KubeCon | CloudNativeCon

North America 2019

KubeCon | CloudNativeCon

North America 2019

SIG Instrumentation - Deep Dive

Han Kang and David Ashpole

What we are going to cover

1. Metrics Stability Framework
    a. Historical context for understanding the problem
    b. How we converged on the design
    c. Future plans
2. Tracing in Kubernetes

SIG Instrumentation (some history!)

SIG Instrumentation

SIG Charter (in-scope) http://bit.ly/sig-inst-charter:

- “***Owns best practices*** for cluster observability through metrics and logging across all Kubernetes components and development of components required for all Kubernetes clusters”
- “SIG-Instrumentation revolves around the ***process*** of instrumenting and exposing observability signals.”
- “***Guidance*** for instrumentation in order to ensure consistent and high quality instrumentation of core Kubernetes components.“
- “Creating, adding and maintaining the Kubernetes instrumentation ***guidelines***.”
- “Reviewing any instrumentation related changes and additions.”

SIG Instrumentation

SIG Charter (out-of-scope) http://bit.ly/sig-inst-charter:

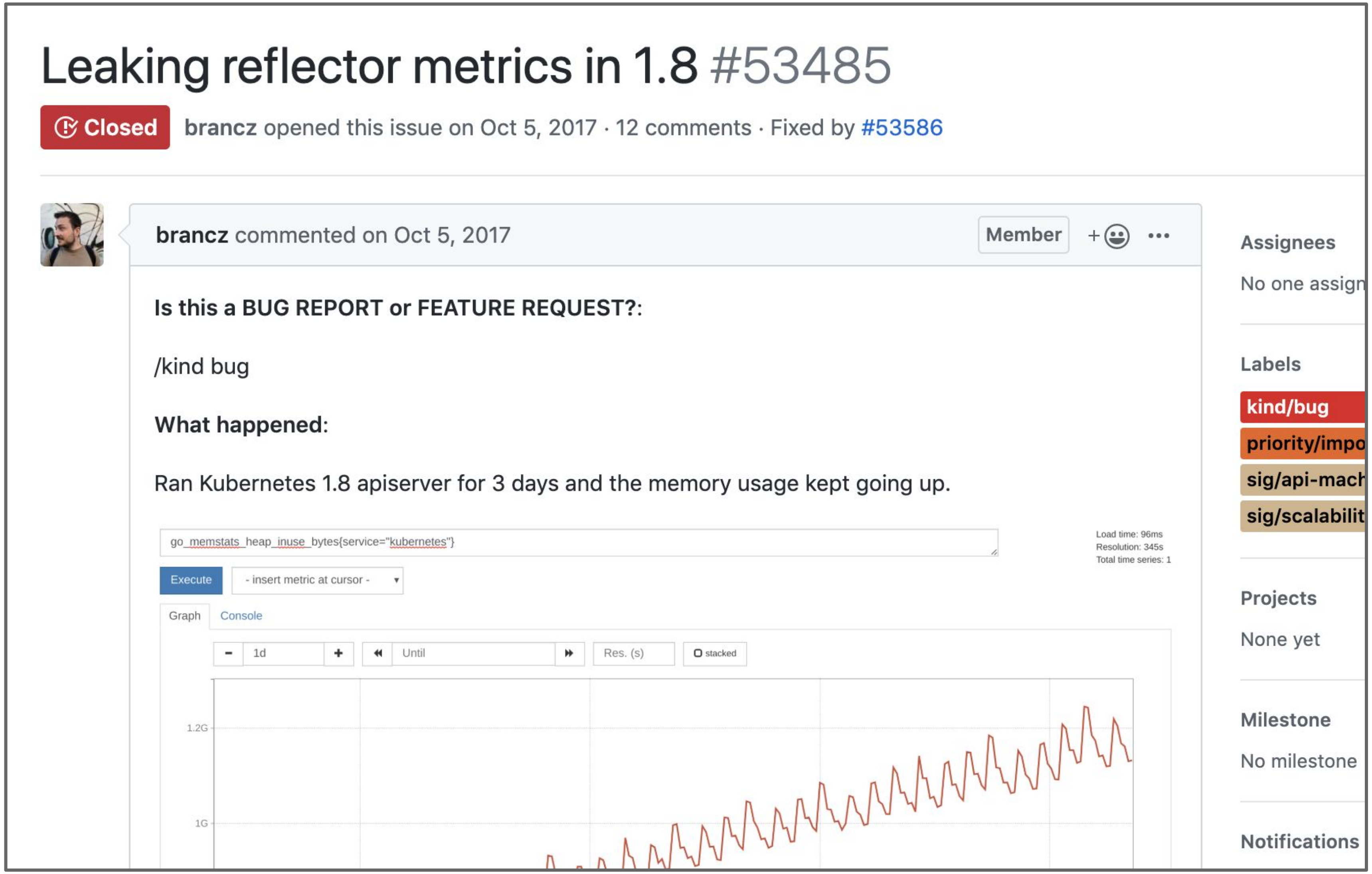
- “Processing of signals. For example ingesting metrics, logs, events into external systems.”
- “Dictating what states must result in an alert. Suggestions or opt-in alerts may be in scope.”
- “The act of instrumenting components not owned by SIG-Instrumentation is out of scope”

SIG Charter

To Recap:

- Providing guidelines and best practices for instrumentation and observability.
- Owning the ***process*** of instrumenting and exposing observability signals.
- Reviewing instrumentation code.
- NOT owning individual metrics or individual logs

Bad Metrics



Leaking reflector metrics in 1.8 #53485

Closed **brancz** opened this issue on Oct 5, 2017 · 12 comments · Fixed by #53586

brancz commented on Oct 5, 2017 — Member

Is this a BUG REPORT or FEATURE REQUEST?:

/kind bug

What happened:

Ran Kubernetes 1.8 apiserver for 3 days and the memory usage kept going up.

```
go_memstats_heap_inuse_bytes{service="kubernetes"}
```

Load time: 96ms
Resolution: 345s
Total time series: 1

Assignees
No one assign

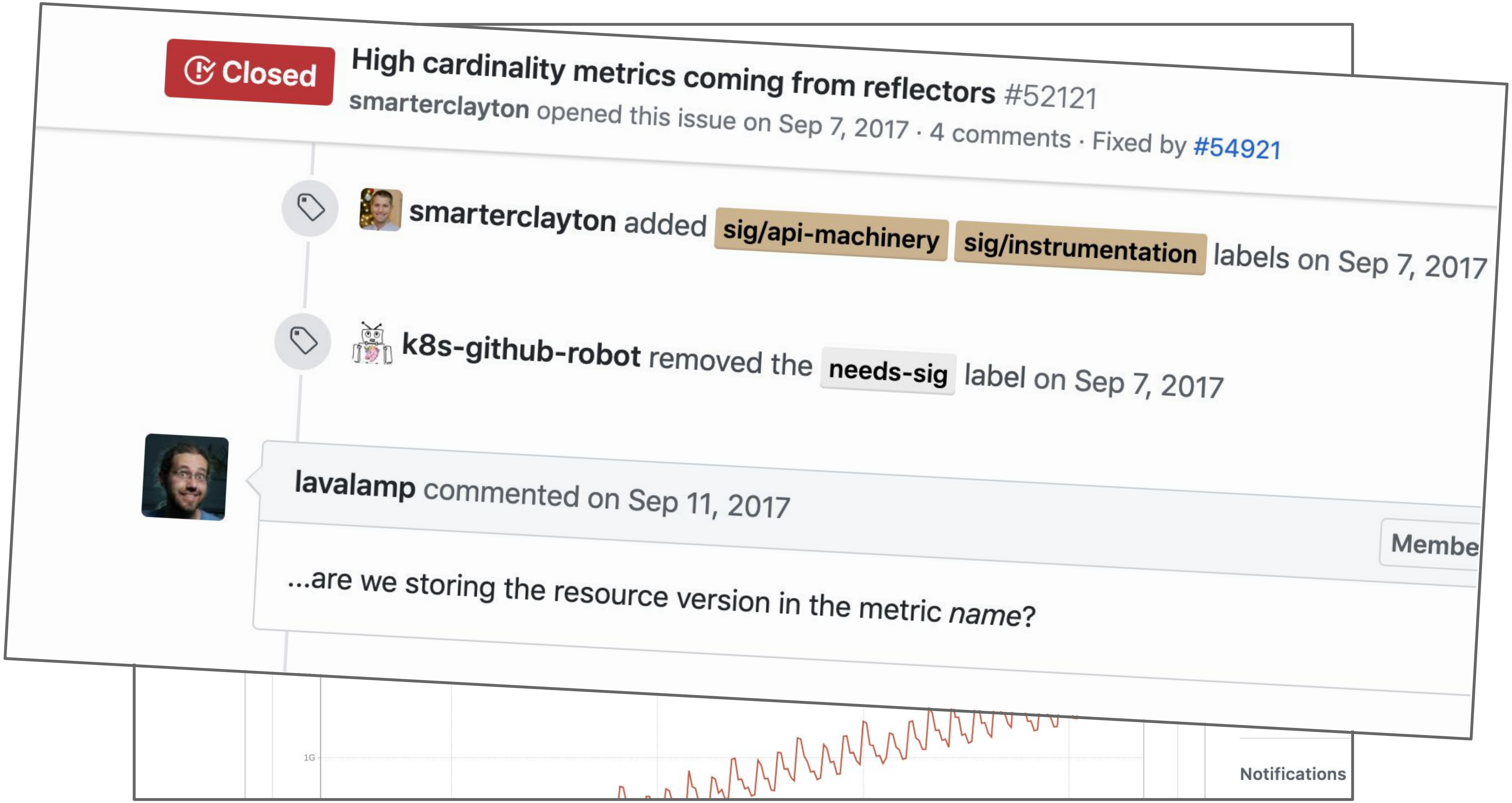
Labels
kind/bug
priority/impo
sig/api-mach
sig/scalabilit

Projects
None yet

Milestone
No milestone

Notifications

Bad Metrics



Closed High cardinality metrics coming from reflectors #52121

smarterclayton opened this issue on Sep 7, 2017 · 4 comments · Fixed by #54921

smarterclayton added sig/api-machinery sig/instrumentation labels on Sep 7, 2017

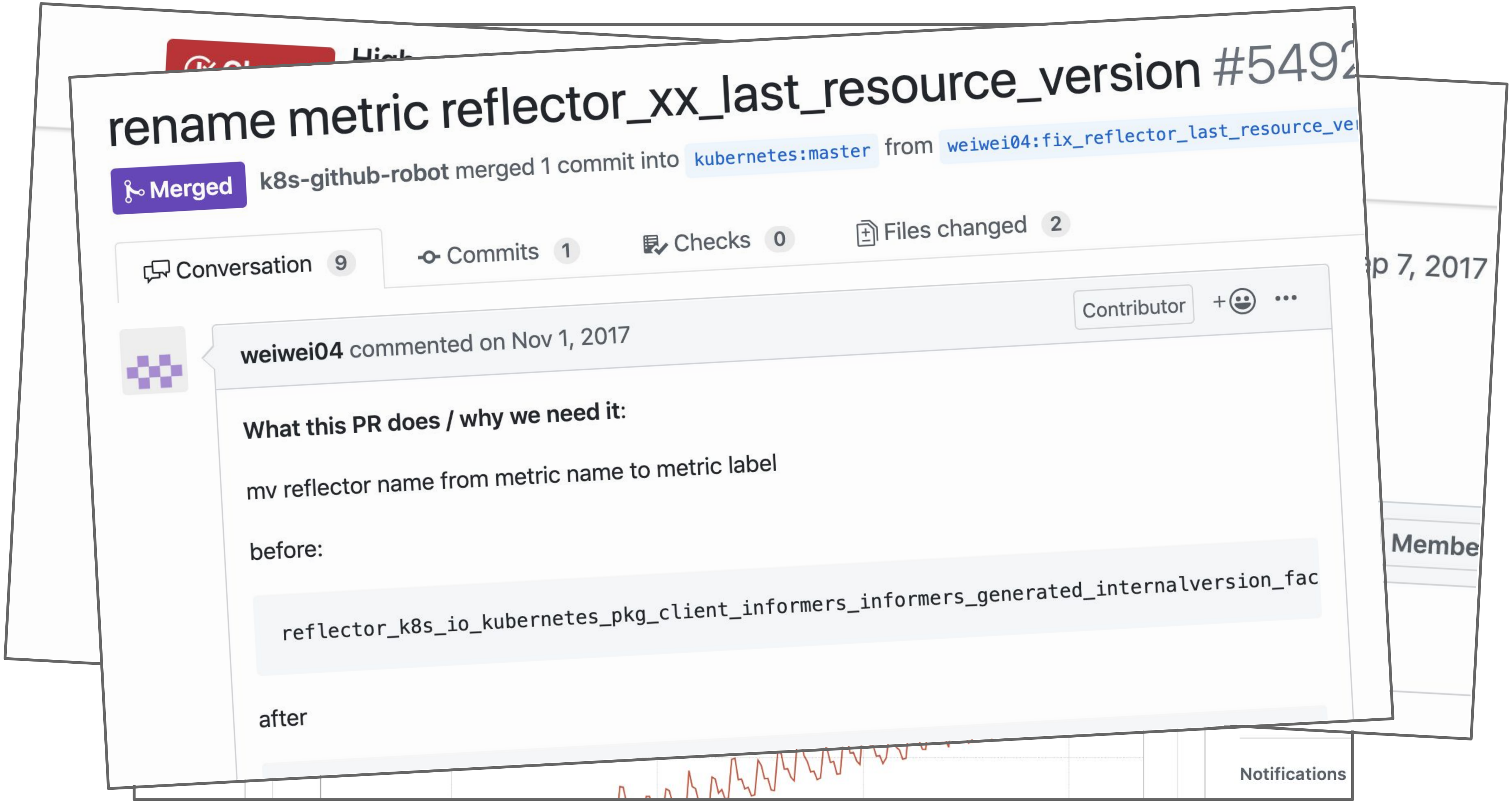
k8s-github-robot removed the needs-sig label on Sep 7, 2017

lavalamp commented on Sep 11, 2017

Membe

...are we storing the resource version in the metric *name*?

