UNIÃO EUROPEIA
Fundos Europeus Estruturais e de Investimento

OREOS - Orchestration and Resource optimization for rEliable and lOw-latency Services

PORTUGAL 2020

Estudos Preliminares na área do Projeto

Nº. Entregável: E2.1 **Atividade:** A2

Data: 30/04/2021 **Versão:** V1.0

Nº. Referência Doc.: OREOS_E2.1 **Nível de Disseminação:** Público

Autor(es): Universidade de Coimbra

Palavras-chave: Orquestração, alocação de recursos, monitoria e sistemas analíticos.

Resumo:

O projeto OREOS visa uma orquestração e uma utilização de recursos ótima em cenários com serviços de baixa latência e de elevada resiliência. Este documento apresenta um conjunto de tecnologias e mecanismos que permitem monitorizar e recolher métricas necessários para os mecanismos de otimização, tendo presente as plataformas de orquestração e de virtualização atualmente existentes.

Autoria e controlo de versões

Autores do Documento

Organização	Autores
Universidade de Coimbra	Bruno Sousa; Karima Velasquez; Zé Luís; Marco Silva; Nuno Lourenço; Marília Curado
Altran Portugal SA	Marco Araújo; Bruno Parreira
Instituto Pedro Nunes	David Abreu

Histórico do documento

Edição	Data	Notas
0.1	2021/04/06	Definição da estrutura do documento
0.2	2021/04/13	Integração de conteúdos
1.0	2021/04/30	Versão final

Este projeto é financiado pelo programa Portugal 2020, ao abrigo do sistema de incentivos à investigação e desenvolvimento tecnológico, contrato número POCI-01-0247-FEDER-049029.

Sumário Executivo

O projeto OREOS visa conceber e implementar uma plataforma de orquestração fim-a-fim para provisionamento e gestão de serviços críticos, incluindo comunicações veiculares, redes de distribuição elétrica, ou comunicações de emergência de segurança pública. Comunicações e redes que assentam nas tecnologias de comunicação móvel de quinta geração (5G). De forma particular, o suporte de *ultra reliable and low latency communications* (URLLC) em 5G é essencial para serviços críticos em redes veiculares e em cidades inteligentes. Os serviços críticos visados pelo projeto OREOS são caracterizados por requisitos de baixa latência e elevada fiabilidade, que beneficiam da combinação otimizada de modelos arquiteturais distintos, tais como computação *Fog*, *Edge* e Cloud. É neste contexto que este documento apresenta abordagens de otimização para serviços críticos, introduz os sistemas de monitoria, os sistemas analíticos para recolha e análise de dados necessários para os mecanismos/algoritmos inteligentes. Adicionalmente são também apresentadas as plataformas de orquestração e de virtualização de recursos que permitem uma gestão autónoma de recursos.

Abstract

OREOS aims to design and implement an end-to-end orchestration platform for provisioning and managing critical services (such as vehicle communications, electrical distribution networks or emergency communications for public safety). The evolution in the 5G networks in particular the support of services based on ultra reliable and low latency communications (URLLC), enables critical services for vehicles communications and smart cities.

The critical services targeted by the OREOS project are characterized by low latency and high reliability requirements, which benefit from the optimized combination of different architectural models, such as Fog, Edge and Cloud computing. It is in this context that the current document presents the suitable optimization approaches for critical services, introduces the monitoring and analytics systems to collect and analyse data, which is deemed as necessary for advanced mechanisms/algorithms. Additionally, the document also presents the orchestration and virtualization platforms that contribute to the autonomous and optimized management of resources.

Table of contents

List of figures

List of tables

Acronyms

AAA	*Authentication Accounting and Authorization*
API	*Application Programming Interface*
BGP	*Border Gateway Protocol*
BSS	*Business Support Systems*
CLAMP	*Closed Loop Automation Management Platform*
CNI	*Container Network Interface*
CNF	*Containerized Network Function*
CNN	*Convolutional Neural Network*
DCAE	*Data Collection, Analytics and Events*
DMaaP	*Data Movement as a Platform*
DNS	*Domain Name Service*
DSL	*Domain Specific Language*
ETL	*Extract Transform Load*
ETSI	*European Telecommunications Standards Institute*
FCAPS	*Fault configuration accounting performance security*
GUI	*Graphical User Interface*
HAS	*Homing and Allocation Service*
HNF	*Hardware Network Function*
JSON	*JavaScript Object Notation*
IaaS	*Infrastructure as a Service*
ILP	*Integer Linear Programming*
IoT	*Internet of Things*
ISG	*Industry Specification Group*
ITU	*International Telecommunications Union*
LCM	*Location-Aware Computing Management*
LDAP	*Lightweight Directory Access Protocol*
LSTM	*Long Short-Term Memory*
LXC	*Linux Containers*
MEC	*Mobile Edge Computing*
ML	*Machine Learning*
MPLS	*Multi Protocol Label Switching*
mTLS	*Mutual Transport Layer Security*

NF	*Network Function*
NFV	*Network Functions Virtualization*
NS	*Network Service*
OAM	*Operations administration and management*
OLAP	*Online Analytical Processing*
ONAP	*Open Network Automation Platform*
OOF	*ONAP Optimization Framework*
OSM	*Open Source MANO*
OSS	*Operational support systems*
OVN	*Open Virtual Network*
PNF	*Physical Network Function*
PKI	*Public Key Infrastructure*
QoE	*Quality of Experience*
QoS	*Quality of Service*
RA	*Reference Architecture*
RIE	*Radio Interfacing Equipment*
SBA	*Service Based Architecture*
SCALE	*Security Cognition Agility Latency Efficiency*
SDH	*Synchronous Digital Hierarchy*
SDN	*Software Defined Networking*
SDC	*Service Design and Creation*
SLA	*Service Level Agreement*
SO	*Service Orchestration*
SONET	*Synchronous Optical Network*
SQL	*Structured Query Language*
SOAFI	*Service Orchestrator Architecture for Fog-enable Infrastructure*
SORTS	*Supporting the Orchestration of Resilient and Trustworthy Fog Services*
SSO	*Single Sign On*
TICK	*Telegraf Influx Chronograf Kapacitor*
TLS	*Transport Layer Security*
TOSCA	*Topology and Orchestration Specification for Cloud Applications*
URLLC	*Ultra reliable and Low-Latency Communications*
QoS	*Quality of Service*

VES	*VNF Event Stream*
VIM	*Virtual Infrastructure Manager*
VM	*Virtual Machine*
VNF	*Virtual Network Function*
VNFI	*Virtual Network Function Infrastructure*
VNFM	*Virtual Network Function Manager*
VNFO	*Virtual Network Function Orchestrator*
WDM	*Wavelength Division Multiplexing*
WIM	*WAN Infrastructure Manager*

1. Introduction

OREOS aims to design and implement an end-to-end orchestration platform for provisioning and managing critical services (such as vehicle communications, electrical distribution networks or emergency communications for public safety). The evolution in the 5G networks in particular the support of services based on ultra reliable and low latency communications (URLLC), enables critical services for vehicles communications and smart cities.

The critical services targeted by the OREOS project are characterized by low latency and high reliability requirements, which benefit from the optimized combination of different architectural models, such as Fog, Edge, and Cloud computing. The application scenario runs in an urban-city context and aims to enhance the safety of people by allowing the exchange of information between the city infrastructure (e.g., video cameras) and the vehicles circulating on the roads. The communication paradigm aims to inform vehicles regarding the presence of people in the pedestrian crossings to avoid accidents.

The critical services run in diverse platforms, which require monitoring solutions to decrease the potential system failures and maximize the network performance. When integrated with analytics systems, the monitoring solutions allow the implementation of network policies in a scalable fashion, where decisions can be taken considering the collected system information and advanced visualization approaches. Monitoring Solutions like Prometheus, Nagios, Zabbix *[KOCJAN_MON]* and others are overviewed regarding their potential use in the OREOS platform. In complement, open-source platforms for data analytics like Druid *[FANGJIN_ANA]*, Drools *[LEY_ANA]*, PNDA [PNDA_GUIDE], among others are also discussed regarding their scalability to analyse high volumes of data, their architectures to distribute data, among several components/nodes to enhance processing of queries. Besides Open Source Big Data (OLAP) systems. Despite the existence of proprietary analytical solutions from major technological players like Microsoft Power BI, IBM Cognos, Google data studio and Google BigQuery, the OREOS project focuses on open source technologies to conceive the OREOS platform.

The combination of the Internet of Things (IoT), Fog, and Cloud embraces a complex scenario where in some cases it is not suitable to migrate or apply well-known solutions or mechanisms from other domains or paradigms. Two particular aspects to take into consideration are: (1) Resource Management, which requires the design and development of mechanisms that handle tasks such as Scheduling, Path Computation, Discovery and Allocation, and Interoperability; and (2) Performance that deals with Latency, Resilience, and Prediction and Optimization from the gauges, mechanisms, and algorithms point of view *[VELASQUEZ_CHALL]*. The document identifies and discusses reference architectures for resource allocation like the OpenFog

[OPENFOG_RAFC]. In addition, the Cloud-to-Object continuum requires appropriate approaches tackling resource management in an efficient manner considering complex scenarios, like the SORTS project *[VELASQUEZ_SORTS]*, or even the Mobile Edge Computing (MEC) architecture promoted by the European Telecommunications Standards Institute (ETSI) *[ETSIMEC_FRA]*.

The allocation of resources is an NP-hard problem commonly tackled by the different techniques that employ a set of decision factors like fairness aspects (e.g., deal with energy consumption, latency issues) *[GODINHO_FAIR]*, resource usage aspects (e.g., placing IoT services in Fog nodes considering Quality of Service (QoS) requirements) *[SKARLAT_RU]*, QoS aspects (e.g., placing applications in Fog nodes to maximize Quality of Experience (QoE) and reduce deployment time) *[MAHMUD_LAT]*, among other aspects.

Optimization plays a crucial role in several fields like architecture, medicine, engineering among others. Its purpose is to find the best solution for the established objective. In this regard, several steps are required for an optimization approach, where the first one includes identifying the decision variables, the definition of the objective function, and a second one devoted to the calculations to obtain the optimal solution. Several approaches for optimization can be followed, including Linear Programming (ILP) techniques that have limitations regarding the required computational effort and the size of the problem. Heuristics methods are often employed due to the reasonable quality with little computational effort. On the other hand, Machine Learning (ML) approaches are gaining momentum in diverse domains; for instance, in autonomous driving the majority of the literature is focused on multi-modal object detection and semantic segmentation *[FENG_ML]*, in resource allocation for 5G networks *[LEE_ML, WANG_ML]* with Deep Learning approaches.

Orchestration platforms support Network Function Virtualization (NFV) Management and Orchestration. For NFV Management virtualization layers are proposed to support different physical infrastructures. In addition, approaches like Software Defined Networking (SDN) are relevant to allow the management of network resources in agnostic fashion regarding the underlying physical technology (e.g., wireless, optical networks, etc). ETSI has been specifying *[ETSI_NFV]* several standards to enhance interoperability between orchestration platforms, to enable the multi-tenancy support in 5G networks, to support the Service Based Architecture (SBA), to facilitate Multi-access Edge Computing (MEC) paradigm *[ETSIMEC_FRA]*, among other critical functionalities related with NFV. Open Network Automation Platform (ONAP) *[KAPADIA_ORC]* is one of the most complete platforms to design, create and manage the lifecycle of network services. In addition, ONAP supports Physical Network Functions (PNF) along with Virtual Network Function (VNF) that can run in different virtualization infrastructures like OpenStack and Kubernetes.

The document also presents current concepts like Service Mesh that are designed to facilitate service-to-service communication through service discovery, routing, load balancing, traffic configuration, encryption, authentication, authorization and monitoring approaches *[NIST_800-204]*.

This document is structured as follows: Section 0 presents a generic application scenario, Section 3 presents the monitoring, analytical solutions, optimization approaches, virtualization and orchestration mechanisms that can be employed in OREOS, considering the general requirements, and Section 4 concludes the paper.

2. Application Scenario

The environment in which this scenario runs in of urban nature, i.e., city context, and aims to enhance the safety of people in their daily life[1]. In particular, when pedestrians are crossing the road, there is the need to inform vehicles that are approaching to reduce the velocity to stop the vehicle before reaching pedestrians.

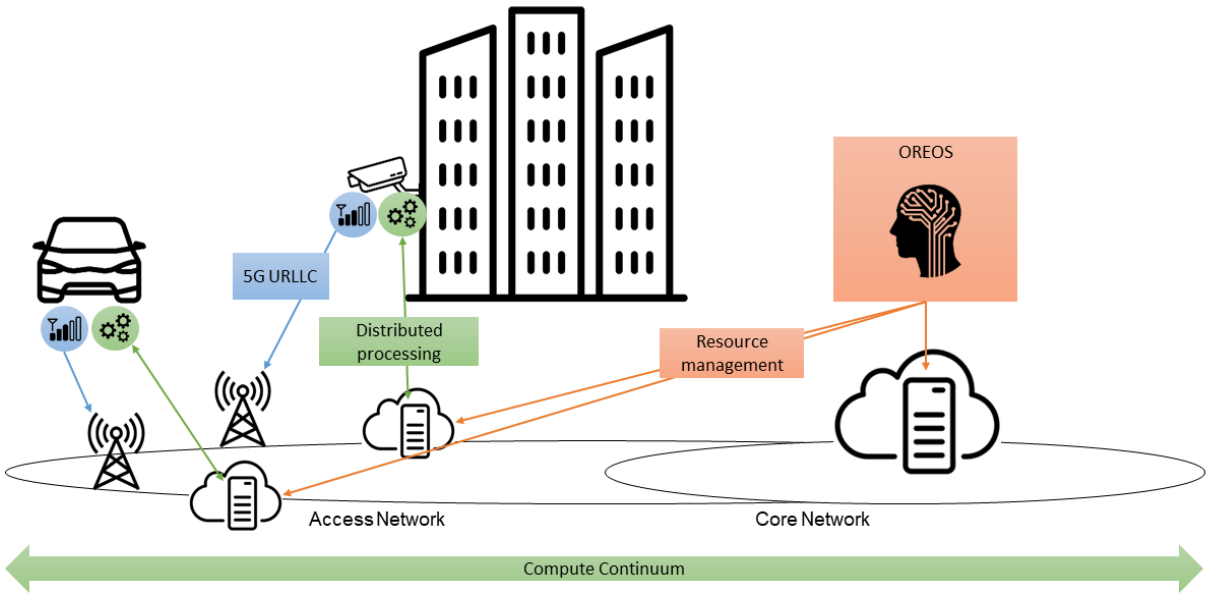

Figure 1: Reference scenario

The detection of pedestrians in the crossings or their intention to cross, is performed through video cameras, employed for city surveillance, which through intelligent mechanisms are able to detect persons, detect pedestrian crossings and the movement of users. With the network information regarding connected, nearby or approaching vehicles, there is the possibility to provide in advance the information of the pedestrian's presence on the roads. This concept is illustrated in Figure 2, whereas a traditional vehicle needs an unobstructed view to proceed safely, by means of telecommunications technology, this view can be provided by a camera or other vehicles with a better view:

[1] Considering that the whole architecture developed in this project holds applicability for URLLC Services, other potential scenarios could be e.g., services for smart cities or autonomous driving.

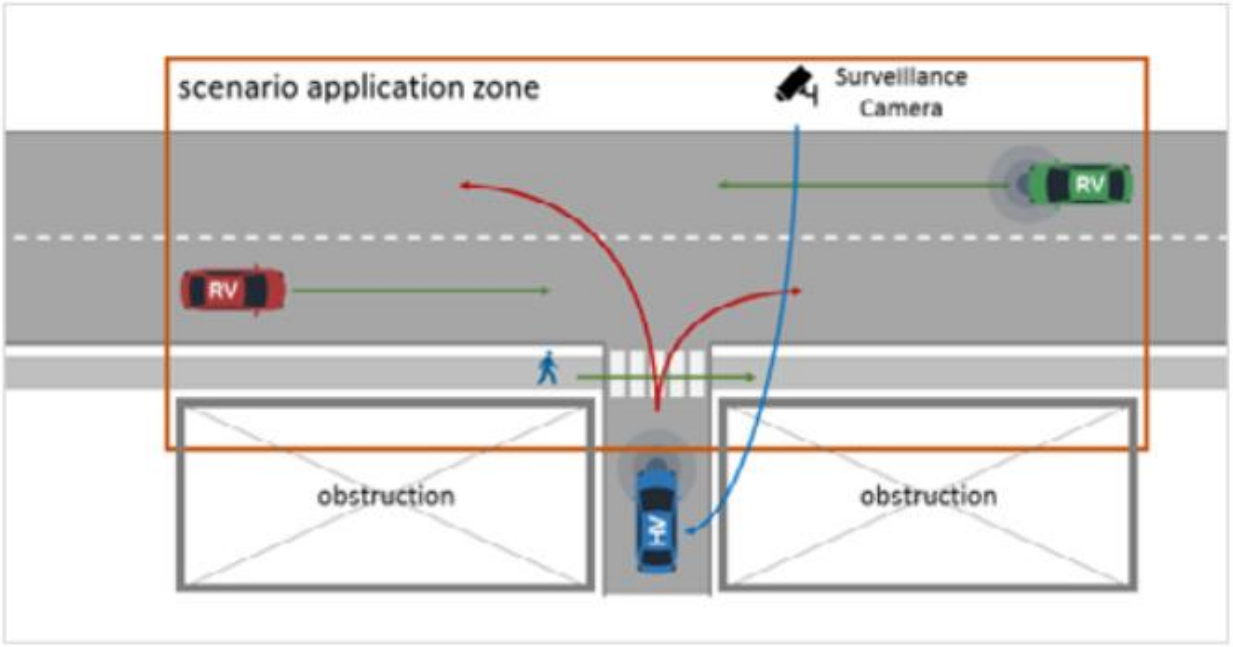

Figure 2: pedestrian crossing (source *[5GAA_USECASE]*)

Some potential service level requirements (SLR) can be devised as follows:

Table 1: Service level requirements for pedestrian crossing (source *[5GAA_USECASE]*)

SLR name	SLR unit	SLR value	Comments
Range	m	500	Message does not need an extreme range, as it only needs to reach nearby vehicles that should stop for a pedestrian. However, vehicles on high-speed roads will need adequate time to stop.
Latency	ms	100	Exchange of messages must happen quickly, but a manoeuvre will only be initiated upon agreement, so slow messaging is not necessarily a safety risk.
Reliability	%	99.9	Since a manoeuvre will only be initiated upon agreement, dropped messages will not result in an immediate and significant safety risk
Velocity	m/s	19.4	Upper end of the speed that a vehicle will be driving at on a road with a pedestrian crossing (70 km/h)
Vehicle Density	Vehicles/km^2	730	Assumes vehicles within 500 m radius of roads with 2 lanes of traffic in each direction.
Positioning	m	0.2	If a pedestrian is standing next to a roadway, it only takes a slight position error to place them in the middle of the street on a map, or directly in the trajectory of a vehicle. Alternatively, if the pedestrian is crossing, a small error could falsely indicate to a nearby vehicle that the pedestrian is on the sidewalk

3. State of the Art

This section overviews the most relevant technologies for the project, including monitoring and analytics solutions. It also overviews resource allocation approaches and the most suitable optimization mechanisms, considering the high-level requirements of the generic scenario(s) presented in Section 0.

3.1 Monitoring and Analytics Solutions

This section provides an overview of monitoring and analytical solutions.

3.1.1 Introduction

Analytical solutions allow network engineers to query, analyse and manipulate data and implement network policies using that data as an underlying asset. Monitoring solutions constitute a handy tool to maximize the network performance and to decrease the potential system failures; by focusing on computation, storage and memory resources of the infrastructure, through the analysis and visualization of the collected system information. An example of a monitoring framework is illustrated in Figure 3.

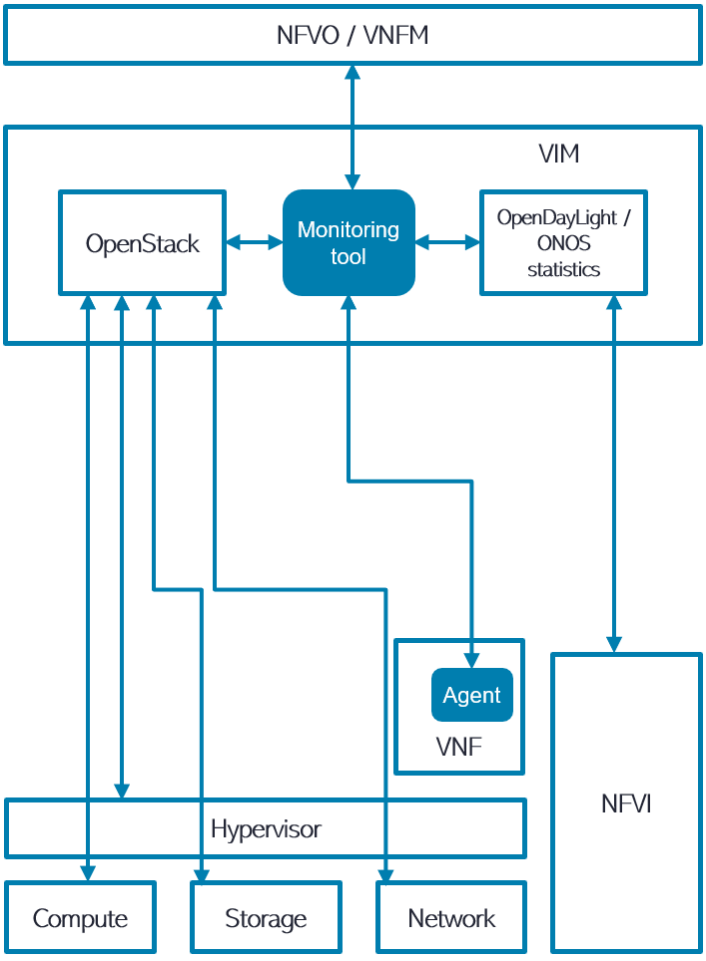

Figure 3: Monitoring framework (source: *[TNOVA_MON]*)

The monitoring tool periodically polls an OpenStack service called *ceilometer* for collecting metrics regarding all deployed physical and virtual resources and saves them into a time series database such as Influx or other. The agent polls VNF metrics and saves them into the same database. A Graphical User Interface (GUI) such as Grafana, or other, is then connected to this database to provide an interactive dashboard.

3.1.2 Monitoring solutions

This section details some of the most promising and popular monitoring solutions currently available.

3.1.2.1 Prometheus

Prometheus is an open-source metrics-based system used for event monitoring and alerting, relying on a robust data model and query language for providing the required metrics to precisely analyse infrastructure and application performance *[BRAZIL_MON]*.

The system provides libraries for the most popular coding languages, such as Python, Java, Go, C#, in order to develop custom applications. Software such as docker and Kubernetes are already instrumented in Prometheus client, and, for third-party software that exposes metrics in a data format not recognized by the Prometheus scrapper, there are exporters already available and public to make the necessary conversions to the known format. These exporters provide metrics for environments that can possibly be running, for example, applications using MySQL, PostgreSQL, SNMP, Kafka, etc.

