KubeCon | CloudNativeCon

Europe 2023

KubeCon | CloudNativeCon

Europe 2023

Making Sense of Your Vital Signals: The Future of Pod and Containers Monitoring

David Porter, Google
Peter Hunt, Red Hat

Making Sense of Your Vital Signals: The Future of Pod and Containers Monitoring

KubeCon | CloudNativeCon

Europe 2023

David Porter
@bobbypage
Sr. Software Engineer
Google

Peter Hunt
@haircommander
Sr. Software Engineer
Red Hat

You were paged! Why?!?

🚨 ALERT 🚨

{pod namespace: AI, name: kube-chatbot}
p90 latency of average chat response rpc > 1 minute

What just happened and why?

You were paged! Why?!?

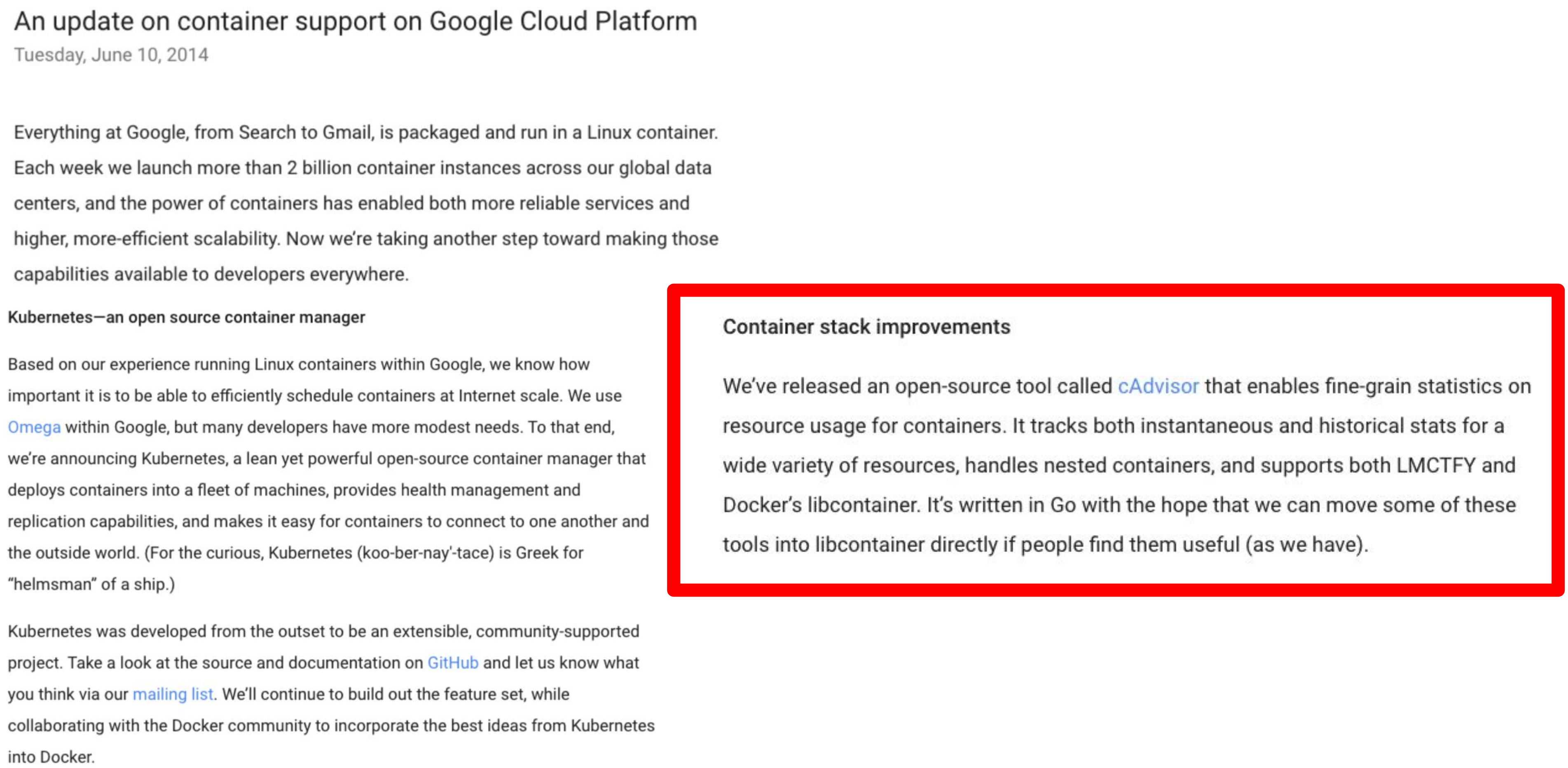
- Observability
  - Understand resource usage, changes with deployments, rollouts
  - Identify issues and unexpected behavior with applications
  - Alerting on unexpected conditions
- SLO/SLIs
- Node stability
  - Kubelet subcomponents
    - (e.g. eviction manager) depend on metrics to understand which pods to evict
      - e.g. pod that over consumes ephemeral storage will be evicted

There's a lot of metrics in k8s...

- Node Level Metrics (i.e. node-exporter)
- Kubernetes component metrics (e.g. api-server, controller manager, scheduler, kubelet etc)
- Derived metrics from API resources (e.g. kube-state-metrics)
- Pod and Containers [Workload] metrics

- Today we will be focusing on **Pod and Containers [Workload] metrics**

cAdvisor Humble Beginning



An update on container support on Google Cloud Platform

Tuesday, June 10, 2014

Everything at Google, from Search to Gmail, is packaged and run in a Linux container. Each week we launch more than 2 billion container instances across our global data centers, and the power of containers has enabled both more reliable services and higher, more-efficient scalability. Now we're taking another step toward making those capabilities available to developers everywhere.

Kubernetes—an open source container manager

Based on our experience running Linux containers within Google, we know how important it is to be able to efficiently schedule containers at Internet scale. We use Omega within Google, but many developers have more modest needs. To that end, we're announcing Kubernetes, a lean yet powerful open-source container manager that deploys containers into a fleet of machines, provides health management and replication capabilities, and makes it easy for containers to connect to one another and the outside world. (For the curious, Kubernetes (koo-ber-nay'-tace) is Greek for "helmsman" of a ship.)