Bad Metrics



rename metric reflector_xx_last_resource_version #5492

Merged k8s-github-robot merged 1 commit into `kubernetes:master` from `weiwei04:fix_reflector_last_resource_ve`

weiwei04 commented on Nov 1, 2017

What this PR does / why we need it:

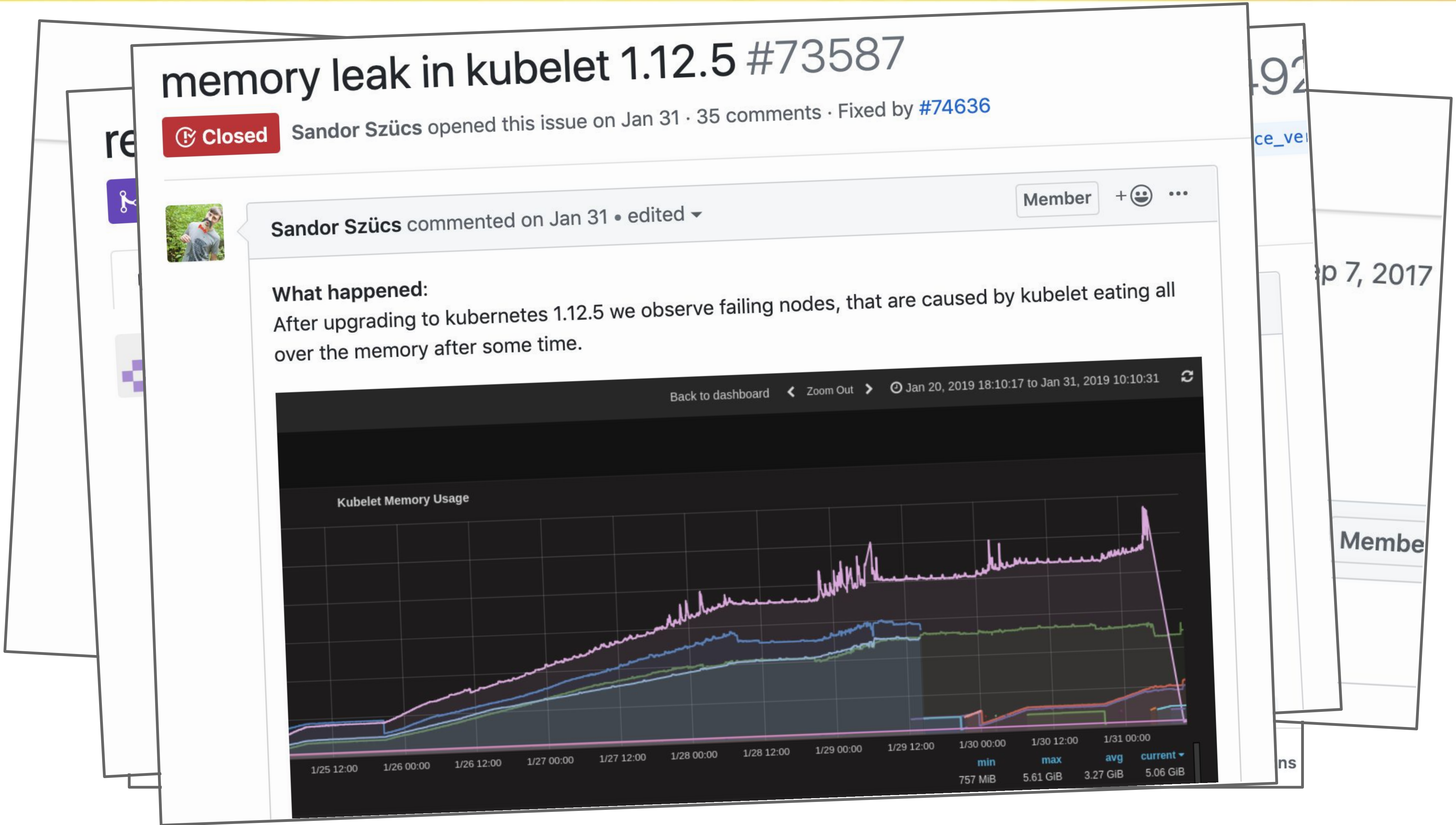
mv reflector name from metric name to metric label

before:

```
reflector_k8s_io_kubernetes_pkg_client_informers_informers_generated_internalversion_fac
```

after

Bad Metrics



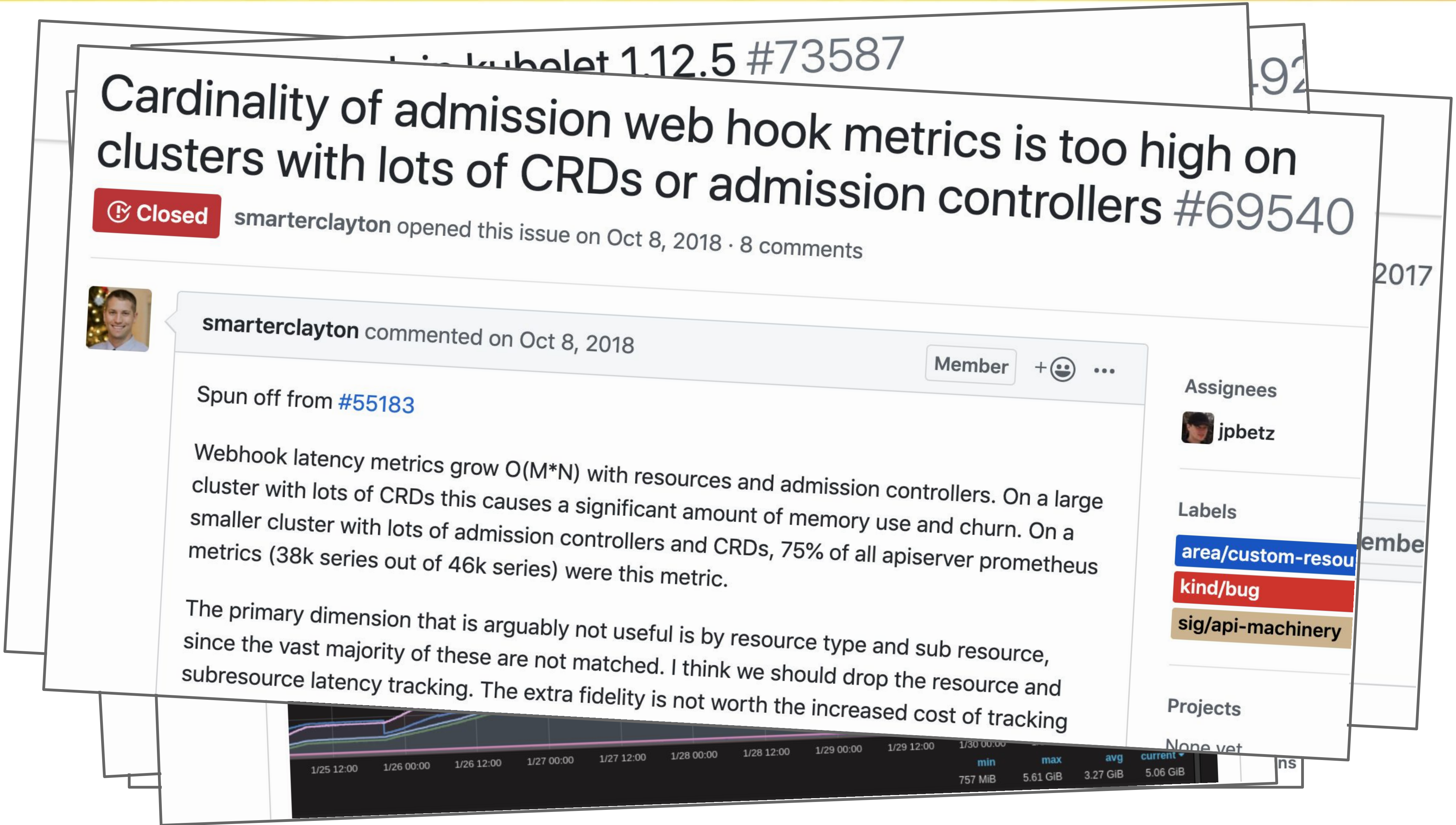
memory leak in kubelet 1.12.5 #73587

Closed Sandor Szücs opened this issue on Jan 31 · 35 comments · Fixed by #74636

Sandor Szücs commented on Jan 31 • edited

What happened:
After upgrading to kubernetes 1.12.5 we observe failing nodes, that are caused by kubelet eating all over the memory after some time.

Bad Metrics



...kubelet 1.12.5 #73587

Cardinality of admission web hook metrics is too high on clusters with lots of CRDs or admission controllers #69540

Closed **smarterclayton** opened this issue on Oct 8, 2018 · 8 comments

smarterclayton commented on Oct 8, 2018 — Member

Spun off from #55183

Webhook latency metrics grow O(M*N) with resources and admission controllers. On a large cluster with lots of CRDs this causes a significant amount of memory use and churn. On a smaller cluster with lots of admission controllers and CRDs, 75% of all apiserver prometheus metrics (38k series out of 46k series) were this metric.

The primary dimension that is arguably not useful is by resource type and sub resource, since the vast majority of these are not matched. I think we should drop the resource and subresource latency tracking. The extra fidelity is not worth the increased cost of tracking

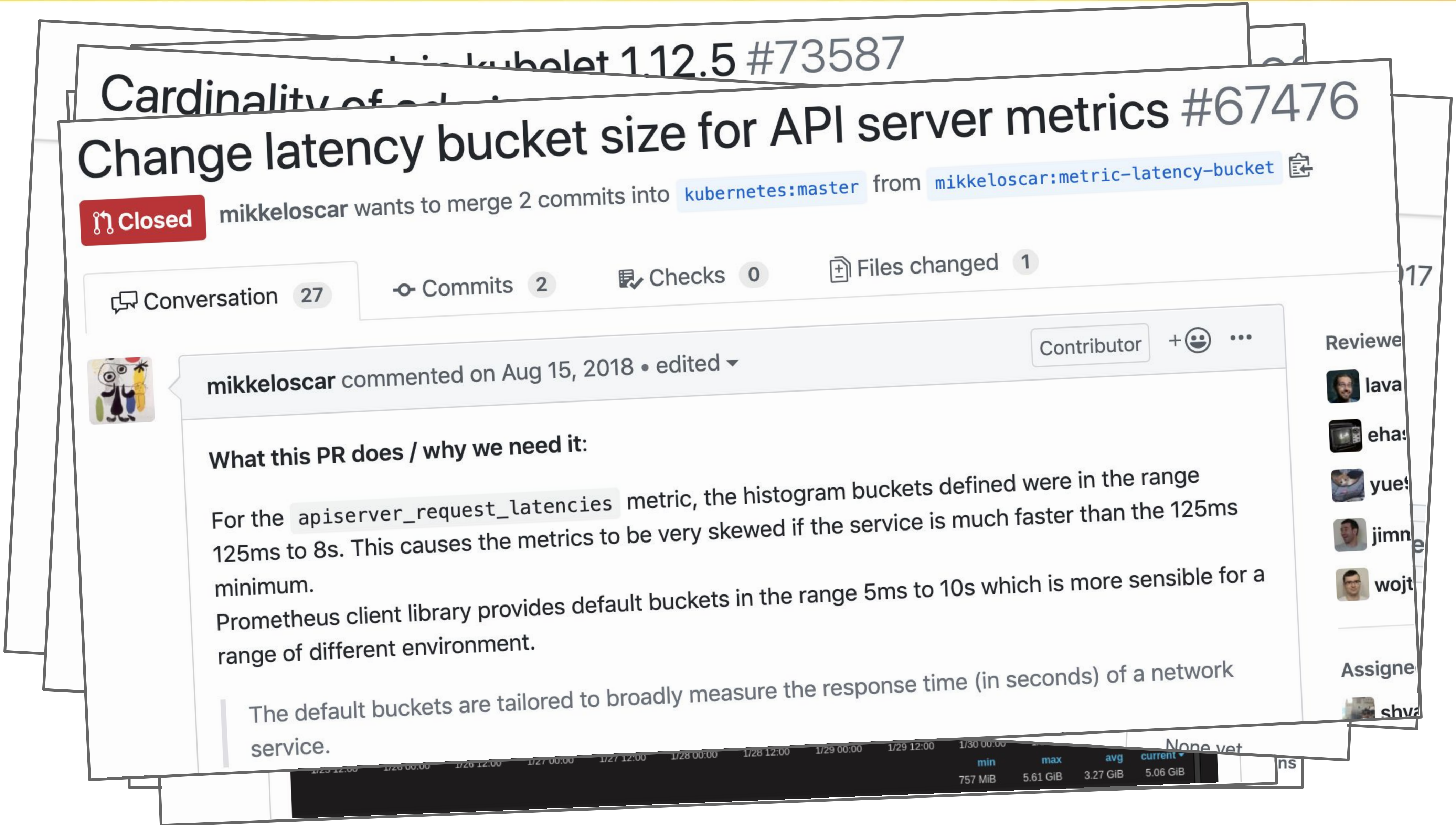
Assignees: jpbetz

Labels: area/custom-resou · kind/bug · sig/api-machinery

Projects: None yet

Bad Metrics

KubeCon | CloudNativeCon North America 2019



kubelet 1.12.5 #73587

Cardinality of ad

Change latency bucket size for API server metrics #67476

Closed mikkeloscar wants to merge 2 commits into `kubernetes:master` from `mikkeloscar:metric-latency-bucket`