The main features of the monitoring tool are the following:

- Multidimensional data model with time series data identified by metric name and key/value pairs
- Flexible Query Language to leverage dimensionality (PromQL)
- No reliance on distributed storage; single server nodes are autonomous
- Time series collection via a pull model over HTTP
- Pushing time series supported via an intermediary gateway
- Targets discovered via service discovery or static configuration
- Multiple modes of graphing and dashboarding support

Figure 4 illustrates the architecture of the Prometheus solution:

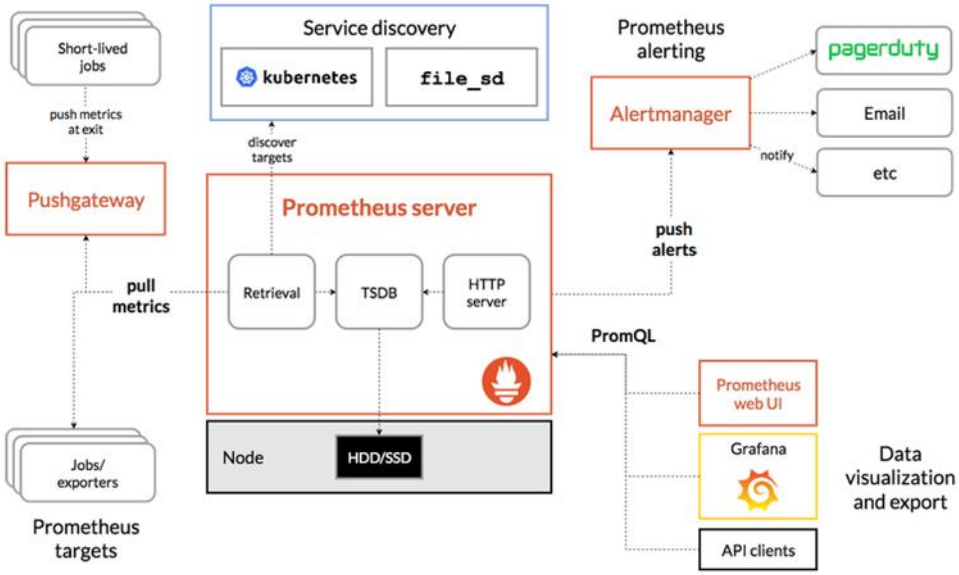

Figure 4: Architecture of Prometheus and some of its ecosystem components (source *[BRAZIL_MON]*)

Prometheus scrapes metrics from instrumented jobs/exporters directly, based on the target provided by the Service Discovery. For components that cannot be scraped in a direct manner, the Prometheus Push gateway is used to allow short-lived jobs to be pushed into an intermediary job, which Prometheus can scrape.

All scraped samples are stored locally, and Prometheus runs rules over this data to either aggregate and record new time series from existing data or generate alerts. Summarizing, Prometheus has the following components:

- Prometheus server to scrape, store and centralize time series data[2].
- Client libraries for instrumenting application code that are available in programming languages such as Go, Python, Java/JVM, Ruby (official ones), #/.NET, Node.js, Haskell, Erlang and Rust (third-party ones), in order to allow the developers to define metrics and add the desired instrumentation to the application code.
- A Service Discovery mechanism, such as Kubernetes, EC2, or Consul, to provide the target system to monitor.

[2] Prometheus stores all data as time series, which are streams of timestamped values belonging to the same metric and set of labelled dimensions and may also generate temporary derived time series as the result of queries, where each time series is uniquely identified by its metric name and optional key-value pairs called labels. Labels enable Prometheus's dimensional data model: any given combination of labels for the same metric name identifies a particular dimensional instantiation of that metric. The query language allows filtering and aggregation based on these dimensions. Changing any label value, including adding or removing a label, will create a new time series.

- A Push gateway that is used for scraping metrics from applications and passing on the data to Prometheus, when the pull method is not possible (e.g., due to a firewall).
- Multiple special-purpose exporters that typically run on the monitored host to export local metrics, since not always will it be possible to add direct instrumentation to the monitoring application and Prometheus deals with this problem with the aid of exporters, which are basically pieces of software that will run alongside the target application which the developer wishes to collect metrics.
- An AlertManager tool that receives alerts from Prometheus servers and turns them into notifications, where related alerts can be aggregated into one notification, throttled to reduce pager storms, and different routing and notification outputs can be configured for each of the different teams monitoring the system.
- A GUI dashboard such as Grafana for reporting purposes that facilitates data analysis, especially when the scope is to assess metrics (Grafana supports the time-series databases from Prometheus and InfluxDB, as well as MySQL and OpenTSDB).

3.1.2.2 Telegraf

Telegraf is a plugin-driven server agent for collecting and sending metrics and events from databases, systems, and IoT sensors. The agent can collect a variety of metrics from a wide variety of inputs into a wide array of outputs. Telegraf is a component of the TICK stack, promoted by Influx *[YTSENG_RTMS5NCP]*.

This is possible because Telegraf has many available plugins at the user's disposal, written by experts, providing great flexibility for the agent to adapt to any of the developer needs. If the required plugin is not available, it is also possible to write a custom plugin to satisfy the developer needs. Among its features, the following can be highlighted:

- Custom high performance datastore written specifically for time series data
- Expressive SQL-like query language to easily query aggregated data (Flux)
- Plugins support for other data ingestion protocols
- Simple, high performing write and query HTTP APIs.
- Continuous queries automatically compute aggregate data to make frequent queries more efficient

Figure 5 illustrates the architecture of the Telegraf solution:

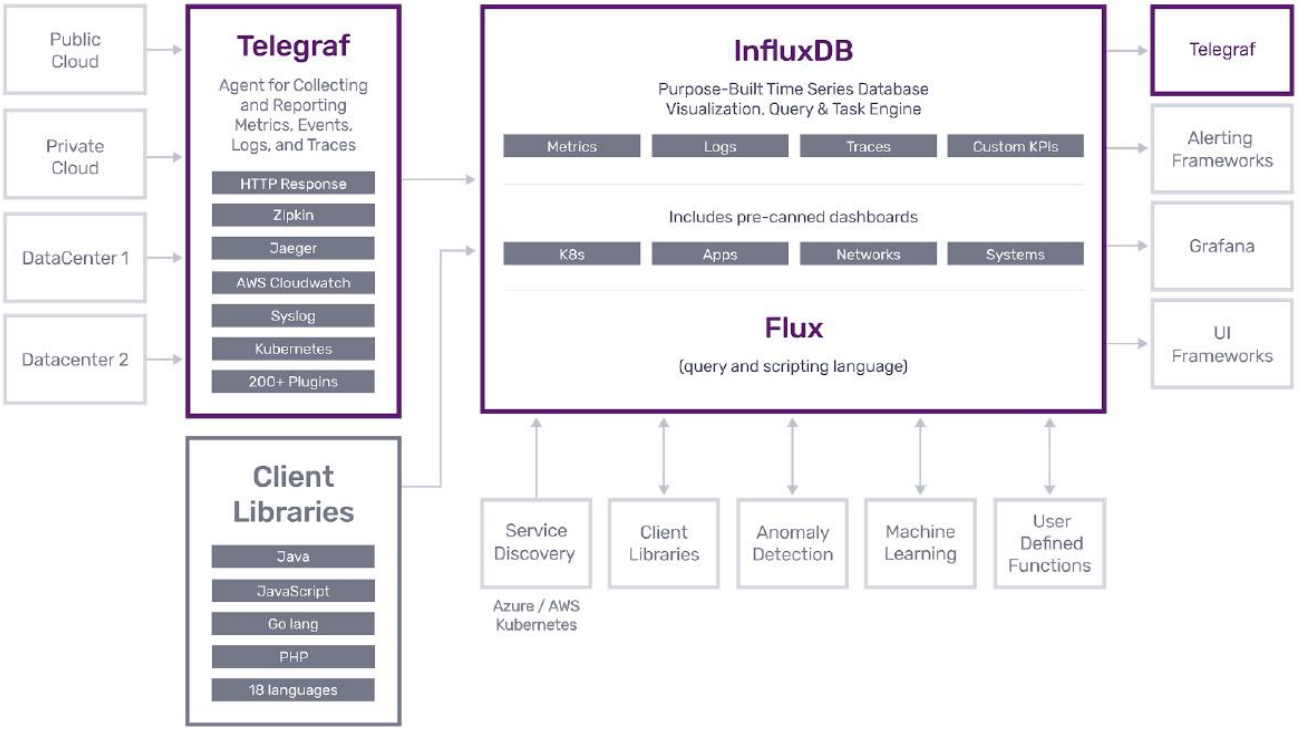

Figure 5: Telegraf architecture (source *[YTSENG_RTMS5NCP]*)

The ecosystem includes a time series database[3], real-time analytics engine and visualization pane and has integrated input plugins to allow pulling metrics from the system or third-party APIs. Also features output plugins that, in this case, will allow the data to be sent to a host of external data stores, services and message queues. If the need arises, there is also the option to build custom plugins upon the platform and make them fit into the monitoring tool.

3.1.2.3 Nagios

Nagios is an open-source tool designed for system monitoring, whose main purpose is to provide constant feedback on monitored systems, networks and IT infrastructure (active checks), being capable of receiving information, regarding other processes or machines, from systems it does not actively monitor (passive checks). System monitoring is split into two different categories, hosts and services. Hosts represent the physical or virtual device in the network, such as routers, switches, servers, etc. For the services, these represent particular functionalities to be monitored, which are running on a host machine, for example, a SecureShell server.

[3] InfluxDB provides a purpose-built time series database to store all the metrics, events and logs in a central platform. The platform also provides a vast set of plugins for the Telegraf collecting agent, and many client libraries for the most popular programming languages, such as Go, Python, Java, Node.js, among others. These libraries allow developers to provide integration between the platform and the applications for data analytics.

In order to perform all of the necessary checks, Nagios uses plugins. These are simply external components, Nagios passes information to be checked and what are the limits which trigger the warning and critical state. Nagios comes with a set of standard plugins that allow performance checks for almost all the services mostly used by organizations. If the plugin is not available, it is possible to write a custom plugin according to the developer needs.

Designed with scalability and flexibility in mind, Nagios main features are as follows *[KOCJAN_MON]*:

- Commands which define how Nagios should perform on particular types of verifications.
- Time periods defining the time periods in which operations should or should not be performed.
- Hosts and host groups to aggregate multiple hosts into a single group. A single host can also be part of various groups.
- Services, various functionalities or resources to monitor on the host.
- Contacts to notify the respective teams dealing with information of how and when they should be contacted.
- Notifications defining who should receive each of the notifications.
- Escalations are an extension of the notifications, defined after a specific object is in the same state for a period of time, and send the information to a different group to handle the situation.

Figure 6 illustrates the architecture of the Nagios solution:

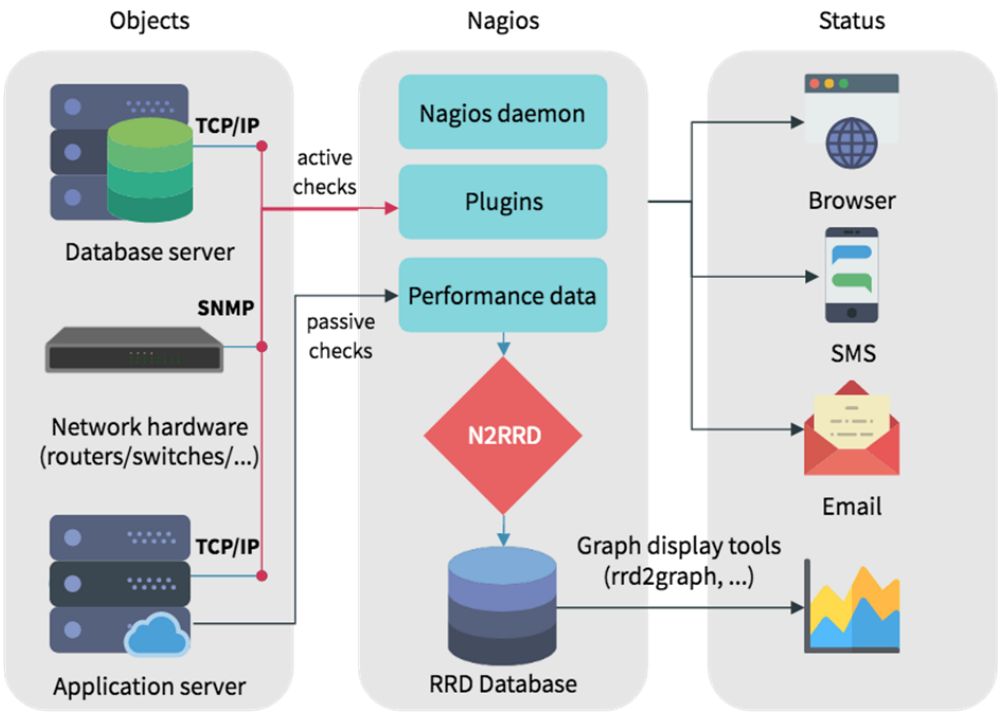

Figure 6: Nagios architecture (source *[KOCJAN_MON]*)

Nagios stores all the numeric data collected from systems and services in a RRD Database. To make use of this database it is used N2RRD (Nagios to Round Robin Database) which is a Nagios add-on tool, which stores performance data generated by Nagios plugins into a Round-Robin-Database.

3.1.2.4 Zabbix

The Zabbix monitoring solution is designed for a server/agent architecture, in which the Zabbix server runs on a standalone machine to be able to collect and aggregate monitoring data sent by the Zabbix agents. In addition, Zabbix supports VMware monitoring, which is used to monitor virtual machine statistics, whereas low level discovery rules are used in order to discover virtual machines and hypervisors *[LIEFTING_MON]*.

Figure 7 *[SHOKHIN_MON]* illustrates a complete architecture that contains all Zabbix components:

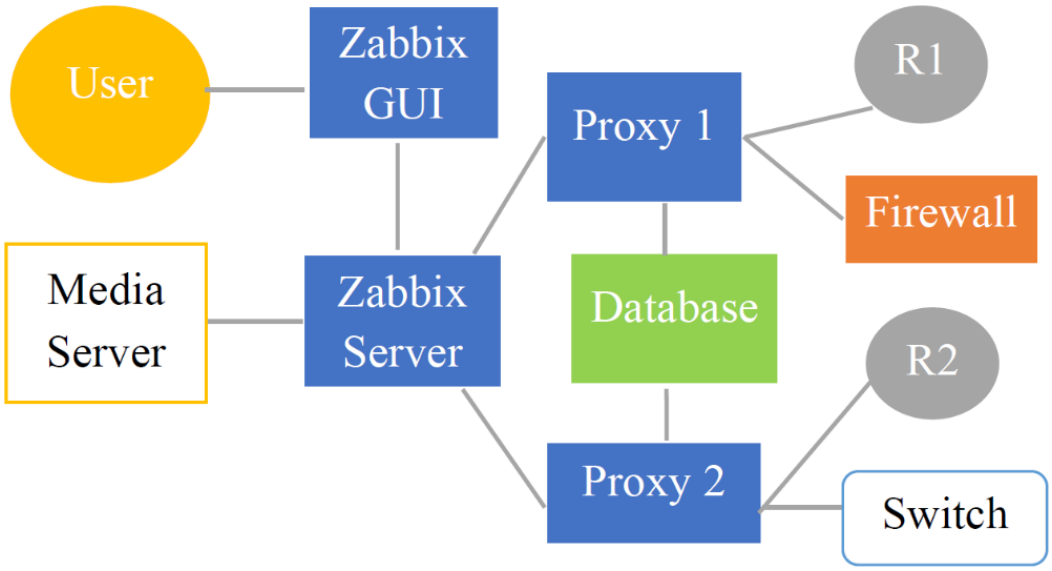

Figure 7: Zabbix components (source *[SHOKHIN_MON]*)

- Zabbix Server: its main goal is to perform remote monitoring of the network itself and its components and to store configurations, historical and operational data.
- Zabbix Proxy: collects local level data into a buffer that will be forwarded to Zabbix Server.
- Zabbix agent: performs local monitoring of the network devices, by monitoring resources such as hard drive, memory and CPU statistics. In order to perform resource monitoring, Zabbix agent has to be locally installed on each device.
- Web Interface/GUI: is launched on the same physical server where Zabbix Server is running.
- Media Server: is responsible for sending alert notification via Email and SMS.
- Database: is used to store configurational and historical data.

3.1.2.5 Summary

Both Zabbix and Nagios are "out-of-the-box" solutions [*HARTUNGA_MON*, *TRAKADAS_MON*] (i.e., they constitute a complete monitoring solution), while Prometheus and Telegraf are not functional without being integrated with other tools (in fact, Prometheus and Telegraf can work together in a single architectural solution):

- Prometheus requires a third-party exporter. An *exporter* is a tool used to help expose metrics for Prometheus (similar to *agents* for Zabbix). Telegraf can be used as an exporter for Prometheus.
- While Zabbix and Nagios have a build-in alerting functionality, to manage alerting with Prometheus it is necessary to install AlertManager separately.
- A GUI such as Grafana needs to be installed, also separately, unlike Zabbix or Nagios who come with a functional build-in GUI.

Table 2 displays a comparison between these solutions *[TAHEROZADEH_MON]*.

Table 2: Comparison between Prometheus + Telegraf, Nagios and Zabbix's functional requirements

Tool	Collection	Container monitoring	Application monitoring	Data storage
Prometheus+Telegraf	Push/Pull	Yes	Yes	InfluxDB
Zabbix	Push/Pull	Yes	Yes	Oracle, MySQL, PostgreSQL, SQLite
Nagios	Pull	No	No	MySQL

In terms of functional requirements, Nagios is clearly outranked. Using Prometheus the solution consists of deploying the Telegraf agent on every host that requires monitoring (but it is only necessary to have just one Prometheus running). Since Telegraf uses output plugins, all that is necessary in order to pass metrics to Prometheus is an *outputs.prometheus_client* output plugin. Then for reading metrics one can use Telegraf's input plugins such as *inputs.mem, inputs.cpu,* etc. (one big advantage is that Telegraf supports Nagios plugins as well). While the usage of InfluxDB is optional, both Grafana, Prometheus and Telegraf are compatible with it; therefore InfluxDB can be used as well (Prometheus is designed for short period data, thus using InfluxDB as an external storage mechanism is highly recommended).

Prometheus holds two significant advantages when compared with Zabbix: i) Zabbix currently does not support the feeding of collected metrics from an agent to a Kafka bus, while Prometheus offers this possibility; ii) Zabbix cannot pull metrics from Collectd, while Prometheus can. Thus, despite the complexity that installing several

third-party addons requires, a combination of the tools Prometheus + AlertManager + Telegraf + InfluxDB + Grafana if well configured can offer a broader scope than Zabbix.

Figure 8 exemplifies a solution based on several tools for a complete monitoring solution focused on:

- Virtual Network Functions (VNF)
- SDN
- Compute virtualization
- Storage virtualization
- Network virtualization
- Data plane

Of the following solutions:

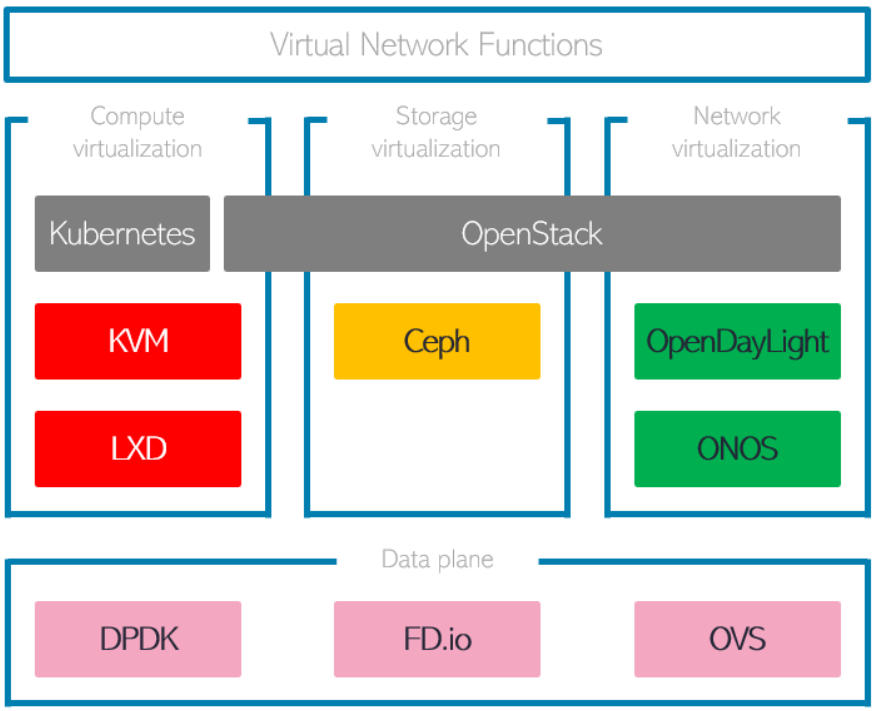

Figure 8: Example of solution for monitoring with diverse tools

A generic architecture for Prometheus/Telegraf monitoring and analytics for these components is illustrated in Figure 9:

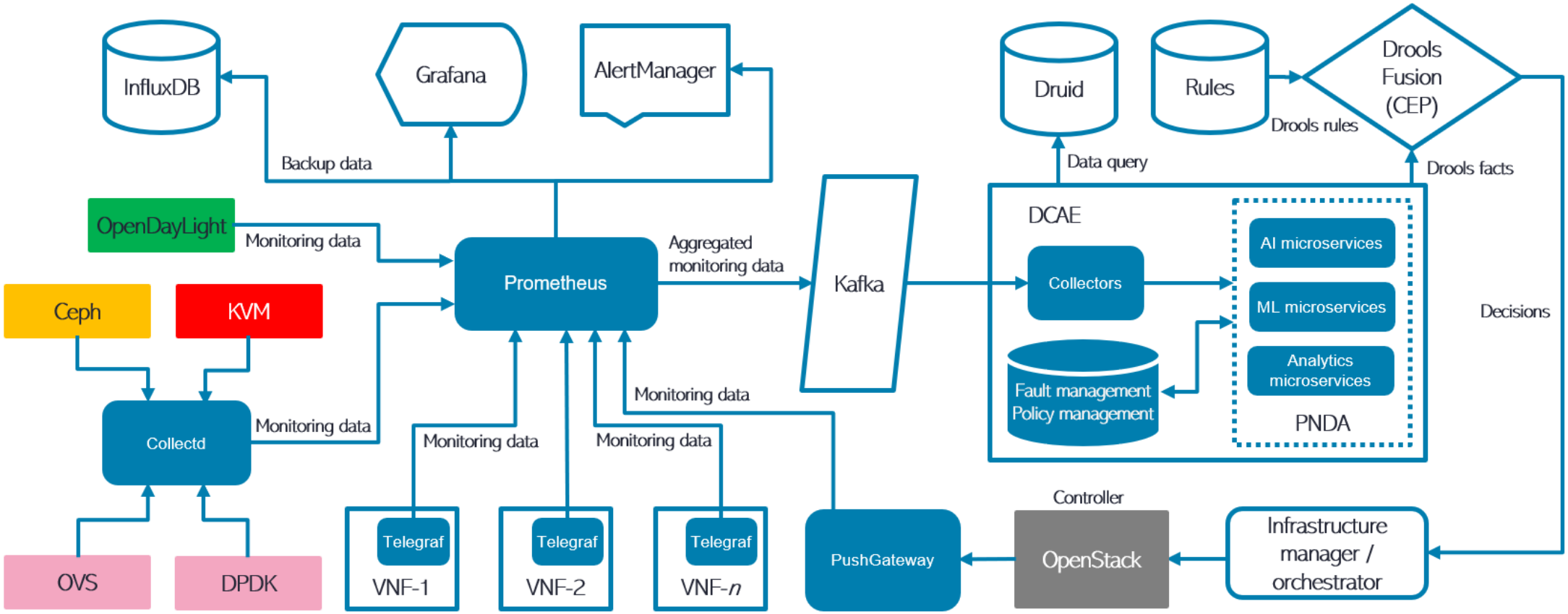

Figure 9: Architecture of monitoring & analytics using Prometheus+Telegraf

- A Telegraf agent is locally installed in each NFV.
- OpenStack offers a module[4] called *ceilometer.publisher.prometheus* that publishes metering data from OpenStack to Prometheus Pushgateway endpoint.
- OpenDayLight offers a feature[5] called *odl-infrautils-metrics-prometheus*, in which Prometheus is configured with the URL of ODL (OpenDayLight) nodes, and scrapes metrics from ODL in configurable regular intervals.
- Collectd is a Unix daemon that can collect statistics from OVS[6], DPDK[7], Ceph[8] and KVM[9] and transfer the data to Prometheus via exporter[10].
- Open vSwitch (OVS) can use the DPDK library to significantly improve its performance, which means that these two are not mutually exclusive.
- Prometheus aggregates the data from the previous mentioned sources, creates a back-up of the data into an InfluxDB and using an adapter feeds this aggregated data to a Kafka bus[11].
- An intelligence layer such as Data collection, analytics, and events (DCAE) from ONAP reads the data from the Kafka bus, proceeds to process the monitoring data in order to perform analysis on said data and eventually applies artificial intelligence and machine learning techniques, through microservices via a tool such as PNDA.
- Druid connects to this intelligence layer for data querying purposes.
- Drools Fusion is the module responsible for adding Complex Event Processing (CEP) capabilities into the platform; when analysing Figure 12, the CEP can be seen as being the inference engine and the facts are the contents of the Kafka bus.