Kubernetes was developed from the outset to be an extensible, community-supported project. Take a look at the source and documentation on GitHub and let us know what you think via our mailing list. We'll continue to build out the feature set, while collaborating with the Docker community to incorporate the best ideas from Kubernetes into Docker.

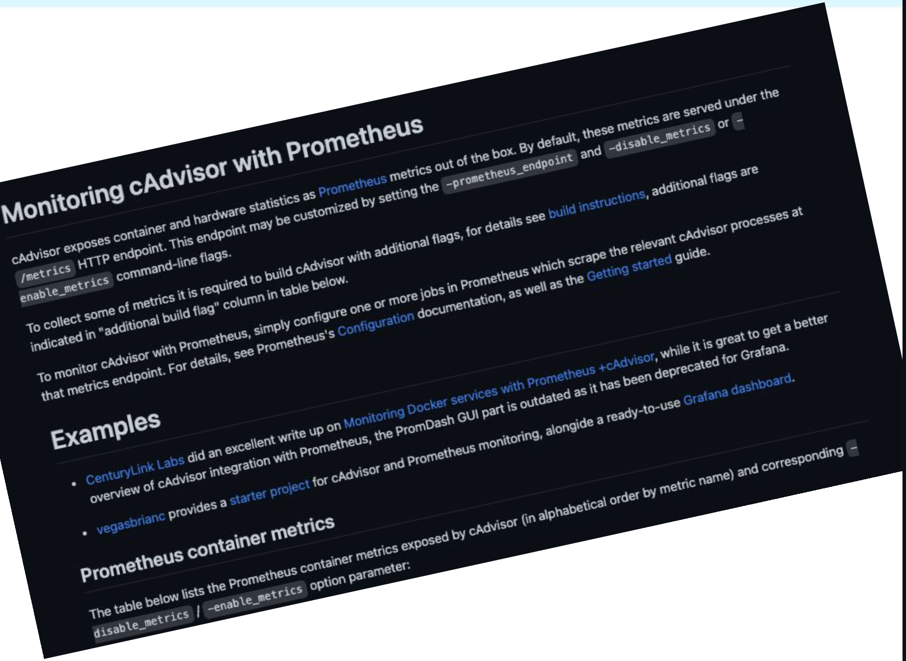
Container stack improvements

We've released an open-source tool called cAdvisor that enables fine-grain statistics on resource usage for containers. It tracks both instantaneous and historical stats for a wide variety of resources, handles nested containers, and supports both LMCTFY and Docker's libcontainer. It's written in Go with the hope that we can move some of these tools into libcontainer directly if people find them useful (as we have).

What is cAdvisor?

- Provides observability for containerized workloads
  - Scrapes and collects containers running on the node
  - Parses the information and provides in multiple formats
  - "In tree" support for docker, containerd, CRI-O
  - Uses runc's libcontainer library to scrape cgroupfs
- cAdvisor can be used in:
  - "standalone mode" - Run as daemonset
  - "library" - Use it as a library from golang
- **Kubelet** depends on cAdvisor as a library

There's quite a few metrics...



Monitoring cAdvisor with Prometheus

cAdvisor exposes container and hardware statistics as Prometheus metrics out of the box. By default, these metrics are served under the `/metrics` HTTP endpoint. This endpoint may be customized by setting the `-prometheus_endpoint` and `-disable_metrics` or `-enable_metrics` command-line flags.

To collect some of metrics it is required to build cAdvisor with additional flags, for details see build instructions, additional flags are indicated in "additional build flag" column in table below.

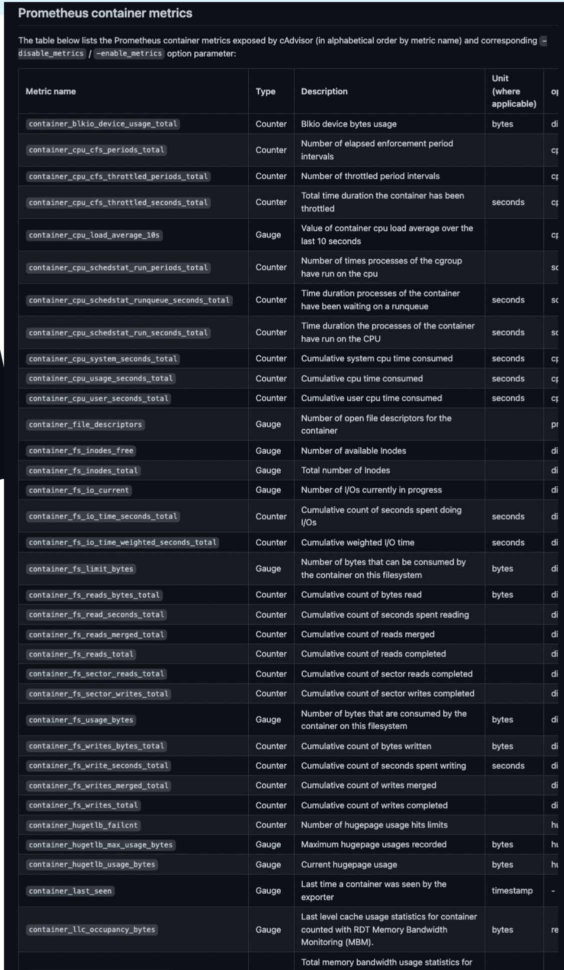
To monitor cAdvisor with Prometheus, simply configure one or more jobs in Prometheus which scrape the relevant cAdvisor processes at that metrics endpoint. For details, see Prometheus's Configuration documentation, as well as the Getting started guide.

Examples

- CenturyLink Labs did an excellent write up on Monitoring Docker services with Prometheus +cAdvisor, while it is great to get a better overview of cAdvisor integration with Prometheus, the PromDash GUI part is outdated as it has been deprecated for Grafana.
- vegasbrianc provides a starter project for cAdvisor and Prometheus monitoring, alongide a ready-to-use Grafana dashboard.

