



UNIVERSIDAD
POLITECNICA
DE VALENCIA



Máster Universitario
en Tecnologías, Sistemas y
Redes de Comunicaciones

Data Analytics Platform for IoT Environments

Author: Abdullah Kassabji

Director: Carlos Enrique Palau Salvador

Start date: 01/02/2017

Objectives

The objective of this master thesis is to develop an analytic platform in virtual environment based on Docker to analyze the network components for any IoT ecosystem. Depends on network components, customized filters can be applied to analyze the received data and index it, store it and introduce it to the user using a capable and simple interface of Kibana to let users have the desired analytic information of these IoT network devices.

Methodology

Create a virtual platform with Ubuntu 16.04 using Docker 17.06.0-ce with its containers: Logstash, Elasticsearch and Kibana. A use case is considered by using two IoT network devices as data sources: Openvswitch log as internal source (installed on the same virtual machine) and TCP connection as external source.

Results

This platform can offer analytic information for different IoT network devices for different technologies (SDN, NFV) that are connected with. Kibana interface facilitates for user having statistical information on the received data, searching for a specific information in a customized timeline

Future lines

This platform can be upgraded to be worked with more complicated IoT networks, automate the analytic production to let the IoT platform to take actions automatically based on the analytic data of its network devices. This analytic platform can be used in 5G!Pogoda project for IoT ecosystem. 5G!Pogoda represents the next evolution step in softwarized networks as supported by NFV, SDN and aimed at by the 5G network evolution.

Abstract

With the appearance of the term of Big Data that is conjugated with the evolution of IoT devices and platforms, the need to have networks for IoT platforms more intelligent, more powerful, more efficient, more secure, more reliable, and more scalable has arised. The technologies of Software Defined Networking (SDN) and Network function virtualization (NFV) play an important role to facilitate the challenges and make IoT platform network components more effective with using less network resources. We tried to build a tool to analyze network data, measure the efficiency, and show statistical information of those IoT platform network components with different network technologies.

Author: Abdullah Kassabji, email: abkas@teleco.upv.es

Director: Carlos Enrique Palau Salvador, email: cpalau@dcom.upv.es

Delivery date: 08-09-17

INDEX

I. Introduction	4
II. IoT Classifications	4
II.1. Consumer IoT (cIoT).....	5
II.2. Industrial IoT (IIoT)	5
III. The IoT architecture model	5
IV. Modern IoT connectivity overview	6
IV.1. Zigbee	6
IV.2. Bluetooth.....	6
IV.3. Wifi and Low-Power Wifi (LP-Wifi)	7
IV.4. Low Power Wide Area (LPWA).....	8
IV.5. 3GPP Cellular: MTC	8
V. Study the state of the art	9
VI. Data analytics platform components	11
VI.1. Docker.....	11
VI.2. Logstash.....	15
VI.3. Elasticsearch	16
VI.4. Kibana.....	18
VII. ELK platform: architecture & setup	18
VI.1. Platform architecture.....	18
VI.2. Platform setup.....	20
VIII. Run ELK platform	22
IX. Results after running ELK platform	24
Acknowledgment	27
Annexes	28
References	32

I. Introduction

The Internet of Things (IoT) is the revolution of the way we live and work by a lot of new services build on smooth interaction between a large number of heterogeneous devices. According to Ericsson, there will be more than 50 billion network-enabled devices connected in the world by 2020 [1] comparing to approximately 3.4 billion people using the Internet at the end of 2016 (47% of the world's population) [2]. Internet of things dramatically widens the internet's scope from people-operated computers towards autonomous smart devices.

Last years, a large number of different communication technologies gradually emerged to support the internet of things. Some of these technologies are prevalent in specific application domains such as Bluetooth Low Energy in Personal Area Networks [3], and Zigbee in Home Automation systems [4]. Others, such as WiFi, Low Power Wide Area Networks (LPWA) [5], and cellular communications (such as 3GPP - 4G machine-type communications), have a much larger scope. In addition, such landscape is constantly and rapidly evolving, with new technologies being regularly proposed, and with existing ones moving into new applications.

II. IoT Classifications

Regardless if the IoT device is cloud based which is connected to the internet directly, there are two kinds of IoT starting to rise regarding to business models and technologies (Fig.1) ; *consumer IoT* (cIoT) and *industrial IoT* (iIoT) [6]. These two services share some *general* communication requirements [7] such as scalability, need for lean protocol stack implementations in constrained

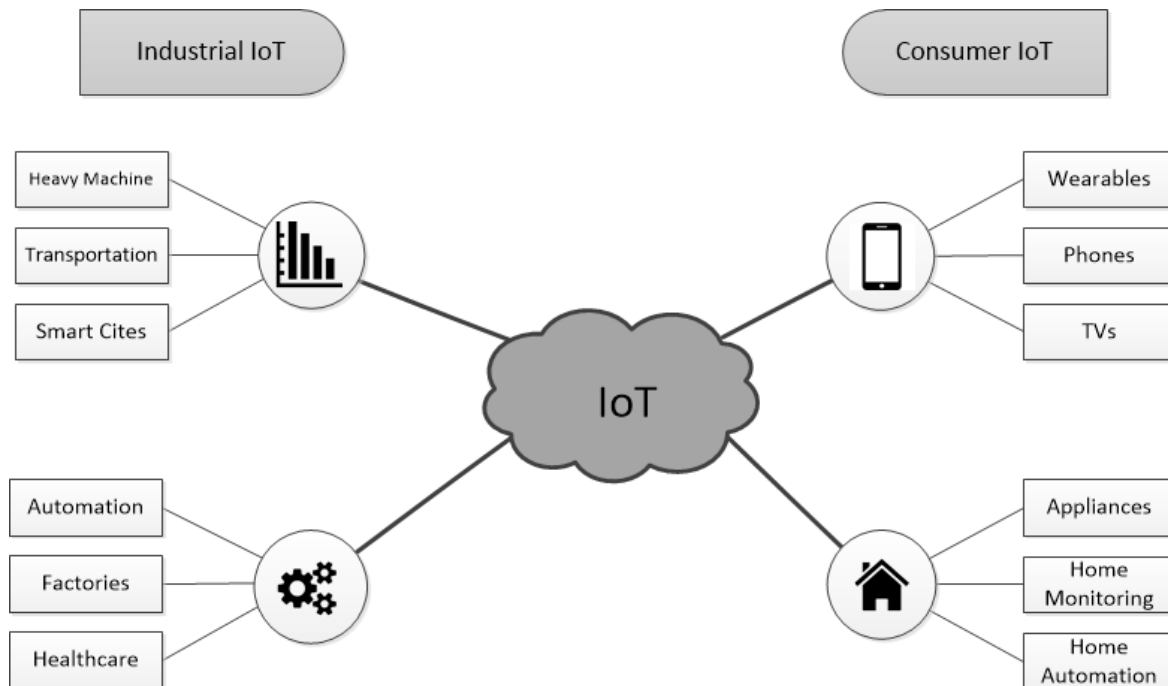


Fig.1 IoT classifications

devices, and friendliness to the IP ecosystem. There are also some *specific* communication requirements [7] can be very different between the two services in terms of reliability, QoS (latency, throughput, etc) and privacy.

II.1 *Consumer IoT (cIoT)*

Consumer IoT aims to improving the quality of people's life by saving time and money. It covers the interconnection of consumer electronic devices, as well as of (virtually) anything belonging to user environments such as homes, offices, and cities.

cIoT communications are typically machine-to-user, and usually in the form of client-server interactions. In cIoT, desirable features of networked things are low power consumption, ease of installation, integration and maintenance. For example, cIoT can found in fitness and health tracking systems, smart watches.

On the same time, applications of cIoT has to be protected by minimize the risk of exposing such sensitive data as someone's health status or life habits.

II.2. *Industrial IoT (iIoT)*

Industrial IoT focuses on the integration between Operational Technology (OT) and Information Technology (IT) and on how smart machines, networked sensors, and data analytics can improve business-to-business services in different market sectors and activities, from manufacturing to public services. It implies machine-to-machine (M2M) interactions, either for application monitoring or as part of a self-organized system, with a distributed control which does not need human interaction.

In general, the vast majority of iIoT communications, with some of cIoT communications, have often to follow binding requirements in terms of timeliness and reliability. Hence, the communication network must be designed in order to: (i) meet stringent delay deadlines; (ii) be robust to packet losses; (iii) be safe and resilient to damages.

3G and 4G cellular technologies, (especially 3GPP LTE) [8], are among the most appealing technologies in the modern IoT connectivity landscape. They offer wide coverage, relatively low deployment costs, high level of security, access to dedicated spectrum, and simplicity of management.

III. The IoT Architecture Model

To understand IoT, it is important to have the IoT architecture model in mind. Fig. 2 showing the IoT architecture [9] [10]. It is layered into Service Layer, Network Layer, and Sensing Layer. The Service Layer provides information services according to the user and system requirements. It may include powerful data centers and different data servers for the data mining, analysis, processing, storage, and applications (IoT platforms).

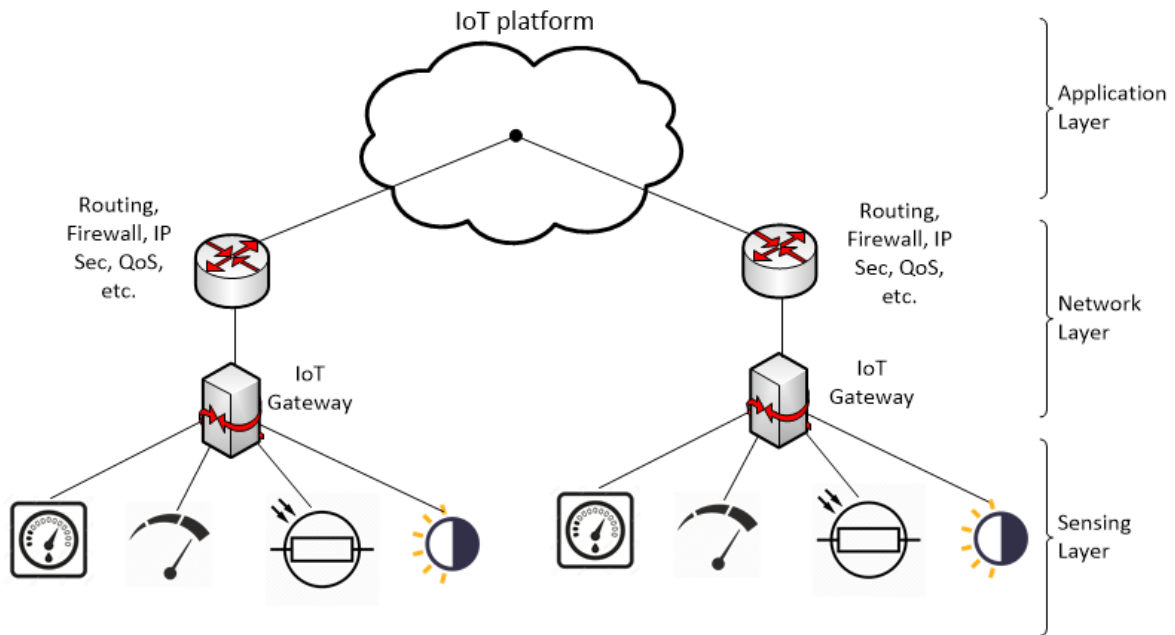


Fig.2 IoT architecture model

The Network Layer includes the gateway and the routes for the data transmission from gateways to different application users. Network layer has developed dramatically with the arrival of new technologies such as SDN and NFV. The Sensing Layer includes sensing devices such as sensors. The collected data are then transmitted to the gateway which is done usually by wireless transmission using MQTT (Message Queue Telemetry Transport) protocol or HPTT (Hypertext Transfer) protocol through different connectivity technologies.

IV. Modern IoT connectivity overview

IoT connectivity nowadays is well improved compared to how it was at the start point of IoT. The initial forms of IoT connectivity started in 80's with the tradition Radio Frequency Identification (RFID) technologies. It followed in 90's with the Wireless Sensor Networks (WSNs).

At the first decade of the 21st century, industrial affiliations and Standards Developing Organizations (SDOs) devoted their work in developing standardizations for low power IoT. The first ones available in the market were dominant solutions such as Z-Wave and WirelessHART.

