SIG Autoscaling

Solly Ross (Google) and Michael Hausenblas (Red Hat)

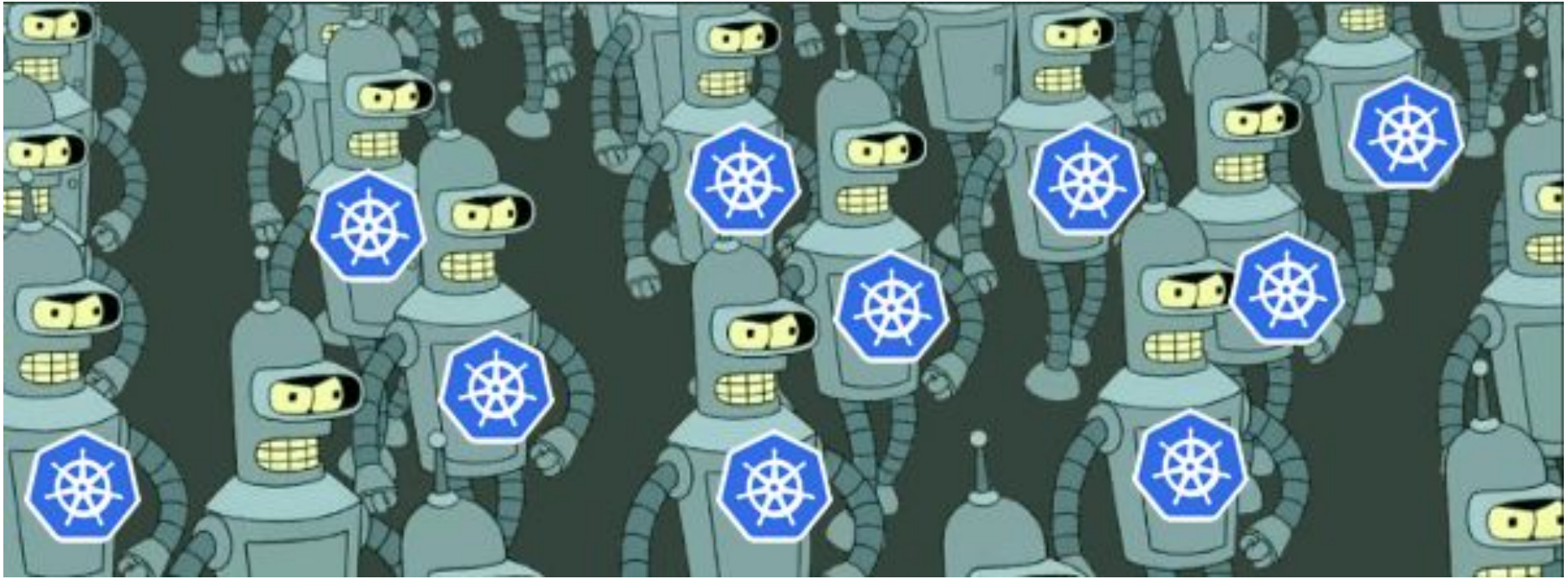
Autoscaling?

- On an abstract level:
  - Calculate resources to cover demand
  - Demand measured by metrics
  - Metrics must be collected, stored and queryable
- Ultimately to fulfill:
  - Service Level Objectives (SLO) ...
  - of Service Level Agreements (SLA) ...
  - through Service Level Indicators (SLI)

Types of autoscaling (in Kubernetes)

- Cluster-level
- App-level
  - Horizontal
  - Vertical

Cluster Autoscaling



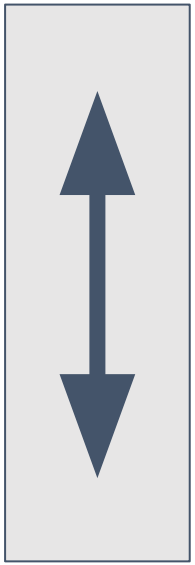
https://cdn-images-1.medium.com/max/1600/1*P8Frvw4Nvhrv_LwPnru5-w.png

Horizontal autoscaling

- Horizontal pod autoscaler
- Resource: replicas
- “Increasing replicas when necessary”
- Requires application to be designed to scale horizontally

Vertical autoscaling

- Vertical pod autoscaler
- Resource: CPU/Memory
- “Increasing CPU/Memory when necessary”
- Less complicated to design for resource increase
- Harder to autoscale



