Marie Skłodowska-Curie Actions (MSCA)

Research and Innovation Staff Exchange (RISE)

H2020-MSCA-RISE-2017

Intelligence-Driven Urban Internet-of-Things Ecosystems for Circular, Safe and Inclusive Smart CITIES

D3.4: Integrated and self-adaptable infrastructure and real time adaptation

Abstract: This deliverable includes state of art report of the adaptive infrastructure in the context of the monitoring, networking, and deployment of cloud-based microservice applications, and the infrastructure requirements for the IDEAL-CITIES prototype.

Contractual Date of Delivery	30/06/2020
Actual Date of Delivery	28/02/2022
Deliverable Security Class	Public
Editor	Jakub Rola
Contributors	FORTH, BU, BLS, NPS

The *IDEAL-CITIES* consortium consists of:

FOUNDATION FOR RESEARCH AND TECHNOLOGY -HELLAS	FORTH	GR
ECOLE NATIONALE DES PONTS ET CHAUSSEES	ENPC	FR
BOURNEMOUTH UNIVERSITY	BU	UK
BLUESOFT SPOLKA Z OGRANICZONA ODPOWIEDZIALNOSCIA	BLS	PL
DGS SPA	DGS	IT
NODAL POINT SYSTEMS	NPS	GR

This project is supported by the European Commission under the Horizon 2020 Program (2014-2020) with Grant agreement no: 778229

Table of Contents

List of Abbreviations

CE	Circular Economy
SDN	Software Defined Networking (or Networks)
CVSS	Common Vulnerability Scoring System
LCA	Life Cycle Assessment
HTTP	Hypertext Transfer Protocol
NVD	National Vulnerability Database
PBX	Private Branch Exchange
IoT	Internet of Things
DPI	Deep Packet Inspection
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
ML	Machine Learning
AI	Artificial Intelligence
QoS	Quality of Service
CAPEX	Capital Expenditure
OPEX	Operational Expenditure
NFV	Network Function Virtualization
MANO	Management and Orchestration
NFVI	Network Function Virtualization Infrastructure
VNF	Virtual Network Functions
VM	Virtual Machine
PNF	Physical Network Function
SFC	Service Function Chain
EM	Element Management
VIM	Virtualized Infrastructure Manage
OSM	Open Source MANO
NSD	Network Service Descriptor
VL	Virtual Link
VNF-FG	VNF Forwarding Graph
SC	SDN Controller
SBI	Southbound Interfaces

NBI	Northbound Interfaces
VTN	Virtual Tenant Networks
ODL	OpenDaylight
STP	Spanning Tree Protocol
MST	Multiple Spanning Tree Protocol
LAN	Local Area Network
VLAN	Virtual LAN
OSI	Open Systems Interconnection
TRL	Technology Readiness Level
K8s	Kubernetes

1 Introduction

Infrastructure is the silent and invisible hero in everyone's life. It is the background that we are used to and we only notice it when it stops working properly. Infrastructure in the IT world comprises of computing centres, network hardware such as hubs and switches, telecommunication networks, and database centres. But, IT infrastructure is also embedded in civil infrastructure like roads, buildings and internal infrastructures like heating and cooling systems.

In this document, we will describe the next layer of the IT infrastructure, necessary for the deployment and operation of cloud-based applications. This layer originates from the aforementioned layers of infrastructure; however, it is more abstract as it contains mainly software components.

This document is divided in four main sections. The first three describe the monitoring, networking, and administration aspect of the infrastructure. The last section contains the specific requirements for the IDEAL-CITIES platform.

2 Monitoring

2.1 Introduction

Monitoring is a critical feature and process that can potentially prevent a system from leaving desired safe states or in the event that failures manifest, it can provide actionable intelligence that would allow an administrator to take corrective actions. Fully integrated monitoring, from a control systems perspective, involves feedback loops where output diversions from a prescribed envelope are observed in real-time. As such, monitoring is a quintessential activity that, when implemented through feedback loops, can trigger changes in the configuration and interaction of system resources. In essence, this delivers the desired level of real time adaptation concept.

From a Circular Economy (CE) perspective, monitoring, and subsequently adaptation, is the key dimension of a data driven approach. The monitored well defined properties of location, condition and availability of an asset can in turn be used to fine tune the circularity objectives of the system, in line with the underpinning CE[1] enabled business models. This approach has been further elaborated and specified in deliverable D2.3 where the circularity patterns are formally defined through pattern rules.

In this section, a description of the current state of the art of monitoring a participatory sensing smart city platform such as the one developed for IDEAL-CITIES with an emphasis on circularity, is presented.

2.2 State of the Art in monitoring for adaptable services

The first level of distinction on monitoring is the level of abstraction. Adopting a bottom-up approach, at the operational assets layer, data exchange happens at the “highest” speed, where the system components interact within fractions of seconds and decisions are made in real time. Moving up the monitoring stream, the management layer witnesses a higher degree of aggregation and as such the data speed is considerably “lower”. At the outliers of the monitoring ecosystem lie audit activities which are essentially performed in non-real-time and mostly in a post-mortem manner.

In order to illustrate the importance of monitoring in a smart and circular city, we first need to consider the capabilities and features of a smart city. In fact, within the IDEAL-CITIES context, we perceive a smart city as the penultimate state of a city that strives to become circular. We consider a city’s maturity model (as specified in D2.1) as depicted in Figure 1. We argue that data-driven CE is reached at the responsive city level, a level where a city can fully respond to stimulus by dynamically reallocating its resources and reconfiguring its major structures.

[1] https://circulareconomy.europa.eu/platform/sites/default/files/categorisation_system_for_the_ce.pdf

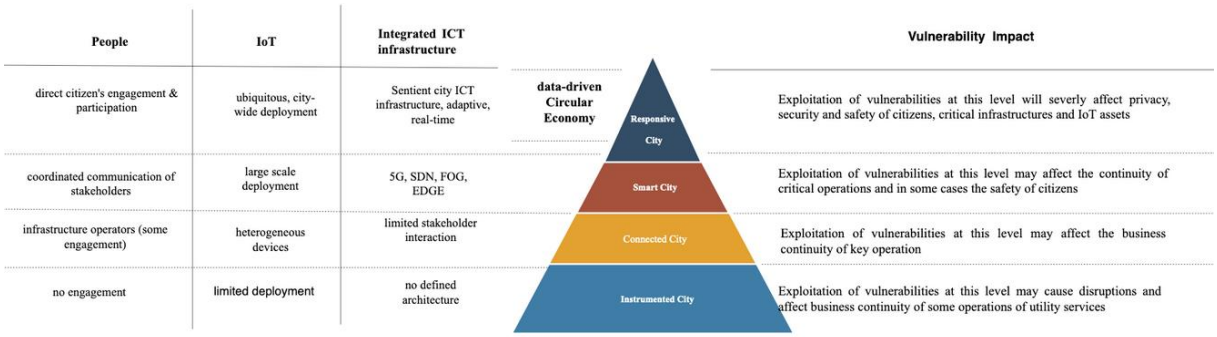

Figure 1. A maturity model for a smart, circular city (extended from D2.1)

In this figure, we also juxtapose the impact of hardware and software vulnerabilities that may exist within the city's ICT infrastructure. We argue that the impact of a given vulnerability is potentially higher, as a city progresses in the maturity pyramid. A representative example is Software Defined Networks (SDN), where the network resources and topology are fully defined and controlled by software. As such, the impact of a vulnerability being exploited at a particular maturity level is expected to increase, as one traverses from a lower to a higher level. This suggests that a vulnerability will carry a larger amount of risk, the higher the maturity level this vulnerability will be placed on. More formally, if $R_m(.)$ is some risk calculation function with m denoting the maturity level, then for a particular vulnerability v:

$$R_i(v) \leq R_j(v)$$

the risk is lower for $i<j$. This means that if two cities have an identical vulnerability profile, the city with the highest maturity will also have the highest risk. This is captured in some quantitative vulnerability measurement systems such as the Common Vulnerability Scoring System (CVSS) [1][2], where the resulting severity of a vulnerability can be adjusted and subjected to the so-called environmental variables. To the best of our knowledge, the development of a methodology for producing a suitable vulnerability scoring system with an environmental metric group for a smart city infrastructure remains a research challenge, see for example [3] where although a concise threat and risk model for smart cities is proposed, there is limited evidence on how the environmental CVSS score dimension is populated. In fact, the CVSS measurement system itself has received criticism from research and practitioner communities [4] with alternatives being proposed (see for example [5]). However, despite its shortcomings, it has still been applied to and validated through a variety of contexts and scenarios [6]. In any case, the two main characteristics of a risk-based approach for a vulnerability driven assessment system would require a vulnerability measuring system and a risk methodology that considers interdependent networks [7].

The above narrative, using vulnerabilities as a vehicle to demonstrate the need to monitor the assets and infrastructure elements (in the above example for security purposes), shows not only the importance of monitoring in general, but its significance in a data driven CE context.

Non-surprisingly, the data-driven CE maturity level of a smart city can be seen as a convergence of the security and resilience properties as expressed by the circularity patterns.

Specifically, the *Condition* property can inform the reliability and resilience of the infrastructure. Consider for example the tree-network topology where the Life Cycle Assessment (LCA) pattern is applied (Figure 2).