Later on, more universal connectivity technologies have been evolved by SDOs like IEEE, 3GPP and IETF, facilitated the interconnection of constrained devices. IEEE802.15.4 and Bluetooth [11] are among short range low power solutions that played an important role in evolving IoT. Meanwhile, the 3rd Generation Partnership Project (3GPP) has been working toward supporting M2M applications on 4G broadband mobile networks, such as UMTS, and LTE, with the final aim of embedding M2M communications in the 5G systems.

In fact, there is no technology can be considered as a market leader. Now, IoT technologies reached a point with plenty of promising radio technologies: Low-Power WiFi, Low-Power Wide Area (LPWA) networks and many improvements for cellular M2M systems.

III.1. *Zigbee*

ZigBee is a low-cost, low-power, wireless mesh network standard which has been widely applied in Wireless Sensors Networks (WSNs). The first concept of ZigBee was in 1998, standardized in 2003, and finally revised in 2006. It builds on the IEEE802.15.4-2006 Physical (PHY) and Medium Access Control (MAC) standard specifications [11].

As many IoT applications are expected to exchange only a few bits and the current IEEE 802.15.4 PHY layer(s) adequate in terms of energy efficiency, thus, a standardized PHY layer will allow ultra-low rate transmissions over very narrow frequency bands using trivial amount of energy.

The IEEE802.15.4e standard [12] defined three new MACs. Among these, the Timeslotted Channel Hopping (TSCH) mode is the most promising one, facilitating energy efficient multi-hop communications, while reducing fading and interference.

III.2. *Bluetooth*

Bluetooth version 4.1 these days is a smart low energy version of Bluetooth (BLE), still designed for short-range communication (up to 50 m), but, mainly suitable for low-power, control and monitoring applications (e.g., automotive, entertainment, home automation, etc.) and it operates in the 2.4 GHz Industrial Scientific Medical (ISM) band, and defines 40 channels, with 2MHz channel spacing. Currently, Bluetooth only supports a single-hop topology with one master device communicating with several slave nodes, and a broadcast group topology, with an advertiser node broadcasting to several scanners.

In 2015, the Bluetooth Special Interest Group (SIG) announced the formation of the Bluetooth Smart Mesh working group to define the architecture for standardized mesh networking for Bluetooth. With this announcement, Bluetooth is intended to be a key enabling technology for some short-range IoT applications, such as in healthcare, and smart home domains [4].

In 2016, SIG has officially adopted Bluetooth version 5 as the standard's new specification with enhancement of "2x speed, 4x range and 8x data". Bluetooth 5 focus on increasing functionality for the Internet of Things (IoT) and delivers a "connectionless" IoT, advancing beacon and location-based capabilities in home, enterprise and industrial applications [13].

In terms of numbers, Bluetooth 5.0 introduces a new capability to increase the bandwidth to 2 Mbps. By doubling the amount of data that devices can transfer and reducing the time required for transmitting and receiving data. Bandwidth can be decreased to achieve up to 4x longer range while maintaining similar power requirements.

As a consequence, this technology will become very soon the most common communication mean for consumer applications.

III.3. *Wifi and Low-Power Wifi (LP-Wifi)*

Wifi or IEEE802.11 standard was released in version in 1997 as the first version, and designed without IoT perspective. Although nowadays is widespread, due to its high energy consumption compared to other standards (ZigBee, Bluetooth), Wifi has not been applied into M2M-IoT use cases.

Many hardware optimizations have been made by the IEEE802.11 society led to extremely energy efficient solutions. Wifi still suffers of poor mobility and roaming support despite the reduced energy consumption. It does not offer any guaranteed QoS, and it is affected by high interference, due to sharing the 2.4 GHz band, together with ZigBee, Bluetooth, and many others devices.

To meet the IoT requirements (large number of devices, large coverage range, and energy constrains), a IEEE802.11ah Task Group (TGah) (Low-Power Wifi) was formed in 2010 by the IEEE 802 LAN/MAN Standards Committee (LMSC). Its responsibilities were of extending the application area of Wifi networks.

Moreover, IEEE 802.11ah draft standard, that have been published in July 2014, introduced a novel hierarchical method to overcome the limited number of stations that can be simultaneously associated with the same access point (AP) of legacy IEEE802.11 [14] to facilitate covering hundreds, or even thousands of IoT devices which periodically transmit short packets. First performance studies [15] show that IEEE802.11ah will support a large set of M2M scenarios, such as agriculture monitoring, smart metering, industrial automation. It will be able to provide a quality of service (QoS) [15] level higher than currently provisioned in mobile networks, and enable scalable and cost-effective solutions.

III.4. *Low Power Wide Area (LPWA)*

The term LPWA which was introduced by Machina Research to the market, stands for high reach, low cost, low power Wide Area Networks [5]. The LPWA technology has recently emerged specifically focusing on low-end IoT applications which require low cost device, long battery life time, small amounts of data exchanged.

According to Machina Research, LPWA will allow to interconnect a large number of low-cost devices (up to 60% by 2022, with three billions LPWA M2M connections, by 2023), making the M2M solution business profitable, and providing a platform to build a large IoT business [5].

The key features of LPWA can be summarized as follow: (i) wide area coverage (up to some tenths of Km), (ii) low cost communication, (ii) long battery life (up to 10 years from a single AA battery), and (iv) low bandwidth communication. The latter limits the LPWA range of applications to a set of M2M use cases, characterized by low data rate, and infrequent transmissions (few hundred bytes of data)

From another point, the use of unlicensed spectrum for long-range communication considered as principal downside. This means that simple operations, like sending an acknowledgement, cannot be executed seamlessly as in 3GPP technologies. Consequently, only a limited set of IoT applications can be supported through this technology.

Additionally, due to an impeding spectrum congestion LPWA cannot fulfill the scalability requirements of large-scale IoT deployments [16]. With the explosion of devices connected through IoT, millions of devices may appear within the coverage area of a single LPWA base station. Many of those will be using other radio technologies that share the spectrum with LPWA (LP-Wifi, Z-Wave, Zigbee, etc.). All these transmissions will be perceived as interference by the LPWA device, having low receiver sensitivity for long-range communication. However, LPWA is expected to be a key enabler for IoT deployments for limited IoT applications.

III.5. 3GPP Cellular: MTC

Within the cellular context, the IoT connectivity solution is referred to as machine-to-machine (M2M) and within the 3GPP standardization body it is referred to as machine-type communications (MTC).

For certain reasons, such as robustness against single point of failures or ease of deployment, MTC is attractive when compared to wired solutions. The main challenge for wireless solutions is related to the cost of the radio, the power consumption but also the variety of M2M services.

Compared to the other IoT connectivity technologies, cellular MTC is able to offer quality of service (QoS) support, mobility and roaming support, as well as billing, security and global coverage. Moreover, MTC has the ability to connect the sensors/devices to the core enterprise systems, all in real-time, scalable and secure.

With the introduction on the NB-IoT standard (narrow-band IoT) into the 3GPP roadmap in June 2016 [17], each base station cell will have the ability to connect to 50000 or more devices at a very small data rate. This new standard will make sensor able to be connected, sending small packets of data at intermittent time instances.

Regardless of its advantages, some serious challenges remain to make MTC an underlying connectivity backbone for the Internet of Things.

V. Study of the state of the art

These days, the diffusion of IoT devices with its different communication technologies and the forecasting about its huge number in the future was coincided with a new concept which is called: "Big Data". This new concept in definition [18] is a term for data sets that are so large or complex that traditional data processing application software is inadequate to deal with them. Challenges include capture, storage, analysis, visualization, updating and information privacy. The term "Big Data" often refers to data analytics methods that extract value from data.

According to Cisco [19], by 2018 IoT will generate a staggering 400 zettabytes (ZB) of data a year. Moreover, by 2020 the total amount of data created (and not necessarily stored) by any device will reach 600 ZB per year [19].

This enormous number arises the need to have reliable cloud platforms to handle this big data. An IoT platform is a group of components that enable:

- Deployment of applications that monitor, manage, and control connected IoT devices
- Remote data collection from connected devices
- Independent and secure connectivity between devices
- Device/sensor management

To accommodate this data leap, network layer should be more intelligent, more powerful, more efficient, more secure, more reliable, and more scalable to meet the requirements of the characteristics of diversity and dynamics. Software Defined Networking (SDN) [20] and Network Functions Virtualization (NFV) [21] are two promising technologies that could help to address these challenges and leverage the IoT architecture in the Cloud era.

SDN is a novel networking paradigm separating the system that makes decisions about where traffic is sent (the control plane) from the underlying system that forwards traffic to the selected destination (the data plane). In traditional routers and switches, the control and data planes are in one device.

Network function virtualization (NFV) [21] is a network architecture concept that proposes using IT virtualization related technologies to virtualize entire network functions into building blocks that may be connected, or chained, to create communication services. A virtualized network function may consist of one or more virtual machines running different applications and processes in network servers, switches, storage, or even cloud computing infrastructure, instead of having custom hardware appliances for each network function.

The emerging of IoT, SDN, and NFV (Fig. 3) to build a platform has great potential for the information service innovation in the Cloud and big data era. The decoupling of the control plane from the data plane in SDN architecture provides IoT users with a centralized system view and control over IoT. On the other hand, the design of efficient SDN-based IoT architecture with NFV implementation is a big technical challenge.

We tried in this research to create a platform for data analytics for IoT network devices. In another words, we tried to build an analytic platform has interoperability with any IoT network devices with different technologies: IoT gateways, SDN and NFV in order to have a statistical and analytical overview about those network devices.

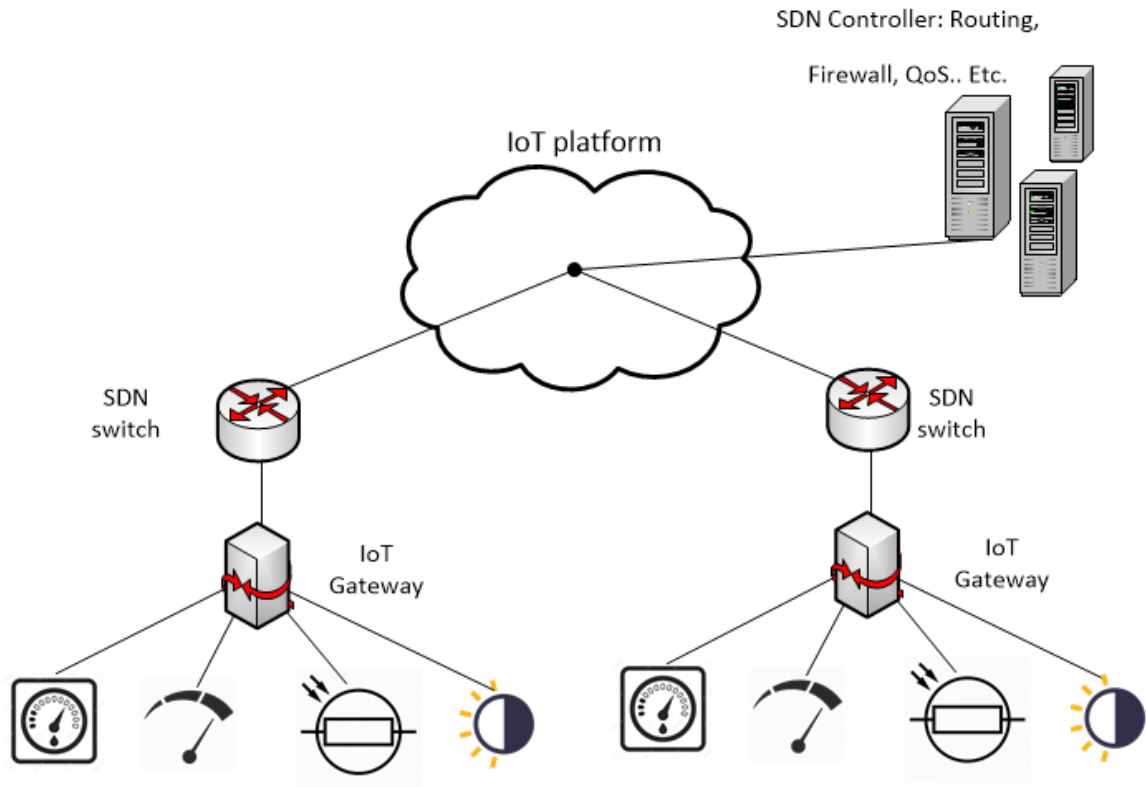


Fig.3 Network Architecture for IoT platform using SDN and NFV Technologies

To create this platform, we relied on cloud computing using a virtual machine for a bunch of application: Docker, Logstash, Elasticsearch and Kibana. We will explain in the next paragraphs each of them and how we build through them our data analytics platform.