But in reality, the architecture for the monitoring component will depend on the chosen orchestration platform, since most platforms already have their monitoring and analytics solution build-in. For example, the orchestrator OpenBaton has is integrated with Zabbix for monitoring (see section 3.3.1.3.1). ONAP also has its own monitoring and analytics solution already integrated (see section 3.3.1.1).

[4] https://docs.openstack.org/ceilometer/rocky/api/ceilometer.publisher.prometheus.html
[5] https://docs.opendaylight.org/projects/infrautils/en/latest/getting-started-guide/features.html#metrics-prometheus-implementation
[6] https://collectd.org/wiki/index.php/Plugin:OVS_Stats
[7] https://collectd.org/wiki/index.php/Plugin:DPDK_Telemetry
[8] https://collectd.org/wiki/index.php/Plugin:Ceph
[9] https://collectd.org/wiki/index.php/Plugin:virt
[10] https://github.com/prometheus/collectd_exporter
[11] https://github.com/Telefonica/prometheus-kafka-adapter

3.1.3 Analytics solutions

This section details some of the most promising and popular analytics solutions currently available.

3.1.3.1 Druid

Druid is an open-source tool for real-time exploratory analytics on large data sets with sub-second latencies, that has its own query language and accepts queries as POST requests with a body in the form of a JSON object. Figure 10 illustrates its architecture *[FANGJIN_ANA]*.

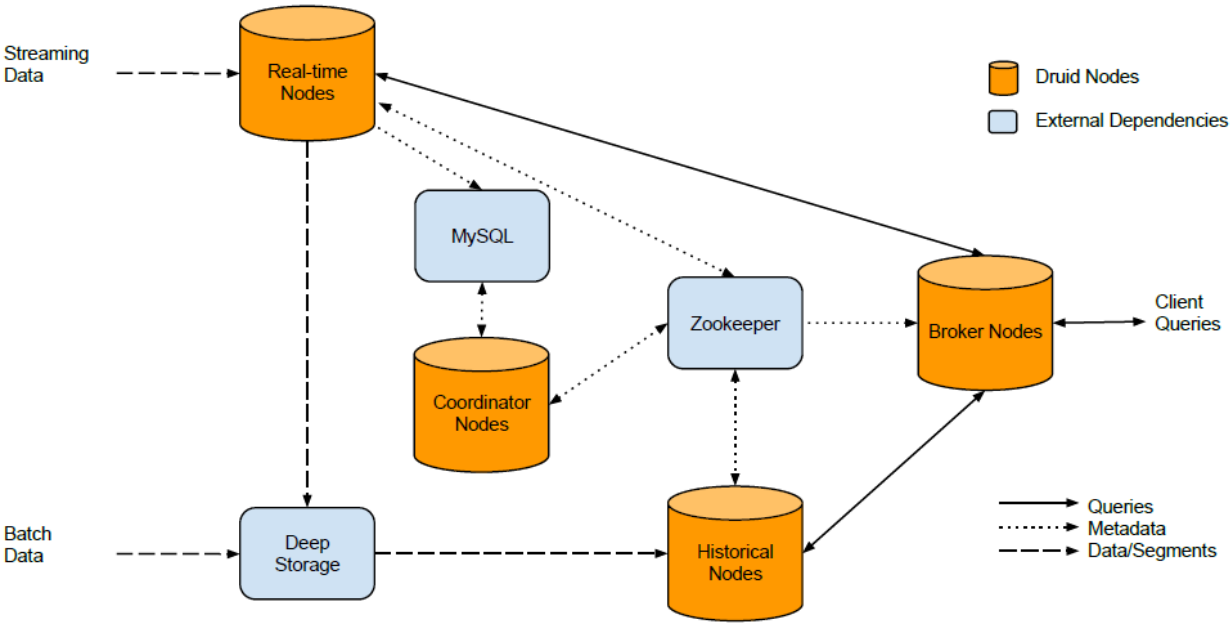

Figure 10: Druid architecture

- Zookeeper: is a centralized service for maintaining configuration information and providing distributed synchronization. This is where nodes announce their online state and the data they are serving in.
- Real-time nodes: a producer (e.g., a Telegraf output plugin) provides data stream to a Kafka message bus, from which the real-time node ingests the data, as per illustrated in Figure 11. The advantage of this solution is that a positional offset that indicates how far a consumer (a real-time node) has read in an event stream, exists. So, in the event of a recover scenario, a node can reload all persisted indexes from disk and continue reading events from the last offset it committed.

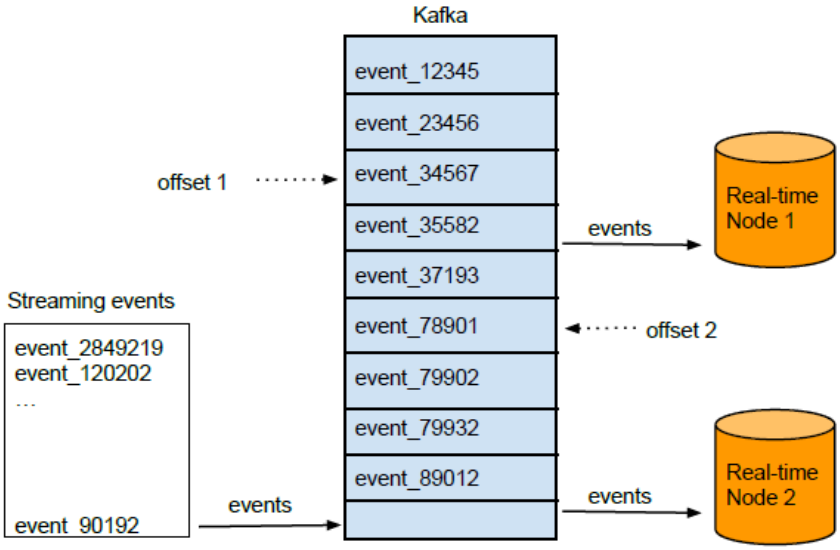

Figure 11: Kafka bus serving real-time nodes

- Historical nodes: are used to load blocks of data created by real-time nodes. The instructions to load and drop segments are sent over Zookeeper and contain information about where the segment is located in deep storage and how to decompress and process the segment.
- Coordinator nodes: these are used to tell historical nodes to load new data, drop outdated data, replicate data, and move data to load balance. They run periodically and make decisions by comparing the actual state of the cluster (this information is obtained through the Zookeeper) with its expected state. A connection to a MySQL database exists with the purpose of loading additional operational parameters and configurations if needed.
- Broker nodes: they act as query routers to historical and real-time nodes.

3.1.3.2 Drools

When a request reaches the decision centre of a telecommunication system, the decision maker is forced to select a choice that invariably, can output different outcomes. Not only analysing the pros and cons of each possible outcome is time consuming, but when in current telecommunications systems these requests appear in rapid and large amounts, the need for decision automation becomes imperative.

Decision automation in simple terms, refers to the coupling of an action with a condition or policy, i.e., when a certain condition or policy is met, then a specific action will be triggered automatically without human manual intervention. This is what is referred to as *rules*.

Drools is an open-source rules engine for applying conditional actions (if/then rules) to data. A rule's basic structure is as follows *[SALATINO_ANA]*:

```
rule "name"
when
        (Conditions)
then
        (Actions/Consequence)
end
```

One can see these conditions as being essentially filters that exclude the data which does not meet a defined criterion. Once this so-called filter is applied, an action is scheduled to be executed. An important characteristic of these rules is that they are based on the *declarative programming* paradigm, which means that the placement of the rule inside the code is irrelevant when determining the flow control of the code.

The core component of the Drools rules engine is the inference mechanism (also known as *inference engine*), which is used to make an assessment of the current state of the environment in order to identify the set of rules that constitute good candidates given the environment's state. In other words, the inference mechanism matches rules against facts or data to infer conclusions that result in new facts or in actions. The Drools inference mechanism uses an improved Rete algorithm for pattern matching, adapted for object-oriented systems, and for Java in particular. Figure 12 *[LEY_ANA]* illustrates the main components of a rule's engine:

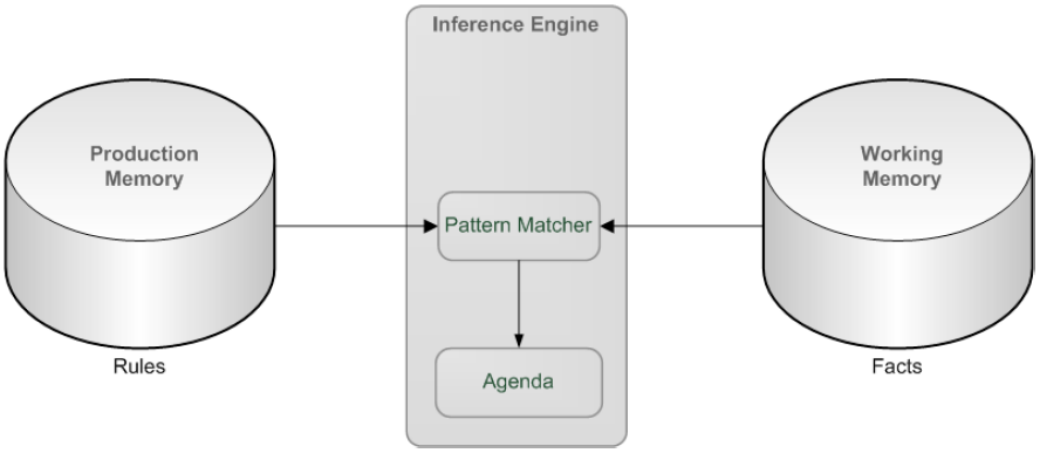

Figure 12: Rule's engine (source *[LEY_ANA]*)

Rules are stored in the production memory, while facts are maintained in the working memory. During a session no rules can be added or removed from the production memory, while facts on the other hand, can be added,

removed or changed in the working memory, by executing rules. Once a change occurs in the working memory, the inference engine is triggered and established which rules are valid for the given facts. In the event that more than one rule is valid, their execution is managed by the Agenda, using a conflict resolution strategy. When a selected rule has been executed, and one or more changes were done in the working memory, the inference engine goes to work again, adapting the agenda and executing the rule that has now reached the highest priority. This iterative process continues until the agenda is empty. Then the analysis session terminates, and results can be queried from the working memory.

3.1.3.3 Kapacitor

The Kapacitor is also an element of the TICK stack promoted by Influx *[YTSENG_RTMS5NCP]*. In particular, Kapacitor is considered as an open-source data processing framework to create alerts, to run Extract Transform and Load (ETL) jobs and to detect anomalies.

Kapacitor is able to support user defined functions to process alerts, analyse metrics for patterns, compute statistical anomalies and perform actions on such alerts *[DHIRAJ_PDAUPOST]*. The actions can be integrated with solutions like HipChat, Opsgenie, Alerta, Sensu, Slack and others to allow, for instance, the notification when a certain value is below a defined threshold.

Kapacitor can be instrumented via a graphical user interface, or via an API, where programs can configure the analysis process in Kapacitor and the specific actions upon certain values (e.g., temperature exceeds 2ºC of predefined threshold in the previous 5minutes). The short comes of Kapacitor are mainly related to the streaming of data that heavily relies on InfluxDB and has no support for advanced analysis mechanisms that can be empowered by Machine Learning techniques. In addition, some authors point scalability issues in the TICK stack (e.g., no clustering support) *[SKJENSEN_TSMS]*, which may impact analysis of high volumes of data. In addition, Kapacitor does not hold features to be classified as an Online Analytical Processing (OLAP) system, as Druid.

3.1.3.4 PNDA

PNDA (Panda) is an open-source Platform for Network Data Analytics, which aggregates data from multiple sources on a network including, real time performance indicators, logs, network telemetry, and other useful metrics and works in combination with Apache Spark in order analyse the data to find useful patterns *[PNDA_GUIDE]*.

PNDA delivers processed data to downstream applications, where it can then be evaluated. It does this using Apache's Kafka and Zookeeper applications to distribute high velocity data. It performs the following procedures with its respective embedded technologies:

- Data entry by a multitude of options, e.g., Logstash, OpenDaylight, Bulk Ingest, SNMP, Telemetry, etc.
- Data distribution by Apache Kafka and Zookeeper.
- High-speed stream processing with Spark Streaming.
- High volume batch processing with Spark.
- Free-form data exploration with Jupyter.
- Structured consultation on big data with Impala.
- Time series manipulation with OpenTSDB and Grafana.

A standard installation includes the following:

- SaltStack: is an open source Python-based software for event-driven IT automation, remote task execution and configuration management.
- OpenStack Heat templates: Heat implements an orchestration mechanism to activate various composite cloud applications based on templates in the form of text files that can be treated as code.
- AWS CFN templates: AWS CloudFormation offers a common language for modeling and provisioning resources from AWS and third-party applications in a cloud environment.
- Apache Kafka: it is literally the "front door" for the entry of data flows originated by the network equipment.
- Bulk Ingest: acting as an alternative to Kafka to accommodate existing data migration scenarios for PNDA.
- Zookeeper for Kafka: is used by Kafka for discovery and search requests from brokers regarding the partitions you want to consume.
- JMX Proxy: used for, among other things in PNDA, the exposure of all MBean attributes available in a given JVM through a simple HTTP request.
- Apache Impala: acting as a parallel execution mechanism for SQL queries with low latency access and interactive data exploration in HDFS and HBase. Impala allows data to be stored in raw format, with aggregation performed at the time of consultation, without the need for initial data aggregation.
- CMAK (aka "Kafka Manager"): as a Kafka cluster management tool.

- ELK (Logserver) Elasticsearch: one of the components of the log architecture employed by PNDA.
- Logstash: one of the components of the log architecture employed by PNDA, used to receive data from numerous sources, transformation and data sharing.
- Kibana: one of the components of the log architecture employed by PNDA, acting as an Elasticsearch data visualization plugin.
- Jupyter Hub and Jupyter Notebook: acts as an application that allows you to create and share documents that contain active code, equations, visualizations and explanatory text. In PNDA, it supports the exploration and presentation of data from HDFS and HBase.
- OpenTSDB: acts as a scalable time series database that allows you to store and supply large amounts of time series data, without losing granularity.
- Grafana: acts like a graph and panel builder to visualize PNDA time series metrics.
- Anaconda: Anaconda is a free and open source distribution of Python and R programming languages for scientific computing. Used for several PNDA functions.
- Consul.io: for managing endpoints and discovering services.
- Apache Flink: acts as a structure and a distributed processing mechanism for stateful calculations on unlimited and limited data flows. Flink is designed to perform in all common cluster environments, perform calculations at memory speed and at any scale.
- Apache Knox: acts as an application gateway to interact with the REST APIs and UIs of Apache Hadoop deployments, providing a single access point for all REST and HTTP interactions with Apache Hadoop clusters.

PNDA in fact proposes to be a scalable and open big data platform, with the purpose of supporting operational and business intelligence analysis for networks and services.

3.1.3.5 Pinot

Pinot is a real-time distributed OLAP datastore *[JEAN_PINOT]* and is an incubator apache project[12]. Pinot as an apache project, has been designed to answer OLAP queries with low latency being able to ingest data in near real-time from streaming data-sources, and has also been designed for high performance, to support a high number of queries per second *[YFU_RTDIUBER]*.

[12] https://pinot.apache.org/

Pinot is employed by enterprises due to the scalability support and the capability to enable near real-time OLAP services and distributed OLAP data store. The internal architecture of Pinot is identical to Druid, but introduces some optimizations regarding the data structures, such as compression of indices to lower the data footprint and to support faster query execution. The architecture of Pinot is illustrated in Figure 13, highlighting the main components, which include controllers, brokers, servers and minions.

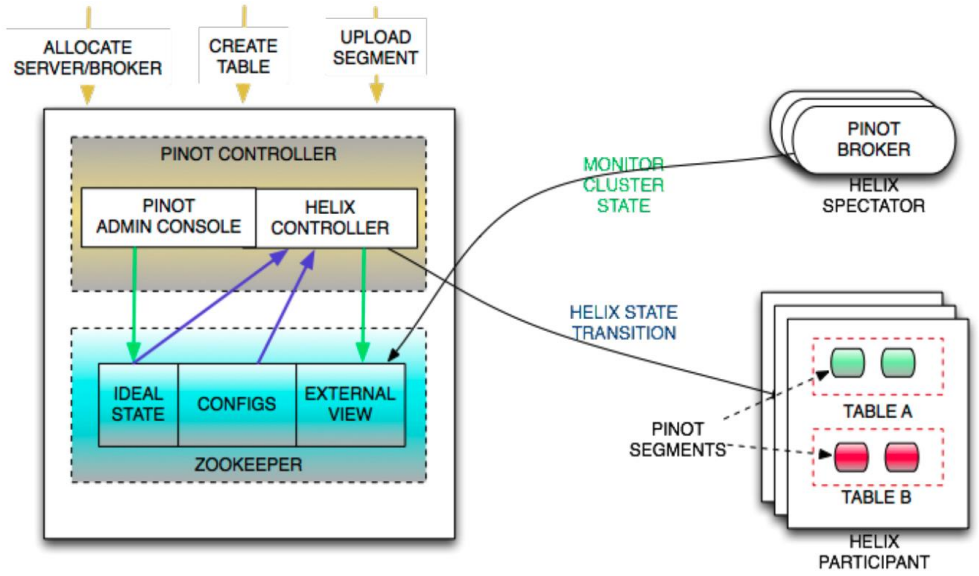

Figure 13: Architecture of Pinot (source *[JEAN_PINOT]*)

The cluster management of Pinot relies on the Apache Helix solution. The server component hosts segments and performs the processing on segments, which can correspond to a UNIX directory. The controller component is responsible for maintaining the mapping between segments and servers. On the other side, the brokers receive the incoming queries and route them to the appropriate server instances and aggregate the results from multiple servers.

3.1.3.6 ClickHouse

The ClickHouse is an OLAP database management system, developed by a Yandex company and employed by Cloudflare for HTTP and DNS analytics *[ROMAN_COSOLAP]*. ClickHouse achieves performance by supporting ingestion of data streams (e.g., from Kafka) and by adopting column stores.

The architecture of ClickHouse differs from Pinot and Druid, in the sense that ClickHouse does not support the concept of segments, and there is not a true separation between storage and processing. For instance, a ClickHouse node performs query processing and persistence of data. While such an approach can provide

benefits in terms of deployment (no need for dedicated data management servers and no need to process data in segments), there can be issues in partitioning the data of big tables between several nodes, as the queries might need to be processed by a high number of nodes. In addition, the growth of the cluster might need to be adjusted in order to weigh the partitioning of data between nodes.

ClickHouse is more aligned with traditional database systems, and its simplicity in terms of deployment can be beneficial in some scenarios. For instance, a single server might be enough to process the OLAP queries *[ROMAN_COSOLAP, YFU_RTDIUBER]*.

3.1.3.7 Non open source analytical solutions

Even though this project relies essentially on open-source tools, it is worth to mention that some valuable non-open source analytical solutions exist in the market. Three of such solutions are Microsoft PowerBI, IBM Cognos and Google different data tools.

Microsoft Power BI is a business intelligence software for data collection, analysis, and reporting that utilizes Microsoft artificial intelligence (AI) technology to predict trends and future performance, as well as to diagnose performance issues. It offers seamless Excel integration for spreadsheet application.

IBM Cognos is an AI-powered platform that can feed in data from third-party tools like Google Analytics. It offers custom dashboards, data exploration using conversational language, detection of anomalies in the data through pattern detection techniques and trend forecast using artificial intelligence.

Google has a whole suite of different services that can be bonded together, such as Google analytics (tracks and reports traffic and associated data), Google data studio (a cloud-based visualization tool with an interactive dashboard) and Google BigQuery (as the name says, to query large chunks of data).

3.1.3.8 Summary [ALTRAN, with UC collab]

Table 3 displays a comparison between the tools described in this section. It is important to note that some of the solutions presented in this table do not provide a full set of features for monitoring or analytics, and in some cases, they complement each other. Such as is the case of Druids and Drools for example.

Table 3: Monitoring solutions

Approach	Advantages (++)	Disadvantages (--)
Druid	• Permits the execution of data queries with extremely low latency	• Its architecture is not straightforward and easy to deploy
Drools	• It is based on declarative programming • Rules are easy to set up	• Documentation is difficult to go through
PNDA	• ONAP is trying to make it a default part of their framework, so support would increase and there would be no need for extra tool configurations • Extremely complete: a standard installation comes with a multitude of different tools	• No activity from its developers in the last two years (could be discontinued)
Kapacitor	• Easier deployment • Provides APIs to allow the definition of rules by other services and applications	• Requires InfluxDB time series database • May have scalability issues with high volumes of data
Pinot	• Modular architecture • Support for high volumes of Big data • Efficient mechanisms for clustering and distribution of data (segment approach) • Designed to reduce OLAP queries	• In simple deployments adds complexity
ClickHouse	• Supports scalability • Facilitates deployments	• More oriented to database systems, which may lead to issues with high volumes of data • Cluster management requires manual intervention

3.2 Resource Allocation mechanisms

This section presents to perform resource allocation in diverse contexts, networks.

3.2.1 Introduction

Fog computing brings challenges at multiple levels. Looking from a broader perspective, one of the first challenging issues is the modelling of the orchestration component that needs to be able to perform the deployment of the Cloudlets *[CHEN_CLOUDLETARCH]* and handle tasks inside the Cloud/Fog ecosystem.

The combination of IoT, Fog, and Cloud embraces a complex scenario where in some cases it is not suitable to migrate or apply well-known solutions or mechanisms from other domains or paradigms. Two particular aspects to take into consideration are: (i) Resource Management, which requires the design and development of mechanisms that handle tasks such as Scheduling, Path Computation, Discovery and Allocation, and Interoperability; and (ii) Performance that deals with Latency, Resilience, and Prediction and Optimization from the gauges, mechanisms, and algorithms point of view *[VELASQUEZ_CHALL]*.

Regarding the resource allocations approach to implement in the Cloud/Fog ecosystem, a key factor to consider is the trade-off between different (sometimes competing) optimization parameters. For instance, it could be required to minimize the energy consumption while also minimizing the latency. These represent a multi-objective optimization problem that renders the allocation process into a non-trivial problem.

These new challenges require novel solutions tailored for the Fog landscape. Some architectures aimed at the orchestration of Fog environments are presented below.

3.2.2 Architectures for resource allocation

The Fog needs to support the orchestration of applications and services on demand, with adaptability, while providing flexible performance. In practice, traditional service orchestration approaches that were applied to Cloud services are not suitable for the large scale and dynamism of Fog services, since they cannot effectively treat the prominent characteristics of the Fog's distributed infrastructure. The OpenFog consortium defined a set of guidelines to orchestrate the Fog, which can be done in a centralized or distributed manner. This section presents the OpenFog consortium architecture, a description of Fog orchestration approaches, and a set of Fog orchestration architectures that tackle the Fog resource allocation problem.

3.2.2.1 OpenFog Consortium Reference Architecture (guidelines)

The OpenFog Consortium has documented a reference architecture *[OPENFOG_RAFC]* and clarified the definition of Fog Computing. The OpenFog Reference Architecture (OpenFog RA) was conceived to enable Fog-Cloud and Fog-Fog interfaces following the **SCALE** approach:

- **S**ecurity -- using the security by design approach to ensure safe and trusted transactions;
- **C**ognition -- enabling awareness of client-centric objectives to achieve autonomy;
- **A**gility -- supporting rapid innovation and affordable scaling under a standard infrastructure;
- **L**atency -- providing real-time time processing, as well as, cyber-physical system control; and
- **E**fficiency -- managing dynamic pooling of local and remote resources to enhance the QoS of end-users.

The SCALE approach allows present Fog computing as a paradigm to distribute computing, storage, control, and network functions closer to users, creating a Cloud-to-object continuum. Fog computing contributes to achieve resilience/reliability, as per the definition of the International Telecommunications Union (ITU) body, where reliability is considered as the capability to transmit a certain amount of traffic (i.e., 32 bytes), in a pre-determined time duration (1ms) with high success probability (i.e., above $1\text{-}10^{-5}$) [ITU-R_M.2410].

