KEP-5359 : Pod-Level Swap Control

Feature Gate: `PodSwapAwareness`

Author: Ajay Sundar Karuppasamy
Last Update: May 20, 2025

Summary

This KEP proposes introducing a pod-level API field to control swap memory usage for individual pods. The initial focus is to allow pods to explicitly disable swap, even if swap is enabled at the node level (via the `LimitedSwap` setting introduced in KEP-2400). This enhancement aims to complement the existing node-level swap support by providing a per-workload opt-out. This per-workload swap behavior is crucial in multi-tenant environments to allow latency-sensitive applications to run alongside pods that can utilize swap.

Motivation

Kubernetes has introduced node-level swap support (KEP-2400), currently Beta3 in 1.33, which allows nodes to be configured with swap memory. The `LimitedSwap` mode restricts swap usage primarily to Burstable QoS pods. While KEP-2400 restricts Guaranteed and high-priority pods from swap for performance guarantees, users cannot specifically exclude other critical pods from potential swap usage.

While node-level swap can improve node stability and resource utilization in certain scenarios, it presents a challenge for latency sensitive applications. For these applications, performance degradation with any swap activity may be undesirable. Currently, if a node has swap enabled, application owners do not have a standard Kubernetes API mechanism to express their workload's intolerance to swap.

This KEP addresses this gap by providing a pod-level control to disable swap for all its containers irrespective of underlying node swap behavior.

Goals

- Introduce a new field `swapPolicy` in PodSpec to allow users to define swap behavior for a pod.
- Initially, support a mode to explicitly `Disable` swap for pods.

- If a pod requests swap to be disabled, Kubelet will enforce this at node by appropriately setting the cgroupv2 controls for the pod.
- Maintain backward compatibility: existing pods that run on swap-enabled nodes should behave as they do by default.
- Provide a mechanism to alleviate concerns regarding “all-or-nothing” nature of node-level swap enablement, potentially unblocking KEP-2400’s path to GA.

Non-Goals

- To introduce fine-grained swap controls at the pod or container level (e.g., setting a specific swap limit other than zero) in this initial KEP.
- To change the fundamental behavior of `LimitedSwap` for Guaranteed, Burstable, or BestEffort pods when the pod-level swap setting is not specified.
- To define how swap contributes to pod eviction or schedulable resource limits beyond disabling its usage.
- To enhance pod scheduling to allow users to specify swap-capable node preferences. This will be achievable with features such as node-capabilities.
- To support cgroupv1 for this feature, as Kubernetes swap support (KEP 2400) is focused on cgroup v2.

Proposal

API Changes

We propose adding a new field `swapPolicy` to `pod.spec`.

```
// PodSpec is a description of a pod
type PodSpec struct {

// .. existing fields..

// SwapPolicy defines the desired swap memory policy for this pod. This field
// is immutable after the pod has been created.
//
// If unspecified, the pod's swap behavior is determined by the node's swap
// configuration (KEP-2400) and the pod's QoS class, equivalent to “NoPreference”
// If mode is set to "Disabled", swap will be disabled for this pod, irrespective of
the node's swap configuration.
//
// +featuregate="PodSwapAwareness"
// +optional
    SwapPolicy *PodSwapPolicy `json:"swapPolicy,omitempty"`
}
```

```
// PodSwapPolicy defines the swap memory policy for a pod.
type PodSwapPolicy struct {
  // Mode defines the desired swap behavior mode for the pod.
  // This field is immutable after the pod has been created.
  // Supported modes:
  // - “Disabled”: Swap will be disabled for this pod. Kubelet will configure pod’s
cgroup to prevent any swap usage
  //- “NoPreference”: The pod adheres to the node’s default swap behavior. This is
the default if swapPolicy is unset.
  //
  // +optional
  Mode SwapPolicyMode `json:”mode,omitempty”`
}

// SwapPolicyMode defines the possible values for mode in pod.swapPolicy.
type SwapPolicyMode string

const (
  // SwapPolicyModeDisabled explicitly disables swap for the pod.
  SwapPolicyModeDisabled SwapPolicyMode = "Disabled"
  // REMOVED: Based on #api-review discussion NoPreference is no longer an explicit
value.
  // This behavior is now achieved by leaving the swapPolicy field unset.
- // SwapPolicyModeNoPreference states the pod should follow node-level swap
configuration.
- SwapPolicyModeNoPreference SwapPolicyMode = "NoPreference"
)
```

