ISSN: 2754-6659

Journal of Artificial Intelligence & Cloud Computing

SCIENTIFIC Research and Community

Review Article

Open Access

Deploying and Managing Containerized Data Workloads on Amazon EKS

Chandrakanth Lekkala

USA

ABSTRACT

In the age of big data and cloud computing, many organizations are moving towards containerization technologies to deploy and manage their data workloads. Amazon Elastic Kubernetes Service (EKS) has recently become a trendy option for using container applications in the cloud. This paper discusses the procedure of deploying and managing containerized data workloads on Amazon EKS. It points out the benefits of containerization in terms of portability, scalability, and consistency of the workload in different environments. The research also looks into the best practices for creating and managing data-intensive applications on EKS, such as data storage, networking, and security issues. Besides, it gives a realistic case of administering a Kafka cluster and streaming data in EKS. This paper is designed to provide important tips and advice to organizations that want to use containerization and Amazon EKS for their data workloads.

***Corresponding author**

Chandrakanth Lekkala, USA.

Received: June 05, 2023; **Accepted:** June 09, 2023; **Published:** June 16, 2023

Keywords: Containerization, Amazon EKS, Data Workloads, Kafka, Streaming Data, Scalability, Portability, Cloud Computing

Introduction

The use of containerization technologies has changed the method of cloud application deployment and management. Containers are lightweight and portable runtime environments that enable applications to run similarly in all computing environments [1]. EKS Amazon Elastic Kubernetes Service (EKS) has become a popular option for running containerized applications in the cloud, providing an easy-to-use managed Kubernetes service that simplifies the deployment and management of containerized workloads [2].

Data-intensive applications, such as big data processing, machine learning, and real-time analytics, will greatly benefit from the scalability and flexibility provided by containerization and Amazon EKS [3]. Nevertheless, the installation and management of containerized data workloads on EKS is challenging. It requires carefully observing different issues, such as data storage, networking, security, and performance optimization [4].

In this paper, it examines the procedure for rolling out and controlling containerized data workloads on Amazon EKS. It discusses the benefits of containerization in terms of portability, scalability, and the same conditions in different environments. Besides, it also examines the best practices for designing and running data-heavy applications on EKS by referring to data storage, networking, and security. Here, it illustrates the case study of managing a Kafka cluster and streaming data in EKS, which is a viable option.

Containerization and its Benefits

Containerization is one of the most preferred solutions for packaging and deploying applications in a portable and efficient way [5]. Containers are a lightweight and isolated runtime environment that wraps an application, its libraries, and configuration files and, thus, creates an isolated application environment [6]. By containerizing applications, organizations can achieve several benefits, including: By containerizing applications, organizations can achieve several benefits, including:

- Portability Containers are how the applications can be packaged in a self-contained manner so they can be easily transported to different computing environments [7]. Containerized applications are easily transferred from this part of deployment to testing and then to production with the help of various cloud platforms or the on-premises infrastructure [8]. The portability of mobile devices helps prevent the occurrence of compatibility problems, and it guarantees the same performance of applications in any environment.
- Scalability is possible through containerization, by which applications can be easily scaled horizontally by running several instances of the same container [9]. Containers are light and can be easily allocated or stopped when needed. Thus, organizations can dynamically modify their application capacity to accommodate varying workloads [10]. This growth is especially useful for data-intensive applications that are subject to fluctuations in demand or need to deal with large amounts of data.
- Consistency Containers are the run time environments for applications that are similar everywhere, meaning that the application runs the same way in different environments [11]. The “it works on my computer” issue is solved by packing an application with its dependencies in a container, where applications perform differently on different machines

because of the differences in the underlying infrastructure or runtime dependencies [12]. The consistency in this system of the application development, testing, and deployment processes makes it easier to achieve, reducing the number of errors and the time it takes to be ready to be launched on the market.

Amazon Elastic Kubernetes Service (EKS)

Amazon EKS is a completely managed Kubernetes service that Amazon Web Services (AWS) offers to users [13]. The EKS enables the easy deployment, managing and scaling of containerized applications using the Kubernetes, an open-source container orchestration platform [14]. Some of the key features of Amazon EKS include:

Managed Kubernetes Control Plane

EKS provides a fully managed Kubernetes control plane, which is responsible for the overall management and orchestration of containers [15]. The control plane is broadly accessible and automatically scales up to guarantee the dependability and steadiness of the Kubernetes cluster [16]. This controlled plane of management of the Kubernetes infrastructure reduced the operational overhead connected to managing the Kubernetes infrastructure, allowing organizations to concentrate on their application development and deployment.

