
KubeCon
CloudNativeCon
North America 2019

Weighing a Cloud:

Measuring Your Kubernetes Clusters

Han Kang, Google & Elana Hashman, Red Hat

Who are we?

Han Kang

Senior Software Engineer

- Cluster Ops Lead at Google
- SIG API-Machinery and SIG Instrumentation Member
- Twitter: **@LogicalHan**
- GitHub: **@logicalhan**

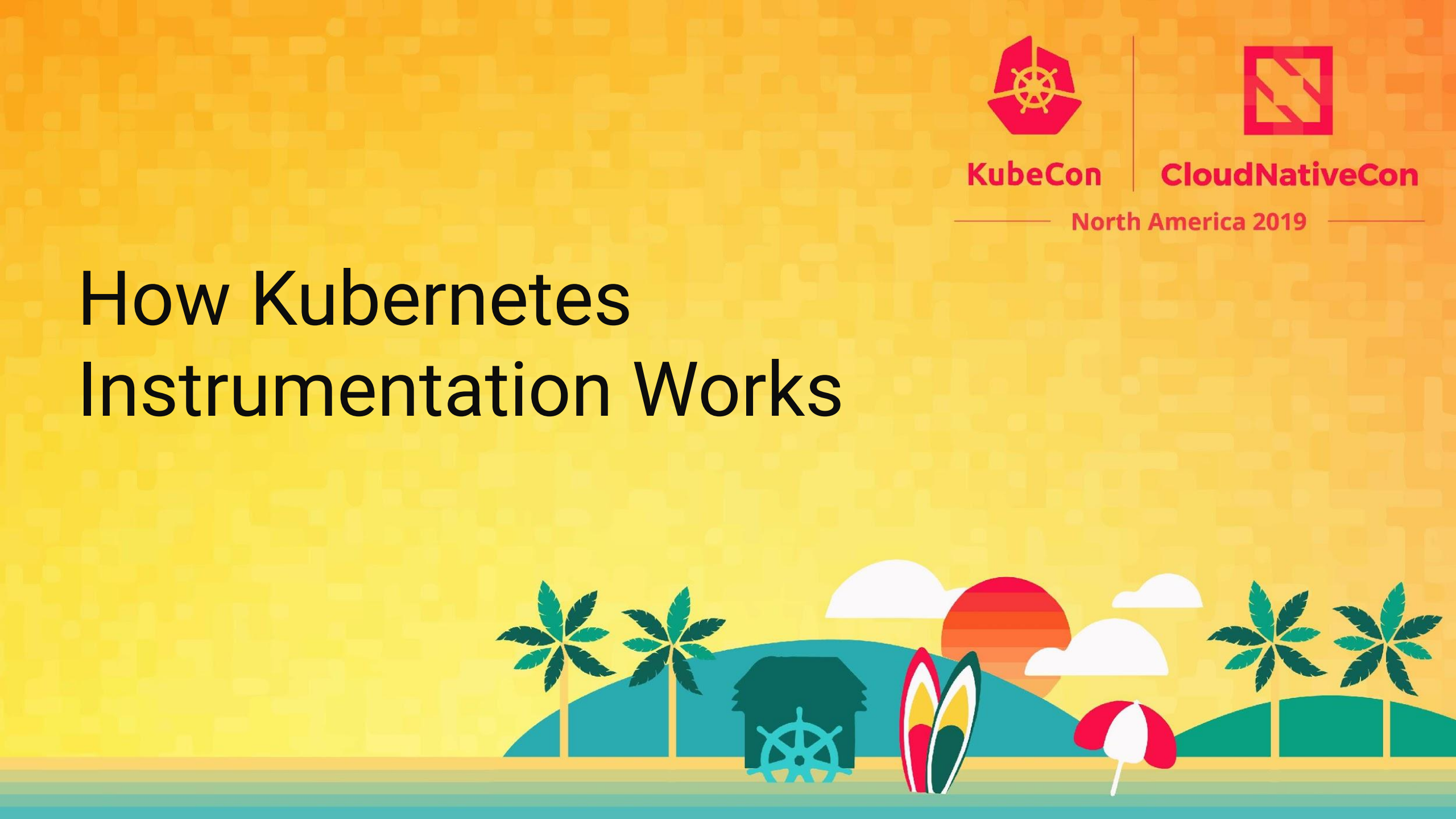
Elana Hashman

Principal Site Reliability Engineer

- Tech Lead on Azure Red Hat OpenShift Team
- SIG Instrumentation Member
- Twitter: **@ehashdn**
- GitHub: **@ehashman**

What we are going to cover

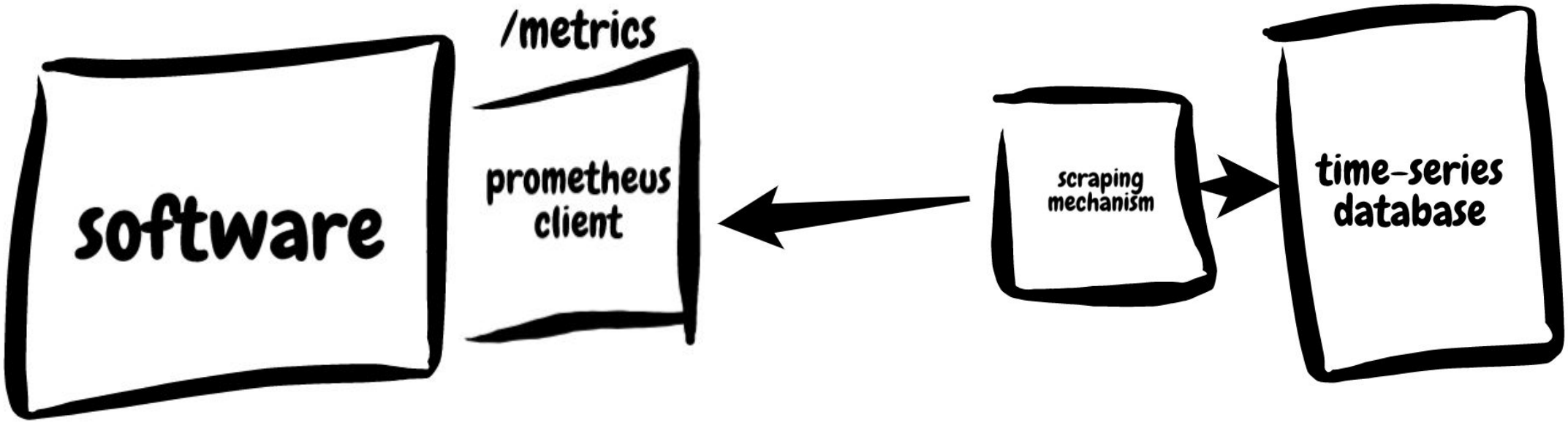
- How instrumentation works in Kubernetes
- Kubernetes control plane instrumentation
- Real-world debugging!
- Metric usability and SIG Instrumentation roadmap



How Kubernetes Instrumentation Works

Prometheus

Kubernetes components integrate with Prometheus, a time-series based monitoring and alerting toolkit.



