

Container Orchestration with Kubernetes (K8s)

Application Frameworks (SE3040)

Vishan Jayasinghearachchi
Lecturer

Department of Software Engineering, Faculty of Computing

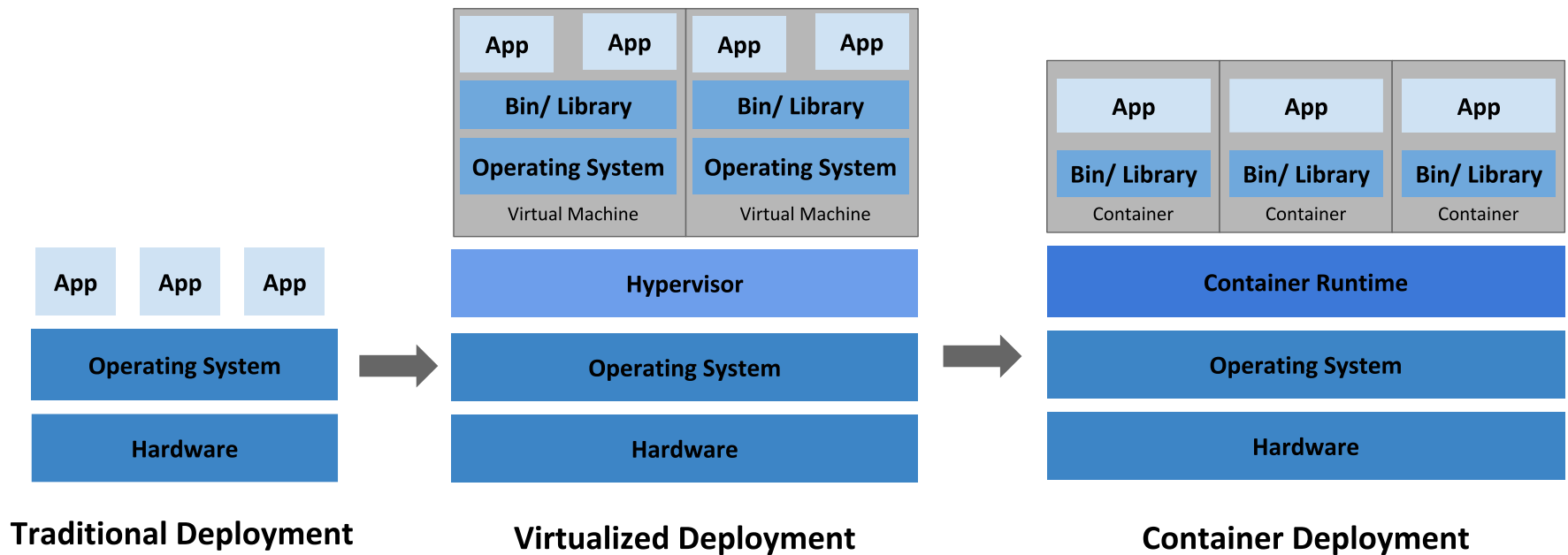
vishan.j@slit.lk



Contents

- Kubernetes Overview
 - What is Kubernetes (K8s)?
 - What can K8s do?
 - K8s Components
 - The Control Plane
 - Nodes
 - How does everything wire together?
- Lesson Recap
- Self-study

Evolution of Deployment Methodologies



Source: <https://kubernetes.io/docs/concepts/overview/>

Container Orchestration

- Why is it needed?
 - In small numbers, one can manually deploy and manage containers.
 - But what would happen in an enterprise scenario, where millions of customers are constantly requesting for different types of services?
Eg: Google search?
 - It is impossible to handle this load manually.
Automation is needed.
 - Container Orchestration is the solution.

It automates deployment, scaling, networking, availability, and lifecycle management of containers in a production environment.

