Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

A Multicloud Observability Support Based on ElasticSearch for Cloud-native Smart Cities Services

(Article begins on next page)

A multi-cloud observability support based on ElasticSearch for Cloud-native Smart Cities Services

Sofia Montebugnoli
DISI
University of Bologna
Bologna, Italy
sofia.montebugnoli3@unibo.it

Luca Foschini
DISI
University of Bologna
Bologna, Italy
luca.foschini@unibo.it

Abstract—**Effective communication and information sharing among different districts and cities are crucial for the management of utility flows, traffic, and emergencies in smart cities. In this scenario, a smart city requires cloud-native solutions to collect and analyze data from various sources, including traffic sensors and public transport vehicles. Thus, a multi-cloud observability approach is proposed to aggregate data from different localities. The solution aims to provide a complete suite for observability capable of collecting data across layers of a multi-cloud and integrating already existing open-source projects.**

Index Terms—**Observability, Multicloud, ElasticSearch, Terraform**

I. INTRODUCTION

In the context of smart cities, communication and information sharing among different districts is crucial for the effective management of utility flows, traffic, and potential disastrous or harmful events that could impact neighbouring areas. Thus, it is imperative to develop and implement applications that can collect and sense data from multiple locations and share relevant information between different regions. From the perspective of IT assets should be capable of gathering real-time data about various aspects of urban life, such as traffic flow, energy consumption, and water usage, to facilitate better decision-making and enable prompt response to emergencies. By leveraging these technologies, cities can enhance their capacity to manage and maintain critical infrastructure, optimize resource utilization, and enhance the citizens quality life.

To facilitate this kind of communication, also a federated application environment is essential. This environment should allow for the collection and analysis of data from various sources, including traffic sensors, public transport vehicles, and other relevant sources. The data should then be processed and shared with all relevant parties in a timely and efficient manner.

In this federated scenario, multiple actors are involved, necessitating to operate seamlessly across different platforms to run applications on the cloud. Since there is a pullulating number of Cloud Service Providers (CSP) providing their Platform as a Service PaaS support, it is likely that different actors involved in the smart cities

services management adopt multiple PaaS based on their need to run digital twins, real-time alerting and other data analytics applications. Many cloud providers already offer Application Performance Monitoring (APM) to observe, manage, and optimize the performance of applications. However, as recently noted [1] this is not a feasible solution to aggregate information in a federated and heterogeneous environment.

In distributed neighbours, cities and companies facilities, spanning multiple cloud providers, logs generated from different tools may be stored across various locations, creating the need for efficient data aggregation and analysis across multiple cloud environments. Collecting and aggregating logs from different cloud platforms can be a complex task that requires careful attention to data consistency and synchronization. Appropriate tools and techniques must be used for efficient data processing.

Although the use of a multi-cloud approach typically entails increased orchestration complexity, the adoption of a practical multi-cloud strategy provides significant benefits in terms of attaining crucial business objectives, such as ensuring high availability, avoiding vendor lock-in, and implementing failover mechanisms. In this regard, the successful implementation of a comprehensive solution for observability that covers different layers of service is paramount to meeting user requirements and serves as a foundation for the (semi)automatic detection and mitigation of faults.

To this end, the proposed solution for observability in a multi-cloud context proposes a federated observability approach to aggregate data from different localities. Our proposal aims at providing a complete suite for observability capable to collect data across layers of a multi-cloud. In addition, our solution aims to integrate already existent open-source projects in order to allow distributed query and storage along with the different districts and cities.

The structure of this article is as follows, Section II covers the motivations and the design guidelines for this work. Section III presents the technologies involved in multi-cloud and observability including the ELK stack, Kubernetes and Terraform. Section IV illustrates an ar-

chitectural solution for observability in the multi-cloud environment, whilst Section V explains the experimental results of the proposed architecture. Section VI shows related work significant for this article. Finally, Section VII draws the conclusions and ongoing work directions.