1. The `swapPolicy` field will be optional.
2. For Alpha, the only accepted modes will be `NoPreference` or `Disabled`.
3. If the `swapPolicy` field is not set or empty `swapPolicy.mode`, it will default to `NoPreference` swap mode, respecting the node's configured swap behavior (like `LimitedSwap`) and ensures backward compatibility.
4. The field `pod.spec.swapPolicy` will be immutable after pod creation.

Example

A pod that wants to disable swap would be configured like this:

```
apiVersion: v1
kind: Pod
metadata:
  name: no-swap-pod
spec:
```

```
  swapPolicy:
    mode: Disabled
  containers:
  - name: my-app
  image: test-image
```

A pod that explicitly wants to follow node's default behavior would look like this:

```
apiVersion: v1
kind: Pod
metadata:
  name: swap-tolerant-pod
spec:
  swapPolicy:
    mode: NoPreference
  containers:
  - name: my-app
  image: test-image
```

Validation Scenarios:

The following pod types are currently excluded from swap usage with `LimitedSwap` (KEP-2400):

- Guaranteed and BestEffort QoS class pods.
- System-priority pods (system-node-critical, system-cluster-critical).

When any of the above pod has 'Disabled' swap-behavior:

1. Kubelet disables the pod's swap cgroup control explicitly.
2. This action is consistent with the node swap policy for the excluded pod categories.
3. Setting the pod swap limit to '0' maintains existing behavior and will not be rejected at the API level.

These node-swap policy exclusions will not have explicit validations because they align with the 'Disabled' swap-behavior.

Kubelet Changes

Kubelet will recognize and act upon the `pod.spec.swapPolicy` field as follows:

1. If the kubelet configuration `memorySwap.swapBehavior` is `NoSwap`, kubelet will disable system-level swap and `pod.spec.swapPolicy` has no effect (existing behavior).

2. When kubelet is in `LimitedSwap` mode and the `PodswapPolicy` feature gate is disabled, the `pod.spec.swapPolicy` field is ignored. In this configuration, burstable pod containers will get a swap proportion based on the memory requested for these containers (existing behavior).
3. **Pod Swap Behavior Enforcement:**
   With `PodswapPolicy` feature-gate enabled,
   a. If the node has swap configured (eg: `LimitedSwap`) and `pod.spec.swapPolicy.mode` is set to `Disabled`, Kubelet will configure the pod's cgroupv2 swap control `memory.swap.max=0` to prevent swap usage. Their containers will not get swap allocation.
   b. If `pod.spec.swapPolicy` is unset or `pod.spec.swapPolicy.mode` set to `NoPreference`, the pod adheres to existing node-level swap policy and QoS rules.

Backward Compatibility

1. The new field is optional. API server will enforce immutability of this field after it is created.
2. Pods created without `swapPolicy` will continue to function as they do today. Existing pods utilizing swap on enabled nodes will continue to benefit from swap without any change.
3. Older kubelets will ignore the field. Newer kubelets will act on `pod.spec.swapPolicy` when feature-gate is enabled.

User Stories

Story 1: As an application owner of a latency-critical service, I set `pod.spec.swapPolicy.mode: "Disabled"` to ensure my pods never use swap in my multi-tenant setup.

Story 2: As an application owner managing swap tolerant batch jobs, I do not set `pod.spec.swapPolicy`, allowing it to use *any* swap as available if the node-policy permits.

Story 3: As a cluster administrator, I enable `LimitedSwap` on nodes for cost optimization. The `pod.spec.swapPolicy` allows specific workloads to opt-out.

