Cloud Cluster Simulation Framework Documentation

(Python Flask & Docker)

M VENKAT CHARAN

System Overview

This system simulates a simplified cloud cluster environment using Docker containers as nodes. It provides the following core functionalities:

- Node Management: Add or remove Docker-based nodes with specified CPU core capacity.
- Pod Scheduling: Deploy pods with CPU resource requirements using a First-Fit scheduling algorithm.
- Node Health Monitoring: Detects unreachable nodes via periodic heartbeat checks.
- Partial Failure Recovery: When nodes are stopped, associated pods are removed.
- RESTful API Interface: Manage the cluster via API, with a basic web UI for status visualization.

System Testing

Test Case 1: Add a Node

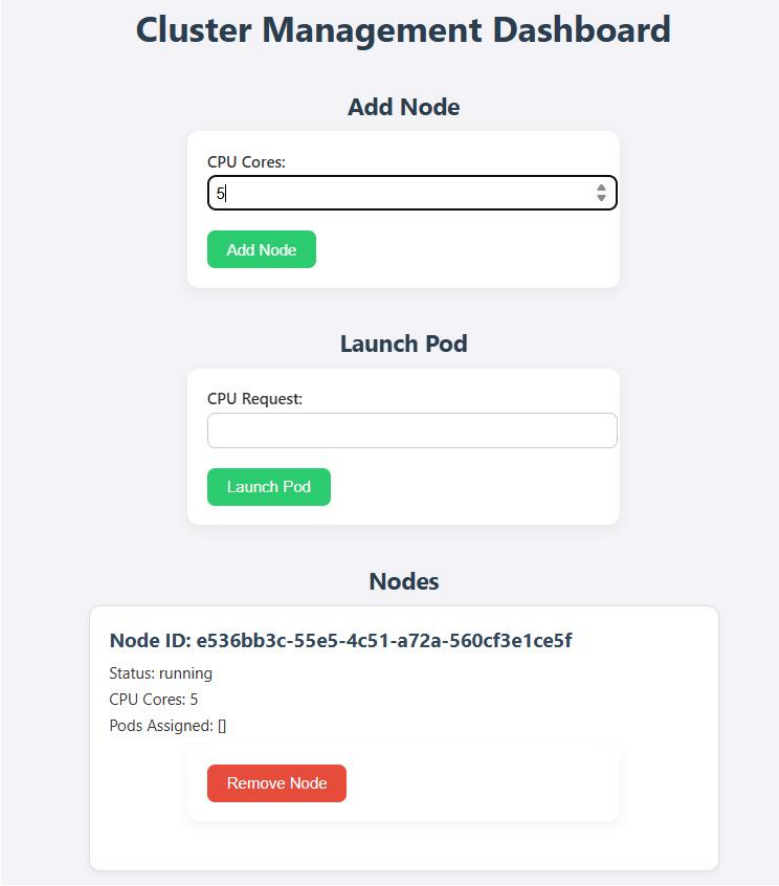
Description: Add a node to the cluster with specified CPU cores
Input: cpu_cores = 5 via web form
Expected Output: Node appears in node list with "running" status
Command:
curl -X POST http://127.0.0.1:5005/add_node -F "cpu_cores=2"

Verification:

Pretty-print ☐


```
{
  "message": "Node added",
  "node_id": "e536bb3c-55e5-4c51-a72a-560cf3e1ce5f"
}
```


