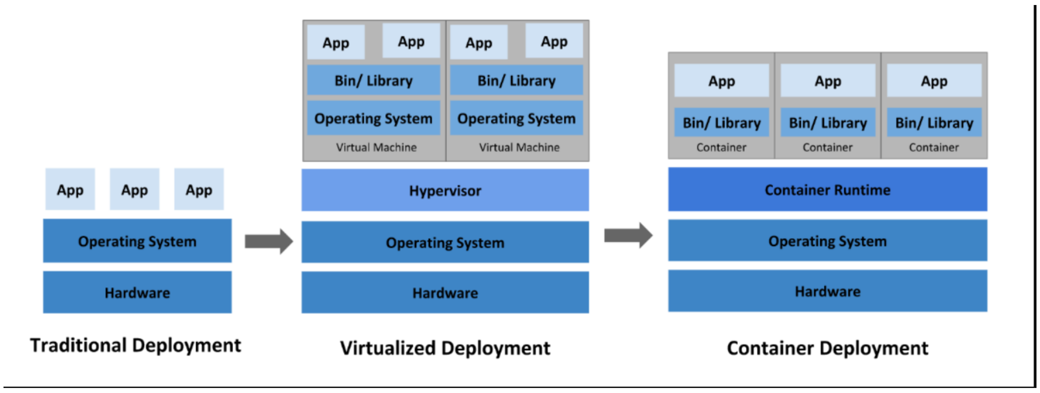
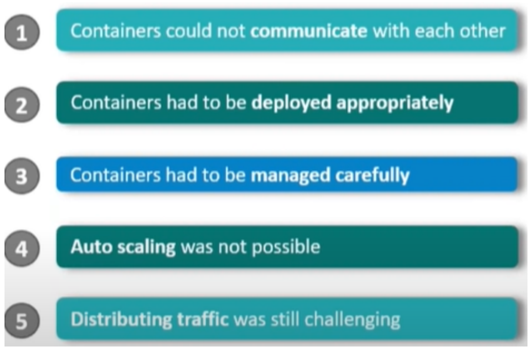
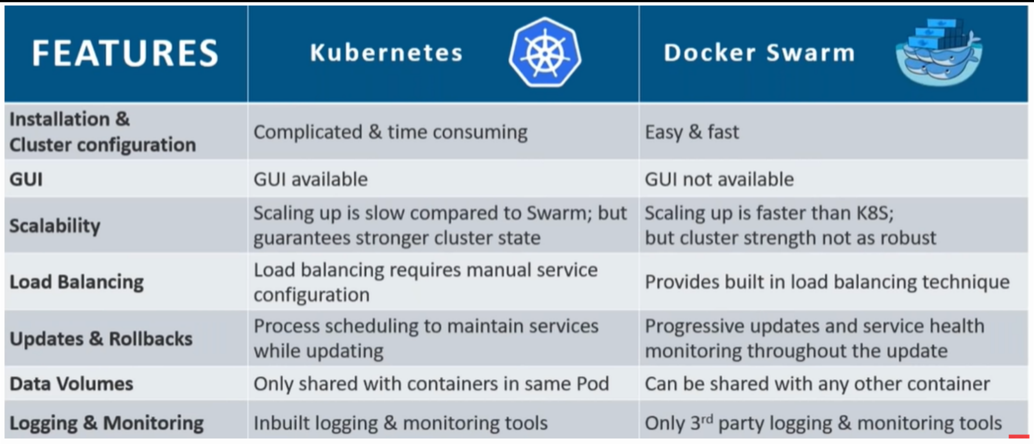
**B.M.S. COLLEGE OF ENGINEERING**
Autonomous Institute, Affiliated to VTU
Estd. 1946