Prometheus Data Model

Timeseries	Value
`up{job="kube-apiserver",instance="api-1"}`	1

```
# HELP up If the scrape target is reachable
# TYPE up gauge
```

Types of metric values:

- Counters
- Gauges
- Summaries
- Histograms

Dimensions of Measurement

1. Availability
   - `up{job="kubernetes-apiservers"}`
2. Latency
   - `apiserver_request_latency_seconds`
3. Capacity
   - `apiserver_request_total`
4. Errors
   - `apiserver_dropped_requests_total`

Using Prometheus Metrics

Prometheus query language (PromQL) powers metrics analysis and aggregation

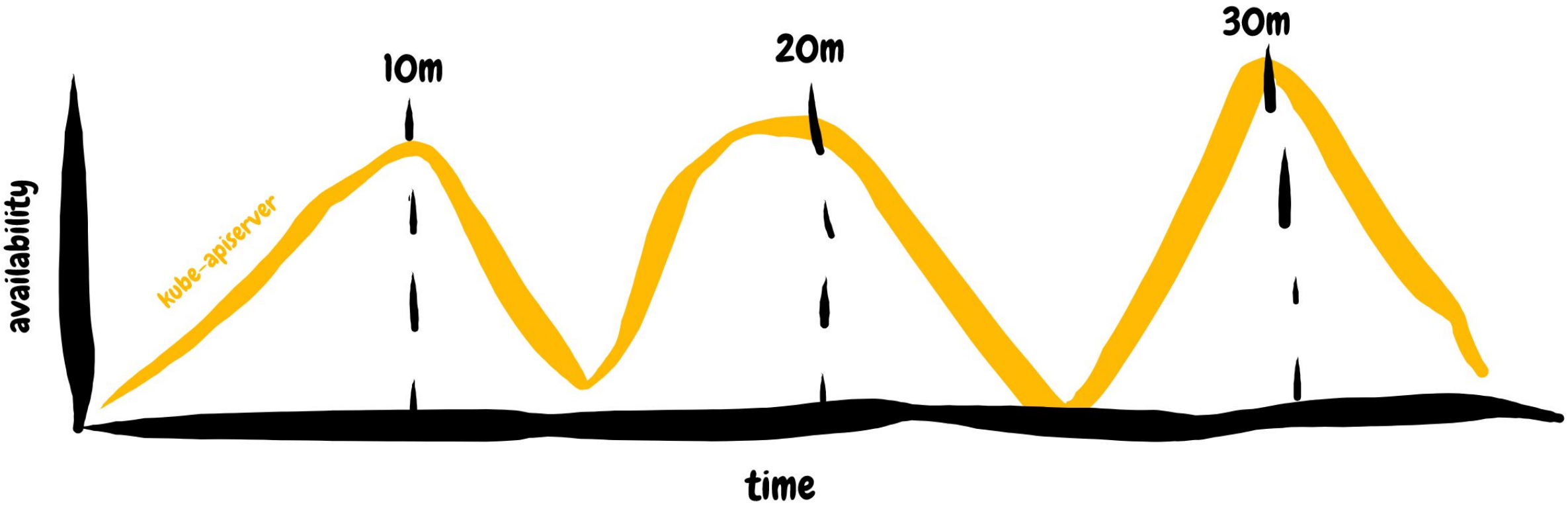
- **For prototyping and exploration:** use the Prometheus UI
- **For permanent dashboards:** attach a Prometheus data source to Grafana
- **For alerting:** set up the Prometheus Alert Manager
- **For arbitrary queries and processing:** query the Prometheus API

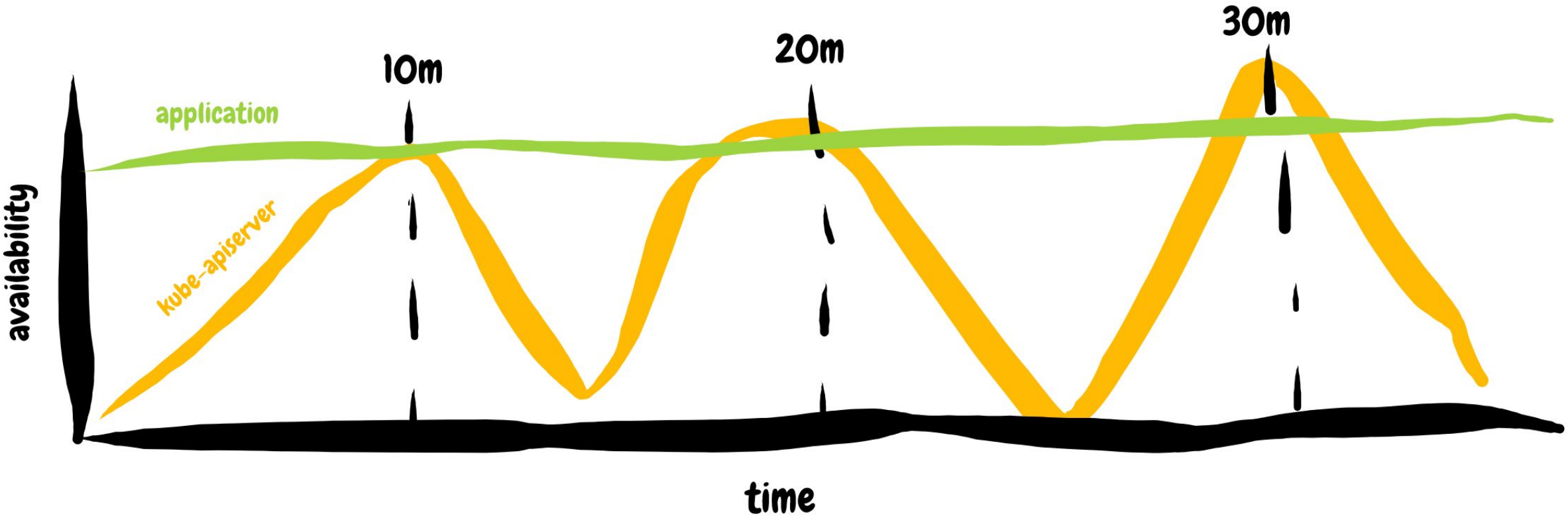
Differential Diagnoses

- Lots of very different issues might manifest the same way
  - e.g. “a node is offline” -- but why?
- A single symptom is not sufficient to form a diagnosis
- Metrics can show **how** something is failing, but not **why**
- We must track down root causes with multiple data sources

Full-Stack Debugging

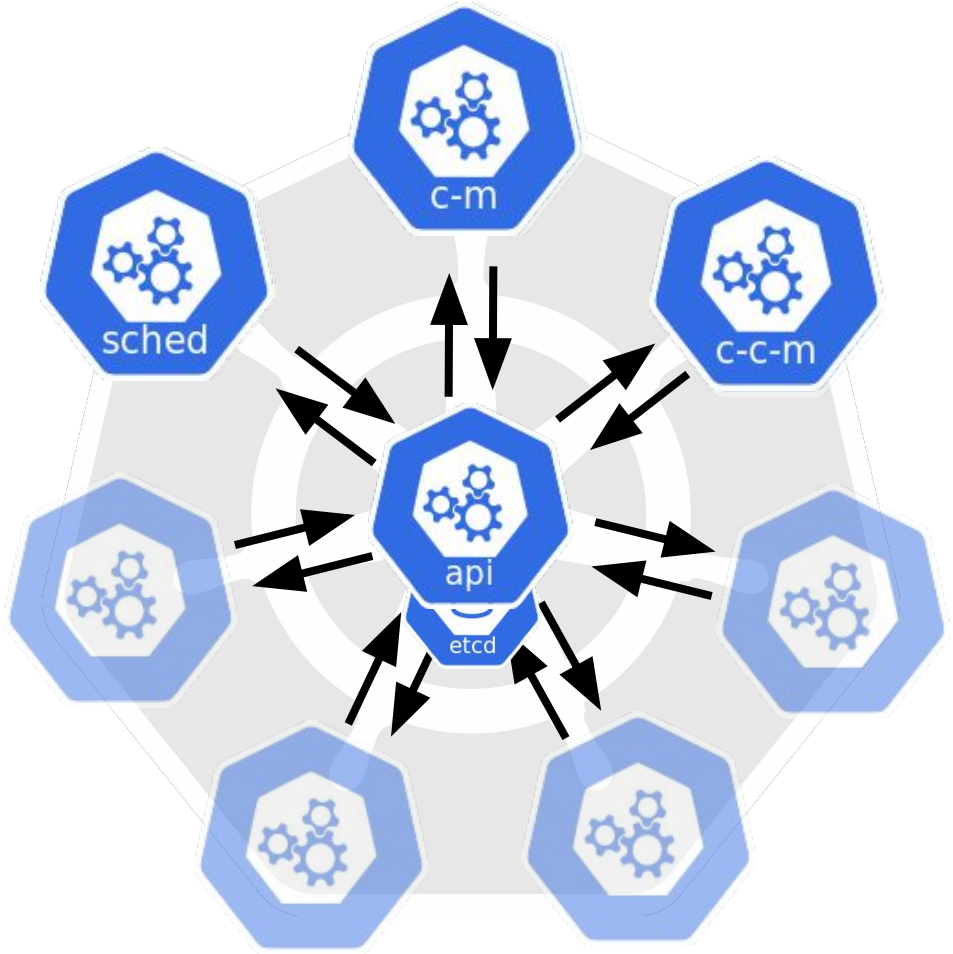
- Metrics can guide you to what you should look at next
- Not just metrics!
  - log files
  - audit logs
  - events
  - etcd (cluster database) dumps
- Metrics are most effective when you understand the context in which they were produced.