Conversation 27 | Commits 2 | Checks 0 | Files changed 1

mikkeloscar commented on Aug 15, 2018 • edited — Contributor

What this PR does / why we need it:

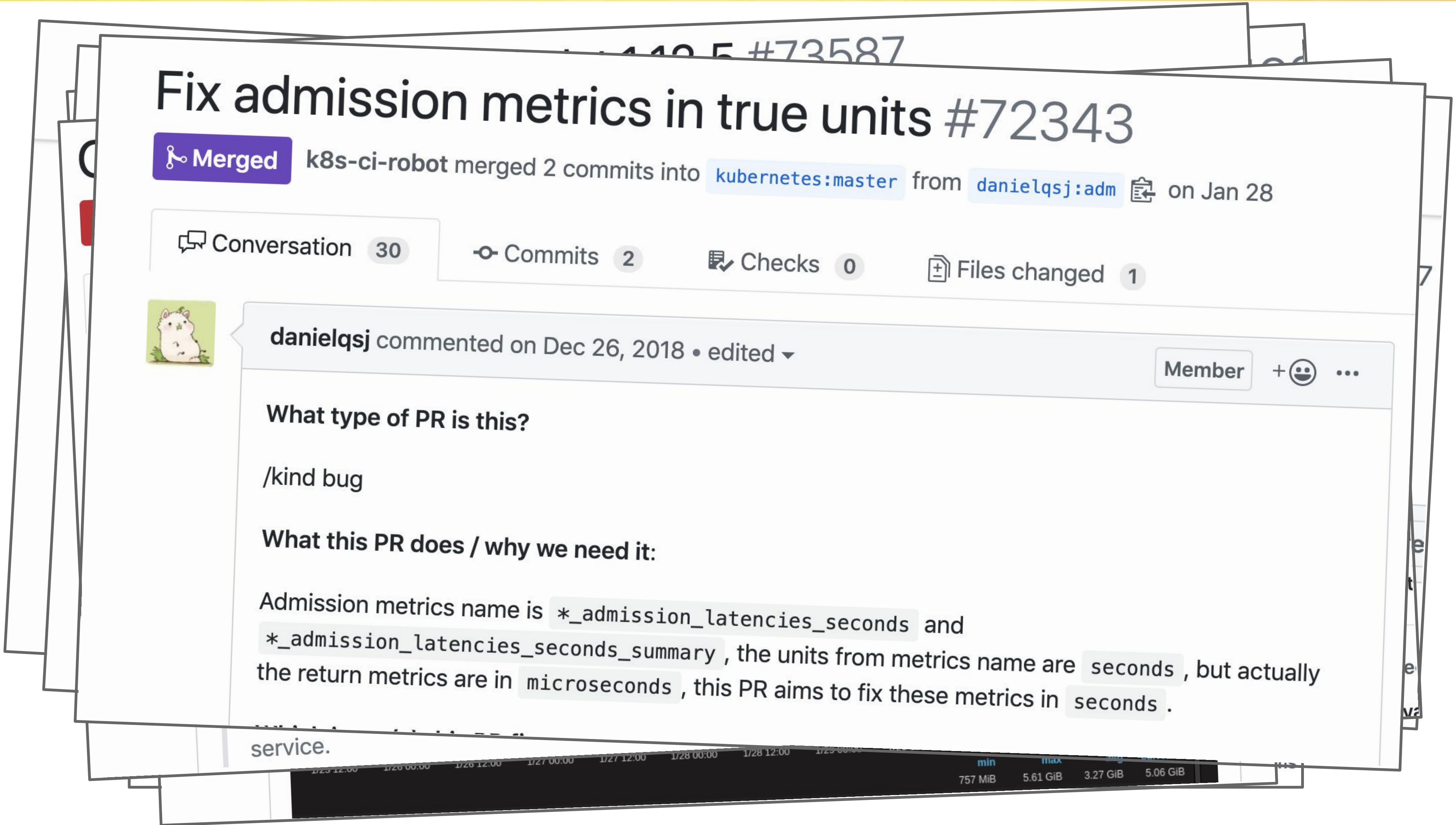
For the `apiserver_request_latencies` metric, the histogram buckets defined were in the range 125ms to 8s. This causes the metrics to be very skewed if the service is much faster than the 125ms minimum.
Prometheus client library provides default buckets in the range 5ms to 10s which is more sensible for a range of different environment.

> The default buckets are tailored to broadly measure the response time (in seconds) of a network service.

Reviewers: lava, ehas, yues, jimn, wojt

Assignees: shya

Bad Metrics



#73587

Fix admission metrics in true units #72343

Merged k8s-ci-robot merged 2 commits into `kubernetes:master` from `danielqsj:adm` on Jan 28

danielqsj commented on Dec 26, 2018 • edited

What type of PR is this?

/kind bug

What this PR does / why we need it:

Admission metrics name is `*_admission_latencies_seconds` and `*_admission_latencies_seconds_summary`, the units from metrics name are `seconds`, but actually the return metrics are in `microseconds`, this PR aims to fix these metrics in `seconds`.

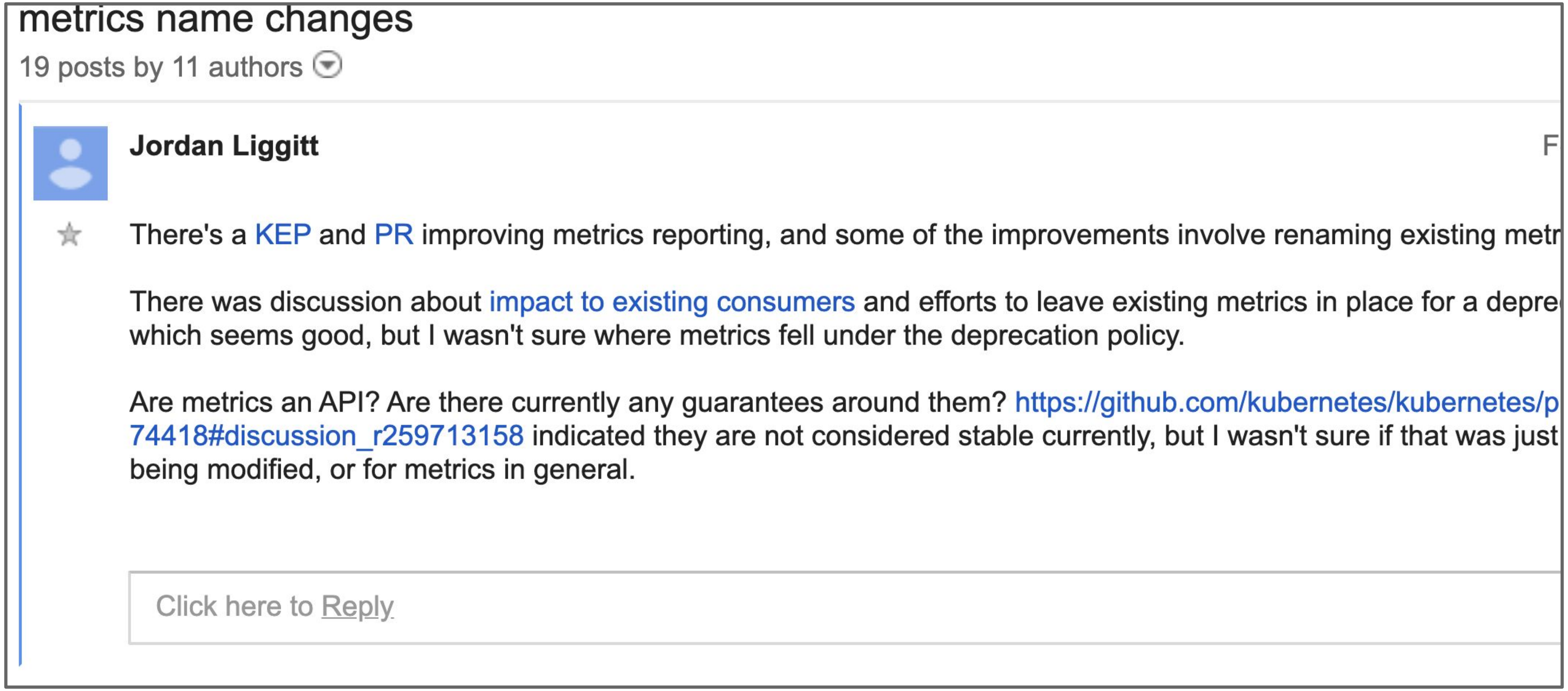
service.

Fixing Existing Metrics

Metrics Overhaul KEP (http://bit.ly/metrics-overhaul):

- bring things in-line with metrics guidelines
- fix known existing metrics issues

Metrics as an API



metrics name changes

19 posts by 11 authors

Jordan Liggitt

There's a KEP and PR improving metrics reporting, and some of the improvements involve renaming existing metr

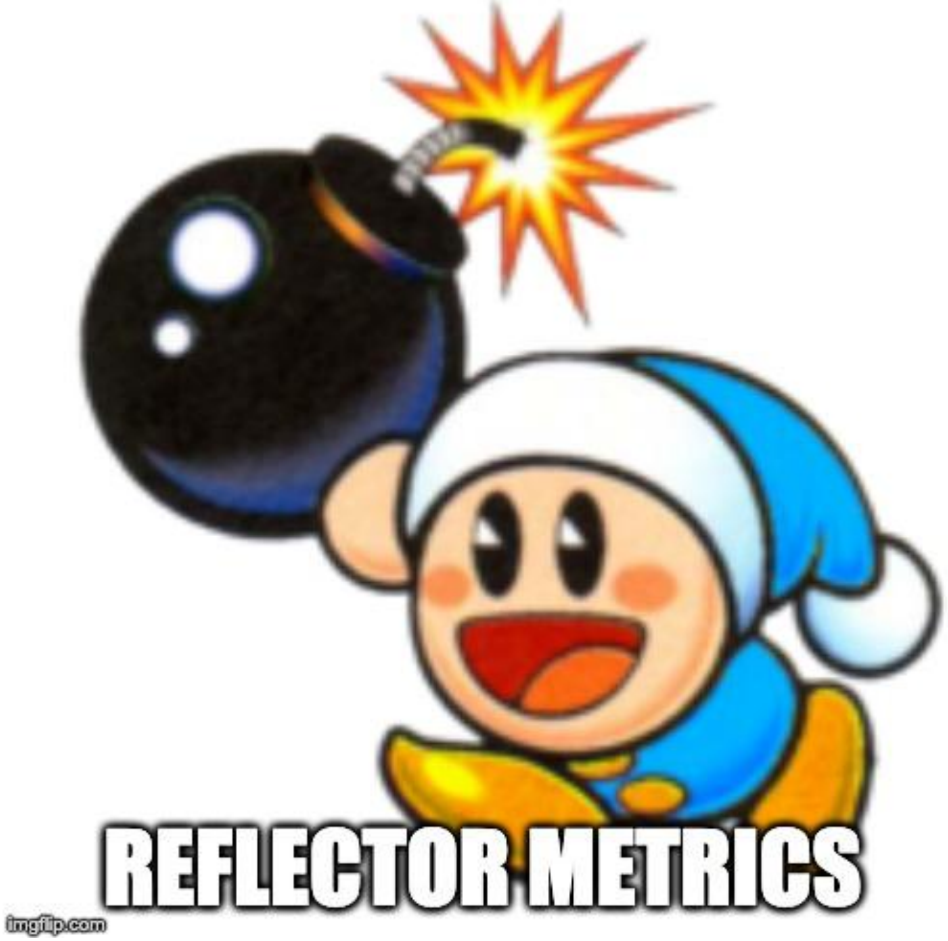
There was discussion about impact to existing consumers and efforts to leave existing metrics in place for a depre which seems good, but I wasn't sure where metrics fell under the deprecation policy.

