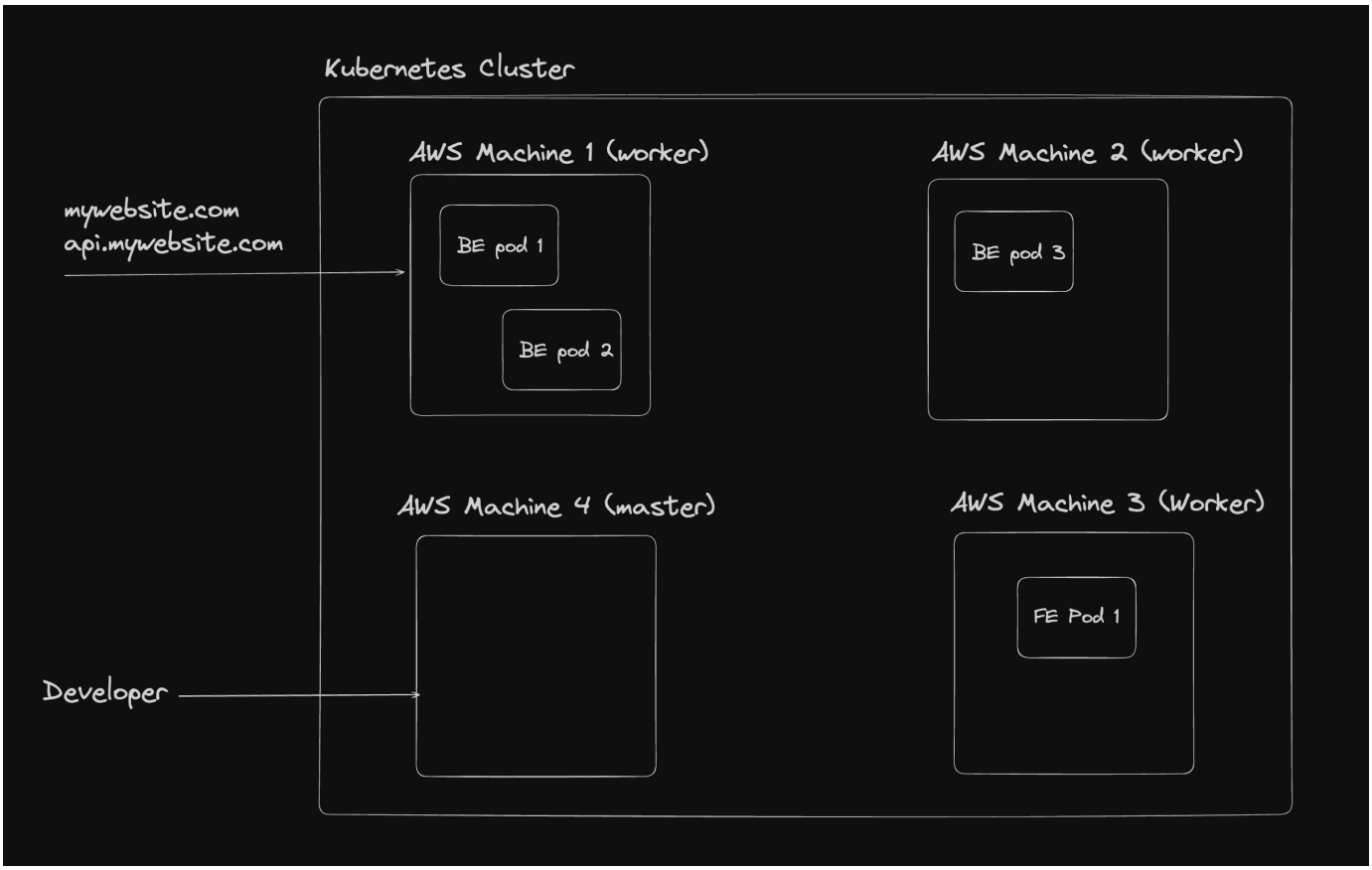
After kubernetes

Your frontend, backend are all pods in your kubernetes cluster