10m
20m
30m
availability
kube-apiserver
time

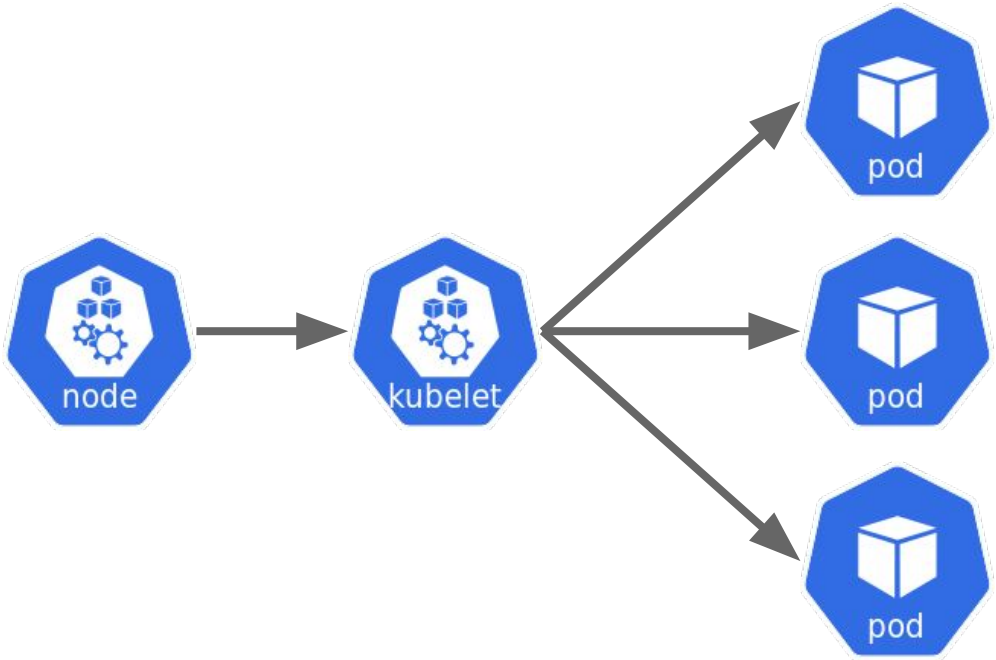

10m
20m
30m
application
kube-apiserver
availability
time

Kubernetes Control Plane Instrumentation

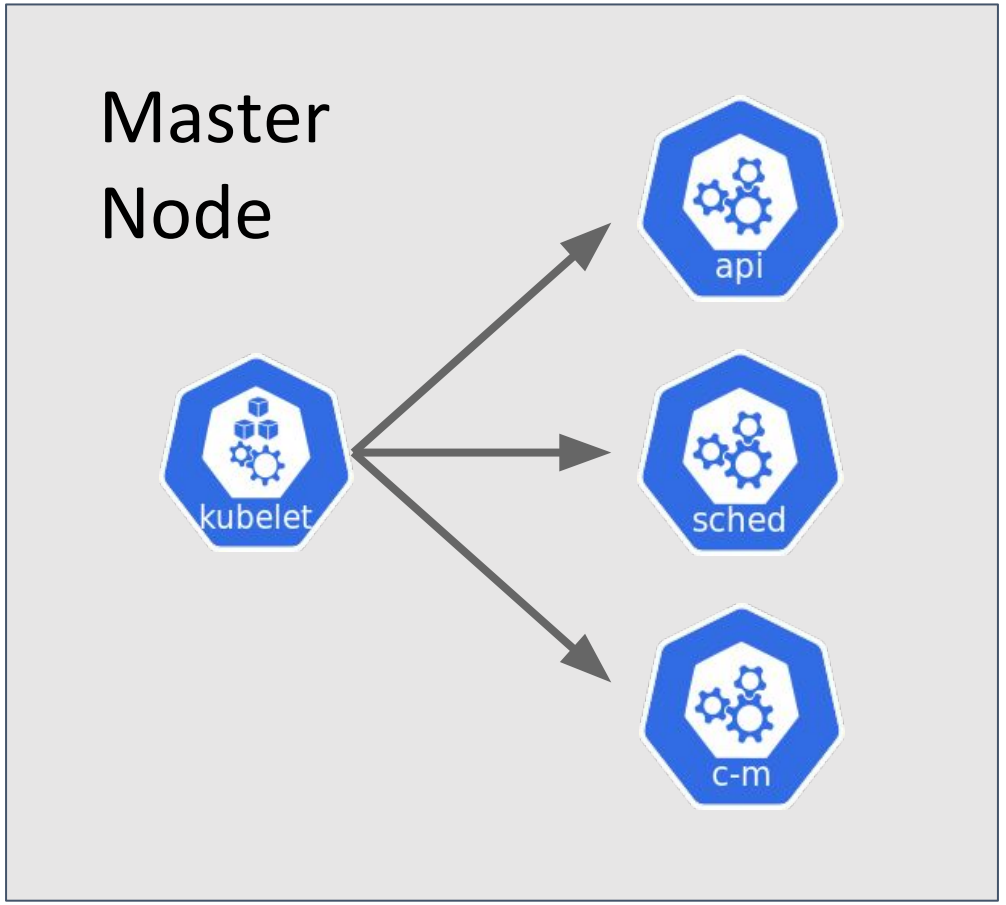
Control Plane



Kubelet



Master Kubelet

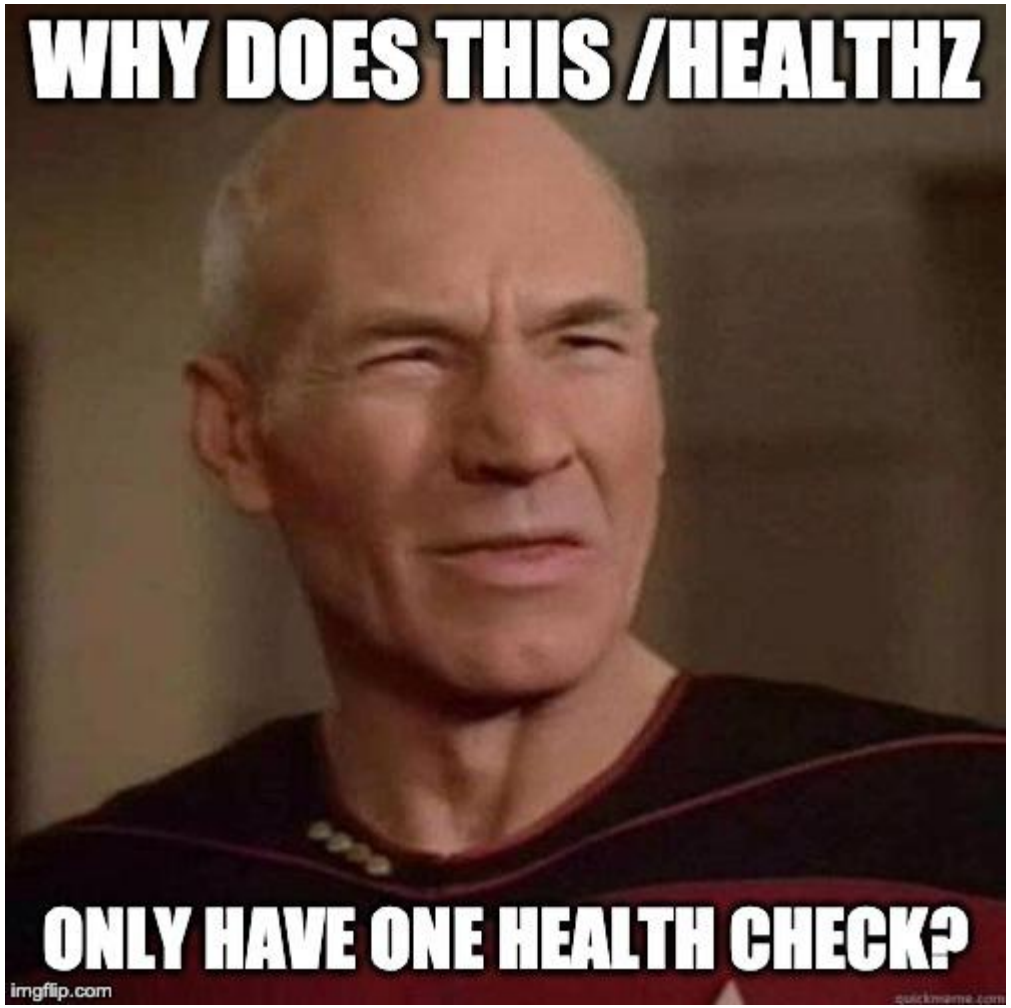