**DEPARTMENT OF CSE**
**CTY Project Work In collaboration with HPE**

| | | | |
|---|---|---|---|
| **Project Title** | Open source monitoring and observability stack on Kubernetes | | |
| **Student Team** | Name: Pooja Srinivasan<br>USN:1BM18CS069<br>SEM:UG-6th SEM | Name: Anusree Manoj K<br>USN:1BM18CS017<br>SEM:UG-6th SEM | |
| | Name: Shikha N<br>USN:1BM18CS149<br>SEM:UG-6th SEM | Name: Niha<br>USN:1BM18CS060<br>SEM:UG-6th SEM | |
| **Faculty Mentor** | Dr. Nandhini V<br>Associate Professor | **HPE Mentors** | Divakar Padiyar<br>Sonu Sudhakaran |
| **Review for the Period** | 09-03-2021 | 18-03-2021 | |
| **Task Given** | Exploring and understanding concepts - Kubernetes<br>Understand Physical Server vs Virtual Machine vs Containers | | |
| **Difficulties Faced** | None | | |
| **Libraries Used** | None | | |
| **Github Link for the code:** | None | | |
| **Code:** None | | | |
| **Physical Server vs Virtual Machine vs Containers** [7] | **Physical server:** A physical server, also known as a 'bare-metal server,' is a single-tenant computer server, meaning that a specific physical server is designated to a single user. The resources and components of a physical server are not shared between multiple users. Each physical server includes memory, processor, network connection, hard drive, and an operating system (OS) for running programs and applications. A bare-metal server is large in size due to the powerful processing components that it contains.<br><br>**Virtual Machine:** A virtual machine (VM) is a software computer used as an emulation of an actual physical computer. A virtual server operates in a "multi-tenant" environment, meaning that multiple VMs run on the same physical | | |

| | |
|---|---|
| | hardware. In this case, the computing resources of a physical server are virtualized and shared among all VMs running on it. The architecture of a virtual server is a little more complex than that of a physical server. Thus, a hypervisor, such as VMware vSphere or Microsoft Hyper-V, is installed on top of physical hardware. A hypervisor is then used to create and manage VMs, which have their own virtual computing resources. After that, you can load multiple guest OSes and server applications on top of the virtual hardware. Thus, virtual servers allow you to run several OSes and applications on the basis of the shared physical hardware, which makes it a more cost-effective option than a physical server.<br><br>**Containers:**Containers provide a way to run these isolated systems on a single server or host OS. Containers sit on top of a physical server and its host OS—for example, Linux or Windows. Each container shares the host OS kernel and, usually, the binaries and libraries, too. Shared components are read-only. Containers are thus exceptionally "light"—they are only megabytes in size and take just seconds to start, versus gigabytes and minutes for a VM.<br>Containers also reduce management overhead. Because they share a common operating system, only a single operating system needs care and feeding for bug fixes, patches, and so on. This concept is similar to what we experience with hypervisor hosts: fewer management points but slightly higher fault domain. In short, containers are lighter weight and more portable than VMs.<br><br> App<br>App<br>App<br>Operating System<br>Hardware<br>Traditional Deployment<br>App<br>App<br>App<br>App<br>Bin/ Library<br>Bin/ Library<br>Operating System<br>Operating System<br>Virtual Machine<br>Virtual Machine<br>Hypervisor<br>Operating System<br>Hardware<br>Virtualized Deployment<br>App<br>App<br>App<br>Bin/ Library<br>Bin/ Library<br>Bin/ Library<br>Container<br>Container<br>Container<br>Container Runtime<br>Operating System<br>Hardware<br>Container Deployment |
| **Why Kubernetes?**[1,4] | In the past,applications were designed using monolithic architecture where all the features were consolidated into a single codebase with dependencies between all the features of the applications not interacting using JSON,which caused reliability,updation and scalability issues.<br>This problem was solved by using microservices where all the features/services have a different code base and are deployed separately (distributed) and all of them interact using API calls.But again there were questions on how and where to deploy these services and scalability,resource utilization issues.<br>All this was solved using docker ,which gives a separate environment to deploy and run services on different servers so as to make them totally independent of each other by creation of containerized microservices.<br>Problems with scaling up containers: |