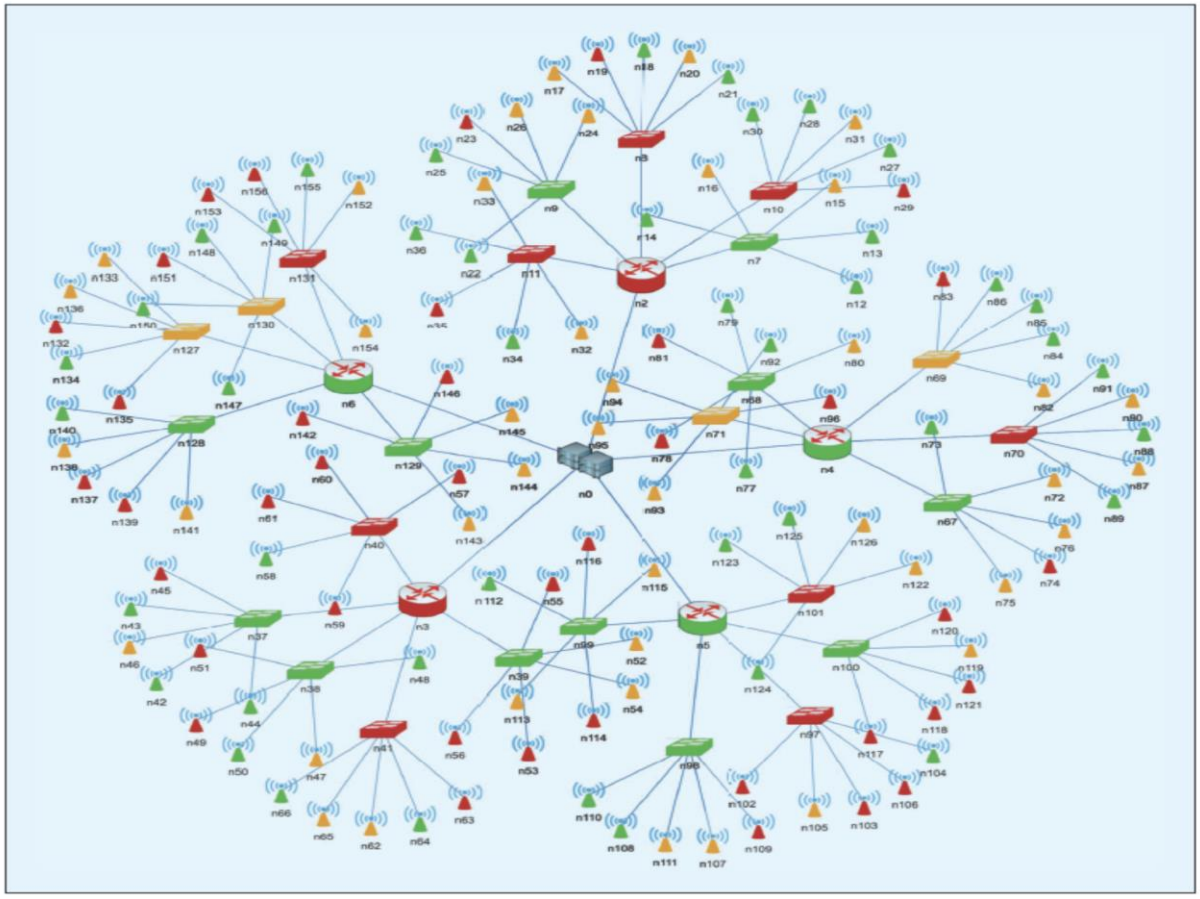

Figure 2. An example deployment of a tree-network topology enriched with LCA pattern information [8]

In this figure, the colour coding (i.e. green, orange, and red) is a visual representation of the condition of the components (i.e. good, requiring repair/refurbishment, or recycle), and its availability factor (i.e. operating or usable). Monitoring at this level would allow the application of visual analytics to identify clusters or constellations that are likely to fail. As such, from a CE perspective, monitoring would not only help achieve the highest level of utilisation of a resource but would also offer resilience by informing the business continuity aspects of the infrastructure by providing actionable intelligence concerning components that are likely to fail.

2.3 IDEAL-CITIES monitoring approach

Following the above positions, the overall requirements defined in upcoming deliverable D4.1 and the monitoring approach outlined in deliverable D3.2, we derive the monitoring assets (i.e. IoT and end user applications), the network components and the microservices

architecture. The latter will be a minikube[2] deployment and its monitoring will be delivered through Prometheus[3] and Grafana[4]. An example of this architecture is depicted in Figure 3.

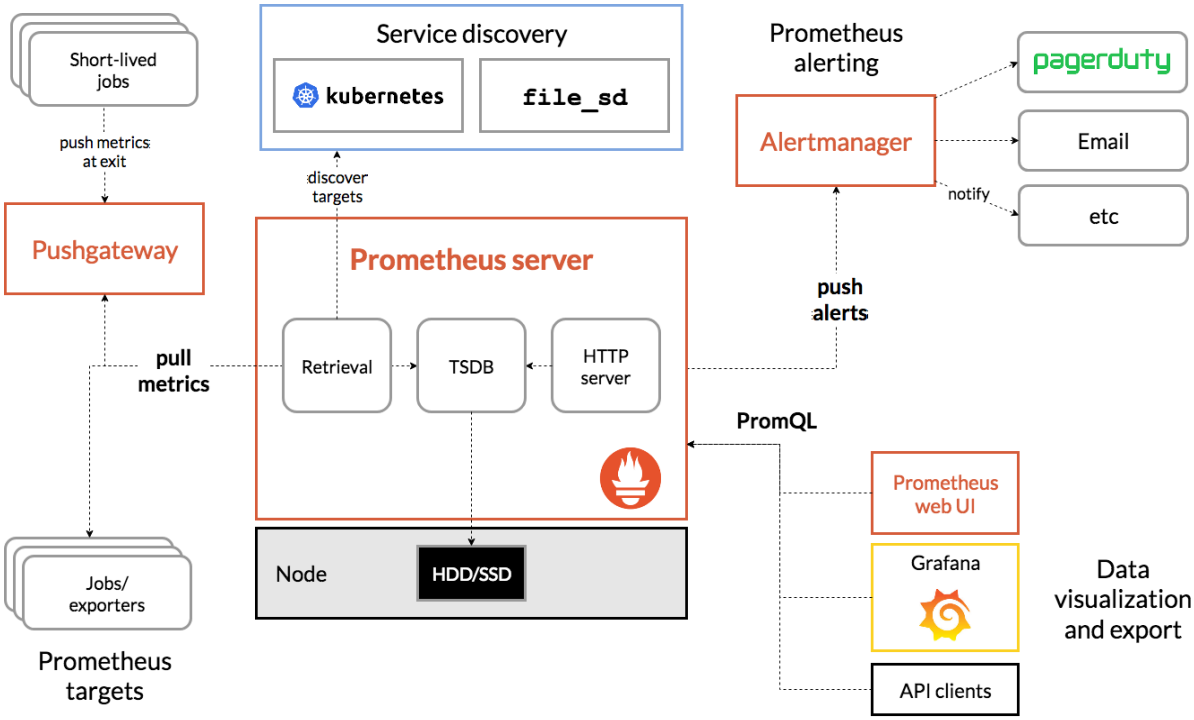

Figure 3. Example Prometheus monitoring architecture [9]

One of the main requirements Prometheus meets, is its capabilities of processing time series of data from different data points. Integration within Kubernetes[5] is seamless, as Prometheus uses a pull system by sending Hypertext Transfer Protocol (HTTP) requests to fetch the data.

Grafana is the visualisation software that normally accompanies Prometheus, similar to the popular software tuple Elastic[6], Logstash[7], Kibana[8]. In other words, Grafana uses Prometheus as a source for ingesting data.

[2] https://minikube.sigs.k8s.io/docs/

[3] https://prometheus.io/

[4] https://grafana.com/

[5] https://kubernetes.io/

[6] https://www.elastic.co/

[7] https://www.elastic.co/logstash/

[8] https://www.elastic.co/kibana/

Although Prometheus and Grafana will be configured to monitor the Kubernetes resources, from a security perspective the deployment will also implement in-depth defence practices, through the following actions:

- Configuration of Kubernetes auditing. The platform will include auditing records (since these exist in the latest versions of Kubernetes), to monitor events relevant to the Kubernetes ecosystem (e.g. creation of a deployment, namespace management, pod patching etc.). The audit records also contain complete metadata information on the events such as users involved, timestamp and namespace information. The audit policy is expressed as a set of rules in the audit policy yaml[9] file.
- Configuration of the audit backend, which is a component needed to enable auditing. As the proof of concept is one minikube deployment, the backend will be the log rather than the webhook backed.

2.3.1 Situational awareness: threat landscaping

Vulnerability monitoring on the platform components will be performed on a regular basis. External sources of vulnerabilities such as cvedetails.com[10] and the National Vulnerability Database[11] (NVD) will be periodically consulted to minimise the exposure window of the deployment. For instance, at the time of this writing, out of the 46 published vulnerabilities[10] for Kubernetes[12] , those that will be prioritised to ensure that the system is not exploitable are related to privilege escalation and information gain (Figure 4). This approach will be followed for all the software components through an established vulnerabilities management scheme.