II. DESIGN GUIDELINES

Cloud Computing technologies have revolutionized many fields and aspects of our life, providing unprecedented access to easy-to-manage computational resources accessible through the network. In Smart Cities, many cloud-native solutions serving city management and providing services to citizens rely on Cloud Computing hosting. The increasing reliance on these solutions seeks mechanisms guaranteeing a high level of availability of these services. Another major issue is represented by the need for interoperability among different cloud providers that are adoptable by the several stakeholders composing the smart cities services scenario.

The Multi-Cloud paradigm by combining services provided by different CSP allows end-users to benefit from the best features offered by each vendor. This approach features portability across multiple cloud infrastructures and it is typically built on open-source cloud-native solutions [2]. Nowadays, the swarming increase of CSP has led to an urgent need for the integration of open-source solutions. In this context, multi-cloud solutions have emerged as a promising approach that provides a more extensive range of features and capacity to end-users, ensuring the best fit for their requirements. However, the extensive heterogeneity and lack of standardization among different vendors pose a significant challenge to the seamless compatibility required for cross-cloud solutions.

As a response to this challenge, the Cloud Native Foundation has been undertaking a significant effort towards enabling broad compatibility of their financed tools to manage a multi-cloud environment. By leveraging this compatibility, users can benefit from the flexibility, horizontal scalability and advantages of multi-cloud solutions, without facing the potential obstacles and complexities of managing different cloud environments from multiple vendors.

Observability refers to the ability to gain insights into the behaviour of complex systems through the collection, analysis, and correlation of data from various sources [3]. These insights are the result of the aggregations of different data sources, which comprehend logs, metrics, and tracing. In multi-cloud environments, achieving observability can be challenging due to the distributed and heterogeneous nature of the infrastructure.

To address these challenges and achieve observability in multi-cloud environments, it is necessary to employ several key practices and technologies. Instrumentation, which involves the use of monitoring tools and agents to collect and transmit data from all relevant components of the multi-cloud environment, is essential. Once the data is collected, it must be aggregated and centralized in a single location for analysis and correlation, which can
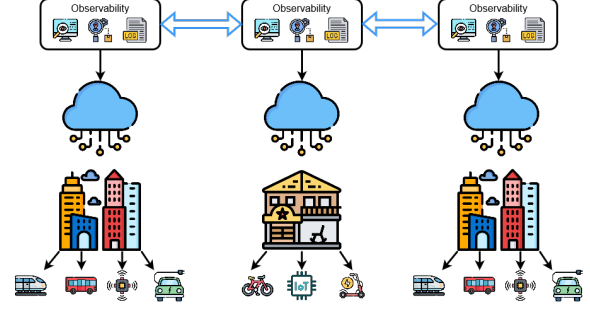

Fig. 1. Architecture overview for smart cities

be accomplished through the use of log aggregation and metrics aggregation platforms.

Data analysis is also critical to gaining insights from the collected data, and advanced analytics and machine learning techniques can be used to identify patterns, anomalies, and correlations that may be indicative of problems or opportunities for optimization. The results of the data analysis should be presented in a clear and actionable manner, such as through dashboards or alerts, to enable quick and effective decision-making.

Collaboration is also important in a multi-cloud environment, as all relevant stakeholders, including developers, operations teams, and business leaders, must have access to the observability data and insights. Collaboration tools and processes can facilitate communication and decision-making across different teams and organizations.

By adopting these practices and leveraging the appropriate technologies, one can achieve observability in multi-cloud environments and gain greater visibility and control over complex systems. In particular, in this paper, we explore state-of-the-art Multi-Cloud enabled observability platforms to support cloud-native services for Smart Cities, as shown in Fig. 1.

III. BACKGROUND

Our solution is at the crossroad of several different technologies. The following sections present some core technologies involved in the observability stack supported by multi-cloud tools.

A. ELK Stack

The ELK stack is an open-source project maintained by the Apache software foundation which aims at providing a full stack for observability purposes [4]. The ELK stack comprises four different tools Beats, Logstash, Elastic-Search and Kibana. The ELK stack offers a huge flexibility to their users, in fact, one does not necessarily have to deploy all the tools, but it can adjust them to its demand.