| | |
|---|---|
| | <br>1 Containers could not **communicate** with each other<br>2 Containers had to be **deployed appropriately**<br>3 Containers had to be **managed carefully**<br>4 **Auto scaling** was not possible<br>5 **Distributing traffic** was still challenging<br>All the issues on orchestration of containers were solved by kubernetes.<br>The primary advantage of using Kubernetes in your environment, is that it gives you the platform to schedule and run containers on clusters of physical or virtual machines (VMs).<br>It helps to fully implement and rely on a container-based infrastructure in production environments.<br>Along with modern continuous integration and continuous deployment (CI/CD) tools, Kubernetes provides the basis for scaling apps without huge engineering effort.<br>**Benefits of Kubernetes**:<br>● Portability,faster,simpler deployment times.<br>● Scalability<br>The ability to run containers on one or more public cloud environments, in virtual machines, or on bare metal means that it can be deployed almost anywhere.<br>● High Availability at both the application and the infrastructure level.<br>● Open Source<br>● Proven, and Battle Tested<br>● Market Leader<br><br>Docker Swarm is ideal for users who want a simple,quick setup of containerized applications with fewer resources whereas Kubernetes would be the best containerization platform for deploying complex applications which utilizes huge numbers of containers in the process as it has high availability policies and auto-scaling capabilities and extensive functionalities and customizable options. |



| FEATURES | Kubernetes | Docker Swarm |
|---|---|---|
| Installation & Cluster configuration | Complicated & time consuming | Easy & fast |
| GUI | GUI available | GUI not available |
| Scalability | Scaling up is slow compared to Swarm; but guarantees stronger cluster state | Scaling up is faster than K8S; but cluster strength not as robust |
| Load Balancing | Load balancing requires manual service configuration | Provides built in load balancing technique |
| Updates & Rollbacks | Process scheduling to maintain services while updating | Progressive updates and service health monitoring throughout the update |
| Data Volumes | Only shared with containers in same Pod | Can be shared with any other container |
| Logging & Monitoring | Inbuilt logging & monitoring tools | Only 3rd party logging & monitoring tools |

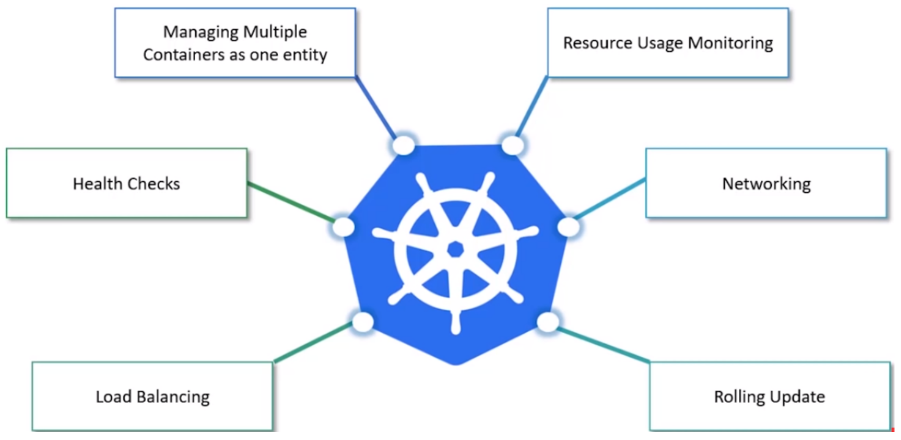
**What is Kubernetes?**[1-2]

Kubernetes is an open source container orchestration tool.
It is used when the application is distributed on multiple containers.
It allows you to run containers, manage them, automate deploys, scale deployments, create and configure ingresses, deploy stateless or stateful applications, etc.
Basically, you can launch one or more instances and install Kubernetes to operate them as a Kubernetes cluster. Then get the API endpoint of the Kubernetes cluster, configure kubectl (a tool for managing Kubernetes clusters) and Kubernetes is ready to serve.

**Features of Kubernetes** [1]

Features of Kubernetes are as follows:

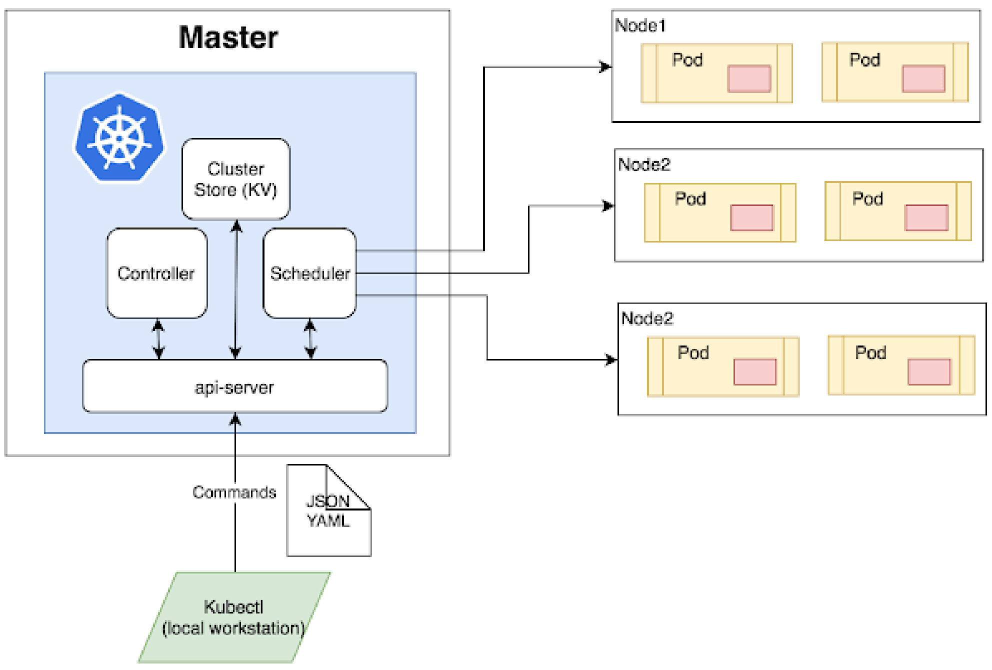
- **Managing Multiple Container as one entity**- If we have ‘n’ containers running and have ‘m’ features running on it then we see only the ‘m’ features which are running and not the containers which are running in the background since the health monitoring and scaling is taken care by kubernetes.We take care about the new services we wish to launch and the features we want to deploy.For deploying a feature we just specify the number of containers and Kubernetes takes care how to deploy and manages the backend.
- **Health Check**-If the container does not run as expected or if the service is down then kubernetes redeploys the container so that the service is always.Hence there is no downtime.
- **Resource Usage Monitoring**-In kubernetes when a container is running we can define how much of our computer resources it is going to utilize.When a container utilizes more resources a new container is deployed.
- **Networking**-In Kubernetes where containers are deployed on community cluster can interact with each other as if they were on the same system.They can be on many nodes but still can interact with each other irrespective of the fact where they exist on the cluster
- **Load Balancing**-If there are multiple replicas of the containers which are running inside the cluster then we can automatically load balance among

| | |
|---|---|
| | these containers based on the request.<br>● **Rolling Update**-If we want to update our application we have an option called Bluegreen deployment.<br><br> Managing Multiple Containers as one entity<br>Resource Usage Monitoring<br>Health Checks<br>Networking<br>Load Balancing<br>Rolling Update |
| **Kubernetes Architecture** [1,5] | Architecture of kubernetes consists of the following components:<br><br>**Master Node**<br>The master node is the first and most vital component which is responsible for the management of Kubernetes cluster. It is the entry point for all kinds of administrative tasks. There might be more than one master node in the cluster to check for fault tolerance.<br><br>Components:<br><br>● **API Server:** The API server acts as an entry point for all the REST commands used for controlling the cluster.<br>● **Scheduler**:The scheduler schedules the tasks to the slave node. It stores the resource usage information for every slave node. It is responsible for distributing the workload.It also helps you to track how the working load is used on cluster nodes. It helps you to place the workload on resources which are available and accept the workload.<br>● **Etcd**:etcd components store configuration detail and wright values. It communicates with the most component to receive commands and work. It also manages network rules and port forwarding activity. |