Container Orchestration

- How does it work? [view previous lecture side](#)
 - Usually, a developer writes a configuration file which is understood by the Orchestration tool.
 - This configuration file defines the **desired state** the containerized application (i.e. the containers which make it up) should be in.
 - The orchestration tool, based on this configuration file, maintains the state of the containers (**actual state**) to resemble the desired state.
 - It manages the deployment of the containers, resiliency, selection of the host (to deploy the containers).
 - Once deployed, it manages scalability, availability and performance and even collecting and logging data for later review.

Kubernetes Overview

- What is Kubernetes (K8s)?
 - Kubernetes is a container orchestration platform for scheduling and automating the deployment, management, and scaling of containerized applications.
 - Originally developed by Google. Now owned by Cloud Native Computing Foundation (CNCF).
 - Kubernetes stands for “Helmsman” or “Pilot” in Greek (Hence the ship’s wheel as the logo).
symbolizing control and direction over container “ships”.



kubernetes

Kubernetes Overview

- What can Kubernetes do?

- Container Deployment and Running

- Automated Rollouts and Rollbacks

Manages application updates without downtime and can revert changes if something goes wrong.

- Service Discovery and Load Balancing

- Storage Orchestration

Mounts local or cloud-based storage automatically as needed by applications.

- Autoscaling

Adjusts the number of running containers

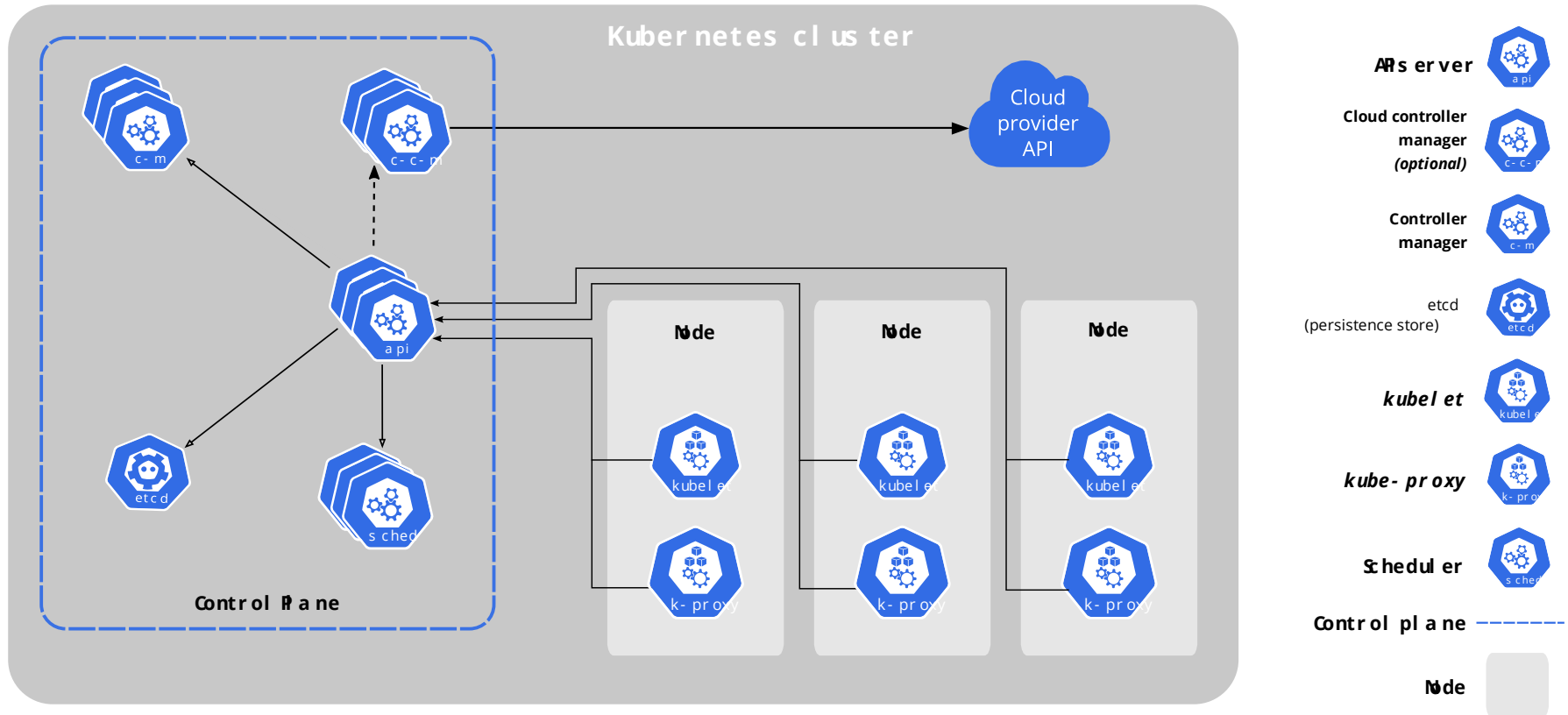
- Self-healing (for High Availability)

