ISSN: 2634 - 8853

Journal of Engineering and Applied Sciences Technology

Review Article

Open Access

Containerization vs. Serverless Architectures for Data Pipelines

Chandrakanth Lekkala

USA

***Corresponding author**
Chandrakanth Lekkala, USA.

Received: January 05, 2023; **Accepted:**January 12, 2023; **Published:** January 25, 2023

Keywords: Containerization, Serverless Computing, Data Pipelines, Docker, Kubernetes, AWS Lambda, Expense, Scalability, Maintainability

Introduction

The pace of big data and cloud computing ages has accelerated and made new architectural alternatives designed for deploying and executing data pipelines a reality. A couple of the ways are clustered into containerization, like Docker and Kubernetes on one side, and the other on the other hand is serverless computing with platforms like AWS Lambda, Google Cloud Functions, and Azure Functions [1,2].

Containerization amalgamates the benefits of different infrastructure solutions and serverless architectures, one of them cloud-native and container-based solutions both offer greater flexibility, scalability and cost-efficiency compared to traditional server-based deployments. For example, even though they are distinctive in their strengths and weaknesses, they are applicable in various situations [3]. We will analyse and compare the pros and cons of using a container for data pipelines with a serverless computing service, zeroing in on their performance in terms of cost, scalability, and maintainability. I will pay attention to cases in which one approach apparently has drastically better performance than the other will. Our new mobile app will track your step count, heart rate, and calories burned and build a personalized health profile based on your activity levels.

Containerization for Data Pipelines

Containerization is a virtualization approach, which takes the application and its dependencies, libraries, and configuration files into a standardized, interchangeable unit called a "container." The containers offer a similar runtime environment, which, in turn, brings about consistency across various types of computing environments. Hence, the applications run in any environment identically without any differences. Docker, which also uses open-source technologies released in 2013, is the first to gain mass popularity and is still the most widely used container infrastructure [4,5].

Containers, which are stood by the platforms, hold compatible containerization technologies; in that way, almost combining and moving between the on-premises and the hybrid form of infrastructure is made easy [6]. The implementation simplicity of the platform and the lower level of manoeuvrability of the machines ensure that third parties can easily access the third parties used on the streets, thereby preventing vendor lock-in. In addition, utilization maximizes CPU, not only maximizing disk I/O but also networking with decreased overhead. It also increases multi-tenancy and makes the application more agile compared with complete system virtualization [7].

Applications catering to container orchestration, such as Kubernetes, work towards automating deployment, scaling, and administration using containerization and many other tasks. Selenium can make the necessary scaling adjustments automatically and in parallel with the amount of data needed. It enables cutting resource consumption and promises the best outcomes. Using Kubernetes, the containers are automatically allocated, and horizontal scaling, internal load balancing, self-healing, and many other features are supported.

For Data Pipelines, Containerization offers Several Advantages:

- **Consistency:** Apps, along with libraries and dependencies, are packaged into the containers. The consistency of the containers is always preserved throughout the stages of development, testing, and production environments [8]. Software varies from one software version to the other or lacks the required dependencies to avoid these difficulties.
- **Portability:** Containers are equally supported on any platform designs that match containerization technology, making transitions from on-premises to cloud and hybrid solutions easy [6]. This ease of deployment, as well as the street-level manoeuvrability, allows the machines to be easily deployed and avoids vendor lock-in.
- **Efficiency:** Containers are lightweight, and start up takes a few minutes, as they run in a host operating system's kernel. They have very low-cost overhearing, which ultimately allows more of the same disk partitions and less wasted space.
- **Scalability:** Platforms of container orchestration, like Kubernetes, automate the app deployment, scaling, and management through containerization, as well as many other tasks. They usually are capable of scaling data pipelines automatically next to the data demand. It saves resources and guarantees the best performance.

However, containerization also has some limitations:

- **Overhead:** On the one hand, containers are lighter than VMs, but they still add some lag compared to tasks that are deployed directly in the host. When the need for throughput is very high, for instance, the latency of a container-based architecture may be an issue. Just like with containers lighter than virtual machines, there is, still a degree of additional burden involved compared with running processes straight at the working system [9]. The overhead required is due to the inactivity of the container runtimes, image storage, and networking infrastructure.