Introspecting Components

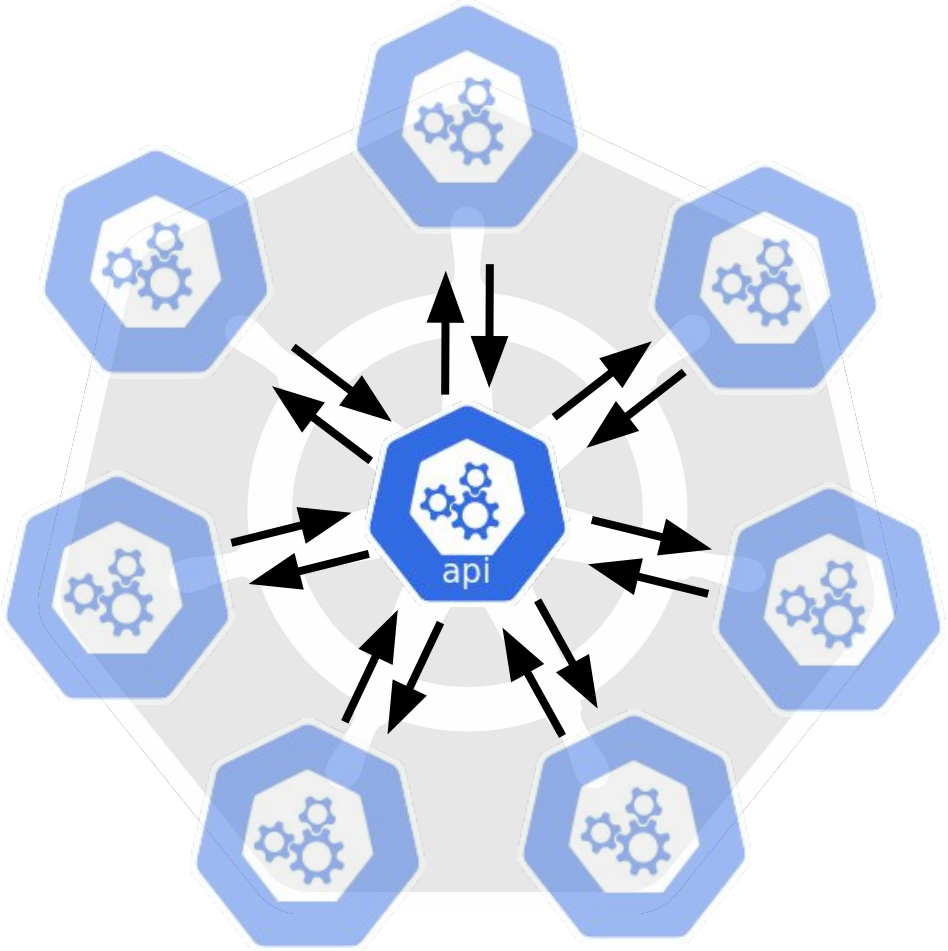
1. health check endpoint(s)
2. metrics
3. logs

Introspecting Components

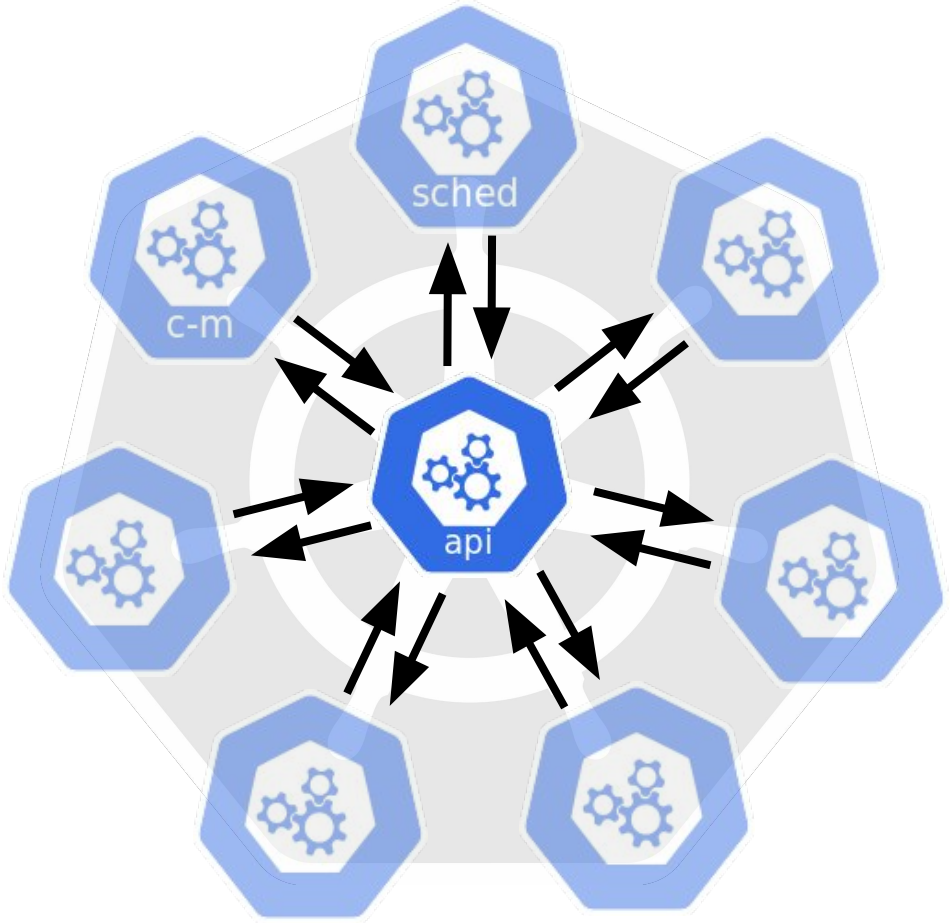

```
$ curl localhost:10251/healthz?verbose
[+]leaderElection ok
healthz check passed
```



KAS (Kube-apiserver)