EKS is easily connected with other AWS services, making AWS the perfect, complex ecosystem for containerized workloads [17]. EKS is a service that can easily integrate with others, such as Amazon Elastic Container Registry (ECR) for storing container images, Amazon Elastic Block Store (EBS) for making persistent storage and Amazon Virtual Private Cloud (VPC) for networking [18]. This fusion of EKS and AWS eliminates the need for complex deployment and management of containers, as organizations can use the familiar AWS services and tools [19].

EKS is a scalable and highly available solution for containerized applications. It instructs the system to scale the worker nodes according to the workload demands automatically. Hence, the cluster has the resources to deal with the increased traffic or processing needs [20]. Moreover, EKS splits the containers into several Availability Zones (AZs) to increase the applications' resilience and fault tolerance [21].

Best Practices for Deploying and Managing Containerized Data

The Best Practices for Deploying and Managing Containerized Data Workloads on EKS Combining the aspects of various factors that must be considered is the main topic of deployment and management of containerized data workloads on EKS. In the given section, it discusses some good practices for the architecture and operation of data-intensive applications on EKS.

Data Storage Selecting

The appropriate data storage solution for containerized data workloads on EKS is vital [22]. EKS can cooperate with different storage types, such as Amazon Elastic Block Store (EBS) for durable storage, Amazon Elastic File System (EFS) for file sharing, and Amazon S3 for object storage [23]. The selection of the storage is based on the particularity of the workload, for example, the type of data access, the performance demands, and the scalability [24].

For crateful applications that are based on the principle of storage persistence, such as databases or message queues, EBS volumes can provide dedicated storage for each container instance [25]. EBS volumes can be dynamically extended and attached to containers, guaranteeing data persistence across container restarts or failures [26].

The top use of EFS is for applications that need shared file storage that can be reached from different containers. EFS can be used for that [27]. EFS is a scalable and fully managed file system that can be mounted at the same time by several containers, which thus leads to easy data sharing and collaboration [28].

Amazon S3 can be the choice of object storage solution for huge data processing and analysis workloads because of its scalability and cost-effectiveness [29]. Containers can read from and write to S3 buckets, thus utilizing S3's high availability and durability for storing and accessing big data [30].

Networking

Networking is one of the most crucial aspects of the EKS when deploying containerized data workloads [31]. On the other hand, Kubernetes uses the Amazon Virtual Private Cloud (VPC) for networking. Thus, organizations can define their network configurations and control traffic [32]. Some of the networking best practices for EKS include:

- **Using VPC for Network Isolation:** Each EKS cluster must be deployed in a separate VPC to maintain network isolation and security [33]. This way, the organizations can manage network access and impose policies at the VPC level.
- **Configuring Network Policies:** Kubernetes network policies are tools that can help regulate the traffic between the containers in the cluster [34]. The network policies regulate the interaction between the containers that can communicate with each other, thereby significantly contributing to the enhancement of security and reduction of the attack surface.
- **Leveraging Load Balancers:** EKS supports the use of Kubernetes load balancers such as the AWS Network Load Balancer (NLB) or the AWS Application Load Balancer (ALB) in the case of exposing containerized services externally [35]. Load balancers are responsible for allocating incoming traffic to the various container instances, which ensures high availability and scalability.

Security

Security of the containerized data workloads on EKS is vital to data protection and the prevention of unauthorized access [36]. Some of the security best practices for EKS include:

- **Implementing Role-Based Access Control (RBAC):** EKS enables the Kubernetes RBAC system, through which organizations can set access rights for the users and applications within the clusters [37]. RBAC guarantees that the users and applications have the least privilege needed to carry out their tasks, which in turn minimizes the chances of unauthorized access.
- **Securing Container Images:** Before using a container image, one must be sure that it is scanned for vulnerabilities and built from trusted sources [38]. Amazon ECR has image-scanning features that can help detect possible security threats in container images [39]. Besides, companies should insist on policies that guarantee that only authorized and screened container images will be allowed in the EKS cluster.
- **Encrypting Data at Rest and in Transit:** Data encryption is needed to protect sensitive information [40]. EKS supports data encryption at rest using Amazon EBS or Amazon EFS encryption [41]. On the other hand, for data in transit, TLS/SSL encryption should be used to secure communication

between containers and with external services.