Automatically restarts failed containers

- Secret and Configuration Management

Manages sensitive data (like passwords) and app config separately from the application code.

Kubernetes Architecture



Source: <https://kubernetes.io/docs/concepts/overview/components/>

Kubernetes Overview

- K8s components

- **Cluster:** A collection of Nodes. set of machines (nodes) that run containerized applications
- **Nodes/ Worker Nodes:** Actual Compute host (Virtual/ Physical Machine). Deploy, run and manage containerized applications via **Pods**.
Kubelet (agent)
Kube-proxy (networking)
Container runtime (like Docker or containerd)
- **Control Plane:** Manages the Worker Nodes and **Pods** in the Cluster. Manages the cluster state and scheduling
- **Pod:** A group of containers (one or many) which share the same compute resources (node) and same IP address. A **pod** is the **smallest deployable unit** of computing that you can create and manage in Kubernetes. A pod can contain one or more containers.

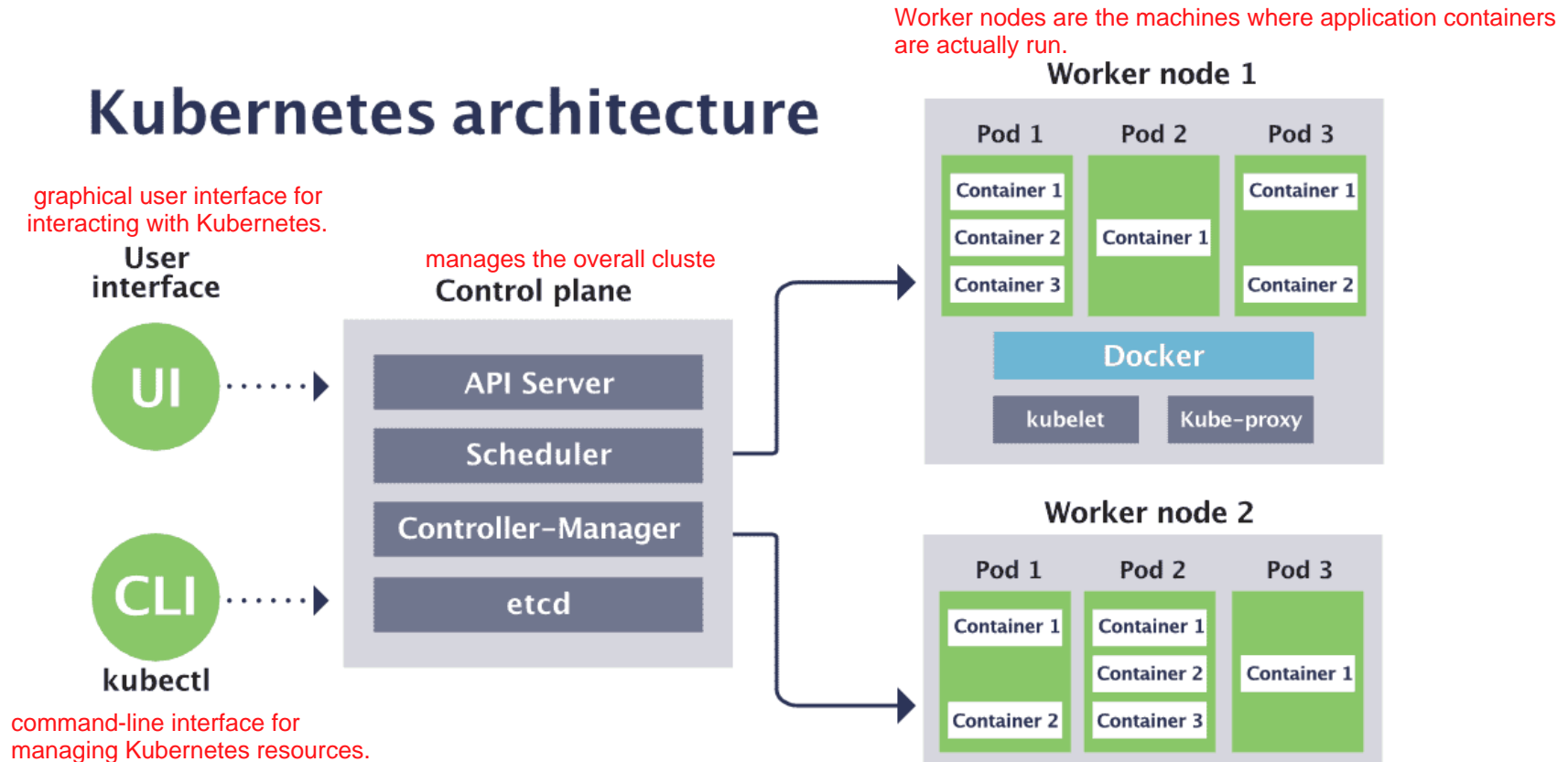
The Control Plane

Control Plane is responsible for managing the overall state of the Kubernetes cluster. It makes global decisions (e.g., scheduling), and responds to events (e.g., restarting a failed pod).

- The control plane's components make global decisions about the cluster, as well as detecting and responding to cluster events.
- Components in the Control Plane:
 - kube-apiserver Acts as the front-end of the Kubernetes control plane.
 - etcd A distributed key-value store. Stores all cluster data (configurations, state)
 - kube-scheduler Assigns newly created pods to nodes
 - kube-controller-manager Runs various controller processes that regulate the state of cluster components.
 - cloud-controller-manager (optional) Manages cloud-specific control logic (e.g., for AWS, GCP, Azure).

Kubernetes Architecture with Pods

Kubernetes architecture



1. **User Interaction:** Users interact with the cluster via UI or kubectl (CLI).
2. **API Server:** Commands are sent to the API Server.
3. **Control Plane:** The Scheduler decides where new pods should run, and the Controller-Manager ensures the desired state is maintained.
4. **etcd:** All cluster state and configuration is stored here.
5. **Worker Nodes:** The kubelet on each worker node receives instructions from the control plane to start or stop containers (via Docker).
6. **Pods and Containers:** The actual applications run inside containers, grouped into pods, on the worker nodes.