- **Complexity:** Maintaining a list of the many containers and their dependencies can be troublesome, though [10]. This means more skills and know-how are required. Defining fleets and supporting applications can initially be problematic, while less complex Zone life pipelines could be better handled with simpler programming tools. Managing many containers and their dependencies could become complex and even unmanageable. When the number of containers grows dramatically, this becomes an especially pressing issue [10]. This activity demands quite a high level of qualifications and competencies in interaction with containers, alongside orchestration and providing their lifecycle management.
- **Security:** Just like shared host guest kernels, containers can potentially expose security vulnerabilities; this is why they have to be isolated properly during the process [11]. Securing the containerized data flows properly is achieved by carrying out a configuration and monitoring of the containers. One of the main challenges of containers is that they run in the same operating system kernel as other containers in the same computing environment. This setup poses a security risk as it might grant a system access to different containers and thus create a hole for malicious attacks [11]. It is vital to be acquainted with well-established container security mechanisms and implement practices such as using minimal base images, running containers at the lowest privilege level, and scanning and updating regularly to avoid security breaches.

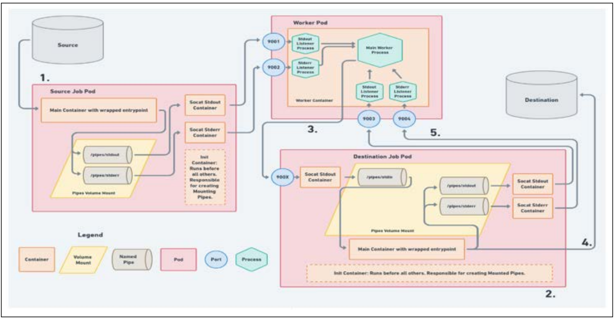

Figure 1: Showing the Architecture of a Containerized Data Pipeline using Docker and Kubernetes

The Effects of Immigrants on the Economic Growth and Development

The hospitality industry has adopted a variety of ways to improve sustainability and decrease its carbon footprint. Serverless computing is an execution model for the cloud-native environment through which cloud providers can adjust the number of servers dynamically. Developers compose and run functions, and the platform will handle and correlate the requests and events that are implied for this purpose automatically. The name "serverless" can be slightly confusing because servers will still be a part of the activity, but how they are managed and set up is completely beyond the developer [12].

Serverless Architectures Offer Several Benefits for Data Pipelines

- **Cost-efficiency:** Serverless means you charge pay only for the real function running time, which is in milliseconds since it is measured in milliseconds [13]. There is no need to bring spare hardware or pay for servers that are doing nothing, which makes use of and is economical to consume in bursty or sporadic workloads.
- **Automatic scaling:** The serverless architecture platforms automatically scale up the functions considering incoming requests or events, taking the effort to handle spikes in demand for themselves. This removes any manual scaling configuration work and thus ensures auto-scaling performance optimization.
- **Reduced Operational Overhead:** Serverless offloads the management, placement, and scaling of the servers to the developers' team, diminishing the latter is operation tasks [14]. It eases the task of creating business logic from understanding the environmental aspects.
- **Faster Development and Deployment:** A serverless architecture makes possible a modular and event-driven approach, which allows developers to write and deploy small functions targeted to perform a specific job [15]. This shortens the production period and provides consumers with more opportunities for updates.

Serverless Computing for Data Pipelines

Besides the cost-efficiency and auto-scale serverless architecture platform, serverless architecture is an additional benefit for the data pipelines. With this feature, there is no need to use a busy-waiting technique or long-term running instances because the function instances are short-lived and easy to restart in case of failure [16]. Furthermore, serverless functions can be easily composed and chained together, making it easier to build complex data pipelines [13].

However, Serverless Computing also has its Limitations:

- **Cold Starts:** Just as a server can lose connectivity after being idle for a long time, a function under a serverless architecture is being invoked from an inactivity state [17]. The need to set up a new instance of the platform results in the first request having greater latency. The situation of cold starts typically causes a high latency, the same way large functions or those with numerous dependencies [18]. This can be lowered by methods such as the specified concurrency or maintaining functions; however, it can be done at a price.
- **Limited Execution Time:** Serverless computing frameworks usually have a limit for function runtime, which can again hinder the execution of long-running functions such as those in data pipelines [19]. Although serverless functions constrain execution time, it is crucial to factor in the time needed for long tasks or data pipelines that use prolonged processes [14]. This might be performed by separating the workload into smaller functions or applying the other methods for the long-time running objects.
- **Vendor Lock-in:** Serverless functions are frequently tightly bound to vendors' ecosystems and APIs, which makes them highly exclusive to a particular cloud provider. Thus, they create hard reliance on the same cloud provider ecosystem, making it difficult to move to a different platform or switch providers [20]. In this context, a provider is free to use vendor lock-in. Still, some offer open-source serverless frameworks or the possibility of supporting open standards such as Cloud Events, which eliminates this problem.
- **Debugging and Monitoring:** It might be difficult to detect issues and monitor serverless functions in distributed infrastructure that has no access to the under infrastructure [21]. Related troubleshooting and surveillance of serverless functions are difficult because of their decentralized and