Prometheus container metrics

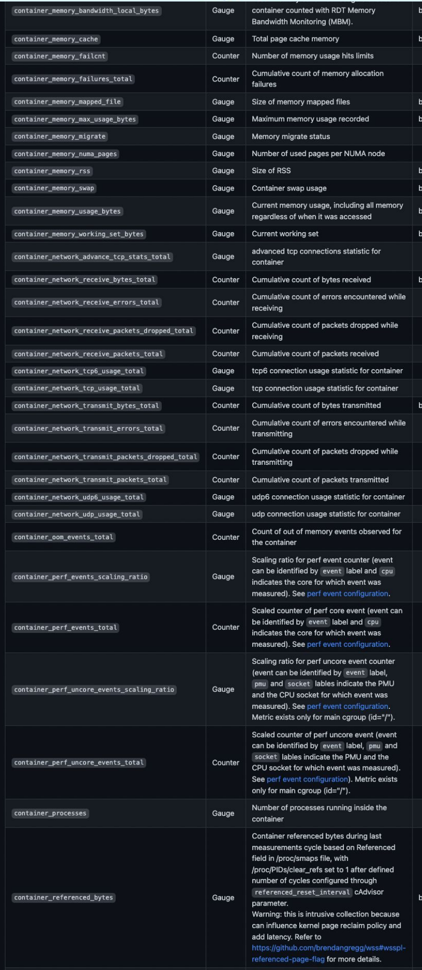
The table below lists the Prometheus container metrics exposed by cAdvisor (in alphabetical order by metric name) and corresponding `-disable_metrics` / `-enable_metrics` option parameter:



Prometheus container metrics

The table below lists the Prometheus container metrics exposed by cAdvisor (in alphabetical order by metric name) and corresponding `-disable_metrics` / `-enable_metrics` option parameter:

Metric name	Type	Description	Unit (where applicable)	op
`container_blkio_device_usage_total`	Counter	Blkio device bytes usage	bytes	di
`container_cpu_cfs_periods_total`	Counter	Number of elapsed enforcement period intervals		cp
`container_cpu_cfs_throttled_periods_total`	Counter	Number of throttled period intervals		cp
`container_cpu_cfs_throttled_seconds_total`	Counter	Total time duration the container has been throttled	seconds	cp
`container_cpu_load_average_10s`	Gauge	Value of container cpu load average over the last 10 seconds		cp
`container_cpu_schedstat_run_periods_total`	Counter	Number of times processes of the cgroup have run on the cpu		sc
`container_cpu_schedstat_runqueue_seconds_total`	Counter	Time duration processes of the container have been waiting on a runqueue	seconds	sc
`container_cpu_schedstat_run_seconds_total`	Counter	Time duration the processes of the container have run on the CPU	seconds	sc
`container_cpu_system_seconds_total`	Counter	Cumulative system cpu time consumed	seconds	cp
`container_cpu_usage_seconds_total`	Counter	Cumulative cpu time consumed	seconds	cp
`container_cpu_user_seconds_total`	Counter	Cumulative user cpu time consumed	seconds	cp
`container_file_descriptors`	Gauge	Number of open file descriptors for the container		pr
`container_fs_inodes_free`	Gauge	Number of available Inodes		di
`container_fs_inodes_total`	Gauge	Total number of Inodes		di
`container_fs_io_current`	Gauge	Number of I/Os currently in progress		di
`container_fs_io_time_seconds_total`	Counter	Cumulative count of seconds spent doing I/Os	seconds	di
`container_fs_io_time_weighted_seconds_total`	Counter	Cumulative weighted I/O time	seconds	di
`container_fs_limit_bytes`	Gauge	Number of bytes that can be consumed by the container on this filesystem	bytes	di
`container_fs_reads_bytes_total`	Counter	Cumulative count of bytes read	bytes	di
`container_fs_read_seconds_total`	Counter	Cumulative count of seconds spent reading		di
`container_fs_reads_merged_total`	Counter	Cumulative count of reads merged		di
`container_fs_reads_total`	Counter	Cumulative count of reads completed		di
`container_fs_sector_reads_total`	Counter	Cumulative count of sector reads completed		di
`container_fs_sector_writes_total`	Counter	Cumulative count of sector writes completed		di
`container_fs_usage_bytes`	Gauge	Number of bytes that are consumed by the container on this filesystem	bytes	di
`container_fs_writes_bytes_total`	Counter	Cumulative count of bytes written	bytes	di
`container_fs_write_seconds_total`	Counter	Cumulative count of seconds spent writing	seconds	di
`container_fs_writes_merged_total`	Counter	Cumulative count of writes merged		di
`container_fs_writes_total`	Counter	Cumulative count of writes completed		di
`container_hugetlb_failcnt`	Counter	Number of hugepage usage hits limits		hu
`container_hugetlb_max_usage_bytes`	Gauge	Maximum hugepage usages recorded	bytes	hu
`container_hugetlb_usage_bytes`	Gauge	Current hugepage usage	bytes	hu
`container_last_seen`	Gauge	Last time a container was seen by the exporter	timestamp	-
`container_llc_occupancy_bytes`	Gauge	Last level cache usage statistics for container counted with RDT Memory Bandwidth Monitoring (MBM).	bytes	re
		Total memory bandwidth usage statistics for		

