Building a Kubernetes Scheduler using Custom Metrics

Mateo Burillo, Integrations Engineer. 05-18

Sysdig

Building a Kubernetes Scheduler using Custom Metrics

1 - INTRO TO KUBERNETES POD SCHEDULING

- Schedulers and scalers
- Birth of a pod, the scheduler role
- Fine-tuning the Kubernetes scheduler
- Hard and soft decision constraints

2 - CREATE YOUR KUBERNETES CUSTOM SCHEDULER

- Multiple schedulers and schedulerName selectors
- Main loop and relevant code sections

3 - CODE OVERVIEW AND DEMO

4 - IMPLEMENTED & PROPOSED IMPROVEMENTS

- Metrics cache
- Failsafe code
- Adding constraints and variables

5 - FAILSAFE CODE DEMO

6 - Q&A

Sysdig

This talk is not about

The Container Intelligence Platform for Kubernetes

And the usual suspects (Prometheus, Docker Swarm, OpenShift, Mesos…)

- Monitoring & Alerts
- Run-time security for containers & microservices
- Troubleshooting & Forensics

Try this at home: https://sysdig.com/sign-up/

Schedulers, Scalers ...

This talk is not about

- **Kubernetes Horizontal Pod Autoscaler or HPA:** Updates the number of pods required in a deployment (scale up / scale down) in response to a metric & threshold value.

https://sysdig.com/blog/kubernetes-scaler/

- **Vertical scalers:** automatically scales the resource limit definition (MutatingAdmissionWebhooks).
- **Node scalers:** Mostly cloud vendor dependant.

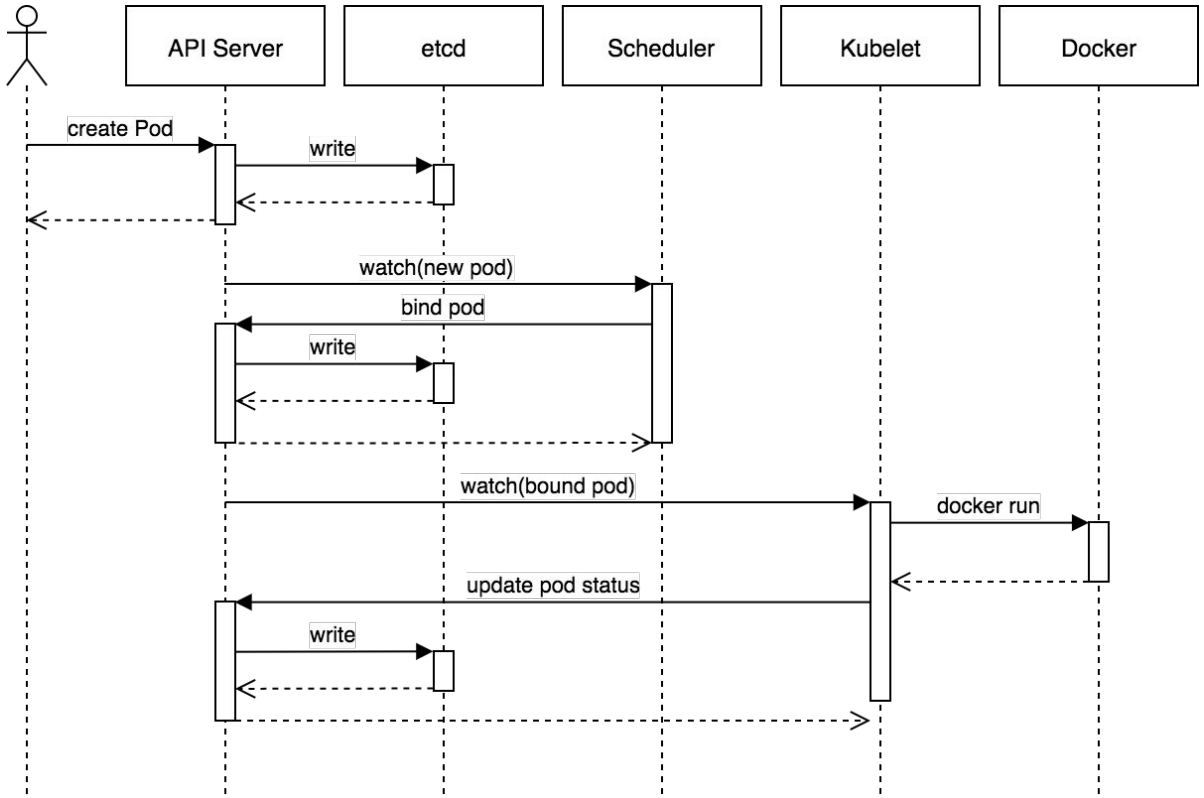
Schedulers, Scalers ...

