Kubernetes 🡺

package your code 🡺 configure file and application 🡺 add dependencies 🡺 build 🡺 docker image🡺 RUN 🡺 container

Kubernetes nodes 🡺 node is a physical or virtual server which holds the container

Kubernetes architecture 🡺

A picture containing icon

Description automatically generated

EKS 🡺 Amazon Elastic Kubernetes services

Kubernetes control plane - self managed🡺

to manage the Kubernetes control plane, we need to do below things

Need to make Control Plane Highly Available

Maintain multiple EC2 in multiple AZ

Scale Control Plane if needed

Keep etcd up and running

Overhead of managing EC2s

AMI Rehydration

Security Patching

Replace failed EC2s

Orchestration for Kubernetes Version Upgrade

Kubernetes control plane- AWS managed 🡺

AWS maintains High Availability - Multiple EC2s in Multiple AZs

AWS Detects and Replaces Unhealthy Control Plane Instances

AWS Scales Control Plane

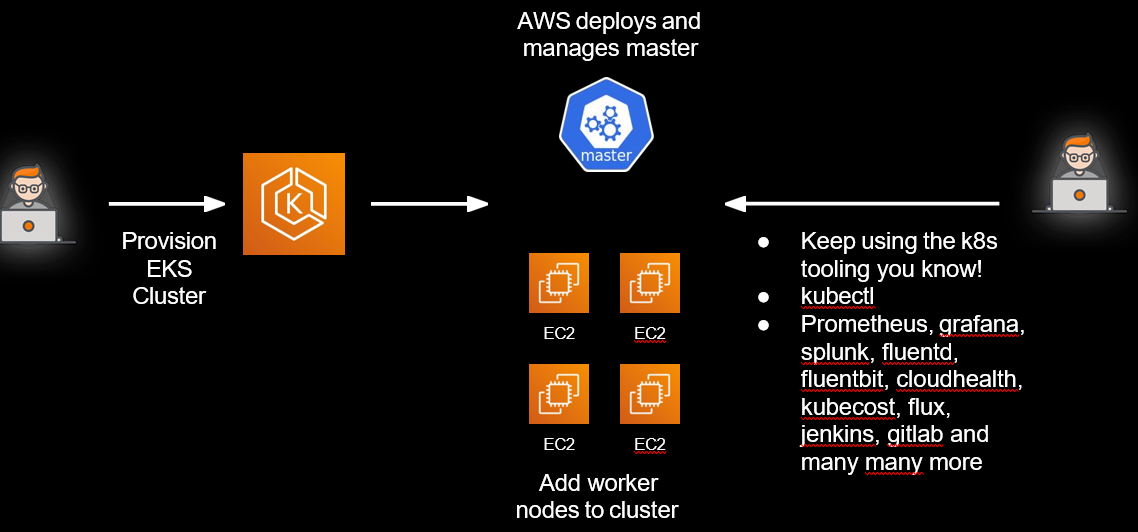
AWS Maintain etcd

Provides Automated Version Upgrade and Patching

Supports Native and Upstream Kubernetes

Integrated with AWS Ecosystem

-



Self managed node groups 🡺

You maintain worker EC2s

You orchestrate version upgrade, security patching, AMI Rehydration, keeping pods up during upgrade

Can use custom AMI

Amazon managed node groups 🡺

AWS manages worker EC2s

AWS provides AMI with security patches, version upgrade

AWS manages pod disruption during upgrade

Doesn’t work with custom AMI

AWS fargate 🡺

No worker EC2 whatsoever!

You define and deploy pods

Container + Serverless!

How an user access EKS 🡺

Graphical user interface

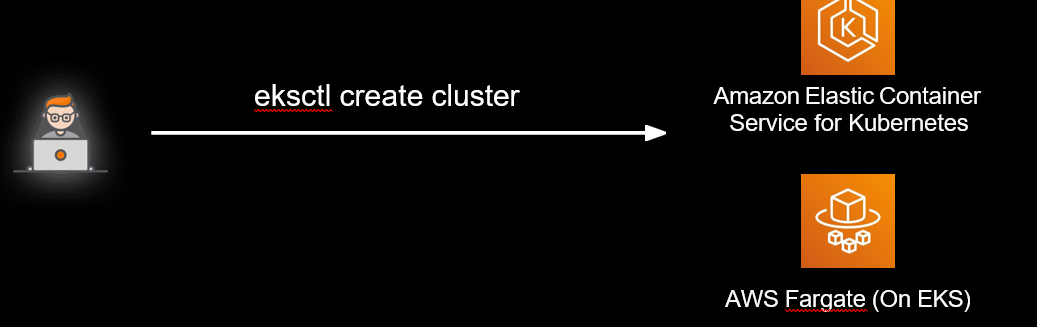
Description automatically generated

eksctl 🡺

it is a CLI tool for creating cluster on EKS.