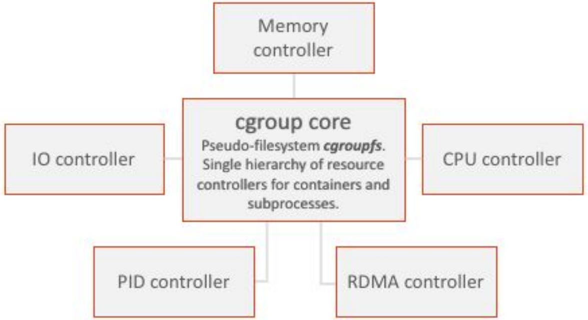


`container_memory_bandwidth_local_bytes`	Gauge	container counted with RDT Memory Bandwidth Monitoring (MBM).	b
`container_memory_cache`	Gauge	Total page cache memory	b
`container_memory_failcnt`	Counter	Number of memory usage hits limits	
`container_memory_failures_total`	Counter	Cumulative count of memory allocation failures	
`container_memory_mapped_file`	Gauge	Size of memory mapped files	b
`container_memory_max_usage_bytes`	Gauge	Maximum memory usage recorded	b
`container_memory_migrate`	Gauge	Memory migrate status	
`container_memory_numa_pages`	Gauge	Number of used pages per NUMA node	
`container_memory_rss`	Gauge	Size of RSS	b
`container_memory_swap`	Gauge	Container swap usage	b
`container_memory_usage_bytes`	Gauge	Current memory usage, including all memory regardless of when it was accessed	b
`container_memory_working_set_bytes`	Gauge	Current working set	b
`container_network_advance_tcp_stats_total`	Gauge	advanced tcp connections statistic for container	
`container_network_receive_bytes_total`	Counter	Cumulative count of bytes received	b
`container_network_receive_errors_total`	Counter	Cumulative count of errors encountered while receiving	
`container_network_receive_packets_dropped_total`	Counter	Cumulative count of packets dropped while receiving	
`container_network_receive_packets_total`	Counter	Cumulative count of packets received	
`container_network_tcp6_usage_total`	Gauge	tcp6 connection usage statistic for container	
`container_network_tcp_usage_total`	Gauge	tcp connection usage statistic for container	
`container_network_transmit_bytes_total`	Counter	Cumulative count of bytes transmitted	b
`container_network_transmit_errors_total`	Counter	Cumulative count of errors encountered while transmitting	
`container_network_transmit_packets_dropped_total`	Counter	Cumulative count of packets dropped while transmitting	
`container_network_transmit_packets_total`	Counter	Cumulative count of packets transmitted	
`container_network_udp6_usage_total`	Gauge	udp6 connection usage statistic for container	
`container_network_udp_usage_total`	Gauge	udp connection usage statistic for container	
`container_oom_events_total`	Counter	Count of out of memory events observed for the container	
`container_perf_events_scaling_ratio`	Gauge	Scaling ratio for perf event counter (event can be identified by `event` label and `cpu` indicates the core for which event was measured). See perf event configuration.	
`container_perf_events_total`	Counter	Scaled counter of perf core event (event can be identified by `event` label and `cpu` indicates the core for which event was measured). See perf event configuration.	
`container_perf_uncore_events_scaling_ratio`	Gauge	Scaling ratio for perf uncore event counter (event can be identified by `event` label, `pmu` and `socket` lables indicate the PMU and the CPU socket for which event was measured). See perf event configuration. Metric exists only for main cgroup (id="/").	
`container_perf_uncore_events_total`	Counter	Scaled counter of perf uncore event (event can be identified by `event` label, `pmu` and `socket` lables indicate the PMU and the CPU socket for which event was measured). See perf event configuration). Metric exists only for main cgroup (id="/").	
`container_processes`	Gauge	Number of processes running inside the container	
`container_referenced_bytes`	Gauge	Container referenced bytes during last measurements cycle based on Referenced field in /proc/smaps file, with /proc/PIDs/clear_refs set to 1 after defined number of cycles configured through `referenced_reset_interval` cAdvisor parameter. Warning: this is intrusive collection because can influence kernel page reclaim policy and add latency. Refer to https://github.com/brendangregg/wss#wsspl-referenced-page-flag for more details.	b

Where do the workload metrics come from?

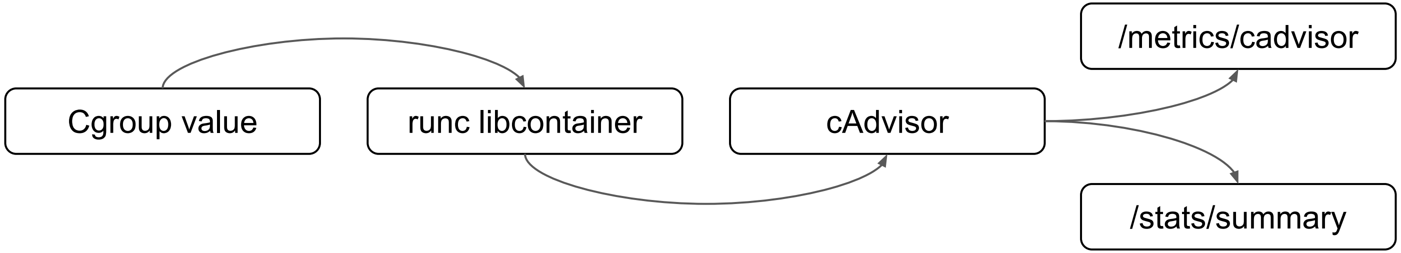
- cgroups [v1 & v2]
  - Linux Kernel feature that provides ability to:
    - Group a set of process hierarchically
    - Set of **controllers** (cpu, memory, io, etc…) to manage and limit resources in groups and **provide monitoring**
  - Allow us to:
    - Limit usage of group of process (amount of CPU or memory or pids).
    - Measure resource usage for a group of processes

See Kubecon NA 2022 -
Cgroupv2 Is Coming Soon To a Cluster Near You -
David Porter, Google & Mrunal Patel, RedHat (xref)



Memory controller
IO controller
cgroup core
Pseudo-filesystem *cgroupfs*. Single hierarchy of resource controllers for containers and subprocesses.
CPU controller
PID controller
RDMA controller