This talk is about

- **Kubernetes Scheduler:** Assigns newly created pods to Kubernetes nodes. You can also use custom metrics to configure your Kubernetes scheduler.

The scheduler watches Kubernetes API, performs iterative steps to converge: Current cluster state -> Declarative cluster model.

Birth of a pod



AFTER:

- **API AUTH**
- **ADMISSION CONTROLLERS**
- **ETCD**

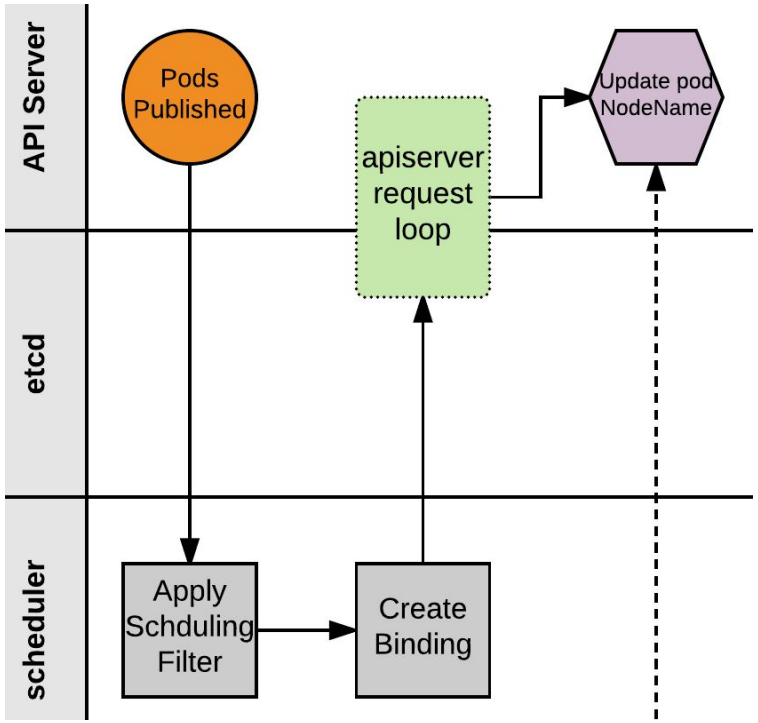
BEFORE:

- **KUBELET PULLS & RUNS THE IMAGE**

Source: Joe Beda's blog
https://blog.heptio.com/core-kubernetes-jazz-improv-over-orchestration-a7903ea92ca

Sysdig

Zooming in - The scheduler job



1 - Watch for pods that:

- Are in PENDING phase
- Have no NodeName assigned
- Are explicitly requesting our scheduler (default otherwise)

2 - Node selection algorithm

3 - Post Pod <-> Node binding to the API Server

4 - Profit!

Scheduler - Basic Behaviour

- Filter
  - Node can fulfil pod resource requests
  - Requested data volumes locally mounted
  - Nodes discarded by selectors / taints / etc
- Rank
  - LeastRequestedPriority (Resource exhaustion)
  - CalculateNodeLabelPriority (Explicitly declared affinities)
  - CalculateSpreadPriority (Favor spreading pods of the same service over different nodes)

It will spread and balance as evenly as possible

Scheduling - Hard constraints

- Taints:
  - Applied to nodes
  - Key, Value and Effect
    - NoSchedule: Master node(s) usually have something like:
      - node-role.kubernetes.io/master=true:NoSchedule
    - NoExecute: Pod eviction
  - Pods can *tolerate* these taints (i.e. DaemonSets)

