Autoscale your Kubernetes Workload with Prometheus

Frederic Branczyk

Software Engineer at Red Hat (previously CoreOS)

Twitter @fredbrancz

GitHub @brancz

What’s to learn today?

- Officially recommended way to autoscale on Kubernetes today
- Architecture of solutions and their evolution over time
- Future outlook

Let’s talk about autoscaling

Autoscaling on an abstract level:

- Calculate resources to cover demand
- Demand measured by metrics
  - Metrics must be collected, stored and made queryable
  - To fulfill SLO of SLA through SLI

Horizontal Autoscaling

- Horizontal Pod Autoscaler
- “Increase replicas when necessary”

Vertical autoscaling

- Vertical Pod Autoscaler
- “Increase resource request/limit when necessary“
  - “Necessary” → described by metric

History of autoscaling on Kubernetes

Autoscaling used to be heavily relying on Heapster

- Heapster collects metrics and writes to time-series database
- Metrics collection via cAdvisor (container + custom-metrics)

HorizontalPodAutoscaler (v1)

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: brancz-com-blog
  namespace: brancz-com-blog
spec:
  scaleRef:
    apiVersion: apps/v1beta1
    kind: Deployment
    name: brancz-com-blog
  minReplicas: 5
  maxReplicas: 10
  cpuUtilization:
    targetPercentage: 70
```

We can autoscale!

Problem with Heapster over time

Heavily reliant on heapster API

- Loosely specified APIs
- Unmaintained vendor implementations
- Push only time-series databases
  - Prometheus extremely popular
  - Heapster monitored by Prometheus → confusing!?

This pipeline needed a redesign

Autoscaling API

Allowing arbitrary metrics

A new solution:

Resource & Custom Metrics API

Resource & Custom Metrics APIs

Well defined APIs:

- Not an implementation, an API spec
- Implemented and maintained by vendors
- Returns single value

Resource Metrics API

- Core metrics: CPU, memory for Pod/Container/nodes (may be extended)
- Metrics-server is the canonical implementation
  - Holds state in memory
  - Collects metrics from kubelet stats API
  - Collected once a minute (Supported: 25000 metrics/second)
    - (10 Metrics x 5000 Nodes x 30 Pods/Node) / 60seconds

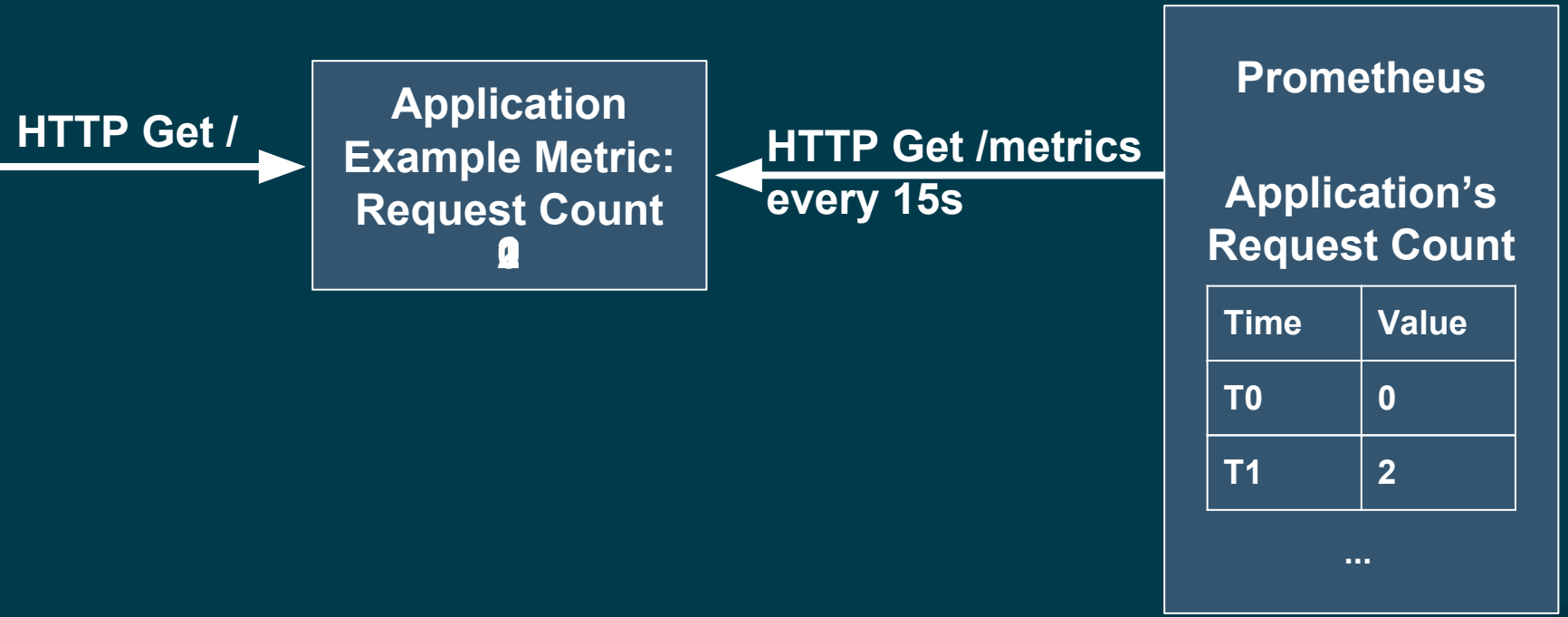
Custom Metrics API

- Same semantics
  - Single value returned
- No canonical implementation
  - Has to be provided specific for monitoring vendor
- Related to a Kubernetes object: Pod, Service, Deployment
- Example: github.com/directXMan12/k8s-prometheus-adapter

External Metrics API

- Not related to a Kubernetes object
- Same semantics
  - Single value returned
- No canonical implementation
  - Has to be provided specific for monitoring vendor

Quick Prometheus intro



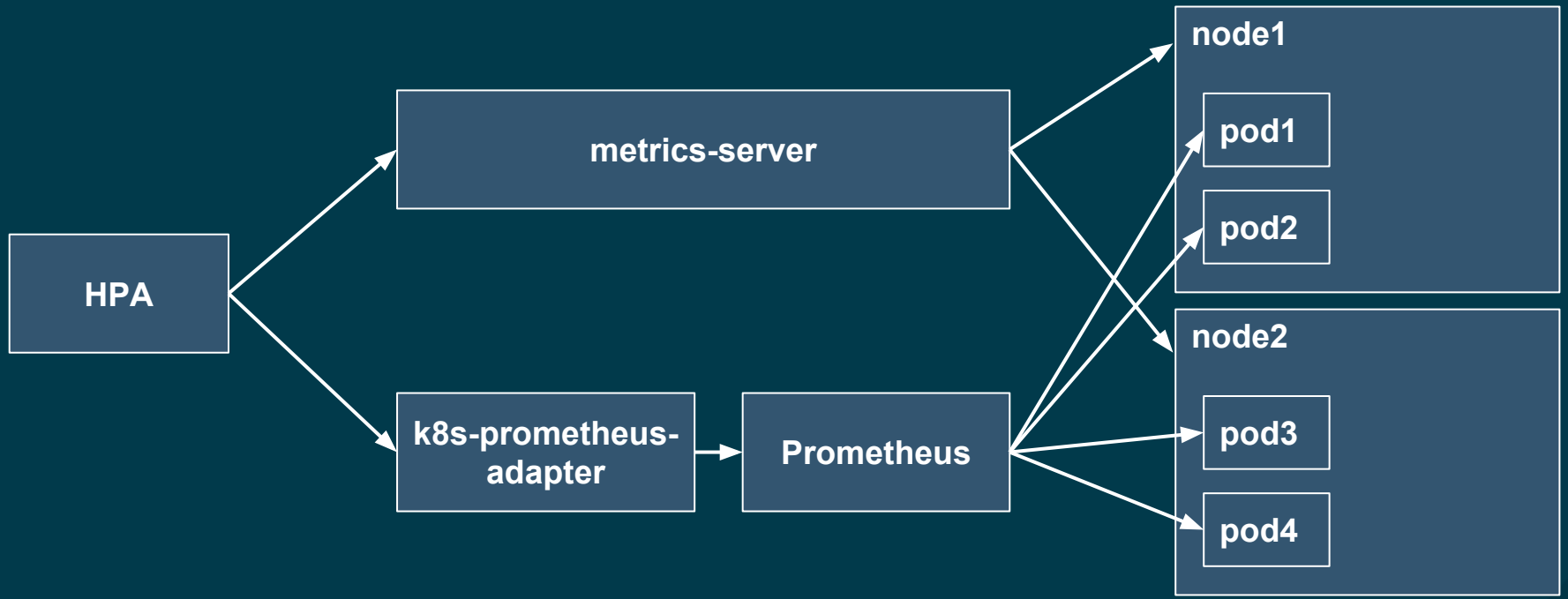
Let’s make the HPA

use this!

HorizontalPodAutoscaler (v2beta1)

