**Kubernetes: Architecture, Functionality, and Impact on Modern Cloud Computing**

Darshana Chauhan[[1]](#footnote-1)

Deutsche Bank, India.

*Abstract –* **Kubernetes is an open-source container orchestration platform. It has changed the deployment and management of containerized application. This paper explores Kubernetes’ architecture, core components, key features, and its profound impact on cloud computing. This paper dives into Kubernetes’ use cases, highlight its advantages and challenges, and examine the future scopes and developments into the Kubernetes.**

*Keywords* **– Kubernetes, orchestration, container, services, nodes.**

1. INRODUCTION

Kubernetes was initially developed by Google and now it is maintained by Cloud Native Computing Foundation (CNCF). It groups containers that make up an application into logical units for easy management and discovery. Containers are a good way to bundle and run your applications. Kubernetes provides you with a framework to run distributed systems resiliently which means you need not to manage containers and don’t need to ensure that there is no downtime.

Kubernetes is an open-source platform for container orchestration. It is a standard technology for automating the building, deployment, scaling, and management of containerized applications. It provides a robust platform for microservices architecture and DevOps practices.

The paper provides basic understanding of Kubernetes, its architecture, key functionalities, and its real-world use cases in cloud computing. Also, it explores the challenges faced by users, and future trends shaping the Kubernetes landscape.

1. LITERATURE REVIEW

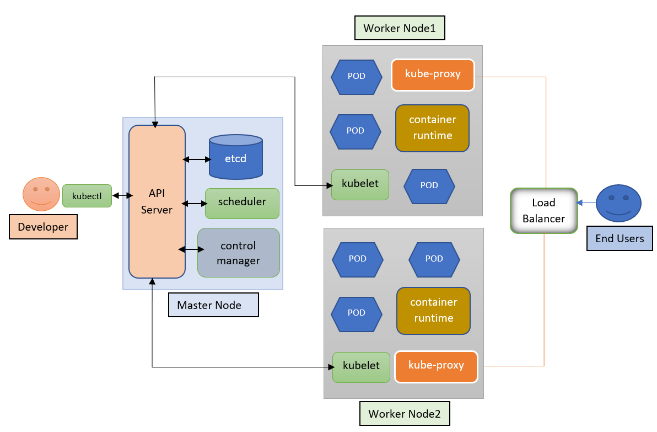
Kubernetes, often called as K8s. It was released by Google in 2014, building on experience with Borg, an internal container management system. It supports diverse variety of workloads, like stateless, stateful and data-processing workload. It is used for automating the server creation and scaling. For that, it uses container orchestration tool to manage containers which includes less human resource to manage the central architecture. Kubernetes has some good features like portability, extensibility, declarative configuration, and automation.

1. METHODOLOGY

**Core Elements**

* **Pods:** A **pod** is the smallest deployable unit in Kubernetes. A group of containers can group together. In the group, multiple independent containers act as single unit which called as pod. It acts as a wrapper around the containers. Pod is an abstraction over container. Pods are ephemeral resources. Pod takes care of interaction, management, and networking of containers.
* **Container:** Container contains Container image, which is lightweight, standalone package of software includes everything to run an application. Container developed by a team from container image.
* **Nodes:** Pods are scheduled onto nodes. Nodes provide the underlying compute resources (CPU, RAM) for running containers. A node could be a virtual or physical machine, depending on the cluster. Each node is managed by the [control plane](https://kubernetes.io/docs/reference/glossary/?all=true#term-control-plane) and contains the services necessary to run [Pods](https://kubernetes.io/docs/concepts/workloads/pods/). Each Kubernetes cluster requires at least one worker node, which is a collection of worker machines that contains the nodes where our container will be deployed.
* **Cluster:** A master and its controlled nodes (worker nodes) constitute a **“Kubernetes cluster”.**
* **Ingress:** Ingress objects provide an easy-to-use frontend that can combine multiple microservices into a single externalized API surface area. When a request comes from internet, it first goes to ingress and then ingress routes the request to service.
* **Services:** Kubernetes services provide load balancing, naming, and isolate one microservice from another. To make pod accessible, each service object has a logical set of endpoints (usually these endpoints are Pods) along with a policy.
* **Namespaces:** They provide isolation and access control, so that each microservice can control the degree to which other services interact with it. Names of resources should be unique within a namespace, but not across namespaces.
* **Control plane (Master Node):**The cluster contained worker nodes and any pods contained within them will be under the control plane. It is also called “Master Node” as it serves as a master to all the worker nodes.