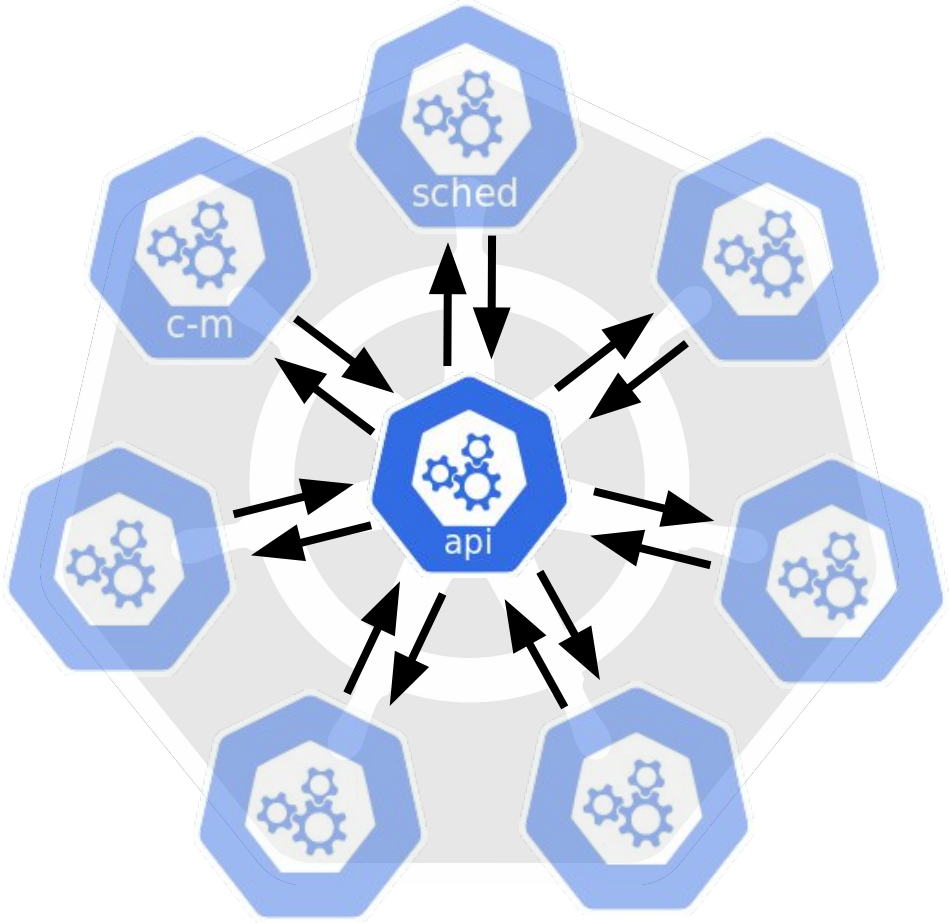
Kube-apiserver



- `kubectl <command> -v=9`

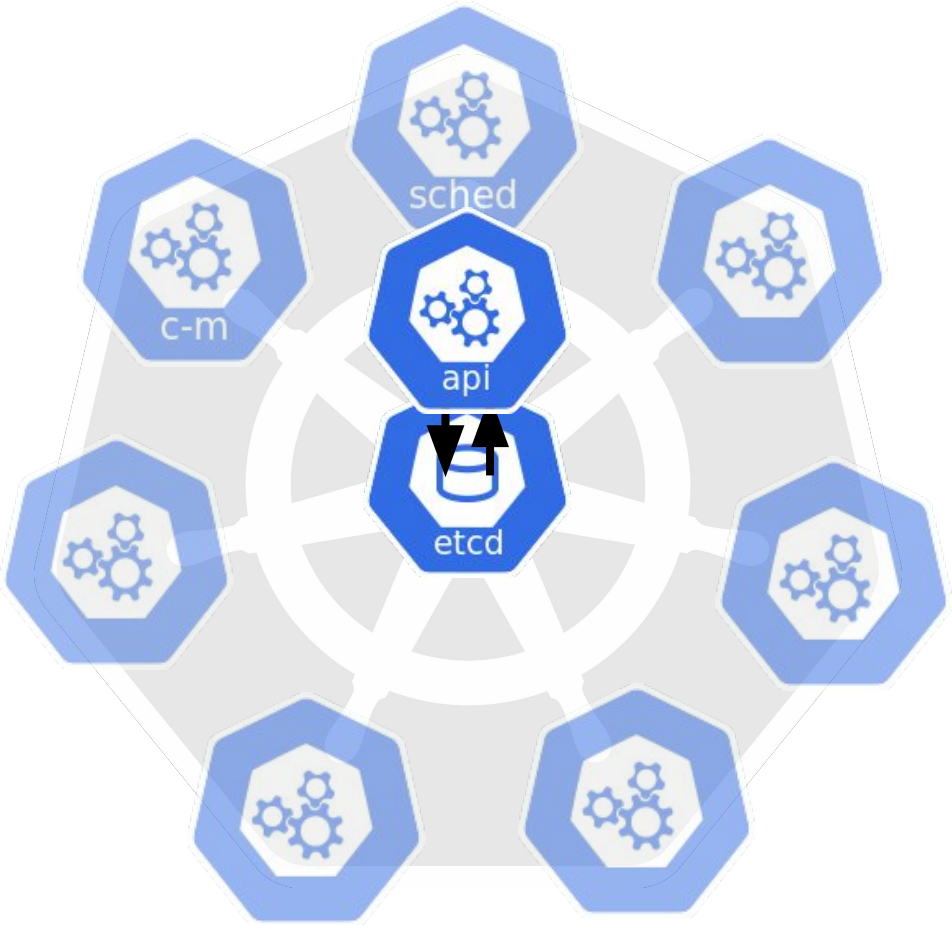
```
...
round_trippers.go:386] curl <some
headers>
'https://masterip/api/v1/components
tatuses?limit=500'
```