easier that console for real

Abstracts lots of stuffs - vpc,subnets,sec,group etc using cloud formation



Available eksctl features (Only on EKS)

Create, get, list and delete clusters

Create, drain and delete nodegroups

Scale a nodegroup

Update a cluster

Use custom AMIs

Configure VPC Networking

Configure access to API endpoints

Support for GPU nodegroups

Spot instances and mixed instances

IAM Management and Add-on Policies

List cluster Cloudformation stacks

Install coredns

Write kubeconfig file for a cluster

eksctl commands 🡺

cerate eks cluster with one node group containing 2 m5.large nodes 🡺

**eksctl create cluster**

cerate eks cluster with K8 version 1.15 with 2 t3.micro nodes 🡺

**eksctl create cluster --name <name> --version 1.15 --node-type t3.micro --nodes 2**

create eks cluseter with managed node group 🡺

**eksctl create cluster --name <name> --version 1.15 --nodegroup-name**

**<nodegrpname> --node-type t3.micro --nodes 2 --managed**

eks cluster with fargate profile 🡺

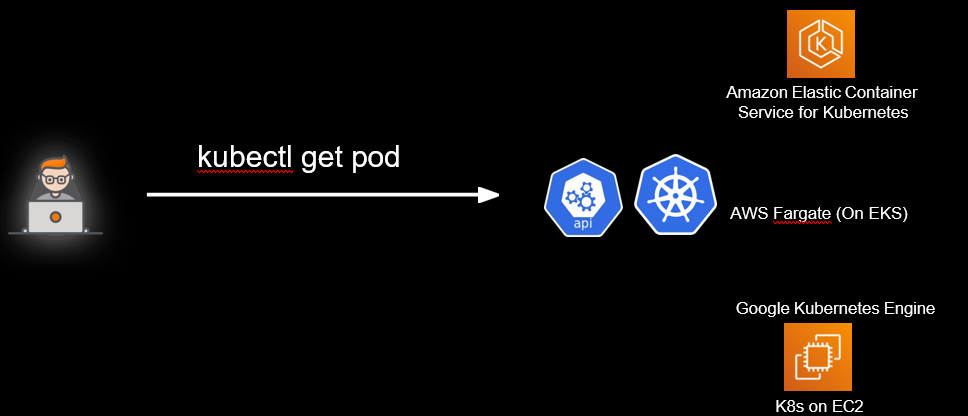
**eksctl create cluster --name <name> --fargate**

**kubectl 🡺**

**kubectl is the cli for running commands against a cluster on K8s resources.**

**communicate via cluster api server.**

**works for any k8s cluster - EKS , k8 on EC2, GKE.. etc.**

****

**kubectl command syntax 🡺**

**Diagram

Description automatically generated**

Create resources based on manifest. Declarative Way! Best Way! 🡺

**kubectl apply -f ./manifest-file.yaml**

**kubectl get nodes 🡺 list all node info**

**kubectl get services 🡺 list all services**

**kubectl get pods -o wide 🡺 list pods with more details**

**kubectl get pod my-pod -o yaml 🡺 get a pod’s yaml**

**kubectl get deployment my-dep 🡺 list a particular deployment**

**lubectl exec -it <podname> -- /bin/bash 🡺get a shell to the running container**

**LAB SETUP FOR EKS 🡺**

**how to setup aws CLI 🡺**

**in windows: install the awscli msi file. check aws site and download it. once itstalled , check the version.**

**aws --version**

**create one user with admin access.**

**go to aws🡺 IAM > create user > admin profile >**

**Install EKSCTL 🡺** **https://docs.aws.amazon.com/eks/latest/userguide/eksctl.html**

**Install kubectl 🡺**

**KUBERNETES \*\*\***

Monolithic application:

1. \*\*Monolithic Architecture:\*\*

- \*\*Definition:\*\* In a monolithic architecture, the entire application is designed as a single, tightly integrated unit. All the components and modules of the application are interconnected and interdependent.

- \*\*Characteristics:\*\*

- Single codebase.

- Centralized database.

- Scaling involves replicating the entire application.

- Changes to one part of the application may require rebuilding and redeploying the entire monolith.

- \*\*Advantages:\*\*

- Simplicity in development and deployment.

- Easier to debug and test.

- \*\*Disadvantages:\*\*

- Scaling can be challenging.

- Updates may require downtime.

- Lack of flexibility in choosing technologies for different components.

