KEP- 4680: DRA Pod Device Health - Detailed Implementation Design

Last Updated: May 12, 2025

John-Paul Sassine

PR: https://github.com/kubernetes/kubernetes/pull/130606

KEP: https://github.com/kubernetes/enhancements/blob/master/keps/sig-node/4680-add-resource-health-to-pod-status/README.md

https://github.com/kubernetes/enhancements/pull/5302

1. Overview of DRA Health Monitoring within KEP-4680

This document outlines the implementation design for adding Dynamic Resource Allocation (DRA) device health status to the PodStatus, as proposed in KEP-4680. The primary goal is to allow Kubelet and external observers to determine if devices allocated to a Pod via DRA are healthy, aiding in troubleshooting and potentially enabling automated remediation (e.g., rescheduling).

This implementation **does not** automatically trigger rescheduling; that responsibility remains external, potentially using this status information. This design aims for integration with existing Kubelet infra.

2. High-Level Architectural Approach for DRA Health

1. **Optional gRPC Stream:** Defining a new, optional gRPC service (NodeHealth) within a dedicated, new API group (`dra-health/v1alpha1`) that DRA plugins can implement. This DRA plugin exposes a server-streaming RPC (WatchResources) so it can proactively send health updates for its managed devices to Kubelet.
2. **Health Information Cache:** Kubelet's DRA Manager maintains a persistent cache

(healthInfoCache) storing the latest known health status (Healthy, Unhealthy, Unknown) and LastUpdated timestamp for each device, keyed by driver name. The cache handles state reconciliation and timeouts for stale data, and persists across Kubelet restarts.

3. **Kubelet Integration:** Kubelet's DRA Manager (pkg/kubelet/cm/dra/manager.go) acts as the gRPC client. Upon plugin registration, it attempts to initiate the WatchResources stream. If successful, it consumes the health stream, updates the healthInfoCache, identifies which Pods are affected by reported health changes, and signals the Kubelet status manager via a dedicated update channel.
4. **Update Channel Aggregation:** The Container Manager (`pkg/kubelet/container_manager_linux.go`) merges update signals from the DRA Manager's update channel(*if the DRA feature is enabled*) and the Device Manager's existing update channel into a single channel (`Updates()`) monitored by the Kubelet status manager.
5. **PodStatus Update:** Kubelet's pod synchronization logic, upon receiving the update signal, or on any other Pod updates, calls the Container Manager's UpdateAllocatedResourcesStatus. This calls the DRA Manager's UpdateAllocatedResourcesStatus method, which reads the current health status for the Pod's allocated devices from healthInfoCache and populates the pod.Status.ContainerStatuses[].AllocatedResourcesStatus field with the correct health information.

Note: kubelet will ONLY currently use this health information to update the Pod Status. DRA plugin will be responsible to update resource slices to taint resources or mark them unhealthy in some other way.

3. Core Components and Logic for DRA Health

3.1. DRA Health Information Cache (pkg/kubelet/cm/dra/healthinfo.go)

- **Purpose:** To maintain the source of truth for device health within Kubelet, reconciling reported data and handling missing/stale information.
- **Data Structure:**
  - **healthInfoCache:** Holds HealthInfo *state.DevicesHealthMap and stateFile string.

- **state.DevicesHealthMap:** map[string]DriverHealthState (key is driver name).
- **state.DriverHealthState:** Contains Devices []DeviceHealth.
- **state.DeviceHealth:** Contains PoolName string, DeviceName string, Health state.DeviceHealthString, LastUpdated time.Time.
- **state.DeviceHealthString:** type DeviceHealthString string(Values: “Healthy”, “Unhealthy”, “Unknown”).

- **Key Features & Functions:**
  - **newHealthInfoCache(stateFile):** Initializes cache, loads from checkpoint file if exists. Handles file-not-found gracefully.
  - **updateHealthInfo(driverName, devices):**
    - Performs full-state reconciliation for the given driverName based on the list of devices reported by the plugin in a single WatchResourcesResponse.
    - Updates LastUpdated timestamps for all devices present in the devices list.
    - Marks devices not present in the update as "Unknown" if time.Since(LastUpdated) > healthTimeout. Existing health status is preserved if within the timeout.
    - Returns (changedDevices []state.DeviceHealth, changed bool, err error): changedDevices contains devices whose Health field was modified in this cycle (newly added, health changed, or timed out to Unknown); changed is true if *any* change occurred (including timestamp updates).
  - **getHealthInfo(driverName, poolName, deviceName):** Returns current health, considering healthTimeout.
  - **clearDriver(driverName):** Removes all data for a driver (used on deregistration/stream end).
  - **saveToCheckpoint() / loadFromCheckpoint():** Handles persistence via JSON marshalling. saveToCheckpoint automatically called by updateHealthInfo & clearDriver
  - **healthTimeout:** Constant (e.g., 30s) for marking stale devices "Unknown". If Kubelet doesn't receive an update containing a specific device within this duration, its health will be reported as "Unknown" in PodStatus by getHealthInfo.
- **Handling Timeouts and 'Unknown' State**
  - **Implicit Timeout Handling:** The transition of a device's health to "Unknown" due to a timeout is primarily handled by the getHealthInfo function. When UpdateAllocatedResourcesStatus requests the health of a device, getHealthInfo checks if time.Since(device.LastUpdated) > healthTimeout. If true, it returns "Unknown" regardless of the previously cached health state.
  - **Reconciliation during updateHealthInfo:** When updateHealthInfo processes a new list of devices from a plugin for a given driver:

        - For devices present in the new list, their LastUpdated timestamp is refreshed, and their health is updated.
        - For devices that were previously known for that driver but are not present in the new list, updateHealthInfo also checks their LastUpdated timestamp. If time.Since(existingDevice.LastUpdated) > healthTimeout, their cached Health is explicitly set to "Unknown", and they are included in changedDevices. This ensures the cache actively reflects timeout-induced "Unknown" states for devices no longer reported by the plugin.
    - **No Separate Expiration Worker:** A separate background worker to periodically scan the cache for expired entries and trigger Pod status updates is not required with this design.
        - The "Unknown" status due to timeout is determined on-demand by getHealthInfo whenever PodStatus is being computed.
        - Kubelet's existing status manager already triggers PodStatus updates for various reasons (Pod events, changes from other managers, periodic resyncs). When these updates occur, UpdateAllocatedResourcesStatus will call getHealthInfo, which will naturally reflect any timeouts.
        - If a plugin stops sending updates for a device, the healthInfoCache still retains the last known health and timestamp. getHealthInfo will report this health until healthTimeout is exceeded, after which it will report "Unknown". The next regular Pod status update will then reflect this change.
        - If a plugin stops sending updates for all its devices (e.g., plugin crashes and its stream ends), HandleWatchResourcesStream calls clearDriver, removing all entries. Subsequent calls to getHealthInfo for these devices will return "Unknown".

3.2. gRPC API for DRA Device Health

- **gRPC API (`staging/src/k8s.io/kubelet/pkg/apis/dra-health/v1alpha1/api.proto`):**
    - Defines NodeHealth service with rpc WatchResources(WatchResourcesRequest) returns (stream WatchResourcesResponse).
    - DeviceHealth: message includes pool_name, device_name, health, last_updated (unix timestamp), and potentially resource_name (optional, e.g., <driver>/<pool>/<device>).
    - Kept separate from core DRA API (Node service) to signify optionality.
        - If it is not implemented then plugin registration would continue and

if device health is ever queried(eg during a call to UpdateAllocatedResourcesStatus) the devices would return state “Unknown”

3.3. Kubelet DRA Plugin Client and Registration for Health Monitoring

- **Plugin Client Logic (`pkg/kubelet/cm/dra/plugin/plugin.go`):**
  - **Plugin struct:** Holds healthClient (drahealthv1alpha1.NodeHealthClient), healthStreamCtx (context.Context), healthStreamCancel (context.CancelFunc).
  - **getOrCreateGRPCConn():** Initializes healthClient after gRPC connection establishment.
  - **WatchResources(ctx):** Method called by registration handler. Initiates the gRPC stream via healthClient.WatchResources.
  - **SetHealthStream, HealthStreamCancel:** Manages the context/cancel specific to the health stream routine.

- **Registration (`pkg/kubelet/cm/dra/plugin/registration.go`):**
- **RegistrationHandler:** Holds streamHandler StreamHandler.
- **NewRegistrationHandler:** Accepts StreamHandler
- **RegisterPlugin:**
  - Attempts pluginInstance.WatchResources.
  - **If Unimplemented error:** Logs info(plugin doesnt support health) continues registration without health monitoring. PodStatus will not be populated for this driver’s resources.
  - **On Success:** store context/cancel using pluginInstance.SetHealthStream(), start background goroutine: go h.streamHandler.HandleWatchResourcesStream(streamCtx, stream, pluginName) (calls via interface).
  - **On Replacement:** Calls oldPlugin.HealthStreamCancel() to stop the previous stream handler.

Deregistration (`pkg/kubelet/cm/dra/plugin/registration.go`):

- DeRegisterPlugin: Calls p.HealthStreamCancel() to stop the stream handler goroutine and cleans up plugin resources (context, connection).

3.4. Kubelet DRA Manager and Container Manager Integration

- **DRA Manager (`pkg/kubelet/cm/dra/manager.go`):**
  - **ManagerImpl:** Implements plugin.StreamHandler. Holds healthInfoCache, healthInfoMutex, update chan resourceupdates.Update.
  - **HandleWatchResourcesStream(**ctx context.Context, stream drahealthv1alpha1.NodeHealth_WatchResourcesClient, pluginName string**):**
    - Goroutine runs per plugin. Receives health updates, calls healthInfoCache.updateHealthInfo. If devices changed state, finds affected Pod UIDs from claimInfoCache and sends UIDs to update channel (non-blocking). Logs errors/cancellation and exits.
    - Defer func added which cleans the cache whenever the function exits to properly clean the health cache.
  - **Updates() <-chan resourceupdates.Update:** Implemented method that returns the `m.update` channel.
  - **UpdateAllocatedResourcesStatus(pod, status):**
    - Reads healthInfoCache, formats v1.ResourceStatus (with Name = claim name, Resources = slice of v1.ResourceHealth for each device with ID and Health) and updates the input PodStatus.
  - **GetWatcherHandler()**: Instantiates `plugin.NewRegistrationHandler`, passing `m` (the ManagerImpl instance) as the `plugin.StreamHandler`
- **Container Manager (`pkg/kubelet/cm/container_manager_linux.go`):**
  - **Updates():** Merges the channel returned by m.draManager.Updates() with the deviceManager's update channel into a single output channel consumed by the main Kubelet sync loop.
  - **UpdateAllocatedResourcesStatus():** Calls device manager's update, then DRA manager's update (guarded by feature gate).

4. Testing Strategy

- **Unit Tests:** Focus on healthinfo.go (reconciliation, timeouts, persistence), registration.go (stream start/cancel logic, error handling), manager.go (HandleWatchResourcesStream's pod finding and channel sending logic).
- **[WIP] E2E Test (test/e2e_node/dra_test.go):**

Discussion:

- **Is the pod marked unhealthy when the devices are unhealthy, how does this work?**