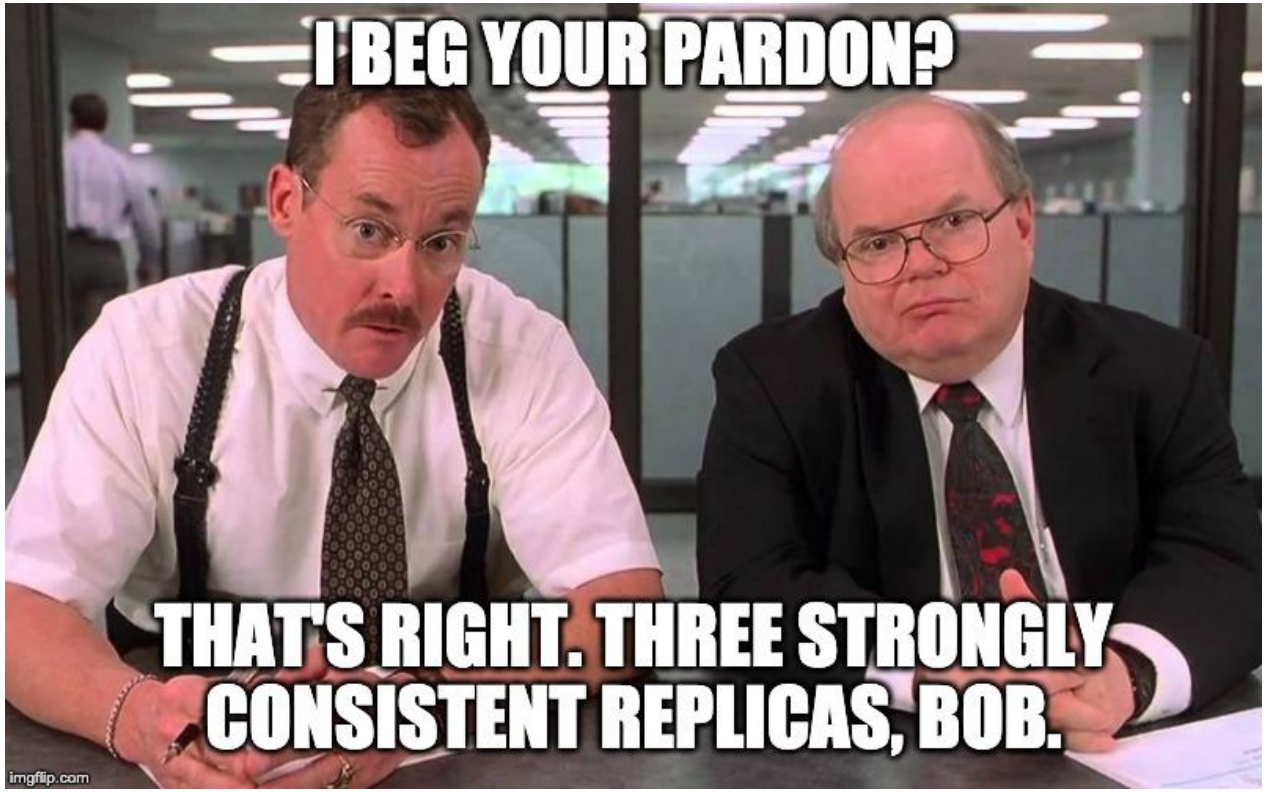
Kube-apiserver



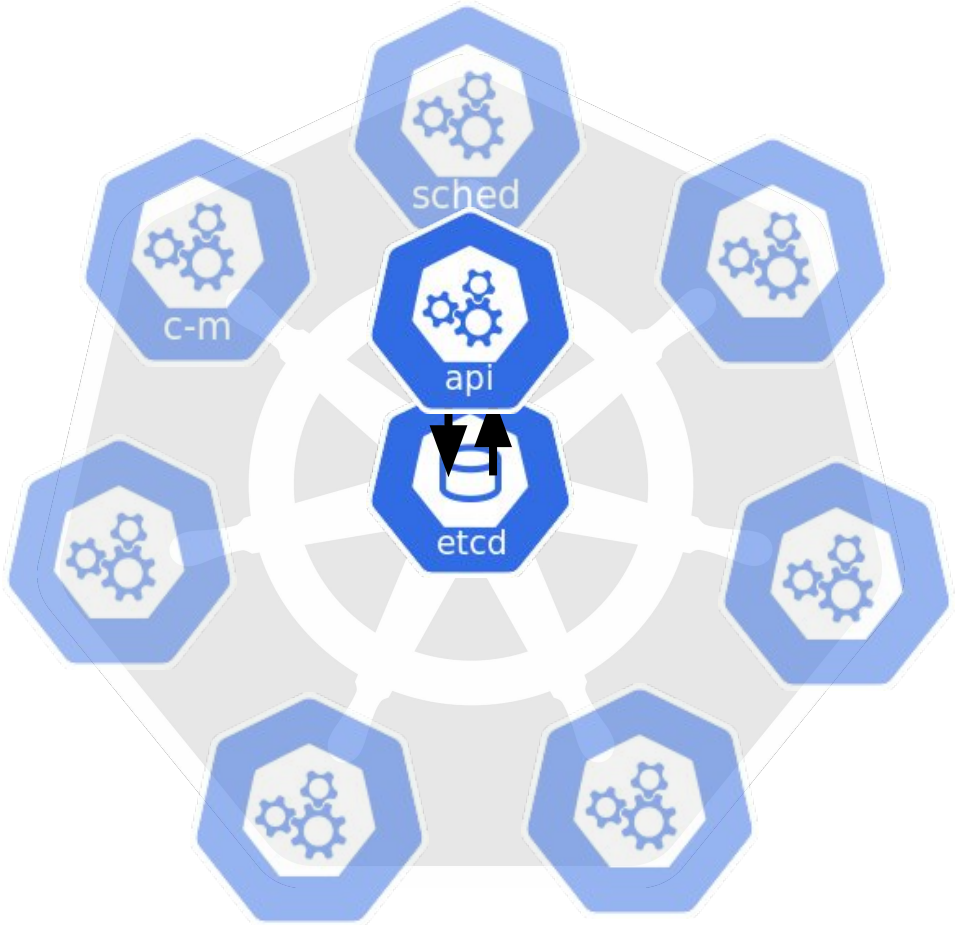
- kubectl <command> -v=9
- kube-apiserver.log
- /metrics
- health endpoints
  - localhost:8080/healthz?verbose
  - localhost:8080/livez (v1.16+)
  - localhost:8080/readyz (v1.16+)
- audit-logs

Etcd


sched
c-m
api
etcd


I BEG YOUR PARDON?
THAT'S RIGHT. THREE STRONGLY CONSISTENT REPLICAS, BOB.
imgflip.com

Etcd



- etcdctl
- auger
- /metrics
- /health
- etcd.log

Real-world Debugging

Kubelet Example

Problem:
Node is down

Kubelet Example

- **Obvious:** Prometheus scrape job is down
  `up{job="kube-nodes"} != 1`

- **Less obvious:** Grey failure indicated by unusually slow scrape time
  `scrape_duration_seconds{job="kube-nodes"} > 2`

Kube-apiserver Example

Problem:
Crash-looping kube-apiserver

Kube-apiserver Example

Detection Strategies:

1. Directly monitor kube-apiserver health endpoints
2. Alerting based off master kubelets ‘metrics/probes’

Kube-apiserver Example

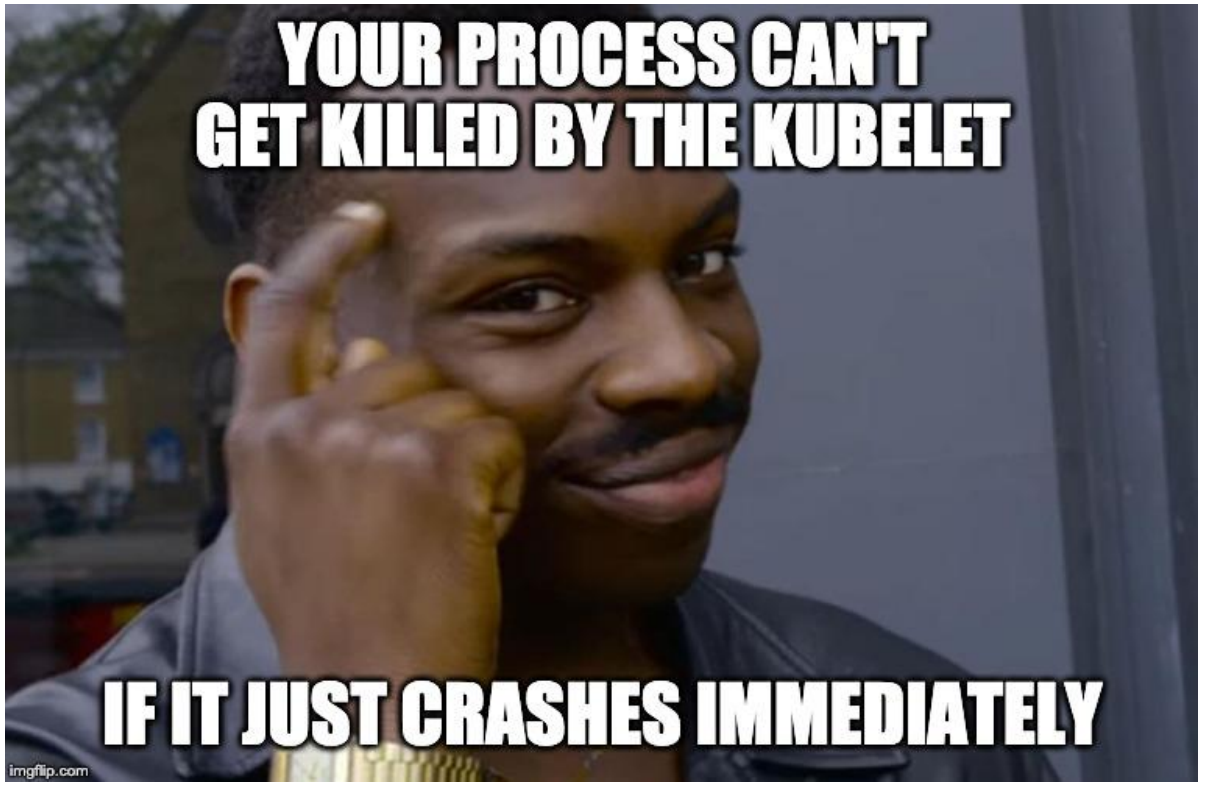

```
# output of kubelet's metrics/probes

# HELP prober_probe_total Cumulative number of a liveness or readiness probe for a container by result.
# TYPE prober_probe_total counter
prober_probe_total{container="kube-apiserver",probe_type="Liveness",result="failed"} 10
prober_probe_total{container="kube-apiserver",probe_type="Liveness",result="successful"} 26457
prober_probe_total{container="kube-apiserver",probe_type="Readiness",result="failed"} 16
prober_probe_total{container="kube-apiserver",probe_type="Readiness",result="successful"} 26458
```