A diagram of a computer

Description automatically generated

Microservices :

A diagram of a software application

Description automatically generated

2. \*\*Microservices Architecture:\*\*

- \*\*Definition:\*\* In a microservices architecture, an application is broken down into a collection of small, independent services that communicate with each other through APIs. Each service is developed, deployed, and scaled independently.

- \*\*Characteristics:\*\*

- Decentralized architecture.

- Each microservice has its own database.

- Independent scaling of services.

- Flexibility in choosing technologies for each service.

- \*\*Advantages:\*\*

- Scalability and flexibility.

- Independent development and deployment of services.

- Fault isolation – issues in one service don't affect others.

- \*\*Disadvantages:\*\*

- Increased complexity in managing a distributed system.

- Service communication overhead.

- Potential challenges in data consistency between services.

**3. \*\*Kubernetes:\*\***

- \*\*Definition:\*\* Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It provides a set of tools for deploying and managing applications in containers at scale.

- \*\*Characteristics:\*\*

- Orchestration of containers.

- Automated scaling and load balancing.

- Rolling updates and rollbacks.

- Declarative configuration and automation.

- it schedules, runs and manages isolated containers which are running on virtual /physical/cloud machine.

- \*\*Advantages:\*\*

- Efficient resource utilization.

- High availability and fault tolerance.

- Simplified deployment and management of containerized applications.

- \*\*Disadvantages:\*\*

- Learning curve for beginners.

- Requires careful configuration and monitoring.

- Overhead in small-scale applications.

\*\*Integration:\*\*

- \*\*Microservices with Kubernetes:\*\* Kubernetes is commonly used to deploy and manage microservices. Each microservice can be containerized, and Kubernetes orchestrates the deployment, scaling, and management of these containers. It helps address some of the challenges associated with managing distributed microservices.

In summary, while monolithic architecture is a traditional and simple approach, microservices architecture and Kubernetes offer solutions to the challenges of scalability, flexibility, and efficient management in modern, complex, and distributed applications. The choice between monolithic and microservices architectures often depends on the specific requirements and goals of the application and the organization. Kubernetes, on the other hand, plays a crucial role in managing containerized applications, whether they follow a monolithic or microservices architecture.

Kubernetes history:

* Google developed an internal system called Borg later named as omega to deploy and manage thousands of google application and services on this cluster.
* In 2014 , google introduced Kubernetes an open source platform written in Golang and later donated to CNCF(cloud native computing foundation)

Kubernetes platfrm for k8s

* Kubernetes playground
* Play with k8s
* Play with Kubernetes classroom

Cloud based K8s services:

* GKE: google Kubernetes services
* AKS: azure Kubernetes services
* Amazon EKS: elastic Kubernetes services

Kubernetes installation tool:

Minicube

Kubeadm

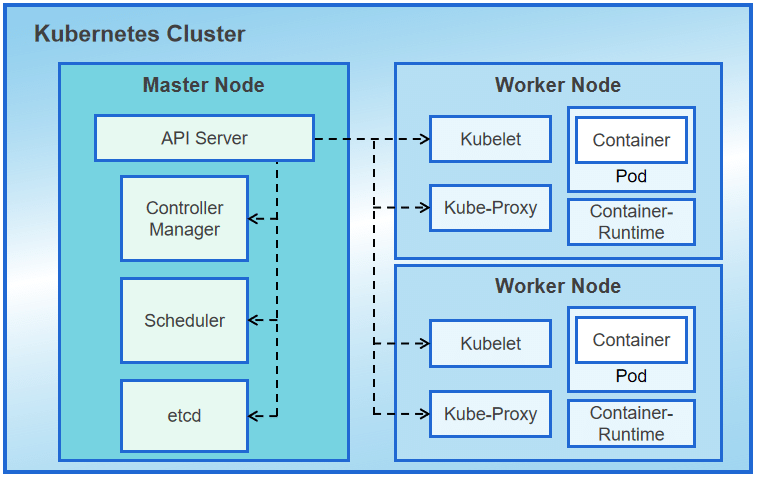
Problems with scaling up the containers:

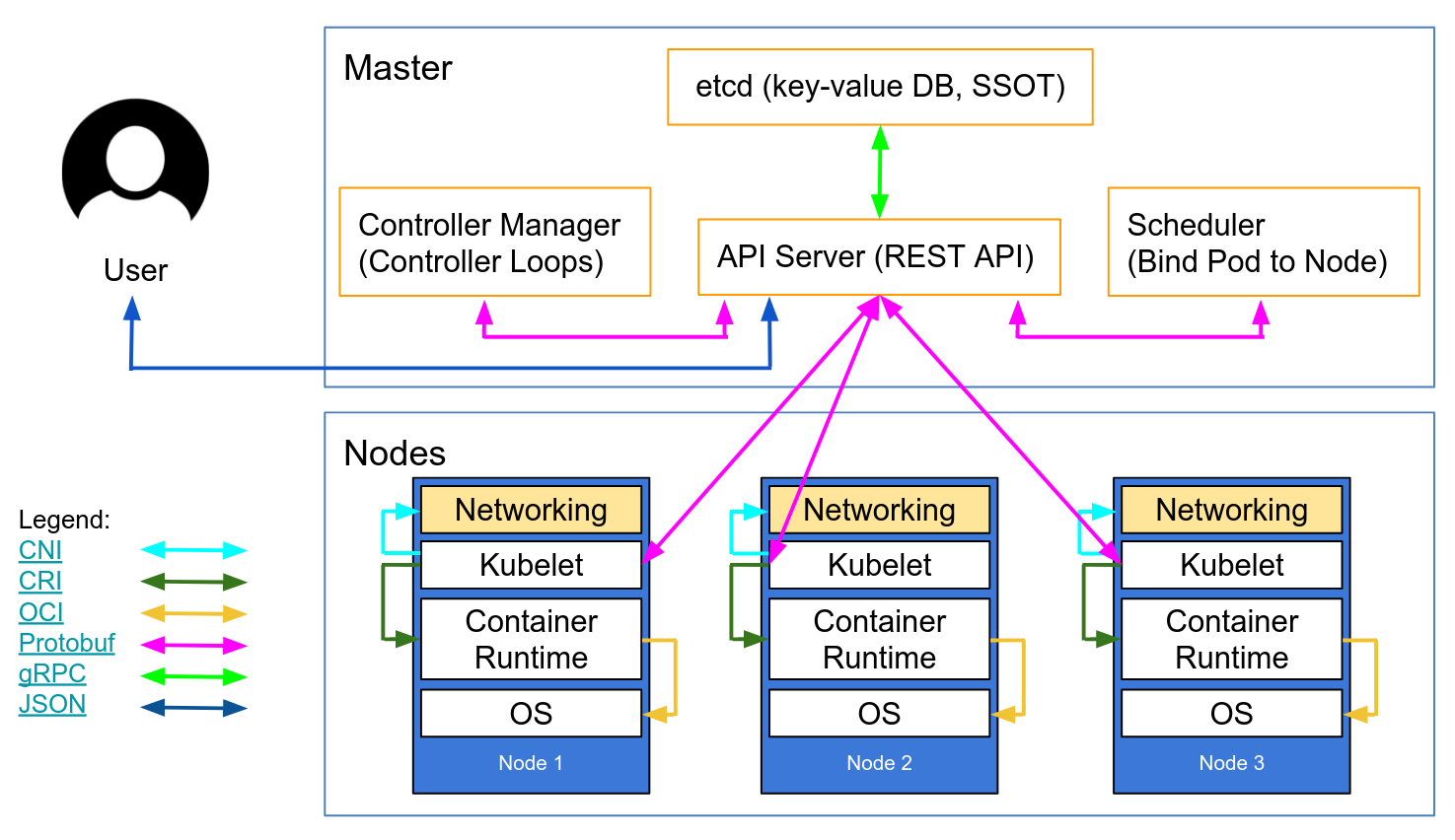
* Containers cannot communicate with each other
* Autoscaling and load balancing was not possible
* Container had to be managed carefully.