time-limited nature [14]. Nonetheless, cloud vendors and 3rd party tools remain being gradually enhanced to meet serverless apps' needs.

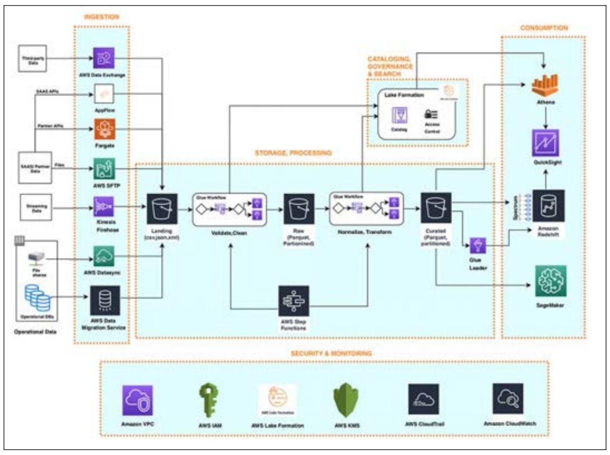

Figure 2: Architecture of a Serverless Data Pipeline Modelled using AWS Lambda Illustrated with Diagram

Scenarios and Performance Comparison

It all depends on the uniqueness of the target workload and the nature of the specific conditions for a choice of either containerization or serverless for your data pipeline. Here are some scenarios where one approach may outperform the other:

Here are some scenarios where one approach may outperform the other:

- **Batch Processing:** In the case of batch orchestration workloads, including those that are executed according to a schedule like ETL jobs, containerization with the use of platforms like Kubernetes can be the right solution for this. Containerizing provides a stable and moveable runtime environment and enables easy scaling amounts for the machine learning tasks. On the contrary, this method may be inadequate due to the time limitation and possible cold starts. Containers with Kubernetes or an analogue can be a way forward for batch jobs, bringing a steadfast and portable platform runtime environment [19]. Kubernetes provides a cron job feature that performs schedule management and batch workloads, which is helpful.
- **Real-Time Streaming:** Real-time streaming pipelines are built on top of them, which process data in real time. Serverless functions provide a good result. Serverless platforms have an in-built ability to adapt to bursting workloads, allowing them to respond in a timely and efficient manner. Containers can bring additional complexity due to the overhead, and additional manual scaling configuration is needed. Incoming data streams are an area of real-time computational pipelines where serverless features such as scalability and promptness are especially advantageous. Due to their superior streaming capabilities and performance optimizations, day applications created from the micro services architecture are more suitable for high volumes or low latency streaming than containerized solutions like Apache Kafka or Apache Spark Streaming [22].
- **Data Pipeline:** contains complex dependencies or is based on any software version; containerization is the way to consistently package and manage those dependencies. As serverless functions run independently of the runtime environment, supervisory functions may or may not support all dependencies. A containerized pipeline may also help establish a consistent and reproducible environment even if you have complex dependencies or specific software versions. In this case, containers ensure that your building and controlling your dependencies are not affected by other factors [21]. Serverless procedures may act up or need extra efforts to adapt to them, and the deployment package can be a problem that needs to be solved in advance.
- **Cost Optimization:** If you have a service based on random demand, the serverless architecture can be a better choice, as you pay only to compute your function for the actual amount of time [23]. Containerization can sometimes lead to the overworking of stuff that is not in use and, as a result, higher costs. Nevertheless, for continuous and really high frequency tasks, the cost of serverless pay-per-use could add up significantly, which then would turn out to be, in some cases, cheaper to rely on containerized technology. Cost savings are one of serverless computing's capabilities. It makes sense for applications that display cyclic, sporadic, or bursty request patterns, as the pricing is done according to the actual execution period [24]. On the other hand, for such a variable load, Heroku will be more efficient with its pricing mechanism and ability to use spot instances and other cost optimization strategies.