**Worker/Slave nodes**

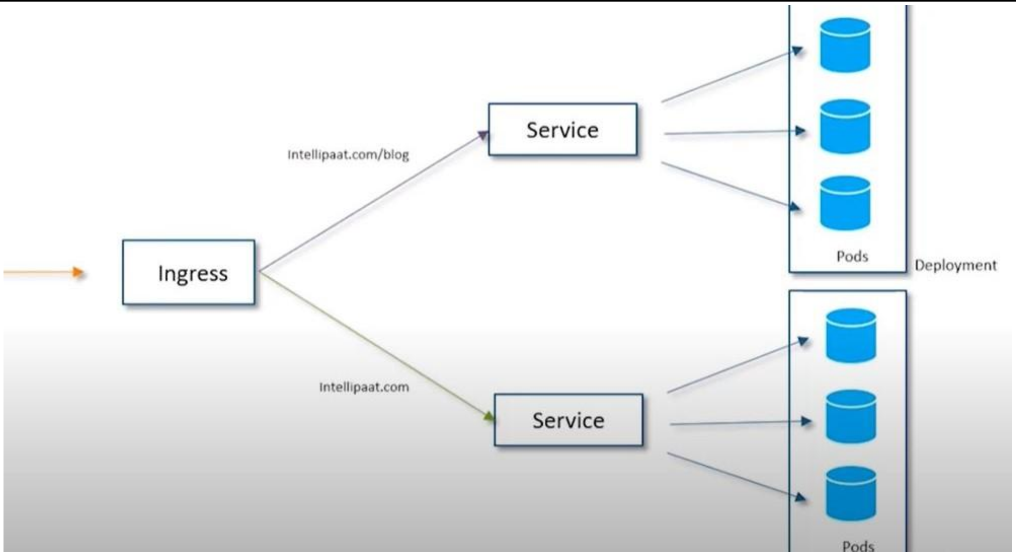
It contains all the required services to manage the networking between the containers, communicate with the master node, which allows you to assign resources to the scheduled containers.

- Kubelet: gets the configuration of a Pod from the API server and ensures that the described containers are up and running.
- Docker Container: Docker container runs on each of the worker nodes, which runs the configured pods
- Kube-proxy: Kube-proxy acts as a load balancer and network proxy to perform service on a single worker node
- Pods: A pod is a combination of single or multiple containers that logically run together on nodes



How an application is deployed:

- Kubernetes architecture plays a major role when a user wants to (deploy) access any application stored in a pod inside a particular cluster.
- When a user wants to search any application, it basically sends the URL of the application for search that in turn points to the server where the application is stored.
- Here, the deployment of the application is done through 2 steps; it has service that is basically an internal load balancer for all the pods which are running inside the cluster for the request provided, it has ingress which directs the request to the specific service based on the URL.

| | |
|---|---|
| |  Ingress<br>intellipaat.com/blog<br>Service<br>intellipaat.com<br>Service<br>Pods<br>Deployment<br>Pods |
| **Observability** [3] | Observability means assembling all fragments from logs, monitoring tools and organizing them in such a way which gives actionable knowledge of the whole environment, thus creating an insight.<br>The types of data that a system should produce to be observable are:<br>● Health checks<br>● Metrics<br>● Log entries<br>● Distributed, request or end-to-end tracing |
| **Monitoring** [3] | Automating manual, repetitive, and error-prone work which results in faster speed, productivity, and scalability.Eliminating errors and bugs reduce the wastage of time and let you deploy software faster and more reliably.<br>Activities in continuous monitoring:<br>● Problem detection<br>● Problem resolution<br>● Continuous improvement |
| **Observability vs Monitoring** [6] | Observability and monitoring complement each other, with each one serving a different purpose.<br>Monitoring tells you when something is wrong, while observability enables you to understand why. Monitoring is a subset of and key action for observability. You can only monitor a system that’s observable.<br>Monitoring tracks the overall health of an application. It aggregates data on how the system is performing in terms of access speeds, connectivity, downtime, and bottlenecks. Observability, on the other hand, drills down into the “what” and “why” of application operations, by providing granular and contextual insight into its specific failure modes. |
| **References** | 1. https://www.youtube.com/watch?v=NsDhBEsTTHs<br>2. https://www.toptal.com/kubernetes/what-is-kubernetes<br>3. https://www.xenonstack.com/insights/observability-vs-monitoring/<br>4. https://www.youtube.com/watch?v=F-p_7XaEC84<br>5. https://www.guru99.com/kubernetes-tutorial.html |

| | |
|---|---|
| | 6. https://thenewstack.io/monitoring-vs-observability-whats-the-difference/#:~:text=Observability%20and%20monitoring%20complement%20each,monitor%20a%20system%20that%27s%20observable<br>7. https://www.nakivo.com/blog/physical-servers-vs-virtual-machines-key-differences-similarities/ |