International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2020): 7.803

Implementing Cloud-Native Technologies for Big Data Processing: A Case Study with Kubernetes and Airflow

Chandrakanth Lekkala

Email: *chan.Lekkala[at]gmail.com*

Abstract: ***This paper presents a case study on the architecture design and implementation details of cloud-native technologies for big data processing. Cloud-native technologies, such as Kubernetes and Airflow, are modern solutions intricately connected as essential components within IT infrastructures. They are designed specifically for processing and managing data in cloud environments. These technologies leverage the scalability and flexibility of cloud computing to enable efficient and reliable data storage, analysis, and retrieval., focusing on Apache Airflow and Kubernetes. Acting as a container orchestrator, Kubernetes efficiently manages a vast number of containers, eliminating the necessity to explicitly outline the configuration for executing specific tasks. Meanwhile, Airflow is the orchestration layer for managing data processing workflows within the Kubernetes environments. The findings of this paper underscore the potential of Kubernetes and Airflow in enabling seamless orchestration and management of big data workflows in cloud environments.***

Keywords: Cloud-native, Apache Airflow, Kubernetes, Big Data Processing, Orchestration Stability, Efficiency

1. Introduction

A confluence of economic, social, and technological trends, including the growing ubiquity of wireless broadband access, widespread adoption of smart devices and infrastructure, and appeal of social networking, are resulting in the generation of vast streams of data, also known as big data. Big data continues to grow exponentially over time, and the datasets are so huge and complex in variety, volume, and velocity that traditional data management systems cannot process, store, or analyze them effectively [1]. Concurrently, the modern landscape of complex applications – with end users expecting continuous innovation and unapparelled responsiveness – requires organizations' systems to be more strategic and increasingly flexible. The demand for cost-effective, scalable, and reliable approaches prompts the exploration of cloud-native services, such as Kubernetes and Airflow, as a viable alternative. Kubernetes orchestrate containerized applications to run on a cluster of hosts and use cloud platforms to automate and manage cloud-native applications [2]. Meanwhile, Airflow's rich user interface complements Kubernetes by making it possible to visualize pipelines running in production, troubleshoot issues, and monitor the progress of complex data pipelines. This paper demonstrates the implementation of Kubernetes and Airflow for big data processing, delving into the architecture and deployment strategies of these cloud-native technologies.

2. Literature Review

The proliferation of big data, as reflected in the widespread of digital devices and mobile subscriptions, has transformed the landscape of modern business, empowering organizations to derive valuable insights and drive innovation. More than 50 billion interconnected devices are estimated to be deployed worldwide in areas such as transport systems, the environment, security, energy control systems, and healthcare [3]. Given that internet penetration exceeds 100% in developing and developed countries within the Organization for Economic Cooperation and Development (OECD) and that wireless broadband penetration is nearly 70% in OECD areas, the source of big data will continue to grow further [4]. The amount of data traffic generated by mobile devices and sensors has been doubling every year, as illustrated in Figure 1. However, harnessing the potential of big data comes with its own set of challenges, including the need for scalable, reliable, and cost-effective data processing solutions. The literature surrounding cloud-native technologies provides valuable insights into the theoretical foundations and practical implementations of these technologies.

