Insights from SONATA: Implementing and Integrating a Microservice-based NFV Service Platform with a DevOps Methodology

Thomas Soenen*, Steven Van Rossem*, Wouter Tavernier*, Felipe Vicens†, Dario Valocchi‡, Panos Trakadas§, Panos Karkazis§, George Xilouris¶, Philip Eardley††, Stavros Kolometsos¶, Michail-Alexandros Kourtis¶, Daniel Guija‖, Shuaib Siddiqui‖, Peer Hasselmeyer‡‡, José Bonnet** and Diego Lopez[x]

*Ghent University - imec, †ATOS, ‡University College London, §Synelixis Solutions, ¶NSCR Demokritos, ††BT, ‖i2CAT Foundation Barcelona, ‡‡NEC Europe Ltd. Heidelberg, **Altice Labs, [x]Telefonica I+D

***Abstract*—In pursuit of a flexible, resource efficient and high-performant 5G infrastructure, many operators, vendors and research consortia are currently developing, testing and integrating their NFV platform with associated management and orchestration (MANO) functionality. The SONATA NFV platform follows a micro-service design, which involves a tight coupling between an SDK, monitoring and MANO functionality, targeting a secure and stable software foundation. This experience paper gives a thorough overview on the encountered challenges, insights and resulting learnings when implementing and integrating the SONATA Service Platform using a continuous integration and delivery DevOps methodology. This is the result of a strong cooperation between prominent equipment vendors, network operators, software companies and universities, providing a set of constructive recommendations in hope of catalysing the development and deployment of NFV platforms.**

I. Introduction

In the next years, 5G infrastructure will become a ubiquitous, flexible, broadband and programmable network that will be in the core of every social, business and cultural process, enabling both economic growth and social prosperity. However, the 5G vision poses significant technical challenges that must be fulfilled, including the concept of agile programmability and the introduction of management mechanisms for the efficient instantiation of network services across heterogeneous network components, virtualised infrastructures and geographically dispersed cloud environments.

In an effort to address these challenges, multiple consortia are trying to build Network Function Virtualization (NFV) service platforms (SP) and Management and Orchestration (MANO) frameworks. These technologies aid in the 5G adoption by increasing network programmability. One such service platform has been designed and developed in the scope of the EU 5G-PPP SONATA project. SONATA's service platform [1] was designed to satisfy the need for flexible and extensible management operations, so it allows telecom operators and communication service providers to cope in a world with rapidly changing technological trends and newly introduced business models. To this end, SONATA made the following design choices for its service platform: i) An extendable plugin-based MANO architecture implemented through microservices allowing the owner to alter its behavior by adding and replacing plugins on the fly and ii) an infrastructure abstraction allowing the service platform to orchestrate multiple Virtual Infrastructure Managers (VIM).

Inspired by common best practices for software development within large, distributed teams, the SONATA service platform was developed following a continuous integration and delivery (CI/CD) methodology [2]. Such methodologies focus on improving the software quality, while decreasing development time and the gap between developers and the operational deployment of the product. By leveraging tools such as GitHub [3], Jenkins [4] and Docker [5], and by implementing multiple verificiation layers, SONATA created a CI/CD pipeline that allowed developers to frequently update and publish their code to a whole ecosystem of developers and admins, while preventing code updates that break the integration from blocking others. Due to a DevOps [6] approach, developers quickly receive feedback on their changes, allowing for a more agile development and increasing the frequency of software iterations.

In this paper, we describe insights and gained experiences from developing and integrating SONATA, a flexible and extendable NFV service platform. This comparison of theory and practice serves as aid for those considering the implementation of a plugin-based MANO framework or the adoption of an agile CI/CD methodology for a development project in the telco environment. Section II focuses on the experiences gained from developing the service platform, while in Section III, we detail our CI/CD methodology and the insights we gained from setting it up and using it. Finally, Section IV concludes the paper and provides some general recommendations for developing and integrating NFV service platforms.