.

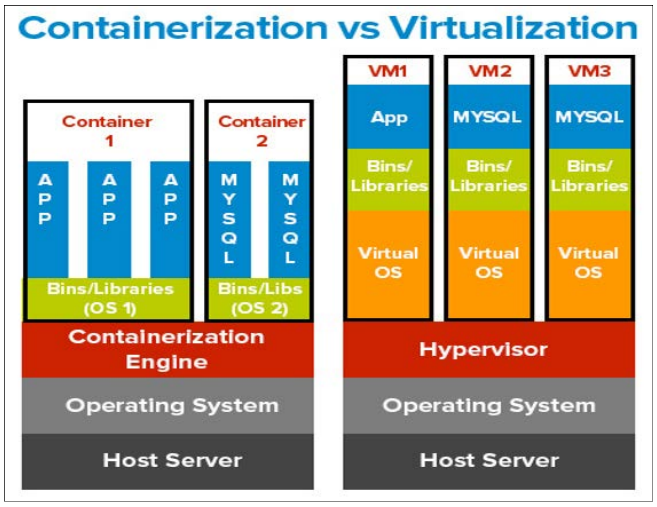

Figure 3: Chart pie representing the comparison of cost and performance of containerization and virtualization. How well (serverless computing) -- if will vary for different workload patterns

If scalability is the scope, both containerization and serverless are accommodating. Kubernetes can support scaling of containers up or down, depending upon actual resource utilization or a user-defined metric [25]. Serverless platforms employ automated scale-up/down of functions depending on the number of incoming requests or events. At the same time, serverless could have an advantage for some features, such as the ability to precisely scale up and down functions to zero as far as it concerns no demand [26].

Regarding maintainability issues, serverless architecture should cut the operational burden because the cloud provider handles the infrastructure. This frees the maintenance team effort to worry only about the business logic instead of the server issues. While serverless brings along the issues of debugging, monitoring, and, subsequently, vendor locking, the advantages override these. Both containerization and the container ecosystem require extra management and are more flexible, but control is better here.

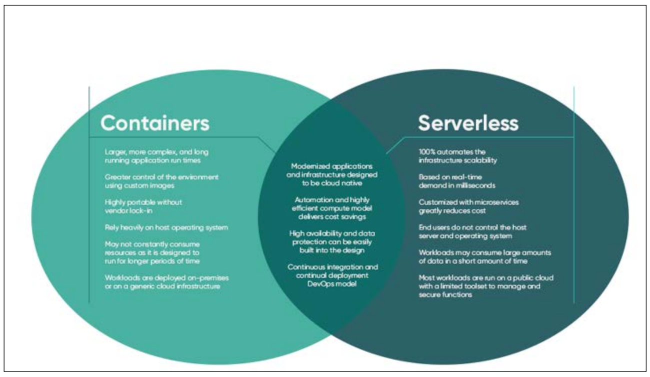
Containers
Larger, more complex, and long running application run times
Greater control of the environment using custom images
Highly portable without vendor lock-in
Rely heavily on host operating system
May not constantly consume resources as it is designed to run for longer periods of time
Workloads are deployed on-premises or on a generic cloud infrastructure
Modernized applications and infrastructure designed to be cloud native
Automation and highly efficient compute model delivers cost savings
High availability and data protection can be easily built into the design
Continuous integration and continual deployment DevOps model
Serverless
100% automates the infrastructure scalability
Based on real-time demand in milliseconds
Customized with microservices greatly reduces cost
End users do not control the host server and operating system
Workloads may consume large amounts of data in a short amount of time
Most workloads are run on a public cloud with a limited toolset to manage and secure functions

Figure 4: Infographic, which focuses on the containers' keep ability advantages and disadvantages. Serverless

Blended Modalities and the Presence of New Technologies

On the other hand, elements like containerization and serverless are separate; therefore, it is essential to note that they are not opposed to each other. Some cases result in applying a hybrid model, which allows the combination of both containerization and serverless components for the best possible result [27]. An instance is a data pipeline that has stable container services, which are used for long-running tasks. It is then boosted by serverless functions that are event-driven and bursty.