Beats. Following the data flow, the Beats are lightweight data shippers, which offer organized data gathering, by distinguishing the different sources like files, packets, metrics, audits, and logs. Beats ship all the required data to the subsequent layers of the stack represented by Logstash or ElasticSearch by conforming it to the Elastic Common Schema (ECS). ECS defines a common set of fields to be used when storing event data in ElasticSearch, such as

logs and metrics. Beats are a plug-and-play tool which is perfectly incorporated into the data ingestion process.

In fact, through modules, the beats can accelerate the data visualization experience by collecting, parsing, and visualizing information from key data sources such as cloud platforms, containers and systems, and network technologies. One of the main advantages of these agents is that they are highly lightweight components and do not require a lot of computational resources to perform data collection. In this way, the performance of the monitored system services will not be affected by the execution of the Beats modules.

Logstash. Logstash fulfils the tasks of the transform and ingestion layer. It simultaneously collects data from the different data sources and based on the specified tailored masks, it filters them [5]. Finally, it forwards the transformed data to ElasticSearch. The Logstash work pipeline begins with the data inputs, thus all the possible data sources which generate events, including the Beats, are suitable inputs. The filtering phase provides some user define filters to select only those events which satisfy certain requirements. Logstash offers many filter functionalities to drop, mutate, clone, geolocate or simply transform the events. Once the transformation phase is completed the filtered events are forwarded to ElasticSearch or eventually stored in some non-relational database like MongoDB or a streaming platform like Kafka.

In the multi-cloud context, there can be many data sources generating logs and metrics with different formats and standards. A large number of input and output plug-ins have been developed making it an extremely flexible and powerful tool that manages to interact with a wide variety of different architectures and technologies. However, this flexibility comes at a price in terms of hardware requirements. For this reason, Logstash is hardly deployed in performance-limited machines. On the other hand, on the other hand, ElasticSearch's Ingest nodes can process documents prior to indexing, applying transformations and enriching the content of logs using fewer computational resources than the tool operating in the Transform&Ingestion layer. These particular nodes duplicate most of the functionality of Logstash, making its use sometimes superfluous.

ElasticSearch. ElasticSearch is a distributed research engine capable of storing large amounts of data in an optimized full-text format [6]. It is evident that ElasticSearch couples the storage and research activity. Its innovative features consist in the capability to comply with high availability and real-time full-text search. ElasticSearch was designed as a modular framework to support parallel searches. In this way, the computational and storage capacity of the system can be increased simply by adding nodes to the system. The reliability of the system is automatically managed by ElasticSearch, which migrates data and elects replica data to primary data as a response to failures of the system.

The communication within the ElasticSearch topology is carried out using the JSON language associated with the REST API. The storage paradigm used in ElasticSearch is the Document Oriented NoSQL, this allows the database to parse large amounts of text in any format, indexing and parsing documents efficiently based on the metadata of the document itself.

In the Storage and search layers, ElasticSearch uses a master-slave paradigm to manage the interaction between nodes. When a new node is added to the cluster, through an election, a master node is chosen from the available ones. The active master node will be responsible for organizing the cluster and managing the data. To ensure the fault-tolerance property, when the master node stops working, ElasticSearch will ensure that another one is selected. Indexes within the ElasticSearch ecosystem are logical namespaces formed by one or more shards, a low-level data distribution unit. Each index shard is a single instance of Lucene. It is through this mechanism that ElasticSearch distributes the data within the cluster.

B. Kubernetes and Terraform

1) Kubernetes: Much like ElasticSearch, Logstash does not natively possess mechanisms that allow it to scale its number of instances as the workload changes. To solve the problem related to the management of the monitoring stack, we use the open-source container orchestration system, Kubernetes [7]. The platform makes it possible to automate the deployment, scalability, and management of containers in multi-cloud environments. Kubernetes technology relies on the vast adoption by the community, which results in a standard de facto [8] in the cloud computing environment [9].

The advantages of the Kubernetes deployment are many. Kubernetes automates the deployment of the Elastic-Search Logstash Kibana (ELK) stack across multiple cloud providers, moreover, it also provides tools for infrastructure management, including auto-scaling, load balancing, self-healing, and rolling updates, which make it an ideal solution for managing containerized workloads at scale.