II. Insights From implementing the SONATA Service Platform

The SONATA NFV service platform provides the virtualisation infrastructure, as well as the management and orchestration functionality to deploy NFV services. The SONATA SP is closely modeled after the ETSI NFV model. Next to the platform itself, SONATA provides a set of development tools (SDK) to assist the developer in developing and packaging NFV services. This section is dedicated to insights gained over

the course of implementing the SONATA service platform, which are applicable to the design and implementation of general flexible NFV orchestration platforms.

Learning 1: The NFV Orchestrator strongly benefits from a task-oriented implementation. To be in accordance with the ETSI functional design for MANO frameworks [7], a subset of the SONATA service platform implementation included the development of an NFV Orchestrator (NFVO). The NFVO is the part of the MANO framework that manages and orchestrates everything on the level of the network service, by implementing a range of workflows such as instantiating, scaling or terminating a service. Figure 1 shows which SONATA components are involved in the NFVO. The workflows are implemented by the Service Lifecycle Manager (SLM). Looking at the service instantiation workflow, the SLM uses the placement plugin to calculate the placement, the Function Lifecycle Manager and Infrastructure Adapter to deploy VNFs and a storage component to save their records. Due to the variety in management and orchestration requirements from services, a straight-forward implementation of these workflows quickly became complex. Different services require different placement algorithms, different scaling solutions and in SONATA, services might come with Service Specific Managers (SSM), processes provided by the service developer that customise the SLM workflow. Implementing each workflow as a single process/thread overloaded the code with if-else clauses and boilerplate, and made extending them a complex task. This made the SLM an inflexible object, contradicting the SONATA requirement for a flexibility and extensibility. To this end, we re-factored the SLM into a task-based engine. The overall functionality of the SLM was chopped into basic tasks, with each task implemented as a seperate thread of control. Workflows are then established by chaining a subset of these tasks together into an ordered schedule, in accordance with the MANO requirements for the service. SSMs can now customise a workflow by overwriting the functionality of one or more generic tasks, or by adding/removing tasks in the schedule. Extending the functionality of a workflow (e.g. interacting with a newly added SP plugins) in the SLM now comes down to implementing new tasks, and inserting them in the workflow schedule. The complexity of the code base reduced, making the SLM more easily maintainable and extendable. This transition made the SLM, and thus the NFVO, the flexible component SONATA requires it to be.

Learning 2: Interfacing a Service Development Kit to a Service Platform requires a strong integration. An SDK is a design environment that provides a range of tools to aid developers of NFV-based services. Such an SDK was developed as part of SONATA, and in order to increase its DevOps capabilities, we learned that it must take into account the specific APIs and input formats expected by the SONATA service platform. Therefore, the SDK requires an explicit linkage to the targeted service platform. The main SDK features regarding validation, profiling and packaging of a network service that are being developed require the implementation of two custom-built interfaces to the service

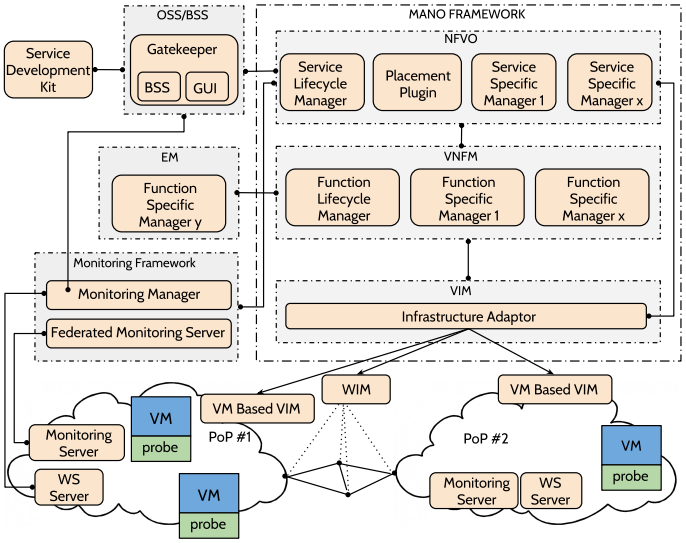

Fig. 1: Mapping of SONATA service platform modules on ETSI MANO framework architecture.

platform: (i) An interface to on-board the service package and deploy it in either a test or production resources of the SP. Instead of using an isolated test environment inside the SDK, a better and more DevOps approach is to use the SP's orchestration framework directly. This way, it is not needed to fully replicate the service deployment functionality inside the SDK and services get tested directly in the service platform they are supposed to run on. (ii) An interface through which monitoring data that is collected by the SP's monitoring framework can be fed back to the SDK and the developer for further analysis that may improve the performance of the service. Both test or production environment can have their own, developer customised, set of metrics collected to export. In the SONATA service platform, the Gatekeeper is the module that mediates interactions between the service platform and the outside world, including these interfaces with an SDK. The Gatekeeper authenticates and authorises all requests from developers to instantiate services and to obtain monitoring data.