Jargon

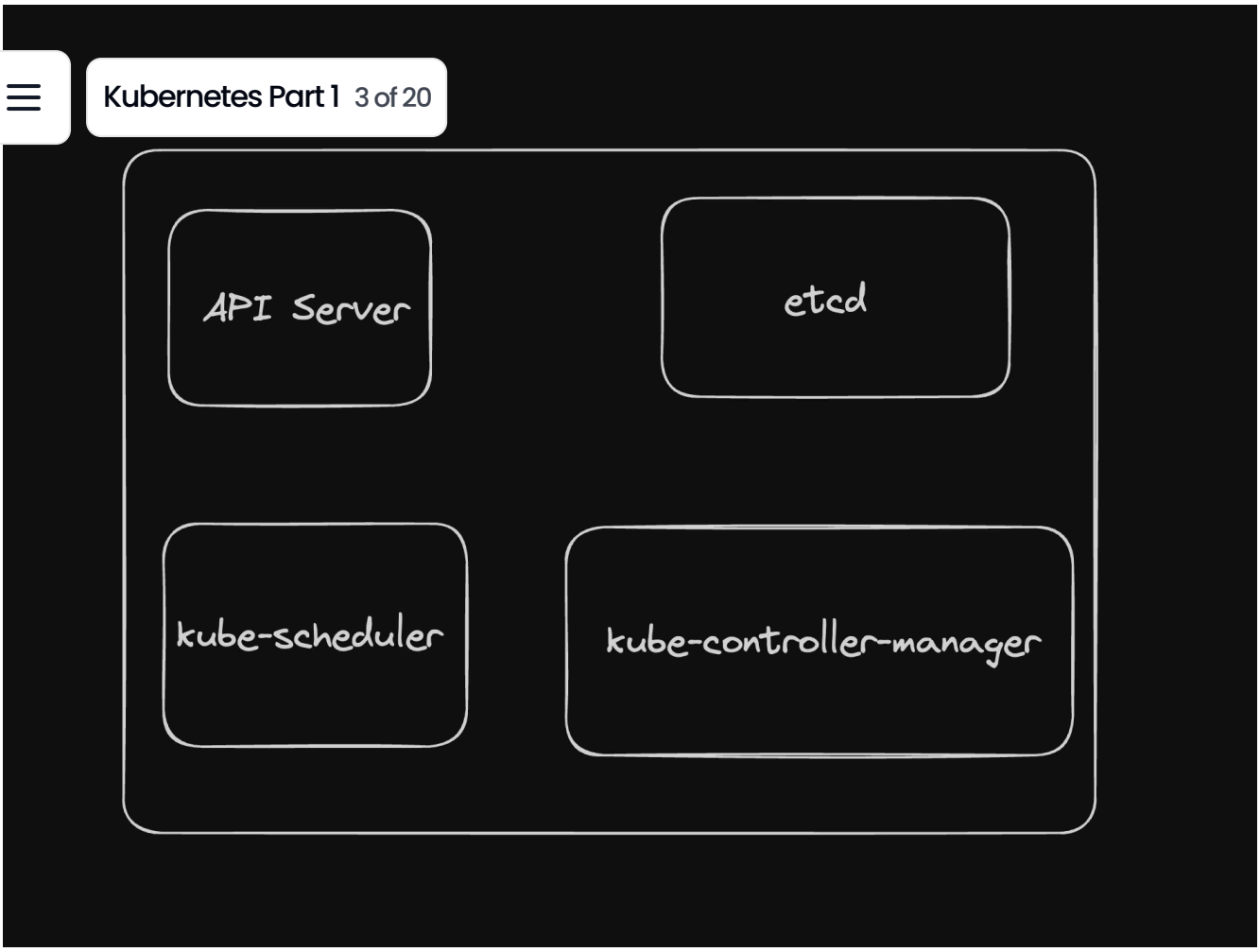
Ref - https://kubernetes.io/docs/concepts/overview/components/