2) Terraform: In order to address the multitude of CSP across a multi-cloud environment, Terraform is employed to manage and provision cloud resources in a programmatic way [10]. Since Terraform is an open source Infrastructure as Code tool for the Kubernetes infrastructure, it results in the best solution for this cross-cloud environment.

Terraform deploys and manages Kubernetes resources across multiple cloud providers, allowing users to take advantage of the strengths of each CSP while avoiding vendor lock-in. With Terraform, it is possible to define Kubernetes resources as code, enabling various benefits, like version control, and automation of the infrastructure deployment and management. Furthermore, it ensures that the Kubernetes infrastructure is consistent and reliable, by applying the same configuration across multiple environments and enforcing the desired state of your infrastructure.
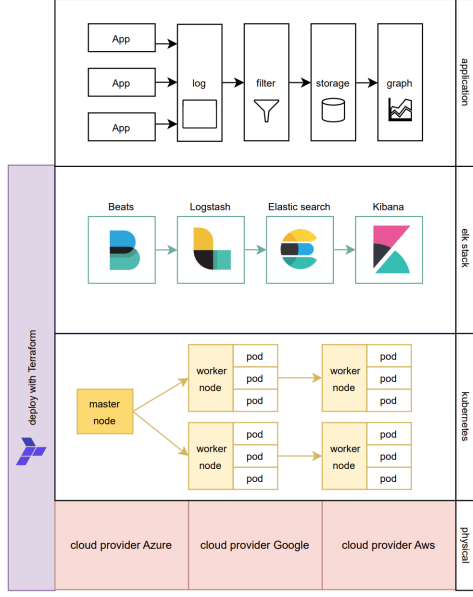
Fig. 2. Proposed architecture

IV. Multi-cloud ElasticSearch deployment for observability of large cloud-native services

This paper proposes a cloud architecture that provides a scalable, flexible, and reliable solution for managing ElasticSearch, Logstash, and Kibana in a production environment.

Fig. 2 shows how we originally integrate the use of Kubernetes for containerization of ElasticSearch, Logstash, and Kibana, combined with Terraform for configuration management in a novel multi-cloud architecture aimed to ease and fasten the development of new cloud-native services for smart cities scenarios. This multi-cloud approach has several benefits.

Kubernetes provides a robust platform for container orchestration, including scaling, health monitoring, and service discovery which runs on multiple cloud providers. By leveraging Kubernetes, the cloud architecture can take advantage of these features to ensure the availability and scalability of the ElasticSearch, Logstash, and Kibana containers.

Moreover, containerizing the ElasticSearch, Logstash, and Kibana components allows for a more modular and flexible deployment. Each component can be scaled independently based on resource needs, and updates can be performed without impacting other components.

The use of Terraform for configuration management of the previous tools provides a consistent and repeatable way to deploy the architecture. Terraform allows for infrastructure as code easing the versioning and maintainance of changes to the deployment over time. Additionally, Terraform can be used to automate the deployment process, reducing the risk of human error and saving time.

Overall, this architecture provides a robust, scalable, and highly available solution for containerizing and managing ElasticSearch, Logstash, and Kibana across different cloud providers, while ensuring efficient and effective data analysis.

To avoid the case where a single instance of Logstash may fail in an attempt to ingest a large amount of input data, a Kubernetes Horizontal Pod Autoscaling (HPA) was created to allow automatic scaling as the computational load of the single pod increases, this is possible by defining a maximum and minimum number of Logstash instances. The Kubernetes LoadBalancer allows load balancing between instances of a given service. In this way, incoming traffic to the service is randomly distributed among all instances of Logstash so as to ensure higher availability and scalability. The use of an HPA to manage the workload variation of the instances of the Transform&Ingestion is justified in cloud-native application scenarios such as smart cities that invest a lot of computational resources to achieve real-time analysis.

