Cluster Autoscaler/Karpenter API Alignment AEP | HACKED BY XABIT

Background

SIG Autoscaling and Karpenter are currently collaborating together in response to an upstream Kubernetes request which proposes that karpenter-core be donated to the Kubernetes project, under the SIG. Cluster Autoscaler (CAS) is currently a sub-project under the SIG and has some feature and API overlap with Karpenter with respect to how they both handle node lifecycle management on the cluster.

This proposal attempts to identify opportunities to standardize API surface areas across CAS and Karpenter as well as any future node lifecycle management projects that implement similar features. By maximizing opportunities for using a common API surface, we can reduce user confusion and churn as a user picks a node management solution that meets their needs. Users that choose to run multiple node management solutions on the same cluster get the added benefit of reasoning about fewer API concepts on pods and nodes.

This proposal also attempts to make progress in clarifying areas of Kubernetes that currently have more ambiguity with respect to areas pertaining to node lifecycle management, including a standard graceful termination flow for nodes

Summary

We identify the following opportunities for standardization across CAS and Karpenter:

1. **Shared Labels/Annotations/Taints APIGroup**: Introduce the node-lifecycle.kubernetes.io API Group to be a shared API group for node lifecycle concepts, including a standardized set of labels/annotations/taints, while allowing CAS-specific concepts to continue in cluster-autoscaler.kubernetes.io and Karpenter to live in karpenter.kubernetes.io
2. **Eviction/Termination Flow:** Introduce the node-lifecycle.kubernetes.io/disruption=candidate:PreferNoSchedule and node-lifecycle.kubernetes.io/disruption=disrupting:NoSchedule taints to drive node drain and termination through Taint-based Eviction
3. **Node Termination Candidacy**: Supersede the cluster-autoscaler.kubernetes.io/safe-to-evict and karpenter.sh/do-not-disrupt with the node-lifecycle.kubernetes.io/do-not-disrupt annotation
4. **Conceptual Documentation**: Settle on conceptual alignment between the overlapping pieces of the two projects, update upstream K8s docs and sub-project docs to publicize these shared concepts

Proposal

Below, we propose the areas of API alignment, which we expect to be adopted by both projects. This list may expand in the future as the two projects evolve and other areas of upstream alignment are discovered.

API Group

Behaviors and APIs driven around host resource startup and health on nodes are fairly well-defined in upstream Kubernetes. These APIs live inside the node.kubernetes.io namespace and are primarily driven by the kubelet and apiserver, monitoring details of the host like remaining cpu, remaining memory, health of filesystems, the liveness of processes, etc. on the host. Examples of these APIs include node.kubernetes.io/unreachable, node.kubernetes.io/not-ready, etc.

While these APIs capture and well-define what happens when host resources are starting or are not performing as expected, they do not well-define behaviors that are driven by external components which orchestrate nodes, including node termination candidacy and graceful node termination. In particular, when looking at CAS and Karpenter, APIs around node lifecycle management and termination candidacy are driven through very similar looking taints and annotations which are shown in the section below.

This similarity in features and APIs lead us to believe that there can be alignment here between Karpenter, CAS, and upstream Kubernetes in general with respect to defining common node

lifecycle behavior with respect to external components managing nodes. This AEP proposes the node-lifecycle.kubernetes.io API group to be this shared namespace for upstream Kubernetes labels, annotations, and taints that define this behavior.

Marking Nodes as Candidates for Termination (Taint)

CAS Equivalent: DeletionCandidateOfClusterAutoscaler=true:PreferNoSchedule
Karpenter Equivalent: None currently defined
Proposed Node Lifecycle Taint:
node-lifecycle.kubernetes.io/disruption=candidate:PreferNoSchedule

Marking Nodes as Actively Being Terminated (Taint)

CAS Equivalent: ToBeDeletedByClusterAutoscaler=true:NoSchedule
Karpenter Equivalent: karpenter.sh/disruption=disrupting:NoSchedule
Proposed Node Lifecycle Taint: node-lifecycle.kubernetes.io/disruption=disrupting:NoSchedule

Marking Pods and Nodes (Annotation)

CAS Equivalent: cluster-autoscaler.kubernetes.io/safe-to-evict
Karpenter Equivalent: karpenter.sh/do-not-disrupt
Proposed Node Lifecycle Taint: node-lifecycle.kubernetes.io/do-not-disrupt

With respect to other CAS and Karpenter APIs that handle cluster-wide configuration (such as the ProvisioningRequest CRD for CAS or the NodePool CRD for Karpenter), we plan to allow these APIs to continue to manage their own namespace, as an extension off of the kubernetes.io namespace.