Learning 3: The monitoring framework needs to be flexible and scalable. A service platform will collect information about many aspects of its performance - the individual elements of the physical and virtual infrastructure, the VNFs, the network services, and so on. Thus, monitoring information is of interest to several parties in the ecosystem, including: the operator of the service platform (to gain a deeper understanding about their service); the end customer (to check that their service level agreement is being met); vendors (who are responsible for various VNFs); network service developers; machine learning specialists (to optimise algorithms that use monitoring data to automatically identify and isolate faults). These factors suggest that the monitoring framework should have two complementary features, it should be flexible and scalable.

Flexibility is needed as different parties will be interested in different information. E.g, an operator needs real time infor-

mation about all the different components; a customer is only interested in summary information. So the framework needs to allow the interested party to describe what they want, and for the owner of that information to approve, adjust or reject the request. In our implementation, the monitoring framework collects and processes data from several sources (VMs, Docker containers, OpenDayLight controllers, OpenStack API, etc.), providing the interested parties the ability to activate metrics and thresholds to capture infrastructure or service-specific performance data. The user can define rules based on metrics gathered from multiple VNFs deployed in one or more NFVIs. In general, the user can subscribe to a message queue to get the real time alert notifications and monitoring data, request them through a RESTful API or directly access them through a web socket URL.

Scalability is needed because there is potentially a vast amount of monitoring information, so it cannot all be communicated or stored in full. Approaches that help scalability include: thresholds (only report a metric when its value exceeds some value); averaging, filtering and other aggregation techniques (e.g. averaging information over some time period); creating tailored alerts (for instance, so a help desk can be pre-warned that there is a problem affecting a service); and an emergency button so the service/network manager can quickly reduce to 'skeleton monitoring' (e.g. if some failure means that monitoring data suddenly consumes a significant fraction of resources). To address scalability, several monitoring components in our implementation had to be distributed across NFVIs. First, each NFVI needs its own web socket server to accommodate users requests for streaming monitoring data. Second, monitoring servers follow a federated architecture. The local servers collect and store metric data from the VNFs deployed in the NFVI, while only the alerts are sent to the federated level for further processing/forwarding to the user. Third, alerting rules and notifications can be based on monitoring data from multiple NFVIs and thus should be evaluated on a federated level. To enable 'skeleton monitoring', the design of the monitoring probe allows dynamic modification so that in cases where the difference of a monitored metric is below a threshold, it will not be sent to the monitoring server.

Learning 4: Security for a microservice-based architecture is best obtained with a token scheme. In the SONATA service platform, the different modules/plugins are implemented as microservices. Microservices are implemented by standalone processes which communicate through remote network calls, such as RESTful APIs or message brokers. To secure this communication, we came to the conclusion that a session based authentication did not fit with requests between different microservices. To this end, the User Management tool in the Gatekeeper was enhanced to provide a token-based authentication mechanism (JWT [8]) for the service platform microservices, just like it does for the end-users. This way, we benefit from a simple and self-contained data way for authentication and authorisation, maintaining statelessness in the platform. Tokens are compact, should be strong enough, fit well in APIs and should be safely transported, to allow for a secured exchange of information between endpoints. They include digitally signed information that can be verified and trusted. The User Management applies the same token mechanism to create User Accounts for end-users and Service Accounts for internal microservices. This approach keeps workflows simple: they both register to the platform, log-in successfully to get a token which grants access for a certain amount of time. Then, end-users and microservices can transmit information including the token, which is used by the User Management to extract and evaluate the rights. The interaction with the User Management server is minimised, avoiding overheads that session data or state would involve.