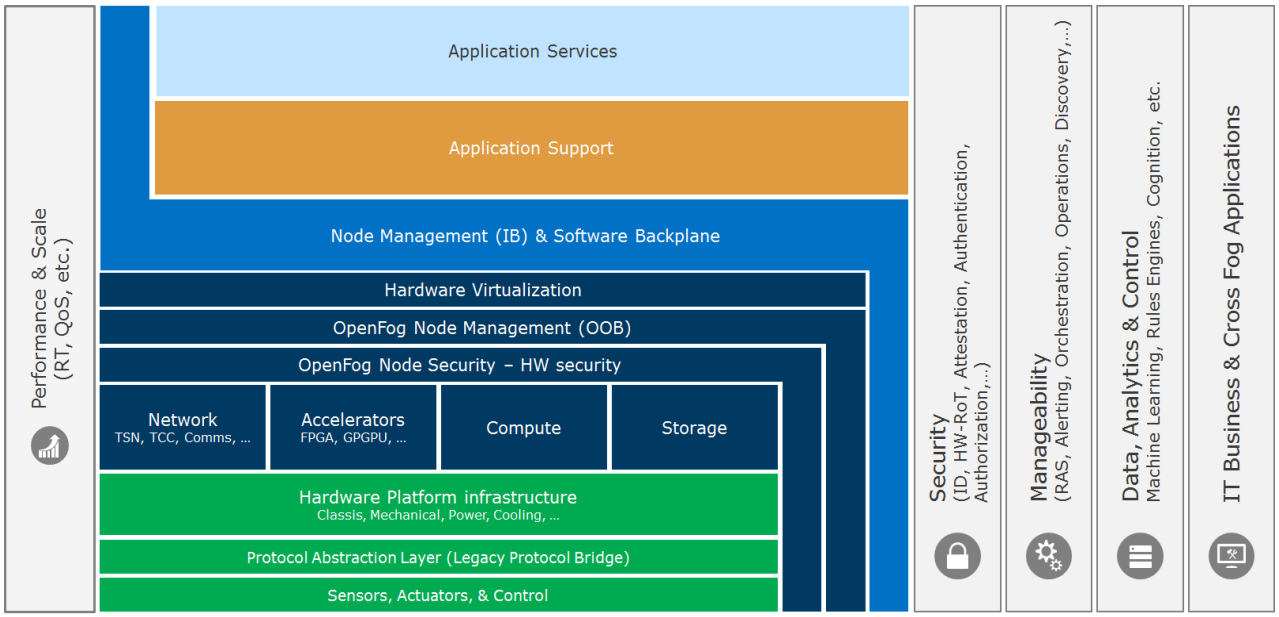

Figure 14: OpenFog RA high-level description (source *[OPENFOG_RAFC]*)

As illustrated in Figure 14, the OpenFog RA includes diverse perspectives:

- Performance - with concerns for low latency, time-critical computing, time-sensitive networking;
- Security - with concerns for end-to-end security, including data integrity mechanisms to address intentional and unintentional corruption;
- Manageability - regarding development activities like DevOps;
- Data Analytics and Control - Data needs to be localized in each Fog node to promote autonomous nodes and include mechanisms for analytics and control; and
- IT Business & Cross Fog Applications - includes the business logic and applications.

The reference architecture puts some emphasis on the scalability aspect since orchestration and allocation of resources need to be managed in the multiple levels of the Fog architecture. For instance, depending on the scenario, the architecture can include Fog nodes (with computing, storage, networking capabilities) located in the Cloud, as well as close to the user, where sensors, actuators are deployed. The OpenFog RA has been used as inspiration for more refined and niche solutions that try to couple a set of particular issues, such as, how to

combine centralized and distributed approaches for the management and orchestration for the Cloud-to-object continuum. In the following subsections, this approach is discussed.

3.2.2.2 Centralized vs. Distributed Approaches

Different approaches can be followed for resource management. A centralized approach is usually denoted *orchestration*, where a single centralized executable process coordinates the interactions among applications and services, managing the resources of the communication infrastructure *[JIANG_ORCH]*. This allows having a unified vision to apply general optimizations to the entire system. However, the response time for managing tasks is increased when using this approach, besides generating the typical problem of centralized solutions, a single point of failure.

On the other hand, a distributed approach is known as *choreography*, which is enabled by the exchange of information, cooperation, and agreements between two or more endpoints *[BITTENCOURT_CHOREO]*, increasing the complexity of the coordination tasks. Common solutions are based on a centralized approach for orchestration with more recent efforts towards a choreography-oriented design. Some proposals even explore the use of a hybrid approach [*VELASQUEZ_SORTS*], using orchestration in the upper layers (Cloud) to provide a global vision of the system allowing to apply optimizations to the entire ecosystem, while using choreography in the lower layers to provide dynamism and reduce response time.

Particularly, the OpenFog Reference architecture *[OPENFOG_RAFC]* proposes a distributed approach for resource allocation and orchestration to avoid relying on a central entity (single point of failure) and to decrease latency of services. The OpenFog reference architecture aims at autonomous orchestration and management where fog nodes located at the edge are also able to perform actions like the instantiation of services, routing data flow, according to policies and programmability.

3.2.2.3 Cloud/Fog Orchestrator Architectures

The Cloud-to-Object continuum requires a well-constructed Orchestrator able to deal with the entire management function for this complex environment, and properly handle all the challenges present in this landscape.

The OpenFog RA provides guidelines for the features a proper Cloud/Fog ecosystem should offer; however, it does not include instructions about the management or orchestration of the scenario and the actors playing key roles in it (e.g., devices, services, and applications). Particularly, the Cloud/Fog continuum can be supported by distributed or hybrid approaches for resource allocation and orchestration enabling synchronized management and orchestration of resources at Cloud and Fog levels. To better understand how to tackle the

orchestration challenges in the Cloud/Fog ecosystem, a subset of the research focused on empowering an end-to-end orchestration following centralized, distributed, or hybrid approaches are outlined below.

3.2.2.3.1 SORTS

A hybrid approach for service orchestration in the Fog is used in the Supporting the Orchestration of Resilient and Trustworthy Fog Services (SORTS) project *[VELASQUEZ_SORTS]*. The infrastructure is divided into levels that are managed using both choreography and orchestration, according to the needs of the different levels. The architecture allows the use of different Orchestrator instances, corresponding to the optimization goals of various scenarios.

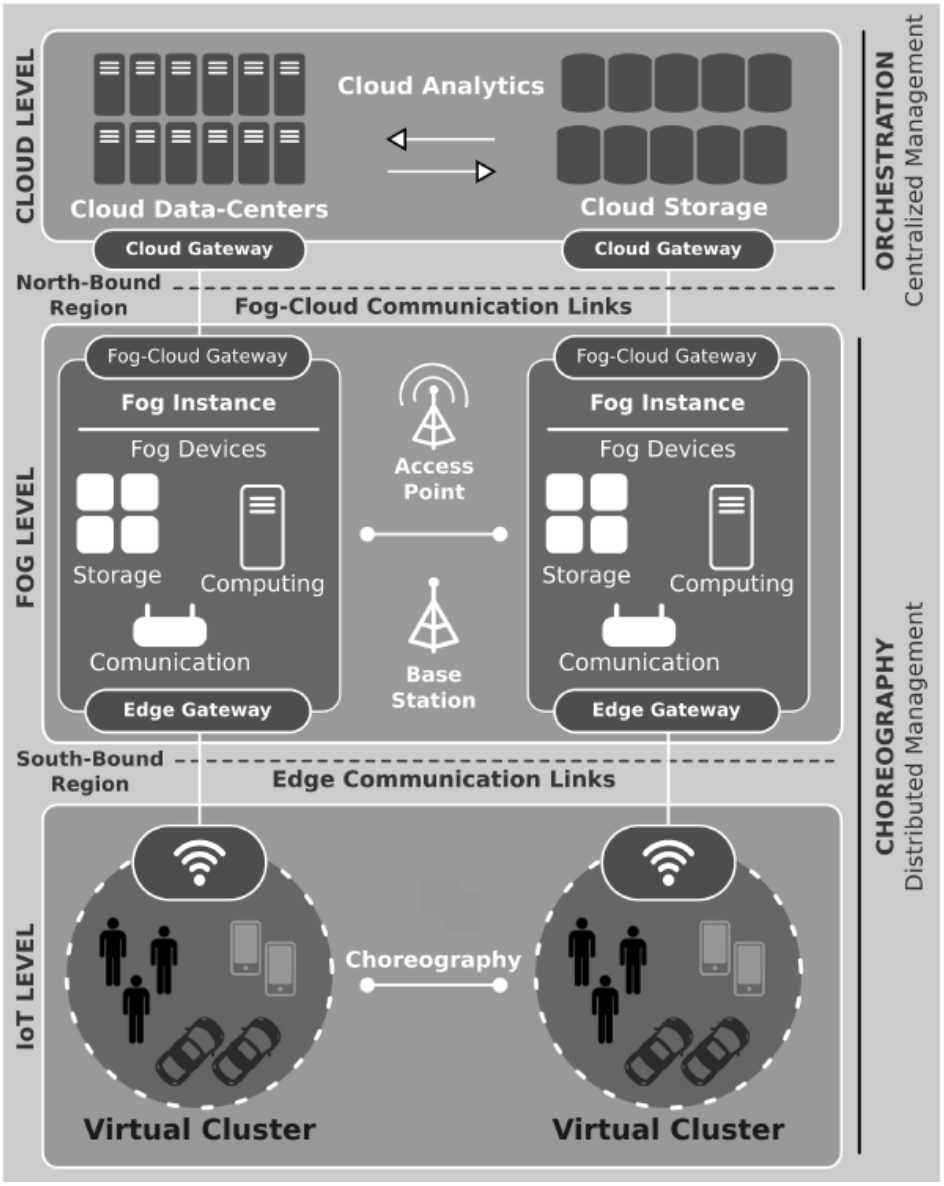

Figure 15: SORTS Logical Infrastructure (source *[VELASQUEZ_SORTS]*)

Figure 15 shows the logical infrastructure, which is divided into three levels: (1) the IoT, (2) the Fog, and (3) the Cloud. At the bottom levels, a choreography approach is used, meaning that the devices cooperate to manage purposes. This allows quicker response times in case of changes in the topology (e.g., shift from one Virtual Cluster to the next) thus increasing the resilience of the system.

Orchestration is used at the Cloud level (top level of the infrastructure). The hybrid approach (choreography plus orchestration) facilitates the achievement of a higher independence for the lower levels, granting them more dynamism and quicker response time in case of failures; at the same time, at the upper levels permits maintaining a global view that allows the implementation of optimization tasks involving the overall system.

The architecture presented in Figure 16 was designed to manage the resources and communication in the scenario previously described. Overlapped instances of the architecture are to be replicated at different Fog Instances and Virtual Clusters allowing the use of the distributed choreography mechanisms; and also at the Cloud level, where a single instance is deployed for global Orchestration.

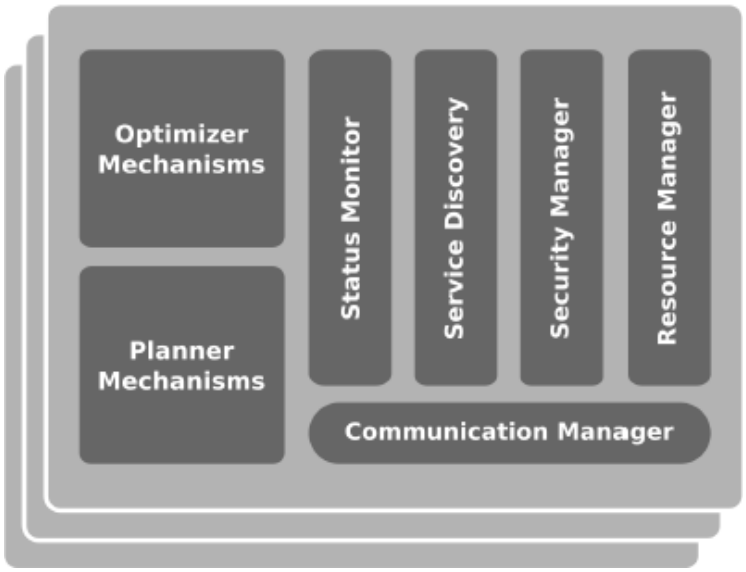

Figure 16: SORTS Orchestrator components (source *[VELASQUEZ_SORTS]*)

The Orchestrator allows the communication of different instances (*Communication Manager*), the monitoring of resource usage (*Resource Manager*), the lookup of available services (*Service Discovery*), and different authentication and privacy mechanisms (*Security Manager*). It is also possible to oversee the activities of the system (*Status Monitor*), schedule processes (*Planner Mechanisms*), and improve the performance of the system by applying *optimization mechanisms* at the upper levels.

3.2.2.3.2 SOAFI

Another Orchestration architecture reference comes from Service Orchestrator Architecture for Fog-enable Infrastructure (SOAFI). In the architecture, depicted in Figure 17, two main elements, organized on a client/server model can be identified as the *Orchestrator* and the *Fog Orchestration Agents.*

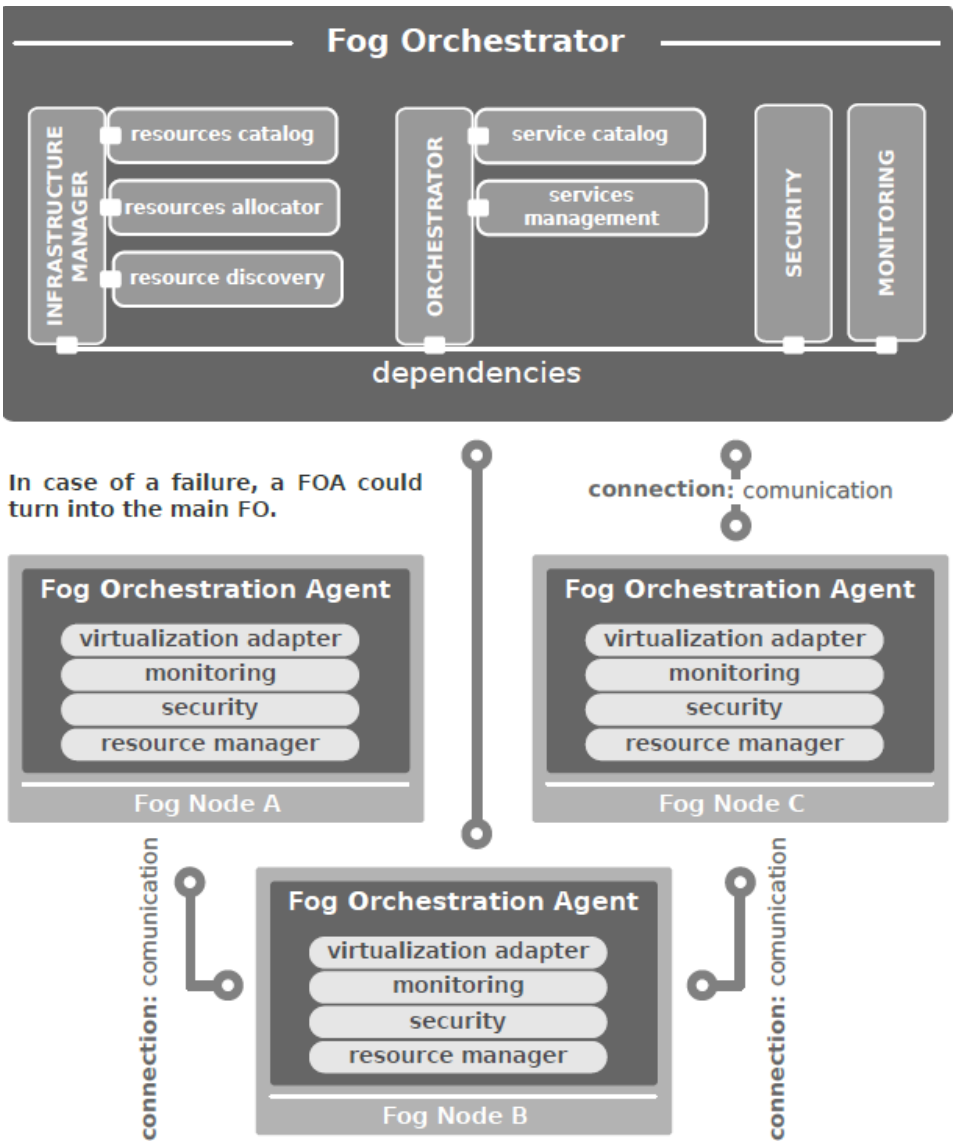

Figure 17: Service Orchestrator Architecture - SOAFI (source *[BRITO_SOAFI]*)

The Fog Orchestrator is a centralized entity that organizes Fog nodes into *Logical Infrastructures*, allowing the use of a hierarchy of objectives and capacities within the framework. The Orchestrator is in charge of the infrastructure management, orchestration, security, and monitoring.

The Fog Orchestration Agents are distributed in clusters in the Fog layer (usually known as Cloudlets), and have the responsibility of allocation within the cluster, discovery of resources, local optimization, and interoperability through standardized communication (e.g., using M2M communication protocols).

3.2.2.3.3 ETSI IGS MEC

The ETSI Industry Specification Group (ISG) has outlined an architecture to enable Mobile Edge Computing (MEC) *[ETSIMEC_FRA]*, as outlined in Figure 18.

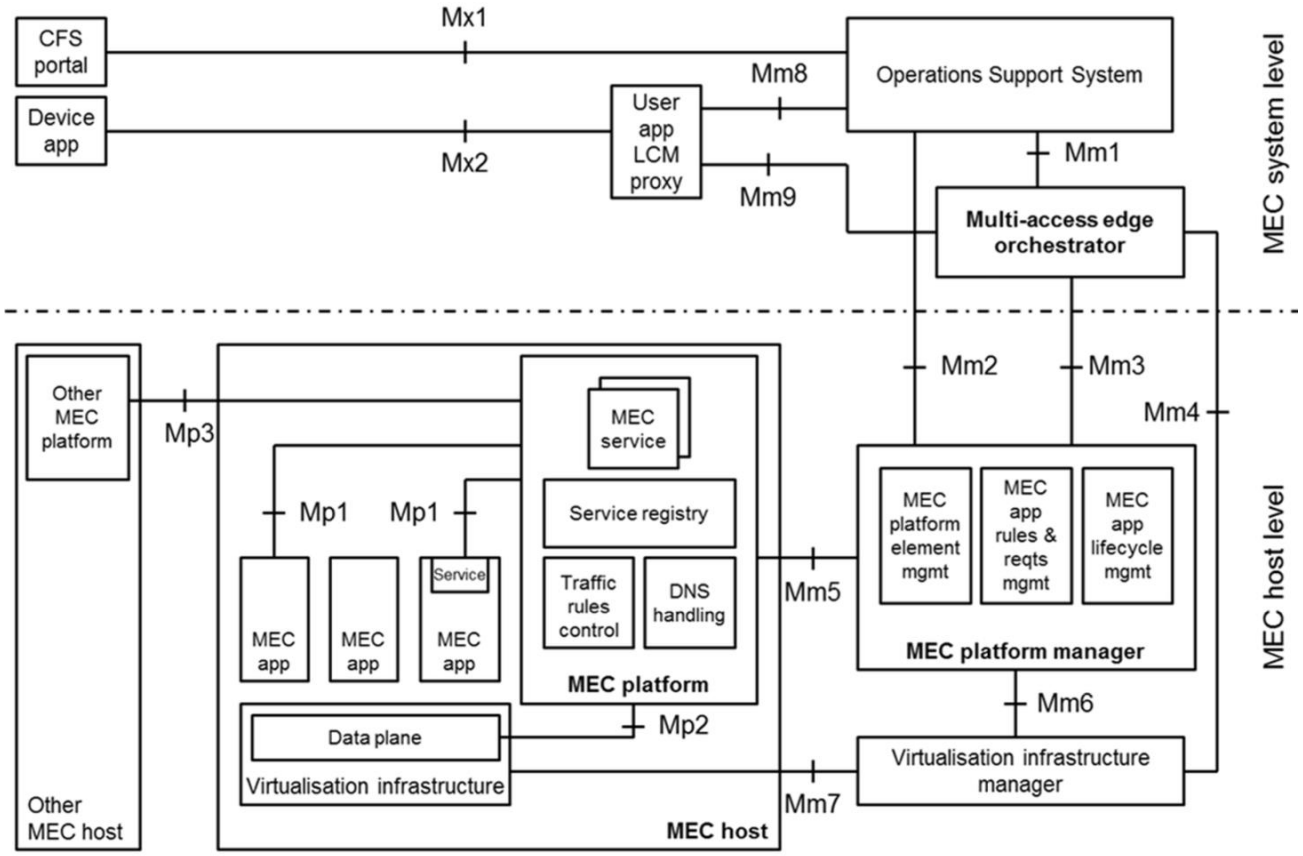

Figure 18: ETSI MEC Reference Architecture (source *[ETSIMEC_FRA]*)

The MEC reference architecture includes reference points between the diverse entities considered, such reference points are considered in three groups:

- Mp - reference points regarding MEC platform functionality, for instance, between MEC applications and MEC services;
- Mm - reference points regarding Management, for instance, between the MEC host and the platform manager or edge orchestrator; and

- Mx - reference points regarding connection with external entities, such as applications running on the edge device.

The entities considered in the MEC architecture are:

- MEC host - an entity that contains a MEC platform and virtualization infrastructure to provide computing, storage and network resources to run MEC applications, considered in the MEC host level;
- MEC platform - set of essential functionalities required to run MEC applications or to run services, considered in the MEC host level;
- MEC applications - applications that implement the desired functionalities, considered in the MEC host level; and
- Multi-access edge orchestrator - a core component of the MEC system, considered in the MEC system level.

The virtualization infrastructure in the MEC host includes a data plane responsible for executing traffic rules, route traffic between applications, services, DNS server(s) and between intra/inter networks. The MEC platform assures a set of functionalities that include the management of an environment to allow MEC applications to discover services, to manage traffic rules, configuration for the data plane of the virtualization infrastructure of the host, to provide access to persistent storage, among other functionalities. MEC applications are assumed to run as Virtual Machines (VMs), which are able to consume and provide MEC services.

From a MEC system-level management perspective the Multi-access edge orchestrator holds multiple functionalities like maintaining an overall view of deployed MEC hosts, resources and services available. In addition, the orchestrator is also responsible for assuring that MEC applications comply with policies (e.g., packages are authentic). In addition, the orchestrator also selects the most appropriate MEC host to instantiate an application considering latency, availability of resources, and available services. The MEC architecture also has connections with NFV.

3.2.2.3.4 CONCERT

A Cloud-based architecture is proposed by Liu et al. *[LIU_CONCERT]*. In CONCERT, the control and data planes are decoupled, and the services are deployed at the edge of the network, closer to end-users. The data plane deals with different physical resources, which are orchestrated at the control plane. The architecture is depicted in Figure 19.

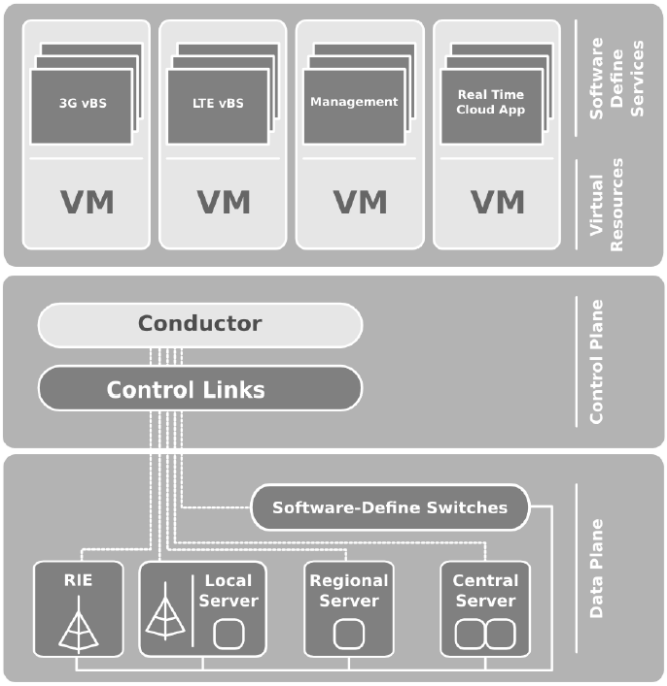

Figure 19: CONCERT Architecture (source *[LIU_CONCERT]*)

At the bottom of Figure 19, there is the *Data Plane*, which encompasses the physical resources which are managed in an upper decoupled control plane by the *Conductor Entity*. The conductor is in charge of the orchestration and virtualization of the data plane resources. Virtualized resources are defined in the upper layer. *Radio Interfacing Equipments* (RIE), SDN switches, and computational resources are included in the data plane.