**Kubernetes Architecture:**

****

**Master Node Components:**

* **API server:** It is similar to an initial gateway to the cluster that listens the updates or queries via CLI like Kubectl. Kubectl communicates with API Server to inform what needs to be done like creating pods or deleting pods etc.  It generally validates requests received and then forwards them for other processes. All the requests should need to go through API server, no direct requests are allowed. It takes care of container orchestration (manages all containers).
* **Scheduler:** The request from API for scheduling will be passed on to the scheduler. It decides on which node to schedule the pod for better efficiency of the cluster.
* **Controller-Manager:** It is responsible for running the controllers that handle the cluster’s control loop. There are four key controllers like: replication controller, endpoint controller, namespace controller, service account controller. replication controller ensures that the desired number of replicas of a given application is running.
* **etcd:** This is one type of database that stores key-value of a cluster, cluster’s state when it gets changed. It tells the scheduler and other processes about which resources are available and about current cluster state, desired state. So, it acts as the brain for the entire architecture.

**Worker Node Components:**

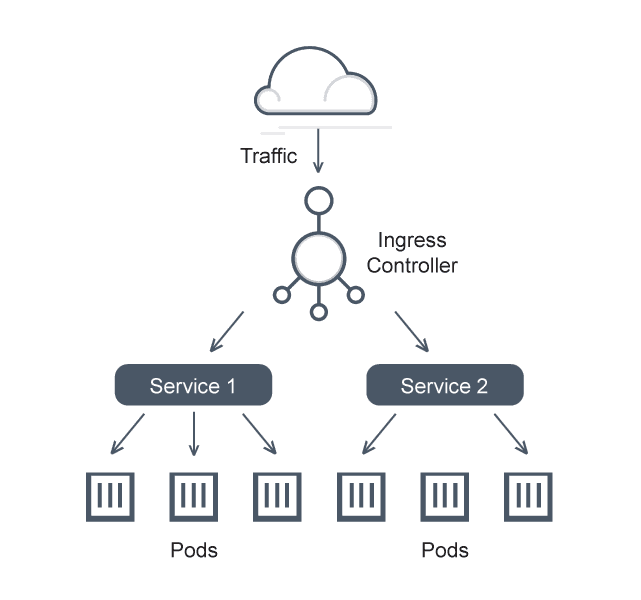
* **Container Runtime:** It is foundational software that enables containers to operate within a host system. It handles the tasks like pulling container images from registries, managing lifecycle, running containers on your system, etc.
* **Kubelet:** It interacts with both container runtime and Node. It is responsible for starting a pod with a container inside.
* **Kube-proxy:** It is a process responsible for forwarding the requests from Services to the pods. It will divert the request to right pod in worker node.

**Method of working:**

The work and connection between the Master node and cluster is via API server. The process will be as followed. The user interacts with the Kubernetes cluster through kubectl, a command line interface, the command will be sent in API server. API server will do appropriate update in controller-manager, scheduler, and etcd to ensure the desired state of cluster. Controller-manager will control the desired state of pods and stores them in etcd via API server. It will also take care of discrepancy between the desired and actual state and takes appropriate action and updates the state via API server. Now, scheduler will schedule a pod in a particular node according to need and save the data in etcd via API server. It also selects an appropriate node for the pod and updates the desired state. A pod needs to be created by different reasons like change in state, user need, previously scheduled.

On the cluster node side, the pod creation request will be sent to kubelet according to the scheduler’s request. It will then interact with container runtime and node to create and manage container. After node creation, kube-proxy will manage DNS requests and route them to proper container. This is how the pod gets created and manages requests.