Are metrics an API? Are there currently any guarantees around them? https://github.com/kubernetes/kubernetes/p 74418#discussion_r259713158 indicated they are not considered stable currently, but I wasn't sure if that was just being modified, or for metrics in general.

Issues

Conflating (and possibly contradictory) concerns:

- Metrics as an API
- Fixing broken metrics
- What do we do if a metric explodes?



Metrics Stability Framework

* http://bit.ly/stability-kep

Alternatives considered

- Versioning the metrics endpoint directly
  - /metrics/v1alpha1 -> /metrics/v1beta -> metrics/v1
- Documenting a set of metrics which are considered “API stable”

Goals

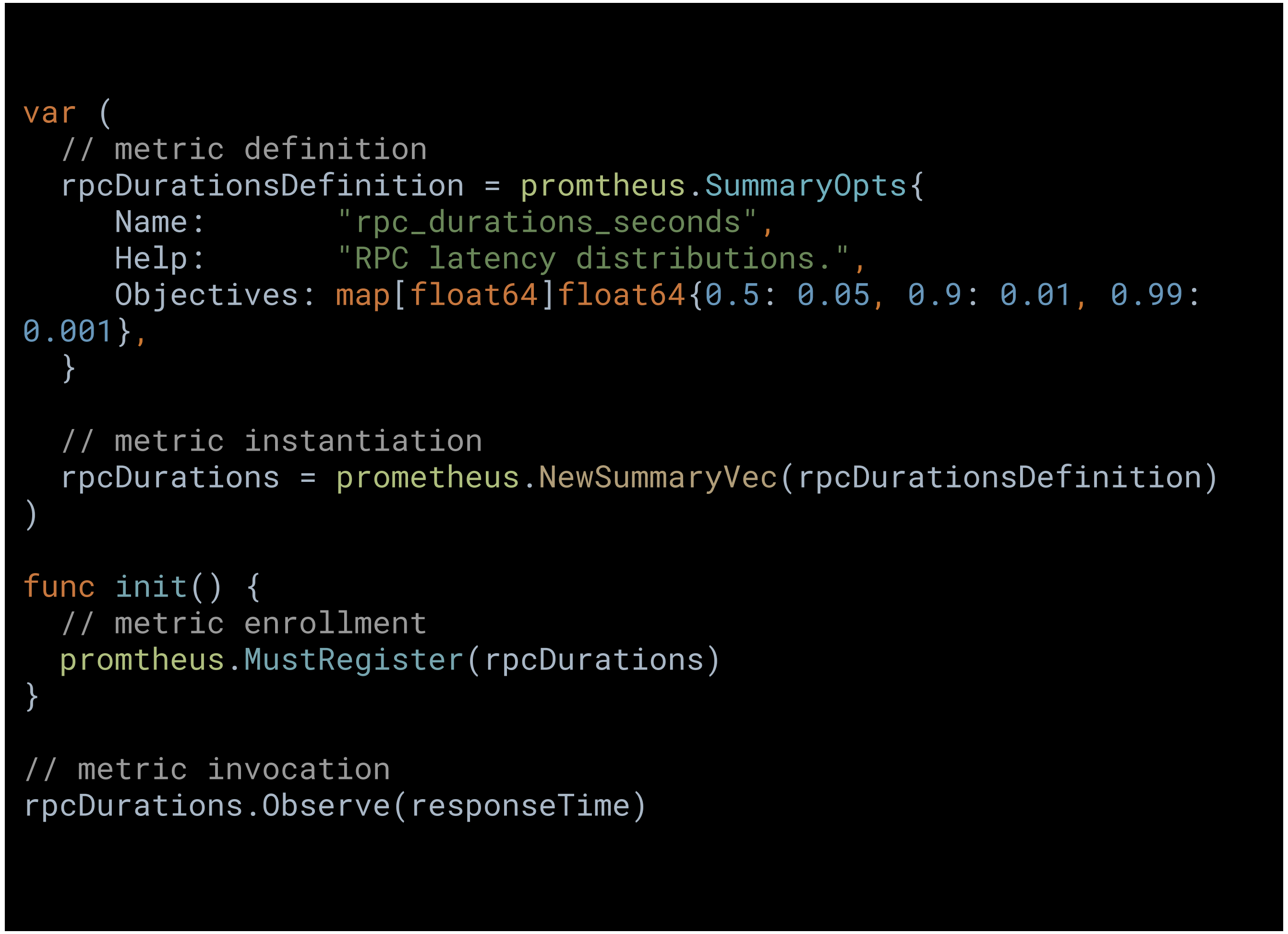
- Provide a framework to expressing metric stability guarantees
- Provide automation around stability levels
- (stretch goal) Provide a mechanism to centralize instrumentation related code and instrumentation processes

Quasi-Versioning Strategy

- Metrics can be individually ‘versioned’
- Not exactly a version
- Stability metadata for metrics

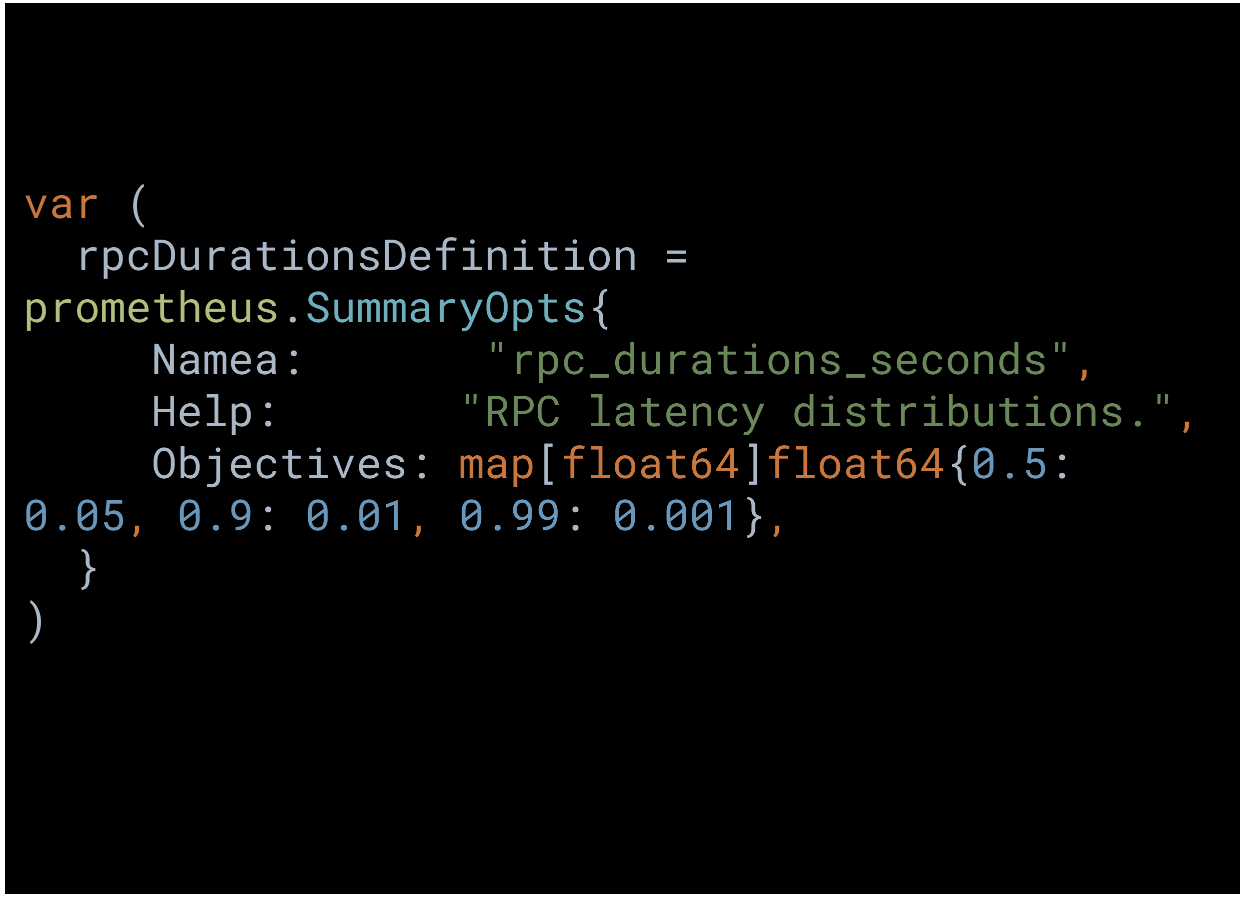
Prometheus Metric Lifecycle

- Metric Definition
- Metric Instantiation
- Metric Enrollment (to a registry)

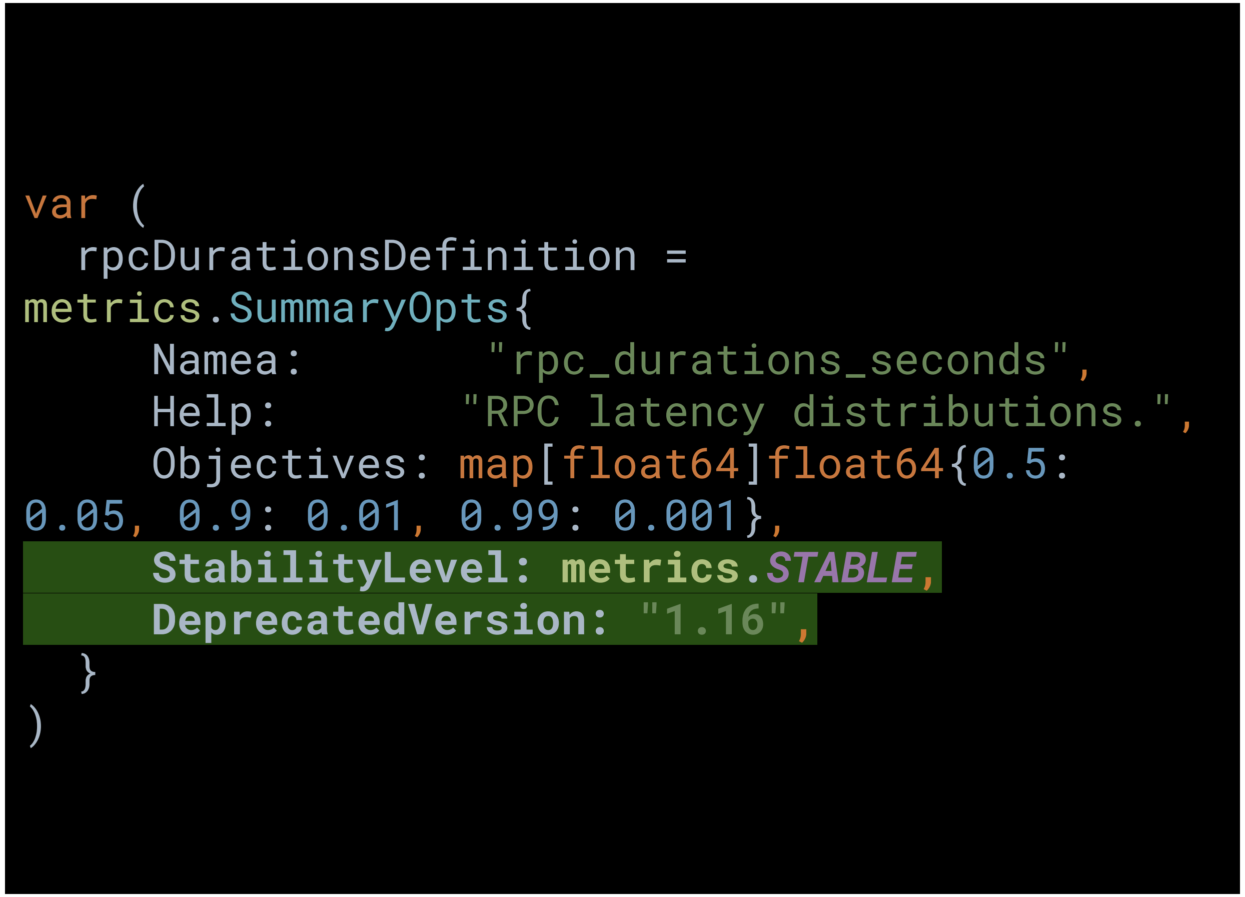


```
var (
  // metric definition
  rpcDurationsDefinition = promtheus.SummaryOpts{
     Name:        "rpc_durations_seconds",
     Help:        "RPC latency distributions.",
     Objectives: map[float64]float64{0.5: 0.05, 0.9: 0.01, 0.99:
0.001},
  }

  // metric instantiation
  rpcDurations = prometheus.NewSummaryVec(rpcDurationsDefinition)
)

func init() {
  // metric enrollment
  promtheus.MustRegister(rpcDurations)
}

// metric invocation
rpcDurations.Observe(responseTime)
```