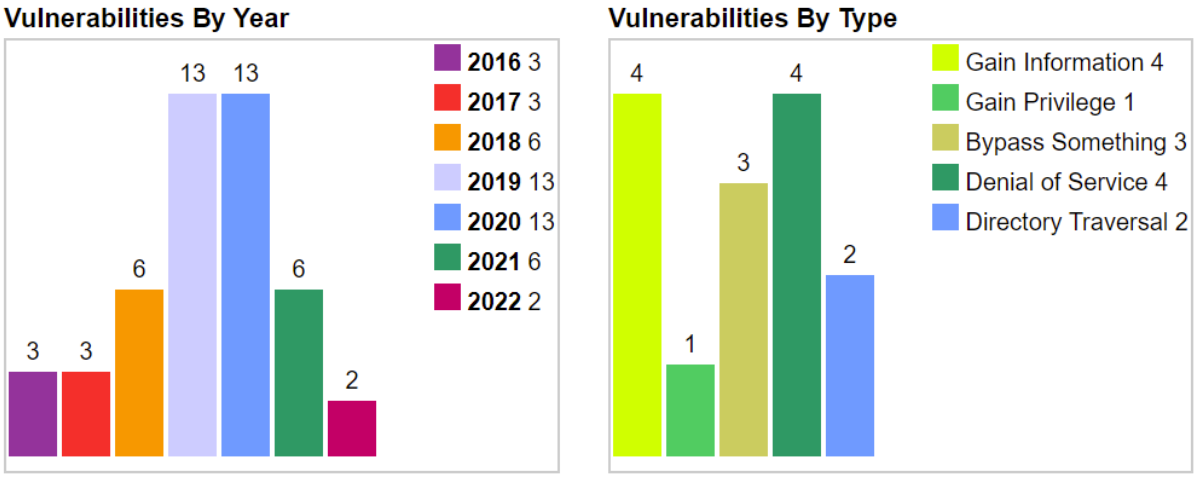

Figure 4. Summary of known published vulnerabilities affecting Kubernetes [10]

[9] https://yaml.org/

[10] https://www.cvedetails.com/

[11] https://nvd.nist.gov/

3 Adaptive Networks

3.1 Introduction

Today's networks have become large and complex and consist of a wide variety of vendors. Meanwhile, there are cases in which the location of the involved components spans to large geographic areas. Some of the types of these components are routers, switches, wireless access points, servers, workstations, printers, Private Branch Exchanges (PBX's) and Internet of Things (IoT) devices. To have a smooth operation of the network, it is vital to monitor most (the most critical), or even all of the involved components. Any network down-time, even a minute one, may compromise the ability to provide essential services.

A network failure may occur due to different reasons such as a hardware failure, an end-user error or even a deliberate malicious action. Network monitoring includes various techniques which are deployed by design to maintain the availability, reliability and security of the network. Network administrators, may better understand network behaviour when they are provided with relevant information, thus enabling them to react faster in troubleshooting and mitigating network anomalies. With such information at hand, network administrator teams will be able to adapt network components to better respond to the challenges that may arise. This can be achieved either by manual intervention, or automatically, based on predefined protocols or rules. Additionally, administrators are expected to be proactive, and therefore traffic monitoring, performance indicators and overall functionality of the entire network, is necessary.

The following sections will provide the requirements specifically for the IDEAL-CITIES platform, a state of the art of available monitoring tools and the holistic design of an adaptive network infrastructure.

3.2 State of the art and description of available monitoring tools

3.2.1 Firewall Monitoring

A firewall is a network security system that monitors and controls the incoming and outgoing network traffic based on predetermined security rules. A firewall typically establishes a barrier between a trusted internal network and untrusted external network, such as the Internet. Firewalls are often categorized as either network firewalls or host-based firewalls. Network firewalls filter traffic between two or more networks and run on network hardware. Host-based firewalls run on host computers and control network traffic in and out of those machines.

3.2.2 Deep Packet Inspection

In Deep Packet Inspection (DPI) packet payloads are matched against a set of predefined patterns. DPI imposes a significant performance overhead, but nevertheless, in one form or another, is part of many network (hardware or software) appliances and middle boxes. As Bremler-Barr et al. [11] have demonstrated, extracting the DPI functionality and providing it as a common service function to various applications (combining and matching DPI patterns from different sources) can result in significant performance gains. A DPI implementation, nDPI [12] can be employed to implement the DPI function, monitor incoming traffic, and

assign it to the (sub-)set of security service functions intended for the corresponding traffic type.

3.2.3 Honeypots

Network-based honeypots have been widely used to detect attacks and malware. A honeypot is a decoy deployment that can fool attackers into thinking they are hitting a real network whereas at the same time it is used to collect information about the attacker and attack method. A HoneyNet[12] is a set of functions, emulating a production network deployment, able to attract and detect attacks, acting as a decoy or dummy target.

3.2.4 Intrusion Detection / Prevention

Intrusion Detection System(s) (IDS) and Intrusion Prevention System(s) (IPS) are services able to monitor traffic or system activities for suspicious activities or attack violations. Detection is based on predefined rules and patterns or heuristic detection algorithms. Newer implementations may also utilize Machine Learning (ML) functionality as well as Artificial Intelligence (AI) techniques for detecting new uncategorized threats. In addition to detection IPS' may also prevent (block) detected attacks by utilizing Firewall-like functionality.

3.2.5 Network Virtualization

The traditional approach in networking, as previously mentioned, has relied on vendor specific special-purpose network nodes, where hardware and software are tightly coupled. The configuration of these proprietary nodes is rather costly and leads to a rather rigid network. This traditional approach is no longer adequate, and several reasons justify this trend.

Firstly, the number of devices requiring network connectivity and the data rate has increased dramatically, mostly due to a huge number of IoT devices and mobile terminals. Secondly, the appearance of IoT demand services with dynamic and heterogeneous Quality of Service (QoS) requirements. In this scenario, the traditional networking approach, based on vendor specific special-purpose nodes, leads to dramatic increases in Capital Expenditure (CAPEX) and Operational Expenditure (OPEX) [13]. These issues are circumvented thanks to a new networking approach based on Network Function Virtualization (NFV).

The aim of NFV is to provide a network that is dynamic, flexible, scalable, programmable and easy to reconfigure. This paves the way to support a huge number of IoT devices and novel IoT services with heterogeneous QoS requirements by allocating the necessary resources. All these features are desirable in the IDEAL-CITIES project, where a massive amount of IoT devices must be connected with the IoT gateways and the backend cloud with different QoS constraints (e.g. latency, reliability, security and privacy). These QoS measures must be guaranteed despite the impairments posed by the network (i.e. data flow paths that guarantee the required QoS must be established dynamically). Moreover, due to latency, computational and communication constraints the IoT data analytics is carried out either at the IoT Gateway or at the backend cloud. Thereby, the network must be flexible and programmable so as to setup and to release the computational and communication resources

[12] https://www.sciencedirect.com/topics/computer-science/honeynets

to convey the information to the appropriate computing resources, and to allow a computation that guarantees the required QoS. For all these reasons, it is mandatory to have a global view of the network state and a global control of the network resources. In NFV this is accomplished thanks to a centralized orchestration in the so-called NFV Management and Orchestration (NFV MANO).

To this end, NFV relies on the following pillars:

- Consideration of general-purpose hardware nodes, rather than vendor-specific special-purpose ones, in specific parts of the network.
- The computational, storage and communication resources of the network nodes are virtualized and exposed as a Network Function Virtualization Infrastructure (NFVI).
- Network services are envisaged as a chain of Virtual Network Functions (VNFs) deployed on top of the NFVI by dynamically allocating the required resources demanded by the service so as to guarantee the specified QoS. The coordination and control of the VNF, as well as the NFVI to deploy them, require a specific functional block, the so-called NFV MANO.

In Figure 5 the functional blocks of an NFV platform are displayed. This architecture is compliant with the one proposed by ETSI [14].

NFV offers flexible, programmable, dynamic, scalable and easy ways to reconfigure network. To this end, NFV follows the following approach.

General purpose hardware devices are considered in different parts of the network. It is assumed that these machines allow the virtualization of their resources in terms of Virtual Machines (VM) or containers yielding a pool of virtual computing, storage and communication resources available to deploy the network services.

The virtualization of the hardware resources is managed by a so-called virtualization layer. As it can be seen in Figure 5, the set of physical hardware resources, the virtualization layer and the virtualized computing storage and networking resources is the so-called NFVI. Thus, NFVI contains all the resources available in the network. NFVI paves the way to obtain a flexible, programmable, dynamic and scalable network, as the virtual network resources exposed to the network services can be dynamically assigned or released in different parts of the infrastructure to meet the required QoS requirements.