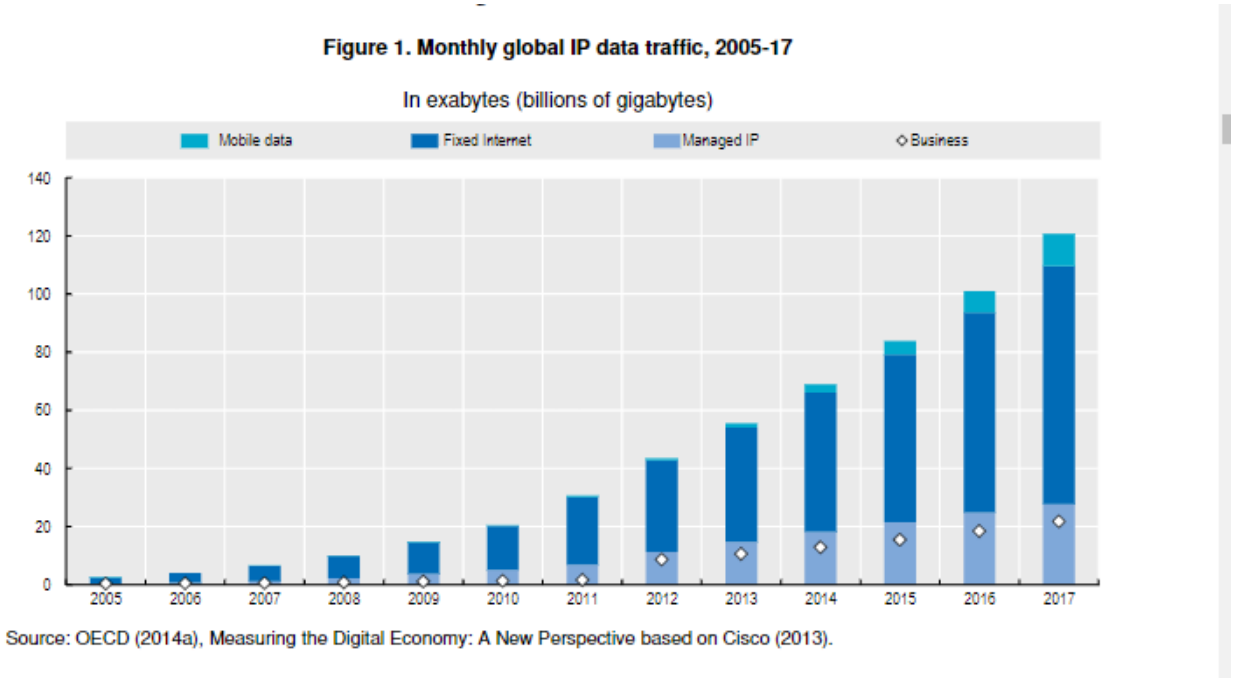

Figure 1: Global IP Data Traffic

1) Cloud-Native Technologies

The term "cloud-native" identifies the practice of developing, deploying, and maintaining modern applications that take the flexible computing characteristics offered in the cloud environment itself [5]. Being that in mind, cloud-native systems are architected and developed using features and capabilities of cloud computing environments such as their scalability, built-in resiliency, ability to easily manage the workload and provide elasticity. The cloud-native technologies equip enterprises with tools for running applications in private, public and hybrid clouds on a massive scale. Characteristics which are microservices architecture, immutable infrastructure, automated scaling, and declarative language in application programming interfaces (APIs) can be named as the main resources of scalability and efficiency of cloud-native technology [6]. These characteristics allow server to provide the same function to all computer system, so the data processing and saving is performed in a similar way without the limitations and failure of devices. Additionally, the unaltered state of cloud-native technologies post-deployment significantly reduces the complexity of such technologies.