Learning 5: The concept of Network slicing is still not clear. Within the 5G concept a large amount of effort is focusing on architectures that support network slicing, implying evolution from the network sharing models towards network isolation, multi-tenancy and end-to-end resource provisioning and guarantees. In SONATA, slicing is considered at the lower service platform layer via the MANO framework, which leverages the IA and Slice Management components, and a distributed monitoring framework. Finally, issues related to service provider peering and recursive operations of the service platform (where slicing is also employed) is tackled by the Gatekeeper. In this context, SONATA considers an SDN capable WAN, managed by a WAN Infrastructure Manager (WIM) that supports slicing (by means of provisioning of isolated multi-tenant networks) plus an OpenStack based VIM integrated with an SDN controller (i.e. OpenDayLight) for the physical and network elements within the NFV infrastructure. In this view the SONATA service platform is able to create per domain slices that are interconnected constituting an end-to-end isolated network and computing resource slices.

The experienced limitations learned by the activities related to slicing are summarised below:

- Infrastructure (domain) operators are using different QoS mechanisms and mappings;
- Different traffic isolation mechanisms are being used, e.g. IP/MPLS and MetroEthernet;
- The heterogeneity of infrastructures, e.g. NFVIs where PoPs are established versus WAN and edge networks;
- The lack of ways to express and enforce SLAs when they imply virtualised resources;
- end-to-end slices are difficult to include at the very edge of the network as the technologies (e.g. Radio resources) are still maturing;
- Operators expose only partial and limited views of their infrastructure topology and technology enablers, creating the need for a consistent and standard way of describing logical topologies related to the provided service [9].

Learning 6: Service platforms cooperate with a hierarchical, recursive architecture. Often delivery of a service to a customer will need the involvement of more than one service platform. For example, the customer may want the service in multiple geographical locations, and no operator is present in them all. Another example is where some operators specialise

in end-customer-facing operations, whilst others specialise in the ”wholesale” provision of infrastructure, or in providing specific types of VNF.

We believe that cooperating service platforms should be organised in a hierarchical architecture, meaning that an ”upper-SP” provides the end-to-end service to the customer, and it chooses to involve a ”lower-SP” to deliver part of the required capability (in other words, they have a ”north-south” rather than ”east-west” relationship). From the customer's perspective, they only interact with, and know about, the upper-SP; from the upper-SP's perspective, the lower-SP is providing a component in their overall network service in a similar manner to the NFVI; and as far as the lower-SP is concerned, the upper-SP is just another customer requesting a service. Further, we believe that the architecture should be recursive, meaning that the lower-SP can in turn arrange for some of the service it provides to be delivered by a yet-lower-SP (and so on). The advantages of such an approach are commercial and technical: it has clear lines of responsibility, allows autonomy and flexibility in service provision (e.g. different SPs could use different orchestrators), and only a single, standardised ”north-south” API is needed.

A couple of issues concern capabilities for discovery and addressing. The upper-SP needs to learn what capabilities can be provided by potential lower-SPs. Our current thinking is that this is best done by the lower-SP publishing the capabilities it can offer (similar to today's Suppliers' Information Notes about network services [10]), instead of a query protocol for instance. On addressing, we need to ensure that packets can flow along a service function chain that spans the SPs. At the moment, we think the most likely approach is that the upper-SP tells the lower-SP constraints, so that the lower-SP makes a good choice about the virtual link address and port identifier that it uses for the VNF(s) it supplies.