VI. Data analytic platform components

VI.1. Docker docker

Cloud computing is inherently rooted in virtualization technologies. Recently, new lightweight virtualization technologies such as containers have become increasingly popular and are nowadays an essential part of cloud offerings. Containers also tightly integrate into the host operating system, reducing the software overhead imposed by virtual machines (VMs) [22].

However, several factors including acceleration of the development cycle (such as agile methods and DevOps), an increasingly complex application stack (mostly Web services and their frameworks), and market pressure to densify applications on servers have triggered the need for a fast, easy to use way of pushing code into production.

Container provides near-bare-metal performance [22] and offers the possibility of seamlessly running multiple versions of applications on the same machine (see Fig.4). New instances of containers can be created instantly to face a customer demand peak, which is convenient for spawning applications on-demand or quickly moving a service, such as when implementing network function virtualization (NFV).

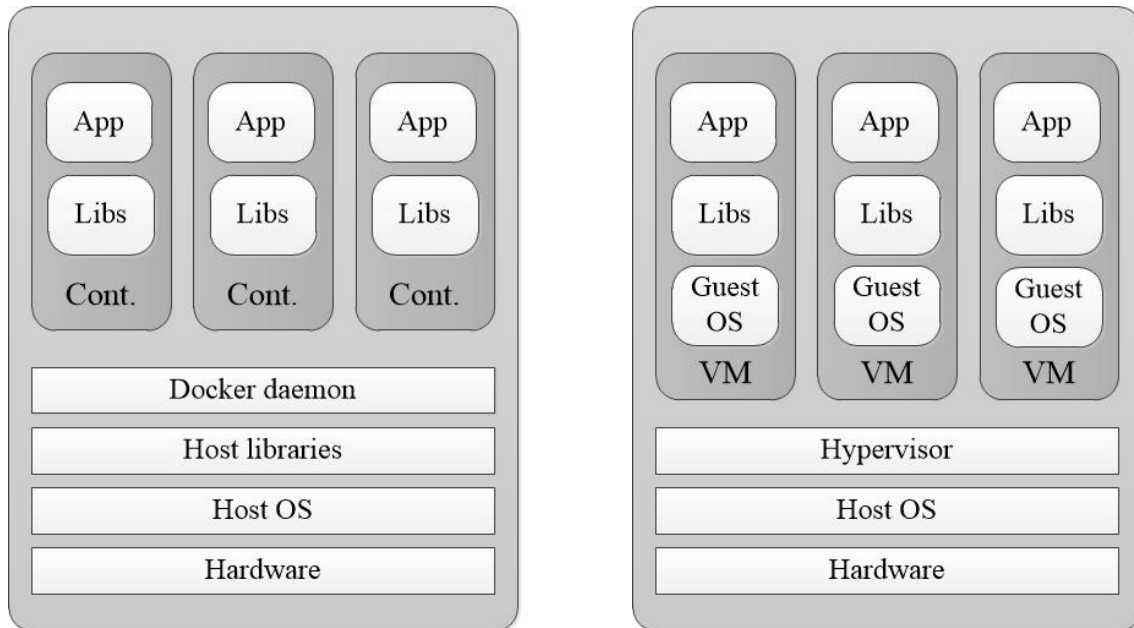


Fig.4 Compare application runtime model for Docker host with VM host

Containers can be integrated in a multitenant environment, thus profiting from resource sharing to increase average hardware use. This is achieved by sharing the kernel with the host machine. Unlike VMs, containers do not embed their own kernel, but rather run directly on the host kernel. This shortens the syscalls execution path by removing the guest kernel and the virtual hardware layer.

Additionally, containers can share software resources (such as libraries) with the host, hence avoiding code duplication. The absence of kernel and some system libraries make containers very lightweight (image sizes can shrink to a few megabytes), which enables a quick boot process. In our project, we will use Docker as the base to build the analytics platform with different containers. Fig.5 shows the lifecycle of Docker with its all component that they be explained in the next paragraphs.

VI.1.1.1. *Dockerfile*

Dockerfiles let users specify a base image and a sequence of commands to be performed to build the image, along with other options such as exposed ports specific to the image. The image is then built with the `docker build` command automatically by reading the instructions from a Dockerfile.

Dockerfiles begin with defining an image `FROM` which the build process starts. Followed by various other methods, commands and arguments (or conditions), in return, provide a new image which is to be used for creating Docker containers.

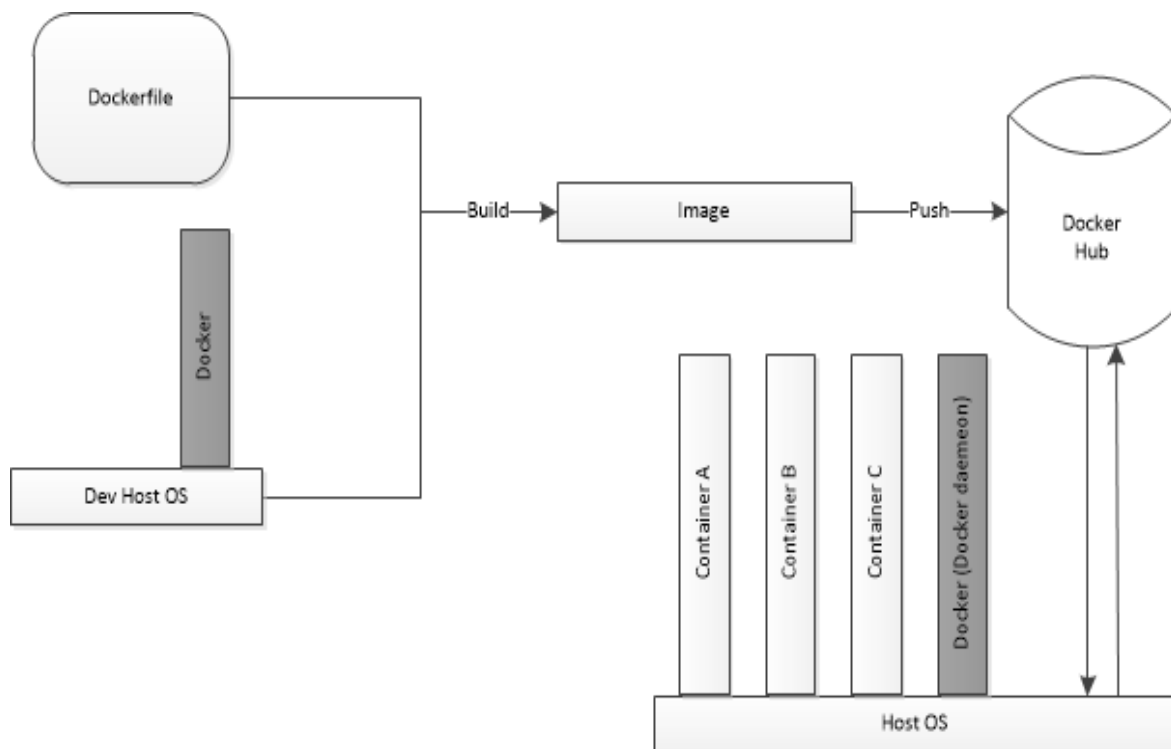


Fig.5 Docker lifecycle

In the example (Fig.6), we are building an image that will run the nginx proxy server. After the standard FROM, MAINTAINER is non-executing command declares the author, hence setting the author field of the images. It should come nonetheless after FROM.

```
FROM ubuntu
MAINTAINER Abdullah Kassabji (email@domain.com)
RUN apt-get update
RUN apt-get install -y nginx
EXPOSE 80
```

Fig.6 Example of a Dockerfile

A RUN instruction is used to execute any commands. In this case we are running a package update and then installing nginx. We are using the EXPOSE command here to inform what port the container will be listening on. The full list of Dockerfile commands can be found on Docker website [23].

VI.1.2. Docker image

An image is an inert, immutable, read-only template file that's essentially a snapshot of a container and they are created with the `docker build` command for a dockerfile. Because they can become quite large, images are designed to be composed of layers of other images, allowing a minimal amount of data to be sent when transferring images over the network. Docker images are the build component of Docker.

The command `docker images` shows local images (Fig.7). Each image has `IMAGE ID` that is the first 12 characters of the true identifier.

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
ubuntu	13.10	5e019ab7bf6d	2 months ago	180 MB
ubuntu	14.04	99ec81b80c55	2 months ago	266 MB
ubuntu	latest	99ec81b80c55	2 months ago	266 MB
ubuntu	trusty	99ec81b80c55	2 months ago	266 MB

Fig.7 Docker images command result

VI.1.3. Docker container

An instance of an image is called a container. In another words, if you start an image, you have a running container of this image. Docker containers are similar to a directory they hold everything that is needed for an application to run. Each container is created from a Docker image.

Docker containers can be run, started, stopped, moved, and deleted. Each container is an isolated and secure application platform. Docker containers are the run component of Docker.

One of the strength points of Docker is the possibility to run many isolated containers from the same image. The command `docker ps` only outputs running containers. To view all containers (running or stopped) with `docker ps -a`.

VI.1.4. Docker daemon

The Docker software runs as a daemon on the host machine. It can launch containers, control their isolation level, monitor them to trigger actions (such as restart), and spawn shells into running containers for administration purposes.

The software can change IP tables rules on the host and create network interfaces. It's also responsible for managing container images, including pulling and pushing images on a remote registry (such as the Docker hub), building images from Dockerfiles, and signing images. The daemon itself runs as root (with full capabilities) on the host and is remotely controlled through a Unix socket. Alternatively, the daemon can listen on a classical TCP socket.

VI.1.5. Docker hub (Registry)

The Docker hub online repository lets developers upload their Docker images and lets users download them. Developers can sign up for a free account, in which all repositories are public, or for a paid account, which lets them create private repositories.

Developer repositories are namespaced that is, their name is “developer/repository.” Official repositories also exist, directly provided by Docker Inc.; these are called “repository.” The Docker daemon, hub, and repositories are similar to a package manager, with a local daemon installing

software on both the host and the remote repositories. Some of these repositories are official, while others are unofficial and provided by third parties.

VI.2. *Logstash* logstash

Logstash is a light-weight, open-source, server-side data processing pipeline that allows to collect data from a wide variety of sources, transform it on the fly, and send it to the desired destination (Fig.8). Logstash is a popular choice because of its tight integration, powerful log processing capabilities, and over 200 pre-built open-source plugins that can help to get your data indexed in the desired way.

Collection is accomplished via configurable input plugins including raw socket/packet communication, file tailing, and several message bus clients. Once an input plugin has collected data it can be processed by any number of filters which modify and annotate the event data.

Logstash routes events to output plugins which can forward the events to a variety of external programs including Elasticsearch, local files and several message bus implementations.

VI.2.1. *Logstash Inputs*

One of the things that makes Logstash special and powerful is its ability to source logs and events from various sources. On the Logstash documentation page [24] there are more than 50 different inputs with different technologies, locations, and services. These include monitoring systems, databases, services, and various others. By using these inputs, data can be imported from multiple sources and manipulate them as the user wants and eventually send them to other systems for storage or processing.

Inputs are the starting point of any configuration. If there is no definition for an input, Logstash will automatically create a standard input. Since it could be multiple inputs, it is important to type and tag them.

VI.2.2. *Logstash Outputs*

Like the inputs, Logstash comes with a number of outputs [25] that enable to push your events to various locations, services, and technologies. It is possible to store events using outputs such as File and CSV, convert them into messages, or send them to various services like HipChat, PagerDuty, or IRC. The number of combinations of inputs and outputs in Logstash makes it a really versatile event transformer.

Logstash events can come from multiple sources, so as with filters, it's important to do checks on whether or not an event should be processed by a particular output. If there is no definition for an output, Logstash will automatically create a standard output.

