Node Monitoring in 1.3 & beyond

Status: Draft
Author: Tim St. Clair (stclair@google.com)

Motivation

There are two types of monitoring within Kubernetes: collection of “core” metrics used by internal systems (kubelet functioning, and eventually scheduling), and monitoring of system state for introspection (at the node or cluster level). Currently, we have one solution for both of these: cAdvisor integrated into the Kubelet & polled by Heapster[1]. However, these monitoring types have different priorities.

For the *core metrics*, we want a system which is:

Reliable - Since core functionality depends on it
Standard - Fewer ways to incorrectly configure the cluster
Lightweight - Since it runs on all deployments & is not optional
Predictable - Specifically with regards to performance, for managing kubelet overhead

For cluster introspection, we want a system which is **extensible**, above all else - since everyone has different needs from such a system. We should also have a **easy** out-of-the-box solution for new users.

Goals

1. Summary API is promoted to GA as the canonical source-of-truth for core metrics
2. Core metrics gathering is light enough that requests for options to disable it are dropped
3. Provide a solution for new users looking for node-level detailed introspection

Non Goals

- Providing a generic cluster-level monitoring abstraction
- Supporting extensibility for core metrics in Kubelet

[1] Heapster actually polls the summary API which is not part of cAdvisor, but most of the stats are aggregated from the cAdvisor libraries.

Summary API Improvements

With Kubernetes 1.2 we launched the summary API as an alpha API, and made it the default endpoint for Heapster metrics gathering. For 1.3, we should work on solidifying the API and endpoint in the following ways.

Testing

We have unit tests operating on fake data in place, and did lots of manual testing before launching, but cAdvisor and the whole metrics pipeline is largely untested when it comes to automated testing on real data. We should define a range of acceptable values for every metric and write e2e tests that verify the metrics. We should also invest in a performance regression test. Of course we should also fix any issues (and pay careful attention to flakes) that are revealed in the tests.
https://github.com/google/cadvisor/issues/945
https://github.com/kubernetes/kubernetes/issues/23411
Work Estimate: 3 days

Endpoints

The summary API is currently an unversioned endpoint (<node_host:port>/stats/summary) which makes it difficult to make changes to. We should define an official API endpoint scheme for all Kubelet endpoints, and make sure the summary API conforms. We should also provide official documentation of the kubelet endpoints. We may want to invest in using the API server machinery for the Kubelet.
Work Estimate: days - weeks, depending on scope

Custom Metrics

We want to remove custom metrics from the core summary API. If we revisit the the goals I think it is clear that custom metrics address all the goals for the introspection system, and none of the goals for the core metrics system. This is slightly complicated by the cross-over use case of custom metrics for HPA, but I think that should be addressed in the cluster-level (Heapster; possibly with node-level component), and not in kubelet itself. The Heapster changes require a much more extensive design (details TBD), but the gist is that alternative (extensible) sources could be added to Heapster to pull from various monitoring options, such as Prometheus, OpenTSDB, etc.
Work Estimate: 1-2 days for kubelet, unknown for Heapster (needs design)

Hyper

TBD - how to integrate hyper monitoring with the summary API.

Performance Improvements

Rather than providing extension points and/or ways of disabling the core metrics gathering, I believe we can meet our extensibility goals by making core metrics collection sufficiently lightweight (in terms of resource usage). There are lots of opportunities for low level optimizations, but the following are some low hanging fruit improvements with large gains:

Disable unused metrics

cAdvisor currently gathers some metrics (such as Disk I/O) which are not included in the summary API. We may decide to include these metrics some day, but as long as we are not directly using them, they should be disabled.
Work Estimate: 1 day

Increase collection period

The current default “housekeeping” (metrics collection) period is 10 seconds, but Heapster only scrapes the metrics from the Node every 30 seconds by default, so 2 of every 3 stats sets are thrown away[2]. We can reduce our resource usage by nearly 66% just by changing the default period to match that of Heapster. This would impact the granularity of metrics available for introspection, but under this model that would be handled by a separate system (possibly another cAdvisor instance). One challenge which needs more consideration here is whether it is acceptable for Heapster to collect metrics up to 30s old. Another option is to switch to an on-demand gather model, if the request latency is acceptable. This would add some complexity within the Kubelet when stats are needed for internal usage, though.
https://github.com/kubernetes/kubernetes/issues/23413
Work Estimate: 2 hours - 4 days (depending on level of synchronization with Heapster)

Improved Serialization

A non-trivial amount of CPU and memory is still spent (de)serializing metrics data. Generated JSON codecs (such as ffjson or ugorji) or binary formats (e.g. protobufs) can speed this up by 2-3x.
https://github.com/kubernetes/kubernetes/issues/23412
Work Estimate: ~2 days, depending on the level of automation

---

[2] Exceptions: CPU instantaneous usage is calculated from the previously collected metrics (issue). Other systems (or users - including the UI) could be scraping the metrics more frequently.

Custom Metrics

Discussed above. Disabling custom metrics in the Kubelet is important from a performance perspective since the cost of custom metrics is theoretically unbounded, which opposes our predictability goal.

Standalone cAdvisor

We should not cripple the introspection potential of the Kubelet without offering users (especially new users) an easy to use (and setup) solution. In the long term, we hope that the community will step up here and offer multiple easy to deploy options.

The simplest solution for now would be to offer an “official” cAdvisor manifest. This would most likely look like a DaemonSet which would run a cAdvisor instance in a pod with sane defaults. This would only include the normal cAdvisor functionality which exposes everything at the container level (i.e. no Pod level abstractions). The manifest would be included with Kubernetes distributions, so users could simply do `kubectl create -f path/to/cadvisor_manifest.yaml` to bring it up.
Work Estimate: 1 week

A more involved solution would be to create a Kubernetes wrapper around cAdvisor and offer that as a DaemonSet. The wrapper could inject Kubernetes concepts (pods, namespaces, volumes, etc.) into the cAdvisor API and endpoints, to make it easier to introspect from a k8s perspective.
Work Estimate: 3+ weeks, depending on scope