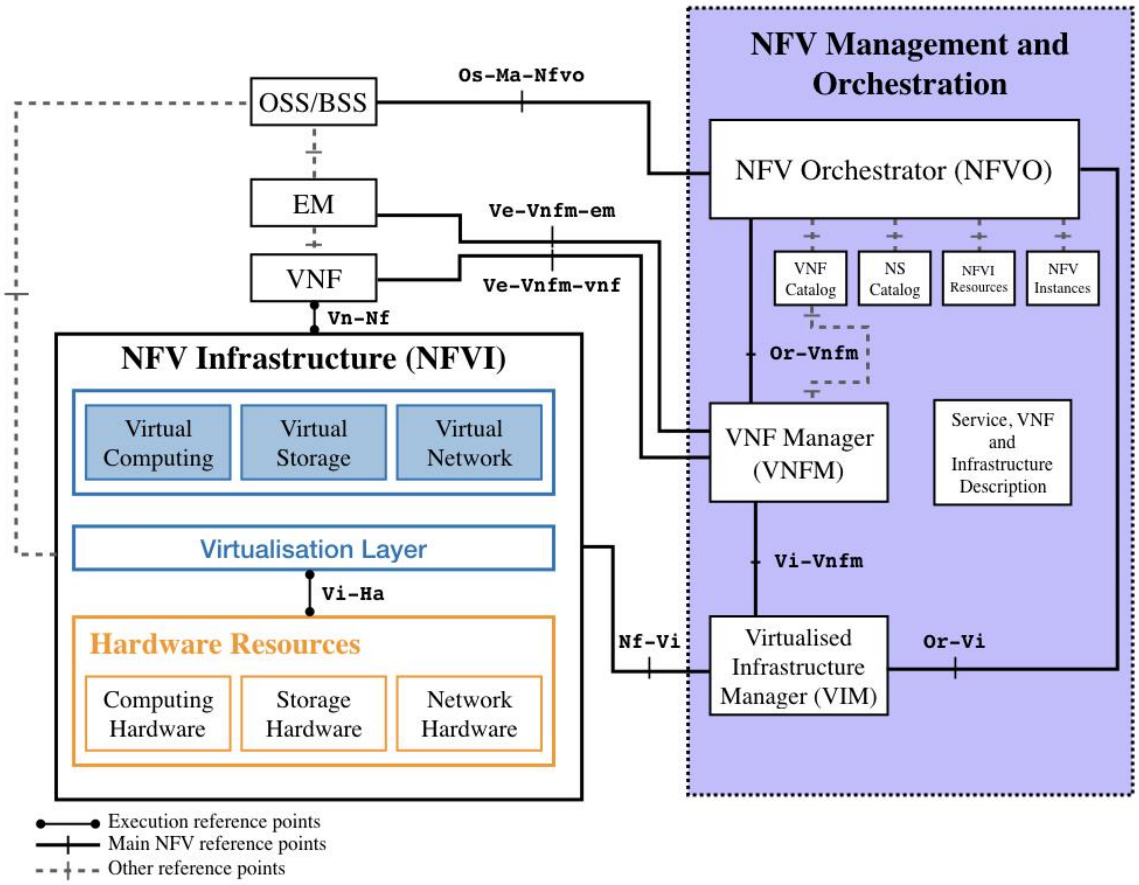

Figure 5 Functional blocks of an NFV platform

With NFV, the network services are implemented as a chain of functional blocks, which are called VNF. A VNF is a virtualization of a network function in a legacy non-virtualized network, referred to as Physical Network Functions (PNF). Moreover, the chain of VNFs that implements a network service is called a Service Function Chain (SFC). As it is displayed in Figure 5, each VNF is deployed on top of the NFVI. Please note that the Element Management (EM) depicted Figure 5 is responsible for VNF management.

Furthermore, virtual computing, storage and network resources are assigned to run the VNFs. This allocation of virtual resources is exemplified in Figure 6. This figure highlights the flexibility, scalability and support for heterogeneous QoS requirements that is provided by an NFV, as virtual resources can be assigned or released easily according to the QoS required by the VNFs and the SFC.

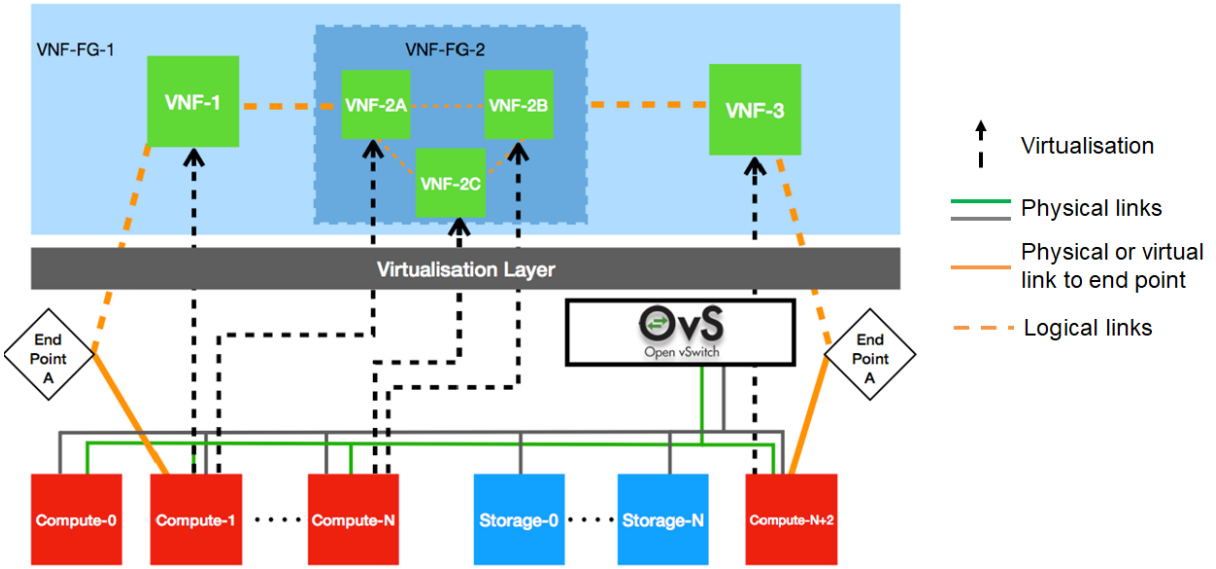

Figure 6 Example of virtual resources allocation to a chain of VNFs

At this point, we have seen how network services are implemented in NFV, in terms of VNFs, or more in general, in SFC. Additionally, it is observed that each VNF requires a set of virtual resources from the NFVI. Obviously, all these blocks, i.e. the network services and the network resources, require a global management. In an NFV architecture, the responsible for this management is the NFV MANO.

The NFV MANO is composed of three main blocks:

- The NFV Orchestrator (NFVO).
- The VNF manager.
- The Virtualized Infrastructure Manager (VIM).

In fact, these NFV MANO blocks are organized in a hierarchical manner in terms of management responsibility. That is, the orchestrator is the responsible of managing the overall NFV. Thus, it manages the communication with the network service providers by exposing proper northbound interfaces. For instance, the Open Source MANO (OSM), which is an ETSI-compliant open-source implementation of the NFVO and VNF manager blocks, provides open standard-based APIs such as NETCONF[13] and REST[14]. Through these interfaces the network service providers can specify the features of their services. Namely, in OSM they use the so-called Network Service Descriptors (NSD) [15]. These NSDs in turn refer to a set of VNF descriptors (VNFD), which characterize the VNFs that the network service requires. The VNFs are connected through virtual links that are defined properly through Virtual Link descriptors (VLD). Moreover, the VNF-FG descriptors (VNFFGD) determine the traffic flows between the VNFs in the service chain associated to the network service. Thereby, the NFVO

[13] https://datatracker.ietf.org/doc/html/rfc6241

[14] https://restfulapi.net/

is the responsible of the network service management lifecycle, which implies the next responsibilities:

- Manage the global computing, storage and communication virtual resources.
- Authorize NFVI resources requests.
- Policy management related to scalability, to reliability, to high availability related to network services instances.
- Manage the catalogue of network services templates.
- Onboarding of new network services and VNFs packages.

As is shown in Figure 5, the NFVO has southbound interfaces (specified via reference points [16]) with the VNF manager and the VIM. Thereby, for a given network service request, the NFVO delegates the management of the VNFs and virtual resources involved in the network service, to the VNF manager and to the VIM, respectively. Moreover, the VIM and the VNF manager use these interfaces in a northbound direction e.g. to send state information on their management and the state of the configurations requested by the NFVO.

Another important block of the NFV MANO is the VNF manager. It is responsible for the VNF lifecycle management. This includes the following responsibilities:

- VNF instantiation or start, given its associated VNF descriptor.
- VNF monitoring by collecting parameters that determine the VNF health, e.g. CPU load or memory usage.
- VNF scaling. Key Performance Indicators (KPI) of the VNF are monitored and if they are above a given threshold a scaling process is started, which implies the creation of new VM to deploy the VNF.
- VNF termination.

The third sub-block that builds an NFV MANO is the VIM, which is the responsible for managing the overall NFVI. Thereby, upon request of the VNF manager and the NFVO, the VIM must assign the necessary virtual compute, storage and network resources to run properly the VNFs that form an SFC. The VIM has also the next roles:

- It manages the inventory of the computing, storage and network resources related to the NFVI.
- The VIM manages the NFVI resources allocation. Thus, it facilitates the assignment, increase or release of resources to VMs to run or to terminate a given VNF.
- It provides logs related to performance issues that arise in the NFVI.
- It has information on the infrastructure faults.
- It collects information to monitor the state of the NFVI and to allow the optimization of its resources.

