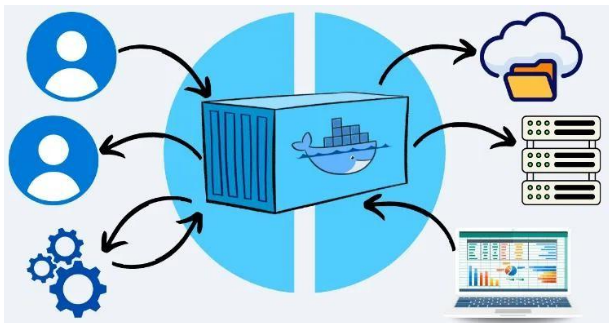
https://www.linkedin.com/in/arun-kumar-akula-3b5180343/

Why Kubernetes Overcomes Docker's Limitations

➢ Overview of Docker :



Docker is a tool that helps you package an application along with its dependencies into a portable unit called a container. This makes it easy to run the app anywhere without worrying about differences between environments.

Limitations of Docker :

- **No Automatic Scaling:** Docker can't adjust the number of running containers based on demand without extra tools, which limits resource efficiency**.**
- **No Built-in Load Balancing:** Docker does not automatically distribute network traffic across multiple containers, so managing traffic flow requires additional tools or manual setup.
- **No Self-Healing:** If a container crashes or stops unexpectedly, Docker does not automatically restart or replace it, which can cause downtime or service interruptions.
- **Limited networking:** Docker's networking is simple and works well on a single host but has limitations handling complex networks across multiple machines.
- **Weak Secrets and Configuration Management:** Docker provides simple methods to store sensitive information like passwords or API keys but doesn't have strong tools for securely managing or updating them.
- **No Built-in Backup Support :** Docker doesn't provide built-in backup solutions for container data, so you need to set up your own backup processes to prevent data loss.

How Docker Supports Kubernetes:

Docker creates and runs containers, while Kubernetes manages and scales those containers across many machines. Together, they make deploying and running apps easier, more reliable, and efficient helping teams update apps faster and keep them running without problems

➢ Overview on Kubernetes (K8'S) :

Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications.
It was originally developed by **Google** and is now maintained by the Cloud Native Computing Foundation (CNCF).
Kubernetes is also called as **K8's**.

Why Choose Kubernetes for Modern App Deployment:

- **Auto-scaling :** Automatically increases or decreases the number of app instances based on traffic or resource usage helping you handle high loads and save costs during low traffic.
- **Self-healing :** Automatically restarts failed containers, replaces dead ones, and reschedules them on healthy nodes.
- **Rolling updates :** Deploy new versions with zero downtime, and roll back easily if something goes wrong.
- **Flexibility :** Kubernetes runs on any cloud (AWS, Azure, GCP), on-prem servers.
- **Built-in Load Balancing :** Automatically balances incoming traffic across all your app instances, preventing overload on any single server and keeping your application always available.
- **Secure and Organized :** With features like namespaces and RBAC (role-based access control), Kubernetes helps organize apps and manage who can do what improving security.

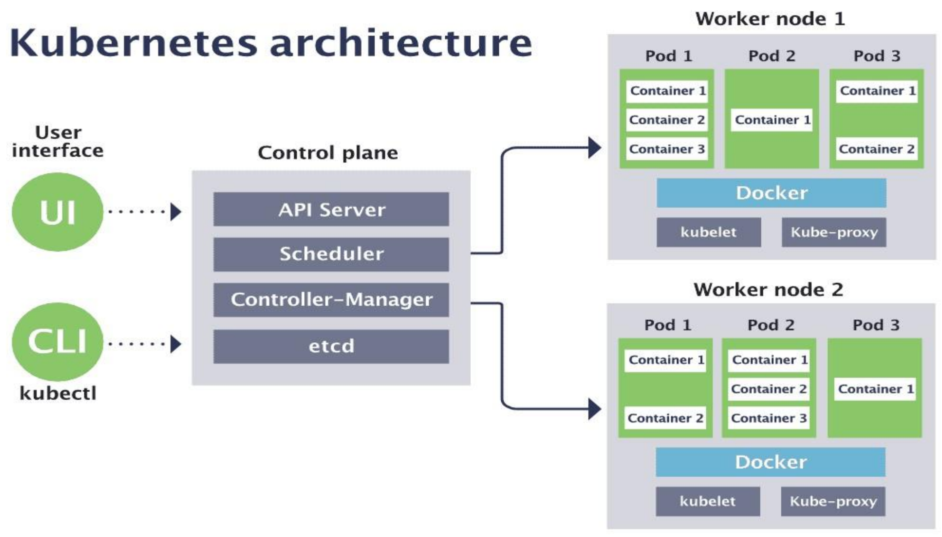
Kubernetes Architecture Overview:

Workstation : A workstation(like your laptop or desktop) is not a core part of the Kubernetes architecture, but it is used to interact with or manage a Kubernetes cluster**.**

master node : master node is also called as control plane. it controls and manages the entire Kubernetes cluster by making decisions and giving instructions to worker nodes.