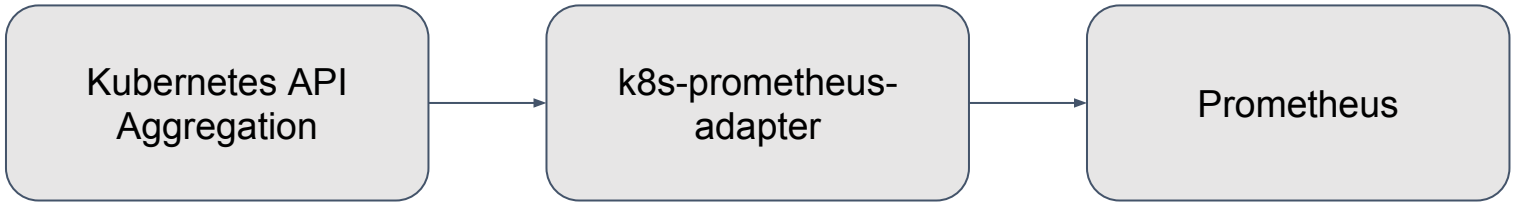
History of Autoscaling on Kubernetes

- Autoscaling used to heavily rely on Heapster
  - Heapster collects metrics and writes to time-series database
  - Metrics collection via cAdvisor (container + custom-metrics)
- We could autoscale!


KubeCon
CloudNativeCon
North America 2018
Custom Metrics API

Resource & Custom Metrics APIs

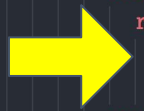
- Well defined APIs:
  - Not an implementation, an API spec
  - Implemented and maintained by vendors
  - Returns single value




KubeCon
CloudNativeCon
North America 2018
Vertical Pod Autoscaling

Background & terminology


```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: hamster
  namespace: default
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: hamster
    spec:
      containers:
      - name: hamster
        image: k8s.gcr.io/ubuntu-slim:0.1
        resources:
          requests:
            cpu: 100m
            memory: 50Mi
        command: ["/bin/sh"]
        args: ["-c", "while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done"]
```



Background & terminology

- Scheduling
  - nodes offer resources
  - pods consume resources
  - scheduler matches needs of pods based on requests
- Types of resources (compressible/incompressible)
- Quality-of-Service (QoS)
  - Guaranteed: limit == request
  - Burstable: limit > request > 0
  - Best-Effort: ∄ (limit, request)

Motivation

Unfortunately, Kubernetes has not yet implemented dynamic resource management, which is why we have to set resource limits for our containers. I imagine that at some point Kubernetes will start implementing a less manual way to manage resources, but this is all we have for now.

Ben Visser, 12/2016
Kubernetes—Understanding Resources

Kubernetes doesn't have dynamic resource allocation, which means that requests and limits have to be determined and set by the user. When these numbers are not known precisely for a service, a good approach is to start it with overestimated resources requests and no limit, then let it run under normal production load for a certain time.

Antoine Cotten, 05/2016
1 year, lessons learned from a 0 to Kubernetes transition

