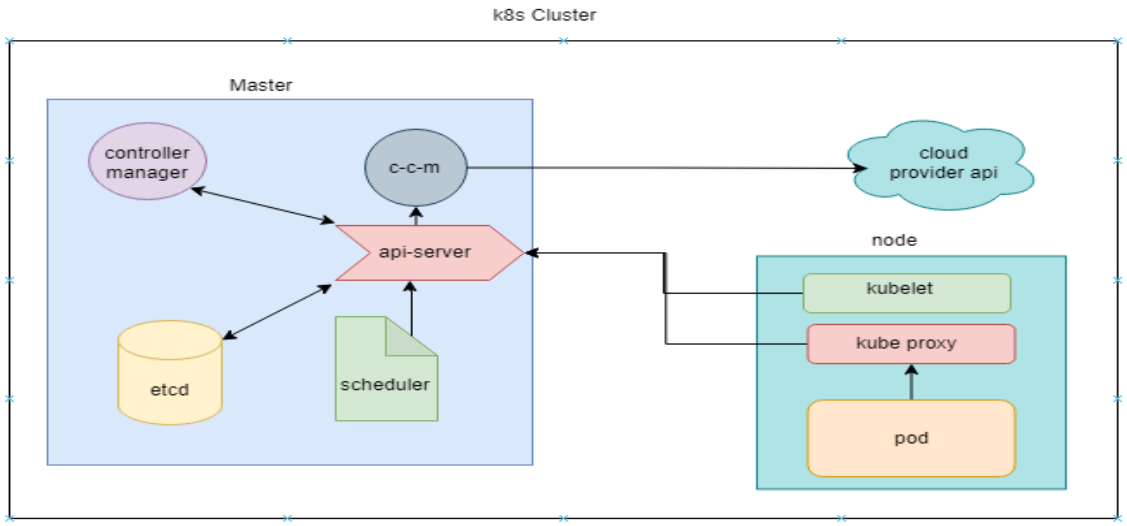
K8s Architcture

Following Kubernetes architecture diagram shows all the components of the Kubernetes cluster and how external systems connect to the Kubernetes cluster.



Overview

- Control Plane: Manages the Kubernetes cluster, handling the orchestration of tasks such as scheduling, maintaining desired state, scaling, and rolling out updates.
- Worker Nodes: Run the applications and handle the workload. Each node contains the necessary services to run the pods.

Control Plane Components

1. **KUBE-APISERVER:**

   The kube-apiserver is a crucial component of the Kubernetes control plane, serving as the central hub for communication within the cluster. Its primary functions and characteristics are detailed below:

Centralized API Management:

- The kube-apiserver acts as the primary access point for all Kubernetes API interactions. It processes RESTful API requests, enabling clients to create, read, update, and delete Kubernetes resources.

Authentication and Authorization:

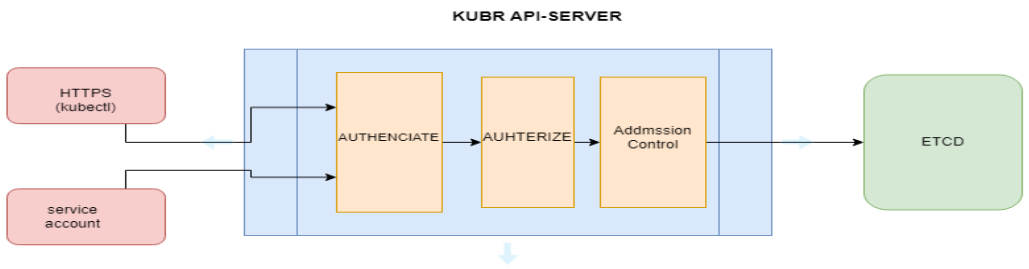
- Authentication: The API server verifies the identity of users and services making requests. It supports multiple authentication methods, including client certificates, bearer tokens, and authentication plugins.
- Authorization: Once authenticated, the API server checks if the request is authorized based on the user's permissions. It supports various authorization modes like Role-Based Access Control (RBAC), Attribute-Based Access Control (ABAC), and Webhook authorization, ensuring that only authorized actions are allowed.

Admission Control:

- Admission controllers are plugins that intercept requests to the API server after authentication and authorization but before the objects are persisted to etcd. They enforce policies and rules on the cluster, such as resource quotas, security policies, and default values.
- Examples of admission controllers include NamespaceLifecycle (ensures that objects are created only in existing namespaces), LimitRanger (enforces resource usage limits), and ResourceQuota (ensures that resource quotas are not exceeded).

Persistence and State Management:

- The API server interacts with etcd, the distributed key-value store used by Kubernetes to maintain the cluster's state. It ensures that the desired state of all Kubernetes objects (like Pods, Services, and ConfigMaps) is stored and retrieved consistently.
- The API server keeps etcd up to date with the current state of the cluster and retrieves the necessary data to handle client requests, maintaining the overall state of the cluster.



