Resource Management for Windows nodes

This page outlines the differences in how resources are managed between Linux and Windows.

On Linux nodes, cgroups are used as a pod boundary for resource control. Containers are created within that boundary for network, process and file system isolation. The Linux cgroup APIs can be used to gather CPU, I/O, and memory use statistics.

In contrast, Windows uses a *job object* per container with a system namespace filter to contain all processes in a container and provide logical isolation from the host. (Job objects are a Windows process isolation mechanism and are different from what Kubernetes refers to as a Job).

There is no way to run a Windows container without the namespace filtering in place. This means that system privileges cannot be asserted in the context of the host, and thus privileged containers are not available on Windows. Containers cannot assume an identity from the host because the Security Account Manager (SAM) is separate.

Memory management

Windows does not have an out-of-memory process killer as Linux does. Windows always treats all user-mode memory allocations as virtual, and pagefiles are mandatory.

Windows nodes do not overcommit memory for processes. The net effect is that Windows won't reach out of memory conditions the same way Linux does, and processes page to disk instead of being subject to out of memory (OOM) termination. If memory is over-provisioned and all physical memory is exhausted, then paging can slow down performance.

CPU management

Windows can limit the amount of CPU time allocated for different processes but cannot guarantee a minimum amount of CPU time.

On Windows, the kubelet supports a command-line flag to set the scheduling priority of the kubelet process: `--windows-priorityclass`. This flag allows the kubelet process to get more CPU time slices when compared to other processes running on the Windows host. More information on the allowable values and their meaning is available at Windows Priority Classes. To ensure that running Pods do not starve the kubelet of CPU cycles, set this flag to `ABOVE_NORMAL_PRIORITY_CLASS` or above.

Resource reservation

To account for memory and CPU used by the operating system, the container runtime, and by Kubernetes host processes such as the kubelet, you can (and should) reserve memory and CPU resources with the `--kube-reserved` and/or `--system-reserved` kubelet flags. On Windows these values are only used to calculate the node's allocatable resources.

> **Caution:**
> As you deploy workloads, set resource memory and CPU limits on containers. This also subtracts from `NodeAllocatable` and helps the cluster-wide scheduler in determining which pods to place on which nodes.
>
> Scheduling pods without limits may over-provision the Windows nodes and in extreme cases can cause the nodes to become unhealthy.

On Windows, a good practice is to reserve at least 2GiB of memory.

To determine how much CPU to reserve, identify the maximum pod density for each node and monitor the CPU usage of the system services running there, then choose a value that meets your workload needs.

Feedback

Was this page helpful?

Yes No

Last modified June 06, 2022 at 2:04 PM PST: Updating Windows intro and resource management pages (#34083) (eb88ef7c3e)