3.2.6 Software Defined Networking

Software Defined Networking (SDN) aims to improve network performance and monitoring by introducing dynamic, programmatically network configuration. In an SDN framework the SDN Controller (SC) employs a set of Southbound Interfaces (SBI) for configuring network

devices (such as NETCONF or OF-CONFIG)[15], and to control such devices' forwarding table (using protocols like OpenFlow[16]). As network devices boot-up, they reach for the controller in order to register the devices for configuration and forwarding table modifications. After this process is finished, the SC establishes a complete, centralized view/control of the network, whose topology may be visualized employing the SC's Northbound Interfaces (NBI) (usually RESTful APIs).

Physical Network Functions (PNFs), such as the IIoT Gateway, as well as the switches and routers at the Field and Network layers, are equipped with compute and storage resources, which, in combination with the Backend-Cloud's resources, are managed and exposed by the Virtual Infrastructure Manager (VIM). These devices and their resources may be used for the spin-off of VNF (like virtual switches, routers, firewalls, load balancers, processing or storage endpoints), and are (virtually) connected together in the form of VNF Forwarding Graphs (VNF-FG) to offer network services. Such connections are achieved by virtual network overlays, or Virtual Tenant Networks (VTN), built by the SC on top of the underlying physical network infrastructure. In order to provide a network service, the VIM and the SC share a common VNF-FG. As the VIM spins-off VNFs (e.g. virtual switches) and specifies how these should be connected together, the SC modifies the forwarding table of the appropriate VNFs in order to ensure secure and reliable communication of the whole VNF-FG.

There are two main ways in which an SC manages the underlying fabric namely, directly or via network overlays [17]. The former consists of the SC directly communicating with SDN-enabled switches via SBI such as OpenFlow [18], NETCONF [19], OVSDB[17], OpFlex[18], and others. These SBIs allow the controller privileged access to the devices' forwarding tables, as well as other device configurations per se. This method is the one used by the most popular off-the-shelf controllers such as OpenDaylight (ODL)[19]. The latter method uses encapsulation protocols (e.g. VXLAN[20], NVGRE[21], IPSEC[22]) on top of conventional networks[23] to build the desired network topologies. Even though it is debatable whether overlays are SDN or not, they are indeed software defined[24].

[15] https://opennetworking.org/wp-content/uploads/2013/02/of-config-1-1-1.pdf

[16] https://en.wikipedia.org/wiki/OpenFlow

[17] https://docs.openvswitch.org/en/latest/ref/ovsdb.7/

[18] https://www.sdxcentral.com/networking/sdn/definitions/cisco-opflex/

[19] https://www.opendaylight.org/

[20] https://en.wikipedia.org/wiki/Virtual_Extensible_LAN

[21] https://en.wikipedia.org/wiki/Network_Virtualization_using_Generic_Routing_Encapsulation

[22] https://en.wikipedia.org/wiki/IPsec

[23] By conventional it is meant that there exists L2 or L3 connectivity among components of the network.

[24] In fact, Virtualized Infrastructure Managers (VIM, e.g.: OpenStack) employ an SDN Controller and overlays to provide private tenant networks.

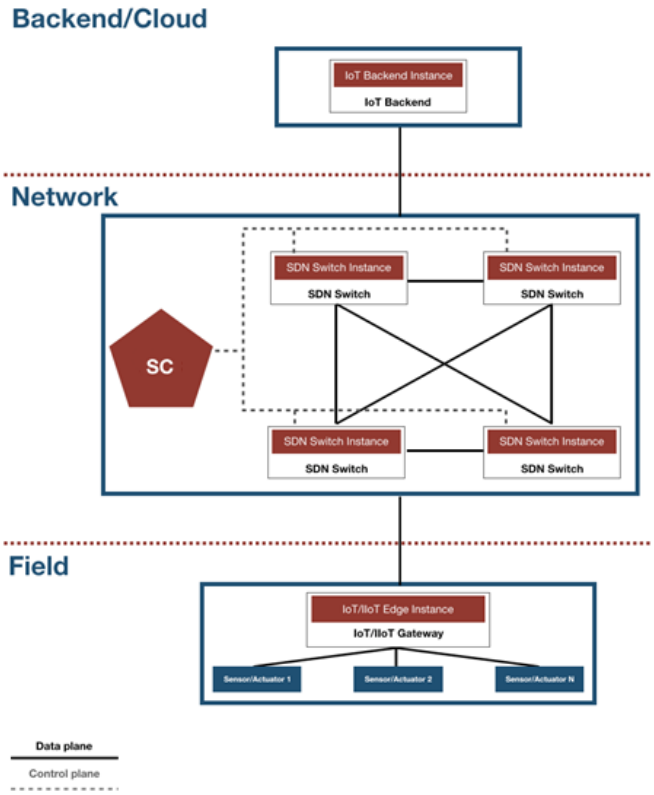

Figure 7: Example of SDN Architecture Deployment View

Regardless of the SDN Controller vendor, there are core features that must be supported by an industrial variant of an SDN Controller. A non-exhaustive description of these is provided below and [17]:

- Data plane programmability: change the way flows are forwarded, apply filters, or dynamically changing packet headers. Via NBI control information concentrated at the SDN Controller could be accessed/changed by SDN Applications. Such applications may then use such information to apply templates that effectively change the network configuration (e.g. satisfy bandwidth constraints, QoS enforcement, forward through least expensive paths etc.)
- Southbound protocol support: the most widely used southbound protocol is OpenFlow. An SDN Controller should be able to interact with OpenFlow agents (or other southbound protocol) in SDN-enabled forwarding devices in order to control the different actions to be performed while forwarding.
    - As mentioned previously, OpenFlow is not the only protocol supported by ODL as SBI. NETCONF, OVSDB and SNMP[25] are some of the other alternatives [17][20].

[25] https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol

- External API support: ensures that it could be used within various cloud orchestration environments, e.g. OpenStack. Through well specified APIs, network policies exchange is realized, allowing the control of the networking resources of a virtual infrastructure.
- Centralization: allowing administrators a complete view of the network. It should also support discovery protocols so new network devices are registered and bootstrapped if necessary.
- Performance: as network devices rely on the SDN Controller for handling incoming flows not contained in the current flow table, controllers should ensure such requests are handled as fast as possible, otherwise the SDN Controller is prone to become a network bottleneck.
- High Availability and Scalability: the ability to work as a cluster allows an SDN Controller to expand its performance and availability by adding more controller nodes and load balancers.
- Security: as the functioning of the network depends on the SDN Controller, it should be capable of authenticating/authorizing members of the network while performing intrusion detection and prevention, e.g.: secure southbound channels, encryption.

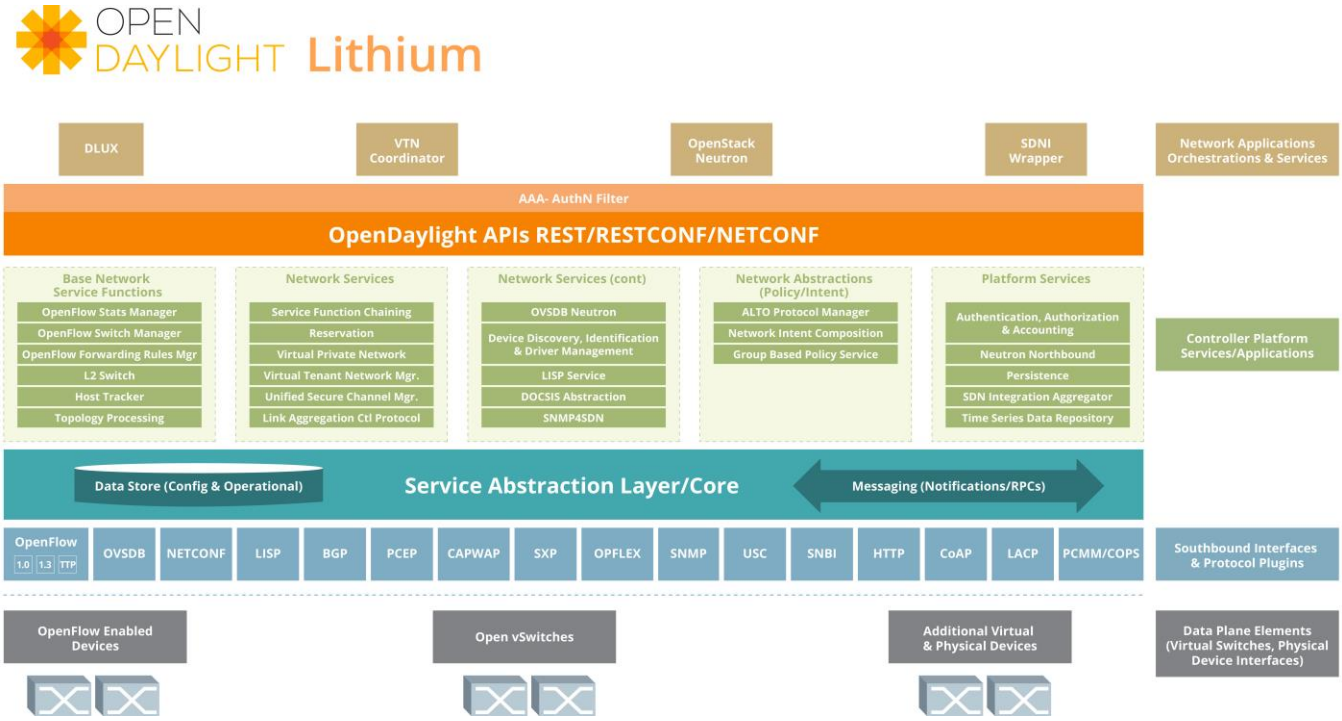

Figure 8: OpenDaylight SDN Controller Platform[20]