Learning 7: Docker based VIMs are not yet mature enough for Service Function Chaining. SONATA set out to orchestrate network services on multiple different VIMs. Due to the growing adoption of Docker, we opted for a Docker based VIM in addition to the more commonly used Virtual Machine based VIMs. Docker [5] is a container based virtualisation mechanism for guest operation systems that is much more lightweight than Virtual Machines (VMs), and can be faster built, tested and deployed. As VNFs are being implemented in both VMs and containers, SONATA targeted to combine container based and VM based VNFs in the same network service. To abstract the different APIs from the heterogeneous VIM landscape, an Infrastructure Adapter was introduced in the SONATA service platform to provide a streamlined API towards the MANO framework, as is depicted on Figure 1. Where VM based VIMs, with OpenStack [11] the most used, have matured when it comes to NFV use, we came to conclude that Docker based VIMs are still missing some critical features. Both Kubernetes [12] and Docker Swarm [13], two of the major Docker based VIMs, fail to provide complete isolation between the deployed containers, introducing security concerns. Docker based VIMs have been designed to host end-point services, causing them to lack features when it comes to hosting containers that are along the data path. One of these is that it is unclear how to integrate Service Function Chaining with the Kubernetes networking capabilities, which is a critical shortcoming when thinking about orchestrating NFV services. Efforts to bridge this gap are being made, e.g. by Multus [14]. Due to these missing features, SONATA was unable to combine VM and container based VIMs in the same service.

III. Continuous Integration and Delivery

This section is dedicated to the continuous integration and delivery methodology that was applied during the development of the SONATA service platform. This way of working aims to improve the quality of the software, reduce the time-to-market of the product and streamline integration actions between different modules of the software. It allows each developer to publish software updates multiple times a day, while assuring that the code integrity is never compromised. To achieve this, our methodology combines a set of software tools with multiple layers of testing to guarantee that each contribution is validated before it enters the master branch of the software. These layers of testing emulate an operational deployment of the service platform, creating a DevOps cycle that provides developers with fast feedback on how the service platform performs in an operational environment.

As the different modules in the service platform are implemented as microservices, we opted to use Docker containers to package each module. The developer defines a Docker descriptor that contains the base image of the programming language used for the module, a path to the code that is going to be included inside the container, and all the dependencies needed for the code. Based on this descriptor, the Docker Engine generates the docker container, which can be deployed on any device that is hosting the Docker Engine. By having the entire SONATA service platform code base implemented in such docker containers, deploying the platform becomes real easy, quick, flexible and platform independent.

A. The CI/CD Pipeline

The CI/CD pipeline can be categorised, as shown on Figure 2, in three phases: development, integration and qualification. In the development phase, the developer updates the code, creates the docker containers that package this code and expose them to the first line of tests, the unit tests, which runs in the developer's local environment and provide the first feedback. These tests are designed to verify the code in an isolated environment. In SONATA, we opted to use GitHub as source code management tool that uses the git version control protocol. Once the code passes the unit tests, the developer can commit the code to the GitHub repository and make a pull request which is a petition to the repository owner to add the new code to the master branch. At this point, the selected continuous integration environment, Jenkins, kicks in. Jenkins provides a range of functionalities that make it easier to set up an integration environment for the software, and to keep track

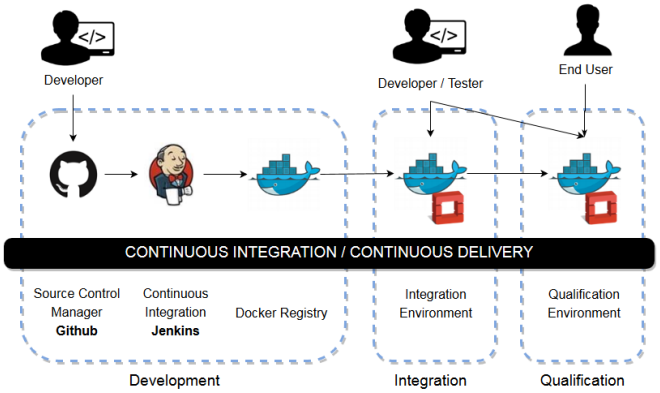

Fig. 2: Continuous Integration and Delivery pipeline.

of its quality. Once a developer makes a pull request, Jenkins will fetch this updated code, generate the new containers with the update and verify they pass the unit tests. Once Jenkins approves the update, the owner of the repository can accept the pull request and the code becomes part of the master branch. Jenkins will also upload the updated containers to the SONATA docker registry, making it available for the next phase. The development phase prevents code that fails the unit tests from becoming part of the integrated software, breaking it and blocking other developers.