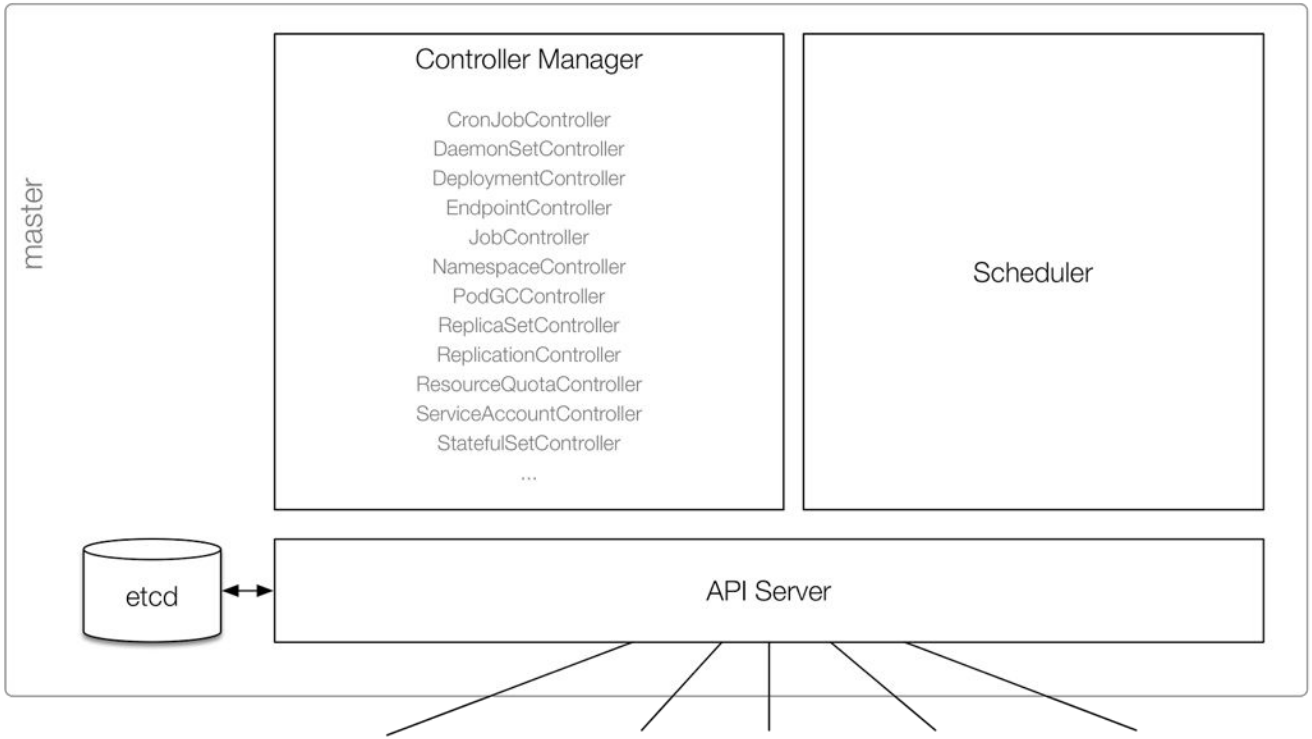
Goals

- Automating configuration of resource requirements
  - manually setting requests is brittle & hard so people don't do it
  - no requests set ➜ QoS is *best effort* :(
- Improving utilization
  - can better bin pack
  - impact on other functionality such as out of resource handling or an (aspirational) optimizing scheduler

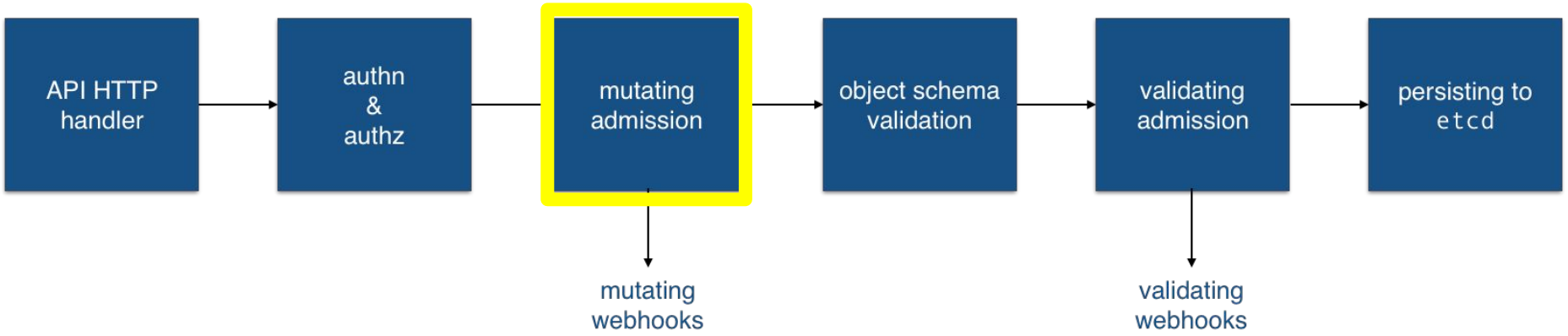
Use Cases

- For stateful apps, for example Wordpress or single-node databases
- Can help on-boarding of "legacy" apps, that is, non-horizontally scalable ones