-



Test Case 2: Launch a Pod

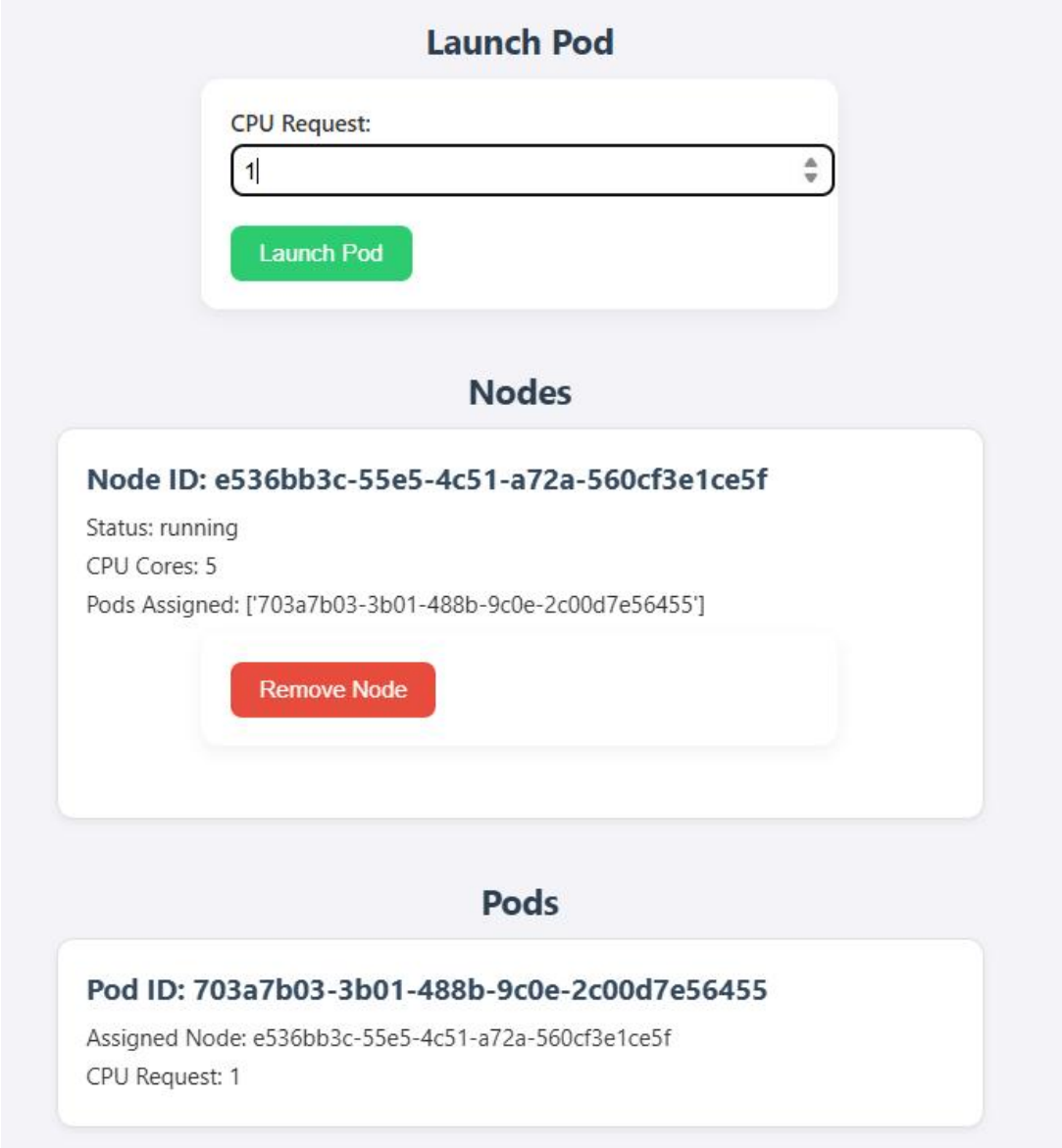
Description: Launch a pod with specific CPU requirement
Input: cpu_request = 1
Expected Output: Pod is assigned to a node with sufficient CPU
Command:

curl -X POST http://127.0.0.1:5005/launch_pod -F "cpu_request=1"

Verification:

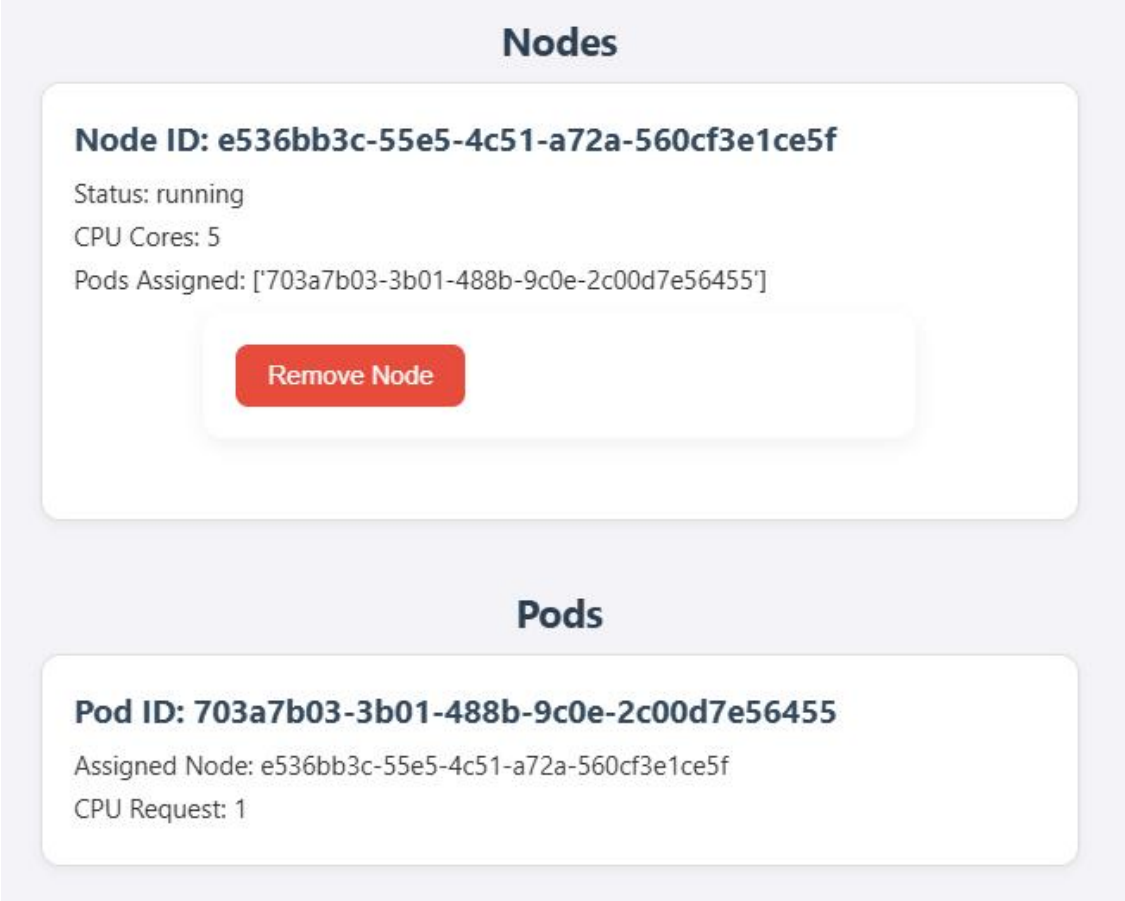
Pretty-print ☐


```
{
  "assigned_node": "e536bb3c-55e5-4c51-a72a-560cf3e1ce5f",
  "message": "Pod launched",
  "pod_id": "703a7b03-3b01-488b-9c0e-2c00d7e56455"
}
```

Test Case 3: View Node and Pod Lists in UI

Description: Use Web UI to inspect nodes and pods
Expected Output: Nodes and pods shown in tables with CPU usage and status
Access: http://127.0.0.1:5005/
Verification:

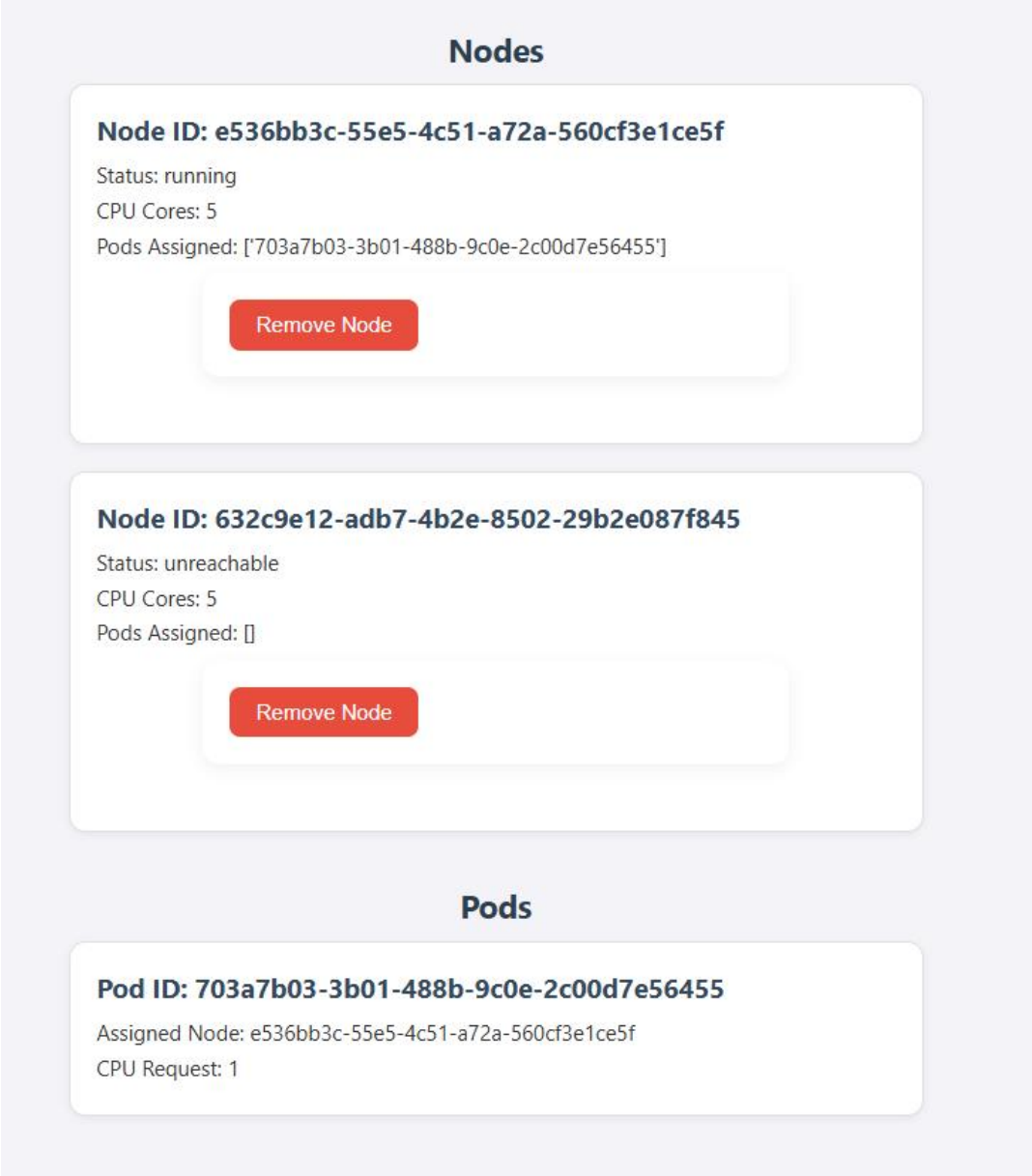


Test Case 4: Stop a Node and Verify Pod Removal

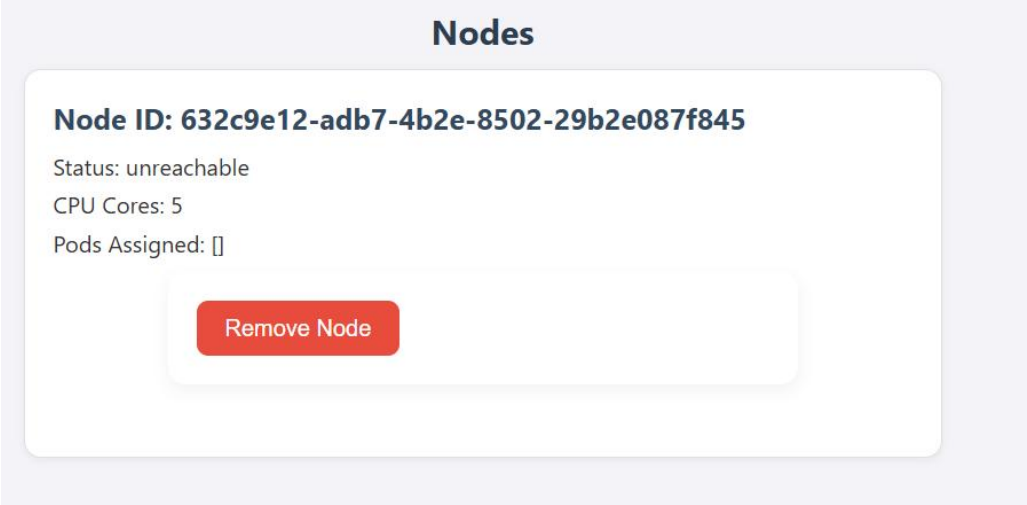
Description: Stop a node and verify its removal and associated pods
Steps:
1. Get node_id via UI or list_nodes API
2. Execute delete command
Command:

curl -X DELETE http://127.0.0.1:5005/stop_node/<node_id>
Expected Output:
- Node is removed from list
- Associated pods are removed from pod list
Verification:

Before Node Removal



After 10 seconds of Node Removal:-

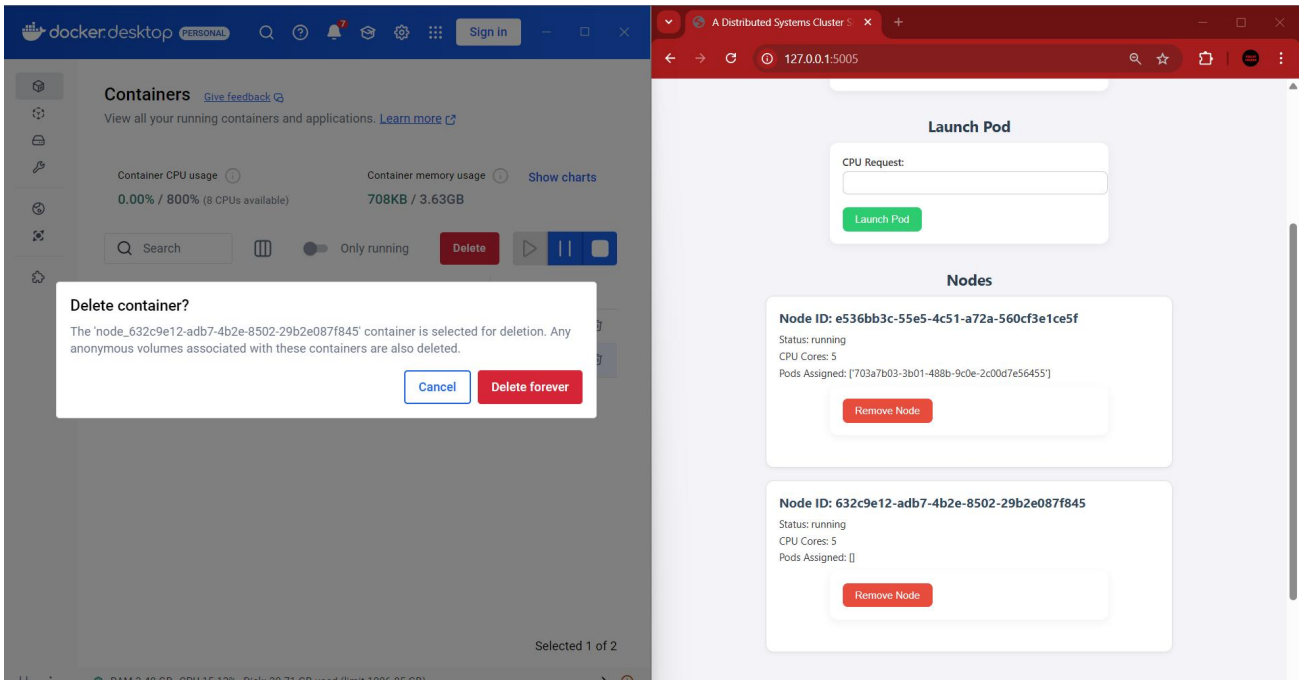


Test Case 5: Health Monitor - Simulate Node Failure

Description: Stop Docker container manually to simulate node failure
Steps:
1. Get container_id via `docker ps`
2. Run: docker stop <container_id>
3. Wait 10–20 seconds
Expected Output: Node status changes to "unreachable"
Verification:

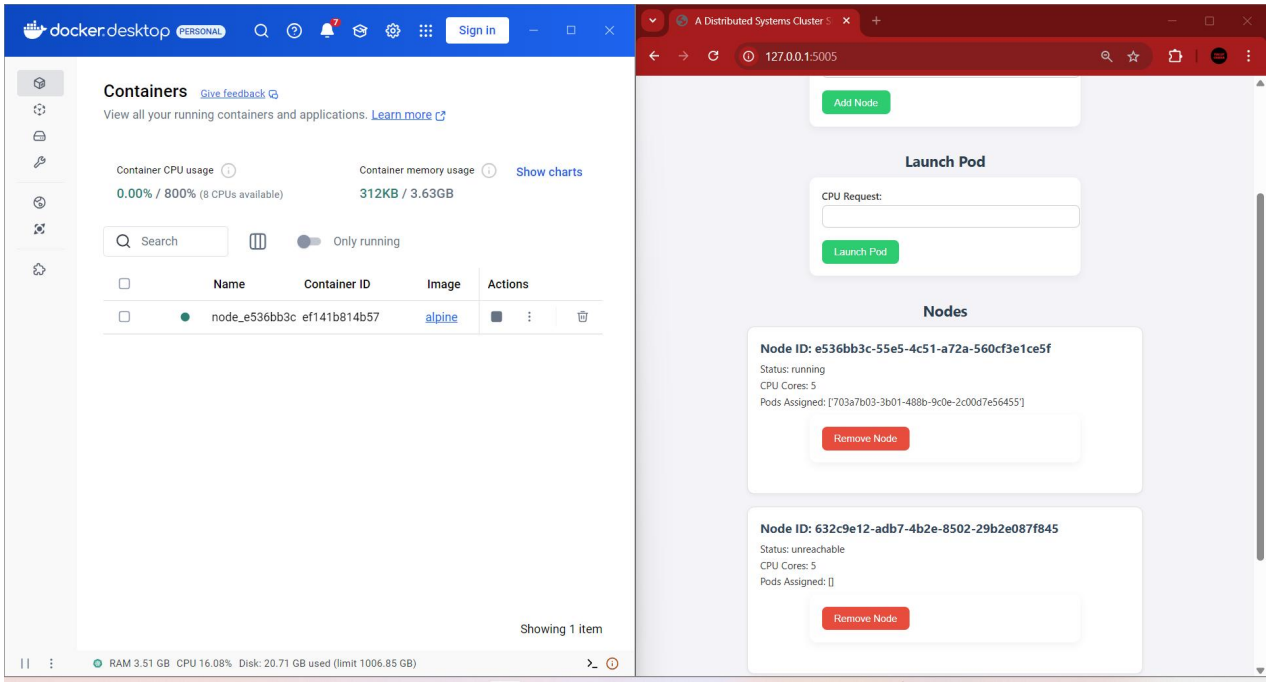
Before Deletion in Docker Desktop:-

Status:-Running



After Deletion in Docker Desktop:-

Status:-Unreachable



API Endpoints

/ [GET] - View cluster status and web UI
/add_node [POST] - Add a node with specified cpu_cores
/list_nodes [GET] - List all nodes in JSON format
/stop_node/<node_id> [DELETE] - Stop and remove a node and its pods
/launch_pod [POST] - Launch a pod with a CPU request
/pod_status/<pod_id> [GET] - Get the node assignment of a pod

Architecture Summary

- Nodes: Docker containers with defined CPU cores
- Pods: Workload requests tracked in memory (not real containers)
- API Server: Flask app managing the REST API
- Node Manager: Handles node lifecycle using Docker SDK
- Pod Scheduler: First-Fit logic in schedule_pod()
- Heartbeat Monitor: Background thread checking node health

Assumptions

- Docker must be installed and running
- Python dependencies: flask, docker
- Run using `python app.py`
- Pods are simulated (not actual containers)

Technology Stack

- Python: Main language
- Flask: Web framework
- Docker SDK (Python): Docker control
- Docker: Used to simulate nodes
- HTML/CSS: Basic web UI