Accelerating Platform Deployments in the Cloud: A qualitative assessment based on CORD

Kentis, Angelos Mimidis; Soler, José; Broadbent, Adam; Veitch, Paul

Published in:
Proceedings of Seventh IEEE International Conference on Software Defined Systems

Link to article, DOI:
10.1109/SDS49854.2020.9143912

Publication date:
2020

Document Version
Peer reviewed version

Accelerating Platform Deployments in the Cloud: A Qualitative Assessment Based on CORD

Angelos Mimidis Kentis *, Jose Soler *
* Department of Photonics Engineering
Technical University of Denmark
Lyngby, Denmark
{agmimi, joss}@fotonik.dtu.dk

Adam Broadbent †, Paul Veitch †
† British Telecom
Ipswich, United Kingdom
{adam.broadbent, paul.veitch}@BT.COM

***Abstract*— Influenced by the success of cloudification of Information Technology (IT) services, 5th Generation (5G) networks are also expected to leverage the benefits of virtualization and network programmability. Doing so will provide the necessary level of automation and flexibility to meet the stringent requirements of 5G services. Multiple orchestration platforms, tailored towards Telco cloud-based deployments, have been released (e.g., CORD, OSM, OPFNV), which facilitate the integration of virtualization and network programmability. However, these platforms are complex software stacks, making their deployment a time-consuming and sometimes challenging task. This paper presents an investigation of possible deployment best practices using the CORD platform as a use-case. The results show a considerable reduction in deployment time (approximately 67%), without an increase in the complexity of the deployment process. Moreover, this paper includes a list of "lessons-learned" from using the CORD platform. These lessons will hopefully help improve future releases of CORD (and similar Telco cloud platforms).**

Keywords—5G, Platform, KPIs, deployment time, NFV, cloud-native

I. INTRODUCTION

5th Generation (5G) is a highly flexible and programmatic ecosystem in which to add, remove, scale, and version services on-demand to cater for changes in the network. This flexibility will be limited if the platforms which host these services follow static and monolithic deployment workflows. Moreover, these platforms have very complex architectures, which makes their deployment a very time-consuming task. This complexity can act as a barrier to the adoption of such platforms by Telco Operators, which may slow down the roll-out of 5G deployments. During the standardization of the 5G network architecture, many organizations [1] [2] [3] have defined Key Performance Indicators (KPIs) for future 5G deployments. The 5th Generation Infrastructure Public-Private Partnership (5G-PPP) initiative, defined 12 KPIs [4] split across three categories (related to business, societal, and performance aspects). The investigation presented in this paper relates to the second performance KPI (P2), which states that the average service creation time for 5G should be reduced from '90 hours' to '90 minutes'. Reducing the time for service deployment as much as possible is crucial for Telco operators, as it translates to a higher degree of flexibility for their network deployments and time-to-market for new services.

The work presented herein is an outcome of the Next Generation Platform as a Service (NGPaaS) project [5] which argues that this KPI should not be limited to service deployment but should include the deployment of the orchestration platforms which host the 5G services as well. In a sense, NGPaaS argues that platform deployment is another cloud-native service of the 5G ecosystem. The scope of this paper is two-fold; 1) to provide a list of deployment best practices for the Central Office Re-architected as a Datacenter (CORD) platform [6], which can considerably reduce its deployment time and 2) to present with several "lessons-learned" from experience with the CORD platform.

The remainder of this paper is structured as follows. Section II provides background information on the CORD platform. Section III presents deployment best-practices. Finally, Section V provides conclusions on this work together with the lessons learned from interacting with CORD.

II. THE CORD PLATFORM

In a Telco operator's infrastructure, a Central Office (CO) or telephone exchange is the network location at which the subscriber lines terminate and thus connect to the operator's core network. While being an integral part of the Telecom infrastructure for decades, COs impose huge Capital and Operational Expenditures (CAPEX/OPEX) to the operators. These expenses mainly originate from the following reasons:

- Since COs terminate the subscriber lines, they must be placed geographically close to them. , means that a Telco operator will have to administrate over multiple COs (in the order of thousands for big Telco operators [6]).
- The variety of access technologies (landline, enterprise, and mobile), translates to diversity in access infrastructure between COs, which can increase their management complexity.