The conductor is the main component of the *Control Plane*, and its role is to orchestrate and export physical resources to the upper layers, performing a variety of tasks via different mechanisms, like Location-Aware Computing Management (LCM), which schedules computational tasks to computational resources. The conductor must collect data about deadlines, resource demands, location, and result destination of the applications and services, to feed the proper mechanisms and algorithms with the needed information to successfully complete their tasks.

At the top-level services are deployed following a software-defined approach, using virtualization and orchestration performed by the Conductor in the control plane.

3.2.2.4 Summary

Table 4 provides a summary of the resource allocation solutions.

Table 4: Resource allocation solutions

Approach	Advantages (++)	Disadvantages (--)
SORTS	• Use of hybrid approach • Architecture details for the orchestrator • Multiple instances allow to tailor the orchestrator according to different needs	• Does not cover interoperability support for heterogeneous devices
SOAFI	• Use of Hybrid approach • Architecture details provided for the orchestrator and the agents	• Does not provide optimization guidelines to deal with latency or resilience • Does not cover authentication aspects • Does not cover interoperability support for heterogeneous devices
ETSI IGS MEC	• Support of a well-known standardization organization • Division of data and control planes • Supports connections with NFV	• Centralized approach • Does not provide optimization guidelines to deal with latency or resilience • Does not support security and privacy aspects
CONCERT	• Division of data and control planes	• Centralized approach • Lack of details for the orchestrator • Does not cover security or authentication aspects

3.2.3 Optimization approaches

Optimization plays a crucial role in several fields and engineering is not an exception. Its purpose is to find the best solution for the established objective. Figure 20 illustrates how the optimization approaches can be employed in the Orchestration/Virtualization platforms for optimized resource management, considering multiple data sources (e.g., monitoring solutions), the policies for services and resource management. The optimization approaches can rely on distinct mechanisms, as described in this sub-section. Section 3.2.4 exemplifies the application of the optimization approaches in scenarios with similar requirements as those of OREOS.

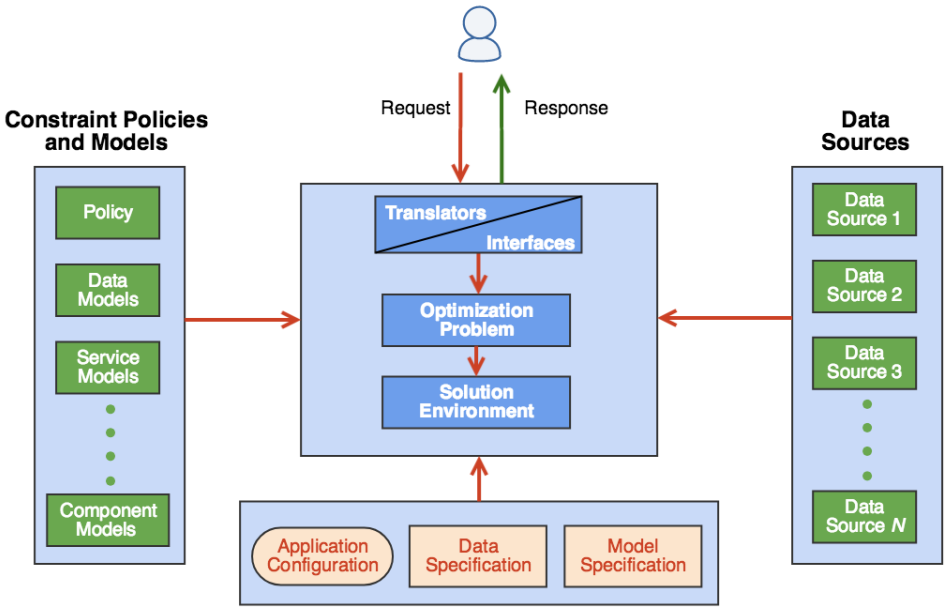

Figure 20: Example of optimization approaches integration, considering policies and multiple data sources (source [*KAPADIA_ORC*])

The optimization approach is divided into two main phases: the design model and the optimization mechanism. The first one is devoted to the optimization model design. Here it is identified the decision variables that define the objective function and the relations between them to get solutions that meet the requirements of the problem. The second phase is devoted to the optimization mechanisms to deal with the model to obtain the optimal solution.

Usually, the optimization problems that appear in real-life situations are hard to solve. Thus, it is usually to divide it into several sub-problems to be manageable and solved following a divide-and-conquer strategy. In this aspect the optimization approach can include an additional phase to support the divide-and-conquer strategy. The graph partition approach uses a network structure to describe the problem and is an example of it. For example, using Graph Partitions, instead of trying to find a global optimal solution it is possible to focus on finding a local optimal in each partition and in later phases try to figure out how to combine them.

3.2.3.1 Design Models

The design of a good model is the key to the success of the optimization approach. To this end, it is necessary to identify the decision variables (values we want to control in the optimization process) and parameters (values previously fixed in the definition of the model) with which it is possible to define the objective function and establish the constraints that characterize all the admissible solutions of the problem.

However, the more detailed the model is, the greater the number of variables and constraints it includes. Thus, the problem becomes more challenging to solve and may even be computationally demanding. In general, we have to establish a trade-off between these two aspects. Thus, we initially consider a more simplified model of the real situation, with the essential characteristics only. If the solution to this problem does not satisfy all the requirements that have been neglected, the model is redefined to overcome these shortcomings. In this way, the model is improved until it reaches a satisfactory solution for the real application in question. One aspect to take into account when defining the model is the number and type of variables (continuous or discrete) as well as the number of constraints since they have a significant impact on the time required to solve the problem.

In problems with continuous variables, algorithms that use some information about the derivatives of the functions (if it exists) can be used, as is the case of line search methods (maximum descent, Newton, Quasi-Newton) and trust-region methods. If that information is not available, methods without derivatives can be used, as is the case of the direct search method or the Nelder-Mead method. This kind of methods usually guarantees convergence to a stationary point that can be only a local minimum of the objective function. Thus, the study of convexity (objective function and restrictions) has an essential role in the characterization of the solution found. In this case, it could be possible to guarantee that it corresponds to the global minimum of the problem.

The problems with discrete variables have a combinatorial nature, so the number of admissible solutions (in general, finite) explodes quickly with the size of the problem. Thus, it is not possible to evaluate all solutions to find the best one. Several algorithms are dealing efficiently with some of these problems, such as Branch-and-Bound methods and dynamic programming.

3.2.3.1.1 Single and Multiple Objective

Problems in the engineering field that frequently arise in real applications are multi-objective in nature since several criteria need to be optimized simultaneously. This approach makes it possible to determine a set of solutions that form the so-called Pareto front, that is, solutions for which it is impossible to improve one of the criteria without worsening one of the others.

In general, the objectives under study are conflicting. Consequently, the Pareto front has a large set of solutions allowing the decision-maker to choose the most appropriate for the application in question. However, the number of existing Pareto optimal solutions can be so large that it makes the complete computation of the

Pareto front too expensive. One approach to deal with this situation is to join all criteria into a single objective function (called utility function), transforming it into a single-objective problem and, therefore, easier to solve. A utility function frequently used in the literature is the weighted sum of the criteria (also known as "multi-attribute utility theory"). It is possible to show that an optimal solution to this problem is always a Pareto optimal solution. However, Pareto solutions may not correspond to the optimal solution of the mono-objective problem regardless of the weights used in the utility function.

3.2.3.1.2 Linear Programming

As mentioned before, designing the model for a given real application should start with a simple one. In this context, linear problems obviously stand out. In problems associated with telecommunications, often emerges network structures. Thus, problems such as the maximum flow, the flow (with a given value) with minimum cost, the minimum spanning tree, location problems like the vertex p-center and the p-median, the transportation problem and the shortest path problem are good initial candidates. In these cases, there are efficient algorithms to deal with them.

Additionally, the model can be defined as a general integer linear program or a mixed-integer linear program, and Branch-and-Bound is usually an efficient method to solve it. If integer variables are not considered, the Simplex method and/or the interior method can be used to solve it.

3.2.3.2 Optimization Mechanisms

Once the model is described, we must choose or propose an algorithm to solve it to obtain a satisfactory solution to the problem at hand. This solution may be the best among all admissible solutions, and, in this case, we need an exact algorithm to get it. These kinds of algorithms tend to converge to local minimizers, being necessary to study the properties of the problem (for example, check convexity) to ensure that this solution is global.

On the other hand, it is possible to stop in advance the search for the exact solution obtaining only a good approximation to it. The advantage of this procedure is the decrease in computational effort without too much loss in the quality of that solution. Other mechanisms, such as heuristics, allow obtaining good solutions involving little computational effort and will also be considered in this project.

3.2.3.2.1 Exact Methods

Exact algorithms seek to determine the best solution among all possible admissible solutions. Obviously, this search cannot be done as a brute force task since there are infinite solutions in continuous problems, while in discrete ones, the number of solutions is finite but very high.

In continuous problems, the study of properties such as differentiability and convexity allows one to choose search directions, such as the steepest descent direction or Newton's direction, which generate a sequence of approximations converging to an optimal solution of the problem. These methods can be combined with quadratic penalty methods, quadratic sequential programming method, augmented Lagrangian method, barrier method, interior points method. To guarantee the solution satisfies all the constraints of the model. However, in real problems, the objective function is often obtained by simulation or is affected by some noise. Then, it is not possible to obtain an explicit expression for the function. In this case, it is impossible to obtain the derivatives, and their approximations are of low quality due to noise.

In discrete problems, it is possible to reduce the search area when they verify the optimality principle, which indicates that every optimal solution is formed by optimal sub-solutions. Thus, if we find a sub-solution that is not optimal during the search, the algorithm does not proceed in that direction. An example of an algorithm based on this principle is dynamic programming. If the problem does not check the optimality principle, the Branch-and-Bound algorithm can be used, for example.
Given the complexity of models in real situations, determining the exact solution is often a task that consumes great computational effort, so these algorithms have a strong limitation in the size of the problem they can solve.

3.2.3.2.2 Heuristics Methods

Heuristics are algorithms that allow determining a solution of reasonable quality with little computational effort. In general, they can be divided into two large groups: constructive and exploratory heuristics. In the first one, it is intended to build only one solution, and the heuristic defines rules for assigning the value of each variable. In the second, the heuristic starts from a set of solutions and tries to improve it at each iteration, returning at the end the best solution obtained.

Constructive heuristics are usually made in a tailor-made algorithmic environment, based on some properties of the problem in question. Thus, they can take advantage of these characteristics, but it becomes challenging to generalize to other situations. Another way to define heuristics of this type is to imitate the pattern of a good

solution obtained by some other method. For example, in the first step, optimal solutions are obtained in small problems and then try to imitate that behaviour in larger ones.

The exploratory heuristics are more general and, therefore, easier to apply. They start from a set of initial solutions (which can be random) and improve on it. They can improve by replicating some existing behaviour in nature, as is the case of evolutionary algorithms (such as genetic algorithms, simulated annealing, ant colony, etc.). Other techniques included in this group are local search algorithms (which explore the neighbourhood of a particular solution) and math-heuristics, where exact algorithms can solve part of the problem or a relaxation. Based on this solution, the algorithm built a good answer to the original problem. Finally, exact algorithms that end before reaching the optimality stop conditions can also be included in this group.

3.2.3.3 Graph Partition

Graph theory is frequently used to tackle a range of engineering and computer science applications. In particular, a graph can be used to represent discrete objects and the relationship among them, such as a communication network infrastructure. The communication nodes can be viewed as the vertices of the graph while the communication links are the edges connecting the vertices. Graph partition can be used to divide the topology into clusters or communities for different purposes.

There are several methods to partition graphs including Girvan and Newman *[NEWMAN_GP]*, which progressively removes edges from the original graph to identify the communities on it. The edges removed are those with the highest betweenness centrality at each step, thus tightly knit communities are created as a result. Another method is the Fluid Communities method *[PARES_GP]*, that creates communities by mimicking the behaviour of several fluids, expanding and pushing one another in the graph until an equilibrium is found, generating communities balanced in size. The PageRank mechanism *[PAGE_GP]* can also be used to create communities, which ranks the nodes via probability propagation according to a metric (e.g., propagation delay). The resulting communities group nodes by a given characteristic. Graph partition mechanisms have proven to be useful to deal with load balancing by building communities in the network topology to share the load among the nodes inside each community.

3.2.3.4 Machine Learning

Over the last decade Machine Learning (ML) became ubiquitous in many areas of research, fuelled by the large quantities of information gathered by researchers, practitioners and industries, and by the growth in the processing capabilities of computer systems.

Without noticing, we interact with ML approaches daily that give us access to recommendation systems; translation and speech recognition tools; email categorisation, spam and malware detection; abnormal behaviour detection (e.g., fraudulent transactions, or video surveillance); credit risk assessment; plagiarism detection; customer behaviour analysis and segmentation; and autonomous driving *[KRIZHEVSKY_ML, FARABET_ML, ESTEVA_ML, HOCHREITER_ML, LOPEZ_ML]*. In what concerns the application of ML to autonomous driving, the majority of the literature is focused on multi-modal object detection and semantic segmentation *[FENG_ML]*.

However, another important challenge where ML is playing an important role in autonomous driving is resource allocation in 5G networks. In fact, it has been shown that the most recent advances in Deep Learning have achieved remarkable results in resource allocation when compared with conventional methods *[LEE_ML, WANG_ML, NASIR_ML]*. Given the demands of vehicular networks, and the challenges in managing slicing in 5G networks, *[TAYYABA_ML]* proposes a ML model to optimise resource allocation taking into account the demands and dynamics of the network. The study evaluates different Deep Neural network architectures such as Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN), and shows that LSTMs are able to obtain impressive results. With the large quantities of data that are being generated, traffic prediction using ML is not only becoming a possibility, but a reality. In fact, LSTMs have attained remarkable results within this context. In *[WANG_ML_SPATIO]* the authors propose a model using two ML learning models to capture the spatio-temporal characteristics of network traffic. The spatial dependence is represented using an AutoEncoder, and the temporal dependence is modelled using LSTMs.

Reinforcement Learning is a ML approach that has been used with success in network slicing *[LI_ML_SLICE]* and multitenant cross-slice resource orchestration *[CHEN_ML]*. Another successful application of the ML in resource allocation is presented in *[LI_ML]*. In this study, the authors present an approach using a Reinforcement Learning algorithm combined with a LSTM. In concrete the proposal uses the Actor-Critic Deep Reinforcement Learning Algorithm, where the *state* is the service demands, and the *actions* are the allocated resources. Since user mobility hinders the ability of the algorithm to perceive the environment, the authors include a LSTM model to track and predict user mobility.

As the works described above show, ML is an asset when dealing with resource allocation in vehicular networks. However, there are some open issues and challenges that one has to deal with when using these types of models *[LIANG_ML]*. The first and foremost is related to how to select the most appropriate architecture and/or parameter settings for a Deep Learning model. We know that there are no free lunches, and as such no single

model configuration will work well in all the tasks. As an example, Convolution Neural Networks (CNNs) exhibit very good results in computer vision tasks, whilst LSTMs are more tailored to sequential data, such as time-series. Since resource allocation has a specific set of characteristics, it is important to take them into account when choosing/designing a ML model. Another issue is related to the bridging between the development and training of the models and its implementation in the real world. Usually, these models are designed and tested in a controlled environment based on simulations. However, building a simulator that is able to include all the characteristics of the real world is difficult, if not impossible. Finally, and when using Reinforcement Learning models, one has to take into account several issues, namely the nonstationarity and partial observability of the environment.

3.2.3.5 Summary

Table 5 summarizes the diverse optimization approaches.

Table 5: Optimization approaches summary

Approach	**Advantages (++)**	**Disadvantages (--)**
Design models	• Allow you to obtain a theoretical model to run simulations on • Allow to perform the sensitivity analysis to some parameters of the problem	• Simplified models for the real case • Mismatch between the performance of the solution in the theoretical model and the real case.
Optimization	• allow finding the solution with the best performance in the theoretical model • heuristics allow obtaining solutions close to optimality without great computational effort	• the most realistic theoretical models are difficult to solve, requiring a high computational effort which forces to consider only small size problems • do not determine the optimal solution and it is difficult to have an approximation of the optimal gap
Graph partition	• Can find a close-to-optimal solution • Successfully used for load balancing • Lower execution times and resource consumption to reach a solution than optimizers	• Does not reach an optimal solution • Some mechanisms lead to unbalanced communities/partition impacting the fairness of the solution
Machine Learning	• For specific tasks there are techniques that achieve very good results (e.g., computer vision with CNNs)	• Hard to validate models in real world including all the characteristics • When using Reinforcement Learning models, there are several issues, like the nonstationarity and partial observability of the environment

3.2.4 Decision approaches

The allocation of resources is an NP-hard problem commonly tackled by the different techniques described in the previous section. However, it is important to define the metric (or set of metrics) that will help guide the decision-making process for this problem. Some of the decision factors frequently used for the resource allocation and orchestration of the Object-to-Cloud continuum are discussed below, including some examples of previous efforts by the research community in these directions.

3.2.4.1 Fairness aspects

The Cloud/Fog Orchestrator must be able to provide access to the resources of the communication and processing infrastructure for different users with a sense of *fairness*, this means that each user has a fair share of the resources in the infrastructure and try to evenly distribute these resources among users.

Godinho et al. *[GODINHO_FAIR]* present a MILP formulation to deal with energy consumption and latency issues with two objectives, maximize Fog usage and evenly distribute the services throughout the system. Additionally, they propose a heuristic using a master/slave approach in Fog nodes clusters. The services were evenly distributed according to their energy consumption throughout the system. Zhao et al. *[ZHAO_FAIR]* address the problem of multi-resource fair allocation in Cloud computing. Their heuristic classifies users into different queues according to their dominant resource requirement, and the goal is to ensure that users from the same queue are treated equally, maximizing resource utilization subject to fairness properties.

Feng et al. *[FENG_FAIR]* include min-max fairness guarantees in their proposal for resource allocation in the Fog aimed at optimizing energy efficiency. They consider the energy consumption of individual links and study the min-max energy efficiency-optimization. Their solution is based on a mathematical model using Lagrangian dual decomposition. Du et al. *[DU_FAIR]* propose a solution for offloading and allocation of tasks in the Cloud/Fog. Their heuristic combines different optimization goals including computational resources, transmit power, and radio bandwidth while guaranteeing user fairness and maximum tolerated delay. Results showed that it was possible to reduce the delay while benefiting mobile users by providing them a fair treatment.

3.2.4.2 Resource usage aspects

Orchestration in the Fog, particularly placement policies using Fog nodes, can benefit the network resource usage in the communication network since there is no need for additional traffic traversing the entire network infrastructure up to the Cloud and back to the endpoints. However, Fog nodes are notoriously more resource

constrained (processing, storage) in comparison with the Cloud, thus a solution for resource allocation in the Fog has to carefully analyse the resource usage.

Skarlat et al. *[SKARLAT_RU]* analyse the placement of IoT services in the Fog according to their QoS requirements. They use an ILP model aimed at maximizing the utilization of the Fog resources, while taking in consideration feasibility constraints. Taneja and Davy *[TANEJA_RU]* describe a heuristic based on a module mapping algorithm to optimize the resource usage in the Cloud to Fog continuum. The solution is compared with a Cloud-only approach, being able to not only optimize resource usage but also reducing the latency. Brogi et al. *[BROGI_RU]* use multi-objective optimization for service placement, balancing a trade-off between QoS-assurance and Fog resource consumption. Their proposal is aimed at helping IT experts in the distribution of application services to Fog infrastructures. The resource consumption is calculated as an aggregated percentage of RAM and HDD over the Fog nodes. Pallewatta et al. *[PALLEWATTA_RU]* use a decentralized application placement policy aimed at using locations closer to the data sources in order to minimize latency and network usage. The use of the policy resulted in an 85% improvement of network resource usage when compared to Cloud-only placement, and around 40% improvement over an alternative Fog application placement method.

3.2.4.3 Quality of Service aspects

The applicability of the scenario described in section 2 is dependent of two major factors, since we are dealing with a URLLC type of service: low latency and high reliability. While the former is more dependent on transmission and resource allocation mechanisms for massively distributed heterogeneous wireless nodes *[VAQUERO_FOG]*, that later is more directly related to service availability and blockage errors, which are both significantly increased when applications and services are shifted towards the edge of the cloud. The service orchestrator plays a major role in guaranteeing that efficient mechanisms are in place to guarantee both the strict latency and reliability requirements.

Mahmud et al. *[MAHMUD_LAT]* use fuzzy logic to place applications in the Fog to maximize QoE and reduce deployment time. Simulations are used for assessment. Guerrero et al. *[GUERRERO_LAT]* propose to place popular services closer to users, using hop count as a metric for closeness. The devices cooperate among themselves in a decentralized heuristic. The evaluation is carried out using simulation. In another work, Guerrero et al. *[GUERRERO_LAT_B]* use genetic algorithms to optimize latency, service spread, and resource usage, using Python for simulations.

Shi et al. *[SHI_LAT]* also explore using genetic algorithms for service placement in the Fog to minimize latency. Their genetic solution optimizes load balancing in distributed Cloud/Fog networks. Furthermore, task reallocation is also considered. Velasquez et al. *[VELASQUEZ_LAT]* use the graph partition to create communities in the Fog nodes in order to deploy applications ranked by their popularity. They also propose an ILP model to find the optimal location in order to minimize latency. The ILP model was able to obtain lower latency levels but at the same time saturating the nodes at the edge, with the graph partition heuristic getting results close to the optimal.

3.2.4.4 Mobility support aspects

It is important to take into consideration the mobile nature of the devices at the edge of the network (e.g., sensors in cars), for which location awareness, dynamism, and geo-distribution support must also be included. Route actualization as well as migration of deployed applications and services must be considered.

Lu et al. *[LU_MOB]* propose a position-based routing scheme for inter-vehicle communication using VANETs, while also minimizing resource usage. The routing scheme is based on a greedy heuristic that also considers repair mode when reaching a maximum number of hops for the delivery of packets. Djemai et al. *[DJEMAI_MOB]* propose a service placement solution based on genetic algorithms aimed at supporting node mobility while ensuring infrastructure energy efficiency and QoS. The GA receives information about node mobility to reach a decision. The work is complemented by a greedy heuristic that calculates the most probable path for each mobile device and then computes the proper placement in order to reduce energy consumption.

Gonçalves et al. *[GONÇALVES_MOB]* use an ILP model for virtual machine placement and migration decisions based on mobility prediction. Two objective functions are used sequentially, one to maximize the acceptance and one to minimize latency. They base their proposal on predictable routes (e.g., buses and trains) for the mobility data.

In 5G scenarios aiming to support URLLC, the non-radio frequency approaches can be followed, considering the mobility predictions. For instance, the information of accelerometers can be employed to determine the mobility patterns *[JIONG_xURLLC]*, through ML models. Nonetheless, the integration of such information in 5G systems is not easy, in particular when considering that the 5G network can have multiple slices.

3.2.4.5 Resilience aspects

The availability of physical and logical devices in scenarios that support critical applications such as traffic control, augmented maps and eHealth becomes a key issue to handle. Different techniques can be applied to enhance the resilience of the services in the Cloud/Fog continuum, like using replication and backup schemes. For instance, using a primary and backup model, where devices and/or services are duplicated for robustness purposes. From the communication point of view, another option is using disjoint paths in order to obtain backup paths that can be activated in case of failures. Migration can also be used in case of disrupted communication.

Lera et al. *[LERA_RES]* propose a service placement policy to enhance the availability and QoS of applications using graph partitions. Simulation results showed that around 70% of the users obtained better response times in cases of failures, in comparison with another approach using ILP. Aidi et al. propose the use of replicas to increase the survivability of Services *[AIDI_RES]*. Two heuristics are proposed for complex scenarios, with the first aimed at determining the maximum number of services to replicate, and the second to allow the spreading of services into physical nodes.