V. Implementation and Experimental Results

The testbed for the infrastructure evaluation has been set up as follows. Five virtual machines were created to host a Kubernetes cluster in the first instance. The VMs specifications present 4 CPUs, 8GB of RAM, a 72GB disk size and the 18.04.6 Ubuntu version. Then, a Kubernetes cluster was deployed with a Kubernetes operator, and the given specification consists of two masters and three slaves. ElasticSearch is deployed upon Kubernetes, by default the ElasticSearch cluster requires three pods to accomplish the three different functions: master, data and client. However, to simulate heavy workloads caused by many searches and update requests, ElasticSearch is provided with two coordination nodes, one master and one for data. The Logstash instances were supplied with 4 CPUs, and 1 GB RAM and received the Filebeats on port 5044.

Listing 1 and 2 show examples of the Terraform deployments for, respectively, ElasticSearch and Logstash on Kubernetes. These are examples of how the proposed architecture eases, in a real testbed, the deployment of components without using Helm charts.

The deployment consists of defining two Kubernetes Deployments, one for ElasticSearch and another for Logstash, along with associated Services and ConfigMaps. The ElasticSearch Deployment includes StatefulSet configuration, which is necessary for data persistence. The Logstash Deployment has ten replicas and mounts a ConfigMap to define the Logstash configuration.

```
resource "kubernetes\_deployment"
    "elasticsearch-master" {
  name = "elasticsearch-master"
  replicas = 1
  selector = {
    app = "elasticsearch"
    component = "master"
  }
  template {
    metadata {
      labels = {
        app = "elasticsearch"
        component = "master"
      }
    }
    spec {
      containers {
        name = "elasticsearch"
```

```
    image = "elasticsearch:7.10.0"
    ports {
      container_port = 9200
    }
    volumeMounts {
     mountPath = "/data"
     name = "elasticsearch-data"
    }
   }
 }

}
```
Listing 1. Terraform description for ElasticSearch master

```
resource "kubernetes\_deployment" "logstash" {
 name = "logstash"
 replicas = 10
 selector = {
  app = "logstash"
 }
 template {
  metadata {
   labels = {
    app = "logstash"
   }
  }
  spec {
   containers {
    name = "logstash"
    image = "logstash:7.10.0"
    ports {
     container_port = 5044
    }
    volumeMounts {
     mountPath = "/etc/logstash/conf.d"
     name = "logstash-conf"
    }
   }
  }
 }
}
```
Listing 2. Terraform description for Logstash

The presented experiments aim to provide a first assessment of our infrastructure to support observability applications in a multi-cloud environment for smart city assets. in particular, we focused our analysis on the more critical components, to evaluate a multi-cloud solution based on the ELK stack. Namely, Logstash and ElasticSearch are stressed to test their response to heavy workloads.

In the first experimental scenario, Logstash is stressed with different workloads with the aim of demonstrating its resistance. The Logstash instances contain some simple filters to simulate the processing of incoming data flow. To test the effective operation of the input balancing service to the instances of Logstash, we decided to create an architecture with two pods and a Load Balancer type service offered by Kubernetes HPA. Fig. 3 represents the request distribution between different Logstash pod instances, since the Logstash is hosted in two VMs the graph outlines the load balancing of the interactions.

The second experimental scenario assesses the resilience of ElasticSearch when exposed to intense data workloads: the master and client nodes are stressed with six scripts which generate 258 requests per second. Each demon operates some searches and randomly applies some updates on the storage. The test of these capabilities is relevant to evaluate the behaviour of ElasticSearch in clusters involving numerous nodes and test its workload management, load balancing and eventually the ability to detect congestions. The graphs in Fig. 4 plots the CPU percentage for the whole experiment duration. It is noteworthy to highlight that the CPU utilization for the master node is particularly

crushed by requests especially due to data indexing, node updates and shards.

The presented results show how is possible to provide an adequate architecture for a multi-cloud environment. However, the stress of ElasticSearch, especially for the master node, proves the limited scalability in case of many update requests. A relevant increase in the number of client nodes on ElasticSearch can then result in slowing down the whole monitoring support and eventually in a possible crash of the master. In order to avoid previous issues, is possible to deploy more master replicas. However, the replication of the master can increase the workload for the coordination nodes. Furthermore, the replication of Logstash is applied to the system to support the heavy workloads from FileBeats, though this can cause congestion in the data flow towards the master node of ElasticSearch, and possibly result in saturation of the data nodes.