- Traditionally the CO infrastructure comprises special-purpose and monolithic hardware appliances which suffer from lack of flexibility and programmability. This characteristic of COs can force the Telco operator to over-provision resources, thus increasing the associated costs.

The CORD platform addresses these issues by implementing a new CO architecture, which promotes network programmability, flexible service provisioning, centralized management, and the use of Commercial off the Shelf (COTS) hardware like x86 servers and white-box switches. CORD achieves these characteristics by leveraging the paradigms of Software Defined Networking (SDN), Network Function Virtualization (NFV), and Cloud. In CORD, the network functions are virtualized instances (e.g., Virtual machines (VMs) or containers) hosted on x86 servers and connected via a highly programmable network fabric. Moreover, the deployment and management of any network function on a CO is facilitated by a dedicated and centralized management and orchestration framework. Finally, since CORD is using open and well-defined Applications Programming Interfaces (APIs) a single top-level orchestrator (e.g., ONAP [7]) can manage multiple CORD instances.

CORD comprises three main functional components: the XOS orchestrator [8], the Open Network Operating System (ONOS) [9] SDN Controller (SDNC), and either/both the Kubernetes [10] and the OpenStack [11] Virtual Infrastructure Managers (VIMs). The XOS orchestrator receives service deployment requests from the administrator and orchestrates their deployment on the available networking and computing infrastructure via communication with ONOS and Kubernetes/Openstack. The OpenStack and Kubernetes VIMs receive and fulfill deployment requests for VM-based and container-based Virtual Network Functions (VNF) respectively. Besides deploying containerized VNFs, the Kubernetes VIM is responsible for the deployment process of the CORD platform itself, since after release 6.0 of CORD all its functional elements have been containerized. Finally, the ONOS SDNC is responsible for receiving and fulfilling network connectivity requests from XOS (e.g., connecting two VNFs over the network). Figure 1 illustrates the architecture of CORD version 6.0 and the interactions between its components and actors.

III. Deployment Best Practices

This section presents a series of experiments that demonstrate that CORD can be deployed within the allotted 90 minutes as defined in [4]. As stated in the introduction, reducing the deployment time for both 5G services and platforms is crucial for Telco operators.

The scope of these experiments is twofold 1) to identify dependencies between the hardware in the infrastructure and the deployment time of CORD 2) to identify software deployment

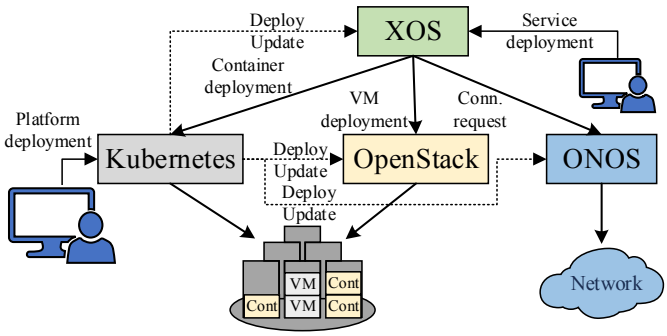

Figure 1: The architecture of the CORD platform

processes that can improve the deployment time of CORD. Through these experiments two primary influences were identified:

1) the effect of the storage medium of the server that hosted CORD in the deployment time, and

2) the method of fetching and deploying Docker images throughout the deployment process of CORD.

For the following experiments, the deployment of CORD was fully automated using Ansible playbooks and Ansible roles. Doing so created a repeatable and reliable testing environment, thus increasing the significance of the collected results. Moreover, all experiments were repeated multiple times and resulted in consistent time measurements. For reference purposes, Table 1 provides the hardware specifications of the server, which was used to deploy the CORD platform.

Table 1: Hardware specifications

CPU	1x Intel(R) Xeon(R) E5-2699 v3 @ 2.30GHz
RAM	32GB
OS	Ubuntu 16.04.06 LTS (4.4.0-131-generic)
HDD	Samsung HD203WI (2000 GB)
SSD	OCZ-VERTEX3 MI (120 GB)

As mentioned, the results highlighted a dependency between the speed of the storage medium and the deployment time of the CORD platform. When comparing deployments of CORD over an SSD and an HDD, there was a deployment improvement of approximately 50% (45 minutes vs. 1 hour 32 minutes). In Figure 2, column 2 and column 4 illustrate these results. Even though this section compares deployments over an HDD and an SSD, it is the actual speed of the storage medium that matters and not the technology itself. For example, an HDD in a RAID array may give similar performance to an SSD.