VI.2.3. Logstash Filters

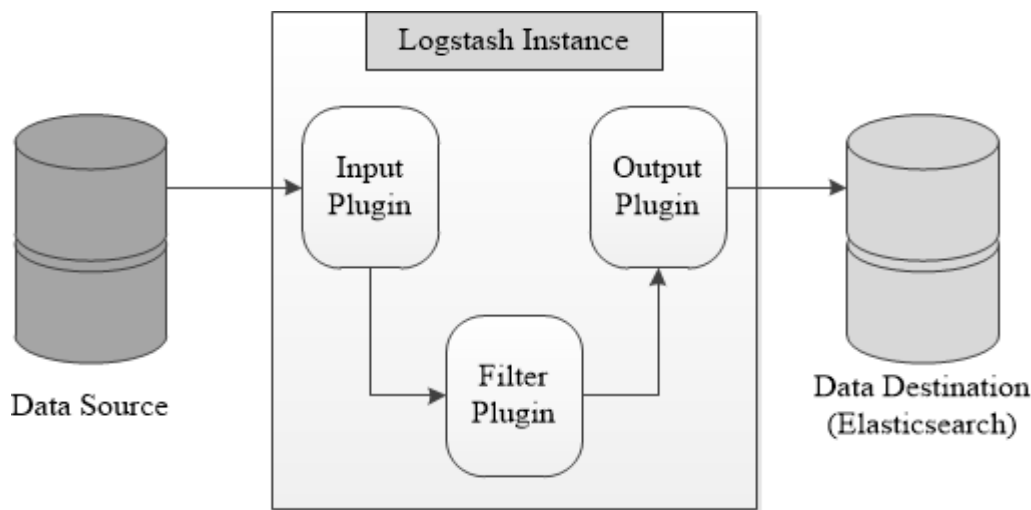


Fig.8 Logstash architecture

A filter performs intermediary processing on an event. Filters are often applied conditionally depending on the characteristics of the event. Logstash has a number of very powerful filters with which you can manipulate, measure, and create events. It's the power of these filters that makes Logstash a very versatile and valuable tool.

Logstash events can come from multiple sources, so as with outputs, it's important to do checks on whether or not an event should be processed by a particular filter. Logstash will be in our platform the data receiver for this platform from network devices of IoT platform.

VI.3. *Elasticsearch* elasticsearch

Elasticsearch is an open-source, broadly-distributable, readily-scalable, enterprise-grade near real time search platform built to handle huge amounts of data volume with very high availability and to distribute itself across many machines to be fault-tolerant and scalable, all the while maintaining a simple but powerful API (Application Programming Interface) that allows applications from any language (including such as Java, Python, .NET, and Groovy) or framework access to the database.

Elasticsearch is horizontally-scalable (See Fig.9) built on APACHE'S LUCENE that delivers a full-featured search experience across terabytes of data with a simple yet powerful API.

Many companies use elasticsearch to help them deploy powerful search capabilities in their applications that are easy to set up, scalable and built for the cloud. Here are a couple of sample use-cases that Elasticsearch could be used for:

- An online web store that allows customers to search for products that this store sell. In this case, Elasticsearch can store the entire product catalog and inventory and provide search and autocomplete suggestions for customers.
- Collecting log or transaction data to analyze and mine this data to look for trends, statistics, summarizations, or anomalies. In this case, Logstash can be used to collect, aggregate, and parse

the data, and then have Logstash feed this data into Elasticsearch. Once the data is in Elasticsearch, searches and aggregations can be run to mine any information that is could be interested.

VI.3.1. *Elasticsearch basic features*

- REST API (Representational state transfer) API: Elasticsearch stores/retrieves objects via a REST API such as PUT, POST, GET, and DELETE APIs. This is what makes it a key value store [26].
- Key Value Store: In Elasticsearch, every piece of data has a defined index and type.
- Multi-tenancy: It can be easily creating, updating, retrieving and deleting indices. That means Elasticsearch is multi-tenant and quite flexible.
- Mapping: Elasticsearch indexes documents (A document is a basic unit of information that can be indexed) is stored by using either a dynamic mapping, or a mapping provided by user (recommended).
- Clusters: A cluster is a collection of one or more nodes (servers) that together holds the entire data and provides federated indexing and search capabilities across all nodes.

```
{
  "id": 1001,
  "author": {
    "name": "Alexander Farah",
    "id": 3874
  },
  "date": "2015-10-07 12:31:00 -0600 CST",
  "title": "The Federalist Papers",
  "subtitle": "Paper #1"
  "text": "AFTER an unequivocal experience of the inefficiency..."
  "similar_posts": [ 1002, 1003, 1005]
  "comments": [
    {"author": "John Adams", "text": "I must beg to differ..."}
  ]
}
```

Fig.9 An example how a blog post can be store in Elasticsearch

- Sharding & Replication: An index can potentially store a large amount of data that can exceed the hardware limits of a single node. For example, a single index of a billion documents taking up 1TB of disk space may not fit on the disk of a single node or may be too slow to serve search requests from a single node alone. To solve this problem, Elasticsearch provides the ability to subdivide the index into multiple pieces called shards. Moreover, Elasticsearch allows to make one or more copies of index's shards into what are called replica shards, or replicas for short. Replication is important for better availability and performance in case a shard/node fails.

VI.4. Kibana kibana

Kibana is an open source analytics and visualization platform designed to work with Elasticsearch to provide visualization capabilities on top of the content indexed on an Elasticsearch cluster. It makes huge and complex data streams more quickly and easily understandable through graphic representation in real time.

Kibana can generate PDF reports either on demand or on schedule and provides a flexible, dynamic dashboard. Generated reports can represent the data in bar, line, scatter plots or pie chart graph formats with customizable colors and highlighted search results. Kibana also includes sharing tools for visualized data.

VII. ELK Platform: architecture & setup

VII.1. Platform architecture

Fig. 10 shows the simple architecture of the analytics platform. This platform is built in Docker and it is composed from Elasticsearch, Logstash and Kibana (ELK stack). This platform works in the following order:

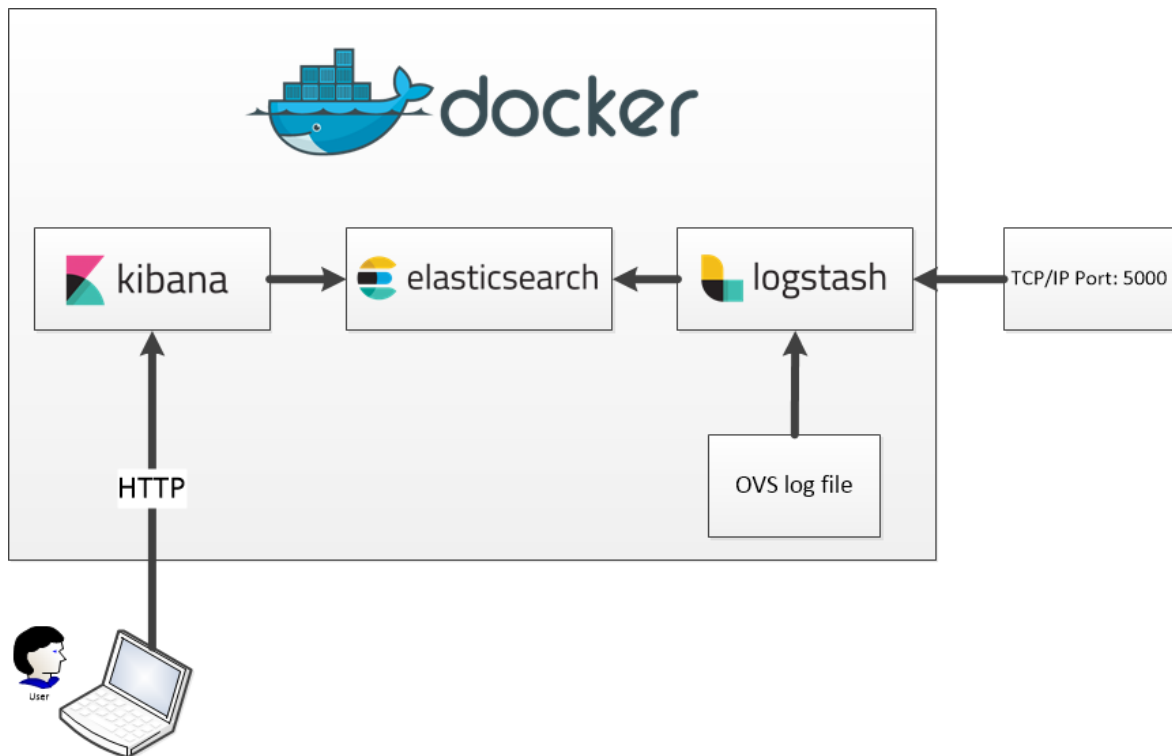


Fig. 10 Overview of the data analytics platform with a use case

Step 1: Logstash receives data from countless IoT network sources. For a use case, we have chosen two kind of sources: (i) external IoT network device is connected through TCP/IP protocol (Port 5000) (ii) internal source from the same device that has Docker. For the internal resource, we used log file of Openvswitch [27] that is connected to multiple IoT network devices (Fig.11).

Openvswitch is a multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols (e.g. NetFlow, sFlow, IPFIX, RSPAN, CLI, LACP, 802.1ag). In addition, it is designed to support distribution across multiple physical servers similar to VMware's vNetwork distributed vswitch or Cisco's Nexus 1000V. Since Openvswitch is a programmable switch, it is used in several IoT platforms and that is why we have chosen it in our use case.

Step 2: The Logstash after receiving data and process the different queues to store the data in the local folders.

Step 3: Elasticsearch performs different operations (e.g. optimize the data structures, create search indexes, group information).

```
2017-08-03T16:15:02.866Z|00021|vlog|INFO|opened log file /var/log/openvswitch/ovs-
vswitchd.log
2017-08-04T16:55:44.138Z|00013|bridge|INFO|bridge br0: added interface br0 on port 65534
2017-08-07T17:00:49.172Z|00006|ofproto_dpif|INFO|system@ovs-system: Datapath supports
recirculation
2017-08-07T17:00:59.124Z|00018|memory|INFO|50132 kB peak resident set size after 10.0
seconds
```

Fig. 11 A sample of Openvswitch log

Step 4: Kibana reads Logstash data structures and read data from Elasticsearch to present them to the users using custom layouts, dashboards and filters.

VII.2. Platform setup

The next steps were performed on Ubuntu 16.04 installed on it Openvswitch 2.5.2.

VII.2.1. Install Docker

It should be noted that Docker requires a 64-bit version of Ubuntu as well as a kernel version equal to or greater than 3.10. The default 64-bit Ubuntu 16.04 meets these requirements. Docker is available in two editions: Community Edition (CE) and Enterprise Edition (EE). We will use Docker Community Edition (CE) because it is oriented for developers and small teams.

First we will show how to get the latest version of Docker from the official Docker repository because the Docker installation package available in the official Ubuntu 16.04 repository may not be the latest version.