Practical Example

The deployment and management of Kafka Cluster and Data Stream in EKS

To illustrate the application and the handling of containerized data workloads on EKS, let me take a real example of managing a Kafka Cluster and Data Stream in EKS.

Kafka is a distributed streaming platform that allows the publishing and subscribing of real-time data streams [42]. The tool is commonly used for constructing data pipelines, real-time analytics, and event-driven architectures, as shown by its wide use for these purposes [43]. In this example it illustrates how to set up a Kafka cluster on EKS and show data streaming using Kafka.

Deploying a Kafka Cluster on EKS

Deploying a Kafka cluster on EKS can be done by using the Strimzi Kafka Operator, which makes the deployment and management of Kafka in Kubernetes simple [44]. The Strimzi Kafka Operator gives a declarative method to setup Kafka clusters by the means of Custom Resource Definitions (CRDs) [45].

Here is an example of deploying a Kafka cluster using the Strimzi Kafka Operator:

```yaml
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-kafka-cluster
spec:
  kafka:
    version: 2.8.0
    replicas: 3
    listeners:
      - name: plain
        port: 9092
        type: internal
        tls: false
      - name: tls
        port: 9093
        type: internal
        tls: true
    config:
      offsets.topic.replication.factor: 3
      transaction.state.log.replication.factor: 3
      transaction.state.log.min.isr: 2
      log.message.format.version: "2.8"
    storage:
      type: jbod
      volumes:
      - id: 0
        type: persistent-claim
        size: 100Gi
        deleteClaim: false
  zookeeper:
    replicas: 3
    storage:
      type: persistent-claim
      size: 100Gi
      deleteClaim: false
```

In this case, Kafka cluster is defined as one with three broker replicas and three ZooKeeper replicas. The Kafka brokers are configured with two listeners: a plain listener for internal communication and a TLS listener for secure communication. The config section lets you set different Kafka configuration parameters, such as replication factors and message format versions.

The storage area specifies the storage structure of the Kafka brokers and ZooKeeper nodes. It uses persistent volume claims (PVCs) to give a cluster persistent storage. After the Kafka cluster is deployed, you can access it using the Kafka client libraries and start producing and consuming data.

Streaming Data with Kafka

To illustrate streaming data with Kafka, a simple example of a producer application that produces sensor data and a consumer application that processes the data in real-time is applied.

Producer Application

```python
from kafka import KafkaProducer
import json
import time

# Kafka producer configuration
bootstrap_servers = ['kafka-service:9092']
topic = 'sensor-data'

# Create Kafka producer
producer = KafkaProducer(bootstrap_servers=bootstrap_servers,
                         value_serializer=lambda v: json.dumps(v).encode('utf-8'))

# Generate and send sensor data
while True:
    sensor_data = {
        'temperature': random.randint(20, 30),
        'humidity': random.randint(40, 60),
        'timestamp': int(time.time())
    }
    producer.send(topic, value=sensor_data)
    print(f"Sent sensor data: {sensor_data}")
    time.sleep(1)
```

In the producer application, its a constructed Kafka producer using the Kafka Producer class from the Kafka-python library. It specifically chooses the bootstrap servers (Kafka brokers) and set up the value serializer to transform the sensor data to JSON format.

The loop's cycle creates random sensor data (temperature and humidity) with a time stamp. The producer's production method sends the sensor data to the desired Kafka topic. Kafka receives the sensor data, which is continuously generated and sent to it every second.