**Operations:**

****

* **Services:** A Service is an abstraction that defines a logical set of pods and a policy to access them. It provides a stable network endpoint (IP address or DNS name) for accessing pods. Types of services include ClusterIP, NodePort, and LoadBalancer. It has differentiated in two different services: external service and internal service. In external service, the pod or replica is accessed by externally, which means it is public service. In internal service, the sara should remain private.
* **Ingress:** Ingress exposes HTTP and HTTPS routes from outside the cluster to services within. It acts as a reverse proxy, handling traffic routing, SSL termination, and load balancing.
* **ConfigMaps:** A ConfigMap holds non-confidential data (key-value pairs, files, or directories). It allows to decouple configuration from application code, and dynamic updates without redeploying pods. It is a useful tool for environment variables, command-line arguments, or configuration files.
* **Secret:** A Secret securely stores sensitive information (e.g., passwords, API keys, tokens). Encoded or encrypted data is stored in base64 format. It is same as ConfigMaps but it stores confidential data.
* **Data storage:** Data storage refers to the overall management of storage resources. It includes ephemeral volumes (Exist till the end of pod) and persistent volume (Exist till the end of master node).
* **Volume:** Volume is a directory associated with pod then mounted into to the containers at some different paths. It can be backed by various storage types. Volumes are essential for stateful applications (e.g., databases, key-value) to maintain consistent storage independent of the container lifecycle. Volumes allow data sharing between containers within the same pod.
* **Distributed system:** Kubernetes itself is a distributed system. It allows to run multiple containerized application across multiple nodes with high availability and scalability. Kubernetes provides the replica of pod in the architecture. So, if one pod dies another will take over place of previous pod. Across all these replicas the traffic is getting distribute.
* **Loadbalancer:** A LoadBalancer service type exposes an external IP and distributes traffic to backend pods. As we have replicas of the pods, traffic will divert to different pod replicas.
* **Deployment:** A deployment provides declarative updates for pods and replicas. You describe a desired state in the deployment and the deployment controller changes the actual state to the desire state. It is an abstraction on pods. It ensures self-healing, rolling updates, and rollbacks. Deployments are declarative and handle pod lifecycle.
* **StatefulSets:** It manages stateful applications (e.g., databases, queues). Also provides stable network identities (DNS names) for each pod. Pods are created in a predictable order and maintain their identity during rescheduling. A StatefulSet maintains a sticky identity for each of its Pods.

**Kubernetes Functionality:**

* **Container orchestration:** It automates the deployment, storage management, scaling, and networking of containers. It is a good tool which help in deploying same application across different environment without doing any changes. Container orchestration is used to automate and manage tasks like provisioning, deployment, configuration and scheduling, resource allocation, container availability, scaling up and down based on balancing workloads across your infrastructure, load balancing, traffic routing, monitoring container health, configuring application inside container, and keeping interactions between containers secure.

**Networking:** Networking is a central part of Kubernetes.

* **Cluster networking:** 
  + Highly coupled container-to-container communications
  + Pod-to-Pod communications
  + Node-to-node communication
  + Pod-to-Service communications
  + External-to-Service communications
* **Service Networking:** Provides stable IP and DNS name for services, enabling external access to applications.
* The network plugin is configured to assign IP addresses to Pods.
* The kube-apiserver is configured to assign IP addresses to Services.
* The kubelet or the cloud-controller-manager is configured to assign IP addresses to Nodes.

1. RESULTS

**Benefits:**

* **Self-Healing:** Kubernetes monitors the health of containers and automatically restarts failed ones. It also replaces unresponsive nodes, ensuring reliability.
* **Horizontal Pod Autoscaling (HPA):** Kubernetes allows you to scale your application horizontally by adding or removing pods dynamically. The HPA automatically increase a workload resources (such as a Deployment or Statefulset) to match demand. When the load decreases, it scales back down.
* **Service Discovery & load Balancing:** To distribute traffic evenly across containers, Kubernetes provides built-in load balancing for services.
* **Automated Rollouts and Rollbacks:** For a new deployment, it creates a new rollout with an increased version and at the time of failing or the deployment is unstable, the rollback will happen to a previous version using saved state for the desired state of the containerized application.
* **Automated Scheduling:** To launch containers on cluster nodes, Kubernetes provides an advanced scheduler. It performs resource optimization.
* **Resource Utilization:** As containers share the same OS kernel, making them lightweight compared to virtual machines (VMs). This efficient resource usage translates to cost savings.
* **Supports Multi cloud and Hybrid cloud:** Kubernetes can be deployed on different cloud platforms and run containerized applications across multiple clouds.
* **Extensibility:** Kubernetes is very extensible and can be extended with custom plugins and controllers.
* **Community Support:** Kubernetes has a large and active community with frequent updates, bug fixes, and new features being added.
* **Automated deployment and management:** Kubernetes doesn’t need any manual intervention for deployment process.Kubernetes will take care of automating the deployment, scaling, and containerizing the application.
* **High Availability:** Kubernetes ensures that applications are always available by automatically distributing workloads across nodes and handling failure.
* **Cost Effectiveness:** Kubernetes will help to reduce resource utilization and control the overprovisioning of infrastructure.
* **Improved Developer Productivity:** Kubernetes will reduce the efforts of deploying the application so, developer can concentrate on the developing part.

