# Getting The Most Out Of Kubernetes

Optimizing Cluster Resource Allocation in Production

# Intro

- What are resource Requests and Limits
- How do you set good Requests and Limits
- Tools 🛠️

# Harrison Harnisch

Staff Software Engineer @ ZEIT

@hjharnis

# Case Study: Links Service

- Preexisting endpoint in the monolith
- Serves the number of times a link is shared within Buffer

# Case Study: Links Service

- Settled on a simple design using Node and DynamoDB

# Case Study: Links Service

- Deployed the service to Kubernetes (4 replicas)
- Manually verified that the service was operational

# 1%

1% ➡️ 10%

1% ➡️ 10% ➡️ 50%

1% ➡️ 10% ➡️ 🔥

# Case Study: Links Service

- Scaled up replicas (5x - 20 pods)
- Helped, but pods still repeatedly dying

# Back to 0%

# Case Study: Links Service

- I had copied and pasted a `Deployment` from another service
- The `Deployment` included resource limits
- `kubectl describe` was reporting `OOMKilled`

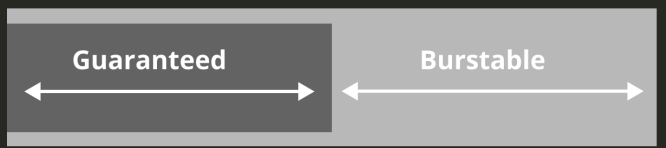
# Resource Limits

- Upper limit on container resources
- Containers run with unbounded CPU and memory limits
- Kubernetes will restart containers when limits are exceeded

# Resource Requests

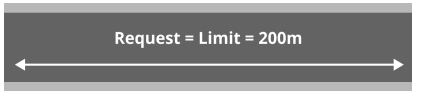
- Allocated resources for a container
- Containers may be throttled back down to request when exceeded
- Matches limit if no requests set explicitly

# Quality Of Service (QOS)




# QOS: Guaranteed

- Highest Priority
- Limit = Request



# QOS: Burstable

- Requested resource is guaranteed
- Limit > Request

# QOS: BestEffort

- YOLO ¯_(ツ)_/¯

# QOS: BestEffort

- Lowest Priority
- Can use any amount of free resources

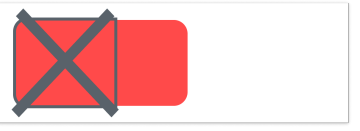
# How do we set CPU and Memory limits?

# Optimal* Limits

- Pods have enough resources to complete their task
- Nodes run maximum number of pods

# Under/Over/Optimal* Resource Allocation

# Under-allocation



# Overallocation

# Overallocation is *tricky*

It becomes a problem when you *scale up* replicas

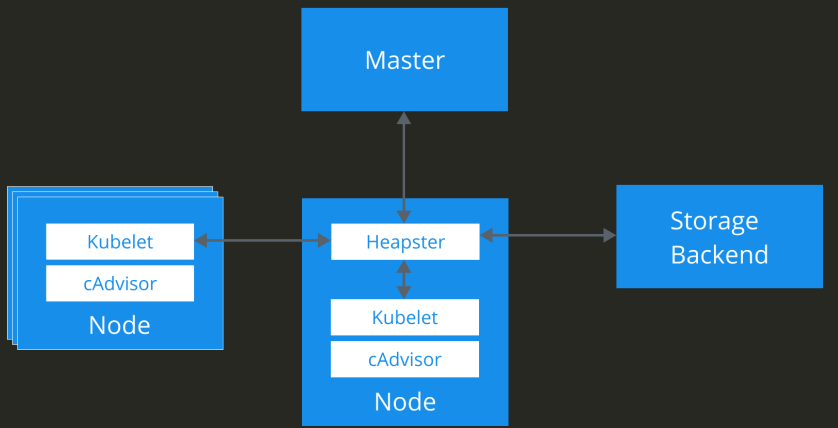
VS

That's one extra pod that could be running

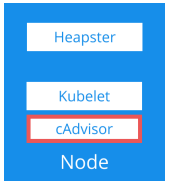
Optimal*

# Kubernetes Monitoring

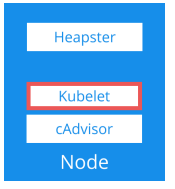
# kubernetes Monitoring



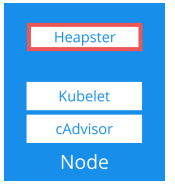
# cAdvisor



# Kubelet



# Heapster



# PSA: Heapster Is Deprecated

- Deprecation in v1.11
- Removal in v1.13
- Suggest migrating to Metrics Server + Prometheus

https://github.com/kubernetes/heapster/blob/master/docs/deprecation.md#heapster-deprecation-timeline