A Day in the Life of a Metric



Cgroup value

runc libcontainer

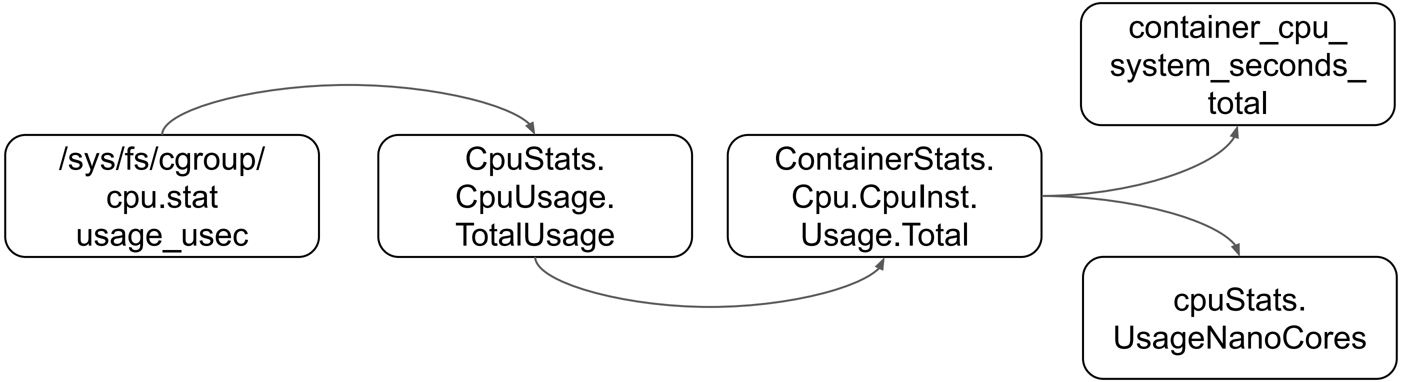
cAdvisor

/metrics/cadvisor

/stats/summary

A Day in the Life of a Metric

KubeCon CloudNativeCon Europe 2023



/sys/fs/cgroup/ cpu.stat usage_usec

CpuStats. CpuUsage. TotalUsage

ContainerStats. Cpu.CpuInst. Usage.Total

container_cpu_ system_seconds_ total

cpuStats. UsageNanoCores

How is cAdvisor exposed today

- Kubelet exposes the cAdvisor metrics via
  - /metrics/cadvisor (direct prometheus)

```
kubectl get --raw "/api/v1/nodes/kind-worker/proxy/metrics/cadvisor"
```

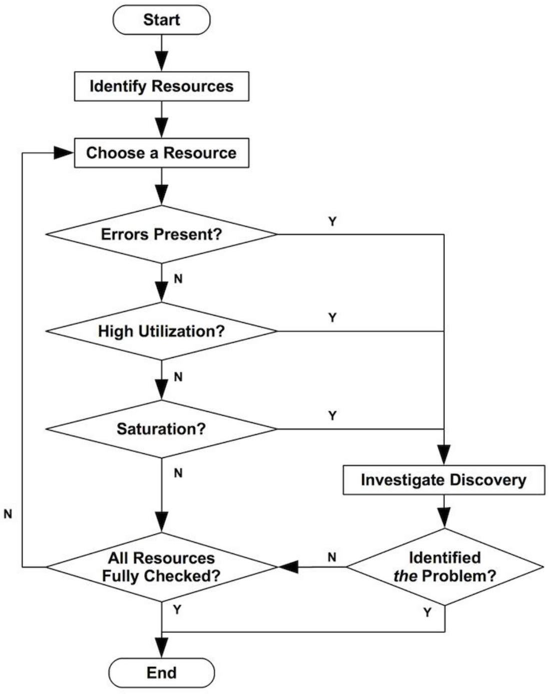
  - /stats/summary (json)

```
kubectl get --raw "/api/v1/nodes/kind-worker/proxy/stats/summary"
```

  - /metrics/resource (metrics server)
- Kubelet also depends on cAdvisor for:
  - Gathering node level stats
  - Eviction Manager

Case study: CPU

- 🚨 ALERT 🚨 My application is slow, let's investigate!
- First, we need a strategy (i.e. methodology)
- USE (Introduced by Brendan Gregg): For each resource (cpu, memory, IO, storage) consider:
    - Utilization
        - The average time that the resource was busy servicing work
    - Saturation
        - The degree to which the resource has extra work which it can't service, often queued
    - Errors (if applicable)



Kube Chat Bot

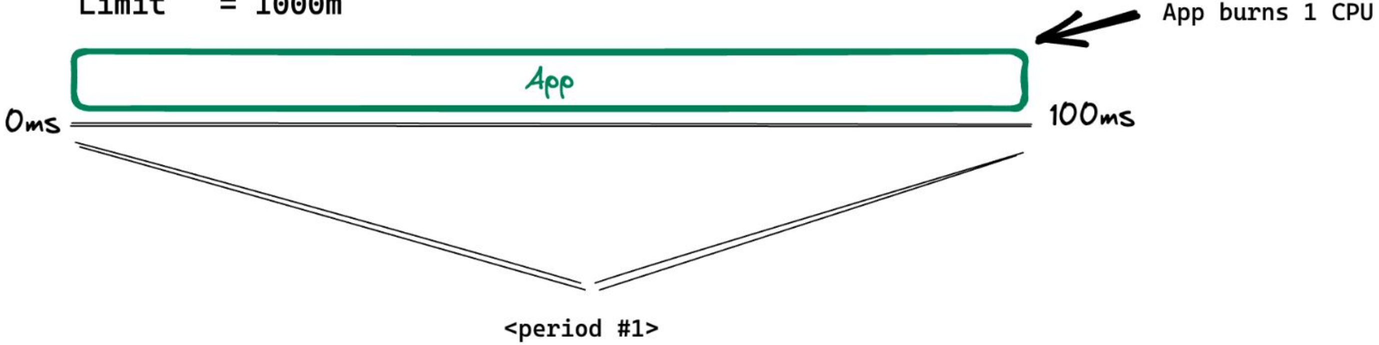
```
$ cat kubechatbot.yaml
apiVersion: v1
kind: Pod
metadata:
  name: kube-chatbot
 spec:
  containers:
  - name: "main"
    image: "chatbot:latest"
    resources:
      requests:
        cpu: "2000m"
      limits:
        cpu: "2000m"
```

CPU Requests & Limits

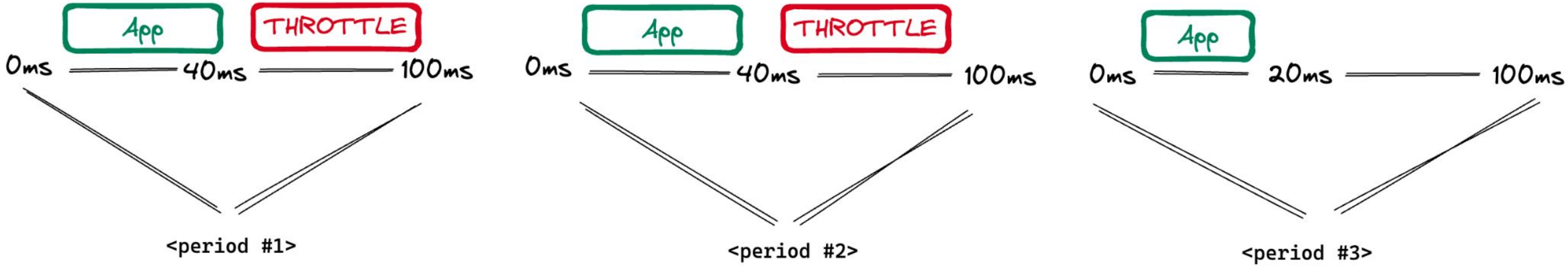
- Requests
  - Minimum Floor for CPU - you will always get CPU request
- Limits
  - Ceiling for CPU - you will be throttled going above limit
- How does it work?
  - CPU Shares
  - CFS Bandwidth Control
    - CPU Quota [time slice] and CPU Period (100ms)