Hijacking Metric Definition

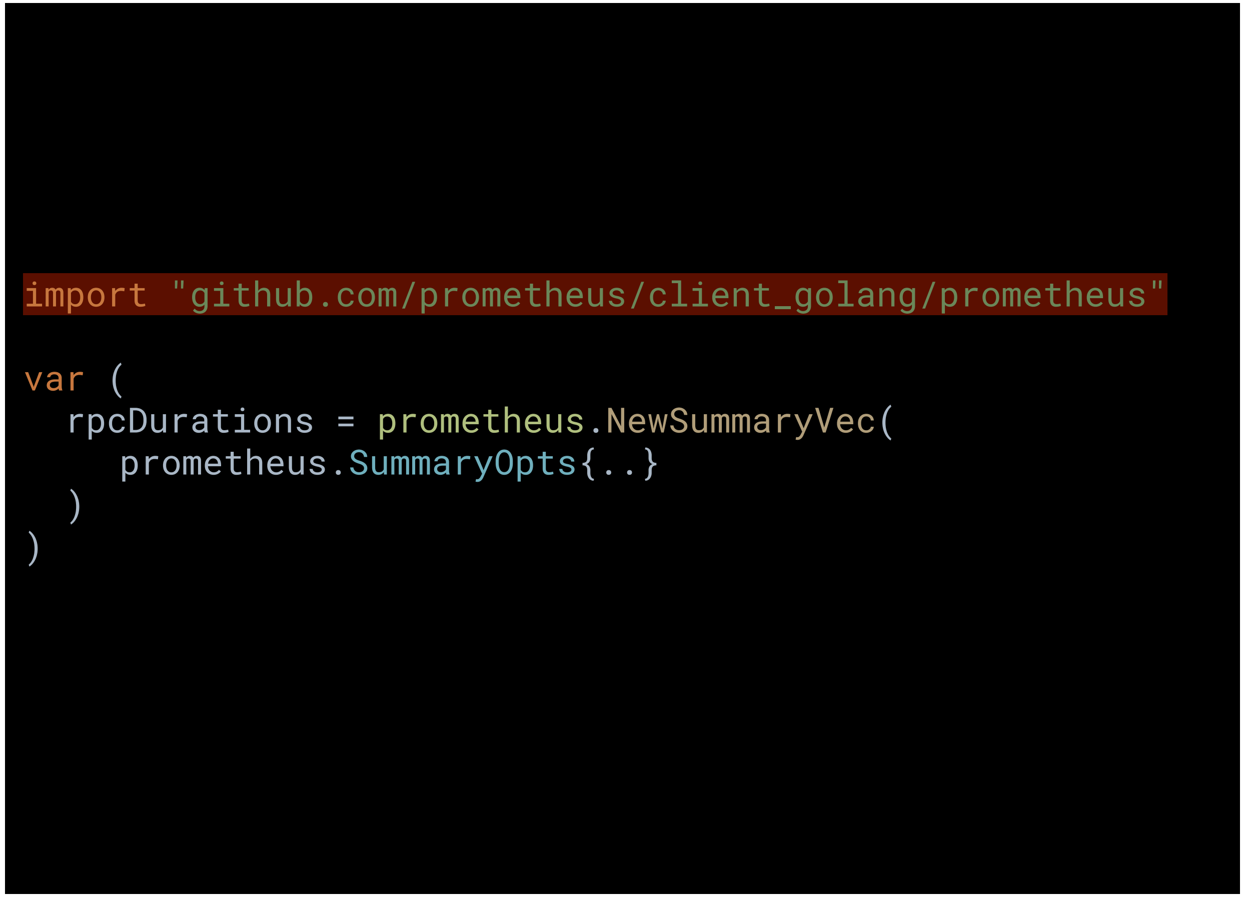


```
var (
  rpcDurationsDefinition =
prometheus.SummaryOpts{
     Namea:       "rpc_durations_seconds",
     Help:       "RPC latency distributions.",
     Objectives: map[float64]float64{0.5:
0.05, 0.9: 0.01, 0.99: 0.001},
  }
)
```

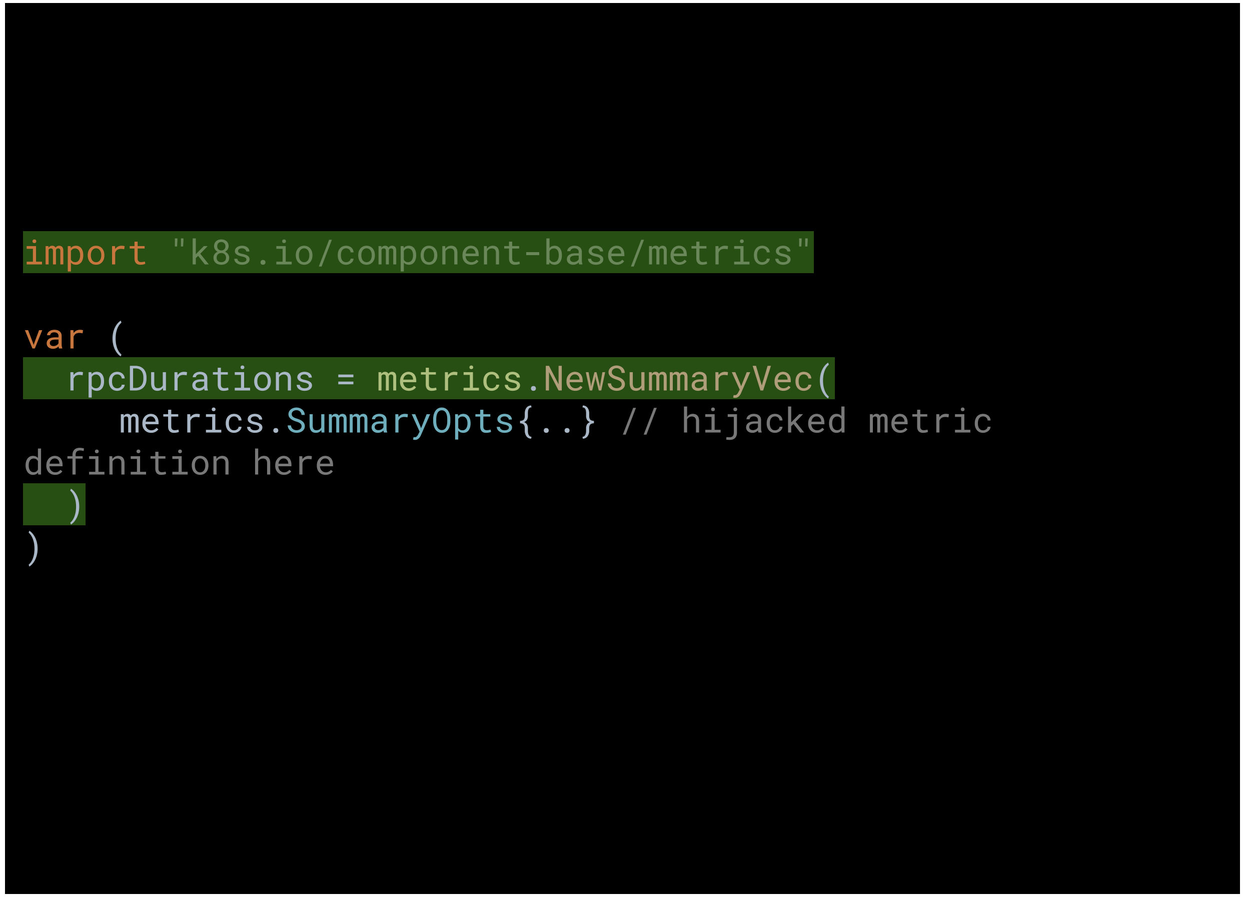


```
var (
  rpcDurationsDefinition =
metrics.SummaryOpts{
     Namea:       "rpc_durations_seconds",
     Help:       "RPC latency distributions.",
     Objectives: map[float64]float64{0.5:
0.05, 0.9: 0.01, 0.99: 0.001},
     StabilityLevel: metrics.STABLE,
     DeprecatedVersion: "1.16",
  }
)
```

Hijacking Metric Instantiation

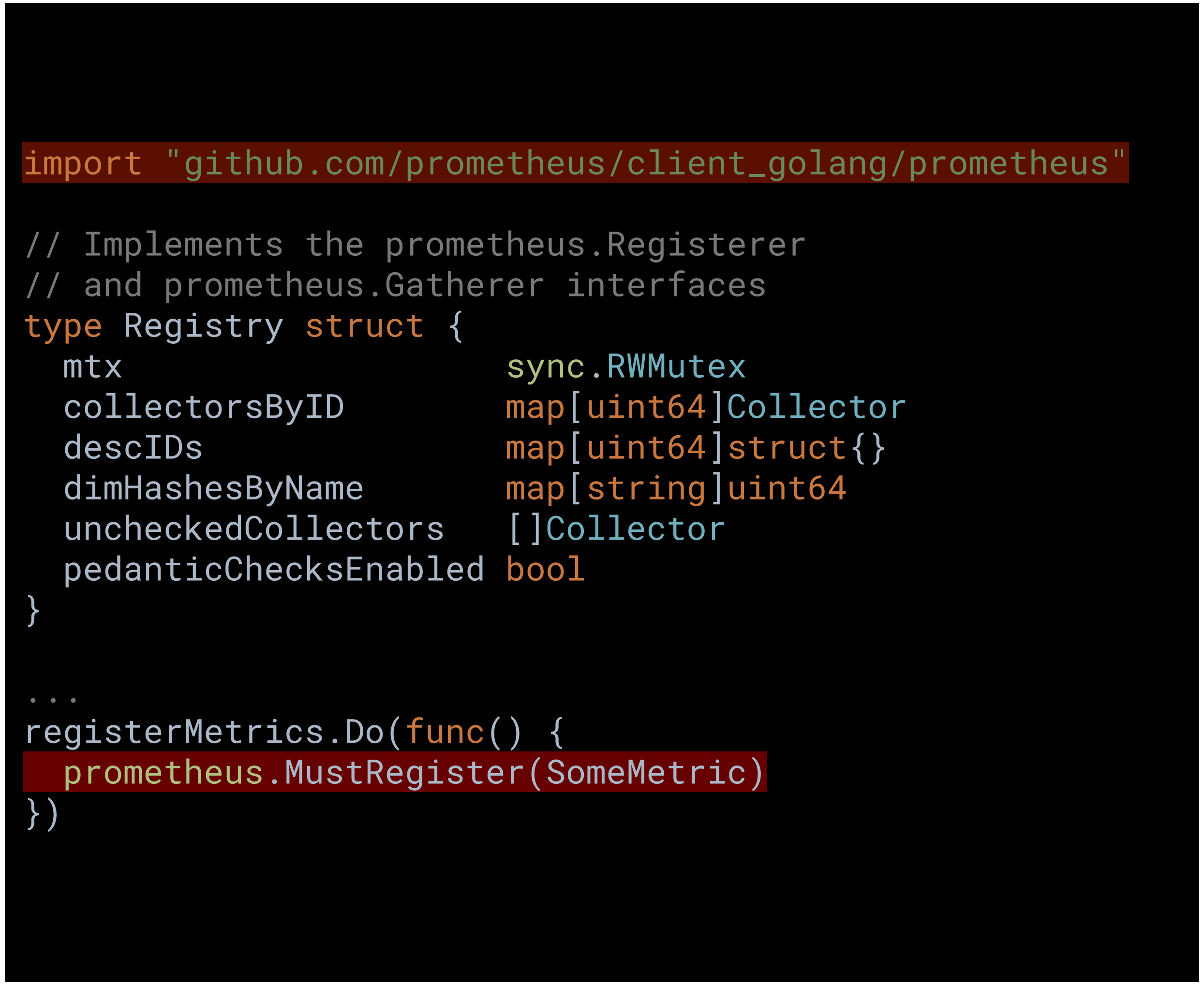


```
import "github.com/prometheus/client_golang/prometheus"

var (
  rpcDurations = prometheus.NewSummaryVec(
     prometheus.SummaryOpts{..}
  )
)
```

