# Load Testing Kubernetes

How To Optimize Your Cluster Resource Allocation in Production

## Harrison Harnisch

Senior Software Engineer @ Buffer

**@hjharnis**

## Case Study: Links Service

- Preexisting endpoint in our 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%

$1\% \rightarrow 10\% \rightarrow 50\%$

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

- Limits can be set on both CPU and memory utilization
- Pods run with unbounded CPU and memory limits
- Kubernetes will restart containers when limits are exceeded

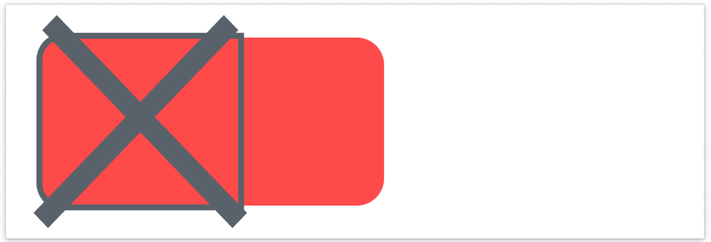
## How do we optimally set CPU and Memory limits?

## Optimal Limits

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

## Under/Over/Even Resource Allocation

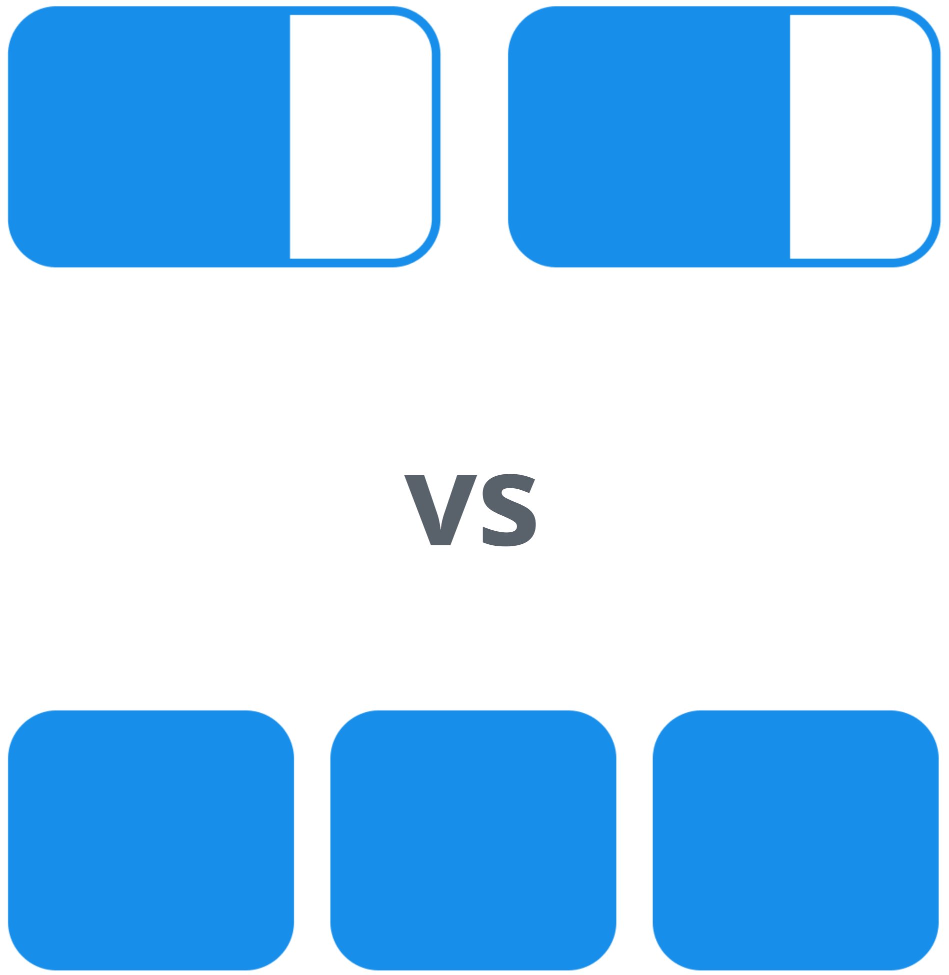
## Under-allocation



## Overallocation

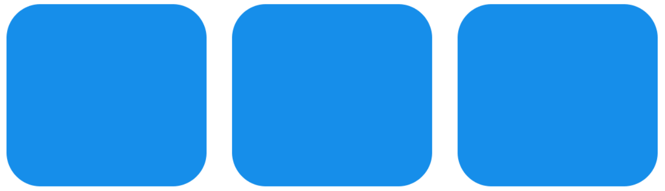
## Overallocation is *tricky*

It becomes a problem when you *scale up* replicas


vs

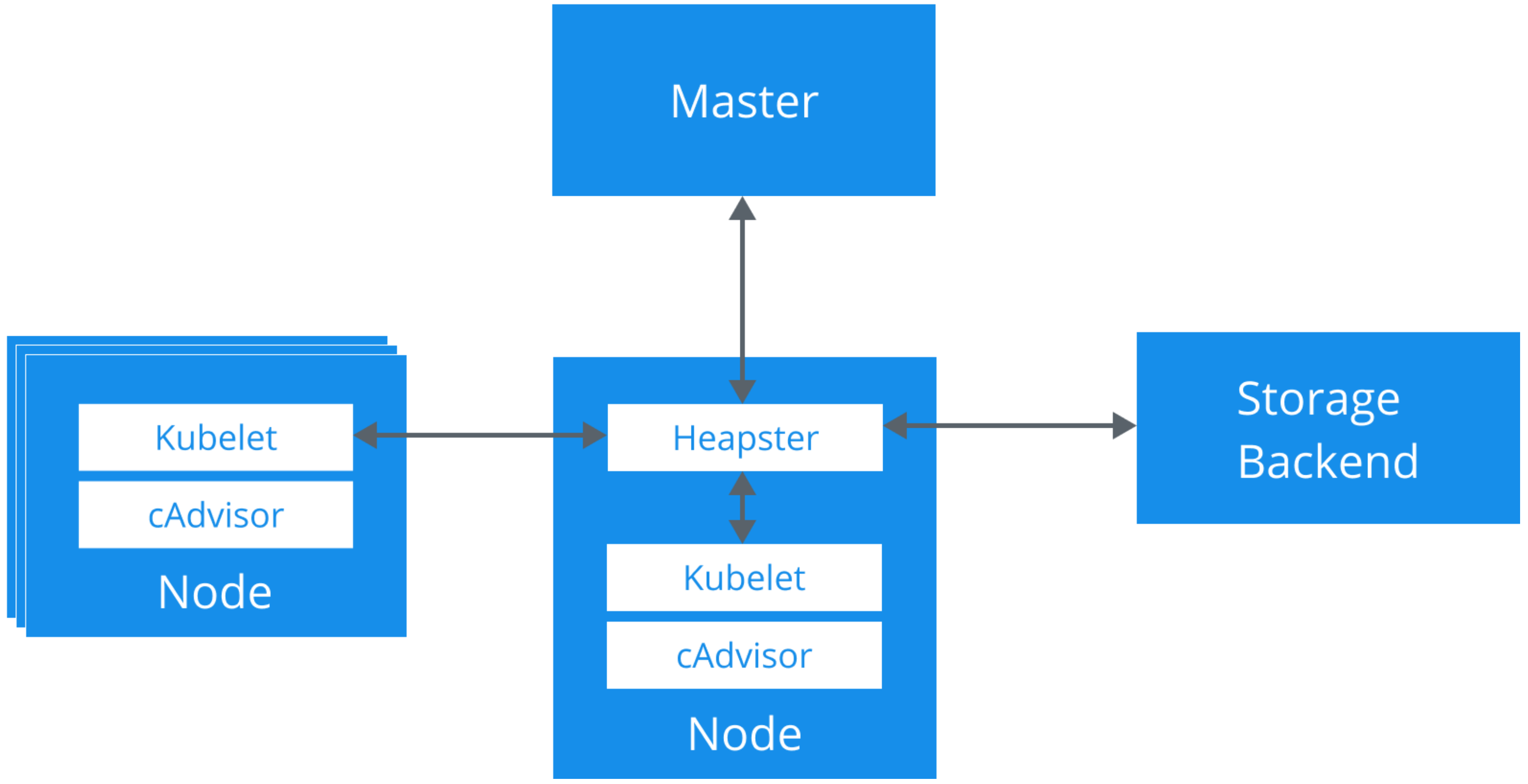
That's one extra pod that could be running