**Security:**

* **RBAC (Role-Based Access Control):** Restricts user actions based on their roles within the organization.
* **Network Policies:** Controls communication between pods.
* **Secrets Management:** Handles sensitive information like password, OAuth tokens, and SSH keys.

**Examples of Kubernetes:**

* E-commerce website (Shopify)
* Media and Entertainment (Netflix, Spotify)
* Financial Services (fintech companies)
* Health Care Services

**Kubernetes in Financial services:**

* Ant Financials’ Hypergrowth Strategy using Kubernetes.
* ING Driving Banking Innovation with cloud Native.
* Capital One: Financial Services company that uses Kubernetes fraud detection and make quick credit decision.
* Monzo: “Mobile” U.K bank leverages modern detection and make quick credit decision.

**Case Study:**

**Monzo[[2]](#footnote-2):**

If company wants to build a fault-tolerant, elastically scalable system, it follows the ability to add and remove servers as hardware fails, and as user demand changes. Running only a single service per host is wasteful (the service may not need all the resources of the host), and the traditional approach of manually partitioning services among hosts is tedious and difficult to scale.

Cluster schedulers are designed to abstract applications away from the hardware on which they run. They use algorithms to schedule applications onto hosts according to their resource requirements, scale them up and down, and replace them when they fail. It’s goal to run the entire system on a scheduler, which means, the company wants to deploy stateful as well as stateless apps in this way.

Here are some steps in which the company has made the Kubernetes:

* + **Planning and requirements Analysis:** Monzo assessed their infrastructure needs, scalability requirements, and compliance.They defined the scope of Kubernetes adoption and identified key services to containerize.
  + **Setting up Kubernetes Cluster:** Monzo chose a cloud provider (e.g., AWS, GCP) and set up a Kubernetes cluster. They configured nodes, networking, and security groups.
  + **Containerization of Application:** Monzo containerized their microservices using Docker. Each microservice became a Docker image.
  + **CI/CD pipeline Integration:** Monzo integrated their CI/CD pipeline (e.g., Jenkins, GitLab CI) with Kubernetes. Automated builds and deployments were triggered when code changes occurred.
  + **Deploying in Kubernetes:** Monzo defined Kubernetes Deployment manifests (YAML files) for each microservice. Deployments included replicas, resource limits, and environment variables.
  + **Networking and Security:** Monzo set up Kubernetes Services for load balancing and internal communication. Network Policies controlled traffic between microservices. Ingress controllers managed external access.
  + **Monitoring and Logging:** Monzo used Prometheus for monitoring Kubernetes clusters. They configured Grafana dashboards for visualization.Logs were collected using Fluentd or Loki.
  + **Scaling and Autoscaling:** Monzo defined Horizontal Pod Autoscalers (HPAs) to automatically adjust replica counts. Autoscaling was based on CPU/memory utilization or custom metrics.
  + **Disaster recovery and High availability:** Monzo set up multi-zone clusters for high availability. Regular backups of etcd and persistent volumes ensured disaster recovery.
  + **Compliance and Auditing:** Monzo adhered to regulatory requirements (e.g., GDPR, PCI-DSS) by securing data. Auditing logs were maintained for compliance purposes.