Rehman et al. *[REHMAN_RES]* present a survey on fault tolerance using SDN. Their work aims at identifying fault tolerance requirements in SDN environments and propose strategies to tackle them. Malik et al. *[MALIK_RES]* used divided data and control planes in their proposal of an approach for fault management at the data plane with the goal of eliminating the convergence time required to allocate new network paths to forward the traffic after a failure. A time window is used for the update of network paths. Beheshti and Zang *[BEHESHTI_RES]* introduce a group of heuristics to improve the resilience of the connection between control and data planes in SDN, based on resilience-aware controller placement and traffic control routing in the network.

3.2.4.6 Summary

The different decision factors presented in this section and summarized in Table 6 sometimes can go in opposite directions, i.e., by optimizing one factor the other gets highly affected. For instance, by optimizing resilience by using replicas, there is an increment in the resource usage, which gets affected. For these cases, it is important to find a trade-off among competing decision factors. On the other hand, some works consider multiple optimization goals, prioritizing them according to some criteria. For instance, aiming at providing mobility support as the main goal but also targeting minimization of resource usage.

Table 6: Decision approaches solutions

Approach	Characteristics of Evaluated Works
Fairness	● Commonly use min-max objectives ● Frequently studied in combination with other optimization goal, such as resource consumption ● Use of different techniques such as MILP, mathematical models, and heuristics ● Majority of use of simulations for evaluation
Resource usage	● Commonly used in combination with another optimization goal, like latency ● Use of different techniques like ILP, multi-objective optimization, and heuristics ● Lack of code available or information that allows to replicate results
QoS/Latency	● Different metrics related to latency, such as network delay or processing delay ● Use of different techniques like fuzzy logic, evolutionary and genetic algorithms, graph partition, and mathematical models ● Majority of use of simulation for the evaluation
Mobility support	● Consider approaches towards routing and towards migration ● Use of different techniques like ILP and genetic algorithms ● Lack of code available or information that allows to replicate results ● Majority of the works used simulation for evaluation
Resilience	● Use of different strategies, commonly using replication of services or paths ● Could affect other decision factors, such as resource usage, by using replicas of services, nodes, and paths ● Use of different techniques such as ILP, heuristics, and graph partition ● Some proposals on routing rely on alternative technologies such as SDN

3.3 Orchestration Platforms

This section presents existing solutions to orchestrate computational resources in diverse computing models.

3.3.1 Introduction

Orchestration platforms support NFV Management and Orchestration. For NFV Management virtualization layers are proposed to support different physical infrastructures. In addition, approaches like SDN are relevant to allow the management of network resources in agnostic fashion regarding the underlying physical technology (e.g., wireless, optical networks, etc). ETSI has been specifying *[ETSI_NFV]* several standards to enhance interoperability between orchestration platforms, to enable the multi-tenancy support in 5G networks, to support the Service Based Architecture (SBA), to enable MEC paradigm *[ETSIMEC_FRA]*, among others key functionalities related with NFV. ONAP *[KAPADIA_ORC]* is one of the most complete platforms to design, create

and manage the lifecycle of network services, while Open Source MANO (OSM) *[ETSI_OSM]* has been employed in diverse projects targeting 5G networks and slicing.

3.3.1.1 ONAP

ONAP is a platform for real-time, policy-driven orchestration and automation of physical and virtual network functions that will support complete lifecycle management and includes a comprehensive design framework to create network services and related items and a runtime framework (which includes the ETSI MANO functionality of NFV Orchestrator (NFVO) and VNF Manager (VNFM)) *[KAPADIA_ORC]*. ONAP primarily interacts with these software systems:

- Operational support systems (OSS): include OAM (operations, administration and management) and FCAPS (fault, configuration, accounting, performance, security).
- Business support systems (BSS): include service provisioning, deployment, service change management, customer information management, service-level agreement (SLA) management, billing and customer support.
- Big data applications: analytics applications to gain business insights and intelligence from data lakes containing network data collected by ONAP.

ONAP provides a comprehensive design framework to create network services and a runtime framework. The runtime framework includes the ETSI MANO functionality of NFVO and VNFM and goes beyond by adding monitoring and service assurance.

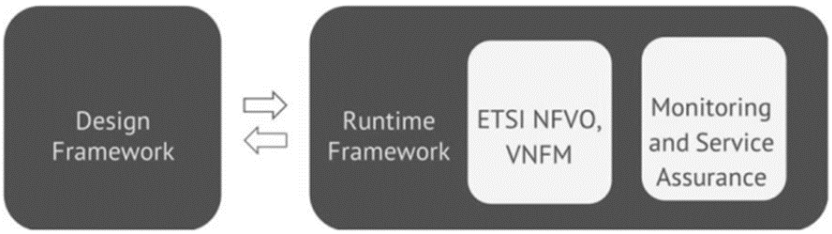

Figure 21: ONAP scope

As per illustrated in Figure 21, ONAP internal high-level architecture defines two major systems — design time and run time — which allows for clean delineation between design and operational roles. The design time environment module, whose tool is called the Service Design and Creation (SDC), is responsible for:

- VNF on-boarding/validation
    - Create license model
    - Onboard VNF

    - Test VNF
    - Store VNF in catalogue
- Network service/SDN service design
    - Create service
    - Test service
    - Store service in catalogue
    - Distribute service
- Create recipes
    - Policy creation
    - Workflow design
    - Closed control-loop design.
- Onboard functions
    - Analytics applications
    - Collectors
    - Microservices
    - Datastores

The runtime environment module is responsible for:

- Service orchestration and lifecycle management
    - Service Orchestration (SO)
    - SDN controller (SDN-C)
    - Application Controller (APP-C) and Virtual Function Controller (VF-C)
    - Infrastructure controller (interface to the VIM).
- Monitoring and service assurance
    - Data collection, analytics, and events (DCAE)
    - Storage of all active and available inventory (A&AI).

The design-time framework brings forth a comprehensive development environment with tools, techniques, and repositories for defining and describing resources, services and products, including, policy design and implementation and an SDK with tools for VNF supplier packaging and validation.

On the other hand, the run-time environment executes the rules and policies distributed by the design and creation environment, as well as the controllers that manage physical and virtual networks.

The most important ONAP components for this project are:

- The Service Orchestrator (SO) component: executes the specified processes by automating sequences of activities, tasks, rules and policies needed for on-demand creation, modification or removal of network, application or infrastructure services and resources, this includes VNFs, CNFs and PNFs. It is used for real-time service creation or lifecycle management by primarily interacting with controllers, SDC and A&AI. SO gets topologies, design time configurations and workflow models via SDC. The SO is invoked through events, APIs from BSS or manually through VID (Virtual Infrastructure Deployment)[13]. SO makes homing (where to place the workload) decisions by interacting with OOF (ONAP Optimization Framework), resource determination and triggers management actions via one of the four controllers at its disposal, or by directly interfacing with OpenStack. SO also handles errors and rollbacks.

- The OOF: provides a policy-driven and model-driven framework for creating optimization applications for a broad range of use cases. OOF Homing and Allocation Service (HAS) is a policy driven workload optimization service that enables optimized placement of services across multiple sites and multiple clouds, based on a wide variety of policy constraints including capacity, location, platform capabilities, and other service specific constraints.

- The runtime Data Collection, Analytics, and Events (DCAE) module: is used for closed control-loop automation, trending, solving chronic problems, capacity planning, service assurance, reporting, and so on. DCAE orchestrates data collectors, microservices, analytic applications, and closed control-loops. Given the model and SDK-driven nature of DCAE, it offers a self-service interface to developers, designers, and operators, interacting with SDC, Policy, and A&AI.

- Policy: the runtime policy software module guides the automated system without code - purely through models that allow for dynamic changes. Simple policies may be specified in YAML (XACML), while complex policies can be described in DROOLS or other languages, all through SDC. An example of a policy design would be:

```
if cpu_load > 80% && duration > 5 minutes {
        add 1 instance of VNF
}
```

[13] The VID application enables users to instantiate infrastructure services from SDC, along with their associated components, and to execute change management operations such as scaling and software upgrades to existing VNF instances.

- The A&AI component: provides real-time views of a system's resources, services, products and their relationships with each other. It tracks products, services, resources - both active and available - and additionally, A&AI tracks relationships by maintaining a graph database, e.g., subscriber → NS → VNFs → VMs → compute/storage/networking nodes, while allowing discovery, registration, and auditing

- The CLAMP (Closed Loop Automation Management Platform) tool: is used to create and configure policies/templates built-in SDC (Service Design and Creation) to create closed control-loop service assurance loops. CLAMP interacts with other systems to deploy and execute the control loop. This component requests from DCAE the instantiation of microservices to manage the control loop flow. Furthermore, it creates and updates multiple policies in the Policy Engine that define the closed loop flow. Figure 22 shows a template being configured with the specific collector, threshold, and string match; it is then "attached" to a network service.

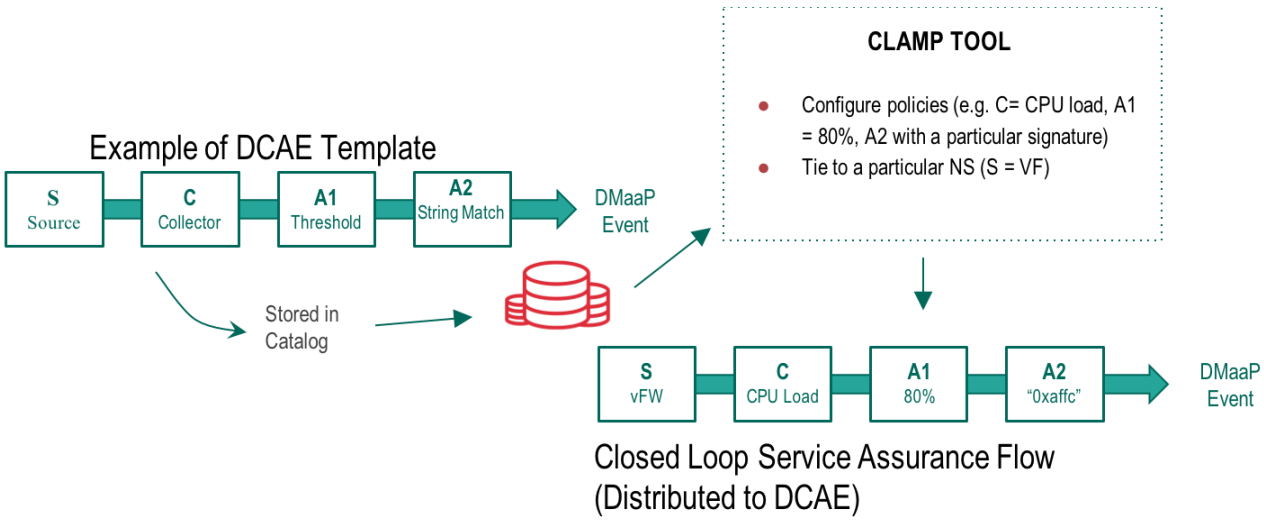

Figure 22: CLAMP workflow (source *[KAPADIA_ORC]*)

- Monitoring and analytics: ONAP has its own monitoring and analytics solution already integrated. It uses VES (VNF Event Stream) to deliver all the relevant data from the VNFs using an integrated set of agent and collector code artifacts. The VES Plugin, consumes the collectd data and translates it to VES format and forwards data to the VES Collector. Then, the VES Collector receives events and publishes them on the ONAP message bus. It also saves them in a local time-series database. An example of such a configuration is illustrated in Figure 23.

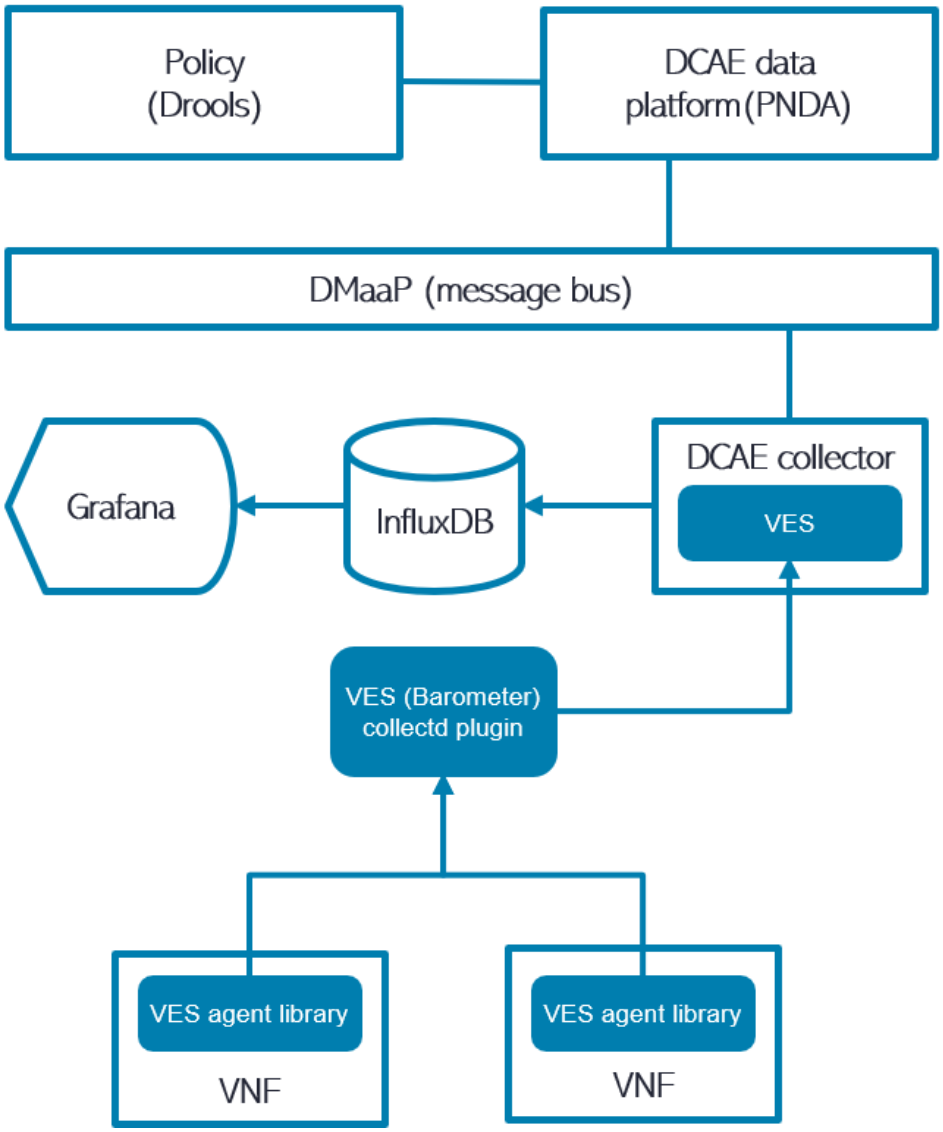

Figure 23: ONAP monitoring & analytics

With this solution, ONAP offers a VES Agent library enabling VNFs to directly support VES for delivering measurements, the DCAE collector, which receives events in the VES format and publishes them on the ONAP message bus DMaaP (Data Movement as a Platform), as well as saves them in a local time-series database and a DCAE data analytics platform that can use PNDA (not necessarily; it is just one of the possible tools). The flow is: VNF → Collector → DCAE bus → Analytics → ONAP bus (DMaaP) → Policy → ONAP bus (DMaaP) → SO

3.3.1.2 OSM

The Open Source MANO (OSM), promoted by ETSI, is an open source management and orchestration stack aligned with the ETSI NFV Information models *[ETSI_OSM]*. The OSM reference architecture is illustrated in Figure 24, highlighting the Service platforms and the common management.

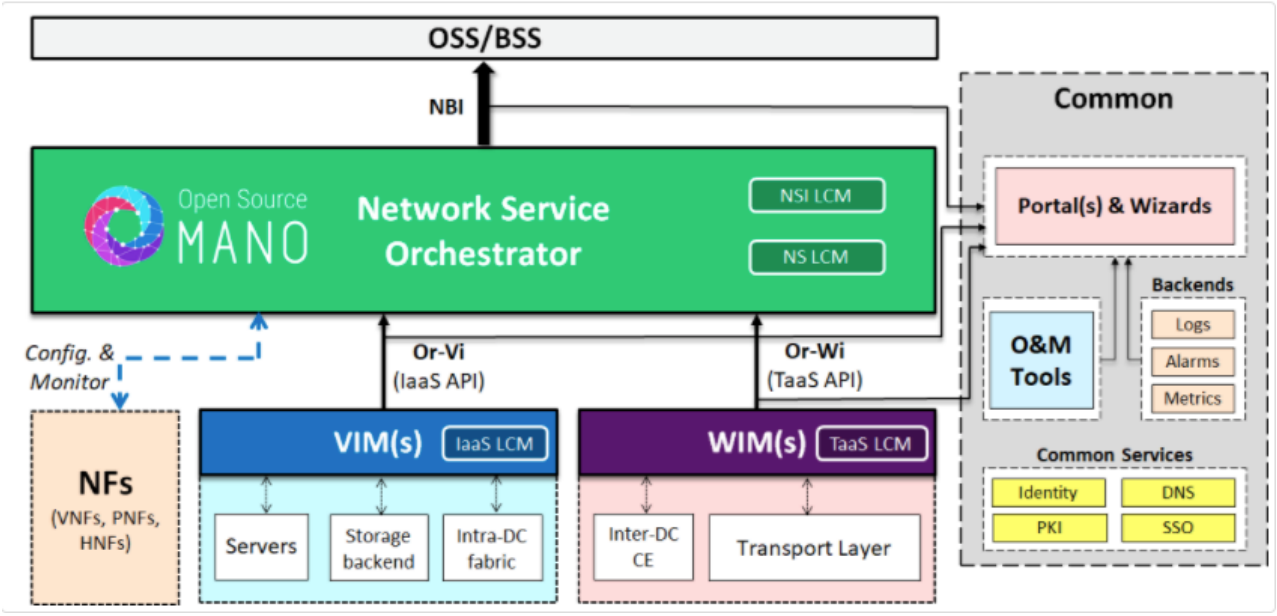

Figure 24: OSM Reference Architecture (source *[ETSI_OSM]*)

The OSM acts as a Network Service Orchestrator, being responsible for Network Services and Network Slices in 5G networks. OSM is able to orchestrate different types of Network Functions (NFs), including VNFs, PNF, Hardware Network Functions (HNFs) and Containerized Network Functions (CNFs). Such functions can run on different infrastructures and are managed through the Infrastructure as a Service (IaaS) API or via the Or-Vi interfaces. The Virtual Infrastructure Manager (VIM) is responsible for managing the required computation resources and storage to run the network functions. When diverse infrastructures (i.e., VIMs are connected), there is the need to manage the inter-datacentre connectivity, as well as the associated security levels (e.g., encryption of transport layer information). The WAN Infrastructure Manager (WIM) components are responsible to make the connection between datacentres employing SDN approaches.

The Common management plane of OSM includes a set of common services like the identity, which is responsible for user identities in the diverse platforms. The management of identity information commonly relies on Lightweight Directory Access Protocol (LDAP). The Single Sign On (SSO) service allows to manage context of users in the diverse platforms, in particular regarding authentication purposes. The PKI and DNS services act as supporting services for certificate management and naming resolution, respectively.

This management plane also includes OAM tools to manage the lifecycle of infrastructures (e.g., operations and maintenance of OpenStack deployments), the backends that allow to collect metrics (i.e. employing time series databases) and run alarmistic on the collected data. In addition, the backends also include logging facilities. Dashboards with information regarding the status of platforms in real-time, but also mechanisms to facilitate the deployment of complex services are also made available.

The OSM has evolved through diverse versions, the version 9 is the most recent (at the date of writing this report). In addition, several projects *[FLAVIO_SLIMANO]* targeting the deployment of VNFs in 5G networks have considered OSM as the underlying architecture to manage and orchestrate resources in a standardized fashion.

OSM includes a monitoring collector, to collect metrics at the infrastructure level and per VNF, as illustrated in Figure 25. The metrics regarding the infrastructure are provided by the VIM (e.g., ceilometer in OpenStack), while the metrics of the VNFs, are described in the templates and collected through JuJu Metrics service.

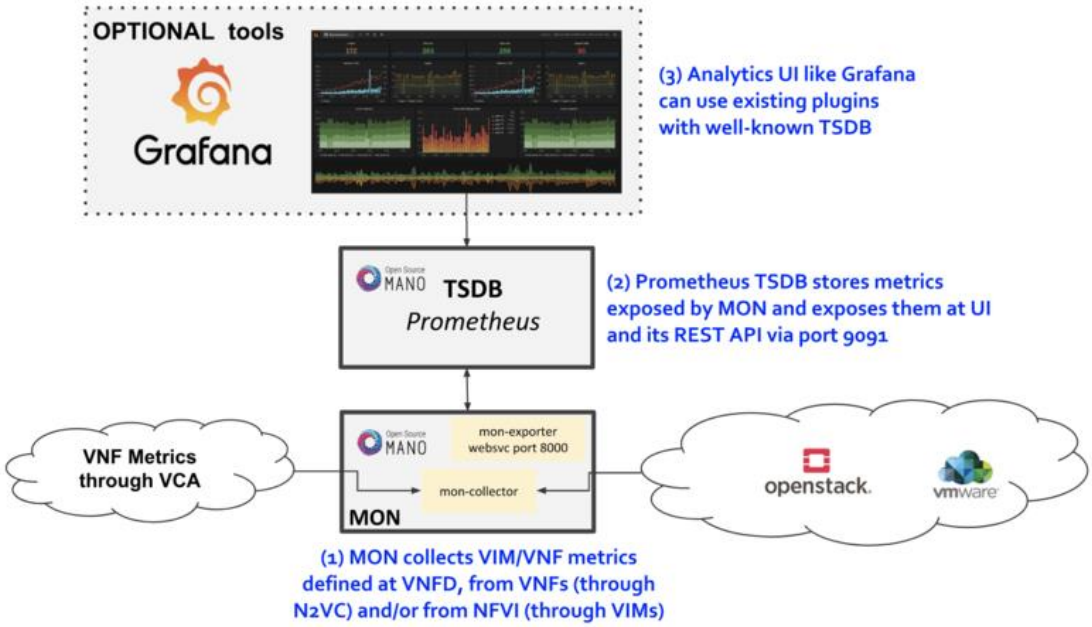

Figure 25: OSM and monitoring (source *[ETSI_OSM_USERGUIDE]*)

OSM does not hold a DCAE module as ONAP [LOCUS_5G], nonetheless OSM is able to support automated network service orchestration, considering the policies defined for the Network Service (NS) and VNF levels. Considering the monitored metrics, at infrastructure and application levels, OSM is able to regulate the VNF scaling operations. In addition, OSM has also a Northbound API that also enable the configuration of the service lifecycle (i.e., start/stop/resume services), such configuration can be performed through 3rd party services that implement resource optimization approaches.

3.3.1.3 Other platforms

This subsection includes other orchestration platforms that are the result of research initiatives or are not open-source and have costs associated for enterprise usage.

3.3.1.3.1 OpenBaton

OpenBaton is an open-source implementation of the ETSI MANO specification, providing a NFV Orchestrator and a generic VNFM *[CARAGUAY_ORC]*. For monitoring, it offers a Zabbix plugin and for VIM implementation, an OpenStack driver *[GIUSEPPE_ORC]*.