## Even

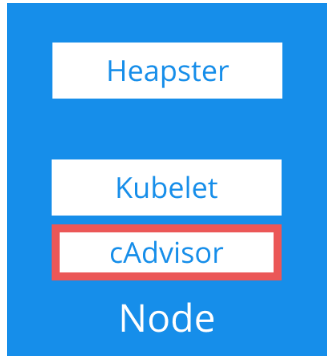


## Kubernetes Monitoring

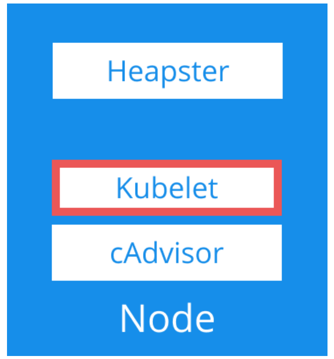
# kubernetes Monitoring



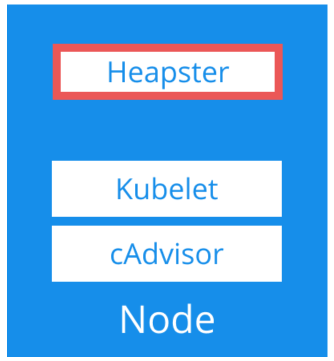
# cAdvisor



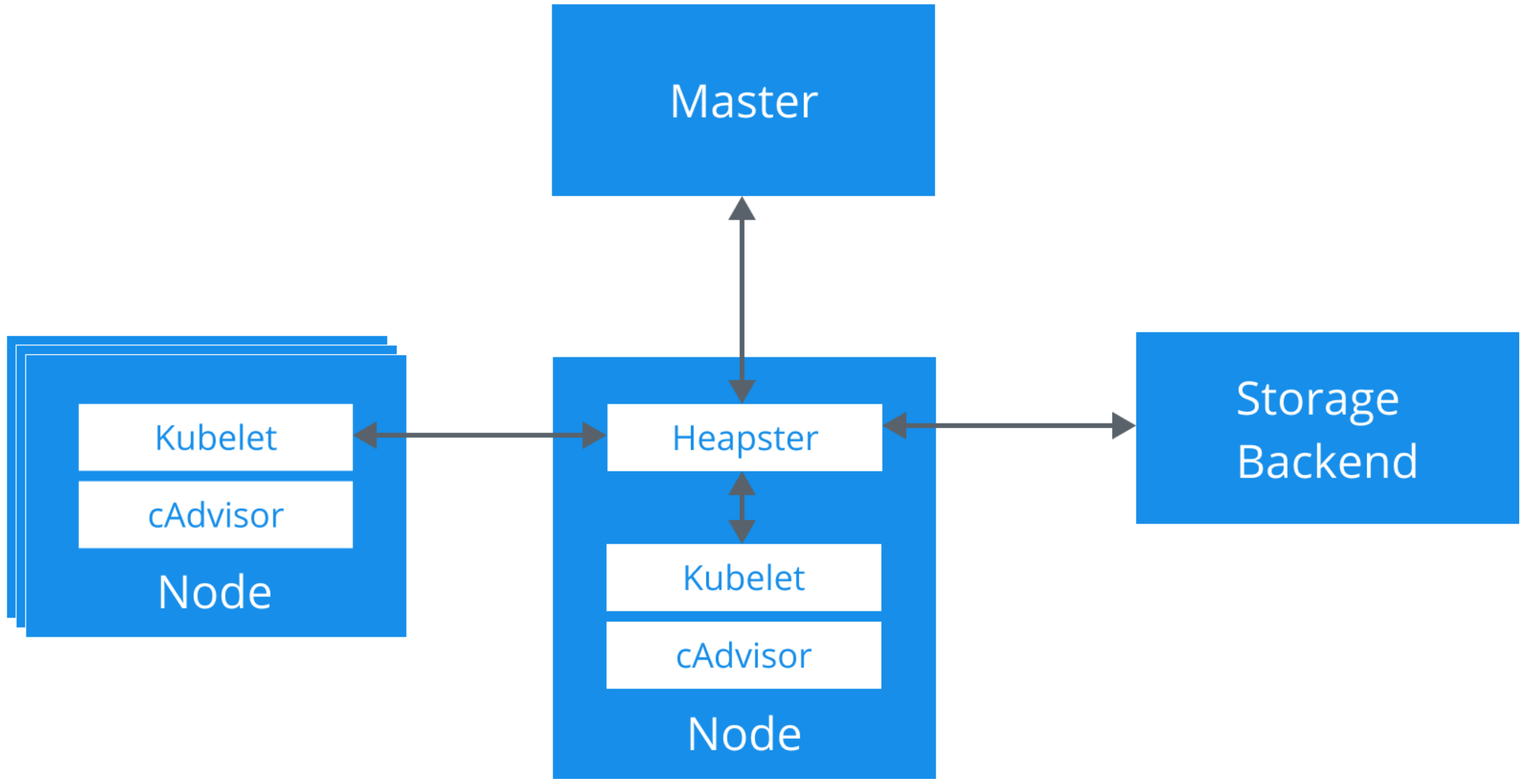
## Kubelet



# Heapster



# kubernetes Monitoring



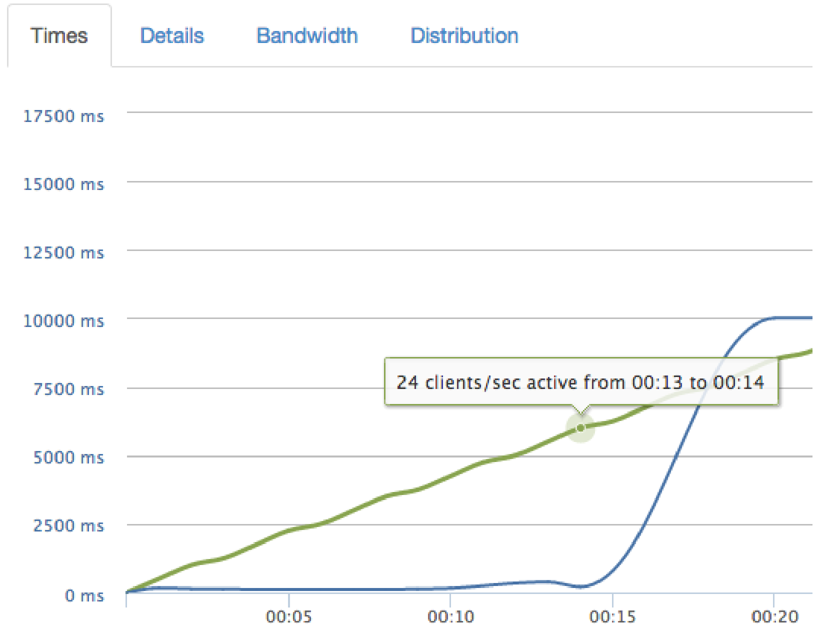
## Setting Limits

- Goal: Understand what **one pod** can handle
- 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
```

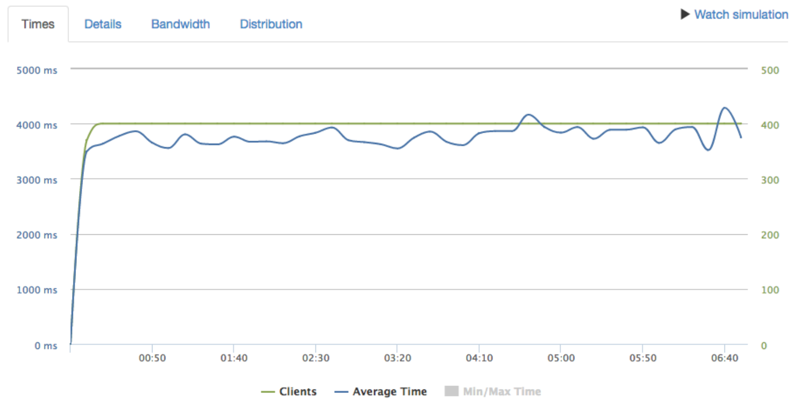
## Testing Strategies

# Ramp Up Test



Times
Details
Bandwidth
Distribution
17500 ms
15000 ms
12500 ms
10000 ms
7500 ms
5000 ms
2500 ms
0 ms
00:05
00:10
00:15
00:20
24 clients/sec active from 00:13 to 00:14

# 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
- Large variance in response times
- Dropped Requests

## Case Study: Links Service

Lessons Learned

## It's About Increasing Predictability

And Getting More Sleep

## Looking Ahead: Kubernetes

- Amazing at monitoring a cluster
- Gap when observing a pod or container

Questions?