## We started by identifying where traffic leaves the platform. Then we started with port-based filtering, we investigated existing solutions SNI and host header inspection, DNS inspection. By b**uilding our own egress firewall,** we came up with a hybrid approach.[[3]](#footnote-3)

**Use cases:**

* **DevOps Practices:** Kubernetes integrates with CI/CD pipelines, automating application deployment and updates. It helps to collaborate development and operations teams.
* **Microservices Architecture:** Microservices can be launched into the cluster and scaled them independently across the different Nodes. The interaction between microservices and monitoring of the microservices can be done using the Kubernetes API.
* **Big Data and AI/ML Workloads:** These tasks involve processing huge amounts of data within computationally intensive pipelines. The main aim is to get visibility that process will work properly, and the output will be produced. Kubernetes clusters support all these requirements.
* **Cloud Native Development:** Cloud-native development involves building applications that are designed to run on cloud infrastructure and take advantage of the cloud facilities. Kubernetes is key component of it.
* **Continuous Integration and Deployment:** CI/CD pipelines be used to automate the deployment and roll out new version of application with minimal downtime after integration with Kubernetes.
* **Hybrid and multi cloud Deployment:** Kubernetes provides cross-cloud deployment. So, you can run different services from single system in various cloud and on-premises environments.
* **High performance Computing:** Kubernetes can be used to manage high-performance computing works, like scientific simulations, machine learning, and big data processing.

**Challenges and Limitations:**

* **Complexity:** Kubernetes is a very complex networking and managing tool. The knowledge for learning and using it effectively takes time and effort. Also, it is very complex from the components point of view.
* **Security concerns:** Security is a significant challenge as it is complex and vulnerable. Improper monitoring can hinder vulnerability detection, providing an easy entry point for hackers.
* **Resource Management:** It is very hard and complex to allocate the resources (CPU, memory) efficiently.
* **Eco System Integration:** Integrating Kubernetes with existing systems (e.g., databases, monitoring tools) can be complex. As communication between components and security are essential keys for Kubernetes.

**Future Trends and Developments:**

* **Edge Computing:** Kubernetes will play a crucial role in managing applications at the edge, as edge computing gains prominence.
* **Serverless Computing:** The convergence toward serverless computing allows developers to build event-driven, auto-scaling applications without managing infrastructure explicitly.
* **Enhanced Security Feature:** As security remains a priority, Kubernetes would continue to evolve its security features, including better access controls, encryption, and vulnerability scanning.
* **AI Automation:** Kubernetes will leverage AI and ML for smarter decision-making and automated operations. This will enhance the resource allocation, workload prediction, and optimization based on historical data.

1. DISCUSSION

So, from all the results, Kubernetes is a good software to automatically increase the cluster at the time of more traffic. Kubernetes defines how the application should run, also handles automatic deployment, scaling and management of the applications based on the specifications.

1. CONCLUSION

Kubernetes has fundamentally transformed the way applications are deployed and managed in cloud environments. It is technology for automating the building, deployment, scaling, and management of containerized applications. Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. Despite its challenges, the future of Kubernetes looks promising, with ongoing developments poised to further enhanced its capabilities and adoption.

1. REFERENCES
2. Kubernetes Official Documentation: <https://kubernetes.io/>
3. [Introduction to Kubernetes (K8S) - GeeksforGeeks](https://www.geeksforgeeks.org/introduction-to-kubernetes-k8s/)
4. <https://www.youtube.com/watch?v=s_o8dwzRlu4>
5. “Kubernetes: Up and Running”, by Kelsey Hightower, Brendan Burns, and Joe Beda.
6. “Kubernetes patterns: Reusable Elements for Designing Cloud-Native Application”, By Bilgin Ibryam and Roland Hub.
7. “The Kubernetes Book”, by Nigel Poulton.
8. CNCF Kubernetes Project: <https://www.cncf.io/projects/kubernets/>
9. Chatbot
10. <https://www.redhat.com/en/topics/containers/>
11. <https://spacelift.io/blog/kubernetes>
12. <https://www.buzzybrains.com/blog/future-of-devops-trends-in-docker-and-kubernetes/>

1. Darshana.chauhan@db.com [↑](#footnote-ref-1)
2. [Building a Modern Bank Backend (monzo.com)](https://monzo.com/blog/2016/09/19/building-a-modern-bank-backend/) [↑](#footnote-ref-2)
3. https://monzo.com/blog/2020/10/19/autoscaling-monzo [↑](#footnote-ref-3)