First, we add the GPG key for the official Docker repository to the system:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
apt-key add -
```

Then, we add the Docker repository to APT sources:

```
$ sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs)
stable"
```

Next step is update the package database with the Docker packages from the newly added repository:

```
$ sudo apt-get update
```

Then we can install from the Docker repo instead of the default Ubuntu 16.04 repository using:

```
$ apt-cache policy docker-ce
```

As a result, we should have in the end:

```
docker-ce:
  Installed: (none)
  Candidate: 17.06.0~ce-0~ubuntu-xenial
  Version table:
 17.06.0~ce-0~ubuntu-xenial 500
 500 https://download.docker.com/linux/ubuntu xenial/stable
amd64 Packages
 17.06.0~ce-0~ubuntu-xenial 500
 500 https://download.docker.com/linux/ubuntu xenial/stable
amd64 Packages
```

Finally, install Docker:

```
$ sudo apt-get install -y docker-ce
```

Docker now is installed, the daemon is started, and the process is enabled to start on boot. Next command is to check that:

```
$ sudo systemctl status docker

docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled;
vendor preset: enabled)
   Active: active (running) since lun 2017-06-14 17:53:56 CEST;
3min 9s ago
     Docs: https://docs.docker.com
```

VII.2.2. *Install Logstash, Elasticsearch, Kibana*

We will use Compose command to install them and run the platform. Compose in Docker is a tool for defining and running multi-container Docker applications. With Compose, user uses a Compose file (`docker-compose.yml`) to configure application's services. Then, using a

single command, user can create and start all the services from the configuration file. In general, Compose is basically a three-step process:

1. Define the app's environment with a Dockerfile so it can be reproduced anywhere.
2. Define the services that make up the app in `docker-compose.yml` so they can be run together in an isolated environment.
3. Run `docker-compose up` and Compose will start and run the entire app.

The aforementioned steps have been followed to create ELK platform using Compose command. It is essential to create a directory (`/home/docker-elk/`) contains three folders: `logstash`, `elasticsearch` and `kibana` in addition to the `docker-compose.yml` (See Fig.13). All the contents of configuration files and `docker-compose.yml` can be found in Annex A.

In Dockerfiles we used the official Docker images for each of `logstash`, `elasticsearch` and `kibana` to be build. In `docker-compose.yml` services start with `elasticsearch` container to be built by reading the `dockerfile` of it and open two ports: 9200 and 9300 with mounting the configuration file that can be modified by user (`elasticsearch.yml`) to be executed. At the end we add this container to a network (`elk`) that will connect all containers together.

After `elasticsearch`, `logstash` container will be built with the same manner by reading the `dockerfile` inside `logstash`'s folder and mounting the configuration file (`logstash.yml`). Sources that can send data to `logstash` are called pipelines and they are two in our case (Fig.10).

`logstash.conf` should be mounted to address those sources. Moreover, we used a `grok` filter for `logstash` to parse the `Openvswitch` log file (Annex A). At the end, `logstash` container is added to the same network (`elk`).

The last container to be build is `Kibana` after reading (`kibana.yml`) configuration file. Port 5601 is mounted to let users to connect to `Kibana` using that port. Also `kibana` container is add to the same network (`elk`).

Later after some experiments, we have found that `Openvswitch` log folder that contains the logs files cannot be mounted to `logstash` container. After analyzing the problem, we discovered that the log folder (`/var/log/openvswitch`) is mounted but is not read by `logstash` because `logstash` does not have read permission.

To solve this problem, execute file "`start.sh`" has been created in (`/home/docker-elk`) to change the permission of `openvswitch` log files to be read by anyone before Compose command starts (Fig.12), then `logstash` reads and analyses the `openvswitch` log file.

```
sudo chmod a+r /var/log/openvswitch/ovs-vswitchd.log.*
sudo docker-compose up
```

Fig.12 `Start.sh` contents

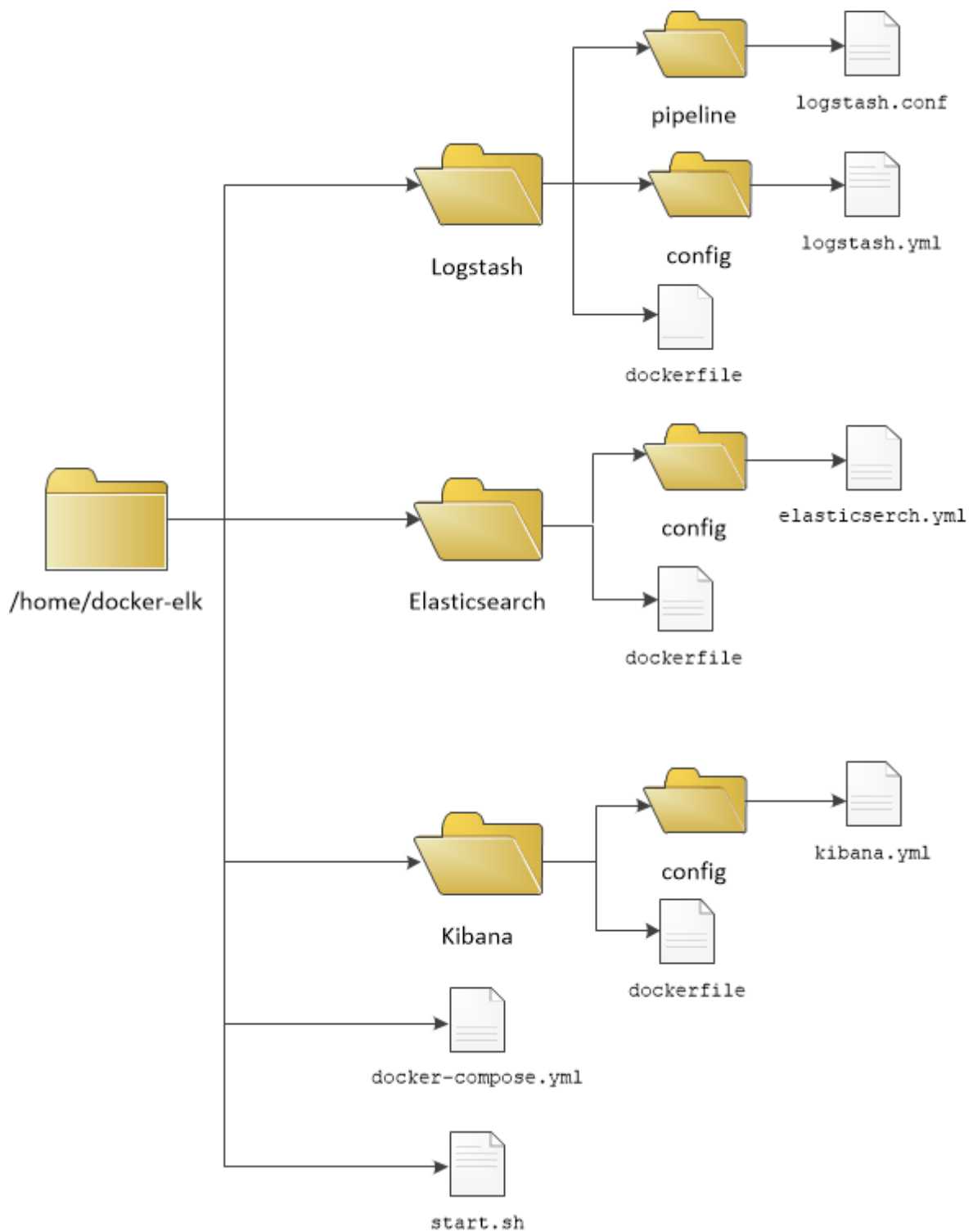


Fig.13 Folders architecture of ELK platform

VIII. Run ELK platform

After installing Docker and ELK stack, we start to run ELK platform by open the terminal and choose the ELK directory (/home/docker-elk/ :

```
$ cd docker-elk
```

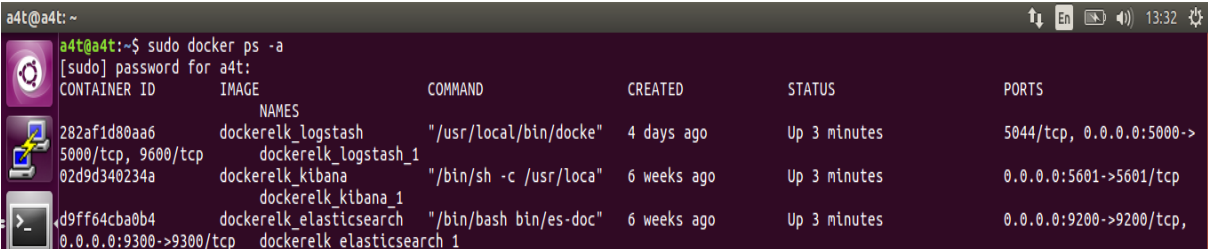

Then we run the execute file “start.sh” that changes the permission and run compose command which reads and execute compose file (docker-compose.yml) :

```
$ ./start.sh
```

First, elasticsearch container will be built, then logstash container and lastly kibana container, the same order in the compose file (docker-compose.yml) see Annex B. To show the running container, we open a new terminal window and use the command:

```
$ sudo docker ps -a
```

We will have (See Fig.14) three running containers: dockernelk_logstash_1, dockernelk_kibana_1 and dockernelk_elasticsearch_1 with the corresponding ports.



```
a4t@a4t:~$ sudo docker ps -a
[sudo] password for a4t:
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
282af1d80aa6      dockernelk_logstash  "/usr/local/bin/docke"  4 days ago         Up 3 minutes       5044/tcp, 0.0.0.0:5000->
5000/tcp, 9600/tcp  dockernelk_logstash_1
02d9d340234a      dockernelk_kibana    "/bin/sh -c /usr/loca"  6 weeks ago        Up 3 minutes       0.0.0.0:5601->5601/tcp
                    dockernelk_kibana_1
d9ff64cba0b4      dockernelk_elasticsearch  "/bin/bash bin/es-doc"  6 weeks ago        Up 3 minutes       0.0.0.0:9200->9200/tcp,
0.0.0.0:9300->9300/tcp dockernelk_elasticsearch_1
```

Fig.14 running containers

Elasticsearch information like version and build date can be shown by access elasticsearch through web browser and access TCP port 9200: <http://localhost:9200> we will have:

```
{
  "name" : "D9cIo0f",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "tMeIXZeGRLy3iPhB99Sq3Q",
  "version" : {
    "number" : "5.4.3",
    "build_hash" : "eed30a8",
    "build_date" : "2017-06-29T00:34:03.743Z",
    "build_snapshot" : false,
    "lucene_version" : "6.5.1"
  },
  "tagline" : "You Know, for Search"
}
```

IX. Results after running ELK platform

Kibana is the container of ELK that is responsible to display the analytics data. User can connect to Kibana through HTTP protocol. For ELK platform we mounted the default port 5601 for Kibana. To open Kibana : <http://localhost:5601> and if everything was setting up correctly we will have as it can be seen in Fig. 15. We confirm the default settings “logstash-*” index and press create button to create the index pattern.

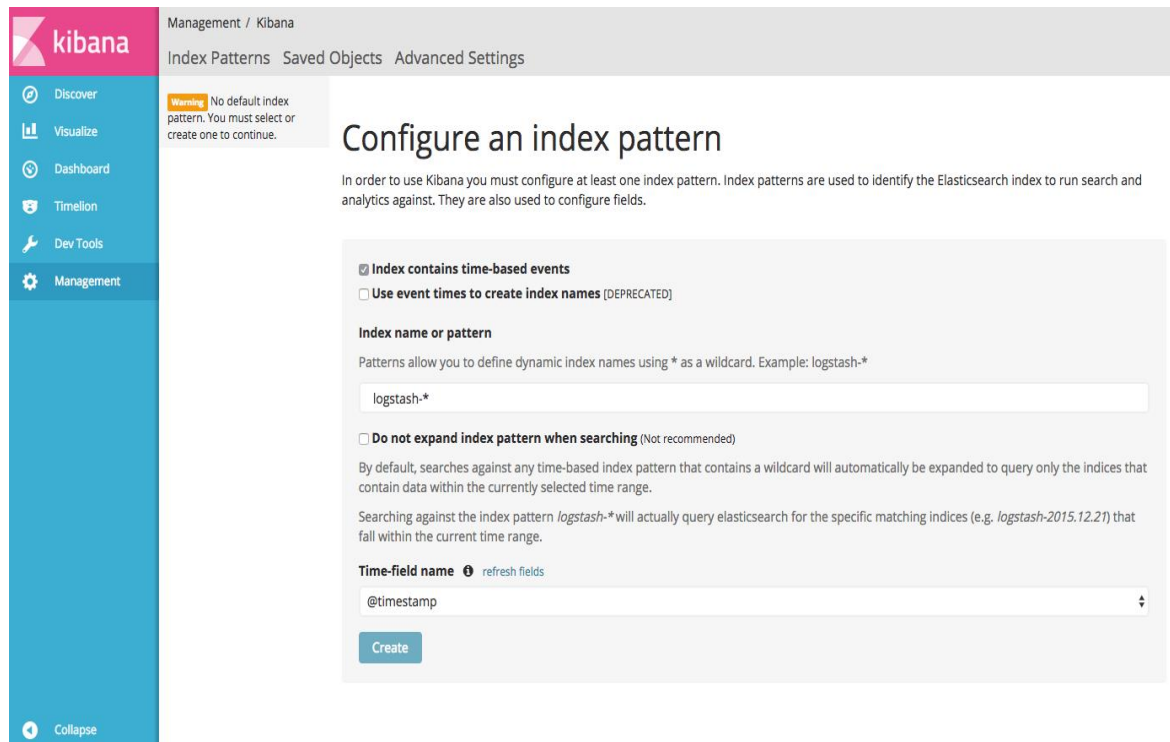


Fig.15 Kibana after the first access

Fig. 16 is shown Kibana after the first access. In the upper section (number 1) there is a search bar underneath it there is the timeline of the received data with graphical presentation, in Fig. 16 it is shown the received messages in the last five minutes. Time frame can be changed as the user wishes.

In the lower part of the page (number 2) there are the messages that have been received by Logstash and indexed by Elasticsearch. Classifications of each received messages are based on Logstash filters.

The gray column on the left (number 3) there are all the classifications or the tags that have been collected from the data received based on Logstash filters.

As we have chosen two kinds data resources internal and external that are receiving data from IoT network devices, we have tagged the received data with its source to use the analytic feature of Kibana to differentiate between them. The IoT data received through TCP protocol with port 5000 it has been tagged with “external”, Internal data of the log of Openvswitch has been tagged with “openvswitch”. It is shown also some failures for grok filter that parse the Openvswitch log.

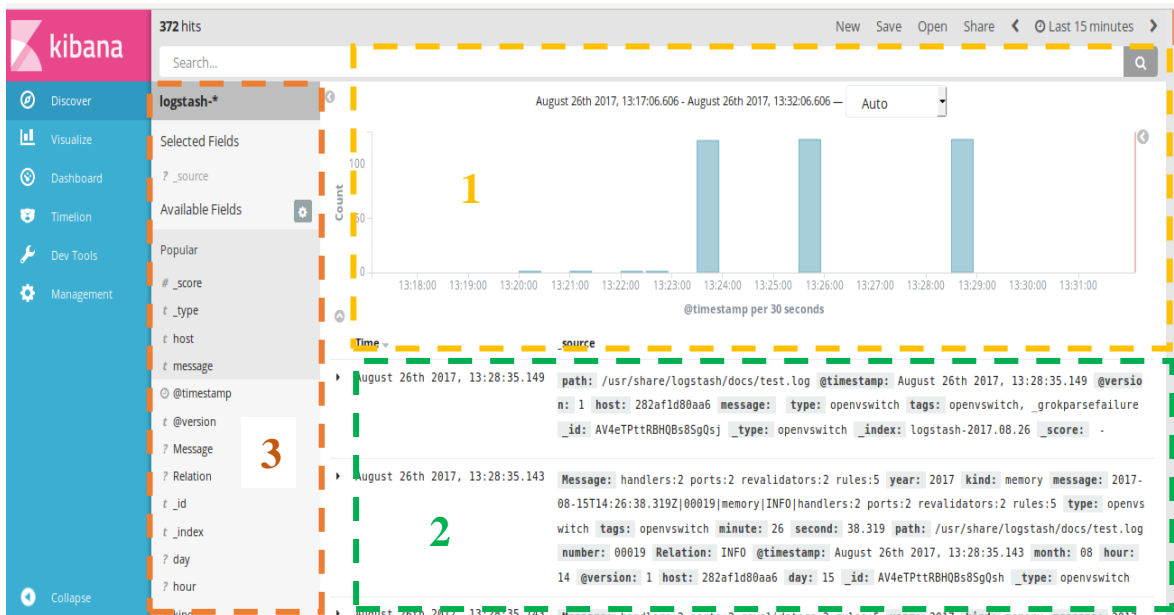


Fig.16 Kibana main interface

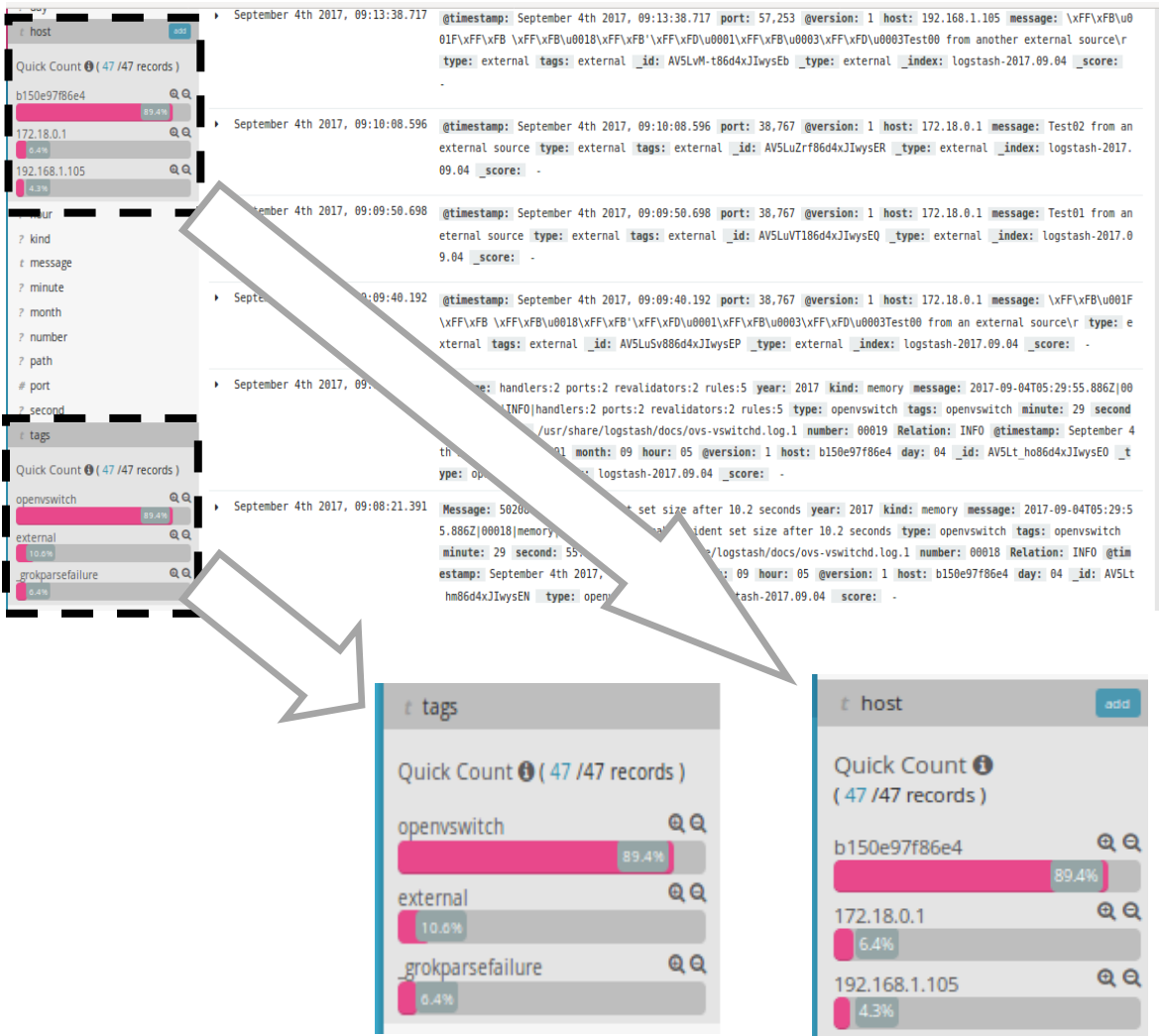


Fig.17 tag field and host field

Fig.17 shows tags field with its statistics of the source type and host field with its statistics of the received data from the virtual machine host (Openvswitch) and two hosts from different networks (TCP) with the total data records.

By choosing only one field that is tags field for example, Kibana will display only the received data with tags field based on the time frame that has been defined.

User can add or remove one or more field from the indexed data to display the correspondent messages. For example, we have chosen tags and hosts fields then Kibana will display the related messages based on the time frame that has been chosen. (Fig. 18).

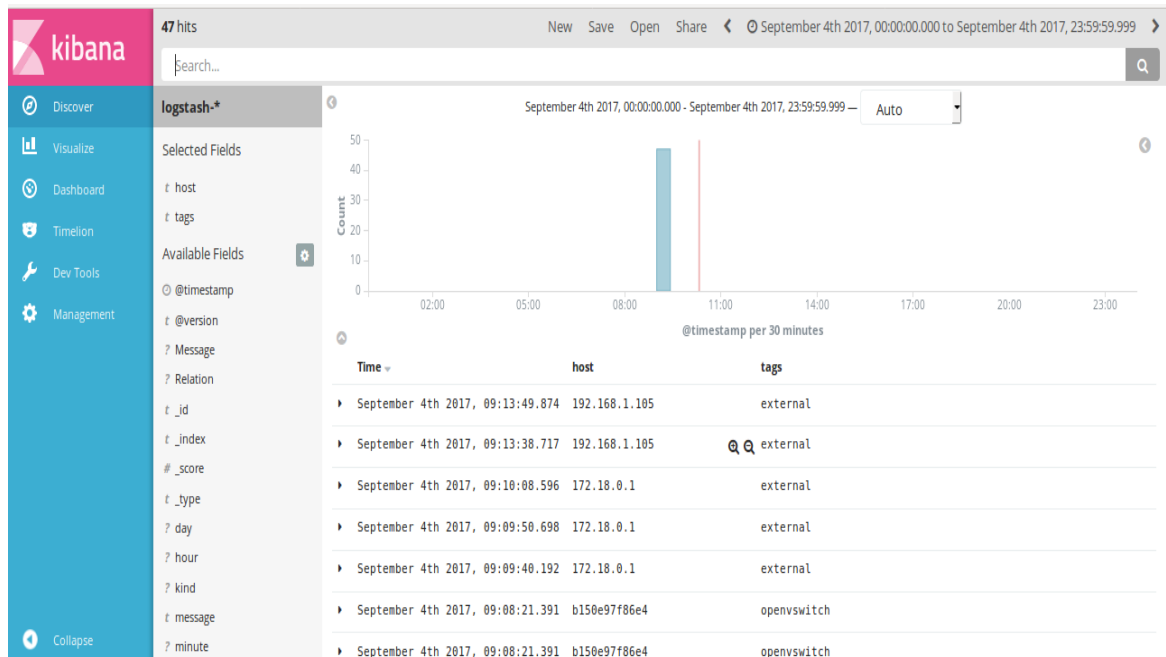


Fig.18 Results after choosing host and tags fields

Those results are just showing the chosen fields and hiding other fields. Pressing the small arrow on the left of each message will unhide the rest fields for this particular message (Fig. 19).

User can also search for a word in all received messages by using the search bar on the top of Kibana with or without choosing any fields. Only received messages that contains this word will be displayed in the results and the searched word will be highlighted (Fig. 19).

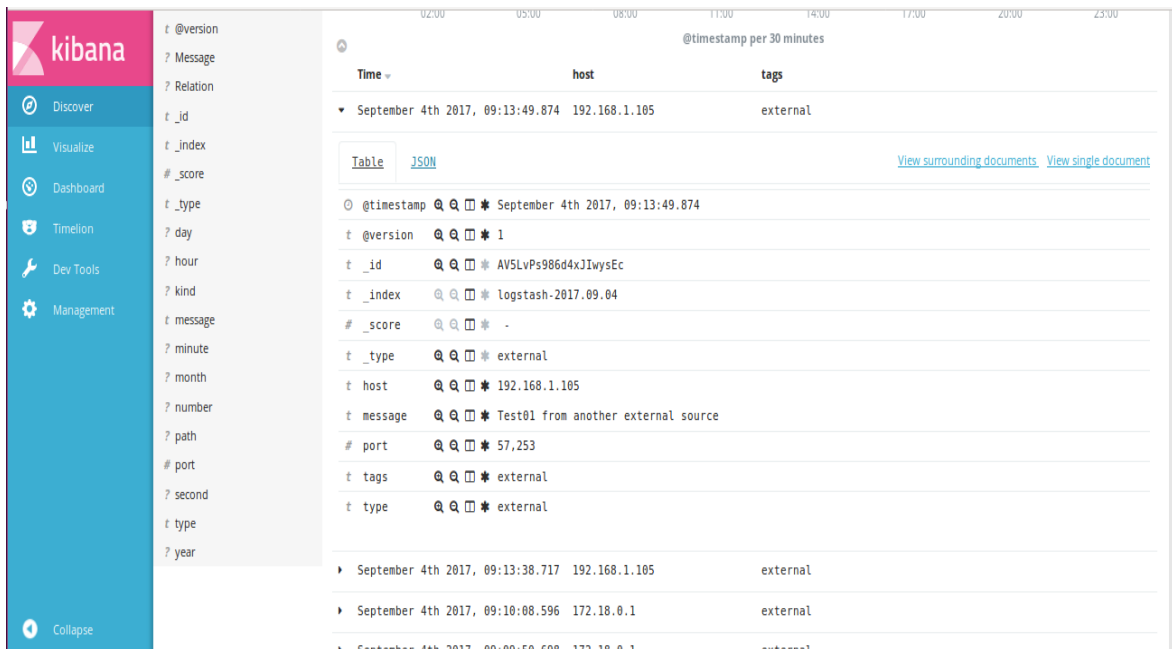


Fig.18 Full details for a message



Fig.19 Messages that contain the searched word “port”

Acknowledgment

I would like to thank Prof. Carlos Palau for his expert advice and encouragement throughout this project. Also I would like to thank my colleagues Jara and Eneko for their wonderful collaboration. In the end, great thank to my soulmate Carol and my family for their infinite support during the hard times.

Annex A. Contains of docker-compose.yml and configuration files of Logstash, Elasticsearch and Kibana