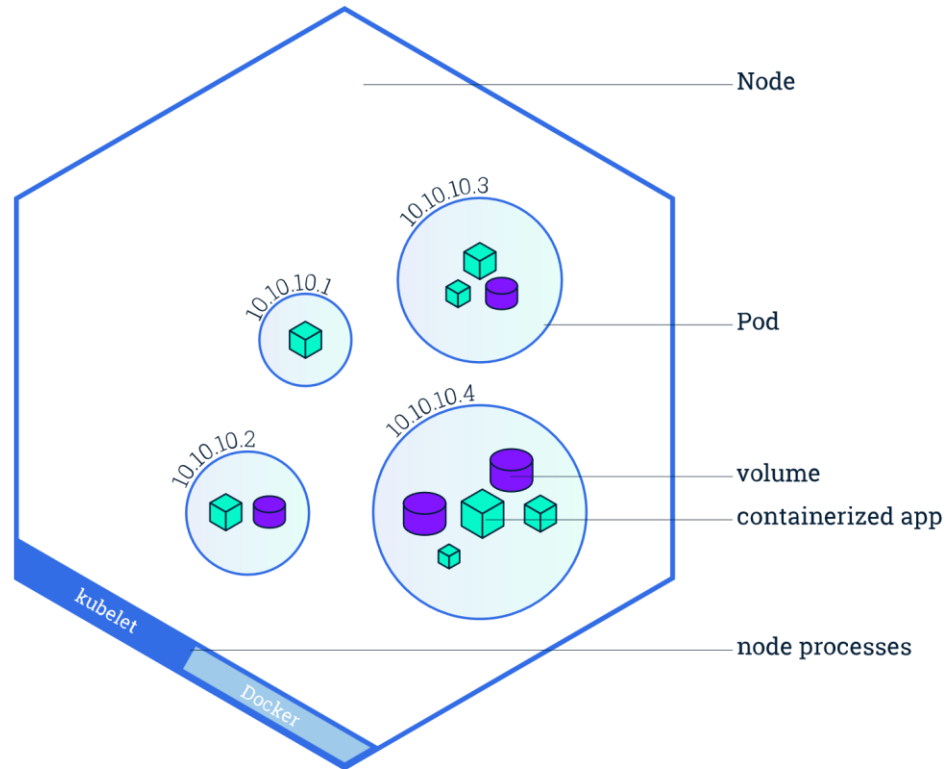
2020/08/Kubernetes-architecture-diagram-1-1.png

Nodes

- Each node runs:
 - **kubelet**, a process responsible for communication between the Kubernetes control plane and the Node; it manages the Pods and the containers running on a machine.
 - **kube-proxy** maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.
 - A **container runtime** (like Docker) responsible for pulling the container image from a registry, unpacking the container, and running the application.

Nodes

- A node.

