Documentation

Team Members and Contributions:

- Soumya Ranjan Mishra: /add_node - register_node (in node_manager.py)
- Sohum Salunke: node_sim.py
- Suhas Venkata: /heartbeat - heartbeat (in node_manager.py)
- Shuklav Reddy: /nodes - list_nodes, check_health (in node_manager.py), Documentation
- Hitha Sree: /launch_pod - assign_pod (in node_manager.py)

Overview

This documentation outlines the core components and working of a Kubernetes-based cluster architecture. Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. The diagram and the subsequent description present how a user's request flows through the Kubernetes system, highlighting the role of various internal modules.

System Architecture Diagram

Below is the architecture diagram illustrating the Kubernetes system workflow:

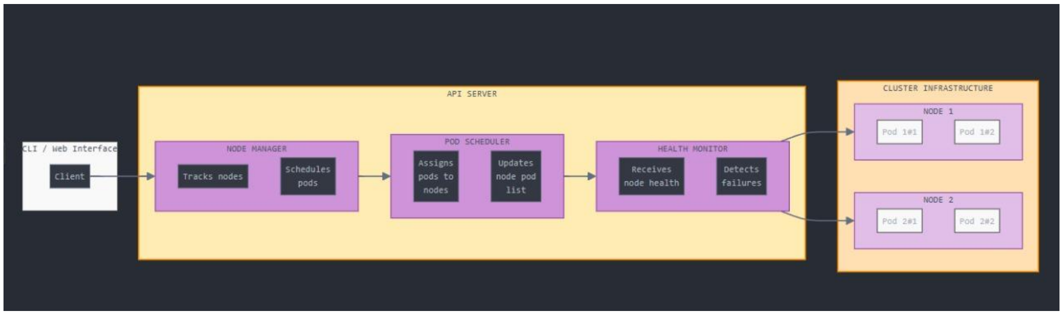

Figure: Kubernetes Cluster Architecture

Components and Workflow

1. Client Interface

The interaction begins through a CLI or Web Interface. A Client (typically `kubectl`) sends requests to the Kubernetes API Server to manage deployments and workloads.

2. API Server

The core of the control plane, the API server handles communication between users and the Kubernetes cluster.

a. Node Manager

- Tracks Nodes: Monitors the availability and state of nodes in the cluster.

- Schedules Pods: Determines which pods should run on which nodes based on availability and resource usage.

b. Pod Scheduler

- Assigns Pods to Nodes: Decides the best node for scheduling a pod using various factors like resource requirements, policies, affinity rules, etc.

- Updates Node-Pod List: Maintains updated information about pod assignments for nodes.

c. Health Monitor

- Receives Node Health: Regularly checks the status of each node in the cluster.

- Detects Failures: Triggers corrective actions when nodes are unresponsive or fail.

3. Cluster Infrastructure

The physical or virtual machines (nodes) that host the actual application pods.

a. Node 1

Hosts pods: Pod 1#1, Pod 1#2

b. Node 2

Hosts pods: Pod 2#1, Pod 2#2

Each pod runs one or more containers with the application workload. These are scheduled, monitored, and managed by the control plane.

API Endpoint to Function Mapping

- /add_node → register_node (node_manager.py)
- /heartbeat → heartbeat (node_manager.py)
- /nodes → list_nodes, check_health (node_manager.py)
- /launch_pod → assign_pod (node_manager.py)

Conclusion

This architecture provides a robust, scalable, and self-healing environment for deploying containerized applications. The system ensures workload distribution, fault tolerance, and health monitoring — all coordinated through Kubernetes' powerful API server.