las tecnologías internas de cada microservicio y de tecnologías externas de desarrollo. Este aspecto hace posible la construcción de aplicaciones altamente portables.

7. Futuros Trabajos

El presente trabajo es una primera aproximación a la orquestación de servicios. El grupo de investigación continúa trabajando en el área con el objeto de implementar sobre una arquitectura física distribuida la orquestación.

Por otro lado, se están comenzando a trabajar en ambientes cloud con el objeto de implementar soluciones basadas en Hadoop / Spark dentro de contenedores que se ejecutarán en cluster sobre la plataforma virtualizada. En este sentido la principal idea se direcciona en la implementación de algoritmos de machine learning.

Además, se ha comenzado a trabajar con bases de datos NewSQL (Spanner) y se planea usar Docker junto a Kebertnet para ofrecer el servicio de almacenamiento de grandes cantidades de datos, con una mínima latencia en los query.

Referencias

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," 2011.

[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Futur. Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013.

[3] S. M. Babu, A. J. Lakshmi, and B. T. Rao, "A study on cloud based Internet of Things: CloudIoT," in *2015 Global Conference on Communication Technologies (GCCT)*, 2015, pp. 60–65.

[4] A. Cocchia, "Smart and Digital City: A Systematic Literature Review," Springer, Cham, 2014, pp. 13–43.

[5] A. Katal, M. Wazid, and R. H. Goudar, "Big data: Issues, challenges, tools and Good practices," in *2013 Sixth International Conference on Contemporary Computing (IC3)*, 2013, pp. 404–409.

[6] M. Parashar, X. Li, and Wiley InterScience (Online service), *Advanced computational infrastructures for parallel and distributed adaptive applications*. John Wiley & Sons, 2010.

[7] H.-M. Chen, R. Kazman, and S. Haziyev, "Agile Big Data Analytics for Web-Based Systems: An Architecture-Centric Approach," *IEEE Trans. Big Data*, vol. 2, no. 3, pp. 234–248, Sep. 2016.

[8] A. S. Tanenbaum and M. Van Steen, *Distributed Systems: Principles and Paradigms, 2/E*. 2007.

[9] D. B. Kahanwal and D. T. P. Singh, "The Distributed Computing Paradigms: P2P, Grid, Cluster, Cloud, and Jungle," Nov. 2013.

[10] A. Reuther *et al.*, "Scalable system scheduling for HPC and big data," *J. Parallel Distrib. Comput.*, vol. 111, pp. 76–92, Jan. 2018.

[11] J. Tu, J. Cheng, and L. Han, "Big Data Computation for Workshop-Based Planning Support," in *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, 2015, pp. 1510–1514.

[12] K. Bakshi, "Microservices-based software architecture and approaches," in *2017 IEEE Aerospace Conference*, 2017, pp. 1–8.

[13] N. Dragoni *et al.*, "Microservices: Yesterday, Today, and Tomorrow," in *Present and Ulterior Software Engineering*, Cham: Springer International Publishing, 2017, pp. 195–216.

[14] N. Dragoni, I. Lanese, S. T. Larsen, M. Mazzara, R. Mustafin, and L. Safina, "Microservices: How To Make Your Application Scale," Springer, Cham, 2018, pp. 95–104.

[15] L. Zhu, L. Bass, and G. Champlin-Scharff, "DevOps and Its Practices," *IEEE Softw.*, vol. 33, no. 3, pp. 32–34, May 2016.

[16] N. Kratzke, "About Microservices,

Containers and their Underestimated Impact on Network Performance," Sep. 2017.

[17] R. Morabito, J. Kjallman, and M. Komu, "Hypervisors vs. Lightweight Virtualization: A Performance Comparison," in *2015 IEEE International Conference on Cloud Engineering*, 2015, pp. 386–393.

[18] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2015, pp. 171–172.

[19] P. Sharma, L. Chaufournier, P. Shenoy, and Y. C. Tay, "Containers and Virtual Machines at Scale," in *Proceedings of the 17th International Middleware Conference on - Middleware '16*, 2016, pp. 1–13.

[20] C. Ruiz, E. Jeanvoine, and L. Nussbaum, "Performance Evaluation of Containers for HPC," Springer, Cham, 2015, pp. 813–824.

[21] S. R. Brus, D. Wirasaet, J. J. Westerink, and C. Dawson, "Performance and Scalability Improvements for Discontinuous Galerkin Solutions to Conservation Laws on Unstructured Grids," *J. Sci. Comput.*, vol. 70, no. 1, pp. 210–242, Jan. 2017.

[22] S. Yangui, M. Mohamed, S. Tata, and S. Moalla, "Scalable Service Containers," in *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, 2011, pp. 348–356.

[23] C. Pahl and B. Lee, "Containers and Clusters for Edge Cloud Architectures -- A Technology Review," in *2015 3rd International Conference on Future Internet of Things and Cloud*, 2015, pp. 379–386.

[24] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro, "Choreography and Orchestration: A Synergic Approach for System Design," Springer, Berlin, Heidelberg, 2005, pp. 228–240.

[25] W. Gerlach, W. Tang, A. Wilke, D. Olson, and F. Meyer, "Container Orchestration for Scientific Workflows," in *2015 IEEE International Conference on Cloud Engineering*, 2015, pp. 377–378.

[26] Google, "Kubernetes Documentation - Kubernetes." [Online]. Available: https://kubernetes.io/docs/home/?path=users&persona=app-developer&level=foundational. [Accessed: 08-Jun-2018].

[27] Apache, "Apache Mesos - Documentation Home." [Online]. Available: http://mesos.apache.org/documentation/latest/. [Accessed: 08-Jun-2018].

[28] Netflix, "GitHub - Netflix/conductor: Conductor is a microservices orchestration engine - https://netflix.github.io/conductor/." [Online]. Available: https://github.com/Netflix/conductor. [Accessed: 08-Jun-2018].

[29] Docker, "Set the orchestrator type for a node | Docker Documentation." [Online]. Available: https://docs.docker.com/ee/ucp/admin/configure/set-orchestrator-type/. [Accessed: 08-Jun-2018].

[30] Docker, "Docker - Build, Ship, and Run Any App, Anywhere." [Online]. Available: https://www.docker.com/. [Accessed: 10-Jun-2018].