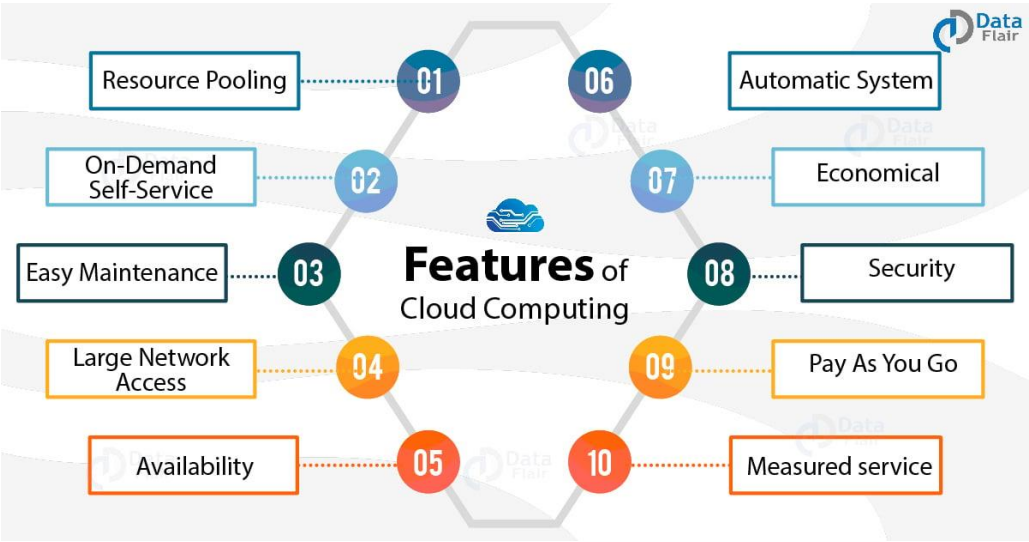

Figure 2: Features of Cloud-Native Technologies

2) Kubernetes

Kubernetes orchestrate container-based workflows and applications to operate on a cluster of hosts. To achieve this aim, Kubernetes facilitates the automated deployment, management, and monitoring of cloud-native applications, regardless of whether they're deployed on public cloud platforms or on-premises infrastructure [7]. The typical architecture of Kubernetes consists of the control plane, nodes, and clusters.

The control plane fulfills two primary roles: (1) serving as the gateway to the Kubernetes API, and (2) overseeing the nodes comprising the cluster. To perform the functions of controlling communications and managing noes, the control plane uses four primary components: A kube-API server, which exposes Kubernetes API, etcd used for data storage; a kube-scheduler which assigns new pods to a node for execution, and a kube-controller-manager which functions as a controller of all the functions of the control plane is stored [8].

Within Kubernetes, applications run in Pods, the fundamental execution unit. These Pods house the containers and are executed on worker nodes. Nodes consist of three primary elements: Kubelet, acting as an agent ensuring container presence within a Kubernetes pod; kube-proxy, serving as a network proxy across every node in a cluster; and the container runtime, responsible for container execution [8]. These components collaborate to establish a reliable framework for automating the deployment, scaling, and oversight of container-based applications. Moreover, they ensure optimal resource utilization through the coordination of kube-scheduler and kube-controller.