Consumer Application

```python
from kafka import KafkaConsumer
import json

# Kafka consumer configuration
bootstrap_servers = ['kafka-service:9092']
topic = 'sensor-data'
group_id = 'sensor-data-consumer'

# Create Kafka consumer
consumer = KafkaConsumer(topic,
                         bootstrap_servers=bootstrap_servers,
                         group_id=group_id,
                         value_deserializer=lambda v: json.loads(v.decode('utf-8')))

# Consume and process sensor data
for message in consumer:
    sensor_data = message.value
    print(f"Received sensor data: {sensor_data}")
    # Process the sensor data as needed
```

The consumer application is where a Kafka consumer with the Kafka Consumer class created. The bootstrap server, the topic to consume from, and the consumer group ID is chosen. In addition, it sets up the value deserializer to extract the JSON-formatted sensor data from the sensor.

The consumer keeps receiving new messages from the specified topic. For every received message, one is able to extract the sensor data from the message value and process it as required. In this case, just print the sensor data; however, in the real world, you would carry out more advanced analysis or take actions depending on the data.

Running the producer and consumer applications via Kafka allows you to see the actual streaming of sensor data from the producer to the consumer in real-time.

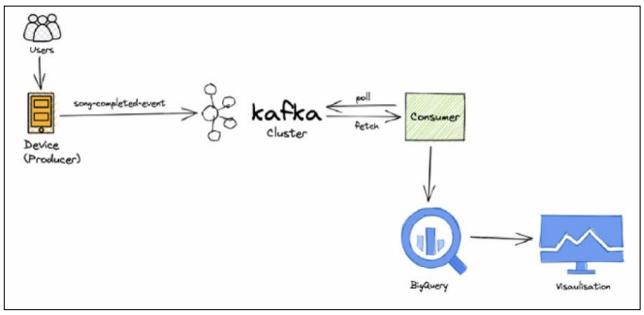

Figure 1: Kafka Streaming Architecture for Real-Time Data Processing

Conclusion and Future Work

In this paper, it discusses the installing and managing containerized data workloads on Amazon EKS. It discussed the advantages, such as mobility, scalability, and environment coherence, which resulted in the containerization. Besides, it presented Amazon EKS and its key benefits, such as the managed Kubernetes control plane, the integration with AWS services, the scalability, and the security.

The research dealt with the best practices of deploying and managing containerized data works on EKS and discussed the issues of data storage, networking, and security. It pointed out the necessity of selecting the appropriate storage options, configuring the network policies, setting up the RBAC, securing the container images, encrypting the data, and monitoring and logging the container ecosystem in containers. An example of the application of a Kafka cluster on EKS was given using the Strimzi Kafka Operator. It also demonstrated data streaming through the use of Kafka, with the data producer application generating sensor data and the data consumer application processing the data in real-time.

Looking ahead, there are several areas for future research and exploration in the field of containerized data workloads on EKS:

- **Serverless Computing:** Researching serverless computing frameworks, such as AWS Lambda, and their integration with EKS to facilitate serverless and event-driven data processing workloads [46].
- **Multi-Cloud Deployments:** Exploring the difficulties and good practices for deploying and managing containerized data workloads on the instances of several cloud platforms leveraging the portability of containers [47].
- **Machine Learning and AI:** The study of the deployment and management of machine learning and AI workloads on EKS, based on the specific requirements and challenges associated with these workloads, is what is being realized [48].
- **Hybrid and Edge Computing:** Exploring the application of EKS in hybrid and edge computing, a scenario of data processing in the cloud and on edge devices [49].

However, organizations are still computerizing and running their data workloads on Amazon EKS, they need to keep up with the current best practices, tools, and techniques. By using containerization and EKS features, organizations can create data workloads that are scalable, portable, and resilient to failure.

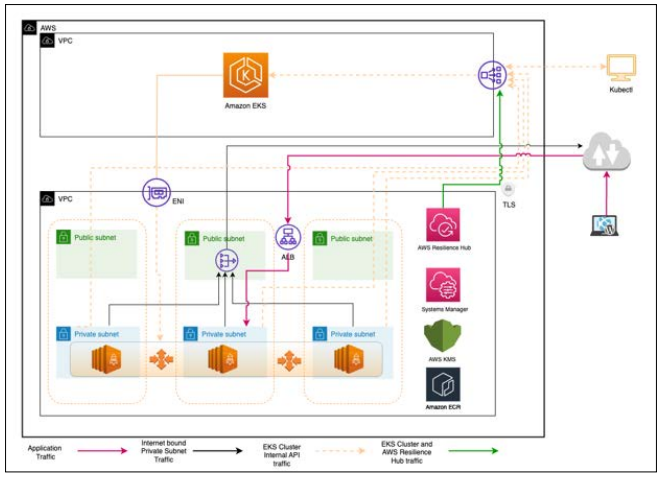

Figure 2: Amazon EKS Architecture for Deploying and Managing Containerized Workloads

