Managing Compute Resources for Containers

When you specify a Pod, you can optionally specify how much CPU and memory (RAM) each Container needs. When Containers have resource requests specified, the scheduler can make better decisions about which nodes to place Pods on. And when Containers have their limits specified, contention for resources on a node can be handled in a specified manner.

Resource types

CPU and memory are each a resource type. A resource type has a base unit. CPU is specified in units of cores, and memory is specified in units of bytes.

CPU and memory are collectively referred to as compute resources, or just resources. Compute resources are measurable quantities that can be requested, allocated, and consumed. They are distinct from API resources. API resources, such as Pods and Services are objects that can be read and modified through the Kubernetes API server.

Resource requests and limits of Pod and Container

Each Container of a Pod can specify one or more of the following:

- `spec.containers[].resources.limits.cpu`
- `spec.containers[].resources.limits.memory`
- `spec.containers[].resources.requests.cpu`
- `spec.containers[].resources.requests.memory`

Although requests and limits can only be specified on individual Containers, it is convenient to talk about Pod resource requests and limits. A Pod resource request/limit for a particular resource type is the sum of the resource requests/limits of that type for each Container in the Pod.

Meaning of CPU

Limits and requests for CPU resources are measured in cpu units. One cpu, in Kubernetes, is equivalent to:

- 1 AWS vCPU
- 1 GCP Core
- 1 Azure vCore
- 1 IBM vCPU
- 1 Hyperthread on a bare-metal Intel processor with Hyperthreading

Fractional requests are allowed. A Container with `spec.containers[].resources.requests.cpu` of `0.5` is guaranteed half as much CPU as one that asks for 1 CPU. The expression `0.1` is equivalent to the expression `100m`, which can be read as “one hundred millicpu”. Some people say “one hundred millicores”, and this is understood to mean the same thing. A request with a decimal point, like `0.1`, is converted to `100m` by the API, and precision finer than `1m` is not allowed. For this reason, the form `100m` might be preferred.

CPU is always requested as an absolute quantity, never as a relative quantity; `0.1` is the same amount of CPU on a single-core, dual-core, or 48-core machine.

Meaning of memory

Limits and requests for `memory` are measured in bytes. You can express memory as a plain integer or as a fixed-point integer using one of these suffixes: E, P, T, G, M, K. You can also use the power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki. For example, the following represent roughly the same value:

```
128974848, 129e6, 129M, 123Mi
```

Here’s an example. The following Pod has two Containers. Each Container has a request of 0.25 cpu and 64MiB (2^{26} bytes) of memory. Each Container has a limit of 0.5 cpu and 128MiB of memory. You can say the Pod has a request of 0.5 cpu and 128 MiB of memory, and a limit of 1 cpu and 256MiB of memory.

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
    - name: db
      image: mysql
      env:
        - name: MYSQL_ROOT_PASSWORD
          value: "password"
      resources:
        requests:
          memory: "64Mi"
          cpu: "250m"
        limits:
```

```
          memory: "128Mi"
          cpu: "500m"
    - name: wp
      image: wordpress
      resources:
        requests:
          memory: "64Mi"
          cpu: "250m"
        limits:
          memory: "128Mi"
          cpu: "500m"
```

How Pods with resource requests are scheduled

When you create a Pod, the Kubernetes scheduler selects a node for the Pod to run on. Each node has a maximum capacity for each of the resource types: the amount of CPU and memory it can provide for Pods. The scheduler ensures that, for each resource type, the sum of the resource requests of the scheduled Containers is less than the capacity of the node. Note that although actual memory or CPU resource usage on nodes is very low, the scheduler still refuses to place a Pod on a node if the capacity check fails. This protects against a resource shortage on a node when resource usage later increases, for example, during a daily peak in request rate.

How Pods with resource limits are run

When the kubelet starts a Container of a Pod, it passes the CPU and memory limits to the container runtime.

When using Docker:

- The `spec.containers[].resources.requests.cpu` is converted to its core value, which is potentially fractional, and multiplied by 1024. The greater of this number or 2 is used as the value of the `--cpu-shares` flag in the docker run command.
- The `spec.containers[].resources.limits.cpu` is converted to its millicore value and multiplied by 100. The resulting value is the total amount of CPU time that a container can use every 100ms. A container cannot use more than its share of CPU time during this interval.

  Note *The default quota period is 100ms. The minimum resolution of CPU quota is 1ms.*

- The `spec.containers[].resources.limits.memory` is converted to an integer, and used as the value of the `--memory` flag in the docker run command.
- If a Container exceeds its memory limit, it might be terminated. If it is restartable, the kubelet will restart it, as with any other type of runtime failure.
- If a Container exceeds its memory request, it is likely that its Pod will be evicted whenever the node runs out of memory.

- A Container might or might not be allowed to exceed its CPU limit for extended periods of time. However, it will not be killed for excessive CPU usage.

Monitoring compute resource usage

The resource usage of a Pod is reported as part of the Pod status.

Heapster enables monitoring and performance analysis in Kubernetes Clusters. Heapster collects signals from kubelets and the api server, processes them, and exports them via REST APIs or to a configurable timeseries storage backend.

Local ephemeral storage

FEATURE STATE: Kubernetes v1.13 beta

Kubernetes version 1.8 introduces a new resource, `ephemeral-storage` for managing local ephemeral storage. In each Kubernetes node, kubelet's root directory (`/var/lib/kubelet` by default) and log directory (`/var/log`) are stored on the root partition of the node. This partition is also shared and consumed by Pods via emptyDir volumes, container logs, image layers and container writable layers.

This partition is "ephemeral" and applications cannot expect any performance SLAs (Disk IOPS for example) from this partition. Local ephemeral storage management only applies for the root partition; the optional partition for image layer and writable layer is out of scope.

Requests and limits setting for local ephemeral storage

Each Container of a Pod can specify one or more of the following:

- `spec.containers[].resources.limits.ephemeral-storage`
- `spec.containers[].resources.requests.ephemeral-storage`

Limits and requests for `ephemeral-storage` are measured in bytes. You can express storage as a plain integer or as a fixed-point integer using one of these suffixes: E, P, T, G, M, K. You can also use the power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki. For example, the following represent roughly the same value:

```
128974848, 129e6, 129M, 123Mi
```

For example, the following Pod has two Containers. Each Container has a request of 2GiB of local ephemeral storage. Each Container has a limit of 4GiB of local ephemeral storage. Therefore, the Pod has a request of 4GiB of local ephemeral storage, and a limit of 8GiB of storage.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
    - name: db
      image: mysql
      env:
        - name: MYSQL_ROOT_PASSWORD
          value: "password"
      resources:
        requests:
          ephemeral-storage: "2Gi"
        limits:
          ephemeral-storage: "4Gi"
    - name: wp
      image: wordpress
      resources:
        requests:
          ephemeral-storage: "2Gi"
        limits:
          ephemeral-storage: "4Gi"
```

How Pods with ephemeral-storage requests are scheduled

When you create a Pod, the Kubernetes scheduler selects a node for the Pod to run on. Each node has a maximum amount of local ephemeral storage it can provide for Pods. For more information, see Node Allocatable.

The scheduler ensures that the sum of the resource requests of the scheduled Containers is less than the capacity of the node.

How Pods with ephemeral-storage limits run

For container-level isolation, if a Container's writable layer and logs usage exceeds its storage limit, the Pod will be evicted. For pod-level isolation, if the sum of the local ephemeral storage usage from all containers and also the Pod's emptyDir volumes exceeds the limit, the Pod will be evicted.