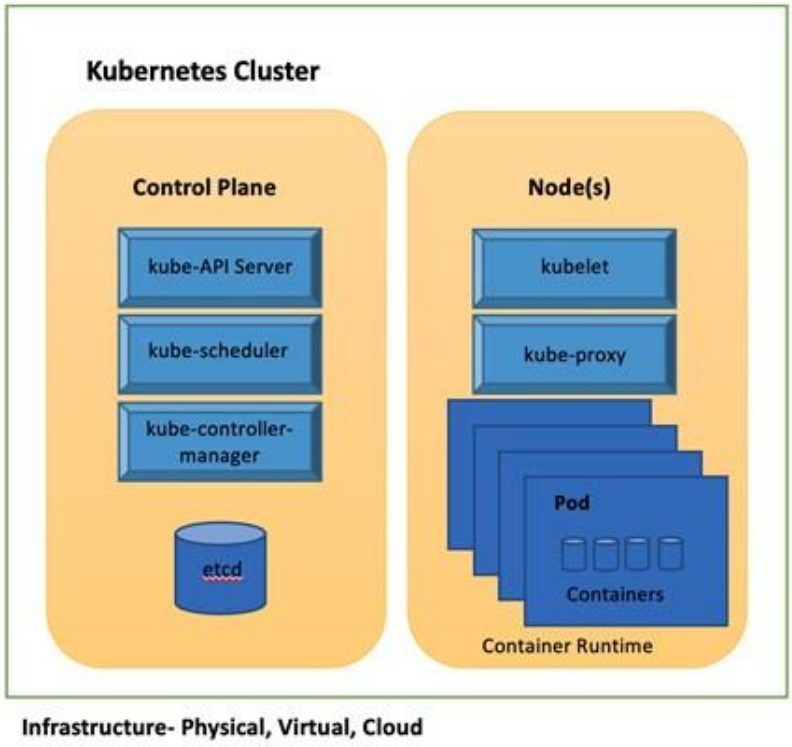

Figure 3: Components of a Kubernetes

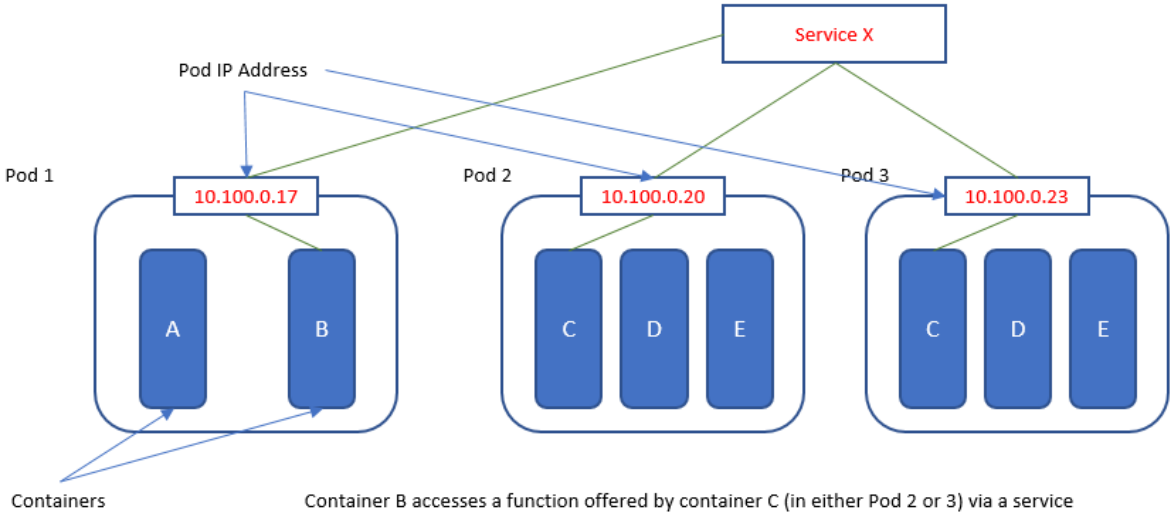

Figure 4: Simplified view showing how services interact with pod networking in the Kubernetes cluster

3) Airflow

Airflow, in this case Apache Airflow, is a workflow management platform that allows for monitoring, scheduling, and orchestration of complex data pipelines. It is used for the orchestration and scheduling of data pipelines [9]. Typically, Orchestrating data pipelines involves arranging and coordinating the flow of data from various sources. Within an Airflow, the data pipelines deliver data sets that are ready for consumption either by data science, machine learning models, or business intelligence applications that support big data applications. The installation of an Airflow comprises the following components:

- Scheduler – Responsible for overseeing the progression of tasks and DAGs, it initiates task instances as soon as their prerequisites have been met.
- Webserver – Presents a user interface to trigger, inspect, and debug the behavior of tasks and DAGs.
- DAG file – A DAG (directed acyclic graph) is the core concept of Airflow. It collects tasks together and defines how dependencies and relationships should run [10].
- Metadata database – This is the database that the components of Airflow use to store the state of workflows and tasks [10].

Overall, Airflow serves as the orchestration layer for managing data processing workflows. Using the above-mentioned components, Airflow facilitates the creation of complex data pipelines by defining DAGs and orchestrating their execution across distributed systems.

3. Case Study: Implementing Kubernetes and Airflow for Big Data Processing

3.1 Use Case Description

This study focuses on TechShop, a fictitious e-commerce company that operates in a highly competitive market where understanding and meeting the ever-evolving consumer preferences is paramount for success. To maintain a competitive edge and remain relevant in the market, TechShop collects data from various sources, including purchase history, social media engagement, website interactions, and demographic information. This information is highly diverse and contains large collections of structured, semi-structured, and unstructured data that are expected to grow exponentially over time. Additionally, the complexity, variety, volume, and velocity of the datasets overwhelm the current data management system, hindering its ability to efficiently process, analyze, and store them. Therefore, TechShop aims to implement a modern, cloud-native solution to process and analyze data effectively. The primary objectives of the cloud-native solution include:

- The solution should be able to handle fluctuations in data volume and velocity without manual intervention.
- The architecture should support iterative development with different analytical models and data processing algorithms.
- The data processing workflows must be fault-tolerant and robust enough to ensure consistent performance.