2. ETCD:

ETCD is a distributed, consistent key-value store that Kubernetes uses to store all of its cluster data. It serves as the primary data store for the Kubernetes control plane, providing reliable persistence and coordination for the entire system. The main function of etcd are:

Consistent Data Store:

- ETCD ensures strong consistency of data across all nodes in the cluster using the Raft consensus algorithm. This guarantees that every change to the data is properly recorded and agreed upon by a majority of nodes, preventing data corruption or loss.
- It is designed to store configuration data, metadata, and the current state of the cluster reliably.

High Availability and Fault Tolerance:

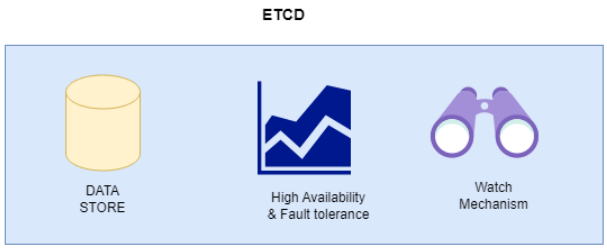
- ETCD is designed to be highly available and resilient to failures. It achieves this through a distributed architecture where data is replicated across multiple nodes.
- In case of node failures, etcd can recover and continue to provide consistent data access without significant downtime.

Leader Election:

- In a multi-node etcd cluster, one node is elected as the leader to handle all write operations. This leader election process ensures that there is a single source of truth for updates, reducing the chances of conflicting changes.
- The other nodes, known as followers, replicate the leader's data and handle read requests. If the leader fails, a new leader is automatically elected from the remaining nodes.

Watch Mechanism:

- ETCD provides a watch mechanism that allows clients to subscribe to changes in specific keys or directories. This is particularly useful in Kubernetes for components that need to react to changes in the cluster's state.
- When a change occurs, etcd sends notifications to all subscribed clients, enabling real-time updates and dynamic adjustments within the cluster.



3. **Kube-scheduler**:

The **kube-scheduler** is a critical component of the Kubernetes control plane responsible for assigning newly created Pods to nodes within the cluster. Its primary functions are Pod Queue Management, Node Selection, Scoring and Ranking Nodes & Binding Pods to Nodes

PreEnqueue:

- These plugins are called prior to adding Pods to the internal active queue, where Pods are marked as ready for scheduling.
- Only when all PreEnqueue plugins return Success, the Pod is allowed to enter the active queue.
- Otherwise, it's placed in the internal unschedulable Pods list, and doesn't get an Unschedulable condition.

Scoring and Binding Cycle Components:

1. Scheduling Cycle:

    - QueueSort: Determines the order in which unscheduled pods are processed.
    - Filter: Removes nodes that do not meet the pod's requirements.
    - PostFilter: Handles situations where no nodes meet the requirements by performing additional actions or providing feedback.

2. Scoring Cycle:

    - Score: Each remaining node is scored based on predefined policies. Scores are assigned to evaluate the suitability of each node for the pod.
    - NormalizeScore: Normalizes the scores to ensure consistency across different plugins.
    - ScorePlugins: Custom plugins can provide additional scoring criteria to influence the decision.

3. Reserve:

    - Temporarily reserves the resources on the chosen node to prevent other pods from being scheduled there until the current pod is fully scheduled.

4. PreBind:

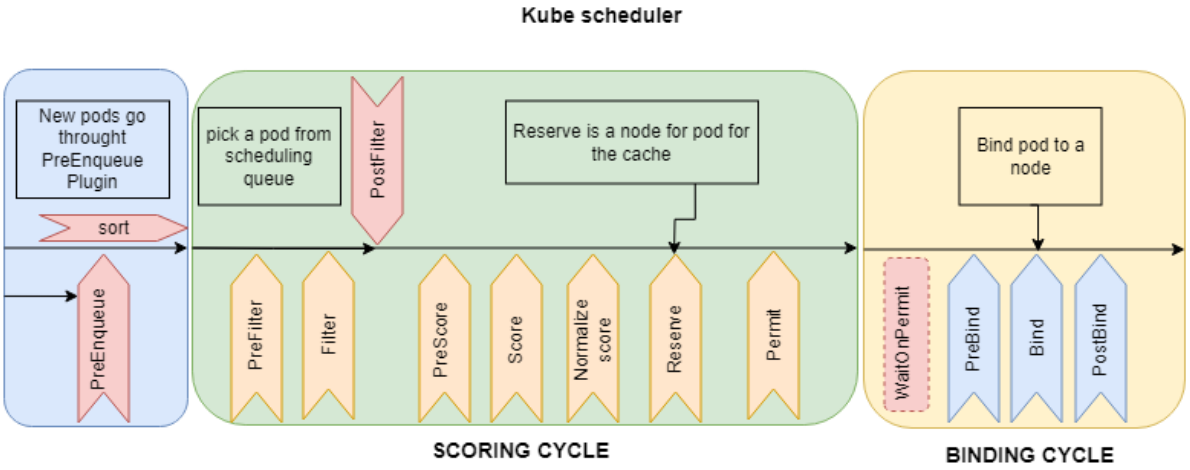
    - Executes any necessary operations before the binding step, such as reserving additional resources or performing pre-scheduling checks.

