**Kubernetes**

**Microservices Architecture:**

- A microservices architecture consists of a collection of small services. Each service is self-contained and should implement a single business module.

- Each service is a separate codebase, which can be managed by a small development team.

- services can be deployed independently. A team can update an existing service without rebuilding and redeploying the entire application.

- Services communicate with each other by using well-defined APIs. Internal implementation details of each service are hidden from other services.

**Containers:**

- Containers are an application-centric method to deliver high-performing, scalable on any infrastructure of you choice. Conatiners are best suited to deliver microservices by providing protable, isolated virtual environments for applications to run without interference from other running applications.

**Container orchestration:**

- Container orchestrators are tools whixh group systems together to form clusters where containers’ deployment and management is automated at scale while meeting the requirements mentioned above.

- Marathon

- Azure service Fabric

- Docker swarm

- Nomad

- Amazon Elastic Container Service (ECS)

- Kubernetes

Kubernetes as a service:

- Amazon Elastic Kubernetes Service (Amazon EKS)

- Azure Kubernetes Service (AKS)

- Google Kubernetes Engine (GKE)

**what is kubernetes?**

- kubernetes is an open-source system for automating deployment, scaling and management of containerizd applications.

- you can think of kubernetes as the **pilot** on a ship of containers.

Features of kubernetes

- **Scalability**: Kubernetes provides horizontal scaling of pods on the basis of CPU utilization. The threshold for CPU usage is configurable and kubernetes will automatically start new pods if the threshold is reached.

**\* vertical scaling:** means to modify the attributed resources (like CPU or RAM) of each node in the cluster (vertical Pod autoscaler)

\* Horizontal scaling: means modifying the compute resources of an existing cluster, for example by adding new nodes to it or by adding new pods by increasing the replica count of pods (Horizontal Pod autoscaler).

**- self-healing:** kubernetes automatically replaces and reschedules containers from failed nodes. It kills and restarts containers unresponsive to health checks, based on existing rules/policy. it also prevents traffic from being routed to unresponsive containers.

**- Automated rollouts and rollbacks:** kubernetes samelessly rolls out and rolls back application updates and configuration changes, constantly monitoring the application’s health to prevent any downtime.

**- Secret and configuration management:** Kubernetes manages sensitive data and configuration details for an application separately from the container image. Secrets consist of sensitive/confidential information passed to the application without revealing the sensitive content to the stack configuration, like github.

**Kubernetes Architecture:**

**\* worker nodes:**

- worker node provides a running environment for client applications. through containerized microservices, these applications are encapsulated in Pods.

- Pods are the smallest deployable units of computing that you can create and manage in kubernetes.

- A Pod is a group of one or more containers with shared storage and network resources.

**worker node components:**

- Container runtime:

+ CRI (container runtime interface)

- containerd

- CRI-O

- Node Agent (kubelet):

+ agent running on each node and communicates with the control plane components from the master node. It receives Pod definitions, primarily from the API server. and interacts with the container runtime on the node to run containers associated with the Pod. It also monitors the health and resources of Pods running containers.

+ The kubelet connects to container runtimes though a plugin based interface: the Container Runtime Interface (CRI)

- Proxy (kube-proxy):

+ network agent which runs on each node responsible for dynamic updates and maintenance of all networking rules on the node. It abstracts the details of pods networking and forwards connection requests to pods.

+ the kube-proxy is responsible for TCP, UDP and SCTP stream forwarding or round-robin forwarding across a set of Pod backends, and it implements forwarding rules defined by users through service API objects.

- Addons (DNS, Dashboard User Interface, Monitoring and loggings):

+ Addons are cluster features and functionality not yet available in kubernetes therefore implemented through 3rd-party pods and services.

**- master nodes:**

+ The master node provides a ruuning environment for the control plane responsible for managing the state of a kubernetes cluster, and it is the brain behind all operations inside the cluster.

+ In order to communicate with the kubernetes cluster, users send requests to the control plane via a command line interface (CLI) tool, a web User-Interface (Web UI) Dashboard or application Programming interface (API)

+ to persist the kubernetes cluster’s state all cluster configuration data is saved to etcd

Master Node Components(Control Plane Components)

- API server (Kube-proxy)

+ API server intercepts RESTful calls from users, operators and externel agents, then validates and processes them.