```
version: '2'

services:

  elasticsearch:
    build: elasticsearch/
    volumes:
      -
    ./elasticsearch/config/elasticsearch.yml:/usr/share/elasticsearch/con
    fig/elasticsearch.yml
    ports:
      - "9200:9200"
      - "9300:9300"
    environment:
      ES_JAVA_OPTS: "-Xmx256m -Xms256m"
    networks:
      - elk

  logstash:
    build: logstash/
    volumes:
      -
    ./logstash/config/logstash.yml:/usr/share/logstash/config/logstash.y
    ml
      - ./logstash/pipeline:/usr/share/logstash/pipeline
      - /var/log/openvswitch:/usr/share/logstash/docs
    ports:
      - "5000:5000"
    environment:
      LS_JAVA_OPTS: "-Xmx256m -Xms256m"
    networks:
      - elk
    depends_on:
      - elasticsearch

  kibana:
    build: kibana/
    volumes:
      - ./kibana/config:/usr/share/kibana/config
    ports:
      - "5601:5601"
    networks:
      - elk
    depends_on:
      - elasticsearch

networks:

  elk:
    driver: bridge
```

Fig A.1. contents of /docker-elk/docker-compose.yml

Dockerfile	Content
logstash	# https://github.com/elastic/logstash-docker FROM docker.elastic.co/logstash/logstash:5.4.3
elastic search	# https://github.com/elastic/elasticsearch-docker FROM docker.elastic.co/elasticsearch/elasticsearch:5.4.3
kibana	# https://github.com/elastic/kibana-docker FROM docker.elastic.co/kibana/kibana:5.4.3

Fig A.2. contents of dockerfiles of logstash, elasticsearch, kibana

```

input {
  tcp {
    port => 5000
    tags => "external"
    type => "external"
  }

  file {
    path => "/usr/share/logstash/docs/ovs-vswitchd.log.1"
    tags => "openvswitch"
    type => "openvswitch"
    start_position => "beginning"
  }
}

filter {
  if [type] == "openvswitch"
  {
    grok {
      match => {"message" => "%{YEAR:year}-%{MONTHNUM:month}-
%{MONTHDAY:day}T%{HOUR:hour}:%{MINUTE:minute}:%{SECOND:second}Z\|%
{WORD:number}\|%{WORD:kind}\|%{WORD:Relation}\|%{GREEDYDATA:Message}"}
    }
  }
}

## Add your filters / logstash plugins configuration here

output {
  elasticsearch {
    hosts => "elasticsearch:9200"
  }
}

```

Fig A.3. contents of /docker-elk/logstash/pipeline/logstash.conf

```
## Default Logstash configuration from logstash-docker.
## from https://github.com/elastic/logstash-
docker/blob/master/build/logstash/config/logstash.yml
#
http.host: "0.0.0.0"
path.config: /usr/share/logstash/pipeline

## Disable X-Pack
## see https://www.elastic.co/guide/en/x-pack/current/xpack-
settings.html
## https://www.elastic.co/guide/en/x-pack/current/installing-
xpack.html#xpack-enabling
xpack.monitoring.enabled: false
```

Fig A.4. contents of /docker-elk/logstash/config/logstash.yml

```
## Default Kibana configuration from kibana-docker.
## from https://github.com/elastic/kibana-
docker/blob/master/build/kibana/config/kibana.yml
server.name: kibana
server.host: "0"
elasticsearch.url: http://elasticsearch:9200

## Disable X-Pack
## see https://www.elastic.co/guide/en/x-pack/current/xpack-
settings.html
## https://www.elastic.co/guide/en/x-pack/current/installing-
xpack.html#xpack-enabling
#
xpack.security.enabled: false
xpack.monitoring.enabled: false
xpack.ml.enabled: false
xpack.graph.enabled: false
xpack.reporting.enabled: false
```

Fig A.5. contents of /docker-elk/kibana/config/kibana.yml

Annex B. Screenshots from terminal after running ELK stack

```

a4t@a4t: ~/docker-elk
a4t@a4t:~$ cd docker-elk/
a4t@a4t:~/docker-elk$ sudo docker-compose up
[sudo] password for a4t:
Starting dockereik_elasticsearch_1
Starting dockereik_kibana_1
Starting dockereik_logstash_1
Attaching to dockereik_elasticsearch_1, dockereik_kibana_1, dockereik_logstash_1
elasticsearch_1 | [2017-08-21T11:24:06,123][INFO] [o.e.n.Node] [] initializing ...
elasticsearch_1 | [2017-08-21T11:24:06,726][INFO] [o.e.e.NodeEnvironment] [D9cIo0f] using [1] data paths, mounts [/(none)], net usable
space [18.1gb], net total_space [33.8gb], spins? [possibly], types [aufs]
elasticsearch_1 | [2017-08-21T11:24:06,727][INFO] [o.e.e.NodeEnvironment] [D9cIo0f] heap size [247.5mb], compressed ordinary object point
ers [true]
elasticsearch_1 | [2017-08-21T11:24:07,808][INFO] [o.e.n.Node] [] node name [D9cIo0f] derived from node ID [D9cIo0fns0yoeboHEZTNN
g]; set [node.name] to override
elasticsearch_1 | [2017-08-21T11:24:07,809][INFO] [o.e.n.Node] [] version[5.4.3], pid[1], build[eed30a8/2017-06-22T00:34:03.743Z]
, OS[Linux/4.10.0-32-generic/amd64], JVM[Oracle Corporation/OpenJDK 64-Bit Server VM/1.8.0_131/25.131-b12]
elasticsearch_1 | [2017-08-21T11:24:07,809][INFO] [o.e.n.Node] [] JVM arguments [-Xms2g, -Xmx2g, -XX:+UseConcMarkSweepGC, -XX:CMS
InitiatingOccupancyFraction=75, -XX:+UseCMSInitiatingOccupancyOnly, -XX:+DisableExplicitGC, -XX:+AlwaysPreTouch, -Xss1m, -Djava.awt.headless=t
rue, -Dfile.encoding=UTF-8, -Djna.nosys=true, -Djdk.io.permissionsUseCanonicalPath=true, -Dio.netty.noUnsafe=true, -Dio.netty.noKeySetOptimiza
tion=true, -Dio.netty.recycler.maxCapacityPerThread=0, -Dlog4j.shutdownHookEnabled=false, -Dlog4j2.disable.jmx=true, -Dlog4j.skipJansi=true, -
XX:+HeapDumpOnOutOfMemoryError, -Des.cgroups.hierarchy.override=/, -Xmx256m, -Xms256m, -Des.path.home=/usr/share/elasticsearch]
elasticsearch_1 | [2017-08-21T11:24:18,479][INFO] [o.e.p.PluginsService] [D9cIo0f] loaded module [aggs-matrix-stats]
elasticsearch_1 | [2017-08-21T11:24:18,479][INFO] [o.e.p.PluginsService] [D9cIo0f] loaded module [ingest-common]
elasticsearch_1 | [2017-08-21T11:24:18,479][INFO] [o.e.p.PluginsService] [D9cIo0f] loaded module [lang-expression]
elasticsearch_1 | [2017-08-21T11:24:18,479][INFO] [o.e.p.PluginsService] [D9cIo0f] loaded module [lang-groovy]
elasticsearch_1 | [2017-08-21T11:24:18,480][INFO] [o.e.p.PluginsService] [D9cIo0f] loaded module [lang-mustache]
elasticsearch_1 | [2017-08-21T11:24:18,480][INFO] [o.e.p.PluginsService] [D9cIo0f] loaded module [lang-painless]
elasticsearch_1 | [2017-08-21T11:24:18,480][INFO] [o.e.p.PluginsService] [D9cIo0f] loaded module [prelocator]
elasticsearch_1 | [2017-08-21T11:24:18,481][INFO] [o.e.p.PluginsService] [D9cIo0f] loaded module [reindex]
elasticsearch_1 | [2017-08-21T11:24:18,482][INFO] [o.e.p.PluginsService] [D9cIo0f] loaded module [transport-netty3]
elasticsearch_1 | [2017-08-21T11:24:18,482][INFO] [o.e.p.PluginsService] [D9cIo0f] loaded module [transport-netty4]
elasticsearch_1 | [2017-08-21T11:24:18,483][INFO] [o.e.p.PluginsService] [D9cIo0f] loaded plugin [ingest-geoip]
elasticsearch_1 | [2017-08-21T11:24:18,484][INFO] [o.e.p.PluginsService] [D9cIo0f] loaded plugin [ingest-user-agent]
elasticsearch_1 | [2017-08-21T11:24:18,484][INFO] [o.e.p.PluginsService] [D9cIo0f] loaded plugin [x-pack]
elasticsearch_1 | [2017-08-21T11:24:32,014][INFO] [o.e.d.DiscoveryModule] [D9cIo0f] using discovery type [single-node]
elasticsearch_1 | [2017-08-21T11:24:34,643][INFO] [o.e.n.Node] [] initialized
elasticsearch_1 | [2017-08-21T11:24:34,643][INFO] [o.e.n.Node] [D9cIo0f] starting ...

```