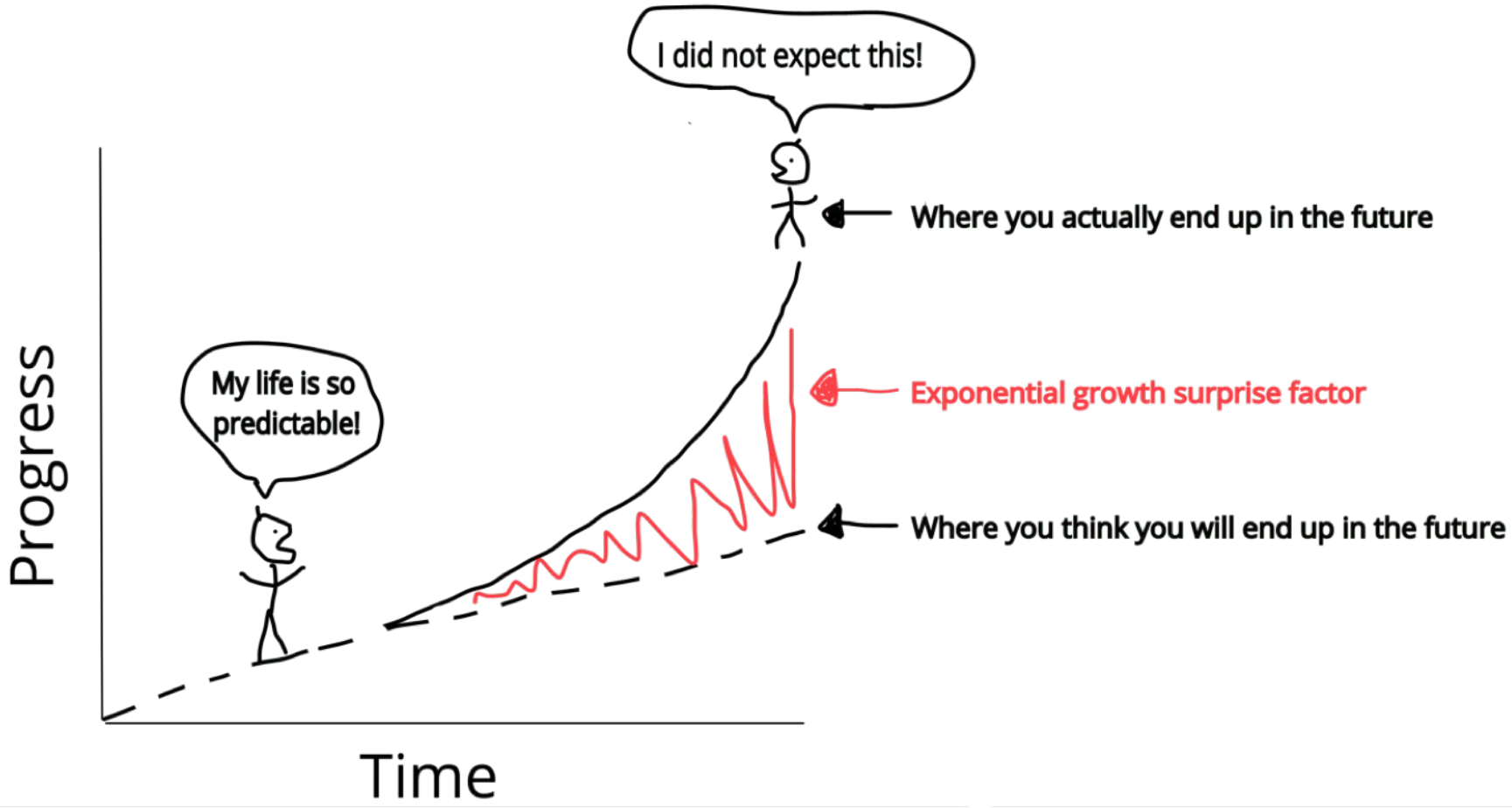
- nodeSelector
  - Explicit requisites inside the pod declaration

```
nodeSelector:
    arch: amd64
```

Scheduling - Soft constraints

- PreferNoSchedule effect: Soft version of ‘NoSchedule’
  - Dedicated hardware, master nodes, etc
- nodeAffinity (beta feature): Soft version of the nodeSelector
- podAffinity:
  - network distance:
    - two components that interchange information often (latency, bandwidth)
  - performance
  - security
    - Special privileged pods VS Sensitive data pods
- Also weight hierarchy!