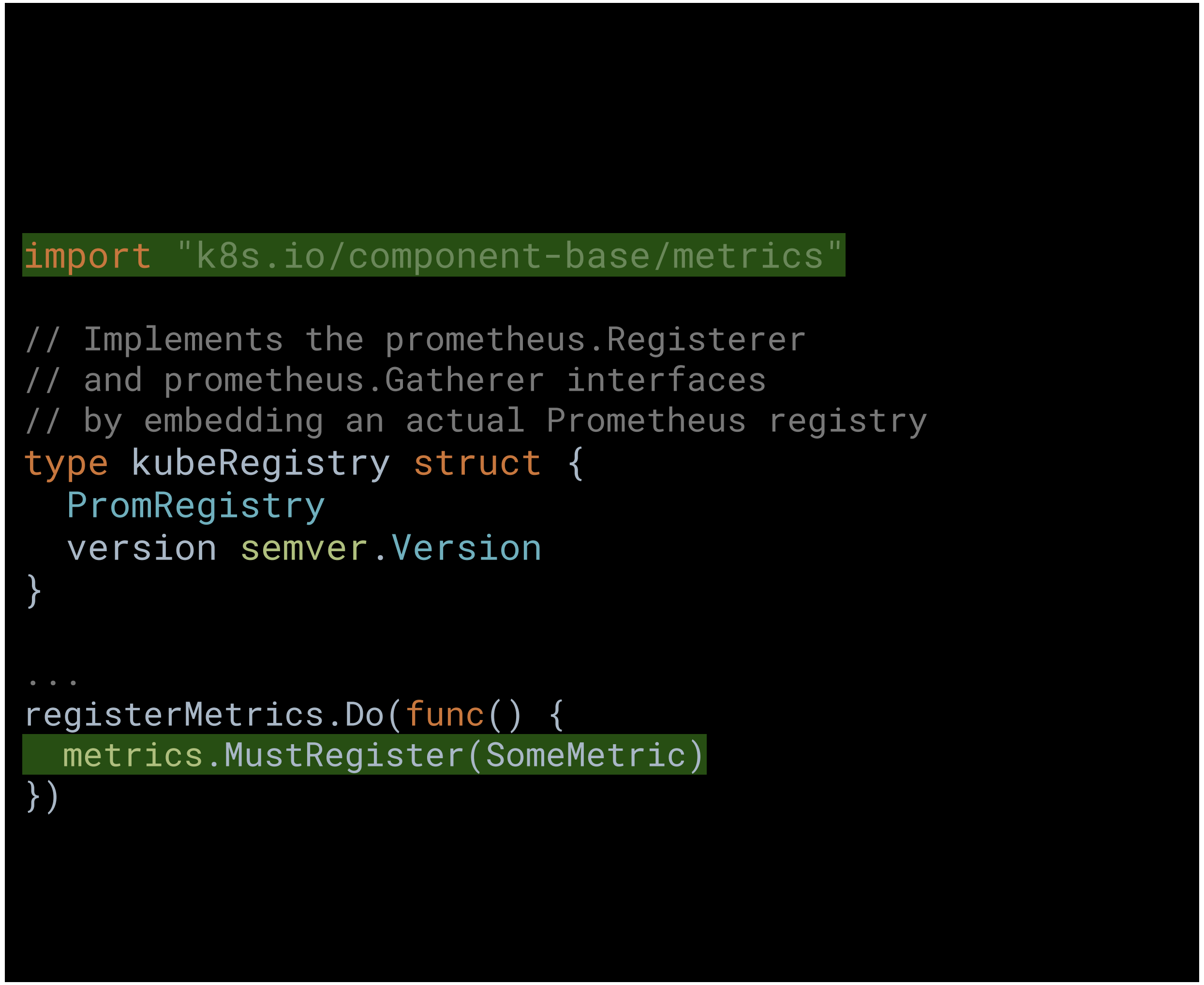


```
import "k8s.io/component-base/metrics"

var (
  rpcDurations = metrics.NewSummaryVec(
     metrics.SummaryOpts{..} // hijacked metric
definition here
  )
)
```

Hijacking Metric Registry



```go
import "github.com/prometheus/client_golang/prometheus"

// Implements the prometheus.Registerer
// and prometheus.Gatherer interfaces
type Registry struct {
  mtx                     sync.RWMutex
  collectorsByID          map[uint64]Collector
  descIDs                 map[uint64]struct{}
  dimHashesByName         map[string]uint64
  uncheckedCollectors     []Collector
  pedanticChecksEnabled bool
}

...
registerMetrics.Do(func() {
  prometheus.MustRegister(SomeMetric)
})
```



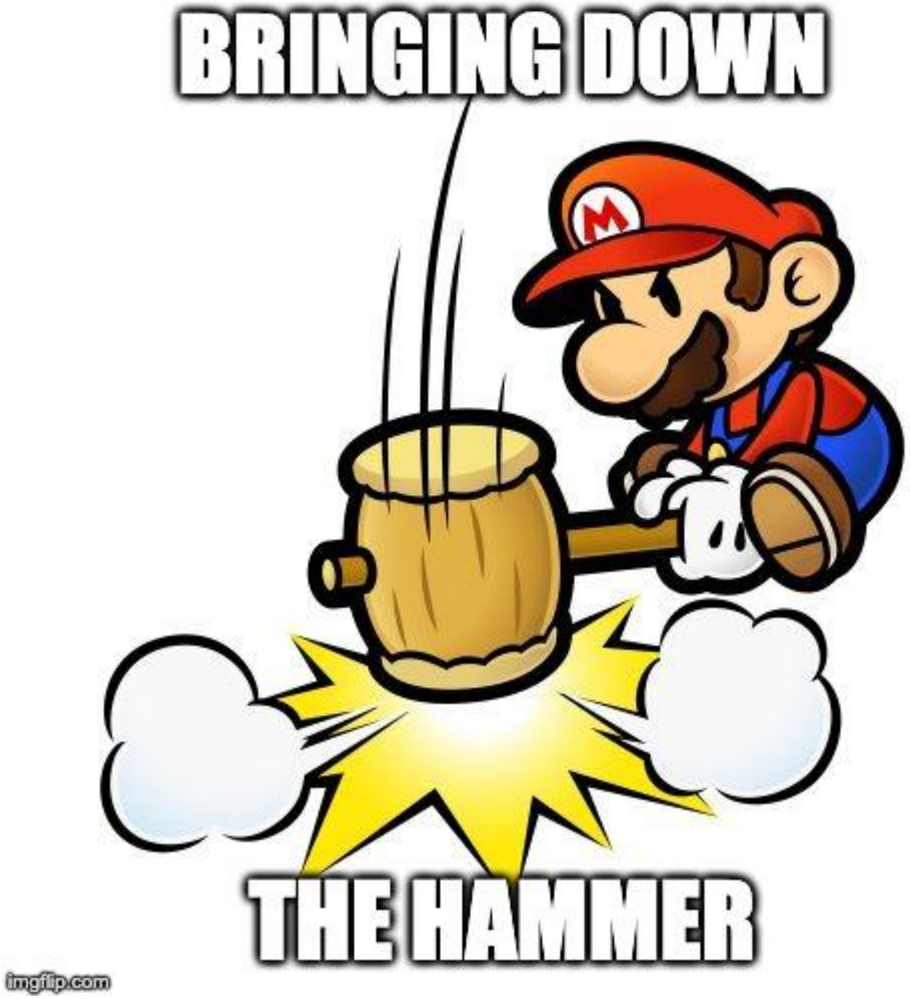
```go
import "k8s.io/component-base/metrics"

// Implements the prometheus.Registerer
// and prometheus.Gatherer interfaces
// by embedding an actual Prometheus registry
type kubeRegistry struct {
  PromRegistry
  version semver.Version
}

...
registerMetrics.Do(func() {
  metrics.MustRegister(SomeMetric)
})
```

Stability Axes (*axises)

- Stability Classes
  - Alpha - no stability guarantees
  - Stable - guaranteed not to change
- Deprecation
  - Intent - to signal future deletion of the metric
  - Lifecycle:
    - Stable (v1.15) -> Deprecated (v1.16) -> Hidden (v1.17) -> Deletion (v1.18)

Enforcing Stability

1. All metrics in Kubernetes use custom registries
2. Verify and validate metrics using static analysis
3. Forbid direct use of prometheus (Beta!)
4. Providing runtime escape hatch for turning off metrics (GA)