Modern SDN Controllers, such as OpenDaylight (ODL), are built following a modular architecture (see Figure 8). Each independent module could leverage services exposed by other modules in order to provide composite functionality. Focusing on the configuration of the data plane, the following details the most relevant southbound protocol modules implemented by ODL (or Southbound Interfaces and Protocol Plugins, as referred to at the bottom of Figure 8):

- NETCONF: ODL supports communication with NETCONF servers at SDN-enabled forwarding devices. Additionally, this module may work as a NETCONF server itself in order to expose Control plane information to external entities, e.g.: SDN applications.
- OpenFlow: Enables the SDN Controller to configure the flow table of OpenFlow enabled SDN devices remotely. OpenFlow Library allows the SDN Controller to listen for OpenFlow messages (spawns a daemon), as well of supporting multiple OpenFlow versions.
- OVSDB: It is used for managing (view, create, modify, and delete functions) OVS-enabled switches (physical or virtual). OVS[19] is an open-source software that implements virtual switching that is interoperable with almost all popular hypervisors. OVS uses OpenFlow to perform message forwarding in the control plane for both virtual and physical ports. OpenDaylight's OVSDB southbound plugin manages OVS devices supporting OVSDB schema and the OVSDB management protocol,
- SNMP4SDN**:** This module allows ODL to interact with SNMP-enabled switches. It uses SNMP methods for writing forwarding as well as configuration information into the devices.

3.2.7 Automated Traffic Recovery

Redundancy is always a desirable feature in a network design so that when a link failure occurs it will be possible to have automated traffic recovery. The obvious solution to achieve redundancy, is to add more than one links between network devices. This approach does not come without issues because it is vulnerable to broadcast storms. When a broadcast storm occurs eventually the links become saturated, and the resources of the switches are consumed to the point that there are no more resources available. A well-known solution to this, is the adoption of the Spanning Tree Protocol (STP) which allows the existence of redundant links while protecting the infrastructure from unwanted loops that result in a total network disruption. At the same time when the primary link fails, the Spanning Tree Protocol allows the automated traffic recovery by enabling the secondary link.

The switches that run STP are aware of the existence of each other by exchanging bridge protocol data units. The result of the STP is a loop-free topology by blocking traffic on the ports that have redundant links. STP has various implementations such as Multiple Spanning Tree Protocol (MST), that can be applied to network topologies depending on their size. For example, in a simple topology where no VLANs exist the standard implementation of STP (IEEE 802.1D) is enough. On the other hand, when the topology is larger along with multiple VLANs then the MST (IEEE 802.1s) is more appropriate.

On the left side of Figure 9 the STP has blocked the port of Switch B that connects with Switch C therefore the traffic originating from client destined to server, travels from Switch C directly to Switch A. In the event of a link failure that connect switches A and C, the STP will unblock automatically the port of Switch B and the traffic will travel from Switch C to Switch B and finally to Switch A.

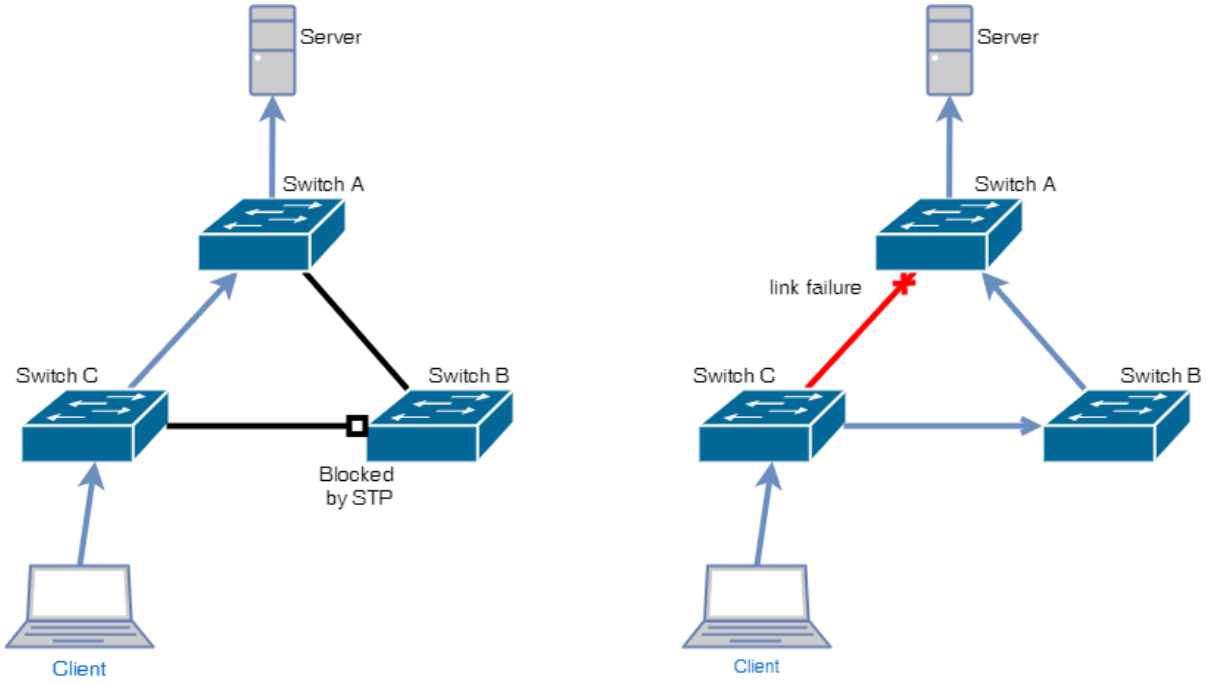

Figure 9 Automated Traffic Recovery by STP

3.3 Design of an Adaptive Network Infrastructure

In order to have the ability to adapt the network infrastructure, an appropriate network design is needed. A hierarchical approach can enable targeted adaptation on the network while avoiding the disruption of communication on the whole network. In the hierarchical design model, the network is divided into modular layers. This breakup of the network allows us to implement specific functions to each layer. Additionally, this approach provides easier scalability while guarantying a consistency in the deployment.

Having a hierarchical design holds a clear advantage over a flat and fully meshed network. Network changes in a meshed network tend to affect larger portions of the network whereas in the hierarchical design they are constrained to a subset of the network resulting in better fault containment management and resiliency. The network devices can be added or removed with minimal impact to the rest of the network thus isolating potential problems.

In a hierarchical design (Figure 10) there are three layers: access, distribution and core.

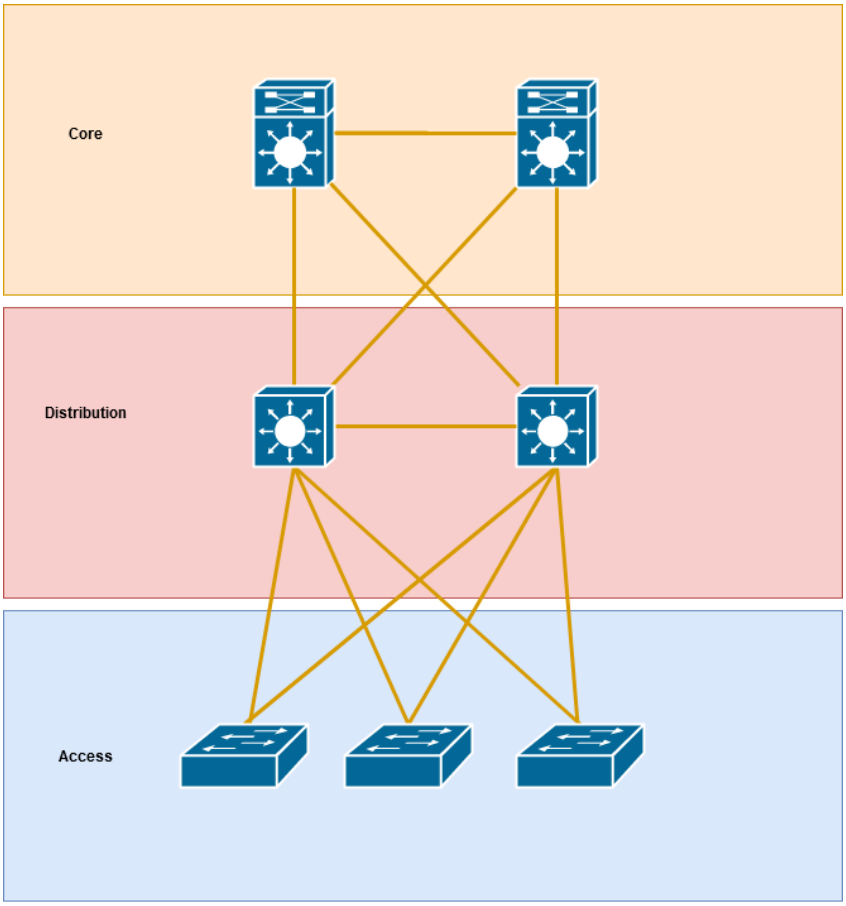

Figure 10 Hierarchical architecture

The access layer is where end-user devices connect to the network with high bandwidth connectivity. Even though the end-user devices do not use the full capacity of bandwidth all the time, having high bandwidth availability improves the quality of experience. Most common types of devices at that layer are personal computers, printers wireless access points etc. For increased performance, management and security the different devices can also be separated into different logical networks such as Virtual LANs (VLANs). Due to the fact that the access layer switches are not directly connected to each other, the communication of the end devices is achieved through the distribution layer. Protection from malicious attacks is achieved at this layer by preventing the end-devices to access services that are not authorized.