CPU Limits (CFS Bandwidth)

```
Request = 1000m
Limit   = 1000m
```

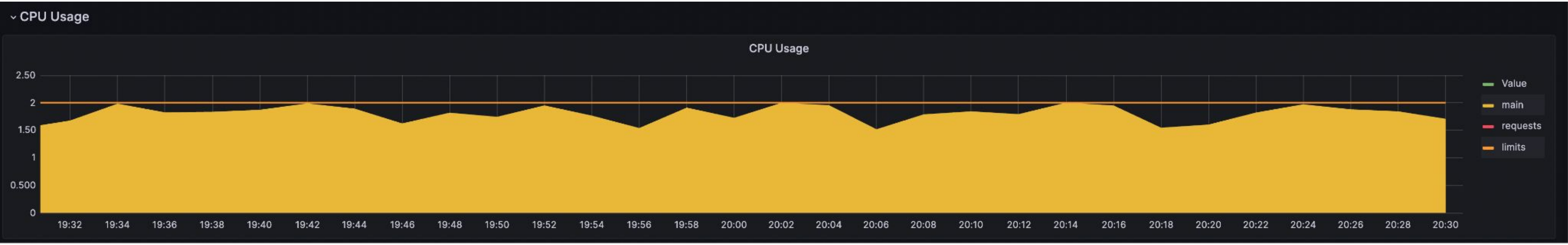


```
Limit   = 400m
```



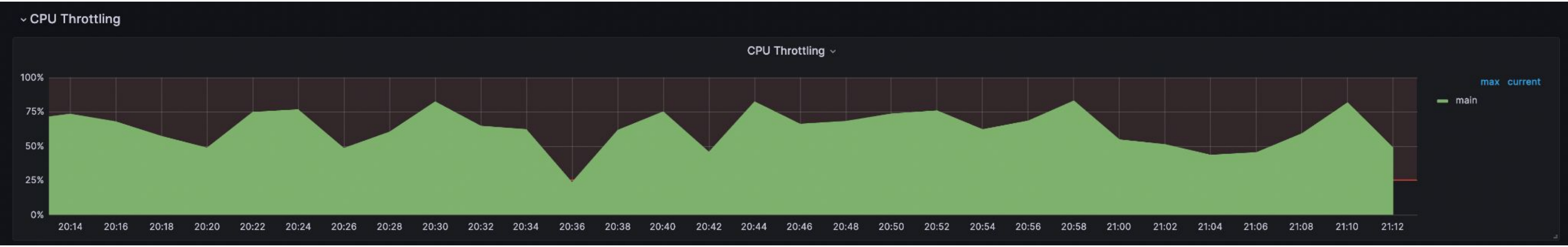
CPU utilization

```
rate(container_cpu_usage_seconds_total)
```



CPU Saturation

- What container workload metrics measure throttling?
  - `container_cpu_cfs_periods_total`
  - `container_cpu_cfs_throttled_periods_total`
- `Throttled Percentage = Throttled Periods / Total Periods`



Case Study: CPU throttling ✅

- How do we fix it?
    - Increase the CPU limit (and requests)!

Future of Metrics

https://www.craiyon.com/

cAdvisor Limitations

- Monolithic design
- Barely CRI aware
- Kernel separated containers not supported
  - Windows
  - Kata containers, etc
- Duplicated
  - Performance impact

https://github.com/google/cadvisor/raw/master/logo.png

Who should own metrics collection?

cri-o
cAdvisor
containerd
kubernetes

KEP-2371

cAdvisor

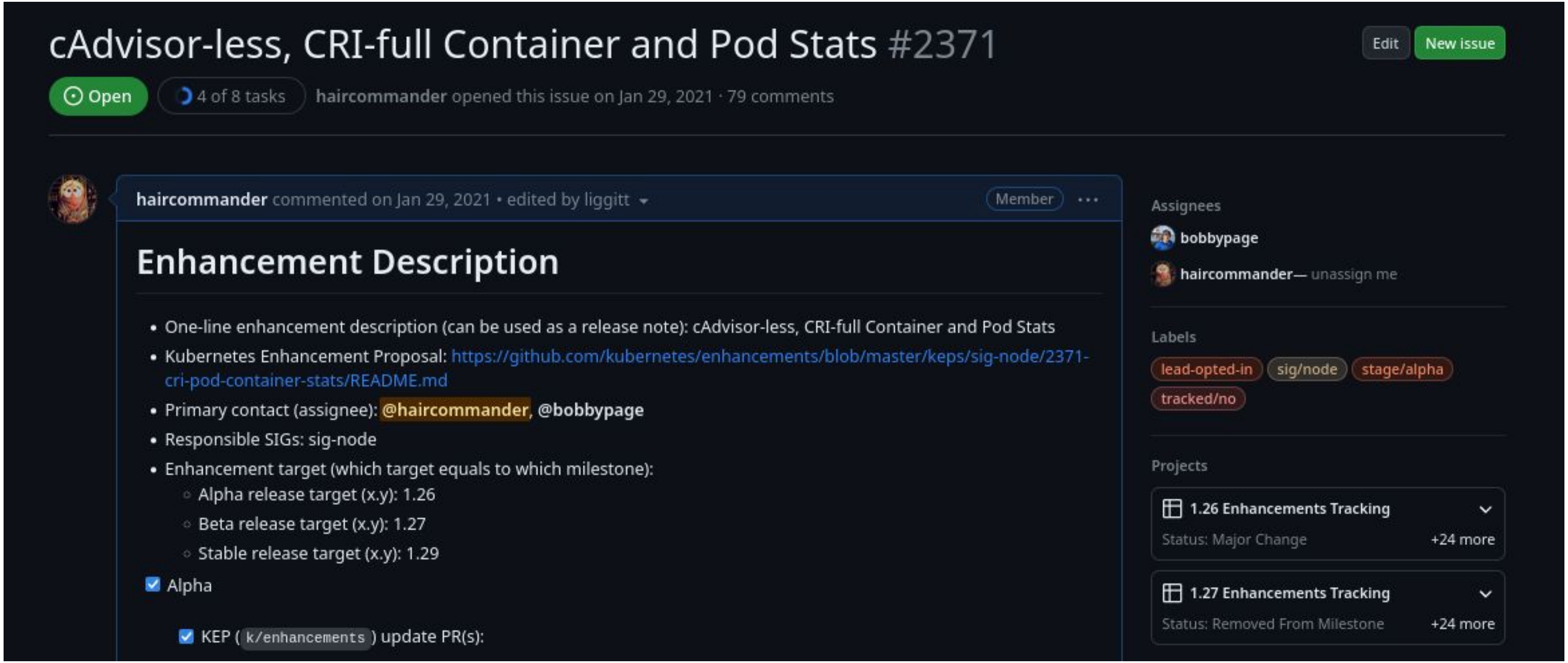
WE SHOULD TAKE METRICS COLLECTION
AND PUSH IT TO THE CRI
imgflip.com