3.2 Architecture Overview

The proposed architecture comprises three main components: data ingestion, data processing, and data analytics. For the data ingestion, raw data and assorted files (purchase history, social media engagement, website interactions, and demographic information) collected from various sources, such as streaming databases and external APIs, will be imported into a single, cloud-based storage medium – Google Cloud Storage (GCS). GCS is the most-suited storage option because it is scalable, durable, and cost-effective, with high availability and low latency access [5]. The data stored in the GCS will then be transformed and stored in a centralized repository. Once ingested, the data will undergo various transformations, which involve aggregations and machine learning algorithms to extract meaningful insights [11]. The processed data will then be analyzed to generate visualizations, reports, and actionable insights for business stakeholders.

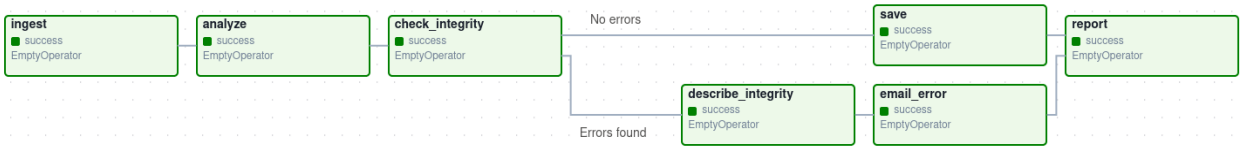

Figure 5: Architecture Overview

3.3 Kubernetes Deployment

TechShop opts to deploy Kubernetes clusters on Google Cloud Platform – a leading public cloud provider – to leverage its scalability features and managed services. Deployment of the Kubernetes will encompass the following components:

- Mater Node – This node controls and manages a set of nodes. It comprises the following components to help manage worker nodes: Kube-API server, Kube-Controller-Manager, etcd, and Kube scheduler [12]. This node will control the overall Kubernetes cluster, managing resource allocation, scaling, and scheduling.
- Pods – Pods are the smallest deployable units in Kubernetes, encapsulating the containers with shared network resources such as data ingestion services, analytics tools, and processing engines [13].
- Services – The services will provide network access to a set of pods, enabling load balancing and task discovery within the cluster.
- Horizontal Pod Autoscaling (HPA) – HPA automatically adjusts the number of replica pods in a deployment, ensuring responsiveness to workload fluctuations and optimal resource allocation.