The distribution layer provides an aggregation point for the switches located at the access layer in a building or campus. This layer separates the Open Systems Interconnection (OSI) layer 2 domain of the access layer and the OSI layer 3 domain of the core layer. This separation facilitates the limitation of the layer 2 faults on the layer 3 side. Furthermore, at the distribution layer occurs the summarization of IP routing information before it enters the core layer. This summarization reduces the overhead and provides faster recovery from failures. The switches at this layer come always in pairs for redundancy and are interconnected to each other. Depending on the size of the environment these switches may be located in different buildings.

The core layer constitutes the backbone of the network. It is the optimal point for aggregating different networks and providing features such as high availability and fast network

convergence. Additionally, the core layer optimizes the network design especially in cases where the distribution layer is not contained into a single location by reducing the complexity. For instance, if we had N distributions points, we would require N x (N - 1)/2 links, but with the implementation of a core layer we can reduce these link to just N as shown in Figure 11. Finally, the core layer allows the communication between different network blocks such as the private cloud, the public cloud the WAN etc.

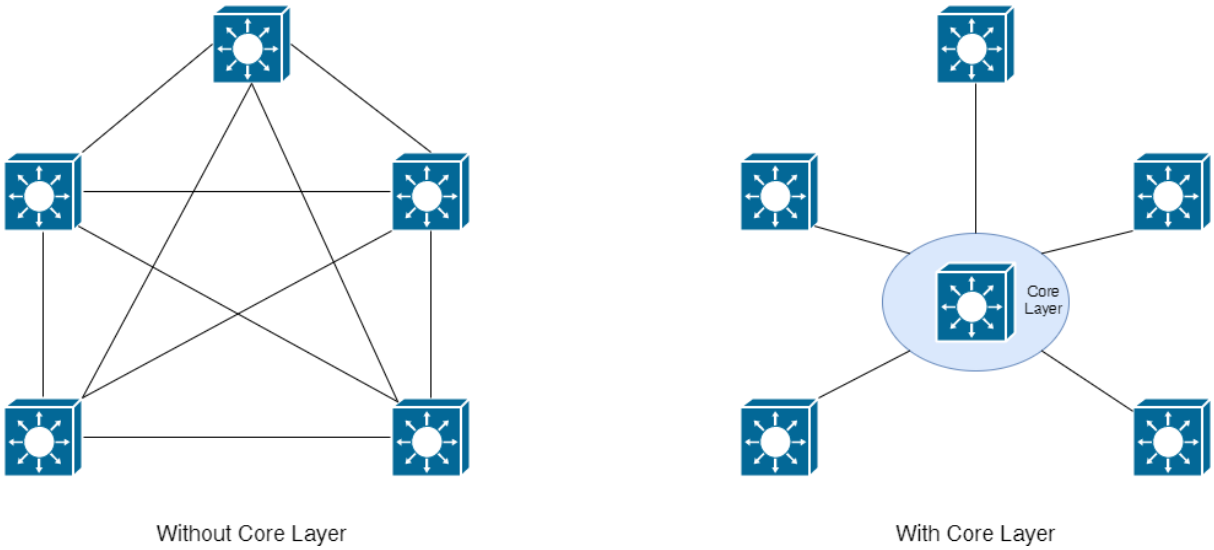

Figure 11 Links in network design with or without Core Layer

3.4 Requirements for IDEAL-CITIES platform

3.4.1 IDEAL-CITIES implementation requirements

The challenge faced for the IDEAL-CITIES platform lies with the fact that the actual environment cannot be realized at its whole in advance. Therefore, it is crucial to have the flexibility for easy scalability when the circumstances require it. In the production (final) version of the IDEAL-CITIES platform and based on the fact that the Hierarchical architecture that already presented is followed, this scalability is feasible because it is inherited by design.

3.4.2 Requirements for the demo

In the demo scenario which will be conducted within the scope of the project, the SDN and NFV solutions will not be utilized (implemented), due to the small scale of the experiments. In the proposed scenario the use of the above technologies will be simulated in real-life situations. Consequently, the consortium has decided to use open global network for communication between the users and the platform's infrastructure.

4 Adaptive Infrastructure

4.1 Introduction

As mention in the beginning of this document, infrastructure is a relative term meaning "the structure beneath a structure" and can be shared among many individuals simultaneously. In the physical world, the infrastructure is referred to public systems like electricity grid, gas, water, sewage systems etc.

Each layer of infrastructure has certain characteristics, including:

- Shared by a larger audience than the structures it supports
- More static and permanent than the structures it supports
- Considered a service, including the people and processes involved in support, rather than just a physical structure or device
- Often physically connected to the structure it supports
- Distinct from the structures it supports in terms of its lifecycle (plan, build, run, change, exit)
- Distinct from the structures it supports in terms of its ownership and the people who execute the lifecycle

In the following sections, IT infrastructure will be defined as physical resources and middleware software that facilitate the deployment and execution of applications. The parameters that will describe this infrastructure will be CPU power, amount of RAM, disk space, and connection bandwidth available to potential clients.

4.2 Requirements for IDEAL-CITIES platform

The IDEAL-CITIES platform aims to provide the tools for better utilizing a city's IoT and communication assets. It provides the functionalities that will increase the citizens' safety and involvement in all aspects of public life. The challenge for a cloud solution like the IDEAL-CITIES platform is the scalability of the number of connected devices, logged users, and exchanged communication (i.e. messages). Due to the unpredictable nature of the workload of the platform, we have decided to use the adaptive infrastructure that can be scaled dynamically to accommodate any increasing demands in platform usability and capacity.

In the days of the large market of cloud providers, access to computing resources is much easier, cheaper, and convenient than ever before. The new challenge is to optimize the utilization of resources which lowers the cost of running the application while the user experience remains at a satisfactory level. New challenges have been raised, such as how to parameterize the user experience, how to monitor it and how to automatize the scaling of the application. As the monitoring aspect has already been described in the second section of this document, in the next sub-sections we will discuss the parametrization and scaling aspects.

4.2.1 Parametrization of user experience

We will consider the user experience in the technical aspect of the cloud-deployed application. The involvement of the user will not be taken into consideration as this aspect is part of the application itself. With the above assumptions, the element of user experience that dependents on the infrastructure is the response time for HTTP requests.

As the IDEAL-CITIES platform implements the microservices architecture the request can be processed by many components, so the parametrization of time processing of the request is dependent on many components.

4.2.1.1 Categorization of the HTTP request

Due to the microservices nature of the IDEAL-CITIES platform, the response time for the HTTP request can be divided into a tier group whose importance can be specified on the role within the application. Each tier of the request will have its maximum response time which cannot be overridden.

4.2.2 Measurement and scaling

For HTTP response time, we shall consider the orchestrators' inbuild mechanisms. With those we can measure the total time for the request independently of how many internal and external calls it can achieve. In a microservices architecture, the crucial part is the identification of the component, which is also the bottleneck of the workflow. For this identification, the human factor (i.e. administrators) cannot provide the capacity as there may not be one-to-one coloration between the endpoint and set of components.

Many endpoints can utilize the same components, and even one component can use different components depending on the payload of the request. A resource usage measurement has to be taken into consideration. Live monitoring of the physical resources provides the capability of observing the utilization of each component as well as the use of its assigned resources.

A potential use of the collected utilization information is setting the thresholds for adding an extra replica (clone) of a component or destring it. The levels at which the action should be taken depends on the current load of the application and the importance of the component.

Monitoring data can also be used in the prediction model of the components replica management system. Prediction mechanisms can use historical data, analyze it, find the patterns of the application load, and preemptively add new replicas of components before they reach a static threshold.

4.3 Adaptive infrastructure orchestrator for deployment application

As the IDEAL-CITIES platform is aimed to be deployed at many local clusters, we will not describe the tools for the hardware administration of the cloud provider server, but rather on a higher level. In the next sub-section, we describe the available tools for the orchestration of the cloud resources.

4.3.1 Docker the backbone of the orchestration

The vast majority of orchestration tools is based on the containerization of software. A container is a standard unit of software that packages a piece of code and all its dependencies, so that the application runs quickly and reliably, consistently from one computing environment to another. The most popular environment for containerization is Docker[26]. It is an open-source initiative that provides the engine to run pre-packaged application images. It

[26] https://www.docker.com/

offers a centralized repository for the storing of images and maintains a large development support community. Due to its popularity, the following tools, as well as many others, utilize Docker's approach to application deployment.

4.3.2 Review of infrastructure orchestration tools

Next, we present some selected orchestration tools for deploying a microservices platform in the cloud.

4.3.2.1 IronWorker

IronWorker [21]: is a solution for handling container-based workloads. It is an enterprise-grade background job processing system based on Docker. It provides application developers with the ability to asynchronously process workloads without having to worry about scaling the infrastructure. It runs on public clouds, private clouds, and on-premises and since the code is executed in Docker containers it's tasks can be written in any language.