In the integration phase, the containers of all the modules are deployed and exposed to the whole ecosystem of developers and admins as an entire service platform. This allows each developer to test and validate how their module integrates with the SP. The different containers are exposed to a range of integration tests that validate whether they are integrated correctly. These tests expose mismatches in APIs between modules and validate whether the integrated behaviour matches the expected one. Integration tests are automatically triggered when Docker containers are updated, others are executed every day at midnight. This tight schedule is in accordance with our DevOps approach. By checking the influence a code update has on the overall integration almost immediately, developers quickly receive feedback on their change. This allows for quicker iterations of the code and a faster progress of the overall functionality.

A new qualification phase is triggered every time all the integration tests succeed. At this point, all the Docker containers are promoted to our qualification environment, which is guaranteed to have a correctly integrated version of the platform. Therefore, it can be used to evaluate the quality of the service platform, e.g. by exposing it to performance, security or stability tests. These tests show us how many requests the platform can manage at the same time, how long it takes to satisfy requests, how the platform behaves when used over a longer period of time, etc. The qualification tests emulate the exposure of the service platform in an operational environment, creating another DevOps feedback loop to the developer. Like the integration phase, Jenkins organises the qualification phase by automatically deploying the qualification version of the platform and scheduling the tests. Due to its stability, the qualification environment can also be used for demo purposes or to let possible end-users test the service platform.

B. Insights from setting up and using the CI/CD methodology

Both processes of setting up the CI/CD pipeline and using it yielded some insights.

Learning 8: Selecting the right software tools at the beginning of the project is crucial: i) to protect the DevOps approach, and ii) backtracking from selected tools is expensive in terms of time and effort. It is important to automate large parts of the CI/CD pipeline, allowing the developers to focus on code development and provide them with quick feedback. To this end, a set of software tools such as GitHub and Jenkins were selected. We learned that this selection process is of the highest importance, as the selection of the wrong tool might conflict with the DevOps goals of the methodology. At one point shortly after the beginning of the project, we adopted OWASP [15], a tool that analyses code for security risks by scanning it for vulnerabilities, performing penetration testing, etc. Extending the Jenkins job that evaluates pull requests with an additional OWASP code check exponentially increased the time it took to analyse the updated code. The addition of OWASP significantly increased the duration of the integration cycle, contradicting our DevOps objective to quickly provide the developer with feedback, which led us to optimise the pipeline and use OWASP in a parallel job to not interrupt the first code check iteration. A learning period should be provisioned to allow developers to discover the selected software tools. This puts additional stress on the software tools selection process, as adopting a new software tool in a later stage of the project leads to a new learning period, which is both costly in terms of effort and time, especially in a distributed consortium like SONATA.

Learning 9: A good CI/CD and DevOps methodology allows for quick detection of design issues. As our CI/CD pipeline performs integration tests from the beginning of development, we were able to detect design issues very fast. For example, early in the project we identified a mismatch between the designed schemas, i.e templates, for service and VNF descriptors and the descriptor data required by the service platform to correctly orchestrate the VNFs. Detecting descriptor issues late in the project would have implied a huge effort to correct, as already developed services for e.g. pilots and SDK tools that aid in the development of such services would need to be changed.

As it is though to detect such design issues before an integration cycle and the cost in terms of time and effort of fixing such issues late in the development process is high, we feel that our CI/CD methodology allowed us i) to keep a clear and complete view of the status of the project at all times and ii) to meet software delivery deadlines in an environment of limited resources. While it gives no guarantees about the quality of the software, our methodology makes us confident about its stability and reliability, since it endured numerous

cycles in the integration and qualification environment before it was released.

Newly introduced features, even late in the project, quickly became stable. For example, the authentication and authorisation feature for service platform microservices was lately introduced through the User Management concept. It was supposed to have a big impact on our integrated platform, as it added a new security wall between its components, demanding extra adaptation. Following the CI/CD loop, it was successfully integrated through a continuous adaptation from the components interfaces with no major impact. Once this feature was deployed, it started to gain stability while it was enhanced with additional features such groups, roles and permissions.