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. Sill A (2016) The Design and Architecture of Microservices. IEEE Cloud Computing 3: 76-80.
3. Sakr S, Gounaris A, Tzoumas K (2019) A Survey of Large-Scale Data Processing Approaches in the Era of Big Data. Handbook of Big Data Technologies.
4. Grambow M, Lehmann F, Bermbach D (2021) Continuous Benchmarking of Microservice Performance. Proceedings of the 14th IEEE International Conference on Cloud Computing (CLOUD).
5. Ebert C, Gallardo G, Hernantes J, Serrano N (2016) DevOps. IEEE Software 33: 94-100.
6. Morabito R, Cozzolino V, Ding AY, Beijar N, Ott J (2018) Consolidate IoT Edge Computing with Lightweight Virtualization. IEEE Network 32: 102-111.
7. Burns B, Grant B, Oppenheimer D, Brewer E, Wilkes J (2016) Borg, Omega, and Kubernetes. Queue 14: 70-93.
8. Kratzke N, Quint PC (2017) Understanding Cloud-native Applications after 10 Years of Cloud Computing - A Systematic Mapping Study. Journal of Systems and Software 126: 1-16.
9. Balalaie A, Heydarnoori A, Jamshidi P (2016) Migrating to Cloud-Native Architectures Using Microservices: An Experience Report. In Advances in Service-Oriented and Cloud Computing https://link.springer.com/chapter/10.1007/978-3-319-33313-7_15.
10. Felter W, Ferreira A, Rajamony R, Rubio J (2015) An updated performance comparison of virtual machines and Linux containers. Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) https://ieeexplore.ieee.org/document/7095802.
11. Alshuqayran N, Ali N, Evans R (2016) A Systematic Mapping Study in Microservice Architecture. In Proceedings of the IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA) https://ieeexplore.ieee.org/document/7796008.
12. Jaramillo D, Nguyen DV, Smart R (2016) Leveraging microservices architecture by using Docker technology. Proceedings of the SouthEast Conference https://ieeexplore.ieee.org/document/7506647.