Risks and Mitigations

1. Ability to disable swap at workload level may hide other concerns of swap such as noisy-neighbor issues, where a heavy swap reliant workload could cause cpu or I/O contention with another workload that doesn't use swap.

    a. `PodSwapAwareness` is not a replacement for node level swap isolation for workloads. `PodSwapAwareness` feature should protect workloads that naturally fit in the same nodes as some swap requiring workloads, but cannot afford the performance cost. The alternative for these workloads is to also use swap or be OOM killed due to memory pressure (when no swap is present).
    b. When node-capability based filtering is available, users can utilize it to select swap-disabled nodes during scheduling for clear-isolation, managing explicit node-swap preferences.
2. User confusion about interactions between API and node-level configuration.
    a. A potential mitigation would be to improve pod-level visibility to clearly indicate whether a workload is actively utilizing swap memory.
    b. Clear documentation on pod-level, node-level and QoS dependencies on swap will be added.

Design Details

Test plan

[X] I/we understand the owners of the involved components may require updates to existing tests to make this code solid enough prior to committing the changes necessary to implement this enhancement.

Prerequisite testing updates

<!--
Based on reviewers feedback describe what additional tests need to be added prior implementing this enhancement to ensure the enhancements have also solid foundations.
-->

Unit tests

<!--
In principle every added code should have complete unit test coverage, so providing the exact set of tests will not bring additional value.
However, if complete unit test coverage is not possible, explain the reason of it together with explanation why this is acceptable.
-->

<!--
Additionally, for Alpha try to enumerate the core package you will be touching to implement this enhancement and provide the current unit coverage for those

in the form of:
- <package>: <date> - <current test coverage>
The data can be easily read from:
https://testgrid.k8s.io/sig-testing-canaries#ci-kubernetes-coverage-unit

This can inform certain test coverage improvements that we want to do before
extending the production code to implement this enhancement.
-->

- `kubernetes/kubernetes/tree/master/pkg/kubelet`: <date> - <test coverage>
    - Parsing `PodSwapAwareness`, feature-gate, `memory.swap.max` manipulation
- `kubernetes/kubernetes/tree/master/pkg/apis/core/v1/validation`
    - Validation of new API field for immutability

Integration tests

<!--
This question should be filled when targeting a release.
For Alpha, describe what tests will be added to ensure proper quality of the enhancement.

For Beta and GA, add links to added tests together with links to k8s-triage for those tests:
https://storage.googleapis.com/k8s-triage/index.html
-->

- Ensure kubelet handles combinations of node swap settings: cgroupv2 x swapPolicy x QoS. Verify correct swap cgroup controls are set.
    - test name: integration master, triage search

e2e tests

- Existing Swap tests must pass fine even after introducing the `PodSwapAwareness` api
- swapPolicy: “Default” on `LimitedSwap` nodes: verify no swap
- swapPolicy: “NoPreference” on `LimitedSwap` nodes: verify existing QoS rules
- Feature-gate disabled continues existing behavior
- Pods with ‘swapPolicy’ configured on a cgroupv1 node: kubelet will report an error event but will ignore the setting.
    - test name: SIG ..., triage search

Graduation Criteria

Alpha

- Feature implemented behind feature flag `PodSwapAwareness` (default false)
- Existing `node e2e` tests around swap must pass
- New e2e tests to ensure kubelet enforces `swapPolicy.mode: “Disabled”`
- Documentation: swapPolicy field, node swap interaction

Beta

- Feature gate PodswapPolicy default to true.
- Consider other “swapPolicy” values for scheduling preferences based on user feedback. Future expansions for improved scheduling awareness could include options like 'Required', 'Preferred', and 'Avoid' to express varying affinity and antiaffinity rules for workloads.

Upgrade / Downgrade Strategy

Upgrade: Feature gate is off by default in Alpha. No changes until enabled on nodes and `swapPolicy` field used on pods.

Downgrade: `swapPolicy` is ignored by older kubelets. Pods set with `swapPolicy.mode: “Disabled”`, will revert to node-level behavior.