Docker images comprise individual layers, stitched together by the underlying file system to present a single disk. As part of the image download process, the Docker agent will fetch these

layers in parallel; however, the process of extracting and verifying them is a serial operation. During the experiments, this serial process was a noticeable bottleneck, especially when multiple CPU cores were available. The work presented herein proposes a modification to the deployment process of CORD that can eliminate this bottleneck. Instead of fetching, extracting, and verifying each Docker image when the associated Docker container is required, all Docker images should be prefetched in parallel, before the deployment of any CORD component. 18 different threads were used for this prefetching, with different performance per thread, depending on the actual network capacity utilization towards each source. Doing so translates to parallel processing when extracting and verifying layers belonging to different Docker images, which improves the overall utilization of the available CPU cores. On an SSD-based server, an improvement in deployment time of approximately 32% was noticed (30 minutes vs. 45 minutes). Columns one and two of Figure 2 illustrate this improvement. In contrast, on an HDD-based server, the improvement was only 6.8% (1 hour 26 minutes vs. 1 hour 32 minutes). Columns three and four of Figure 2 illustrate this improvement. A likely explanation for this behavior is that the disk IO was acting as a bottleneck when extracting more than one Docker image.

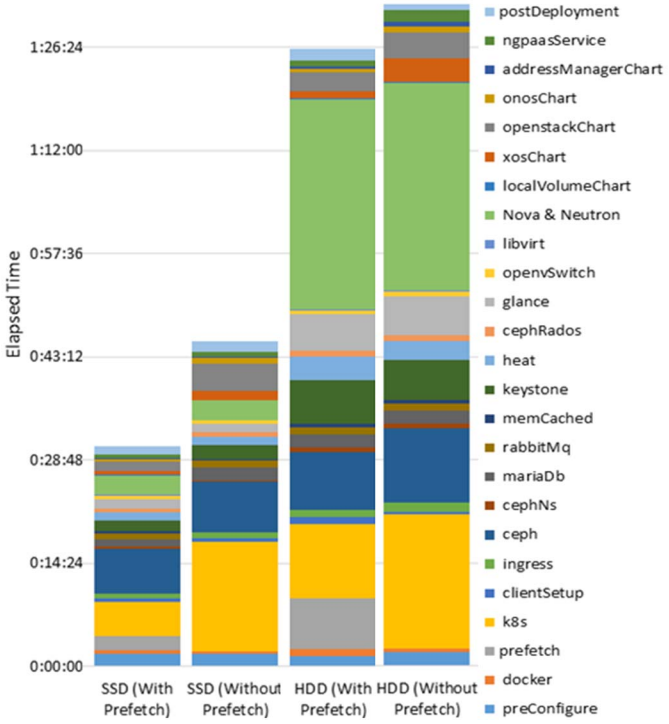

Figure 2: SSD vs. HDD with and w/o image prefetching

While CORD comprises of 4 functional elements (XOS, OpenStack, Kubernetes, and ONOS), on deployment these are decomposed into multiple subcomponents (e.g., Neutron, Nova, Glance for OpenStack). Thus, it is of interest to investigate which of these subcomponents are affected the most by the proposed deployment improvements or have the most significant contribution to the deployment time of the CORD platform. As illustrated by the results of Figure 2, the answer to both questions is the same set of subcomponents, namely the Nova & Neutron modules of OpenStack and the Kubernetes VIM. For example, there is a significant impact on the deployment of the Nova & Neutron subcomponents when deploying over an SSD-based server. The SSD deployment time is approximately 2:43 minutes and the HDD deployment time is approximately 28:00 minutes. Figure 2 provides a comparison between all four experiments (HDD without prefetch, HDD with prefetch, SSD without prefetch and SSD with prefetch). Besides, Figure 2 breaks down the deployment time of each experiment into the individual subcomponents of the CORD platform.

Comparing a worst-case scenario (HDD no prefetching of Docker images) to the best-case scenario (SSD with prefetching of Docker images), an improvement of 66.8% was achieved (30 minutes vs. 1 hour 32 minutes) A final observation is that with these improvements the deployment of CORD falls to 30 minutes, well below the 90-minute threshold set by the 5G PPP KPI.