5. Bind:

    - The actual binding step where the pod is assigned to the chosen node. This updates the pod's specification in the API server to reflect the chosen node.

6. PostBind:

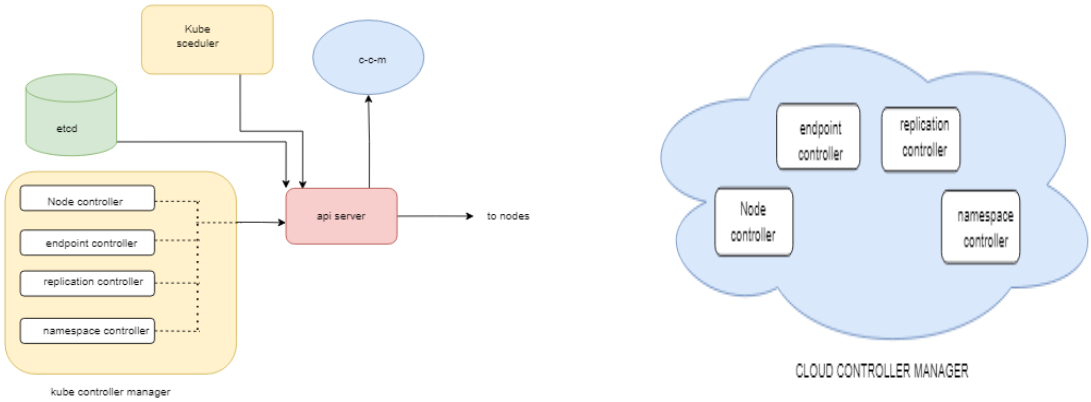
    - Any actions that need to be performed after the binding step, such as notifying other components or triggering subsequent tasks.



4. kube-controller-manager:

It is a crucial component of the Kubernetes control plane. It runs controllers, which are background processes that handle routine tasks in the cluster. These tasks include managing node operations, ensuring the desired state of objects in the system, and maintaining the lifecycle of various resources. Each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process. Includes various controllers such as

- **Node Controller**: Monitors the status of nodes and performs actions such as marking nodes as unhealthy or removing them from the cluster if they become unresponsive.
- **Replication Controller**: Ensures the specified number of pod replicas are running at any given time. If a pod fails or is deleted, the replication controller creates a new pod to replace it.
- **Endpoints Controller**: Populates the Endpoints object, which is used in services to keep track of which pods are providing the service.
- **Service Account & Token Controllers**: Manages the creation of default accounts and tokens for new namespaces.
- **Resource Quota Controller**: Monitors resource usage and ensures that quotas set for namespaces are not exceeded.
- **Namespace Controller**: Manages the lifecycle of namespaces, ensuring that all resources within a namespace are properly cleaned up when the namespace is deleted.
- **Job Controller**: Manages the creation and lifecycle of Job objects, ensuring that specified tasks are completed successfully.



5. Cloud-controller-manager (C-C-M):

The Cloud Controller Manager (c-c-m) in the Kubernetes architecture plays a crucial role in managing interactions between your Kubernetes cluster and the cloud provider. The Cloud Controller Manager is responsible for handling cloud-specific control loops. It enables Kubernetes to be cloud-agnostic by abstracting the details of the underlying cloud provider. Its Functions is to controller these controllers:

- **Node Controller**: Manages nodes in the cloud. It detects if a node is deleted in the cloud and removes the corresponding node from the Kubernetes cluster.
- **Route Controller**: Configures routes in the cloud to ensure network traffic for Kubernetes pods can flow correctly.
- **Service Controller**: Manages cloud load balancers. It ensures services of type LoadBalancer are correctly configured with cloud provider load balancers.
- **Volume Controller**: Manages persistent volumes. It interacts with the cloud provider to create, attach, and mount persistent storage volumes.

The c-c-m communicates with the Kubernetes API Server to receive updates on cluster state changes and to update the API Server about the status of cloud resources and it also interacts with the Cloud Provider API to perform its cloud-specific operations such as creating and managing load balancers, instances, and storage. Its key features are:

- **Modularity**: Allows Kubernetes to support multiple cloud providers by implementing cloud-specific logic.
- **Scalability**: Facilitates scaling of Kubernetes resources based on cloud infrastructure capabilities.
- **Resiliency**: Ensures Kubernetes resources remain available and properly configured even if underlying cloud infrastructure changes.