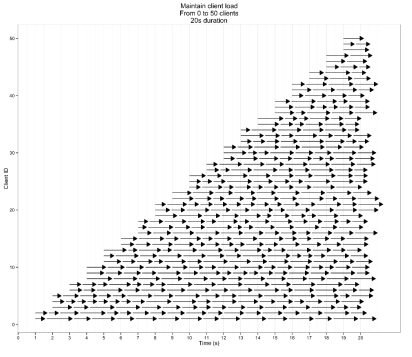
# Setting Limits and Requests

- Goal: Understand what **one pod** can handle
- **Use limits** during testing
- Start with a very conservative set of limits
- Only change one thing at time and observe changes

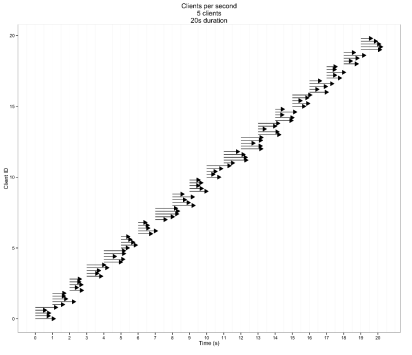
```
# limits might look something like
replicas: 1
...
cpu: 100m # ~1/10th of a core
memory: 50Mi # 50 Mebibytes
```

# Testing Strategies

# Ramp Up Test


Maintain client load
From 0 to 50 clients
20s duration
Client ID
Time (s)

# Duration Test



# Demo

Setting Limits For etcd

# Keep A Fail Log

# Some Observed Failure Modes

- Memory is slowly increasing
- CPU is pegged at 100%
- 500s
- High response times
- Dropped Requests
- Large variance in response times

# Case Study: Links Service

Lessons Learned

# It's About Increasing Predictability

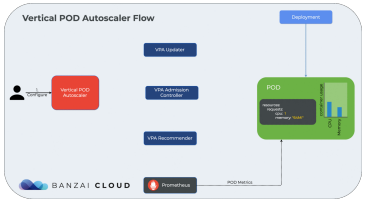
And Getting More Sleep

# Horizontal Pod Autoscaler (HPA)

- Change Deployment replica count based on a metric (scale up or down)
- Custom metrics from Prometheus, Azure Adapter, and StackDriver
- Well supported and feature rich
  - Cooldown/Delay Settings
  - Multiple Metrics
  - External Metrics

# Vertical Pod Autoscaler (VPA)

- Change Pod resource requests in place
- Pod restart is required to change limits
- **Alpha** Feature



# Looking Ahead: Kubernetes Developer Tools

- Tooling for aggregate metrics are fantastic (Prometheus, Datadog, etc.)
- Need high resolution tools to analyze individual Deployments, Pods and containers

# KubeScope CLI 🔬

https://github.com/hharnisc/kubescope-cli


tmux
k
k8s_test-server_docker-test_default_643d3b34-da1a-11e8-88a5-74e4c9f4a959_2
cpu percent
0.0
0.0
0.0
0.0
0.0
0.0
16:33:36 16:33:46 16:33:57 16:34:07 16:34:17 16:34:28 16:34:38 16:34:48
memory percent
0.30%
0.3
0.2
0.2
0.1
0.1
0.0
16:33:36 16:33:46 16:33:57 16:34:07 16:34:17 16:34:28 16:34:38 16:34:48
carbon-wolf→ kubescope : cli ✘ :★ ▷ httpstress -c 5 -g http://localhost:3000
[kubescope0:..cts/kubescope*
"carbon-wolf" 11:34 31-Oct-18

# Questions?