Node manager Agent

WebRTC Virtual Classroom PLatform

Table of Contents

Overall design 2

node agent 2

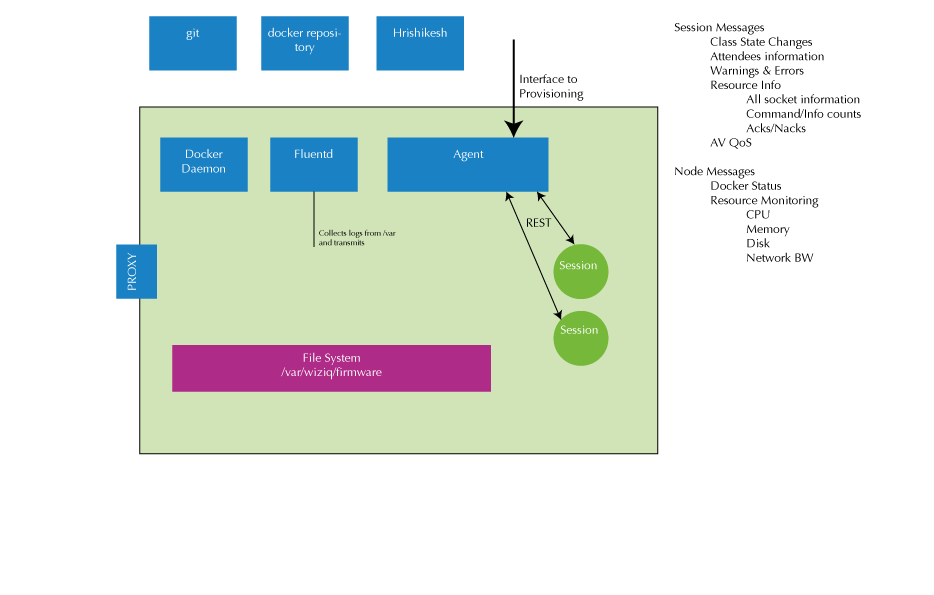
Provisioning Interface 2

Session Interface 3

# Overall design

Insert design diagram here.

# Node agent



The node agent server’s diagram above shows 2 (1 external and 1 internal) interfaces. The rest of the API section for the node agent server will conform to this view.

There exists 4 paths:

1. PROVISIONING -----> AGENT

2. AGENT ---> SESSION

3. SESSION -----> AGENT

4. AGENT ------> PROVISIONING

## provisioning side interface ( 1 )

The following operations need to be handled on the node:

|  |  |  |
| --- | --- | --- |
| **Category** | **API** | **Comment** |
| **SESSION** | Start a session | The full class configuration and run time information is provided in the request body |
| **SESSION** | Stop a session | Stop a running or about to start class. This would be called in the case of a manual intervention. Under normal circumstances, the session will manage it self and terminate itself. |
| **Session?** | Update a session | Update some information mid-way, say a log server going down and another one coming up (later) |
| **NODE** | Get status | Should return a big JSON object containing all it’s information (maybe including all sessions information as well) |
| **NODE** | Acquire | Acquire a node when told about a new node (via the admin command). A new node is brought up with no configuration effort. The provisioning is then told about the node (its IP Address). The provisioning server then invokes the ‘acquire’ API on the target node, supplying it with all the information it needs. |
| **NODE** | Modify | Update node specific information like change of log server etc. |

### Start A Session

## inner interface ( 2 )

|  |  |  |
| --- | --- | --- |
| **Category** | **API** | **Comment** |
| **SESSION** | Start a session | The full class configuration and run time information is provided in the request body |
| **SESSION** | Stop a session | Stop a running or about to start class. This would be called in the case of a manual intervention. Under normal circumstances, the session will manage it self and terminate itself. |
| **Session?** | Update a session | Update some information mid-way, say a log server going down and another one coming up (later) |

## SESSION TO AGENT ( 3 )

Docker status

System resource monitoring

\_\_\_\_\_\_\_meant for provisioning\_\_\_\_\_\_\_\_\_\_

Class state

Attendee snapshot

Resources info

AV QOS

## agent to provisioning ( 4 )

Some calls are same as 3 (where agent is just acting as a proxy).

## Monitoring & Keepalives

The following operations need to be supported:

|  |  |  |
| --- | --- | --- |
| **Category** | **API** | **Comment** |
| **PING** | Ping | Ping the Provisioning server periodically. Should the provisioning server Pong back? |
| **Feedback** | Node health | Periodic node health status (still debating whether this should be a push or a pull. If the node goes down these messages will not come. Which means, the provisioning will have to ‘guess’ that Oh! No message since X minutes, so possibly the node is down. This implies that the provisioning will have to maintain a timer. So if it \_has\_ to, then why not the provisioning server itself poll for health? That way the logic of determining if a node is down become very natural). This applies to the ping/pong scenario as well. |
| **Feedback** | Session status | Including all state transitions |
| **Feedback** | Async Notifications | Event driven asynchronous notifications, usually for fatal errors. |

# Appendix A – Data maintained

The Provisioning server ought to maintain the following data sets, in fast as well as persistent storage: NOTHING?