KubeCon | CloudNativeCon

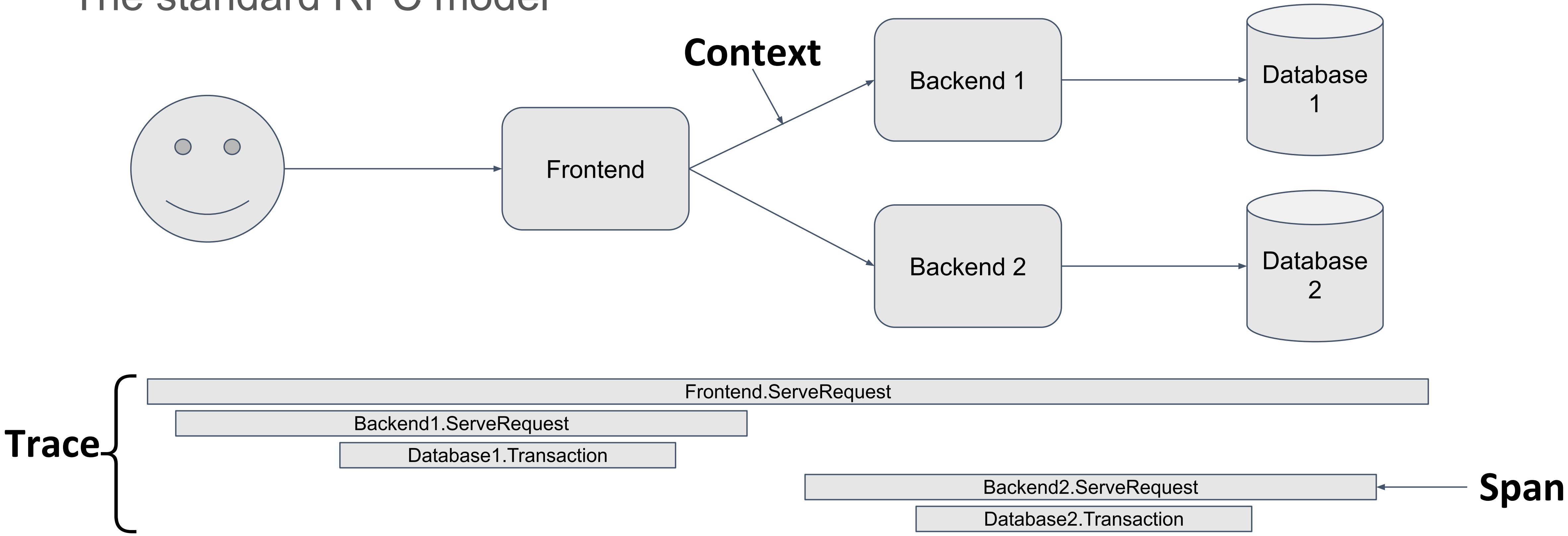
North America 2019

Tracing in Kubernetes

An in-progress proposal:
http://bit.ly/tracing-kep

Tracing

The standard RPC model



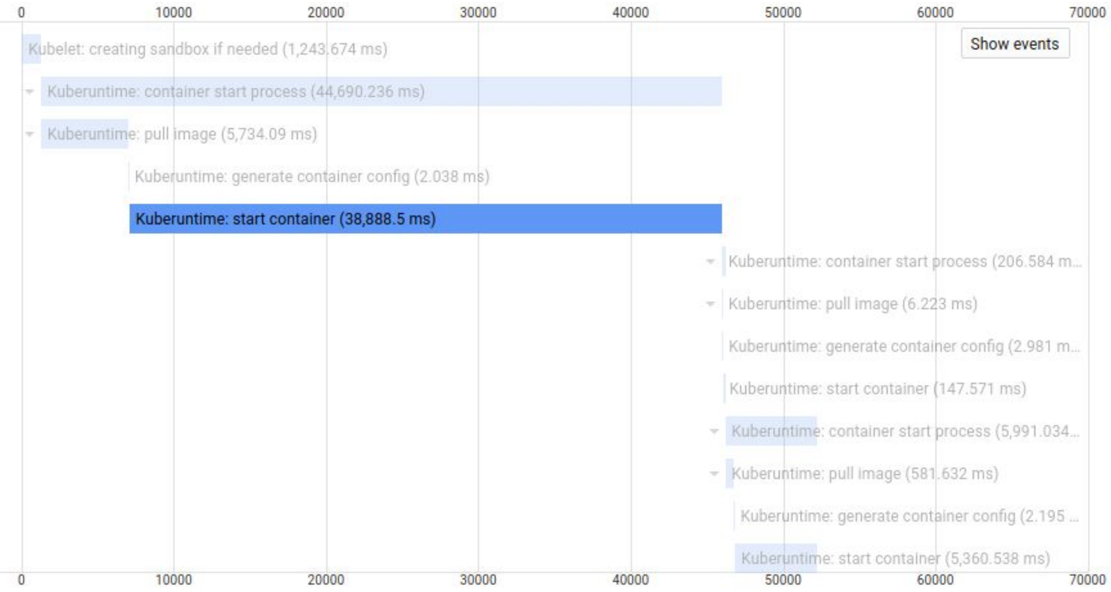
Story Time

Pod Startup should take ~3 seconds

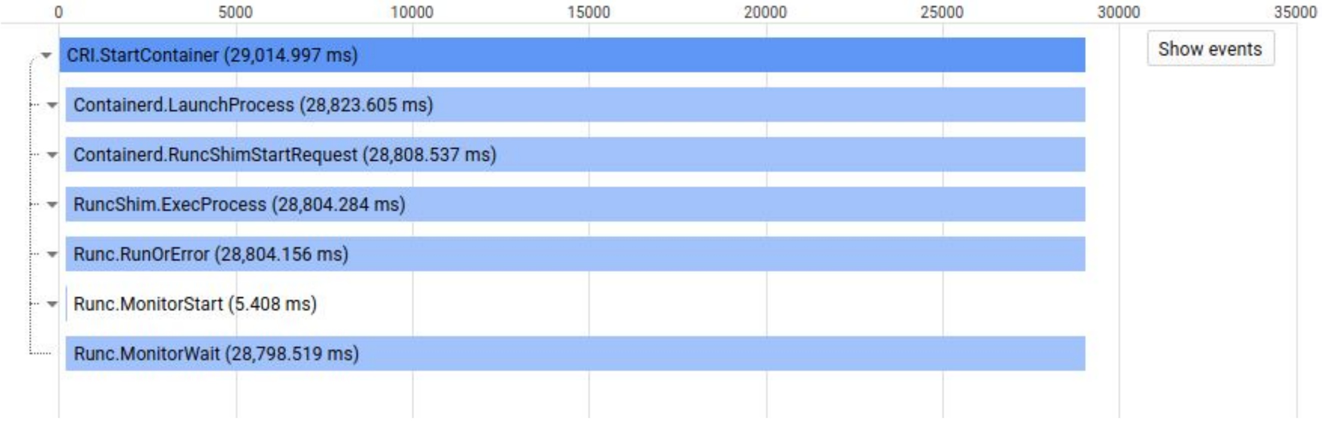
Customer seeing pod startup take **> 50 seconds**!

Detective Sam is on the case:

@Monkeyanator



Kubelet: creating sandbox if needed (1,243.674 ms)
Kuberuntime: container start process (44,690.236 ms)
Kuberuntime: pull image (5,734.09 ms)
Kuberuntime: generate container config (2.038 ms)
Kuberuntime: start container (38,888.5 ms)
Kuberuntime: container start process (206.584 m...
Kuberuntime: pull image (6.223 ms)
Kuberuntime: generate container config (2.981 m...
Kuberuntime: start container (147.571 ms)
Kuberuntime: container start process (5,991.034...
Kuberuntime: pull image (581.632 ms)
Kuberuntime: generate container config (2.195 ...
Kuberuntime: start container (5,360.538 ms)



CRI.StartContainer (29,014.997 ms)
Containerd.LaunchProcess (28,823.605 ms)
Containerd.RuncShimStartRequest (28,808.537 ms)
RuncShim.ExecProcess (28,804.284 ms)
Runc.RunOrError (28,804.156 ms)
Runc.MonitorStart (5.408 ms)
Runc.MonitorWait (28,798.519 ms)

Tracing in Kubernetes

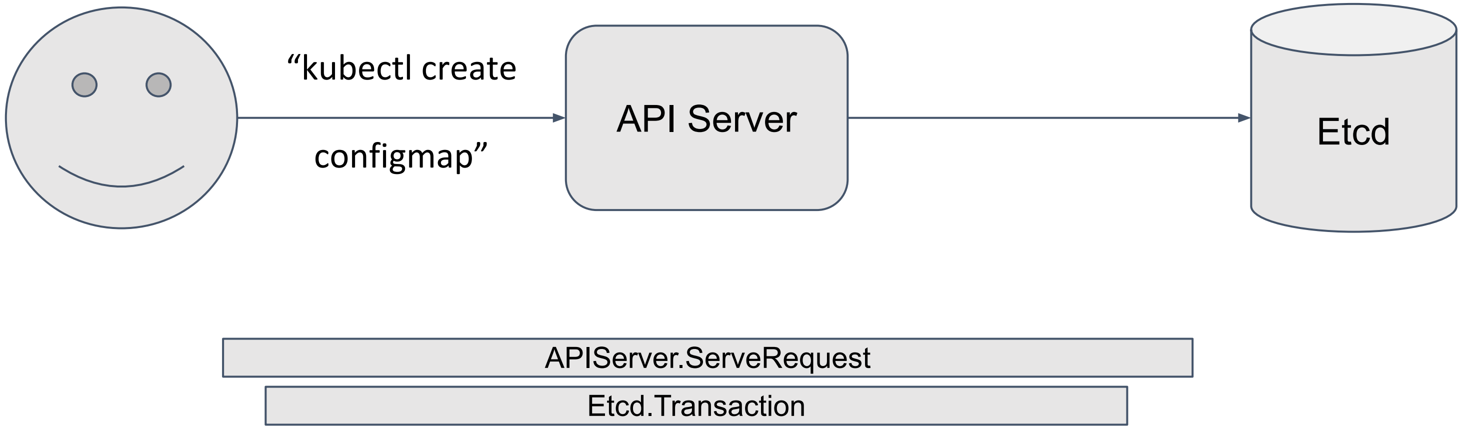
Why use tracing in Kubernetes?

- Logs:
  - Are fragmented between controllers
  - Are not consistently associated with objects (e.g. name vs UID)
- Metrics:
  - Have little metadata because of cardinality constraints.
- Events:
  - Are only kept for an hour
  - Are not easy to visualize

Tracing lets me know “What happened?” within seconds.

Tracing in Kubernetes

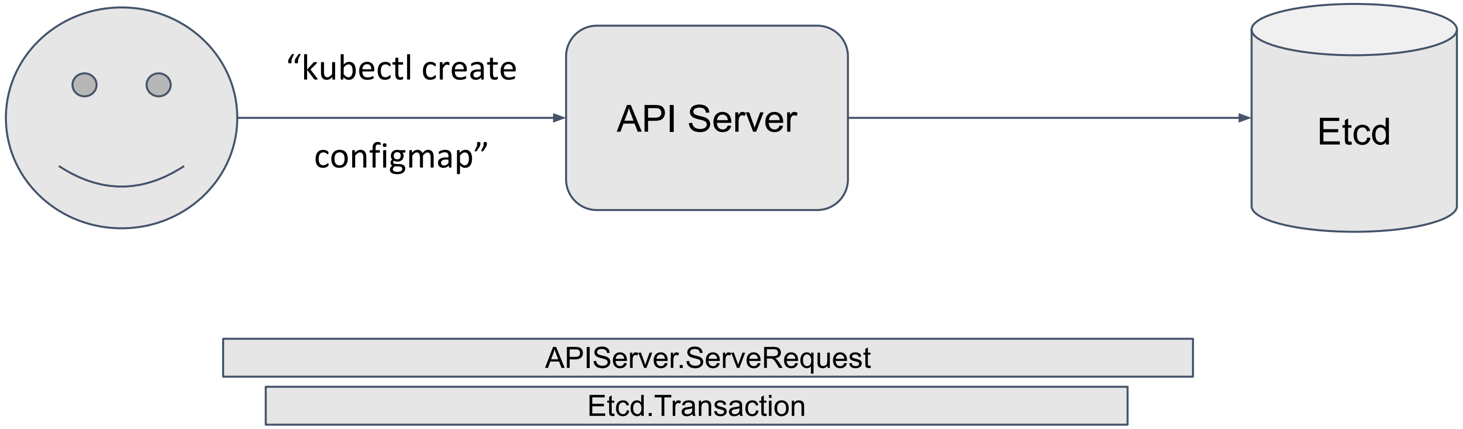
The standard RPC model



Demo time!

Tracing in Kubernetes

The standard RPC model

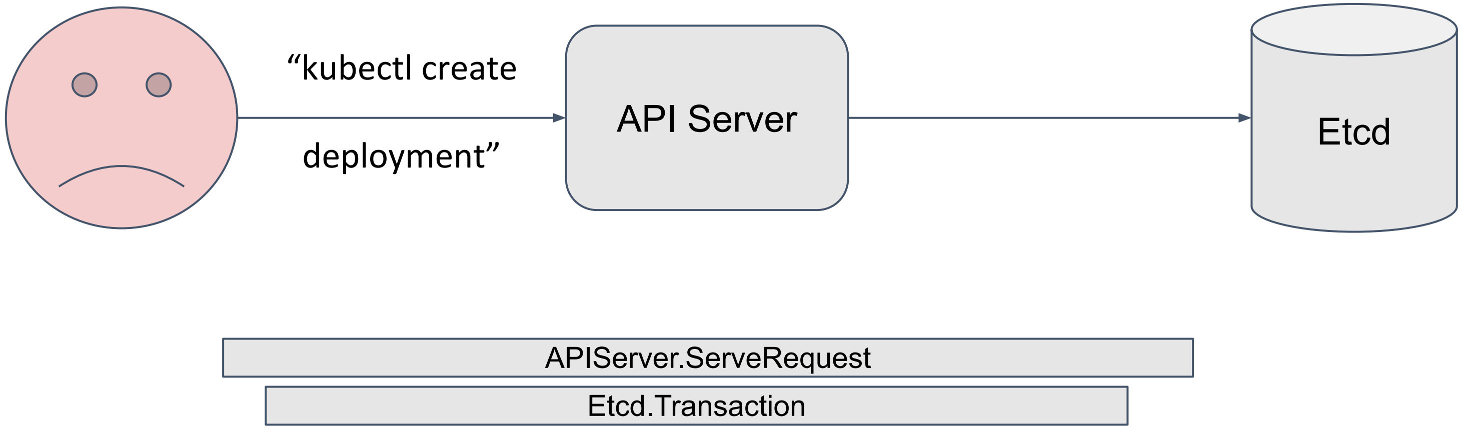


```
// WithTracing adds tracing to requests if the incoming
// request is sampled. This is used in the API Server
// http handler for incoming requests.
func WithTracing(handler http.Handler) http.Handler {
        return &ochttp.Handler{
                Handler: handler,
        }
}
```

```
// TracingOption returns a DialOption that traces
// outgoing RPCs if the request is sampled. This is used
// in the API Server grpc client for etcd.
func TracingOption() grpc.DialOption {
        return grpc.WithStatsHandler(
                &ocgrpc.ClientHandler{},
        )
}
```

Tracing in Kubernetes

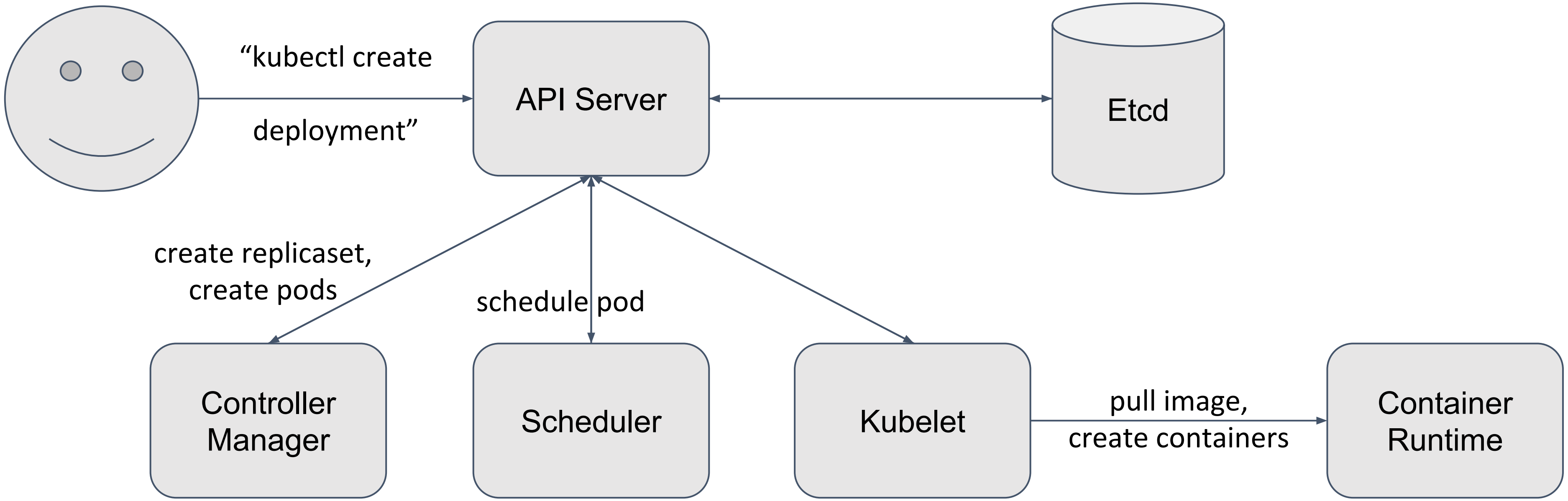
The standard RPC model



There's more to it than that!