Nodes

In kubernetes, you can create and connect various machines together, all of which are running kubernetes . Every machine here is known as a node

There are two types of nodes

▼ Master Node (Control pane) - The node that takes care of deploying the c e developer to understand what to deploy



▼ API Server

1. **Handling RESTful API Requests**: The API server processes and responds to RESTful API requests from various clients, including the `kubectl` command-line tool, other Kubernetes components, and external applications. These requests involve creating, reading, updating, and deleting Kubernetes resources such as pods, services, and deployments
2. **Authentication and Authorization**: The API server authenticates and authorizes all API requests. It ensures that only authenticated and authorized users or components can perform actions on the cluster. This involves validating user credentials and checking access control policies.
3. **Metrics and Health Checks**: The API server exposes metrics and health check endpoints that can be used for monitoring and rmance of the control plane.

4. **Communication Hub**: The API server acts as the central for the Kubernetes control plane. Other components, such as the scheduler, controller manager, and kubelet, interact with the API server to retrieve or update the state of the cluster.

▼ etcd

Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data. Ref - https://etcd.io/docs/v3.5/quickstart/

▼ kube-scehduler

Control plane component that watches for newly created Pods with no assigned node, and selects a node for them to run on. Its responsible for pod placement and deciding which pod goes on which node.

▼ **kube-controller-manager**

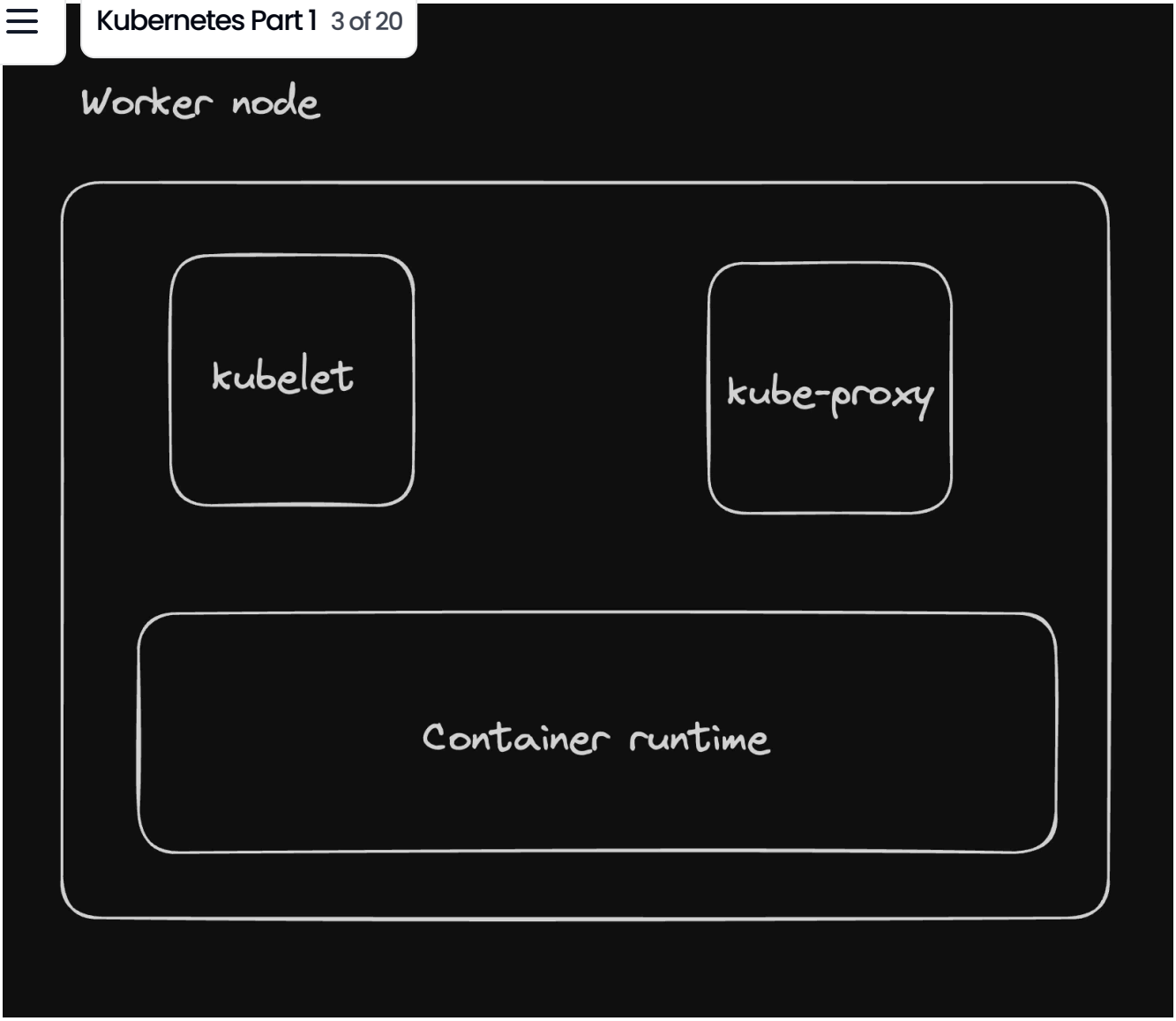
Ref - https://kubernetes.io/docs/concepts/architecture/controller/