Scheduling - Affinity Complexity



The affinity / anti affinity & friends graph of constraints is not linear !

Add only important constraints: there is a computational trade-off.

Scheduling - Experimental features

- Pod priority (alpha in 1.10+, present since 1.8+)
  - Preemption: Evict less important pods (if needed) to fit important ones.
  - Scheduling priority (since 1.9) in the queue of Pending pods.
  - Out of resource eviction: If the node starts to run out of resources it will evict less important pods first.

```yaml
apiVersion: scheduling.k8s.io/v1alpha1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000000
```

Scheduling: Experimental features

- TaintBasedEvictions (alpha)
  - NoExecute: Representing node problems dynamically using taints
  - tolerationSeconds: If your pod has “expensive” local state and there is a chance of recovery, you can tolerate the node failure for a while.

```
tolerations:
- key: "node.alpha.kubernetes.io/unreachable"
  operator: "Exists"
  effect: "NoExecute"
  tolerationSeconds: 6000
```

Building a Kubernetes Scheduler using Custom Metrics

1 - INTRO TO KUBERNETES POD SCHEDULING

- Schedulers and scalers
- Birth of a pod, the scheduler role
- Fine-tuning the Kubernetes scheduler
- Hard and soft decision constraints

2 - CREATE YOUR KUBERNETES CUSTOM SCHEDULER

- Multiple schedulers and schedulerName selectors
- Main loop and relevant code sections

3 - CODE OVERVIEW AND DEMO

4 - IMPLEMENTED & PROPOSED IMPROVEMENTS

- Metrics cache
- Failsafe code
- Adding constraints and variables

5 - FAILSAFE CODE DEMO

6 - Q&A

Sysdig

Why custom metrics

- You are limited to hardware constraints:
  - CPU, memory
  - Disk pressure
  - Out of disk, etc
- Or static labeling (Affinity, Taints, etc)

What if you want to schedule your pods based on variable application metrics:

- **HTTP requests per second, HTTP latency:** Get the pods close to the consumer
- **GPU performance:** Currently available FLOPS in the node
- **Bottleneck:** Overall performance is good (HPA business), but one particular node is struggling

Custom Kubernetes scheduler

- From 1.6 Kubernetes supports “multiple schedulers”.
  - Don’t worry, you will still have the default one!

```
spec:
  schedulerName: sysdigsched
```

Your scheduler can run outside or inside the cluster, living as a pod that executes the algorithm itself.

- Outside the cluster: Testing, developing
- Pod container scheduler: Cleaner, autocontained

You have the flexibility of running any algorithm you can implement, including contacting third party APIs for extra data

Pod scheduler - RBAC credentials

The scheduler needs to contact the API and write new Bindings

serviceAccount and RBAC

You will need a new serviceAccount

```
spec:
  serviceAccount: cm-scheduler
```

Bind this serviceAccount with the *kube-scheduler* clusterrole:

```
system:kube-scheduler
ClusterRole.v1.rbac.authorization.k8s.io
```

https://sysdig.com/blog/kubernetes-security-rbac-tls/

Two different “Binding” resources in the API

- Binding resource (Deprecated in 1.7)
- pods/binding subresource

```
$ kubectl get clusterrole system:kube-scheduler -o yaml | grep binding
  - bindings
  - pods/binding
```

Building a Kubernetes Scheduler using Custom Metrics

1 - INTRO TO KUBERNETES POD SCHEDULING

- Schedulers and scalers
- Birth of a pod, the scheduler role
- Fine-tuning the Kubernetes scheduler
- Hard and soft decision constraints

2 - CREATE YOUR KUBERNETES CUSTOM SCHEDULER

- Multiple schedulers and schedulerName selectors
- Main loop and relevant code sections

3 - CODE OVERVIEW AND DEMO

4 - IMPLEMENTED & PROPOSED IMPROVEMENTS

- Metrics cache
- Failsafe code
- Adding constraints and variables

5 - FAILSAFE CODE DEMO

6 - Q&A

Sysdig

Building a Kubernetes Scheduler using Custom Metrics

1 - INTRO TO KUBERNETES POD SCHEDULING

- Schedulers and scalers
- Birth of a pod, the scheduler role
- Fine-tuning the Kubernetes scheduler
- Hard and soft decision constraints