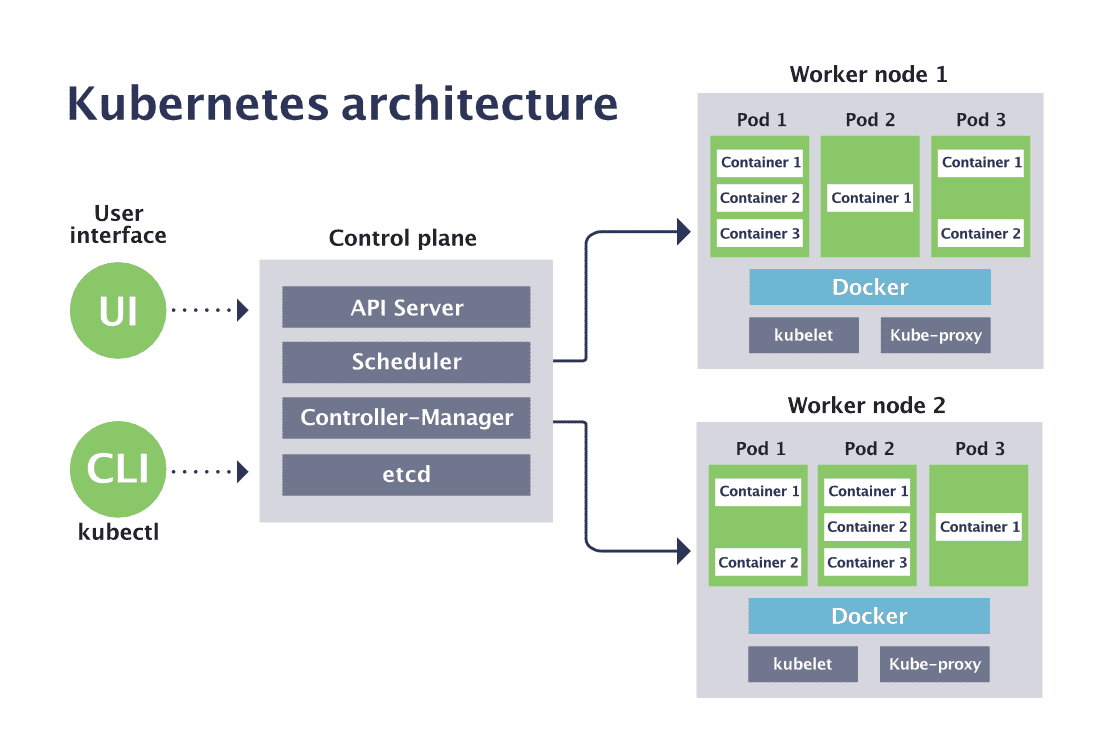
Features of Kubernetes:

* Orchestration (clustering of any no of containers running on different networks)
* Autoscaling (vertical & Horizontal)
* Load balancing
* Auto healing
* Platform independent (cloud/virtual/phisycal)
* Fault tolerance(nod/pod failure)
* Rollback (going back to previous version)
* Health monitoring of containers
* Batch execution (one time sequential, parallel)

|  |  |  |
| --- | --- | --- |
| Feature | Kubernetes | Docker SWARM |
| Installation and cluster configuration | Complicated and time consuming | Fats and easy |
| Supports | K8s can work with almost all container types like rocket,docker,containerD | Work with docker only |
| GUI | GUI is available | GUI not available |
| Data volumes | Only shared with containers in same pod | Can be shared with any other container |
| Updates & Rollback | Process scheduling to maintain services while updating | Progressive updated of services health monitoring throughout the update |
| Autoscaling | Supports both vertical n horizontal | No support |
| Logging and monitoring | Inbuilt tool present for monitoring | Used 3rd party tools like splunk |







Working with Kubernetes

* We create manifest (yml)
* Apply this tri cluster (master) t bring into desired state
* Pod runs on node, which is controlled by master.

Role of master node:

* Kubernetes cluster contains containers running or bare metal/vm instances/cloud instances/all mix
* Kubernetes designate s one or more of these as master and all others as workers
* The master is now going to run set of K8s process . These process will ensure smooth functioning of cluster . these process are called control plabe.
* Can be multi master for high availability.
* Master runs control plane to run cluster smoothly.

Etcd :

etcd is a distributed key-value store that is a critical component in the architecture of Kubernetes. It is used to store and manage the configuration data, state information, and metadata of a Kubernetes cluster. The name "etcd" stands for "distributed etc directory," highlighting its role as a distributed configuration store.

- \*\*Role in Kubernetes:\*\* etcd is a distributed key-value store that serves as the primary data store for Kubernetes. It stores the configuration data, metadata, and the current state of the entire cluster.

### 2. \*\*Distributed Key-Value Store:\*\*

- \*\*Consistency:\*\* etcd provides strong consistency, ensuring that all nodes in the cluster see the same data at the same time.

- \*\*Atomic Transactions:\*\* Operations in etcd are atomic, meaning they either fully succeed or fail.

### 3. \*\*Cluster Coordination:\*\*

- \*\*Raft Consensus Algorithm:\*\* etcd uses the Raft algorithm for distributed consensus, facilitating leader election and maintaining consistent data across the cluster.

- \*\*Leader Node:\*\* One node in the etcd cluster serves as the leader, handling updates and ensuring consensus.

### 4. \*\*Data Model and Storage:\*\*

- \*\*Key-Value Pairs:\*\* etcd organizes data as key-value pairs, where each key represents a unique identifier for configuration or state information.

- \*\*Hierarchical Structure:\*\* Keys can be organized hierarchically, allowing for structured storage.

### 5. \*\*API and Watch Mechanism:\*\*

- \*\*HTTP API:\*\* etcd exposes a RESTful HTTP API, allowing users and Kubernetes components to interact with it.

- \*\*Watch Mechanism:\*\* Kubernetes components, like the API server, use etcd's watch mechanism to be notified of changes in real-time, enabling efficient event-driven architecture.