The `kube-controller-manager` is a component of the Kubernetes control plane that runs a set of controllers. Each controller is responsible for managing a specific aspect of the cluster's state.

There are many different types of controllers. Some examples of them are:

- Node controller: Responsible for noticing and responding when nodes go down.
- Deployment controller: Watches for newly created or updated deployments and manages the creation and updating of ReplicaSets based on the deployment specifications. It ensures that the desired state of the deployment is maintained by creating or scaling ReplicaSets as needed.
- **ReplicaSet Controller**: Watches for newly created or updated ReplicaSets and ensures that the desired number of pod replicas are running at any given time. It creates or deletes pods as necessary to maintain the specified number of replicas in the

▼ Worker Nodes - The nodes that actually run your Backend/frontend



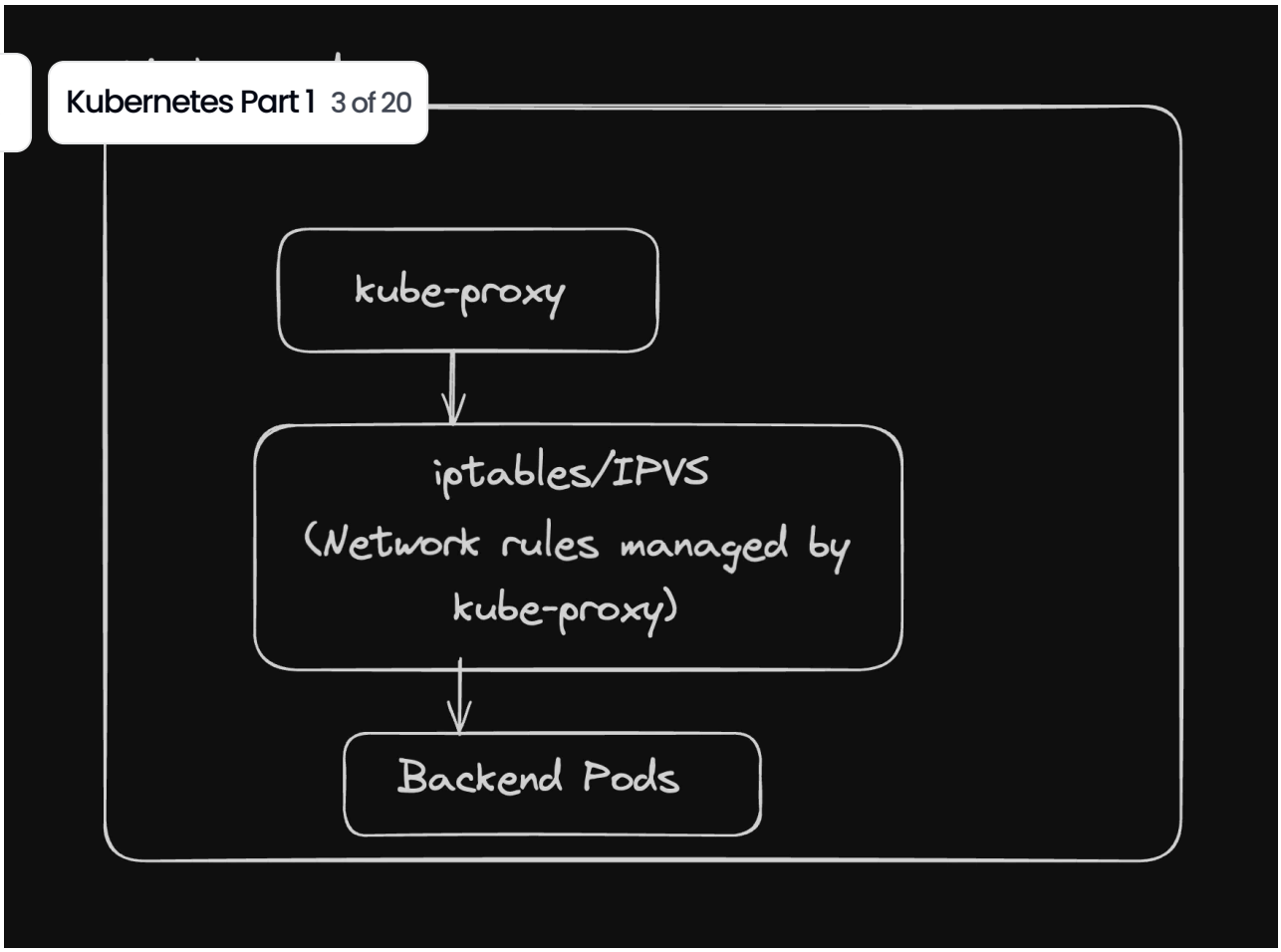
▼ **kubelet -** An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod.

How the kubelet Control Loop Works

1. **Watch for PodSpecs**: The kubelet watches the API server for PodSpecs that are scheduled to run on its node. This includes both new pods that need to be started and existing pods that may need to be updated or terminated.
2. **Reconcile Desired State**: The kubelet compares the current state of the node (which pods are running, their statuses, etc.) with the desired state as defined by the PodSpecs.

3. **Take Action**: Based on this comparison, the kubelet takes actions to state with the desired state:
    - **Start Pods**: If there are new PodSpecs, the kubelet will pull the necessary container images, create the containers, and start the pods.
    - **Monitor Health**: The kubelet performs health checks (liveness and readiness probes) on running containers. If a container fails a health check, the kubelet may restart it according to the pod's restart policy.
    - **Update Pods**: If there are changes to the PodSpecs (e.g., configuration changes), the kubelet will update the running pods accordingly.
    - **Stop Pods**: If a pod is no longer needed or should be terminated, the kubelet will stop and remove the containers.
4. **Report Status**: The kubelet periodically reports the status of the pods and the node back to the API server. This includes resource usage (CPU, memory, etc.) and the status of each container.

▼ **kube-proxy** - The `kube-proxy` is a network proxy that runs on each node in a Kubernetes cluster. It is responsible for you being able to talk to a pod



▼ **Container runtime** - In a Kubernetes worker node, the container runtime is the software responsible for running containers.