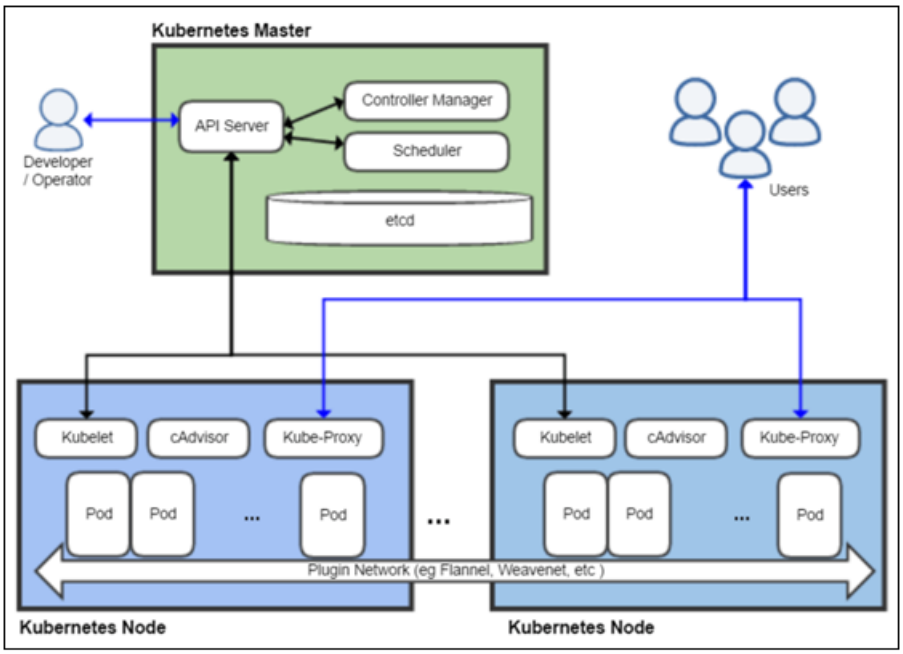

Figure 6: Kubernetes Architecture Diagram

3.4 Airflow Integration

Apache Airflow will be deployed as a set of Kubernetes pods, each representing a different component, including the scheduler, webserver, and worker nodes. The key components of the Airflow include DAGs, operators, schedulers, and executors. Apache Airflow DAGs will be written in Python using the Airflow API. Similarly, tasks will be implemented as Python operators, which can execute data processing logic, interact with external sources, and handle data discrepancies. Configuring the Airflow architecture as a code (Python) allows for dynamic pipeline generation [14]. This will allow the respective operators to write code that instantiates data pipelines dynamically. In the implementation stage, TechShop will use a distributed Airflow Architecture, meaning that the DAG files will be synchronized between all the components that use them – workers, triggered, and scheduler. The greatest strength of Apache Airflow is its flexibility, offering easy extensibility through its plug-in framework [15]. Additionally, Apache Airflow provides a wide range of integrations for services on various cloud providers.

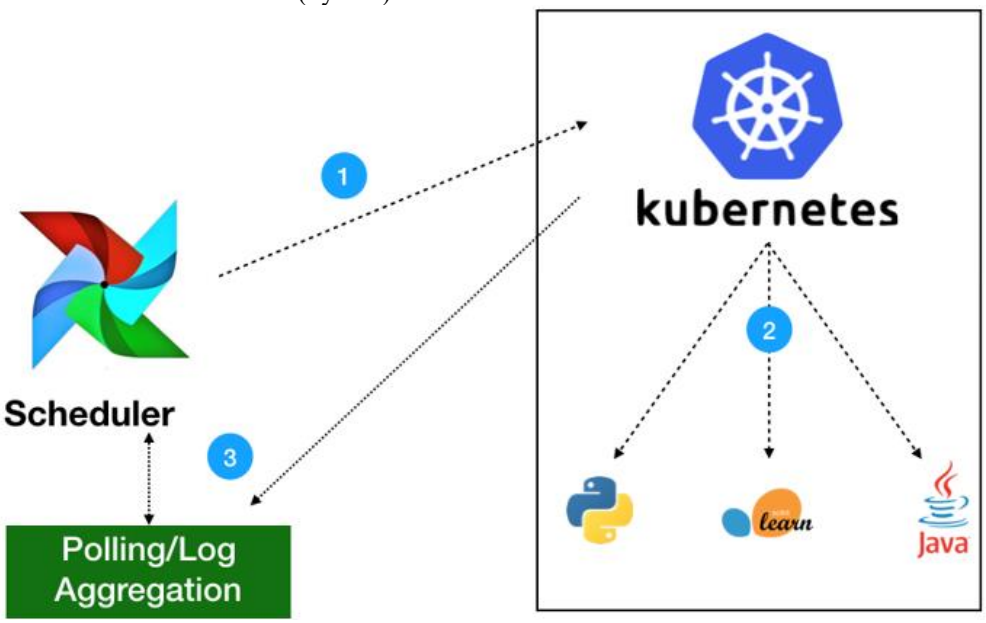

Figure 7: Airflow integration into Kubernetes

TechShop will leverage Kubernetes manifests to define the desired states of the Apache Airflow deployment and manage its lifecycle, as shown in Figure 7. The Kubernetes Operator will use Python to generate a request that will be processed by

the API server. Subsequently, Kubernetes will launch TechShop's pods based on the defined specs, enabling the collected data to be loaded under a single command. Once the tasks are launched, the operators will only need to track logs while gathering logs for the scheduler or any other distributed logging service within the Kubernetes cluster. The integration of Kubernetes and Airflow will enable TechShop to leverage the scalability and flexibility of Kubernetes for processing big data workflows. Kubernetes' auto-scaling capabilities will allow the cluster to dynamically adjust resource allocation based on workload demands, ensuring optimal performance during peak periods.

4. Conclusion

In summary, Cloud-native technologies involve building, deploying, and managing modern applications that leverage the distributed computing capabilities offered in cloud delivery models. This means that cloud-native services, such as Kubernetes and Airflow, are designed and built to exploit the scale, resiliency, flexibility, and elasticity of cloud computing environments. In this case study, we deploy Kubernetes clusters on the Google Cloud Platform to leverage its scalability and manage services. Meanwhile, Airflow is the orchestration layer for managing data processing workflows within the Kubernetes environments. Kubernetes' auto-scaling capabilities ensure optimal resource utilization while Airflow orchestrates the execution of concurrent workflows.