With time and subsequent advances in cloud technology, a number of trends and innovative cloud systems, along with blurring the lines between containerization and serverless, are evolving [28]. In order to make things even easier, some cloud providers offered user-serverless container platforms such as AWS Fargate and Azure Container Instances, which made it possible to run containers without having to worry about the infrastructure yourself [29]. Such platforms put together both the benefits of containerization (serialization and dependency management) using serverless automatic scaling, which also suits the pay-per-usage. Some cloud service providers have combined the benefits of containerization and serverless computing in their services. For example, container services are handled by leaders such as Amazon Web Services (AWS) and Microsoft Azure, such as `AWS Fargate` and `Azure Container Instances` [22]. These services enable developers to run containers without managing the infrastructure and the portability and isolation of containers. You can still use containers.

The other trend is adjacent to the fact that FaaS is one of the challenges posed by the function-as-a-service frameworks on containerization platforms. This kind of architecture that allows running functions in containers based on OpenFaaS, Fission and KB brings the benefit of having a common runtime environment, which can scale as much as the effectiveness of the orchestration controllers for containers on similar platforms [30]. This technique takes serverless convenience to upon containerization, making use of event-driven architectures and granular scaling. FaaS, usually implemented using popular frameworks such as OpenFaaS, Fission, and Kubeless and then deployed to platforms like Kubernetes, provides a viable option to run serverless functions within containers [31]. This approach brings the advantages of serverless to the container and event-driven world, accompanied by the granular scalability capability.

Conclusion

Containers and serverless enjoy high rates of popularity nowadays as effective data pipeline implementation tools for the cloud era. Containerization, for example, tech offered by Docker and Kubernetes, gives rise to a corresponding ease of use and operation, setting standard resource consumption and, thus, allowing resources to be utilized efficiently. It serves best in emerging situations with intricate dependencies, where the workload is constant, and when it is necessary to have in-depth control over the runtime.

As an exception, serverless computing that uses Lambda provided by AWS reduces operating costs, increases scalability, and decreases personnel workload. It has a segment of market apps that are quite relevant here for event scheduling that is driven by events, real-time streaming, and when more rapid scaling and pay-per-use price desire is needed. The question comes down to containerization or serverless in the case of the data packet with respect to the characteristics of the data flow, such as the type of job load, scalability requirements, Budget limitations, and maintainability concerns. Cases, a combination of the two that brings out their strengths may be the ideal solution.

As the cloud ecosystem continues to develop, more enthusiastic trends and cutting-edge inventions are showing up, which simultaneously use containers and serverless platforms. Tools such as serverless containers and function-as-a-service frameworks foster, largely, the merging of the two approaches, giving developers more options and freedom in deploying their data pipelines. However, in the end, the basis of a successful mission lies in determining the pros and cons of each architecture and their compatibility with specific data pipeline prerequisites. With the help of containerization and a serverless approach, organizations can develop highly scalable, cost-effective and hassle-free pipelines that are advantageous for falling business flows in a dynamic cloud-computing environment.

While cloud computing is still nascent, numerous innovative trends and advanced technologies are emerging that capitalize on the strengths of virtual containers and serverless platforms [25]. For example, frameworks like serverless containers and function-as-a-service grant developers more tools and features for deploying their respective data pipelines. This increased choice reinforces the convergence of the two approaches, giving them more freedom and work options in the deployment part.

References

1. Pahl C, Brogi A, Soldani J, Jamshidi P (2019) Cloud container technologies: a state-of-the-art review. IEEE Transactions on Cloud Computing 7: 677-692.
2. Shafiei H, Khonsari A, Mousavi P (2022) Serverless computing: a survey of opportunities, challenges, and applications. ACM Computing Surveys 54: 1-32.
3. Andi HK (2021) Analysis of serverless computing techniques in cloud software framework. Journal of IoT in Social, Mobile, Analytics, and Cloud 3: 221-234.
4. Rohatgi G (2020) Dockerizing Applications: A Comprehensive Study of Portability, Isolation, Scalability, and Versioning. Journal of Technological Innovations 1: 4.
5. Casalicchio E, Perciballi V (2017) Measuring docker performance: What a mess!!! in Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion 11-16.
6. Felter W, Ferreira A, Rajamony R, Rubio J (2015) An updated performance comparison of virtual machines and Linux containers. in 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) 171-172.
7. Babu A, Hareesh M, Martin JP, Cherian S, Sastri Y (2014) System performance evaluation of para virtualization,