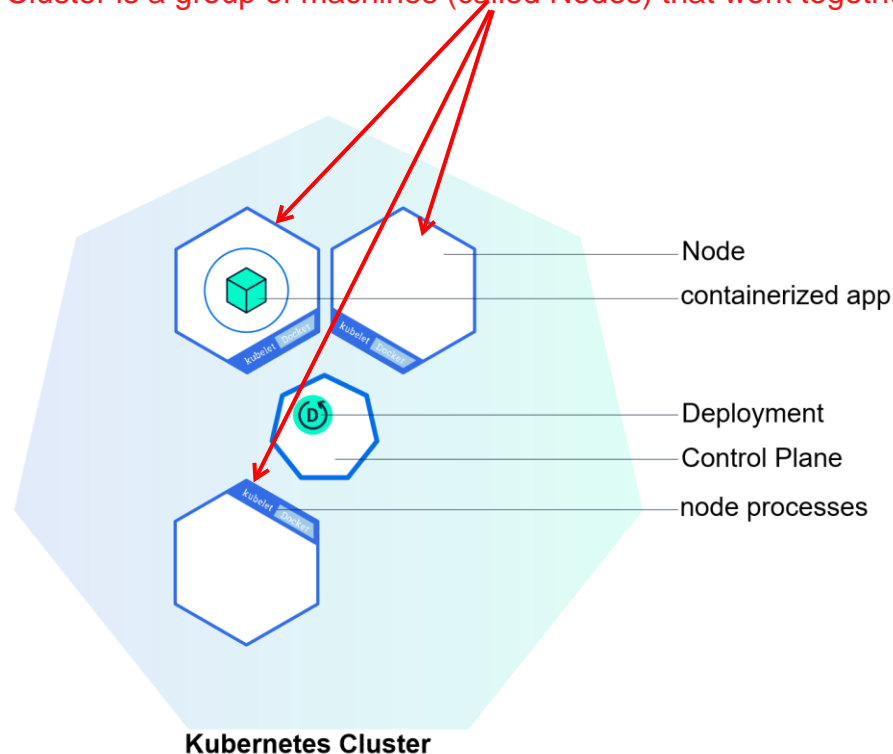


Source: <https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/>

A Cluster

- ## Kubernetes Cluster with a Deployment

A Kubernetes Cluster is a group of machines (called Nodes) that work together to run your containerized applications.




Source: <https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/>

How does everything wire together?

- How do all these wire together?
 - The **desired state** of a K8s cluster can be defined in a configuration called **Deployment**.
 - The Deployment instructs Kubernetes how to create and update instances of your application.
 - The Deployment Controller changes the **actual state** to the **desired state**.
 - E.g.: If the Node hosting an instance goes down or is deleted, the Deployment controller replaces the instance with an instance on another Node in the cluster (self-healing).
 - An **application instance** is hosted in a **Pod**.
 - Traffic to a pod is handled through a **Service**.
 - A Service routes traffic across a set of Pods. Services are the abstraction that allows pods to die and replicate in Kubernetes without impacting your application.

A Deployment

controllers/nginx-deployment.yaml 

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```

Source: <https://kubernetes.io/docs>

What is a Deployment?

A **Deployment** is a set of instructions that tells Kubernetes:

- How many copies (instances) of your application you want to run.
- What container image to use for your application.
- How to update or change your application over time.

The Deployment makes sure that the desired number of application instances are always running in the cluster.

How Does It Work?

1. You create a **Deployment** by telling Kubernetes what app to run and how many copies you want.
2. The **Kubernetes control plane** looks for available nodes and schedules your app instances (called **Pods**) to run on them.
3. The **Deployment Controller** constantly watches your app instances. If a node goes down or a pod crashes, it automatically creates a new pod on another node. This is called **self-healing**.
4. Your app runs inside **Pods**, which are the smallest deployable units in Kubernetes.
5. To send traffic to your pods, Kubernetes uses a **Service** that acts like a load balancer, routing requests to the right pods even if pods are added or removed.

Deployments

- Watch [this video](#) on how Kubernetes Deployments work.

Why is this Important?

Self-healing: If something breaks, Kubernetes fixes it automatically.

Scaling: You can easily increase or decrease the number of app instances.

Rolling updates: You can update your app without downtime.

Load balancing: Services distribute traffic evenly across your app instances.

Some Cloud Offerings

- [AWS Containers Services](#)
- [Azure Container Services](#)

Lesson Recap

- Microservices Architecture
- Containerization
 - Docker
- Container Orchestration
 - Kubernetes

Lesson Recap

- Why did we learn all these?
 - Microservices is an architectural approach to creating cloud native applications.
 - Containers are a way of implementing Microservices.
 - Docker is a platform for packaging, deploying, and running applications in containers.
 - Kubernetes is a system for managing containerized applications across a cluster of nodes.
 - Cloud service providers such as AWS and Azure provide managed Docker and Kubernetes services which can be utilized to run applications without having to manage anything on our own.

Self-study

- [“Learn Kubernetes Basics”](#) – Do this simple hands-on tutorial on your own. No need to install anything.
- Then do [this Azure Kubernetes Service \(AKS\) Tutorial](#) to see how managed services simplify using Kubernetes.

The end



memenetes
@memenetes

...

Every kubernetes tutorial ever



Always refer the official K8s documentation.
Many outdated resources are available
online.

9:01 AM · Jan 31, 2022 · Buffer

Acknowledgements and Additional Reading

- Kubernetes (K8s) Overview
- What is Kubernetes API?