VI. RELATED WORKS

The pervasive diffusion of mobile and IoT devices continuously generating and sensing data in the city contexts, has immediately highlighted the need for monitoring systems able to gather those data and present them through aggregated forms through dashboards or query services. In the following, without any pretense of being exhaustive, we report a set of solutions that similarly to ours address the cloud-native service observability problem, starting with those that are more distant to terminate with the closer ones.

We start with [11] where authors proposed a monitoring system to estimate the positioning of IoT devices in cities in order to achieve an optimized transmission of data. The proposed solutions leverage cloud computing technologies and an automation layer that takes advantage of analytics delivered by the monitoring platform and provides insights on how to deploy devices. In [12] the authors proposed Triangulum a cloud-based prototype that supports city management with real-time monitoring and visualizations of information concerning city trends. The proposed solutions also support online analysis and queries. Through the proposed solutions the authors aim to provide effective support and communication between users and urban services providers giving also a global perspective on the state of the managed city.

In [13] authors propose and validate an extensive use of the ElasticSearch framework as a support to the collection, aggregation and analysis of big data coming from various sources in the city context. Analyzed data are then offered to urban service providers who make informed decisions. While validating ElasticSearch as a valuable technical solution for the monitoring of smart city aspects authors do not consider the issue of cloud availability nor the more than likely eventuality that different actors in the smart city ecosystem could opt for different cloud providers following conveniences or peculiar needs.

The theme of observability in a multi-cloud environment has been already addressed in the literature. In particular, in [14] authors proposed a novel middleware able to
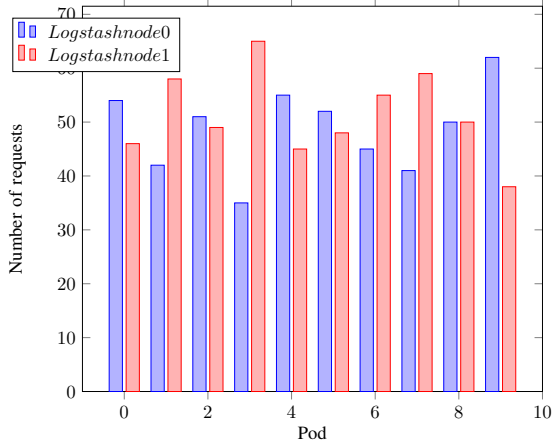
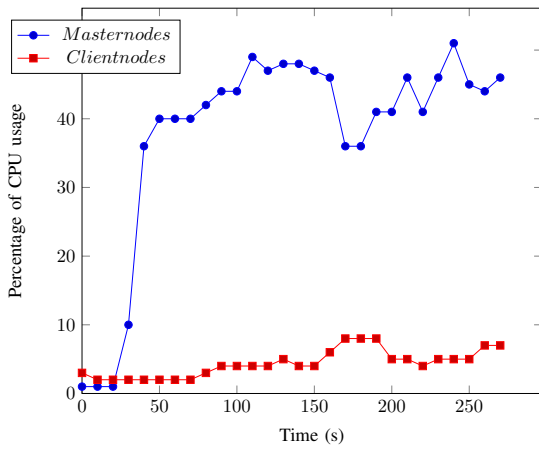Fig. 3. Distribution of the requests on the Logstash pods

Fig. 4. Percentage of CPU usage for master node and client nodes

effectively monitor the availability and performance of services hosted on public cloud Platform as a Service (PaaS). The proposed solution exploits gathered metrics to opportunistically redirect service traffic achieving adaptive load balancing and fault mitigation. However, the approach addresses observability at the infrastructural layer, monitoring the health and performance of services in execution on the different Cloud and not exploiting the capabilities of these frameworks to aggregate and extract insights of real use cases in the Smart City context.