References

[1] Anagnostopoulos, I., Zeadally, S., & Esposito, E. (2016, February). Handling big data: research challenges and future directions. *The Journal of Supercomputing*, *72*, 1494-1516. https://link.springer.com/article/10.1007/s11227-016-1677-z

[2] Barika, M., Garg, S., Zomaya, A. Y., Wang, L., Moorsel, A. V., & Ranjan, R. (2019, September). Orchestrating big data analysis workflows in the cloud: research challenges, survey, and future directions. *ACM Computing Surveys (CSUR)*, *52*(5), 1-41. https://doi.org/10.1145/3332301

[3] Hammi, B., Khatoun, R., Zeadally, S., Fayad, A., & Khoukhi, L. (2018, January). IoT technologies<? show [AQ ID= Q1]?> for smart cities. *IET networks*, *7*(1), 1-13. https://doi.org/10.1049/iet-net.2017.0163

[4] Kuss, D. J., & Lopez-Fernandez, O. (2016, March). Internet addiction and problematic Internet use: A systematic review of clinical research. *World journal of psychiatry*, *6*(1), 143. https://doi.org/10.5498%2Fwjp.v6.i1.143

[5] Laszewski, T., Arora, K., Farr, E., & Zonooz, P. (2018, August). *Cloud Native Architectures: Design high-availability and cost-effective applications for the cloud*. Packt Publishing Ltd.

[6] Jakóbczyk, M. T., & Jakóbczyk, M. T. (2020, February). Cloud-Native Architecture. *Practical Oracle Cloud Infrastructure: Infrastructure as a Service, Autonomous Database, Managed Kubernetes, and Serverless*, 487-551.

[7] Toffetti, G., Brunner, S., Blöchlinger, M., Spillner, J., & Bohnert, T. M. (2017, July). Self-managing cloud-native applications: Design, implementation, and experience. *Future Generation Computer Systems*, *72*, 165-179. https://doi.org/10.1016/j.future.2016.09.002

[8] Sayfan, G. (2017, May). *Mastering Kubernetes*. Packt Publishing Ltd.

[9] Koutoulakis, E. (2020, September). Implementation of a federated workflow execution engine for life sciences through virtualization services. https://apothesis.lib.hmu.gr/bitstream/handle/20.500.12688/9638/KoutoulakisEmmanouil2020.pdf?sequence=1&isAllowed=y

[10] Panhalkar, S. (2019, January). Libflow: A Platform to Schedule and Manage Workflows Using DAGs.

[11] Palazzo, C., Mariello, A., Fiore, S., D'Anca, A., Elia, D., Williams, D. N., & Aloisio, G. (2015, July). A workflow-enabled big data analytics software stack for eScience. In *2015 International Conference on High Performance Computing & Simulation (HPCS)* (pp. 545-552). IEEE. https://doi.org/10.1109/HPCSim.2015.7237088

[12] Larsson, L., Tärneberg, W., Klein, C., Elmroth, E., & Kihl, M. (2020, August). Impact of etcd deployment on Kubernetes, Istio, and application performance. *Software: Practice and Experience*, *50*(10), 1986-2007. https://doi/org/10.1002/spe.2885

[13] Calcote, L., & Butcher, Z. (2019, October). *Istio: Up and running: Using a service mesh to connect, secure, control, and observe*. O'Reilly Media.

[14] Shubha, B., & Prasad, A. (2019, June). Airflow directed acyclic graph. *J Signal Process*, *5*(2). https://core.ac.uk/download/pdf/230490858.pdf

[15] Hummer, W., Muthusamy, V., Rausch, T., Dube, P., El Maghraoui, K., Murthi, A., & Oum, P. (2019, June). Modelops: Cloud-based lifecycle management for reliable and trusted AI. In *2019 IEEE International Conference on Cloud Engineering (IC2E)* (pp. 113-120). IEEE. https://doi.org/10.1109/IC2E.2019.00025