### 6. \*\*Kubernetes Integration:\*\*

- \*\*API Server Backend:\*\* etcd serves as the backend storage for the Kubernetes API server.

- \*\*Cluster Consistency:\*\* All changes to the cluster, such as pod creation or service updates, are coordinated through etcd, ensuring a consistent view.

### 7. \*\*Security Features:\*\*

- \*\*Authentication:\*\* etcd supports client authentication mechanisms to control access to the cluster.

- \*\*Transport Security:\*\* Communication between etcd nodes and clients is secured using Transport Layer Security (TLS).

### 8. \*\*Backup and Restore:\*\*

- \*\*Snapshot Functionality:\*\* etcd allows administrators to create snapshots, capturing the current state of the cluster for backup purposes.

- \*\*Data Restoration:\*\* Snapshots can be used to restore the etcd cluster to a previous state in case of data loss or corruption.

### 9. \*\*Cluster Scaling:\*\*

- \*\*Horizontal Scaling:\*\* etcd clusters can be horizontally scaled by adding more nodes, ensuring scalability as the Kubernetes cluster grows.

- \*\*Load Balancing:\*\* Load balancing mechanisms can be employed to distribute requests across etcd nodes for better resource utilization.

### 10. \*\*etcdctl Command-Line Tool:\*\*

- \*\*Administrative Tool:\*\* etcd provides a command-line tool called etcdctl for administrators to interact with etcd, enabling operations like querying and managing the cluster.

- \*\*Troubleshooting:\*\* etcdctl is a valuable tool for troubleshooting and monitoring the state of the etcd cluster.

### 11. \*\*Version Compatibility:\*\*

- \*\*Kubernetes Compatibility:\*\* Different versions of Kubernetes have specific requirements for etcd versions. It's crucial to check the Kubernetes documentation for compatibility guidelines.

Understanding these key aspects of etcd is essential for those managing and maintaining Kubernetes clusters, as etcd's reliability and consistency directly impact the stability and correctness of the entire Kubernetes environment.

The Kubernetes Scheduler is a crucial component in the Kubernetes control plane that automates the placement of workloads (such as pods) onto nodes within a cluster. Its primary responsibility is to ensure efficient resource utilization, high availability, and adherence to user-defined constraints and policies. Let's explore the Kubernetes Scheduler in detail:

### Key Responsibilities:

1. \*\*Pod Scheduling:\*\*

- The scheduler decides which node within the cluster a newly created pod should be scheduled to.

- It takes into consideration various factors, including resource requirements, node capacity, and user-defined constraints.

2. \*\*Node Selection:\*\*

- Evaluates the available nodes in the cluster to determine the best fit for a pod.

- Factors such as CPU and memory resources, hardware constraints, and node affinity/anti-affinity rules influence node selection.

3. \*\*Pod Affinity and Anti-Affinity:\*\*

- Supports pod affinity and anti-affinity rules that allow users to influence the placement of pods based on their relationships with other pods or nodes.

4. \*\*Resource Constraints:\*\*

- Ensures that the resource requirements and limits specified in a pod's configuration are taken into account during scheduling.

- Helps prevent resource contention and ensures optimal resource utilization across nodes.

5. \*\*Topology Awareness:\*\*

- Understands the topology of the cluster, including node location and network proximity.

- Utilizes this information for making intelligent scheduling decisions to enhance performance and reduce latency.

6. \*\*Interoperability with Custom Schedulers:\*\*

- Kubernetes allows users to implement and integrate custom schedulers if the default scheduler does not meet specific requirements.

- Custom schedulers can implement policies tailored to the unique needs of the cluster.

### How the Scheduler Works:

1. \*\*Pod Creation Request:\*\*

- When a user creates a pod, they define its resource requirements, constraints, and other configuration parameters.

2. \*\*API Server:\*\*

- The pod creation request is initially handled by the Kubernetes API server, which stores the pod's desired state.

3. \*\*Scheduler Watches for Unscheduled Pods:\*\*

- The Scheduler continuously watches for unscheduled pods by querying the Kubernetes API server.

4. \*\*Filtering and Scoring:\*\*

- The Scheduler applies a set of filtering and scoring algorithms to evaluate each node in the cluster based on resource requirements, affinity rules, and other factors.

- Filtering eliminates nodes that do not meet basic requirements, while scoring ranks the remaining nodes.

5. \*\*Selecting the Best Node:\*\*

- The Scheduler selects the node with the highest score as the optimal placement for the pod.

6. \*\*Binding the Pod:\*\*

- The Scheduler then updates the pod's desired state to include the selected node, and the pod is marked as "Scheduled."