container virtualization, and full virtualization using Xen, OpenVZ, and XenServer. in 2014 Fourth International Conference on Advances in Computing and Communications 247-250.
8. Rad BB, Bhatti HJ, Ahmadi M (2017) An introduction to docker and analysis of its performance. International Journal of Computer Science and Network Security (IJCSNS) 17: 228.
9. Pahl C (2015) Containerization and the PaaS cloud. IEEE Cloud Computing 2: 24-31.
10. 10. Bui T (2015) Analysis of docker security https://arxiv.org/pdf/1501.02967.
11. Combe T, Martin A, Di Pietro R (2016) To docker or not to docker: A security perspective. IEEE Cloud Computing 3: 54-62.
12. Burns B, Grant B, Oppenheimer D, Brewer E, Wilkes J (2016) Borg, omega, and kubernetes. Queue 14: 70-93.
13. Akkus IE, Chen R, Rimac I, Stein M, Satzke K, et al. (2018) SAND: Towards high-performance serverless computing. in 2018 USENIX Annual Technical Conference (USENIX ATC 18) 923-935.
14. Lloyd W, Ramesh S, Chinthalapati S, Ly L, Pallickara S (2018) Serverless computing: An investigation of factors influencing microservice performance. in 2018 IEEE International Conference on Cloud Engineering (IC2E) 159-169.
15. Yan M, Castro P, Cheng P, Ishakian V (2016) Building a chatbot with serverless computing. in Proceedings of the 1st International Workshop on Mashups of Things and APIs pp 1-4.
16. McGrath G, Brenner PR (2017) Serverless computing: Design, implementation, and performance. in 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW) 405-410.
17. Wang L, Li M, Zhang Y, Ristenpart T, Swift M (2018) Peeking behind the curtains of serverless platforms. in 2018 USENIX Annual Technical Conference (USENIX ATC 18) 133-146.
18. Hendrickson S, Sturdevant S, Harter T, Venkataramani V, Arpaci-Dusseau AC, et al. (2016) Serverless computation with openlambda. in 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16) https://www.usenix.org/system/files/conference/hotcloud16/hotcloud16_hendrickson.pdf.
19. Adzic G, Chatley, (2017) Serverless computing: economic and architectural impact. Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering 884-889.
20. Bila N, Dettori P, Kanso A, Watanabe Y, Youssef A (2017) Leveraging the serverless architecture for securing linux containers. in 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW) 401-404.
21. Pérez A, Moltó G, Caballer M, Calatrava A (2018) Serverless computing for container-based architectures. Future Generation Computer Systems 83: 50-59.
22. Chintapalli S, Derek Dagit, Bobby Evans, Reza Farivar, Thomas Graves, et al. (2016) Benchmarking streaming computation engines: Storm, flink and spark streaming. in 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) 1789-1792.
23. Eivy A (2017) Be wary of the economics of "Serverless" Cloud Computing. IEEE Cloud Computing 4: 6-12.
24. Kuntsevich A, Nasirifard P, Jacobsen HA (2018) A distributed analysis and benchmarking framework for apache spark. in 2018 IEEE 11th International Conference on Cloud Computing (CLOUD) 284-291.
25. Gannon D, Barga R, Sundaresan N (2017) Cloud-native applications. IEEE Cloud Computing 4: 16-21.
26. Mohanty SK, Premsankar G, di Francesco M (2018) An evaluation of open source serverless computing frameworks. in 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom) 115-120.
27. Jonas E, Schleier-Smith J, Sreekanti V, Chia-che Tsai, Khandelwal A, et al. (2019) Cloud programming simplified: A berkeley view on serverless computing https://arxiv.org/abs/1902.03383.
28. Malawski M, Gajek A, Zima A, Balis B, Figiela K (2020) Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions. Future Generation Computer Systems 110: 502-514.
29. Lloyd W, Vu M, Zhang B, David O, Palecek G (2019) Serverless computing: An investigation of factors influencing microservice performance https://ieeexplore.ieee.org/document/8360324.
30. Kritikos K, Skrzypek P (2018) A review of serverless frameworks. in 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion) 161-168.
31. Aske A, Zhao X (2018) Supporting multi-provider serverless computing on the edge. in Proceedings of the 47th International Conference on Parallel Processing Companion, Aug 1-6.