Interlude: API server



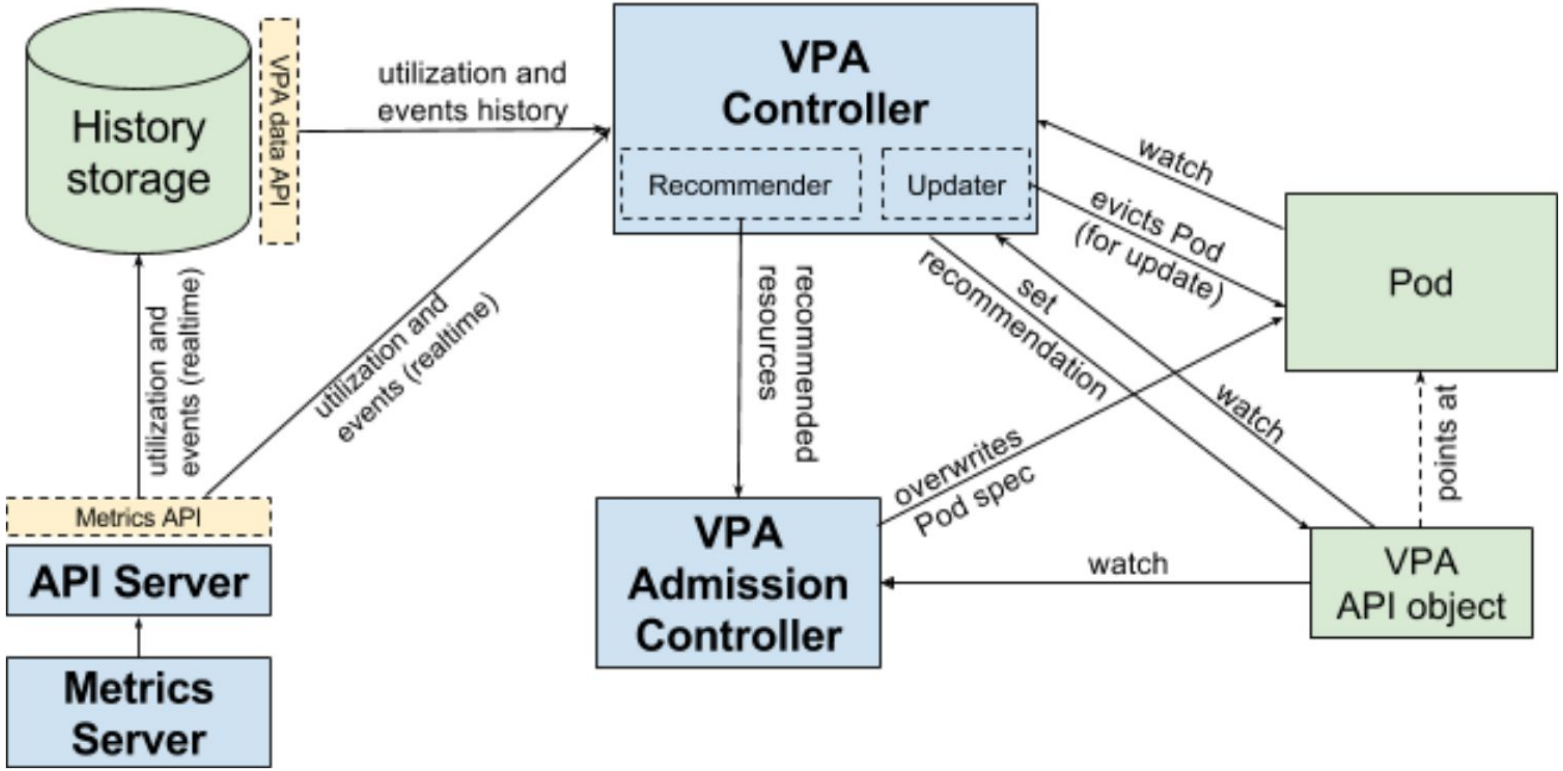
Interlude: API server



Basic idea

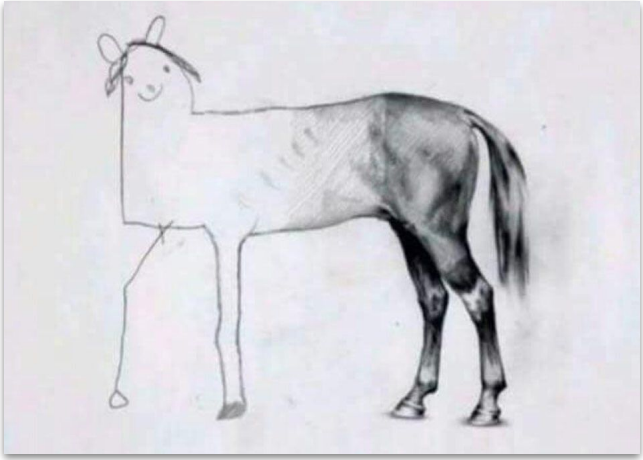
- observe resource consumption of all pods
- build up historic profile (recommender)
- apply to pods on an opt-in basis via labels (updater)