Version Skew Strategy

Standard n-2 skew.

- Newer API / Older Kubelet: Kubelet ignores field
- Older API / Newer Kubelet: new field is not settable.

Production Readiness Review Questionnaire

<!--
This section must be completed when targeting alpha to a release.
-->

Feature Enablement and Rollback

How can this feature be enabled / disabled in a live cluster?

- ☑ Feature gate (also fill in values in kep.yaml)
  - Feature gate name: `PodSwapAwareness`

- Components depending on the feature gate: `Kubelet`

- [ ] Other
  - Describe the mechanism: `Kubelet needs to be restarted when enabling/disabling this feature`
  - Will enabling / disabling the feature require downtime of the control plane? `No`
  - Will enabling / disabling the feature require downtime or reprovisioning of a node? `Yes`

Does enabling the feature change any default behavior?

This feature introduces a new user facing behavior for swap. However the default behavior of pods are not changed with this feature.

Can the feature be disabled once it has been enabled (i.e. can we roll back the enablement)?

Requires kubelet restart. See Rollout, Upgrade and Rollout planning.

What happens if we reenable the feature if it was previously rolled back?

When the feature-flag is disabled, new field swapPolicy will be ignored by kubelet. Re-enabling the feature-flag will enable this behavior and any pods that start later on this node will adhere to the swapPolicy configured at the pod spec.

Are there any tests for feature enablement/disablement?

Rollout, Upgrade and Rollback Planning

<!--
This section must be completed when targeting beta to a release.
-->

How can a rollout or rollback fail? Can it impact already running workloads?

<!--
Try to be as paranoid as possible - e.g., what if some components will restart
mid-rollout?

Be sure to consider highly-available clusters, where, for example,
feature flags will be enabled on some API servers and not others during the
rollout. Similarly, consider large clusters and how enablement/disablement
will rollout across nodes.

-->

What specific metrics should inform a rollback?

<!--
What signals should users be paying attention to when the feature is young
that might indicate a serious problem?
-->

Were upgrade and rollback tested? Was the upgrade->downgrade->upgrade path tested?

<!--
Describe manual testing that was done and the outcomes.
Longer term, we may want to require automated upgrade/rollback tests, but we
are missing a bunch of machinery and tooling and can't do that now.
-->

Is the rollout accompanied by any deprecations and/or removals of features, APIs, fields of API types, flags, etc.?

<!--
Even if applying deprecation policies, they may still surprise some users.
-->

Monitoring Requirements

How can an operator determine if the feature is in use by workloads?

How can someone using this feature know that it is working for their instance?

Events
- Event Reason:

API .status
- Condition name:
- Other field:

Other (treat as last resort)
- Details:

What are the reasonable SLOs (Service Level Objectives) for the enhancement?

What are the SLIs (Service Level Indicators) an operator can use to determine the health of the service?

Metrics
- Metric name:
- [Optional] Aggregation method:
- Components exposing the metric:

Other (treat as last resort)
- Details:

Are there any missing metrics that would be useful to have to improve observability of this feature?

Dependencies

Does this feature depend on any specific services running in the cluster?

Scalability

Will enabling / using this feature result in any new API calls?

Will enabling / using this feature result in introducing new API types?

Will enabling / using this feature result in any new calls to the cloud provider?

Will enabling / using this feature result in increasing size or count of the existing API objects?

Will enabling / using this feature result in increasing time taken by any operations covered by existing SLIs/SLOs?

Will enabling / using this feature result in non-negligible increase of resource usage (CPU, RAM, disk, IO, ...) in any components?

Can enabling / using this feature result in resource exhaustion of some node resources (PIDs, sockets, inodes, etc.)?

Troubleshooting

How does this feature react if the API server and/or etcd is unavailable?

What are other known failure modes?

What steps should be taken if SLOs are not being met to determine the problem?

Implementation History

Drawbacks

Alternatives

Infrastructure Needed (Optional)