7. \*\*Controller Manager:\*\*

- The Kubernetes Controller Manager notices the change in the pod's desired state and triggers the creation of the pod on the selected node.

### Important Points to Remember:

- \*\*Extensibility:\*\* Kubernetes allows for the extension and replacement of the default scheduler with custom schedulers to meet specific use cases.

- \*\*User-Defined Policies:\*\* Through labels, node affinity, anti-affinity rules, and other mechanisms, users can express policies that guide the scheduler's decision-making process.

- \*\*Integration with Node Auto-Scaler:\*\* The scheduler works in conjunction with the Kubernetes Node Auto-Scaler, which automatically adjusts the number of nodes in the cluster based on resource demand.

- \*\*Event-Driven Operation:\*\* The scheduler operates in an event-driven manner, continuously monitoring the state of the cluster and making scheduling decisions as new pods are created.

- \*\*Avoiding Single Point of Failure:\*\* Kubernetes allows deploying multiple schedulers to avoid a single point of failure. Each scheduler can be configured to manage different pods or namespaces.

In summary, the Kubernetes Scheduler is a critical component that plays a pivotal role in optimizing resource allocation, ensuring high availability, and respecting user-defined policies within a Kubernetes cluster. Understanding its behavior and capabilities is essential for efficient cluster management.

In Kubernetes, the Controller Manager is one of the core components of the control plane. It is responsible for managing various controllers that regulate the state of the cluster, ensuring that the desired state matches the current state. Each controller focuses on a specific aspect of the system, such as nodes, pods, or services. Let's explore the Kubernetes Controller Manager in detail:

### Key Responsibilities:

1. \*\*Node Controller:\*\*

- \*\*Objective:\*\* Ensures that the correct number of nodes is running and healthy in the cluster.

- \*\*Actions:\*\* It detects and addresses issues related to nodes, such as node failures or changes in capacity.

2. \*\*Replication Controller/ReplicaSet Controller:\*\*

- \*\*Objective:\*\* Ensures the desired number of pod replicas are running at all times.

- \*\*Actions:\*\* If the actual number of replicas diverges from the desired state, the controller creates or deletes pods to reconcile the difference.

3. \*\*Endpoints Controller:\*\*

- \*\*Objective:\*\* Populates the Endpoints object (a list of IP addresses and ports) for services that match a set of pods.

- \*\*Actions:\*\* Monitors the pods and services, updating the Endpoints object when changes occur.

4. \*\*Service Account & Token Controller:\*\*

- \*\*Objective:\*\* Creates default accounts and access tokens for new namespaces.

- \*\*Actions:\*\* Monitors the creation of namespaces and ensures that associated service accounts and tokens are generated.

5. \*\*Service Controller:\*\*

- \*\*Objective:\*\* Maintains and updates the set of services in the cluster.

- \*\*Actions:\*\* Listens for changes in services and manages the associated load balancers or endpoints.

6. \*\*Namespace Controller:\*\*

- \*\*Objective:\*\* Maintains the set of namespaces in the cluster.

- \*\*Actions:\*\* Ensures that namespaces are created, updated, or deleted as needed.

7. \*\*Persistent Volume (PV) Controller:\*\*

- \*\*Objective:\*\* Manages the lifecycle of Persistent Volumes in the cluster.

- \*\*Actions:\*\* Ensures that PVs are dynamically provisioned or reclaimed based on the storage classes and claims.

8. \*\*Persistent Volume Claim (PVC) Controller:\*\*

- \*\*Objective:\*\* Manages the lifecycle of Persistent Volume Claims in the cluster.

- \*\*Actions:\*\* Creates, updates, or deletes PVCs based on user demand.

### Controller Manager Components:

1. \*\*Controller Manager Binary:\*\*

- The Controller Manager is implemented as a single binary (`kube-controller-manager`) that combines the functionality of multiple individual controllers.

2. \*\*Separation of Concerns:\*\*

- Each controller operates independently, focusing on a specific aspect of the cluster's state. This separation of concerns simplifies the design and enhances maintainability.

3. \*\*Custom Controllers:\*\*

- Kubernetes allows users to create custom controllers tailored to specific needs, extending the system's capabilities beyond the built-in controllers.

### Controller Manager Operation:

1. \*\*Watch-Act Cycle:\*\*

- Each controller operates on a watch-act cycle. It watches the current state of resources in the cluster and takes actions to reconcile any differences between the current state and the desired state.

2. \*\*Event-Driven:\*\*

- The Controller Manager operates in an event-driven manner, responding to changes in the cluster's state and taking appropriate actions.