Learning 10: There is no clear benchmark that indicates when NFV service platforms are ready for operational deployment. Due to our CI/CD pipeline, we feel confident about the stability and reliability of SONATA's latest 3.0 release[1]. The service platform has been vetted by a range of integration and qualification tests over a significant period of time before the software was released. It is however a non-trivial task to determine when the platform is ready for an operational roll-out, i.e. which performance, stability and reliability tests constitute a good enough test base. Such requirements should entail a range of open-access network services and VNFs that can be used for testing and bench marking, which go beyond minimal test setups with a single PoP, a service with two VNFs, few chaining and no scaling. Such requirements can create an environment where MANO framework performance can be compared against other frameworks and against operational readiness.

Learning 11: Maintenance and updating of the CI/CD pipeline takes priority over code development. The success of the used CI/CD pipeline is directly correlated with its care and enforcement. In the SONATA project, a significant subset of the software developers was also responsible for the maintenance of the integration and qualification environment, causing the description and updating of new tests to slack at times. When the outcome of integration tests is ignored or they are not updated when new features appear in the software, the DevOps feedback loop is lost causing integration issues to appear in a later stage of the development. It is therefore of the utmost importance that the CI/CD pipeline is strictly followed and enforced, and gets prioritised over code development during every stage of the project.

IV. Conclusions and recommendations

Implementing, integrating and deploying the SONATA NFV platform proves to be a moving target. This is the result of the proliferation of NFV concepts, architectures, technologies, platforms, and standards in flux. The SONATA approach is tackling this challenge through the development of an SDK and a MANO platform using a microservice design and a DevOps approach involving continuous development and integration. However, to truly advance the area of NFV platforms, experiences of this undertaking provide a couple of recommendations. A range of NFV-related concepts and models need harmonisation. We argue that the service programming model and descriptor formats need to be unified in the community, reducing the dependency on existing cloud models. The development and adoption of NFV technology would drastically benefit from such a common model, together with openly available network functions and services, as well as commonly agreed functional and non-functional benchmarks. This would allow to overcome the situation where most NFV platforms provide only support for simplistic NFV services, or rehashing existing cloud functionality without any telecom-specific performance guarantees. Next, as performance and overhead are of extreme importance when focusing on telecom-specific packet processing tasks on the NFV execution platforms, the community needs increased focus on enhancing VIM technology to provide native network control for the interconnection of container instances. In addition, the SONATA project confirms that NFV MANO architectures can be made significantly more reliable, and scalable, when following a microservice architecture, as well as including a task-based NFVO. Scalability and autonomy in orchestration is drastically improved when supporting hierarchic NFVO setups. Also, when focusing on the software development and integration methodology, a continuous development and integration strategy has the potential to significantly increase the stability, reliability and thus operational readiness of an NFV platform, provided that adequate (re-)education and reinforcement on this approach is enforced on a regular basis.

[1]https://sonata-nfv.github.io/

Acknowledgement

This research has been partly funded by the European Commission H2020 5G-PPP project SONATA (671517). The views expressed here are those of the authors only.

References

[1] S. Dräxler *et al.*, "Sonata: Service programming and orchestration for virtualized software networks," in *IEEE ICC conference*, 2017.
[2] M. Shahin, M. A. Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," *IEEE Access*, vol. 5, 2017.
[3] sep 2017. [Online]. Available: https://github.com/
[4] sep 2017. [Online]. Available: https://jenkins.io/
[5] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, 2014.
[6] A. Balalaie *et al.*, "Microservices architecture enables devops: migration to a cloud-native architecture," *IEEE Software*, vol. 33, no. 3, 2016.
[7] ETSI, "GS NFV-MAN 001 V1." 2014.
[8] M. Jones *et al.*, "Json web token (jwt)," Tech. Rep., 2015.
[9] T. Soenen *et al.*, "A model to select the right infrastructure abstraction for service function chaining," in *IEEE NFV-SDN conference*, 2016.
[10] sep 2017. [Online]. Available: btplc.com/sinet/Newlyaddeddocuments
[11] sep 2017. [Online]. Available: https://www.openstack.org/
[12] E. A. Brewer, "Kubernetes and the path to cloud native," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*. ACM, 2015.
[13] sep 2017. [Online]. Available: https://docs.docker.com/engine/swarm/
[14] M. Siddiqui *et al.*, "Enabling new features with kubernetes for nfv," Intel, Tech. Rep., 2017.
[15] sep 2017. [Online]. Available: https://www.owasp.org/