All aforementioned approaches support the validity of exploiting observability systems to monitor and manage cloud-native services for Smart Cities extracting important analytics from them. Our proposal aims to expand this solution by providing a more reliable architecture supporting Smart Cities services development and enabling to overcome eventual vendor lock-ins.

VII. Conclusions

The observability of cloud-native services in Smart Cities plays a crucial role in city management providing the capabilities of aggregating and analyzing big volumes of data in a near real-time fashion with also the possibility of integrating query and alerting systems as well as visual-

ization dashboards. Cloud Computing technologies are at the forefront of this revolution by providing fast scalable computational resources on which host observability services. However, centralized cloud solutions can not offer strong guarantees in terms of availability, since the failure of a single provider could cause major outages having drastic consequences on the smart city offer. The creation of an observability platform able to leverage a Multi-Cloud approach could overcome these issues and offer also the possibility to city service offers of adopting different cloud providers following their preferences. In this paper, we proposed a monitoring solution based on the ElasticSearch stack that exploits built-in federation systems and it is able to leverage multiple cloud providers. The proposed solution is tested in a real-use case environment simulating the exchange of information among several smart city actors showing that our proposal suits the requirements of this scenario, and is capable of managing a multi-cloud scenario.

References

[1] S. Chasins, A. Cheung, N. Crooks, A. Ghodsi, K. Goldberg, J. E. Gonzalez, J. M. Hellerstein, M. I. Jordan, A. D. Joseph, M. W. Mahoney, A. Parameswaran, D. Patterson, R. A. Popa, K. Sen, S. Shenker, D. Song, and I. Stoica, "The sky above the clouds," 2022.

[2] J. Hong, T. Dreibholz, J. A. Schenkel, and J. A. Hu, "An overview of multi-cloud computing," in *Web, Artificial Intelligence and Network Applications: Proceedings of the Workshops of the 33rd International Conference on Advanced Information Networking and Applications (WAINA-2019) 33*, pp. 1055–1068, Springer, 2019.

[3] M. Usman, S. Ferlin, A. Brunstrom, and J. Taheri, "A survey on observability of distributed edge & container-based microservices," *IEEE Access*, 2022.

[4] Y. Gupta and R. K. Gupta, *Mastering Elastic Stack*. Packt Publishing Ltd, 2017.

[5] J. Turnbull, *The Logstash Book*. James Turnbull, 2013.

[6] C. Gormley and Z. Tong, *Elasticsearch: the definitive guide: a distributed real-time search and analytics engine*. " O'Reilly Media, Inc.", 2015.

[7] K. D. Tsaousi, "Elasticity of elasticsearch," 2021.

[8] C. Carrión, "Kubernetes as a standard container orchestrator-a bibliometric analysis," *Journal of Grid Computing*, vol. 20, no. 4, p. 42, 2022.

[9] N. Naydenov and S. Ruseva, "Cloud container orchestration architectures, models and methods: a systematic mapping study," in *2023 22nd International Symposium INFOTEH-JAHORINA (INFOTEH)*, pp. 1–8, 2023.

[10] L. R. De Carvalho and A. P. F. de Araujo, "Performance comparison of terraform and cloudify as multicloud orchestrators," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pp. 380–389, IEEE, 2020.

[11] M. Dryjanski, M. Buczkowski, Y. Ould-Cheikh-Mouhamedou, and A. Kliks, "Adoption of smart cities with a practical smart building implementation," *IEEE Internet of Things Magazine*, vol. 3, pp. 58–63, 4 2020.

[12] M. Farmanbar and C. Rong, "Triangulum city dashboard: An interactive data analytic platform for visualizing smart city performance," *Processes 2020, Vol. 8, Page 250*, vol. 8, p. 250, 2 2020.

[13] A. Talaş, F. Pop, and G. Neagu, "Elastic stack in action for smart cities: Making sense of big data," *Proceedings - 2017 IEEE 13th International Conference on Intelligent Computer Communication and Processing, ICCP 2017*, pp. 469–476, 11 2017.

[14] A. Sabbioni, A. Bujari, L. Foschini, and A. Corradi, "An efficient and reliable multi-cloud provider monitoring solution," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pp. 1–6, 2020.