13. (2023) Amazon EKS. Amazon Web Services https://aws.amazon.com/eks/.
14. (2023) Production-Grade Container Orchestration. Kubernetes https://kubernetes.io/.
15. Verma A, Pedrosa L, Korupolu MR, Oppenheimer D, Tune E, et al. (2015) Large-scale cluster management at Google with Borg. Proceedings of the European Conference on Computer Systems (EuroSys) https://dl.acm.org/doi/10.1145/2741948.2741964.
16. Burns B, Grant B, Oppenheimer D, Brewer E, Wilkes J (2016) Borg, Omega, and Kubernetes. Queue 14: 70-93.
17. (2023) AWS Elastic Container Registry. Amazon Web Services https://aws.amazon.com/ecr/.
18. (2023) Amazon Elastic Block Store (EBS). Amazon Web Services https://aws.amazon.com/ebs/.
19. Zaharia M, Reynold Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, et al. (2016) Apache Spark: A Unified Engine for Big Data Processing. Communications of the ACM 59: 56-65.
20. Agneeswaran VS (2014) Big Data Analytics Beyond Hadoop: Real-Time Applications with Storm, Spark, and More Hadoop Alternatives. Pearson Education https://www.oreilly.com/library/view/big-data-analytics/9780133838268/.
21. Armbrust M, Reynold Xin, Cheng Lian, Yin Huai, Davies Liu, et al. (2015) Spark SQL: Relational Data Processing in Spark. In proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) https://dl.acm.org/doi/10.1145/2723372.2742797.
22. (2023) Amazon S3. Amazon Web Services https://aws.amazon.com/s3/.
23. (2023) Amazon Elastic File System (EFS). Amazon Web Services https://aws.amazon.com/efs/.
24. Kreps J, Narkhede N, Rao J (2011) Kafka: A Distributed Messaging System for Log Processing. Proceedings of the 6th International Workshop on Networking Meets Databases (NetDB) https://martin.kleppmann.com/papers/kafka-debull15.pdf.
25. Toshniwal A, Siddarth Taneja, Amit Shukla, Karthikeyan Ramasamy, Jignesh M Patel, et al. (2014) Storm@twitter. Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD) https://dl.acm.org/doi/10.1145/2588555.2595641.
26. Noghabi SA, Kartik Paramasivam, Yi Pan, Navina Ramesh, Jon Bringhurst, et al. (2017) Samza: Stateful Scalable Stream Processing at LinkedIn. Proceedings of the VLDB Endowment 10: 1634-1645.
27. Gulisano V, Jiménez Peris R, Patiño Martínez M, Soriente C, Valduriez P (2012) Stream Cloud: An Elastic and Scalable Data Streaming System. IEEE Transactions on Parallel and Distributed Systems 23: 2351-2365.
28. Akidau T, Alex Balikov, Kaya Bekiroğlu, Slava Chernyak, Josh Haberman, et al. (2013) Mill Wheel: Fault-Tolerant Stream Processing at Internet Scale. Proceedings of the VLDB Endowment 6: 1033-1044.
29. Zaharia M, Das T, Li H, Hunter T, Shenker S, et al. (2013) Discretized Streams: Fault-Tolerant Streaming Computation at Scale. Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP) 423-438.
30. Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, et al. (2015) Apache Flink: Stream and Batch Processing in a Single Engine. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 36: 28-38.
31. Ongaro D, Ousterhout J (2014) In Search of an Understandable Consensus Algorithm. Proceedings of the USENIX Annual Technical Conference (USENIX ATC) 305-320.
32. (2023) Amazon Virtual Private Cloud (VPC). Amazon Web Services https://aws.amazon.com/vpc/.
33. (2023) Network Policies. Kubernetes https://kubernetes.io/docs/concepts/services-networking/network-policies/.
34. (2023) Ingress. Kubernetes https://kubernetes.io/docs/concepts/services-networking/ingress/.
35. (2023) Elastic Load Balancing. Amazon Web Services https://aws.amazon.com/elasticloadbalancing/.
36. (2023) Kubernetes Security. Kubernetes https://kubernetes.io/docs/concepts/security/overview/.
37. (2023) Using RBAC Authorization. Kubernetes https://kubernetes.io/docs/reference/access-authn-authz/rbac/.
38. (2023) Amazon Inspector. Amazon Web Services https://aws.amazon.com/inspector/.
39. (2023) AWS Secrets Manager. Amazon Web Services https://aws.amazon.com/secrets-manager/.
40. (2023) AWS Key Management Service (KMS). Amazon Web Services https://aws.amazon.com/kms/.
41. (2023) Amazon EBS Encryption. Amazon Web Services https://docs.aws.amazon.com.
42. (2023) The Apache Software Foundation. Apache Kafka https://kafka.apache.org/.
43. Garg N (2013) Apache Kafka. Packt Publishing Ltd https://subscription.packtpub.com/search?query=Apache+Kafka.
44. Strimzi (2023) Strimzi: Apache Kafka on Kubernetes. Strimzi https://strimzi.io/.
45. (2023) Custom Resources. Kubernetes https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/.
46. Baldini I, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, et al. (2017) Serverless Computing: Current Trends and Open Problems. In research Advances in Cloud Computing, Springer: https://link.springer.com/chapter/10.1007/978-981-10-5026-8_1.
47. Villari M, Fazio M, Dustdar S, Rana O, Ranjan R (2016) Osmotic Computing: A New Paradigm for Edge/Cloud Integration. IEEE Cloud Computing 3: 76-83.
48. Crankshaw D, Xin Wang, Giulio Zhou, Michael Jay Franklin, Joseph E Gonzalez, et al. (2017) Clipper: A Low-Latency Online Prediction Serving System. Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI) https://dl.acm.org/doi/10.5555/3154630.3154681.
49. Shi W, Cao J, Zhang Q, Li Y, Xu L (2016) Edge Computing: Vision and Challenges. IEEE Internet of Things Journal 3: 637-646.