For CAS, this can continue to be the cluster-autoscaler.kubernetes.io namespace. For Karpenter, we propose this namespace should be karpenter.kubernetes.io.

Eviction/Termination Flow

CAS and Karpenter take similar approaches with respect to how they handle the eviction and deprovisioning flow for nodes that are chosen to be removed from the cluster. This AEP proposes that CAS and Karpenter align in their eviction and termination flows by performing the following actions when considering terminating the node:

1. Taint the node with node-lifeycycle.kubernetes.io/disruption=candidate:PreferNoSchedule when considering a node a candidate for deletion. This is not a hard requirement, but both CAS and Karpenter have timeouts between the time that a node is considered a candidate for deletion and when the node is actually confirmed to be disrupted. A PreferNoSchedule taint ensures that we attempt to schedule workloads to other nodes on the cluster, if possible to increase the chance that we will be able to disrupt the node at the end of the timeout.

2. Taint the node with `node-lifeycycle.kubernetes.io/disruption=disrupting:NoSchedule` when a node is recognized that it should be deleted (this gives time for a new node to come up so that the old node can be safely terminated)
3. Once the old node can be safely terminated, initiate pod draining
    a. Because the disruption taint still exists on the node, this means that DaemonSet eviction would be enabled by default and could only be disabled by adding the matching tolerations to DaemonSets that you wish to opt-out of this eviction process
4. Once the termination mechanism has observed that all pods that can be evicted from the node are evicted (this means that it will ignore pods that tolerate this taint, since these pods would immediately reschedule, if evicted), it proceeds to terminate the instance using Cloud Provider APIs and remove the node

Node Termination Candidacy

CAS and Karpenter currently take different stances with respect to whether they choose to allow pods to block termination by default. CAS currently prevents scale-down of nodes for any nodes that meet a set of conditions specified in the FAQ section of the CAS documentation. Karpenter has some overlap with some of these conditions; however, it has chosen to take a more aggressive stance when it comes to deprovisioning, generally assuming that a node can be deprovisioned and disrupted unless a user explicitly prevents the node to be deprovisioned through PDBs or annotations.

This AEP proposes that CAS and Karpenter align on the handling of when the nodes can be terminated for specific API constructs, specifically to allow nodes to be disrupted in all cases, with some shared exceptions:

1. Nodes will not be disrupted when a PodDisruptionBudget would block the eviction of any pods on the node
2. Nodes will not be disrupted when the pods wouldn't be able to schedule on any existing node or on any simulated launched node
3. Nodes will not be disrupted when any pod running on the node has the `node-lifecycle.kubernetes.io/do-not-disrupt: "true"` annotation
4. Nodes will not be disrupted when it has the `node-lifecycle.kubernetes.io/do-not-disrupt: "true"` annotation

This is not a comprehensive list. Projects are free to define additional tenets outside of this list that add additional layers to allow/deny disruption; however, the recommendation is to keep this list as small as possible to index to allow nodes to be treated as ephemerally as possible.

Conceptual Documentation

As we are agreeing on these areas of technical alignment, we should also create a space in upstream Kubernetes documentation where we can house this shared technical agreement with respect to projects that pertain to node lifecycle management. These concepts include but are not limited to:

1. How node lifecycle management projects evaluate pods and nodes for scale-ups
2. How node lifecycle management projects evaluate pods and nodes for scale-downs

3. How node lifecycle management projects orchestrate the termination flow for graceful termination of nodes

To ensure consistency with these areas of conceptual alignment that we are proposing in upstream documentation, we should agree on a set of terms that will be used in documentation when considering these concepts moving forward:

1. **Node Provisioning**: The process through which a node is created based on some node lifecycle management-driven heuristic (pod resource requests in the case of CAS and Karpenter)
2. **Node Disruption**: The process through which a node is terminated. In general, this describes the entire process beginning from evaluating the node for disruption candidacy to preventing the node from having pods scheduled to it, to draining the node, to deletion
3. **Node Consolidation:** The process through which a single node or set of nodes is considered for removal or replacement based on node resource underutilization.