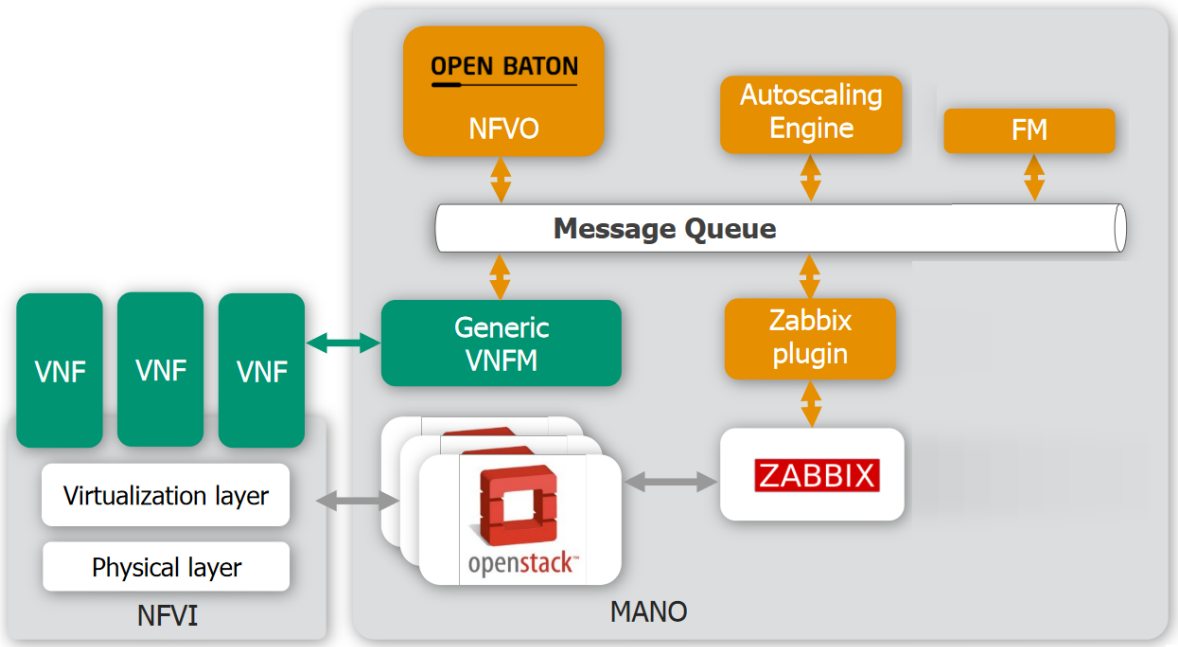

Figure 26: OpenBaton architecture (source *[CARAGUAY_ORC]*)

An autoscaling engine interacts with the monitoring system for retrieving real time data and dynamically scales a network service record based on policies contained in the network service descriptor. The Fault Management System (FM) creates rules for detecting faults on the monitoring system and registers triggers (and every time a trigger is received, it executes an action).

3.3.1.3.2 Cloudify

Cloudify is a multi-cloud orchestration software based in Topology and Orchestration Specification for Cloud Applications (TOSCA) that leverages both NFVO and Generic Virtualized Network Function Manager (G-VNFM) in the context of the ETSI NFV-MANO *[AMARAL_ORC]*. One of its key characteristics is the modeling of composed services using the Domain-Specific Language (DSL). It allows the integration with multiple VIMs, containers, and even with external and non-virtualized infrastructure. Cloudify includes deployment in any cloud or datacenter environment, monitoring all aspects of the deployed application, detecting problems and failures, remediating assets manually or automatically, and handling maintenance tasks.

3.3.1.4 Summary

Table 7 *[SOUSA_ORC]* displays a comparison performed among orchestration platforms (*legacy* refers to the capability of handling technologies such as MPLS, BGP, SONET / SDH, and WDM):

Table 7: Comparison of orchestration platforms

Solution	Leader	VNF definition	SDN	NFV	Legacy	VIM	Multiple domains	O-RAN support
ONAP	Linux Foundation	TOSCA, YANG	✓	✓	✓	✓	✓	✓
OSM	ETSI	YANG	✓	✓		✓	✓	✓
OpenBaton	Fraunhofer Institute Technical U. Berlin	TOSCA		✓		✓		

ONAP and OSM are shown as the most complete orchestration platforms, especially since they offer the possibility of spanning across multiple domains including RAN and optical transport, which enables an end-to-end service. OSM in recent versions (v8 onwards) brings support for multiple domains with the WAN Infrastructure Manager components, with ONAP having the advantage of having TOSCA support. In addition, comparative studies *[GIRMA_OSM]* highlight that the performance of ONAP is superior in terms of resource usage (e.g., OSM tends to use more CPU and memory), and is almost similar regarding the deployment processing delays.

Not only OSM and ONAP have the strongest community of developers, due to their open-source nature and support from ETSI and Linux Foundation, both solutions were the only two chosen orchestration frameworks for the Open-RAN, an alliance of telecom companies responsible for the specification of APIs and interfaces that can enable RAN virtualization.

Another advantage of ONAP is that it offers integration with Cloudify, by using the Cloudify's DCAE and OOM (ONAP Operations Manager) on the ONAP framework. Since Cloudify is TOSCA based, when merged with ONAP, a TOSCA SO, such as Aria, must be used. Figure 27 illustrates this concept.

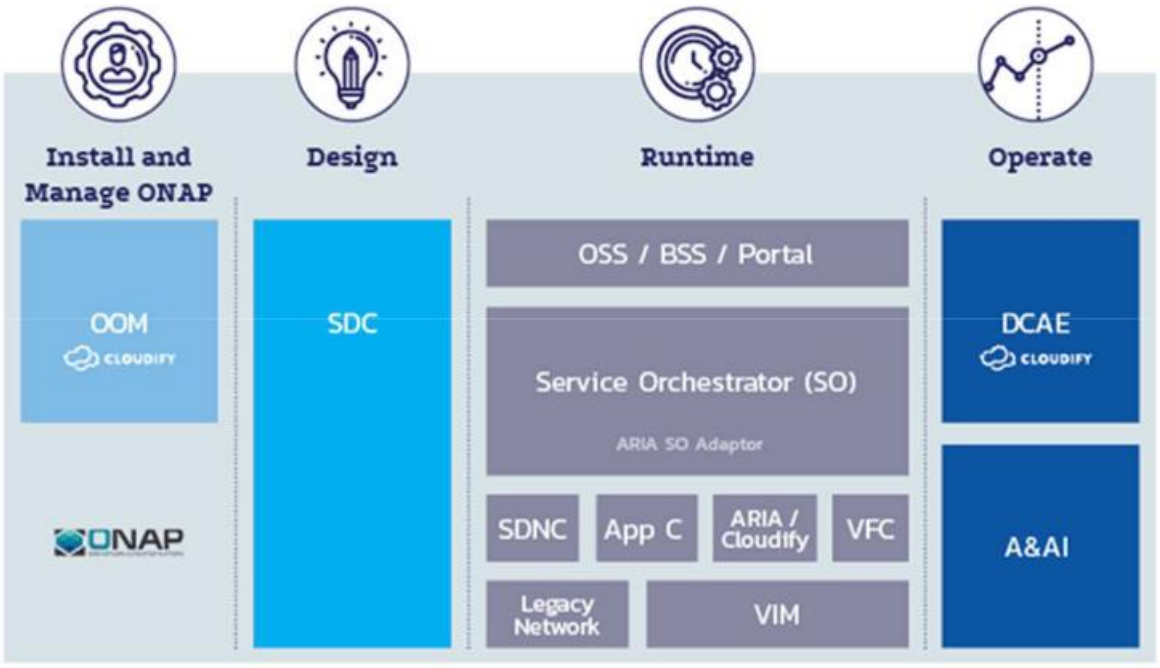

Figure 27: Cloudify/ONAP fusion (source *[SOUSA_ORC]*)

A potential drawback of such a solution as the one illustrated in this figure, is that some features of Cloudify are only available in the premium (paid) version.

3.4 Virtualization

This section overviews solutions to virtualize infrastructures, to support network functions in different hardware platforms.

3.4.1 Introduction

The fundamental concept of virtualization relies on the coexistence of host and guest machines, where the host is the machine in which the virtualization occurs, and the guest machines represent the virtual machines operating through the host. The tool, used with the purpose of supervising the virtual machines, is called the hypervisor, which comes in two flavours: if the hypervisor replaces host system, this is called a native (or bare metal) hypervisor; and if the hypervisor operates under the operating system, this is called a hosted hypervisor.

3.4.1.1 OpenStack

The core component of a cloud is the cloud controller, that provides cross-domain automation *[SOLBERG_VIR]*. The most popular open source cloud controller is OpenStack, followed by Apache CloudStack and OpenNebula. Figure 28 illustrates the main components of OpenStack and their relationship with Compute, Network and Storage blocks:

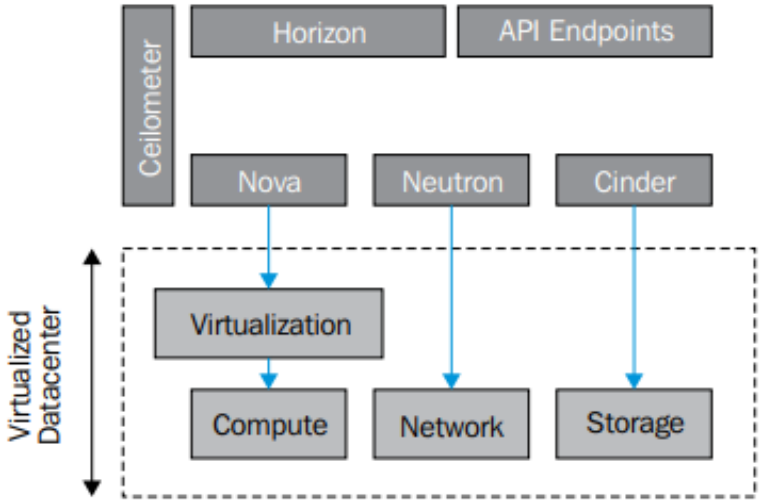

Figure 28: OpenStack virtualization (source: *[SHRIVASTWA_VIR]*)

Other important components of OpenStack are the Keystone, the Glance and the Swift. The OpenStack flow between the various components is illustrated in Figure 29.

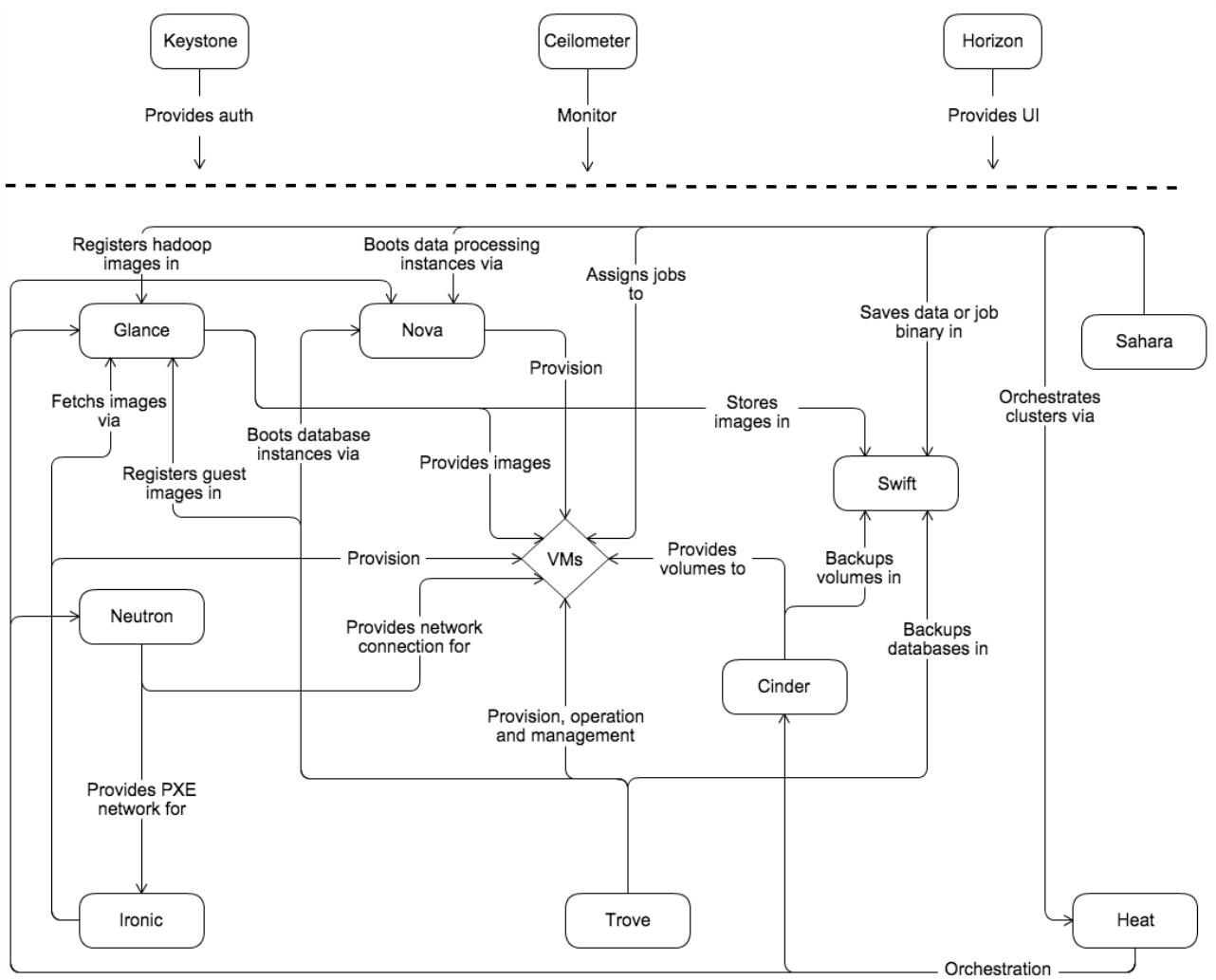

Figure 29: Flow of OpenStack components

- Horizon is the canonical implementation of OpenStack's dashboard, which is extensible and provides a web based user interface to OpenStack services.
- Neutron provides an API for creating ports, subnets, networks, and routers. Additional network services such as firewalls and load balancers are provided in some OpenStack deployments. It is focused on delivering networking-as-a-service (NaaS) in virtual compute environments.
- Nova supports various hypervisors for virtual machines such as KVM, and VMware. It also supports Linux Containers (LXC). Its purpose is to implement services and associated libraries to provide massively scalable, on demand, self-service access to compute resources, including bare metal, virtual machines, and containers.
- Glance service allows the storage and retrieval of images and corresponding metadata, i.e., it allows the store of OS templates that need to be made available for the users to deploy. Glance image services

include discovering, registering, and retrieving virtual machine images. Glance has a RESTful API that allows querying of VM image metadata as well as retrieval of the actual image. VM images made available through Glance can be stored in a variety of locations from simple filesystems to object-storage systems like the OpenStack Swift project.

- Cinder provides block storage to the Nova VMs. Its subsystems include a volume manager, a SQL database and an authentication manager. It virtualizes the management of block storage devices and provides end users with a self-service API to request and consume those resources without requiring any knowledge of where their storage is actually deployed or on what type of device. This is done through the use of either a reference implementation (LVM) or plugin drivers for other storage.
- Ceilometer service is used to collect metering data. Its goal is to efficiently collect, normalise and transform data produced by OpenStack services. The data it collects is intended to be used to create different views and help solve various telemetry use cases.
- Keystone service provides identity and access management for all the components of OpenStack.
- When a compute instance is terminated, the ephemeral storage associated with the instance is deleted from the compute host on which it resided. Persistent storage provided in the OpenStack system is object storage, which is provided by the Swift service (the swift service is used to actually store Glance images).

Finally, Heat is a service that allows developers to store the requirements of a cloud application in a file that defines what resources are necessary for that application. Note that these are just some of the blocks, since various other ones are available, although they are not mandatory.

3.4.1.2 Kubernetes

Kubernetes, also known as k8s, is a platform for container[14] orchestration, which enables the deployment, scaling of containerized applications. Kubernetes supports self-healing approaches based on user-defined health checks, or on the unavailability of containers, enables load balancing to distribute the load between microservices.

[14] Containers is a standard unit of software with code and its dependencies to assure that the applications runs quickly and in a reliable fashion. Containers allow applications to run easily in different computing environments.

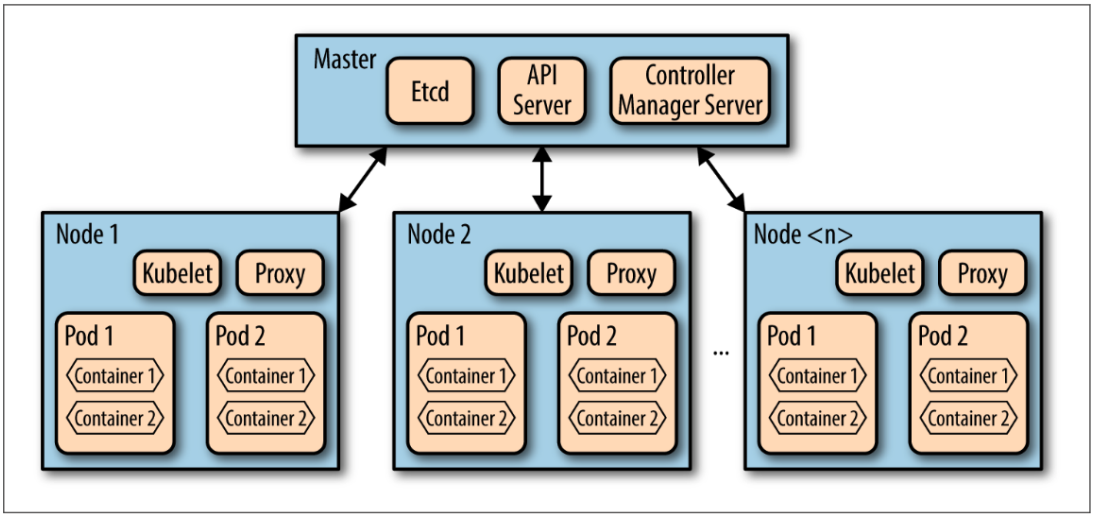

Figure 30: Architecture of Kubernetes (source *[DAVID_K8S]*)

The architecture of Kubernetes includes several components, as illustrated in Figure 30. The master manages the microservices and the load between nodes. Such management is performed through the API server, the controller management server and a daemon responsible to manage all the configurations (e.g., secrets, certificates, among others).

The node holds the Kubelet that is responsible to make the interface with the master (e.g., receive commands), the proxy component to hold network services for naming the diverse Pods and microservices. The Pods correspond to a collection of containers and volumes that share common resources like the filesystem and IP addresses. Other entities in the K8S architecture are also included, like the Replication controllers, which allow the distribution of load between the pods, and assure that the required availability levels are assured.
The networking model of Kubernetes mainly relies on the kube-proxy, which assure intra-pod communication via the localhost concept and ports for the microservices. The Inter-pod communication is performed through routable IP addresses and assigned names. In this regard, K8S uses an internal domain service to assign names to Pods *[ADRIAN_K8S]*. The flexibility of K8S allows it to adopt different solutions for DNS (KubeDNS, coreDNS), as well as services to manage policies like which applications can access a certain pod (access based on IP information). Besides the base services for networking, there is also the Container Network Interface (CNI) which provides IP address management for Pods and maintains routes for the virtual interfaces in a standard fashion.

The CNI allows the development of different solutions for the networking in k8s, via a system of network plugins. such plugins have different design goals, for instance Flannel is developed with simplicity in mind to enable L3

routing. The Calico plugin is intended to enable performance with security and flexible policies specification. Other plugins like KubeOVN aim to promote the integration of networking with enterprise level mechanisms, such as monitoring (e.g., integration with Prometheus for fault analysis), dynamic QoS for pods, policies to assign IP addresses for instance, static IP addresses can be assigned to stateful Pods. KubeOVN relies on the Open Virtual Network (OVN) project to promote the integration of physical network elements with logical network elements (virtual switches, virtual routers).

The management of microservices is also supported at the edge through, platforms that require fewer resources, but provide at the same time, a synchronized approach with the cloud, where k8s clusters run. The Kubedge is one of the projects under the umbrella of the CNCF[15] to enable resource optimization at the edge and cloud, in different computing platforms like ARM and x86. Kubedge can contribute to enhance resilience support, where service functions can be deployed close to the user, to manage IoT devices in a scalable fashion (e.g., manage video cameras). Through Kubedge the service functions can be orchestrated at the edge and synchronized with the cloud running in K8S or other orchestration platform.

The employment of K8s and related technologies provides benefits, in 5G context *[NOKIA_5G_CONTAINERS].* The risk of deploying services is reduced, for instance the maintenance of microservices is more flexible than services implemented as VMs (e.g., updates in VMs are costly). In addition, micro-services increase the resilience support due to the fault isolation, where a fault in a critical component (running as microservice) does not affect the whole system.

3.4.1.3 Service Mesh

Service Mesh is a dedicated infrastructure layer to facilitate service-to-service communication, through service discovery, routing, load balancing, traffic configuration, encryption, authentication, authorization and monitoring approaches *[NIST_800-204]*. Service mesh allows the declaration of policies defining the network behaviour, node's identity, and traffic flow.

[15] https://www.cncf.io/

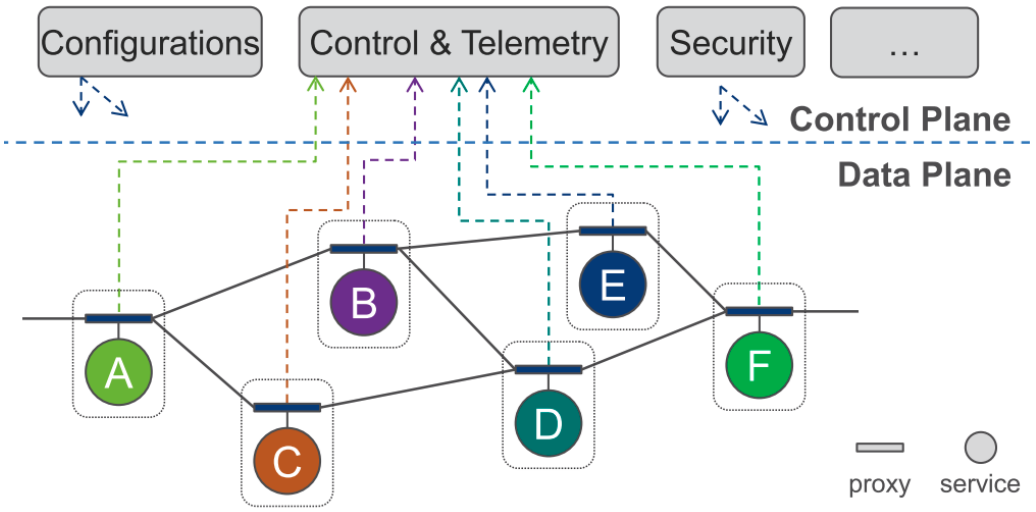

Figure 31: Service Mesh Architecture (source *[WLI_SMCSAFRO]*)

Service mesh includes two planes, as illustrated in Figure 31. The data plane includes proxies (often called as side-car proxy) to handle application request traffic between service instances (deployed as microservices or as VNFs). The control plane includes elements to configure the data planes, and provides points of aggregation for telemetry, load balancing.

Service Mesh is mainly designed for microservices architecture like Istio, Linkerd, but other approaches also support Virtual Machines (VM) deployments like Kuma or Consul. Service Mesh can be performed with VMs, through different approaches:

- Using a proxy sidecar (commonly relying on the Envoy solution *[ISTIO_BOOKINFO]*), where services running in the VM connect to the proxy to handle all the traffic and security. This approach, has the advantage of facilitating the interconnection with a running K8S deployment, as illustrated in Figure 32 for a simple application.
- Using specific interfaces, or a connect-native application approach [CONSUL_NATIVEAPP], where services connect to an API, or use a library to connect to the service mesh data plane.

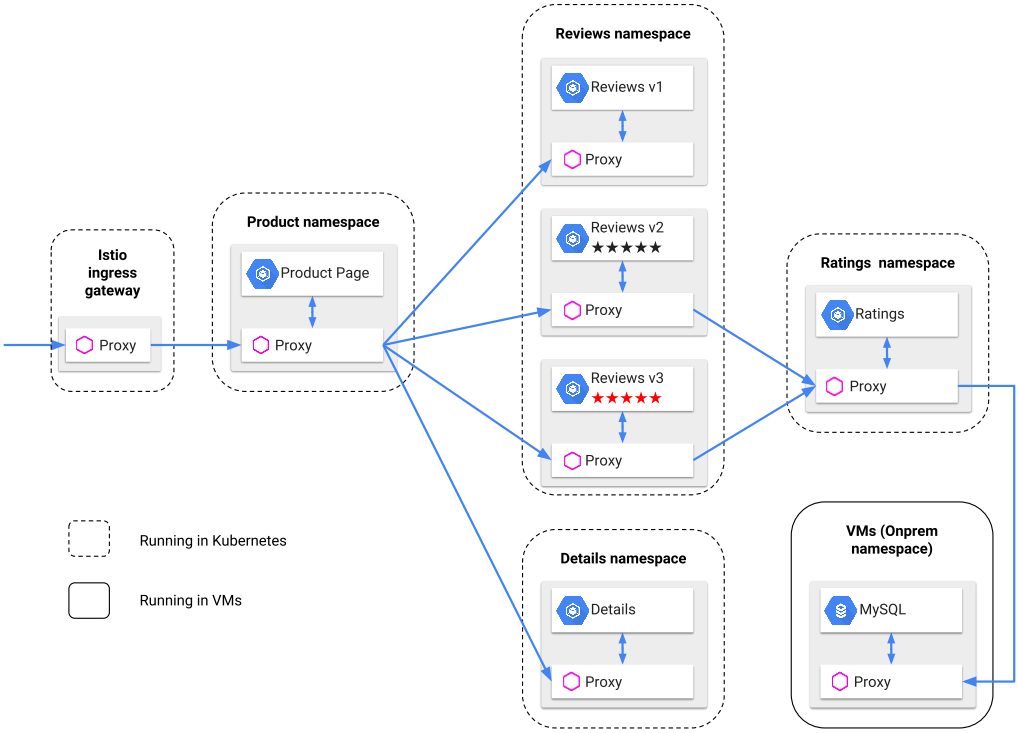

Figure 32: Service mesh example of application running in VMs and K8S (source *[ISTIO_BOOKINFO]*)

One of the benefits of Service Mesh is the added security by enabling strong identity, robust policies, mutual TLS (mTLS) encryption, and authentication, authorization and accounting (AAA) tools.