Kube-apiserver Example

Possible reasons:

a. kubelet in repair mode
b. kubelet initiated crashloops

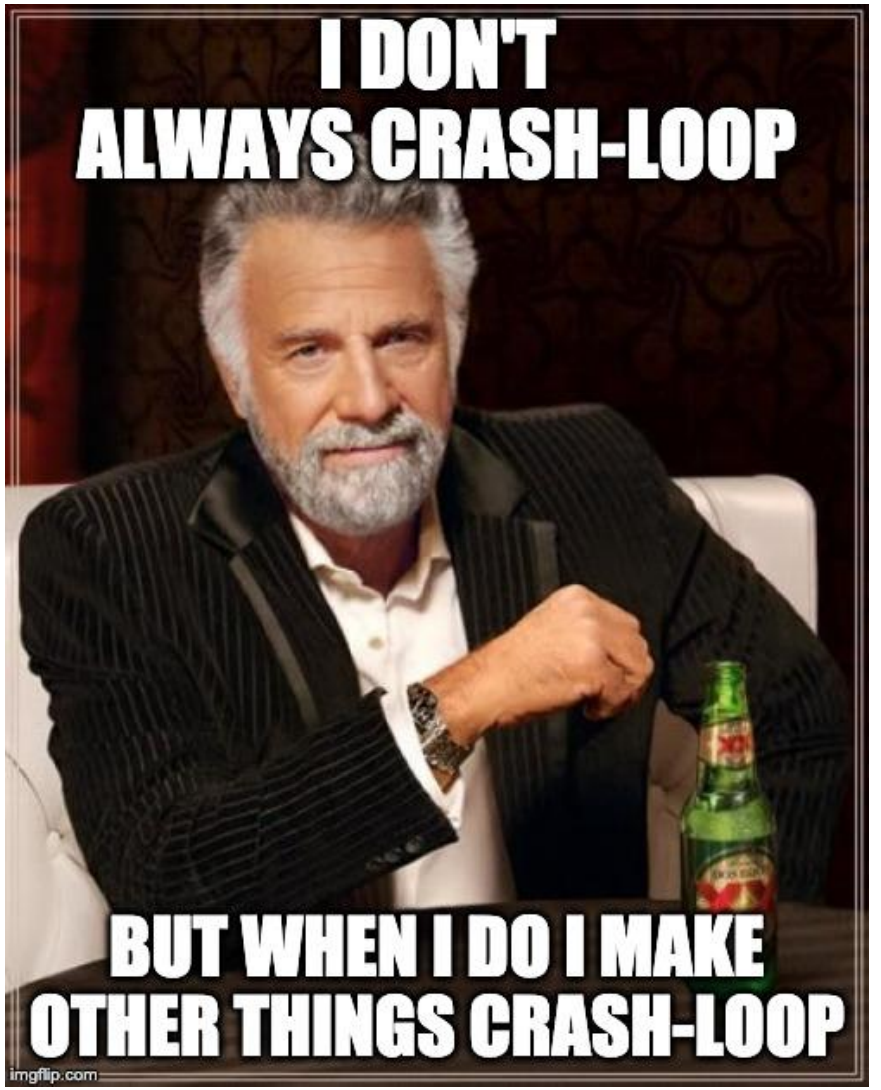


Kube-apiserver Example

kube-apiserver /healthz

```
$ curl localhost:8080/healthz?verbose

[+]ping ok
[+]log ok
[-]etcd failed: reason withheld
..... ok
[+]autoregister-completion ok
healthz check failed
```



Etcd Example


```
# HELP etcd_object_counts Number of stored objects at the time of last check split by kind.
# TYPE etcd_object_counts gauge
etcd_object_counts{resource="somecrd"} 1000000
```

Storage size limit

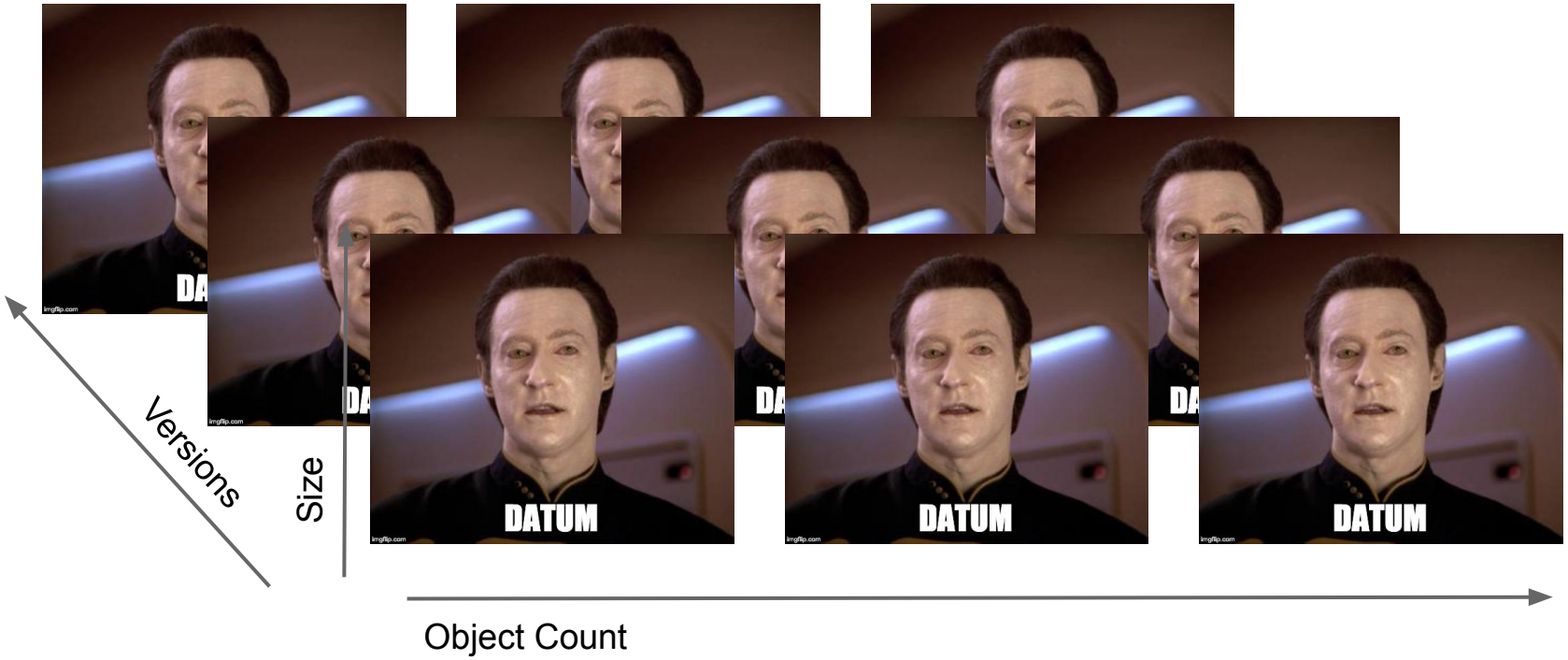
(https://github.com/etcd-io/etcd/blob/release-3.4/Documentation/dev-guide/limit.md**)**

The default storage size limit is 2GB, configurable with `--quota-backend-bytes` flag. 8GB is a suggested maximum size for normal environments and etcd warns at startup if the configured value exceeds it.

Etcd Example


```
etcd_object_counts{resource="somecrd"} 1
apiserver_request_count{resource="somecrd", verb="UPDATE"} 1200
```

Etcd Example




```
# (Revisited):
etcd_object_counts{resource="somecrd.io"} 1
apiserver_request_count{resource="somecrd.io", verb="UPDATE"} 1200
```

Etcd Example


```
$ kubectl get -ojson somecrd.io datum | wc -c
```

```
$ auger extract -f <dbfile> -k <key> | wc -c
```