4.3.2.2 AWS Fargate

AWS Fargate [22]: is a compute engine that can be used with Amazon Elastic Container Service[27] (ECS) to run containers without having to manage servers or clusters of Amazon EC2[28] instances. Faregate provides many pre-configured settings like scaling thresholds, networking logging, resource limits etc. It forwards the deployment process and allows developers to focus on the developing application and not the administrating aspect.

4.3.2.3 Rancher

Rancher [23]: is an open-source platform that implements a purpose-built infrastructure for running containers in production. Rancher is based on Kubernetes [25] and it runs on raw computing resources from any public or private cloud. The key product features of Rancher include cross-host networking, container load balancing, persistent storage services, multi-tenancy & user management, resource management and multi-orchestration engines. It combines many other open-source tools developed around Kubernetes and makes it easier to use and manage as an extra combined layer.

4.3.2.4 Kontena Classic

Kontena Classic[29]: is a developer-friendly, open-source platform for orchestrating applications that are run on Docker containers. It simplifies deploying and running containerized applications on any infrastructure. By leveraging technologies such as Docker, Container Linux[30], and Weave Net[31], it provides a complete self-contained solution for organizations of any size.

---

[27] https://aws.amazon.com/ecs/

[28] https://aws.amazon.com/ec2/

[29] https://github.com/kontena/kontena

[30] https://en.wikipedia.org/wiki/Container_Linux

[31] https://www.weave.works/docs/net/latest/overview/

4.3.2.5 Nomad

Nomad [24]: is a flexible workload orchestrator that enables an organization to easily deploy and manage any containerized or legacy applications, by using a single unified workflow. It enables developers to use declarative infrastructure-as-code for deploying applications and can run a diverse workload of Docker, non-containerized, microservice and batch applications. It is configured and managed by jobs (i.e. declarative specification tasks to be executed by Nomad).

4.3.2.6 Kubernetes

Kubernetes [25]: also known as K8s, is a world-leading open-source system for automating deployment, scaling, and management of containerized applications. It uses the Docker container for deploying the application, with core features like automated rollouts and rollbacks, storage orchestration, horizontal scaling self-healing IPv4/IPv6 dual-stack, and many more. It is armed with many other open-source projects that provide monitoring, graphical interfaces, preconfigure sets of applications etc. It becomes the root for many other orchestration projects that are built upon it.

4.3.2.7 Summary

All the above-mentioned tools are able to work with Docker images. Due to this fact, we recommend the “dockerization” of all components that are designed to be deployed in the cloud.

4.4 Design of Adaptive Infrastructure in the IDEAL-CITIES platform

The IDEAL-CITIES platform, which by definition will facilitate the integration of modern web, mobile, and IoT technologies in living cities environments, is oriented so and can benefit by taking advantage of generic methods of deployment.

For that reason, we have picked Kubernetes as the orchestration tool of choice. Kubernetes is the most popular among the orchestration projects, as it can be run on bare metal, private clouds, and also on most of the public clouds.

The Technology Readiness Level (TRL) 4/5 planned for the IDEAL-CITIES’ platform at the end of the project, empowers us to use an open-source highly configurable orchestrator solution rather than a commercial ready-to-use application deployment.

Moreover, using Kubernetes will allow us to demonstrate the scalability aspect of the IDEAL-CITIES platform, while at the same time guarantee consistent deployment on any cloud infrastructure that supports Docker and Kubernetes.

5 Use case specific requirements

The proposed demo scenario in the IDEAL-CITIES project will be conducted on a small scale. Consequently, the minikube, as described earlier in this document, will be used in the selected scenario. The final solution will enable the possibility of implementation on a low resource infrastructure with easy transformation to a full-scale infrastructure cloud or local.

Our proposed solution of architecture, provides flexibility in terms of the available infrastructure and network ecosystem.

6 Summary

In this document we presented the IDEAL-CITIES platform approach, in regards to scaling and monitoring and we performed a presentation as well as a comparison of the available technologies, their capabilities and advantages.

The main consideration for the consortium at this stage of the project, was to choose the best suited set of tools and deployment applications that would perform the required and desired actions, within a certain timeframe, in a preferably automated deployment manner, while also providing the necessary monitoring capacity and scalability.

For the prototype phase of the IDEAL-CITIES platform, the minikube will be utilized. It will be deployed on a VM provided by the FORTH. This approach will demonstrate a working solution that can be deployed with low resource demand. Moreover, it will also prove that it can be easily scaled up if it is needed, as the switch from the minikube to a full-blown version utilizing a Kubernetes-managed cloud will not require any fundamental changes (from the prototype version).

Choosing of Kubernetes will provide the opportunity of using the related publicly-available tools listed in this document. An example would be Prometheus that was specifically developed to monitor K8s resources while at the same time provides fast integration with Grafana in order to yield impressive power analytics within a rich interactive visualization web environment.

Lastly, Kubernetes can be found at the core of most public cloud solutions, featuring a solid foundation preferred by many IT providers. This encourages the IDEAL-CITIES consortium to support that the chosen tools will have the desired capabilities as well as wide support and therefore can deliver the desired project platform that could be easily deployed and scaled to accommodate private and cloud infrastructures.

7 References

[1] FIRST, Common Vulnerability Scoring System, https://www.first.org/cvss/ [Accessed February 2022]

[2] Mell, Peter and Scarfone, Karen and Romanosky, Sasha. 2006. Common vulnerability scoring system. IEEE Security & Privacy, 4(6), 85-89

[3] Wang, Paul and Ali, Amjad and Kelly, William, 2015. Data security and threat modelling for smart city infrastructure. 2015 International Conference on Cyber Security of Smart Cities, Industrial Control System and Communications (SSIC), 1-6.

[4] Johnson, Pontus and Lagerstrom, Robert and Ekstedt, Mathias and Franke, Ulrik, 2018. Can the common vulnerability scoring system be trusted? a Bayesian analysis. IEEE Transactions on Dependable and Secure Computing, 15(6), 1002-1015.

[5] Spanos, Georgios and Angelis, Lefteris. 2015. Impact Metrics of Security Vulnerabilities: Analysis and Weighing. Information Security Journal: A Global Perspective, 24(1-3), 57-71.

[6] Cheng, Pengsu and Wang, Lingyu and Jajodia, Sushil and Singhal, Anoop, 2012. Aggregating CVSS Base Scores for Semantics-Rich Network Security Metrics, IEEE 31st Symposium on Reliable Distributed Systems, 31-40.

[7] Prezelj, Iztok and Ziberna, Ales, 2013. Consequence-, time- and interdependency-based risk assessment in the field of critical infrastructure. Risk Management, 15(2), 100-131.

[8] A. Miaoudakis *et al*., "Pairing a Circular Economy and the 5G-Enabled Internet of Things: Creating a Class of Looping Smart Assets," in *IEEE Vehicular Technology Magazine*, vol. 15, no. 3, pp. 20-31, Sept. 2020, doi: 10.1109/MVT.2020.2991788.

[9] https://prometheus.io/docs/introduction/overview/ [Accessed February 2022]

[10] https://www.cvedetails.com/product/34016/Kubernetes-Kubernetes.html?vendor_id=15867 [Accessed February 2022]

[11] A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Koral, “Deep Packet Inspection as a Service,” in *10th ACM International on Conference on Emerging Networking Experiments and Technologies*, 2014

[12] “http://www.ntop.org/products/deep-packet-inspection/ndpi/,” [Accessed February 2022] .

[13] F. Yousaf, M. Bredel, S. Schaller and F. Schneider, "NFV and SDN, key technologie enablers for 5G networks," *IEEE Journal on Selected Areas in Communications,* vol. 35, no. 11, 2017.

[14] ETSI, "ETSI.org: Network Functions Virtualisation (NFV); Architectural Framework," 10 2013.[Online].Available: https://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf. [Accessed February 2022].

[15] ETSI OSM, "OSM Information Models," 2019. [Online]. Available: https://osm.etsi.org/gitweb/?p=osm/IM.git;a=tree;f=models/yang;h=ac67adaec00123ef4a68911ff0082fb35556b03a;hb=HEAD. [Accessed January 2019].

[16] ETSI, "ETSI.org: Network Functions Virtualisation (NFV); Management and Orchestration (ETSI GS NFV-MAN 001)," December 2014. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf. [Accessed December 2021]

[17] R. Toghraee, Learning OpenDaylight: The art of deploying successful networks, Birmingham: Packt Publishing Ltd., 2017.

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review,* vol. 38, no. 2, pp. 69-74, 2008.

[19] R. Enns, M. Bjorklund, J. Schoenwaelder and A. Bierman, "RFC 6241: Network Configuration Protocol (NETCONF)," June 2011. [Online]. Available: http://www.rfc-editor.org/info/rfc6241. [Accessed October 2021].

[20] OpenDaylight, "OpenDaylight Lithium," [Online]. Available: https://www.opendaylight.org/what-we-do/current-release/lithium. [Accessed January 2022].

[21] https://www.iron.io/ [Accessed October 2021].

[22] https://docs.aws.amazon.com/AmazonECS/latest/developerguide/AWS_Fargate.html [Accessed October 2021].

[23] https://rancher.com/ [Accessed October 2021].

[24] https://www.nomadproject.io/ [Accessed October 2021].

[25] https://kubernetes.io/ [Accessed October 2021].