```
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: brancz-com-blog
  namespace: brancz-com-blog
spec:
  scaleTargetRef:
    apiVersion: apps/v1beta1
    kind: Deployment
    name: brancz-com-blog
  minReplicas: 5
  maxReplicas: 10
  metrics:
  - type: Pods
    pods:
      metricName: http_requests
      targetAverageValue: 200 # targetUtilization: 10%
```

Monitoring pipeline overview



```
...
  metrics:
  - type: Pods
    pods:
      metricName: http_requests
      targetAverageValue: 200 # targetUtilization: 10%
```

```
   sum(rate(http_requests_total{namespace="hpa-namespace"}[5m])) by(pod)
```

Demo!

What has the future in store

(known)

Mark Heapster as Deprecated #2022

Open **DirectXMan12** wants to merge 1 commit into `kubernetes:master` from `DirectXMan12:docs/deprecate-heapster`

Conversation 16 | Commits 1 | Files changed 2

DirectXMan12 commented 5 days ago — Member

This marks Heapster as deprecated, adding a deprecation timeline that culminates in the retirement of the Heapster project for the Kubernetes 1.13 release.

As per the discussion in SIG Instrumentation.

More vertical pod

autoscaling/autosizing!

Autoscale

CustomResourceDefinitions!

Shout out to sttts and nikhita!

Stable metrics!

What has the future in store

(predictions)

Cluster auto scaler not a special case

Standardization of monitoring

gRPC: github.com/grpc-ecosystem/go-grpc-prometheus

ServiceMesh: conduit, istio, nginx

Reusable

Alerts, Dashboards

SLO/SLI autoscaling

Let’s have a look how our HPA is doing

Thank you!

Questions?

Frederic Branczyk

Twitter: @fredbrancz

GitHub: @brancz