Worker Node : A worker node is a machine in a Kubernetes cluster that runs your applications in containers. It receives instructions from the master node and does the actual work**.**

https://www.linkedin.com/in/arun-kumar-akula-3b5180343/



- This architecture shows a Kubernetes environment with one Control Plane managing two Worker Nodes.

➢ Control plane(Master Node) Components :

API Server : The API Server is the main communication hub for managing the cluster. It receives requests from the UI and CLI (like kubectl).

Scheduler: Assigns newly created pods to suitable worker nodes based on resource availability.

Controller Manager: Responsible for managing the cluster state, monitoring node health, and ensuring the desired number of pod replicas are running.

etcd: A distributed key-value store that stores the all cluster data and configuration information.

➢ Worker Nodes :

Pods : The smallest deployable unit in Kubernetes, uses pods to manage and deploy containers as a single unit on worker nodes.

Containers: These are your actual applications/services running (e.g., NGINX, MySQL).

Kubelet : Runs on each worker node and makes sure that the containers in a pod are healthy and running as expected.

kube-proxy : Handles networking on each node by managing traffic routing and enabling communication between services.

Docker(CRI) : A container runtime used to build and run containers; Kubernetes can also work with other runtimes like containerd.

➢ User Interface(Workstation):

UI: The Kubernetes Dashboard provides a graphical UI to monitor and manage cluster resources.

CLI (kubectl): Command-line tool used by users to interact with the cluster.

How It Works Together:

- A user gives a command using the CLI (kubectl).
- The API Server receives the request from kubectl and updates the information in etcd .
- The Scheduler picks the best worker node to run the pod.
- The Controller Manager makes sure the pod is created as expected.
- The pod is created and containers start running on a worker node.
- The **kubelet** checks that everything is running properly, and **kube-proxy** manages the network so services can talk to each other.

Kubernetes Setup Script (Kops + Kubectl + AWS CLI)

The following Bash script automates the installation of essential tools required to set up and manage a Kubernetes cluster using Kops on AWS :

```
#!/bin/bash

sudo apt update

 sudo apt install -y curl unzip

 curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip" unzip

awscliv2.zip sudo ./aws/install

curl -Lo kops https://github.com/kubernetes/kops/releases/latest/download/kops-linuxamd64

chmod +x kops sudo mv kops /usr/local/bin/

curl -LO "https://dl.k8s.io/release/$(curl -sL https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

chmod +x kubectl
```

Kops Script to Create and Validate a Kubernetes Cluster on AWS :

```
#!/bin/bash set -e

export AWS_ACCESS_KEY_ID="ACCESS_KEY_" export

AWS_SECRET_ACCESS_KEY="SECRET_ACCESS_KEY" export

AWS_DEFAULT_REGION="us-east-1" aws s3 mb s3://arunk8-s3

export KOPS_STATE_STORE="s3://arunk8-s3" kops create cluster \

  --name=arun.k8s.local \

  --zones=us-east-1a,us-east-1b \

  --node-count=2 \

  --image=ami-0b529f3487c2c0e7f \

  --state=${KOPS_STATE_STORE} \

  --yes

export KOPS_STATE_STORE="s3://arunk8-s3" kops validate cluster --name=arun.k8s.local --
state=${KOPS_STATE_STORE} --wait 5m
```

This script sets up a Kubernetes cluster on AWS. It connects to AWS, creates storage for the cluster setup, launches the cluster with 2 nodes, and checks if everything is working properly.

- **After running this script and creating the Kubernetes cluster with Kops, here are some important things you can check:**

- **Check Nodes Status :** using below commands

```
kubectl get nodes
 kubectl get nodes -o wide
```

- **Check Pods (System Components)** : Verifies that system pods (like CoreDNS, kube-proxy) are running fine.

```
kubectl get pods --all-namespaces
```

- **Check Cluster Info :**

```
kubectl cluster-info
```

- **Run a Test Pod (Optional) :**

```
kubectl run nginx --image=nginx --port=80

kubectl get pods
```

- **(Optional) Clean Up :**

```
kops delete cluster --name=arun.k8s.local --
```

It deletes the entire Kubernetes cluster that was created with Kops.