2 - CREATE YOUR KUBERNETES CUSTOM SCHEDULER

- Multiple schedulers and schedulerName selectors
- Main loop and relevant code sections

3 - CODE OVERVIEW AND DEMO

4 - IMPLEMENTED & PROPOSED IMPROVEMENTS

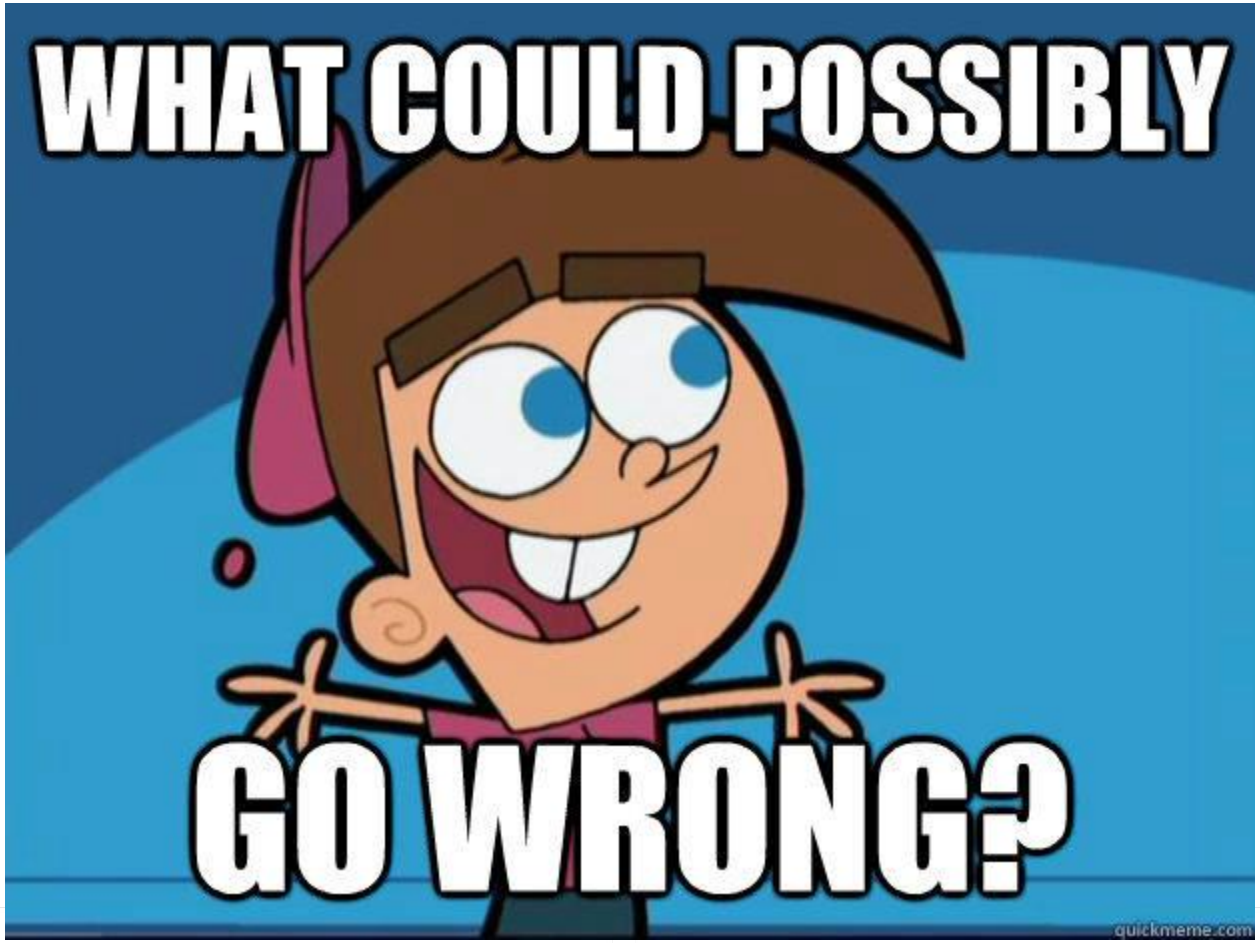
- Metrics cache
- Failsafe code
- Adding constraints and variables

5 - FAILSAFE CODE DEMO

6 - Q&A

Sysdig

Creating your own scheduler...