### Important Points to Remember:

- \*\*High Availability:\*\* To enhance high availability, multiple instances of the Controller Manager can be run in the cluster, each handling different sets of controllers.

- \*\*Separation of Concerns:\*\* Each controller focuses on a specific aspect of the system, allowing for modularity and ease of maintenance.

- \*\*Extensibility:\*\* Kubernetes provides an extensible framework for creating custom controllers, allowing users to implement controllers that address specific requirements.

- \*\*Consistency:\*\* The Controller Manager ensures that the desired state specified by users or applications is consistently maintained across the cluster.

- \*\*Observability:\*\* Monitoring the logs and metrics of the Controller Manager is important for understanding its behavior and diagnosing potential issues.

Understanding the role and responsibilities of the Controller Manager is crucial for effectively managing and maintaining a Kubernetes cluster, as it plays a central role in orchestrating the state of the system.

WORKER NODE components:

Kublet:

* Agent running on the node
* Listens to kubernets master.
* Use port 10255
* Send success/fail report to master

Container engine :

* Works with kubelet
* Pulling images
* Start stop containers
* Exposing containers on ports specified in manifest

Kube proxy :

* Assign IP to each pod
* It is required to assign IP address to pods
* Kube-proxy runs on each node and this make sure that each pod will get its wn unique ip address
* These 3 components collectively consist node

PODS:

* Smallest unit in Kubernetes
* POD is a group of one or more containers that are deployed together on the same host.
* A cluster is a group of nodes.
* A cluster has atleast one worker node and master node
* In kuberentes the control unit is the pod, not container
* Consists of one or more tightly couples containers
* Pod runs on node which is controlled by master
* Kubernets only knows about pods(does not know about individual container)
* Can not start contains without a pod
* One pod usually contains one container

Multi – container PODS:

* Share access to memory space.
* Connect to each other using local host (container port)
* Share access to the same volume
* Containers within pod are deployed in an all-or-nothing manner.
* Entire pod is hosted on the same node(scheduler will decide about which node)

POD limitations:.

* No by default auto scaling or auto healing , you can do it using higher level Kubernetes objects
* POD crashes

Higher level Kubernetes objects:

* Replication set: auto scaling and auto healing
* Deployment : versioning and rollback
* Service: static ip and networking
* Volume:; non-ephemeral storage

Important:

Kubectl : single cloud

Kubeadm : on premises

Kubefed

KUBERNETES LAB SETUP\*\*\*\*\*

Execute these commands in master node

**apt-get install apt-transport-https**

**apt install docker.io -y**

**docker --version**

**systemctl start docker**

**systemctl enable docker**

**sudo apt-get install gnupg**

**sudo curl -s https://packages.cloud.google.com/apt|sudo apt-key add**

**sudo curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg|sudo apt-key add**

**vi /etc/apt/sources.list.d/kubernetes.list 🡺** deb [http://apt.kubernetes.io/](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbEFjem5HNjZQWlpyNTZWeTNTbzlXMkR5X1JYQXxBQ3Jtc0tuT09QYVpuM0V0WHUteDZnSFhaeVVvTGoyLWJ6NXN4Rkh1dXNST0g1MWtER2w1V050eUJJZ0JmMWtCb0t2RmdPZERVeFFfdzV1a1JNX0lTNXEtc0dRMGdORjRKM2hDR2ZLd1B0WWx1OVZTRTdaOHJYUQ&q=http%3A%2F%2Fapt.kubernetes.io%2F&v=ftrAFHL6w2c) kubernetes-xenial main

**apt-get update**

**apt-get install -y kubelet kubeadm kubectl kubernetes-cni**

kubeadm init

mkdir -p $HOME/.kube

cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

chown $(id -u):$(id -g) $HOME/.kube/config

sudo kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

sudo kubectl apply -f <https://raw.githubusercontent.com/coreos/flannel/master/Documentation/k8s-manifests/kube-flannel-rbac.yml>

**Execute these commands in worker node**

**apt-get update**

**apt-get install apt-transport-https**

**apt install docker.io -y**

**docker --version**

**systemctl start docker**

**systemctl enable docker**

**sudo curl -s https://packages.cloud.google.com/apt... | sudo apt-key add**

**sudo apt-get install gnupg**

**sudo curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg|sudo apt-key add**

**vi /etc/apt/sources.list.d/kubernetes.list**

**apt-get update**

**apt-get install -y kubelet kubeadm kubectl kubernetes-cni**

**kubeadm join 10.0.0.4:6443 --token 9d74t8.7fbc7tg8dg9u2c3c --discovery-token-ca-cert-hash sha256:7a9742b0b6498564de23f481f70bc32374364e5ce5003a9e660d38cc8697797f**