```

a4t@a4t:~/docker-elk
and will be disallowed in 6.0, use [copy_to] instead.
elasticsearch_1 | [2017-08-21T11:24:44,503][WARN] [o.e.d.i.n.TypeParsers] [] field [include_in_all] is deprecated, as [_all] is deprecated,
and will be disallowed in 6.0, use [copy_to] instead.
elasticsearch_1 | [2017-08-21T11:24:44,506][WARN] [o.e.d.i.n.TypeParsers] [] field [include_in_all] is deprecated, as [_all] is deprecated,
and will be disallowed in 6.0, use [copy_to] instead.
elasticsearch_1 | [2017-08-21T11:24:44,507][WARN] [o.e.d.i.n.TypeParsers] [] field [include_in_all] is deprecated, as [_all] is deprecated,
and will be disallowed in 6.0, use [copy_to] instead.
kibana_1 | {"type":"log","@timestamp":"2017-08-21T11:24:44Z","tags":["status","plugin:kibana@5.4.3","info"],"pid":1,"state":"green","m
essage":"Status changed from uninitialized to green - Ready","prevState":"uninitialized","prevMsg":"uninitialized"}
kibana_1 | {"type":"log","@timestamp":"2017-08-21T11:24:44Z","tags":["status","plugin:elasticsearch@5.4.3","info"],"pid":1,"state":"yellow",
"message":"Status changed from uninitialized to yellow - Waiting for Elasticsearch","prevState":"uninitialized","prevMsg":"uninitialized"}
kibana_1 | {"type":"log","@timestamp":"2017-08-21T11:24:44Z","tags":["status","plugin:xpack_main@5.4.3","info"],"pid":1,"state":"yellow",
"message":"Status changed from uninitialized to yellow - Waiting for Elasticsearch","prevState":"uninitialized","prevMsg":"uninitialized"}
elasticsearch_1 | [2017-08-21T11:24:45,024][WARN] [o.e.d.i.n.TypeParsers] [] field [include_in_all] is deprecated, as [_all] is deprecated,
and will be disallowed in 6.0, use [copy_to] instead.
elasticsearch_1 | [2017-08-21T11:24:45,033][WARN] [o.e.d.i.n.TypeParsers] [] field [include_in_all] is deprecated, as [_all] is deprecated,
and will be disallowed in 6.0, use [copy_to] instead.
elasticsearch_1 | [2017-08-21T11:24:45,036][WARN] [o.e.d.i.n.TypeParsers] [] field [include_in_all] is deprecated, as [_all] is deprecated,
and will be disallowed in 6.0, use [copy_to] instead.
elasticsearch_1 | [2017-08-21T11:24:45,038][WARN] [o.e.d.i.n.TypeParsers] [] field [include_in_all] is deprecated, as [_all] is deprecated,
and will be disallowed in 6.0, use [copy_to] instead.
kibana_1 | {"type":"log","@timestamp":"2017-08-21T11:24:45Z","tags":["status","plugin:searchprofiler@5.4.3","info"],"pid":1,"state":"y
ellow","message":"Status changed from uninitialized to yellow - Waiting for Elasticsearch","prevState":"uninitialized","prevMsg":"uninitializ
ed"}
kibana_1 | {"type":"log","@timestamp":"2017-08-21T11:24:45Z","tags":["status","plugin:tilemap@5.4.3","info"],"pid":1,"state":"yellow",
"message":"Status changed from uninitialized to yellow - Waiting for Elasticsearch","prevState":"uninitialized","prevMsg":"uninitialized"}
kibana_1 | {"type":"log","@timestamp":"2017-08-21T11:24:45Z","tags":["status","plugin:watcher@5.4.3","info"],"pid":1,"state":"yellow",
"message":"Status changed from uninitialized to yellow - Waiting for Elasticsearch","prevState":"uninitialized","prevMsg":"uninitialized"}
kibana_1 | {"type":"log","@timestamp":"2017-08-21T11:24:45Z","tags":["status","plugin:console@5.4.3","info"],"pid":1,"state":"green",
"message":"Status changed from uninitialized to green - Ready","prevState":"uninitialized","prevMsg":"uninitialized"}
kibana_1 | {"type":"log","@timestamp":"2017-08-21T11:24:45Z","tags":["status","plugin:metrics@5.4.3","info"],"pid":1,"state":"green",
"message":"Status changed from uninitialized to green - Ready","prevState":"uninitialized","prevMsg":"uninitialized"}
kibana_1 | {"type":"log","@timestamp":"2017-08-21T11:24:45Z","tags":["status","plugin:timelion@5.4.3","info"],"pid":1,"state":"green",
"message":"Status changed from uninitialized to green - Ready","prevState":"uninitialized","prevMsg":"uninitialized"}
kibana_1 | {"type":"log","@timestamp":"2017-08-21T11:24:45Z","tags":["listening","info"],"pid":1,"message":"Server running at http://0
:5601"}

```

```

a4t@a4t: ~/docker-elk
een", "message": "Status changed from red to green - Kibana index ready", "prevState": "red", "prevMsg": "Elasticsearch is still initializing the kibana index."}
kibana_1 | {"type": "log", "@timestamp": "2017-08-21T11:24:48Z", "tags": ["status", "ui settings", "info"], "pid": 1, "state": "green", "message": "Status changed from red to green - Ready", "prevState": "red", "prevMsg": "Elasticsearch plugin is red"}
kibana_1 | {"type": "log", "@timestamp": "2017-08-21T11:24:49Z", "tags": ["license", "info", "xpack"], "pid": 1, "message": "Imported license information from Elasticsearch for [data] cluster: mode: trial | status: expired | expiry date: 2017-08-04T16:58:02+00:00"}
kibana_1 | {"type": "log", "@timestamp": "2017-08-21T11:24:49Z", "tags": ["status", "plugin:xpack_main@5.4.3", "info"], "pid": 1, "state": "green", "message": "Status changed from red to green - Ready", "prevState": "red", "prevMsg": "Elasticsearch is still initializing the kibana index."}
kibana_1 | {"type": "log", "@timestamp": "2017-08-21T11:24:49Z", "tags": ["status", "plugin:searchprofiler@5.4.3", "info"], "pid": 1, "state": "green", "message": "Status changed from red to green - Ready", "prevState": "red", "prevMsg": "Elasticsearch is still initializing the kibana index."}
kibana_1 | {"type": "log", "@timestamp": "2017-08-21T11:24:49Z", "tags": ["status", "plugin:tilemap@5.4.3", "info"], "pid": 1, "state": "green", "message": "Status changed from red to green - Ready", "prevState": "red", "prevMsg": "Elasticsearch is still initializing the kibana index."}
kibana_1 | {"type": "log", "@timestamp": "2017-08-21T11:24:49Z", "tags": ["status", "plugin:watcher@5.4.3", "info"], "pid": 1, "state": "green", "message": "Status changed from red to green - Ready", "prevState": "red", "prevMsg": "Elasticsearch is still initializing the kibana index."}
logstash_1 | [2017-08-21T11:24:49,977][INFO ][logstash.outputs.elasticsearch] Elasticsearch pool URLs updated {:changes=>{:removed=>[], :added=>[http://elasticsearch:9200/]}}
logstash_1 | [2017-08-21T11:24:49,982][INFO ][logstash.outputs.elasticsearch] Running health check to see if an Elasticsearch connection is working {:healthcheck_url=>http://elasticsearch:9200/, :path=>"/"}
logstash_1 | [2017-08-21T11:24:50,118][WARN ][logstash.outputs.elasticsearch] Restored connection to ES instance {:url=>#<URI::HTTP:0x486d79dc URL:http://elasticsearch:9200/>}
logstash_1 | [2017-08-21T11:24:50,120][INFO ][logstash.outputs.elasticsearch] Using mapping template from {:path=>nil}
logstash_1 | [2017-08-21T11:24:50,234][INFO ][logstash.outputs.elasticsearch] Attempting to install template {:manage_template=>{"template">"logstash-*", "version">50001, "settings">{"index.refresh_interval">"5s"}, "mappings">{"_default_">{"_all">{"enabled">true, "norms">false}, "dynamic_templates">[{"message_field">{"path_match">"message", "match_mapping_type">"string", "mapping">{"type">"text", "norms">false}}, {"string_fields">{"match">"*", "match_mapping_type">"string", "mapping">{"type">"text", "norms">false, "fields">{"keyword">{"type">"keyword"}}}], "properties">{"@timestamp">{"type">"date", "include_in_all">false}, "@version">{"type">"keyword", "include_in_all">false}, "geoip">{"dynamic">true, "properties">{"ip">{"type">"ip"}, "location">{"type">"geo_point"}, "latitude">{"type">"half_float"}, "longitude">{"type">"half_float"}}}]}}}
logstash_1 | [2017-08-21T11:24:50,246][INFO ][logstash.outputs.elasticsearch] New Elasticsearch output {:class=>"LogStash::Outputs::ElasticSearch", :hosts=>[<URI::Generic:0x6d74ca43 URL://elasticsearch:9200>]}
logstash_1 | [2017-08-21T11:24:50,351][INFO ][logstash.pipeline] Starting pipeline {"id">"main", "pipeline.workers">4, "pipeline.batch.size">125, "pipeline.batch.delay">5, "pipeline.max_inflight">500}
logstash_1 | [2017-08-21T11:24:50,424][INFO ][logstash.inputs.tcp] Starting tcp input listener {:address=>"0.0.0.0:5000"}
logstash_1 | [2017-08-21T11:24:50,793][INFO ][logstash.pipeline] Pipeline main started
logstash_1 | [2017-08-21T11:24:50,910][INFO ][logstash.agent] Successfully started Logstash API endpoint {:port=>9600}

```

References

- [1] Ericsson, "Ceo to shareholders : 50 billion connections 2020," 2010. [Online]. Available: <http://mb.cision.com/Main/15448/2246220/662223.pdf>.
- [2] B. Sanou, "ICT Facts and Figures 2016," *ITU Telecommunication Development Bureau*, 2016. [Online]. Available: <http://www.itu.int/en/ITU-D/Statistics/Pages/facts/>.
- [3] C. Gomez, J. Oller, and J. Paradells, "Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology," *Sensors*, vol. 12, no. 12, pp. 11734–11753, 2012.
- [4] M. Siekkinen, M. Hienkari, J. K. Nurminen, and J. Nieminen, "How low energy is bluetooth low energy? Comparative measurements with ZigBee/802.15.4," in *2012 IEEE Wireless Communications and Networking Conference Workshops, WCNCW 2012*, 2012, pp. 232–237.
- [5] M. Research, "The need for low cost, high reach, wide area connectivity for the Internet of Things. A Mobile Network Operator's perspective," *white Pap.*, 2014.
- [6] D. Bandyopadhyay and J. Sen, "Internet of things: Applications and challenges in technology and standardization," in *Wireless Personal Communications*, 2011, vol. 58, no. 1, pp. 49–69.
- [7] M. R. Palattella *et al.*, "Internet of Things in the 5G Era: Enablers, Architecture, and Business Models," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 3, pp. 510–527, 2016.
- [8] 3GPP, "Study on provision of low-cost machine-type communications (MTC) user equipments (UEs) based on LTE," *3GPP TR 36.888*. [Online]. Available: <http://www.3gpp.org/dynareport/36888.htm>.
- [9] "A Reference Architecture for the Internet of Things," *WSO2*. [Online]. Available: <http://wso2.com/>.
- [10] "Internet of Things - Architecture." [Online]. Available: <http://www.iot-a.eu/public>.
- [11] IEEE Computer Society, *Part 15.1: Wireless medium access control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPANs)*, vol. 2005, no. June. 2005.

- [12] L. A. N. Man, S. Committee, and I. Computer, *IEEE Standard Part 15.4e: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC sublayer*, vol. 2012, no. April. 2012.
- [13] Bluetooth SIG Inc., “Adopted Specifications,” 2016. [Online]. Available: <https://www.bluetooth.com/specifications/adopted-specifications>.
- [14] E. Khorov, A. Lyakhov, A. Krotov, and A. Guschin, “A survey on IEEE 802.11ah: An enabling networking technology for smart cities,” *Comput. Commun.*, vol. 58, pp. 53–69, 2015.
- [15] T. Adame, A. Bel, B. Bellalta, J. Barcelo, and M. Oliver, “IEEE 802.11AH: The WiFi approach for M2M communications,” *IEEE Wirel. Commun.*, vol. 21, no. 6, pp. 144–152, 2014.
- [16] ETSI, “Low Throughput Networks (LTN); Use Cases for Low Throughput Networks,” vol. 1, pp. 1–24, 2014.
- [17] GSMA, “Mobile IoT: Low Power Wide Area Connectivity GSMA INDUSTRY Paper,” pp. 1–20.
- [18] Wikipedia, “Big Data.” [Online]. Available: https://en.wikipedia.org/wiki/Big_data.
- [19] Cisco, “Cisco Global Cloud Index : Forecast and Methodology , 2015–2020,” *White Pap.*, pp. 1–41, 2016.
- [20] Open Networking Foundation, “SDN Architecture Overview,” *Onf*, pp. 1–5, 2013.
- [21] R. Jain and S. Paul, “Network virtualization and software defined networking for cloud computing: A survey,” *IEEE Communications Magazine*, vol. 51, no. 11. pp. 24–31, 2013.
- [22] M. G. Xavier, M. V Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. a F. De Rose, “Performance Evaluation of Container-based Virtualization for High Performance Computing Environments,” *Proc. 2013 21st Euromicro Int. Conf. Parallel, Distrib. Network-Based Process.*, no. LXC, pp. 233–240, 2013.
- [23] “Dockerfile reference.” [Online]. Available: <https://docs.docker.com/engine/reference/builder/>.
- [24] “Logstash Input plugins.” [Online]. Available: <https://www.elastic.co/guide/en/logstash/current/input-plugins.html>.
- [25] “Logstash Output plugins.” [Online]. Available: <https://www.elastic.co/guide/en/logstash/current/output-plugins.html>.
- [26] R. Kuc and M. Rogozinski, *Elasticsearch Server*, vol. 53, no. 9. 2014.
- [27] “OpenvSwitch.” [Online]. Available: <http://openvswitch.org/>.