It interacts with the kubelet, which is the agent running on each node, to manage the lifecycle of containers as specified by Kubernetes pod specifications. The container runtime ensures that the necessary containers are started, stopped, and managed correctly on the worker node.

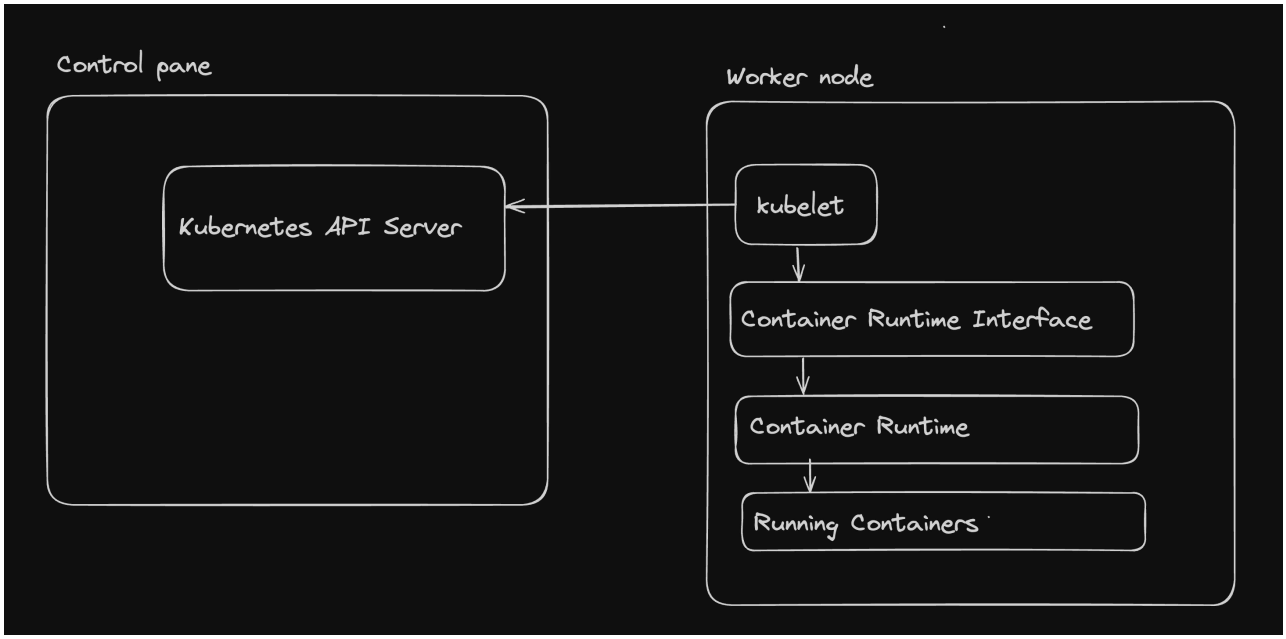
Common Container Runtimes for Kubernetes

1. containerd
2. CRI-O
3. Docker

Kubernetes Container Runtime Interface (CRI)

RI) is a plugin interface that allows
ainer runtimes without needing to

know the specifics of each runtime. This abstraction layer enables u multiple container runtimes, providing flexibility and choice to users.



Cluster

A bunch of worker nodes + master nodes make up your kubernetes cluster . You can always add more / remove nodes from a cluster.

Images

A **Docker image** is a lightweight, standalone, and executable software package that includes everything needed to run a piece of software, including the code, runtime, libraries, environment variables, and configuration files. Images are built from a set of instructions defined in a file called a Dockerfile.

Eg - https://hub.docker.com/_/mongo

Containers

A example if you run

docker run -p 5432:5432 -e POSTGRES_PASSWORD=mysecretpassword -d pos

Pods

A pod is the smallest and simplest unit in the Kubernetes object model that you can create or deploy