https://imgflip.com/memegenerator/Put-It-Somewhere-Else-Patrick

cri-o

containerd

KEP-2371



cAdvisor-less, CRI-full Container and Pod Stats #2371

haircommander opened this issue on Jan 29, 2021 · 79 comments

haircommander commented on Jan 29, 2021 • edited by liggitt

Enhancement Description

- One-line enhancement description (can be used as a release note): cAdvisor-less, CRI-full Container and Pod Stats
- Kubernetes Enhancement Proposal: https://github.com/kubernetes/enhancements/blob/master/keps/sig-node/2371-cri-pod-container-stats/README.md
- Primary contact (assignee): @haircommander, @bobbypage
- Responsible SIGs: sig-node
- Enhancement target (which target equals to which milestone):
  - Alpha release target (x.y): 1.26
  - Beta release target (x.y): 1.27
  - Stable release target (x.y): 1.29

☑ Alpha

  ☑ KEP (`k/enhancements`) update PR(s):

Assignees

bobbypage

haircommander— unassign me

Labels

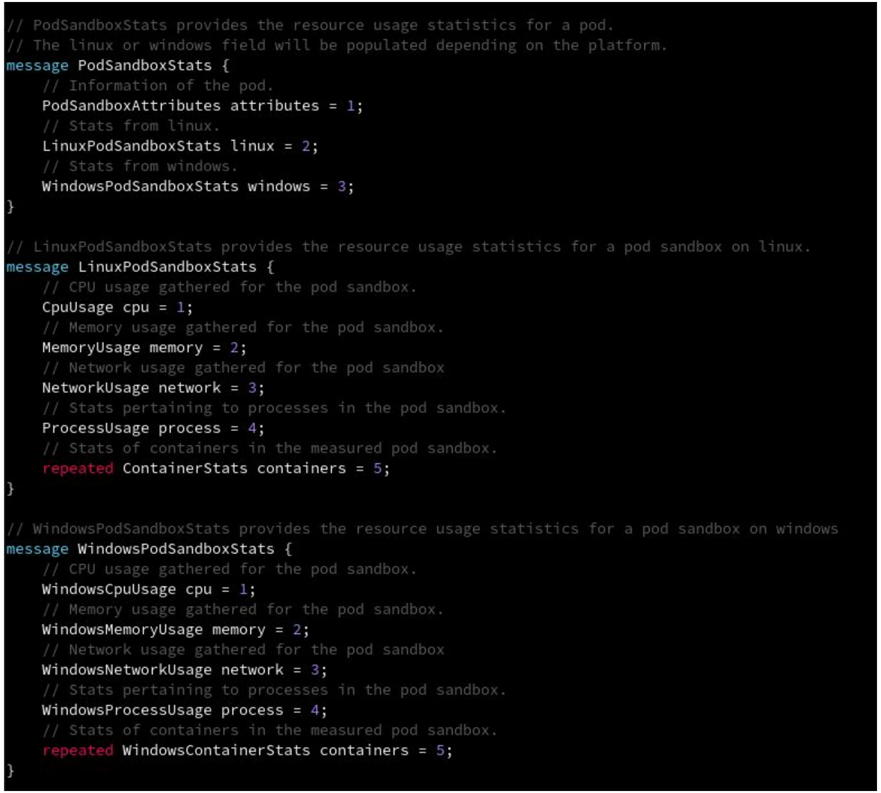
lead-opted-in sig/node stage/alpha tracked/no

Projects

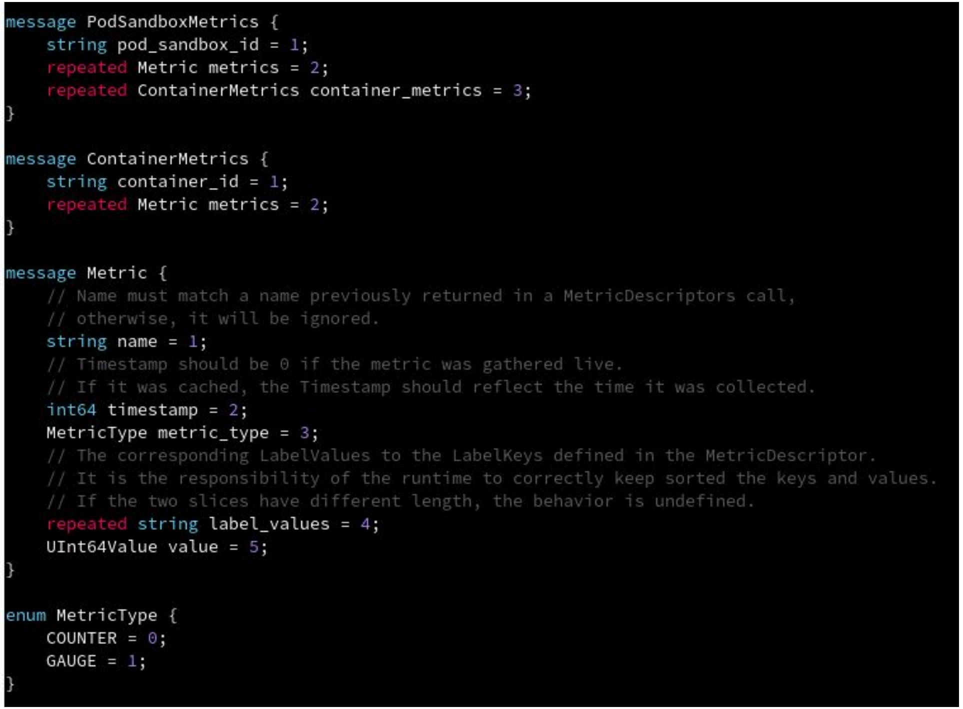
1.26 Enhancements Tracking
Status: Major Change +24 more