Improvements - Metrics cache

Metrics Cache

- New deployment, you may have 10 identical pods waiting
- Schedulers need to be fast, API calls are expensive
- Implement a cache
  - Timestamp metrics to decide if they are still “fresh” enough, reuse fresh metrics
  - Adaptive obsolescence time in response to metrics change rate

Improvements - Failsafe plans

Failsafe mechanisms

- Metrics API is non responsive
  - Cache may mitigate this for a short period of time, eventually, you have to throw an exception

- No good candidate node
  - Relax requisites?

Improvements - Failsafe plans

Failsafe mechanisms

- Decision timeout
  - Missing the point of optimization
  - Start a timeout clock and catch timeout event

- Multiple exceptions, code has bugs
  - You will always need a plan B, pending forever is not good

Improvements - Failsafe plans

Failsafe mechanisms

- Delegate to default scheduler (next demo)
- Requeueing: You can return an error condition requesting to get the Pod back in the queue

Building a Kubernetes Scheduler using Custom Metrics

1 - INTRO TO KUBERNETES POD SCHEDULING

- Schedulers and scalers
- Birth of a pod, the scheduler role
- Fine-tuning the Kubernetes scheduler
- Hard and soft decision constraints

2 - CREATE YOUR KUBERNETES CUSTOM SCHEDULER

- Multiple schedulers and schedulerName selectors
- Main loop and relevant code sections

3 - CODE OVERVIEW AND DEMO

4 - IMPLEMENTED & PROPOSED IMPROVEMENTS

- Metrics cache
- Failsafe code
- Adding constraints and variables

5 - FAILSAFE CODE DEMO

6 - Q&A

Sysdig

Improvements - Honor constraints

Adding more variables to the mix

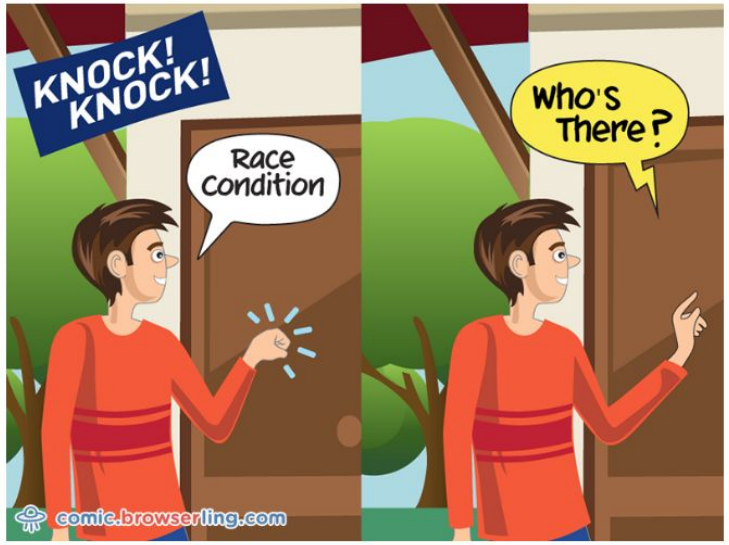
- NoSchedule, NotReady, Unreachable, etc labels
- Using hardware pressure as an additional factor
- Software affinities and anti-affinities

Improvements - Modify Pod YAML “on the fly”

- **Modify existing YAML definitions**
  - MutatingAdmissionWebhook (beta in 1.9)
    - Istio uses this to inject sidecar containers
  - After auth, before etcd persistence
    - Pod definition is still mutable at this point

Pod scheduler - Improvements V

- **Race conditions**
  - Lock objects
  - Leader election



Thank You.

Questions?
Mail: mateo.burillo@sysdig.com
Twitter: @mateobur
Want moar K8S stuff?:
https://sysdig.com/newsletters/
Code repo:
https://github.com/draios/kubernetes-scheduler