Another kube-apiserver example

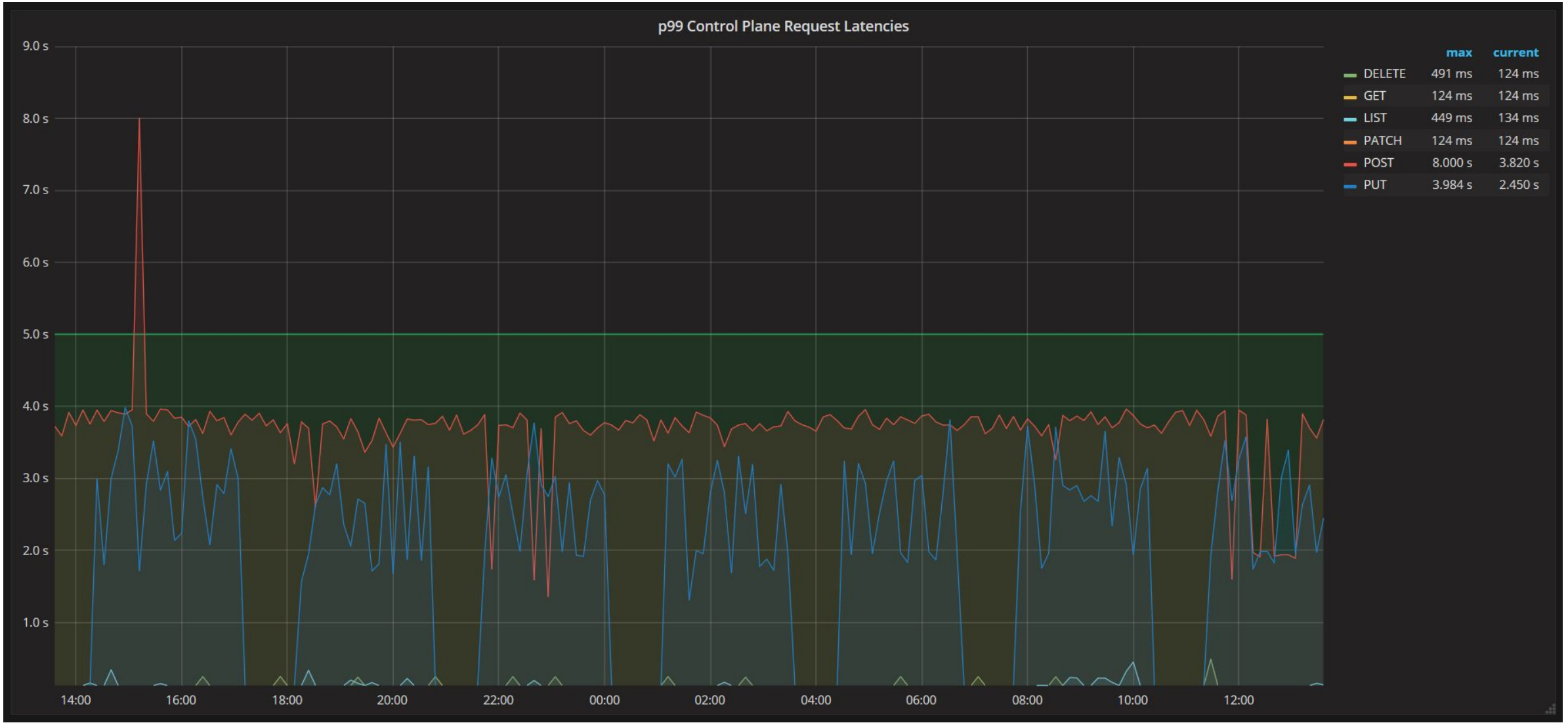
Problem:
API-servers are slow.

Another kube-apiserver example

- **Obvious:** p99 request latency is high

```
histogram_quantile(
  0.99,
  sum(rate(apiserver_request_latencies_bucket[1m]))
    by (le, verb)
)
```

Another kube-apiserver example



Another kube-apiserver example

- **Less obvious:** API server metrics prior to 1.14 release are limited to buckets between 125ms and 8s!

Another kube-apiserver example

Adjust buckets in apiserver request latency metrics

Browse files

master (#73638) v1.15.0-alpha.0 ... v1.14.0-alpha.3

wojtek-t committed on Feb 1 — 1 parent a3c14ec — commit d0508c7e872f60826d68c58c458cfd865554b486

Showing 1 changed file with **5 additions** and **2 deletions**.

Unified Split

7 staging/src/k8s.io/apiserver/pkg/endpoints/metrics/metrics.go — View file

```
8 +72,11 @@ var (
            prometheus.HistogramOpts{
                    Name: "apiserver_request_latency_seconds",
                    Help: "Response latency distribution in seconds for each verb, group, version, resource, subresource, scope and co
                    // Use buckets ranging from 125 ms to 8 seconds.
                    Buckets: prometheus.ExponentialBuckets(0.125, 2.0, 7),
                    // This metric is used for verifying api call latencies SLO,
                    // as well as tracking regressions in this aspects.
                    // Thus we customize buckets significantly, to empower both usecases.
                    Buckets: []float64{0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0,
                            1.25, 1.5, 1.75, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 40, 50, 60},
            },
            []string{"verb", "group", "version", "resource", "subresource", "scope", "component"},
    )
```

Metric Usability & SIG Instrumentation

Handling metric issues

- SIG Instrumentation needs to be able to fix metric bugs and issues
- Updating metrics between releases could break monitoring stacks
- Bad metrics can’t be disabled, requiring a full upgrade to address
- How can we coordinate developers to address this and responsibly communicate to end users?

Metrics Overhaul (1.14)

- Many **broken metrics** were identified
  - Labels did not match instrumentation guidelines, couldn't be joined
  - Wrong data types prevented aggregation
  - Units were not standardized
- SIG Instrumentation KEP: "Kubernetes Metrics Overhaul"
- **Fixes rolled out** in the 1.14 release

Metric Stability Framework

- SIG Instrumentation KEP: “Kubernetes Control Plane Metrics Stability”
- **Treat metrics as a proper API:** multi-release notice period for changes to stable metrics
- **Deprecation lifecycle:** slowly phase out obsolete metrics across releases before deletion
- **Enforcing Stability:** metrics migration, static analysis for stability validation, beta enforcements

Stability Metadata


```
var rpcDurations = metrics.NewSummary(
    metrics.SummaryOpts{
      Name:           "rpc_durations_seconds",
      Help:           "RPC latency distributions.",
      StabilityLevel: metrics.STABLE,
      DeprecatedVersion: "1.15",
    },
)
```

More to come!

- Stable metric criteria and promotion
- Runtime flags for disabling individual metrics
- Distributed tracing
- Structured logs
- More metric improvements!

Learn more: **SIG Instrumentation Intro // Deep Dive**

Today @ 4:25pm in 6E // Tomorrow @ 3:20pm in 6D


KubeCon
CloudNativeCon
North America 2019
Questions?


KubeCon
CloudNativeCon
North America 2019

Image Citations

- Slide 16 : Title*: LIveness Probe Meme;* Site: Meme Generator; *URL:* https://imgflip.com/memegenerator; Date: 11/15/19; Publisher: imgflip
- Slide 18 : Title*:Count on me Meme;* Site: Meme Generator; *URL:* https://imgflip.com/memegenerator; Date: 11/15/19; Publisher: imgflip
- Slide 19 : Title*: Only one health check Meme;* Site: Meme Generator; *URL:* https://imgflip.com/memegenerator; Date: 11/15/19; Publisher: imgflip
- Slide 20 : Title*: Talk to the hand Meme;* Site: Meme Generator; *URL:* https://imgflip.com/memegenerator; Date: 11/15/19; Publisher: imgflip
- Slide 23 : Title*: Etcd Meme;* Site: Meme Generator; *URL:* https://imgflip.com/memegenerator; Date: 11/15/19; Publisher: imgflip
- Slide 26 : Title*: Bambi Meme;* Site: Meme Generator; *URL:* https://imgflip.com/memegenerator; Date: 11/15/19; Publisher: imgflip
- Slide 28: Title*: Oh nos Meme;* Site: Meme Generator; *URL:* https://imgflip.com/memegenerator; Date: 11/15/19; Publisher: imgflip
- Slide 31 : Title*: Can't crash a crashed process Meme;* Site: Meme Generator; *URL:* https://imgflip.com/memegenerator; Date: 11/15/19; Publisher: imgflip
- Slide 32: Title*: Causes other crashloops Meme;* Site: Meme Generator; *URL:* https://imgflip.com/memegenerator; Date: 11/15/19; Publisher: imgflip
- Slide 35 : Title*: Datum Meme;* Site: Meme Generator; *URL:* https://imgflip.com/memegenerator; Date: 11/15/19; Publisher: imgflip
- Slide 37 : Title*: Silly latency metric Meme;* Site: Meme Generator; *URL:* https://imgflip.com/memegenerator; Date: 11/15/19; Publisher: imgflip