1.27 Enhancements Tracking
Status: Removed From Milestone +24 more

Alpha state



```
// PodSandboxStats provides the resource usage statistics for a pod.
// The linux or windows field will be populated depending on the platform.
message PodSandboxStats {
    // Information of the pod.
    PodSandboxAttributes attributes = 1;
    // Stats from linux.
    LinuxPodSandboxStats linux = 2;
    // Stats from windows.
    WindowsPodSandboxStats windows = 3;
}

// LinuxPodSandboxStats provides the resource usage statistics for a pod sandbox on linux.
message LinuxPodSandboxStats {
    // CPU usage gathered for the pod sandbox.
    CpuUsage cpu = 1;
    // Memory usage gathered for the pod sandbox.
    MemoryUsage memory = 2;
    // Network usage gathered for the pod sandbox
    NetworkUsage network = 3;
    // Stats pertaining to processes in the pod sandbox.
    ProcessUsage process = 4;
    // Stats of containers in the measured pod sandbox.
    repeated ContainerStats containers = 5;
}

// WindowsPodSandboxStats provides the resource usage statistics for a pod sandbox on windows
message WindowsPodSandboxStats {
    // CPU usage gathered for the pod sandbox.
    WindowsCpuUsage cpu = 1;
    // Memory usage gathered for the pod sandbox.
    WindowsMemoryUsage memory = 2;
    // Network usage gathered for the pod sandbox
    WindowsNetworkUsage network = 3;
    // Stats pertaining to processes in the pod sandbox.
    WindowsProcessUsage process = 4;
    // Stats of containers in the measured pod sandbox.
    repeated WindowsContainerStats containers = 5;
}
```



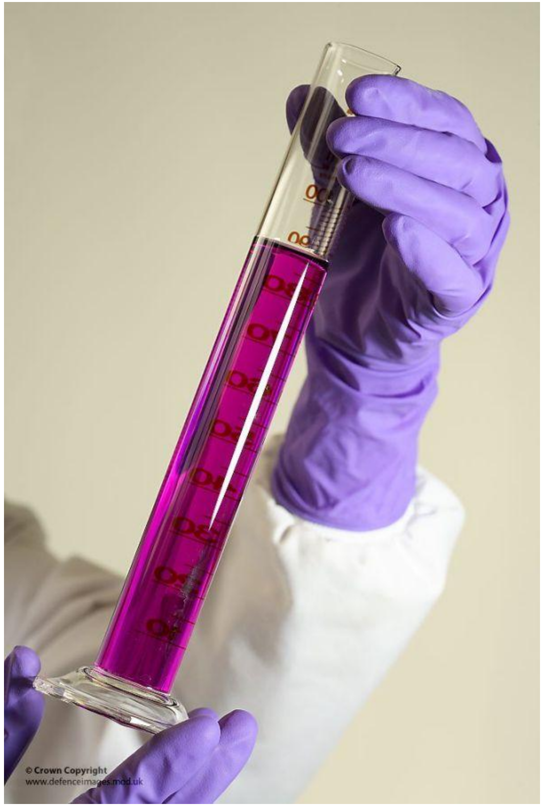
```
message PodSandboxMetrics {
    string pod_sandbox_id = 1;
    repeated Metric metrics = 2;
    repeated ContainerMetrics container_metrics = 3;
}

message ContainerMetrics {
    string container_id = 1;
    repeated Metric metrics = 2;
}

message Metric {
    // Name must match a name previously returned in a MetricDescriptors call,
    // otherwise, it will be ignored.
    string name = 1;
    // Timestamp should be 0 if the metric was gathered live.
    // If it was cached, the Timestamp should reflect the time it was collected.
    int64 timestamp = 2;
    MetricType metric_type = 3;
    // The corresponding LabelValues to the LabelKeys defined in the MetricDescriptor.
    // It is the responsibility of the runtime to correctly keep sorted the keys and values.
    // If the two slices have different length, the behavior is undefined.
    repeated string label_values = 4;
    UInt64Value value = 5;
}

enum MetricType {
    COUNTER = 0;
    GAUGE = 1;
}
```

How CRI stats will be exposed tomorrow

- Kubelet exposes the CRI metrics via
  - `/metrics/cadvisor`
    - Interpreted from Metrics object of CRI
  - `/stats/summary`
    - Interpreted from Stats object of CRI
  - `/metrics/resource`
    - Interpreted from Stats object of CRI
- Kubelet **still** depends on cAdvisor for:
  - Gathering node level stats
  - Eviction Manager

Beta goals

- Testing
  - Accuracy
  - Metric coverage
  - Performance



Scientist Holding Tube of Liquid at Porton Down
CC BY-NC

End user impact?

Hopefully, none!

- /stats/summary is a stable API of the Kubelet and its fields can be relied upon
- /metrics/cadvisor has become a “stable” API of the Kubelet
- In general: testing should™ prevent regressions

https://openverse.org/image/6a6b8dec-8550-4e4d-b0d6-3afe3e768761?q=asteroid%20impact

Summary

- Observability helps gain insights in your application performance
- cAdvisor powers workload and container monitoring in Kubernetes
- We are working on KEP-2371, "cAdvisor-less, CRI-full Container and Pod Stats KEP" moves pod and container stats into the CRI
    - Contributions welcome, chat to us in SIG-Node!

Thank you

- SIG Node Community
- cAdvisor maintainers
- Container Runtime Maintainers (containerd, CRI-O, Moby/Docker)
- runc maintainers (libcontainer / runc)

More Resources

- KEP issue - https://kep.k8s.io/2371
- Full KEP https://github.com/kubernetes/enhancements/blob/master/keps/sig-node/2371-cri-pod-container-stats/README.md
- cAdvisor - github.com/google/cadvisor
- Container CPU Throttling - https://aws.amazon.com/blogs/containers/using-prometheus-to-avoid-disasters-with-kubernetes-cpu-limits/

Any Questions?

Please scan the QR Code above
to leave feedback on this session