IV. Lessons Learned and Conclusions

While the following observations are CORD-specific, they could translate to other NFV platforms. This is possible because regardless of their architectural differences, most of the implementations of NFV platforms are based on similar workflows and technologies.

Recent years have seen an increase in the involvement of Telco and cloud operators in open-source projects [12], [13] as well as in the adoption of such projects in production environments [14]. However, this move to open-source components has some issues associated with it. One such issue is unreliable versioning. For example, while CORD versions 4.1 and 6.0 were stable releases, changes in their code-base led to stability issues (e.g., changes merged from the master branch, broke a stable branch). Additionally, in open-source, there is usually an emphasis on functionality and ease of development over performance. Which often means that open-source components are 'fast enough' on their own, but once combined into a complex architecture (e.g., CORD), they aggregate to worse overall performance. Finally, open-source software comes without any technical support., which might imply personal involvement for resolving bugs. On the other hand, the ability to contribute to the code-base implies the possibility to enhance the software with desired features. The CORD community is an example of both cases since they promote personal involvement both for solving bugs and for enhancing CORD with new features.

As introduced in the previous sections, platforms like CORD are very complex, which makes their deployment and management time-consuming, error-prone, and challenging tasks.

However, if they follow a micro-service and containerized architecture and are properly orchestrated (e.g., via Kubernetes), then this complexity can be made transparent to the operator. For example, when comparing CORD version 4.1 and CORD version 6.0 (which introduced container orchestration), a significant improvement in stability and ease of deployment can be observed. However, the use of these many individual open-source components can create hidden/obscure dependencies to the user or developer. For example, an incorrectly exposed change in an underlying program (e.g., the XOS orchestrator API), could lead to cascading failures and obscure errors.

This paper provided a twofold contribution. It provided a list of deployment best-practices for the CORD platform which can reduce its deployment time by 66.8%, (from 1 hour 32 minutes to 30 minutes) placing it well below the KPI defined by 5G PPP. Besides, it listed several "lessons-learned" which came after extensive use of the CORD platform and which can be useful as a starting point for improving future releases of CORD or other similar platforms.

ACKNOWLEDGMENT

This work has been performed in the framework of the NGPaaS project, funded by the European Commission under the Horizon 2020 and 5G-PPP Phase2 programmes, under Grant Agreement No. 761 557 (http://ngpaas.eu).

REFERENCES

[1] "International Telecommunication Union (ITU)." [Online]. Available: https://www.itu.int/. [Accessed: 26-Jun-2019].

[2] "3rd Generation Partnership Project (3GPP)." [Online]. Available: https://www.3gpp.org/. [Accessed: 26-Jun-2019].

[3] "5G Infrastructure Public Private Partnership (5G PPP)." [Online]. Available: https://5g-ppp.eu/. [Accessed: 26-Jun-2019].

[4] D. Kennedy, "Euro-5G – Supporting the european 5G initiative," 2017.

[5] "Next Generation Platfom as a Service." [Online]. Available: http://ngpaas.eu/. [Accessed: 02-Jul-2019].

[6] L. Peterson, "CORD : Central Office Re-Architected as a Datacenter," 2015.

[7] "Open Network Automation Platform (ONAP)." [Online]. Available: https://www.onap.org/. [Accessed: 20-Jun-2019].

[8] "XOS." [Online]. Available: https://wiki.opencord.org/display/CORD/XOS+and+NEM [Accessed: November 2019].

[9] "Open Networking Operating System (ONOS)." [Online]. Available: https://onosproject.org/. [Accessed: 20-Jun-2019].

[10] "Kubernetes." [Online]. Available: https://kubernetes.io/. [Accessed: 04-Jul-2019].

[11] "OpenStack." [Online]. Available: https://www.openstack.org/. [Accessed: 04-Jul-2019].

[12] "OCP Members." [Online]. Available: https://www.opencompute.org/membership/membership-organizational-directory. [Accessed: 08-Jul-2019].

[13] "Board Members of ONF." [Online]. Available: https://www.opennetworking.org/board/. [Accessed: 19-Jul-2019].

[14] "Telefonica and openCORD." [Online]. Available: https://opencord.org/wp-content/uploads/2018/01/Day1_Session5_CORD_build-17-Telefonica-use-cases.pdf.