VPA architecture

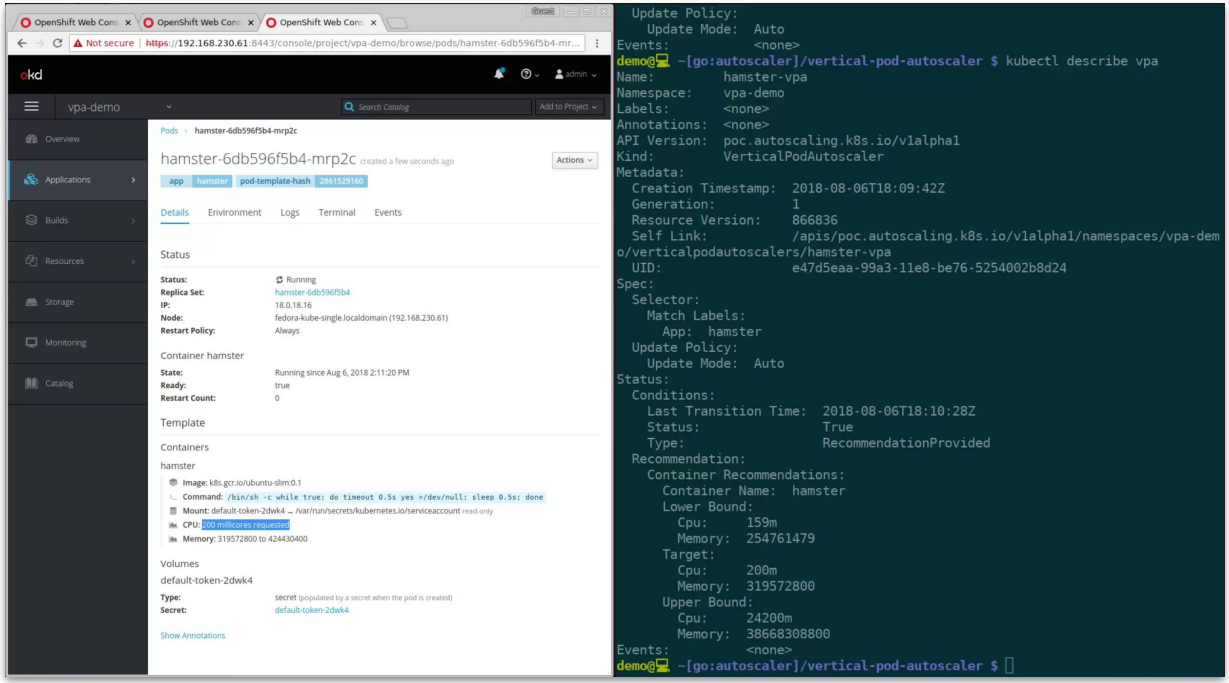


Limitations

- pre-alpha, so need testing and tease out edge-cases
- in-place updates (requires support from container runtime)
- usage spikes—how to deal with it best?



VPA demo




```
  Update Policy:
    Update Mode:  Auto
Events:           <none>
demo@ ~[go:autoscaler]/vertical-pod-autoscaler $ kubectl describe vpa
Name:         hamster-vpa
Namespace:    vpa-demo
Labels:       <none>
Annotations:  <none>
API Version:  poc.autoscaling.k8s.io/v1alpha1
Kind:         VerticalPodAutoscaler
Metadata:
  Creation Timestamp:  2018-08-06T18:09:42Z
  Generation:          1
  Resource Version:    866836
  Self Link:           /apis/poc.autoscaling.k8s.io/v1alpha1/namespaces/vpa-dem
o/verticalpodautoscalers/hamster-vpa
  UID:                 e47d5eaa-99a3-11e8-be76-5254002b8d24
Spec:
  Selector:
    Match Labels:
      App:  hamster
  Update Policy:
    Update Mode:  Auto
Status:
  Conditions:
    Last Transition Time:  2018-08-06T18:10:28Z
    Status:                True
    Type:                  RecommendationProvided
  Recommendation:
    Container Recommendations:
      Container Name:  hamster
      Lower Bound:
        Cpu:     159m
        Memory:  254761479
      Target:
        Cpu:     200m
        Memory:  319572800
      Upper Bound:
        Cpu:     24200m
        Memory:  38668308800
Events:          <none>
demo@ ~[go:autoscaler]/vertical-pod-autoscaler $
```

Resources & what's next?

- VPA issue 10782
- VPA design
- Test, provide feedback
- SIG Autoscaling—come and join us on #sig-autoscaling
  - or weekly online meetings on Monday
- SIG Instrumentation and SIG Autoscaling work towards a historical metrics API—get involved there!