Tracing in Kubernetes

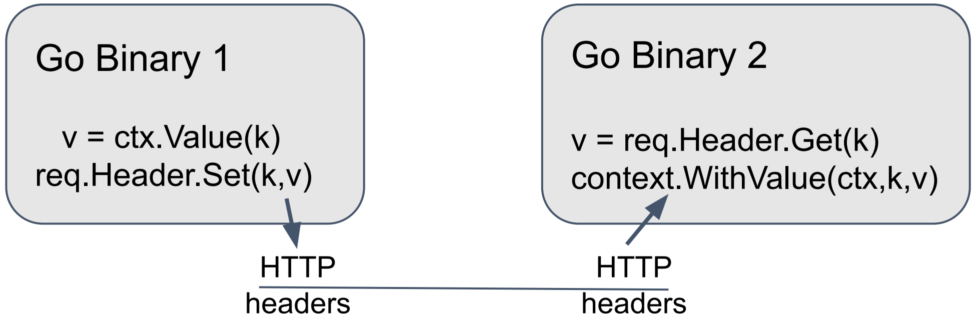
The standard RPC model misses some stuff



Span Context Propagation

How can we propagate context to controllers?

HTTP



Go Binary 1

```
v = ctx.Value(k)
req.Header.Set(k,v)
```

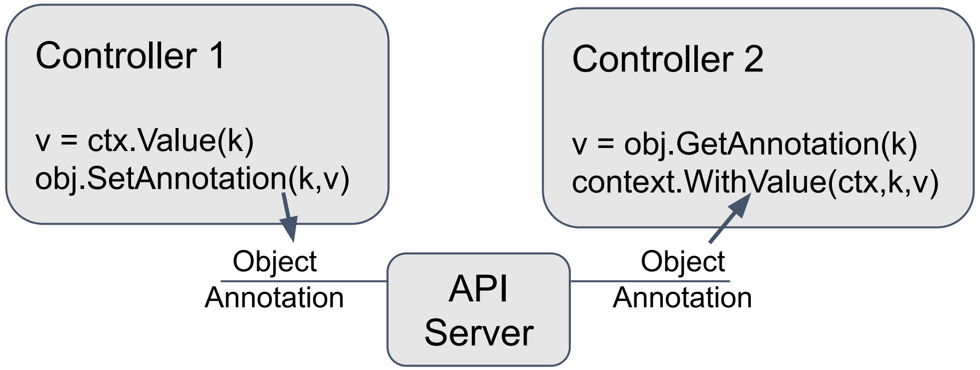
HTTP headers

HTTP headers

Go Binary 2

```
v = req.Header.Get(k)
context.WithValue(ctx,k,v)
```

Kubernetes Objects



Controller 1

```
v = ctx.Value(k)
obj.SetAnnotation(k,v)
```

Object Annotation

API Server

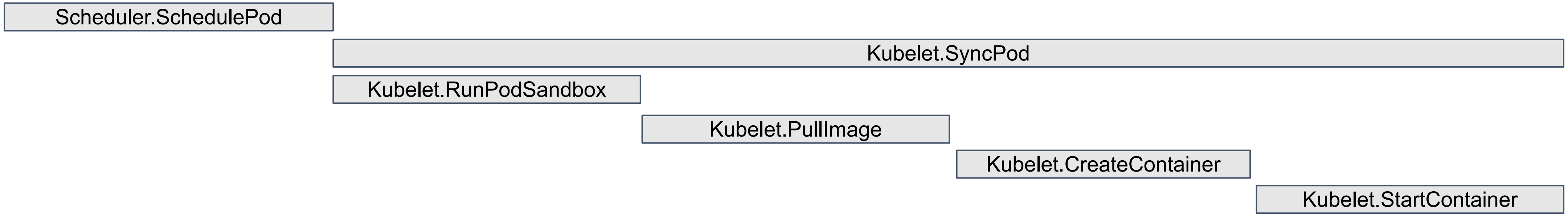
Object Annotation

Controller 2

```
v = obj.GetAnnotation(k)
context.WithValue(ctx,k,v)
```

Tracing a Pod

What should a pod trace look like?



Demo time!

Tracing a Pod Creation

Code changes

```go
// WithTracing adds tracing to requests if the outgoing
// request is sampled. This is used in client-go to
// allow kubernetes clients to trace requests.
func WithTracing(transport http.Transport) http.Handler {
        return &othttp.Transport{
                Base: transport,
        }
}
```

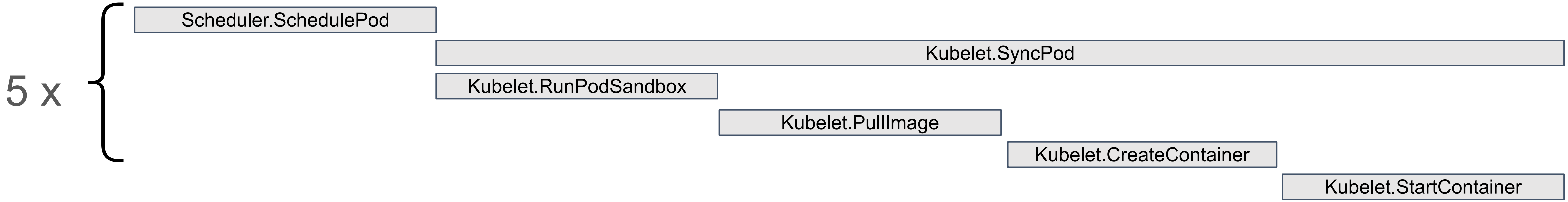
```go
// TracingOption returns a DialOption that traces
// outgoing RPCs if the request is sampled. This is used
// in the Kubelet grpc client for the CRI.
func TracingOption() grpc.DialOption {
        return grpc.WithStatsHandler(
                &otgrpc.ClientHandler{},
        )
}
```

```go
// scheduleOne does the entire scheduling workflow for a
// single pod.  It is serialized on the scheduling
// algorithm's host fitting.
func (sched *Scheduler) scheduleOne() {
        ...
        pod := sched.NextPod()
        ...
        _, schedulePodSpan := traceutil.StartSpanFromObject(pod, "kube-scheduler.SchedulePod")
        defer schedulePodSpan.End()
        ...
}
```

```go
// SyncPod syncs the running pod into the desired pod
func (m *kubeGenericRuntimeManager) SyncPod(pod *v1.Pod, ...) {
        ...
        if podContainerChanges.isEmpty() {
                return
        }
        ctx, syncPodSpan := traceutil.StartSpanFromObject(pod, "kubelet.SyncPod")
        defer syncPodSpan.End()
        ...
        // Create pod sandbox, pull images, start containers, etc.
        // Pass ctx to all CRI calls.
}
```

Tracing a Deployment Creation

What should a deployment creation trace look like?

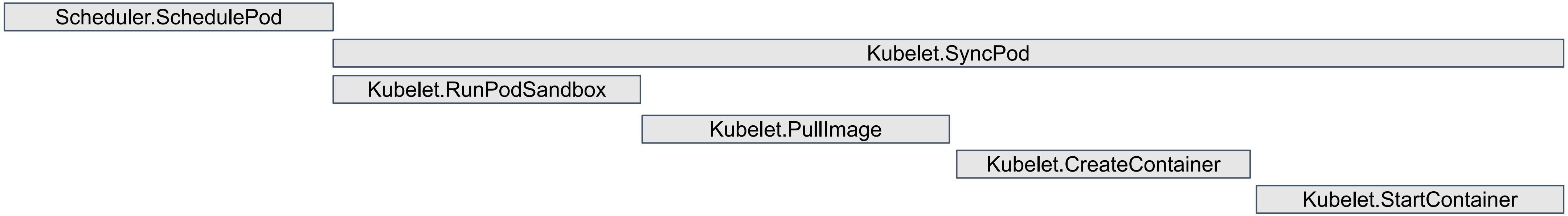


Just many pod traces?

One trace with many pods?

Tracing a Deployment Creation

What should a deployment creation trace look like?



Scheduler.SchedulePod

Kubelet.SyncPod

Kubelet.RunPodSandbox

Kubelet.PullImage

Kubelet.CreateContainer

Kubelet.StartContainer

~~Just many pod traces?~~

One trace with many pods

Demo time!

Tracing a Deployment Creation

Code Changes

```
// getNewReplicaSet returns a replica set that matches the intent of the given
// deployment. This propagates the context from the Deployment to the ReplicaSet,
// and uses the context in client requests to the API Server.
func (dc *DeploymentController) getNewReplicaSet(d *apps.Deployment, ...) {
        ...
        ctx, span := traceutil.StartSpanFromObject(d, "deployment.CreateReplicaSet")
        defer span.End()
        ...
        newRS := apps.ReplicaSet{...}
        traceutil.EncodeContextIntoObject(ctx, &newRS)
        newRs, err := dc.client.AppsV1().ReplicaSets(d.Namespace).Create(ctx, &newRs)
        ...
}
```

```
// manageReplicas checks and updates replicas for the given ReplicaSet. This
// propagates the context from the ReplicaSet to each Pod, and uses the context in
// client requests to the API Server.
func (rsc *ReplicaSetController) manageReplicas(pods []*v1.Pod, rs *apps.ReplicaSet){
        ...
        slowStartBatch(..., func() error {
                ctx, span := traceutil.StartSpanFromObject(rs, "replicaset.CreatePod")
                defer span.End()
                ...
                traceutil.EncodeContextIntoObject(ctx, pod)
                newPod, err := rsc.Client.CoreV1().Pods(d.Namespace).Create(ctx, pod)
                ...
        }
        ...
}
```

Tracing, Generalized

What actions should components export a Span for?
A: **When doing work that moves the object toward its desired state**

What object should the exported Span be associated with?
A: **The object whose actual state is moved towards its desired state**

```go
// Reconcile implements the kubebuilder controller reconciler.
func (r *myReconciler) Reconcile(request reconcile.Request) (reconcile.Result, error) {
        myObj := &v1.MyObject{}
        if err := r.Get(context.Background(), request.NamespacedName, myObj); err != nil {
                return reconcile.Result{}, err
        }
        if !updatesRequired(request) {
                return reconcile.Result{}, nil
        }
        ctx, span := traceutil.StartSpanFromObject(myObj, "mycontroller.Reconcile")
        defer span.End()
        // perform updates and send ctx with requests
        ...
}
```

Tracing, Generalized

What actions should components export a Span for?
A: **When doing work that moves the object toward its desired state**

What object should the exported Span be associated with?
A: **The object whose actual state is moved towards its desired state**

```
// Reconcile implements the kubebuilder controller reconciler.
func (r *myReconciler) Reconcile(request reconcile.Request) (reconcile.Result, error) {
      myObj := &v1.MyObject{}
      if err := r.Get(context.Background(), request.NamespacedName, myObj); err != nil {
            return reconcile.Result{}, err
      }
      /* ______________________________________________
         | This could send spans when no work is done! |
      */ ______________________________________________
      ctx, span := traceutil.StartSpanFromObject(myObj, "mycontroller.Reconcile")
      defer span.End()
      // perform updates and send ctx with requests
      ...
}
```

Tracing, Generalized

When should controllers propagate context from object A to object B?

A: **When updating object B’s desired state in order to move object A’s actual state towards its desired state.**

```
// Reconcile implements the kubebuilder controller Reconciler interface.
func (r *myReconciler) Reconcile(request reconcile.Request) (reconcile.Result, error) {
        objectA := &v1.MyObject{}
        if err := r.Get(context.Background(), request.NamespacedName, objectA); err != nil {
                return reconcile.Result{}, err
        }
        if !updatesRequired(request) {
                return reconcile.Result{}, nil
        }
        ctx, span := traceutil.StartSpanFromObject(objectB, "mycontroller.ReconcileMyObject")
        defer span.End()
        objectB := &v1.MyOtherObject{...}
        traceutil.EncodeContextIntoObject(ctx, objectB)
        objectB, err := r.MyV1().MyOtherObjects(request.NamespacedName.Namespace).Create(ctx, objectB)
        // Use ctx in all other requests done as part of this reconcile.
        ...
}
```

This is a Work-In-Progress

KEP: github.com/kubernetes/enhancements/pull/650

There are a few hard problems I missed...

When should a trace end?

- When updating object status to (Desired State == Actual State)

What happens when an update happens before the previous has finished?

- Link the new trace to the old trace?

```
// TODO(dashpole): Get KEP Approved
// TODO(community): Instrument All The Things!
```

OpenTelemetry

A lesson learned from Heapster and cAdvisor:

Quality integrations with many vendors is difficult to maintain.

Using OpenTelemetry would allow vendors* to integrate with our telemetry

… while keeping Kubernetes components **vendor-neutral.**

* These are the vendors that had tracing integrations with OpenCensus

KubeCon | CloudNativeCon

North America 2019