+ during processing the API server reads the kubernetes cluster’s current state from the etcd data store and after a call’s execution, the resulting state of the kubernetes cluster is saved in the distributed key-value data store for persistence.

- Scheduler (kube-scheduler):

+ The kubernetes scheduler is a control plane process which assigns Pods to Nodes. The scheduler determines which nodes are valid placements for each Pod in the scheduling queue according to constraints and available resources. The scheduler then ranks each valid Node and binds the Pod to a suitable Node.

- Controller Manager (kube-controller-manager)

+ The controller managers are control plane components on the master node running controllers to **regulate** the state of the kubernetes cluster. Controllers are continuously running and comparing the cluster’s desired state (provided by objects’ configuration data) with its current state (obtained from etcd data store via the API server). In Case of a mismatch corrective action is taken in the cluster until its current state matches the desired state

+ The kube-controller-manager runs controllers responsible to act when nodes become unavailable to ensure pod counts are as expected to create endpoints service accounts and API access tokens

- Cloud Controller Manager (cloud-controller-manager)

+ The cloud controller manager embeds cloud specific control logic. the cloud controller manager lets you link your cluster into your cloud provider’s API, and separates out the components that interact with that cloud platform from components that only interact with your cluster.

- Data Store (etcd)

+ etcd is a distributed reliable key-value store for the most critical data of a distributed system. It is used to persist a kubernetes cluster’s state. only the API server is able to communicate with the etcd data store.

+ etcd’s CLI management tool etcdctl provides backup, snapshot and restore capabilities which come in handy especially for a single etcd instance kubernetes cluster, common in development and learning environments.

Kubectl:

- The kubernetes command-line tool, allows you to run commands against kubernetes clusters. You can use kubectl to deploy applications, inspect and manage cluster resources and view logs.

**Kubernetes Concepts Explained**

**Pod:** refers to one or more containers that should be controlled as a single application. A pod encapsulates application containers, storage resources, a unique network ID and other configuration on how to run the containers.

**Node:** A node is a worker machine in kubernetes and may be either a virtual or a physical machine, depending on the cluster. Each Node is managed by the control plane. A node can have multiple pods and the kubernetes control plane automatically handles scheduling the pods across the nodes in the cluster.

**cluster:** A kubernetes cluster is a set of nodes that run containerized applications. Kubernetes clusters allow containers to run across multiple machines and environments: virtual, physical, cloud-based and on-promises. Kubernetes containers are not restricted to a specific operating system, unlike virtual machines. Instead, they are able to share operating systems and run anywhere

**Namespaces:** namespaces are a way to organize clusters into virtual sub-clusters, they can be helpful when different teams or projects share a kubernetes cluster.

**Service:** A kubernetes service is a logical abstraction for a deployed group of pods in a cluster which all perform the same function. Since pods are ephemeral, a service enables a group of pods, which provide specific functions to be assigned a name and a unique IP address.

**deployment:** A kubernetes deployment is used to tell kubernetes how to create or modify instances of the pods that hold a containerized application. Deployments can scale the number of replica pods, enable rollout of updated code in a controlled manner, or roll back to an earlier deployment version if necessary.

**Workloads:** A workload is an application running on kubernetes. whether your workload is a single component or several that work together, on kubernetes you run it inside a set of pods.

**Volume:** Similar to a container volume in docker, but a kubernetes volume applies to a whole pod and is mounted on all containers in the pod. The volume will be remeoved only when the pod gets destroyed. Also a pod can have multiple volumes assoiciated.

**ReplicaSet:** A replicaset’s purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the avalability of a specified number of identical Pods.

**Ingress:** An API object that provides routing rules to manage external users’ access to the services in a kubernetes cluster, typically via HTTPS/HTTP, Ingress may provide load balancing, SSL termination and name-based virtual hosting.

**Orchestration:** Container orchestration is the automation of much of the operational effort required to run containerized workloads and services. This includes a wide range of things software teams need to manage a container’s lifecycle, including provisioning, deployment, scaling (up and down), networking, load balancing.