In addition, there is also ongoing specification regarding a standard interface for Service Management, the Service Management Interface (SMI) to enable a common interface for service mesh in Kubernetes *[LIU_ISTIO]*. Relying on the SMI, approaches like Meshery act as the service mesh management plane enabling the management of service mesh and their workloads in a unified fashion.

Commercial solutions, like the VMWARE-NSX [VMWARE_NSX] implement service mesh to enhance the flexibility of deployment in cloud data centers within different virtualization technologies (e.g., VMs, containers). In such contexts, service mesh includes support for telemetry of protocols like DNS, HTTP/S, firewall support and load-balancing at the data plane.

The OREOS platform can leverage from service mesh, considering the benefits for service reliability within approaches different virtualization technologies (e.g., containers, VMs).

3.4.1.4 Summary

Table 8 summarizes the virtualization approaches, highlighting their advantages and disadvantages.

Table 8: Virtualization solutions

Approach	**Advantages (++)**	**Disadvantages (--)**
OpenStack	• Modular architecture • Most popular solution with a large community of users and developers	• OpenStack is a collection of different projects, so it is necessary to install and configure each module separately.
Kubernetes	• Tailored for microservices. • adopted as the industry standard for microservices	• The split of applications into microservices, might not meet stringent latency requirements.
Service Mesh	• Does not require modifications to applications to enable resilience • Enables high availability and load balancing, even between different cloud solutions. • Facilitates securing services with support for mTLS.	• Mainly for microservices. • Mainly targeted for web services, relying on HTTP, gRPC. • Does not virtualize resources.

4. Conclusion

5G networks bring support for critical services, vehicular communications, with URLLC communications and other mechanisms like network slicing. The 5G SBA architecture allows the distribution of service functions at the edge, following MEC computing paradigm, or exploiting the benefits of different architectural models, such as Fog, Edge and Cloud computing.

The orchestration platform to be designed and developed in OREOS aims to comply with ETSI MEC specification, to enable applications to run at the edge, and distribute the diverse functions considering Fog and Cloud computing models. In this regard, the OREOS architecture will be aligned with the OpenFog reference architecture to support the Cloud-to-Object continuum paradigm.

The critical services targeted in the application scenarios of OREOS require low-latency and reliability. For this aim, the orchestration platform of OREOS will integrate the information from multiple tools, like monitoring, analytics and orchestration platforms such as ONAP or OSM. The monitoring tools allow the collection and monitoring of resources, like CPU, memory or disk usage, and network load in a given moment. The analytics solutions allow to perform intelligent queries in the monitored data, or to enable actions considering certain thresholds (e.g., raise an alarm when CPU is above 75% in a time period of five minutes). Analytics solutions, like Druid support a distributed architecture to store data and to perform exploratory analytics on large data sets with low latency. In this regard, it is expected that the orchestration platform of OREOS will support data streams from multiple sources (e.g., kafka, publish-subscribe models) for enhanced and complete analytics.

The orchestration of VNFs or CNFs in OREOS can leverage from the orchestration functionalities provided by ONAP or OSM, which are the most complete solutions, supporting different infrastructures like OpenStack, VMWare. In addition, ONAP already includes components to perform optimization of resources like the DCAE component, which is able to integrate policies and to automate the service lifecycle. Through DCAE the scaling of VNFs can be instrumented considering inputs from analytics solutions. The resources for VNFs can be optimized considering allocation policies that consider multiple criteria in NP-Hard problems. For instance, following a MILP formulation, or a heuristic approach to reduce energy consumption and latency in Fog nodes.

The OREOS architecture can also leverage from approaches like Kubedge or service mesh technologies. These can reduce latency and enhance the resilience support in services with minimal modifications. Solutions like Kubeedge facilitate the interconnection between the edge and cloud, and data acquisition at the edge in a

scalable fashion. The service mesh technology provides high availability mechanisms that are agnostic of the infrastructure and underlying orchestration platform.

This document presents the main technologies and approaches to enable the OREOS orchestration platform, aggregating contributions from tasks 2.1 and task 2.2.

Finally, the document will act as the base for the work in activity 3, regarding the specification of use cases and the analysis of respective requirements, in activity 2 regarding the specification of the architecture and the integration of the optimization approaches for resource allocation.

5. References

[5GAA_USECASE] T. Linget. *"C-V2X Use Cases Volume II: Examples and Service Level Requirements"*. 5GAA Automotive Association. White paper, 2020

[ADRIAN_K8S] Adrian Goins, Alena Prokharchyk, Murali Paluru. "Diving Deep into Kubernetes Networking", 2020, RANCHER.

[AIDI_RES] S. Aidi, M. F. Zhani, and Y. Elkhatib. "On improving service chains survivability through efficient backup provisioning". In 2018 14th International Conference on Network and Service Management (CNSM), 2018. pages 108–115, Rome, Italy. IEEE.

[AMARAL_ORC] T. Amaral et al. "Closed-loop Orchestration in 5G: Designing a Service Operation in Fulfillment Frameworks". In Joint EuCNC & 6G Summit, 2021 Porto, Portugal

[BEHESHTI_RES] N. Beheshti, and Y. Zhang. "Fast failover for control traffic in Software defined Networks". In Global Communications Conference (GLOBECOM), 2012 IEEE, pages 2665–2670, Anaheim, USA. IEEE.

[BITTENCOURT_CHOREO] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, and O. Rana. "The Internet of Things, Fog and Cloud continuum: Integration and challenges". Internet of Things, 2018. 3-4(1):134 – 155.

[BRAZIL_MON] B. Brazil. "Prometheus: Up & Running". O'Reilly Publishing, 2018

[BRITO_SOAFI] de Brito MS, Hoque S, Magedanz T, Steinke R, Willner A, Nehls D, Keils O, Schreiner F. "A service orchestration architecture for fog-enabled infrastructures". In: 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC). Valencia: IEEE; 2017. p. 127–32.

[BROGI_RU] A. Brogi, S. Forti, and A. Ibrahim. "Optimising QoS-Assurance, Resource Usage and Cost of Fog Application Deployments". Cloud Computing and Services Science, 2019. 168–189.

[CARAGUAY_ORC] A. Caraguay. "Monitoring and discovery for self-organized network management in virtualized and software defined networks". PhD thesis. 2017, Universida Complutense de Madrid

[CHEN_ML] Chen, X., Zhao, Z., Wu, C., Bennis, M., Liu, H., Ji, Y., & Zhang, H. (2019). "Multi-tenant cross-slice resource orchestration: A deep reinforcement learning approach". IEEE Journal on Selected Areas in Communications, 37(10), 2377-2392.

[CHEN_CLOUDLETARCH] Chen, M., Hao, Y., Li, Y., Lai, C.F., Wu, D. "On the computation offloading at ad hoc cloudlet: architecture and service modes". IEEE Communications Magazine 53(6), 18 - 24 (2015).

[DAVID_K8S] David K. Rensin, Kubernetes. "Scheduling the future at cloud scale". October 6, 2015, O'Reilly.

[DHIRAJ_PDAUPOST] Dhiraj Bhuyan, "Practical Data Analysis: Using Python & Open Source Technology", 2019.

[DJEMAI_MOB] T. Djemai, P. Stolf, T. Monteil and J. M. Pierson, "Mobility Support for Energy and QoS aware IoT Services Placement in the Fog," 2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Split, Croatia, 2020, pp. 1-7

[DU_FAIR] J. Du, L. Zhao, J. Feng and X. Chu, "Computation Offloading and Resource Allocation in Mixed Fog/Cloud Computing Systems with Min-Max Fairness Guarantee," in IEEE Transactions on Communications, vol. 66, no. 4, pp. 1594-1608, April 2018

[ESTEVA_ML] A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun, and J. Dean, "A guide to deep learning in healthcare," Nature Medicine, vol. 25, no. 1, pp. 24–29, 2019. [Online]. Available: https://doi.org/10.1038/s41591- 018- 0316-z

[ETSIMEC_FRA] ETSI, "Multi-access Edge Computing (MEC) Framework and Reference Architecture", ETSI GS MEC 003, v2.1.1, 2019-01, [Online]. Available at: https://www.etsi.org/deliver/etsi_gs/mec/001_099/003/02.01.01_60/gs_mec003v020101p.pdf

[ETSI_NFV] ETSI. (2012) "ETSI - Industry Specification Group for NFV". [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/nfv

[ETSI_OSM] ETSI, Open Source MANO – OSM Release NINE Release notes, December 2020 [Online]: Available: https://osm.etsi.org/wikipub/images/0/01/OSM_Release_NINE_-_Release_Notes.pdf

[ETSI_OSM_USERGUIDE] ETSI, Open Source MANO – OSM User Guide [Online]: Available: https://osm.etsi.org/docs/user-guide/05-osm-usage.html#monitoring-and-autoscaling

[FANGJIN_ANA] Fangjin Yang et al. "Druid: A Real-time Analytical Data Store". In Proceedings of the ACM International Conference on Management of Data (SIGMOD), 2014. Pages 157–168

[FARABET_ML] C. Farabet, C. Couprie, L. Najman and Y. LeCun, "Learning Hierarchical Features for Scene Labeling," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 8, pp. 1915-1929.

[FENG_FAIR] J. Feng, L. Zhao, J. Du, X. Chu and F. R. Yu, "Energy-Efficient Resource Allocation in Fog Computing Supported IoT with Min-Max Fairness Guarantees," 2018 IEEE International Conference on Communications (ICC), Kansas City, MO, USA, 2018, pp. 1-6

[Feng_ML] Feng, D., Haase-Schuetz, C., Rosenbaum, L., Hertlein, H., Glaeser, C., Timm, F., Wiesbeck, W., & Dietmayer, K. "Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges". IEEE Transactions on Intelligent Transportation Systems, 2020.

[FLAVIO_SLIMANO] F. Meneses, M. Fernandes, D. Corujo, and R. L. Aguiar, "SliMANO: An Expandable Framework for the Management and Orchestration of End-to-end Network Slices," Proceeding 2019 IEEE 8th Int. Conf. Cloud Networking, CloudNet 2019, 2019.

[GIRMA_OSM] G. M. Yilma, Z. F. Yousaf, V. Sciancalepore, and X. Costa-Perez, "Benchmarking open source NFV MANO systems: OSM and ONAP," Comput. Commun., vol. 161, no. June, pp. 86–98, 2020.

[GIUSEPPE_ORC] Giuseppe Carella et al. "Prototyping NFV-based Multi-access Edge Computing in 5G ready Networks with Open Baton". In IEEE Conference on Network Softwarization (NetSoft), 2017 Bologna, Italy

[GODINHO_FAIR] N. Godinho, M. Curado and L. Paquete, "Optimization of Service Placement with Fairness", 2019 IEEE Symposium on Computers and Communications (ISCC), Barcelona, Spain, 2019, pp. 1-6.

[GONÇALVES_MOB] D. Gonçalves, K. Velasquez, M. Curado, L. Bittencourt and E. Madeira, "Proactive Virtual Machine Migration in Fog Environments," 2018 IEEE Symposium on Computers and Communications (ISCC), Natal, Brazil, 2018, pp. 00742-00745

[GUERRERO_LAT_B] C. Guerrero, I. Lera, and C. Juiz. "Evaluation and efficiency comparison of evolutionary algorithms for service placement optimization in fog architectures". Future Generation Computer Systems, 2019. 97(1):131 – 144.

[GUERRERO_LAT] C. Guerrero, I. Lera, and C. Juiz. "A lightweight decentralized service placement policy for performance optimization in fog computing". Journal of Ambient Intelligence and Humanized Computing, 2019. 10(1):2435–2452.

[HARTUNGA_MON] M. Hartunga and M. Korner. "SOFTmon - Traffic Monitoring for SDN". Procedia Computer Science, 2017. 110:516–523.

[HOCHREITER_ML] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Computation, 1997.

[ISTIO_BOOKINFO] Istio, bookinfo App [online], Available: https://istio.io/latest/docs/examples/virtual-machines/

[ITU-R_M.2410] ITU-R M.2410-0, Minimum requirements related to technical performance for IMT-2020 radio interface(s).

[JEAN_PINOT] J. F. Im et al., "Pinot: Realtime OLAP for 530 Million Users ACM Reference Format," Sigmod, pp. 583–594, 2018.

[JIANG_ORCH] Y. Jiang, Z. Huang, and D.H.K. Tsang . "Challenges and Solutions in Fog Computing Orchestration". IEEE Network, 2018. 32(3):122–129.

[JIONG_xURLLC] Jiong Park et al., "Extreme URLLC: Vision, challenges, and key enablers," arXiv, 2020.

[KAPADIA_ORC] A. Kapadia. "ONAP Demystified", 2nd ed. Aarna Networks, Inc., 2018

[KOCJAN_MON] W. Kocjan and P. Beltowski. Learning Nagios. Packt Publishing, 2016

[KRIZHEVSKY_ML] Krizhevsky, A., Sutskever, I. and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," Communications of the ACM, 60(6), pp. 84–90, 2017.

[LEE_ML] Lee, W., Kim, M., & Cho, D. H. (2018). "Deep power control: Transmit power control scheme based on convolutional neural network". IEEE Communications Letters, 22(6), 1276-1279.

[LERA_RES] I. Lera, C. Guerrero, and C. Juiz. "Availability-aware service placement policy in fog computing based on graph partitions". IEEE Internet of Things Journal, 2019. 6(2):3641–3651.

[LEY_ANA] E. Ley, D. Jacobs. "Rules-based Analysis With JBoss Drools: Adding Intelligence to Automation". In Proceedings of ICALEPCS, 2011 Grenoble, France.

[LI_ML] Li, R., Wang, C., Zhao, Z., Guo, R., & Zhang, H., "The LSTM-based advantage actor-critic learning for resource management in network slicing with user mobility". IEEE Communications Letters, 24(9), 2020.

[LI_ML_SLICE] Li, R., Zhao, Z., Sun, Q., Chih-Lin, I., Yang, C., Chen, X., ... & Zhang, H., "Deep reinforcement learning for resource management in network slicing". IEEE Access, 6, 74429-74441, 2018.

[LIANG_ML] Liang, L., Ye, H., Yu, G., & Li, G. Y., "Deep-learning-based wireless resource allocation with application to vehicular networks". Proceedings of the IEEE, 108(2), 341-356, 2019.

[LIEFTING_MON] N. Liefting, B. Baekel. "Zabbix 5 IT Infrastructure Monitoring Cookbook", Packt Publishing, '21.

[LIU_CONCERT] Liu J, Zhao T, Zhou S, Cheng Y, Niu Z. "Concert: a cloud-based architecture for next-generation cellular systems". IEEE Wireless Communications. 2014;21(6):14–22

[LIU_ISTIO] Liu Sun and D. Berg, "Istio Explained Getting Started with Service Mesh", 2020

[LOCUS_5G], Locus H2020 project, D2.4, 2021, available at: https://www.locus-project.eu/wp-content/uploads/2021/02/Deliverables-officially-submitted_D2.4_D2.4-9-8-20-nbm-final.pdf

[LOPEZ_ML] M. M. Lopez and J. Kalita, "Deep learning applied to NLP," CoRR, vol. abs/1703.03091, 2017. [Online]. Available: http://arxiv.org/abs/ 1703.03091

[LU_MOB] Lu, T., Chang, S., and Li, W. "Fog computing enabling geographic routing for urban area vehicular network". Peer-to-Peer Networking and Applications, 2018. 11(4):749–755.

[MAHMUD_LAT] R. Mahmud, S.N. Srirama, K. Ramamohanarao, and R. Buyya. "Quality of Experience (QoE)-aware placement of applications in Fog computing environments". Journal of Parallel and Distributed Computing, 132(1):190 – 203, 2019.

[MALIK_RES] A. Malik, B. Aziz, M. Adda, and C.-H.Ke, C.-H. "Smart routing: Towards proactive fault handling of software-defined networks". Computer Networks, 2020. 170(1):1389–1286.

[NASIR_ML] Nasir, Y. S., & Guo, D., "Multi-agent deep reinforcement learning for dynamic power allocation in wireless networks". IEEE Journal on Selected Areas in Communications, 37(10), 2239-2250, 2019.

[NEWMAN_GP] M. E. J. Newman and M. Girvan. "Finding and evaluating community structure in networks", Phys. Rev. E, 69:1–15, 2004.

[NIST_800-204] R. Chandramouli, "NIST Special Publication 800-204 - Security strategies for microservices-based application systems," NIST Spec. Publ., 2019.

[NOKIA_5G_CONTAINERS] Rob McManus, "Containers and the evolving 5G cloud-native journey," Nokia, 2020.

[OPENFOG_RAFC] OpenFog Consortium, "OpenFog Reference Architecture for Fog Computing", February 2017.

[PAGE_GP] L. Page, S. Brin, R. Motwani, and T. Winograd. "The PageRank Citation Ranking: Bringing Order to the Web", Technical Report 1999-66, Stanford InfoLab, 1999.

[PALLEWATTA_RU] S. Pallewatta, V. Kostakos, R. Buyya. "Microservices-based IoT Application Placement within Heterogeneous and Resource Constrained Fog Computing Environments". 12th IEEE/ACM International Conference on Utility and Cloud Computing, Pages 71–81, 2019

[PARES_GP] F. Parés, D. G. Gasulla,A. Vilalta, J. Moreno, E. Ayguadé, J. Labarta, U. Cortés, and T. Suzumura. "Fluid communities: A competitive,scalable and diverse community detection algorithm". InComplex Networks & Their Applications VI, pages 229–240, Lyon, France. Springer, 2018.

[PNDA_GUIDE] PNDA, "PNDA Guide", [Online], Available: http://pnda.io/guide.

[REHMAN_RES] A. U. Rehman, R. L. Aguiar, and J. P. Barraca. "Fault-tolerance in the scope of software-defined networking (SDN)". IEEE Access, 2019. 7(1):124474 – 124490.

[ROMAN_COSOLAP] Roman Leventov, "Comparison of Open Source OLAP Systems for Big Data: ClickHouse, Druid, Pinot", [Online] Available at: https://leventov.medium.com/comparison-of-the-open-source-olap-systems-for-big-data-clickhouse-druid-and-pinot-8e042a5ed1c7

[SALATINO_ANA] M. Salatino, M. Maio and E. Aliverti. "Mastering JBoss Drools 6". Packt Publishing, 2016

[SHI_LAT] C. Shi, Z. Ren, K. Yang, C. Chen, H. Zhang, Y. Xiao, and X. Hou, "Ultralow latency cloud-fog computing for industrial Internet of Things," in 2018 IEEE Wireless Communications and Networking Conference (WCNC). Barcelona, Spain: IEEE, June 2018, pp. 1–6.

[SHOKHIN_MON] A. Shokhin. "Network monitoring with Zabbix". Bachelor's Thesis, 2015, Mikkeli University of Applied Sciences

[SHRIVASTWA_VIR] A. Shrivastwa and S. Sarat. "Learning OpenStack". Packt Publishing, 2015

[SKARLAT_RU] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner. "Optimized IoT service placement in the fog". Service Oriented Computing and Applications, 2017.11(4):427–443.

[SKJENSEN_TSMS] S. K. Jensen, T. B. Pedersen, and C. Thomsen, "Time series management systems: A survey," arXiv, vol. 29, no. 11, 2017.

[SOLBERG_VIR] M. Solberg and B. Silverman. "OpenStack for architects". Packt Publishing, 2017.

[SOUSA_ORC] N. Sousa et al. "Network Service Orchestration: A Survey". Computer Communications, 2019.

[TAHEROZADEH_MON] S. Taherizadeh et al. "Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review". The Journal of Systems and Software, 2018. 136: 19–38.

[TANEJA_RU] M. Taneja, and A. Davy. "Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm". In 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pages 1222–1228, 2017. Lisbon, Portugal. IEEE.

[TAYYABA_ML] Tayyaba, S. K., et al., "5G vehicular network resource management for improving radio access through machine learning". IEEE Access, 8, 6792-6800, 2020.

[TRAKADAS_MON] P. Trakadas et al. Scalable Monitoring for Multiple Virtualized Infrastructures for 5G Services. In The Seventeenth International Conference on Networks (ICN), 2018

[TNOVA_MON] I. Koutras et al, *"Monitoring and maintenance"*. Deliverable D4.41 of grant agreement 619520, 2015

[VAQUERO_FOG] Vaquero LM, Rodero-Merino L., "Finding your way in the fog: Towards a comprehensive definition of fog computing". SIGCOMM Comput Commun Rev. 2014; 44(5):27–32.

[VELASQUEZ_CHALL] K. Velasquez, D. Perez Abreu, et al., "Fog orchestration for the internet of everything: state-of-the-art and research challenges". Journal of Internet Services and Applications, 2018. 9(1):1–23.

[VELASQUEZ_LAT] K. Velasquez, D. Perez Abreu, L. Paquete, M. Curado, and E. Monteiro, "A Rank-based Mechanism for Service Placement in the Fog," in 2020 IFIP Networking. Paris, France: IEEE, 2020, pp. 64–72.

[VELASQUEZ_SORTS] K. Velasquez, D. Perez Abreu, D. Gonçalves, L. Bittencourt, M. Curado, E. Monteiro, and E. Madeira. "Service Orchestration in Fog Environments". In 2017 IEEE 5th International Conference onFuture Internet of Things and Cloud (FiCloud), 2017. pages 329-336, Prague,Czech Republic. IEEE.

[VMWARE_NSX] Vmware, VMWARE-NSX [online], Available: https://www.vmware.com/topics/glossary/content/service-mesh.html

[WANG_ML] Wang, S., Liu, H. et al., "Deep reinforcement learning for dynamic multichannel access in wireless networks". IEEE Transactions on Cognitive Communications and Networking, 4(2), 257-265, 2018.

[WANG_ML_SPATIO] Wang, J., Tang, J., Xu, Z., Wang, Y., Xue, G., Zhang, X., & Yang, D. "Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach". In IEEE INFOCOM 2017-IEEE Conference on Computer Communications (pp. 1-9), 2017.

[WLI_SMCSAFRO] W. Li, Y. Lemieux, J. Gao, Z. Zhao, and Y. Han, "Service Mesh: Challenges, state of the art, and future research opportunities," Proc. - 13th IEEE Int. Conf. Serv. Syst. Eng. SOSE 2019, 10th Int. Work. Jt. Cloud Comput. JCC 2019 2019 IEEE Int. Work. Cloud Comput. Robot. Syst. CCRS 2019, pp. 122–127, 2019.

[YFU_RTDIUBER] Y. Fu and C. Soman, "Real-time Data Infrastructure at Uber", vol. 1, no. 1. ACM, 2021.

[YTSENG_RTMS5NCP] Y. Tseng, G. Aravinthan, B. Berde, S. Imadaliz, D. Houatra, and H. Qiu, “Re-Think Monitoring Services for 5G Network: Challenges and Perspectives,” Proc. - 6th IEEE Int. Conf. Cyber Secur. Cloud Comput. CSCloud 2019 5th IEEE Int. Conf. Edge Comput. Scalable Cloud, EdgeCom, pp. 34–39, 2019.

[ZHAO_FAIR] L. Zhao, M. Du and L. Chen, "A new multi-resource allocation mechanism: A tradeoff between fairness and efficiency in cloud computing," in China Communications, vol. 15, no. 3, pp. 57-77, March 2018.