
KubeCon
CloudNativeCon
North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022

KubeCon | CloudNativeCon
North America 2022

BUILDING FOR THE ROAD AHEAD

DETROIT 2022

SIG Autoscaling

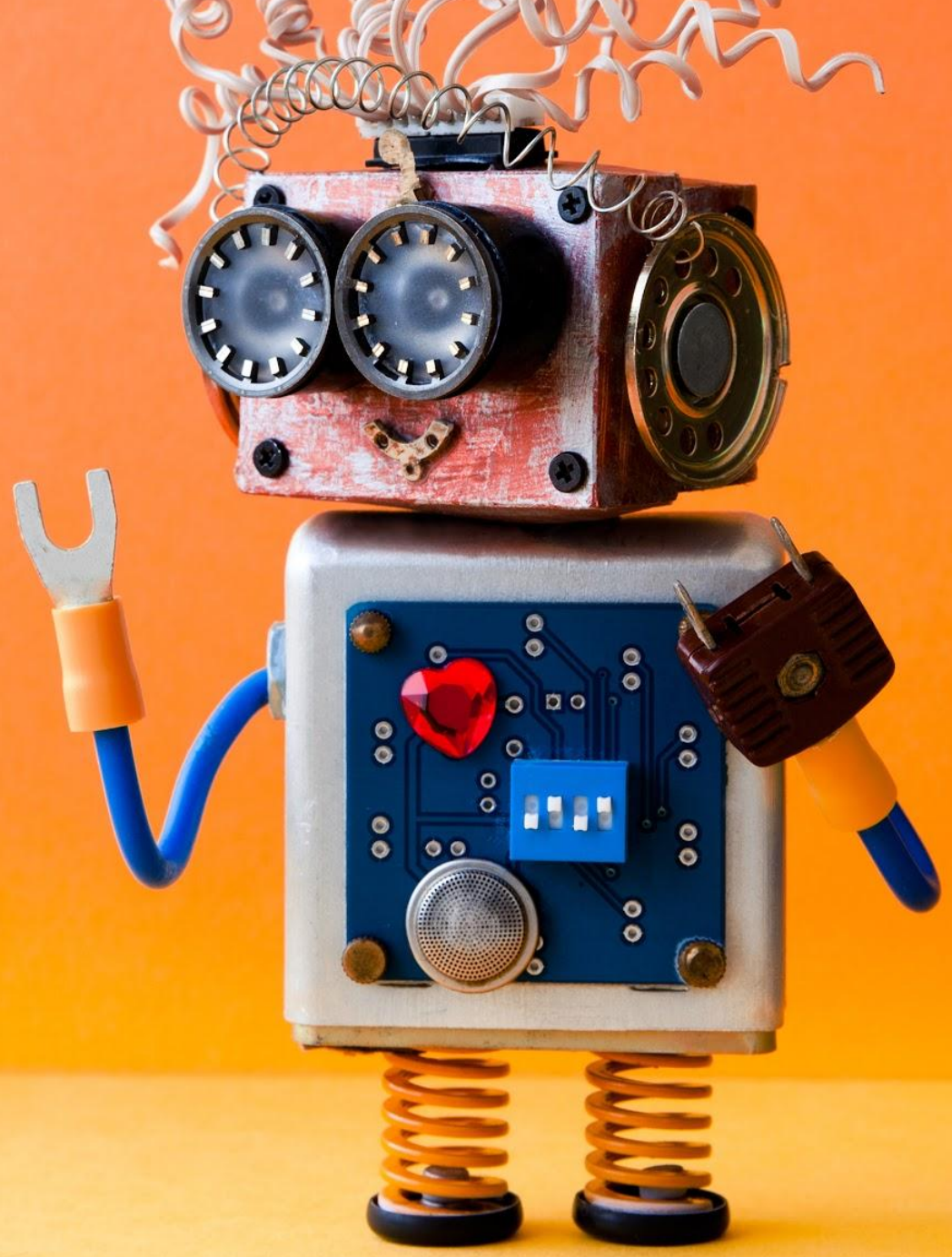
Updates and Feature Highlights

Marcin Wielgus
Diego Bonfigli

What is
SIG-Autoscaling

Owned
Components

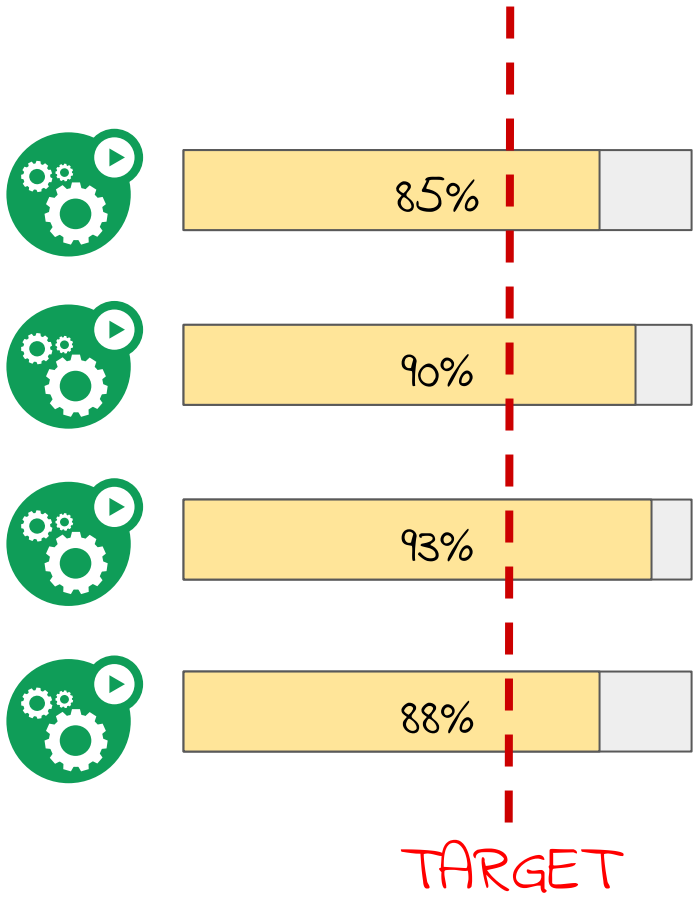
Horizontal Pod Autoscaler



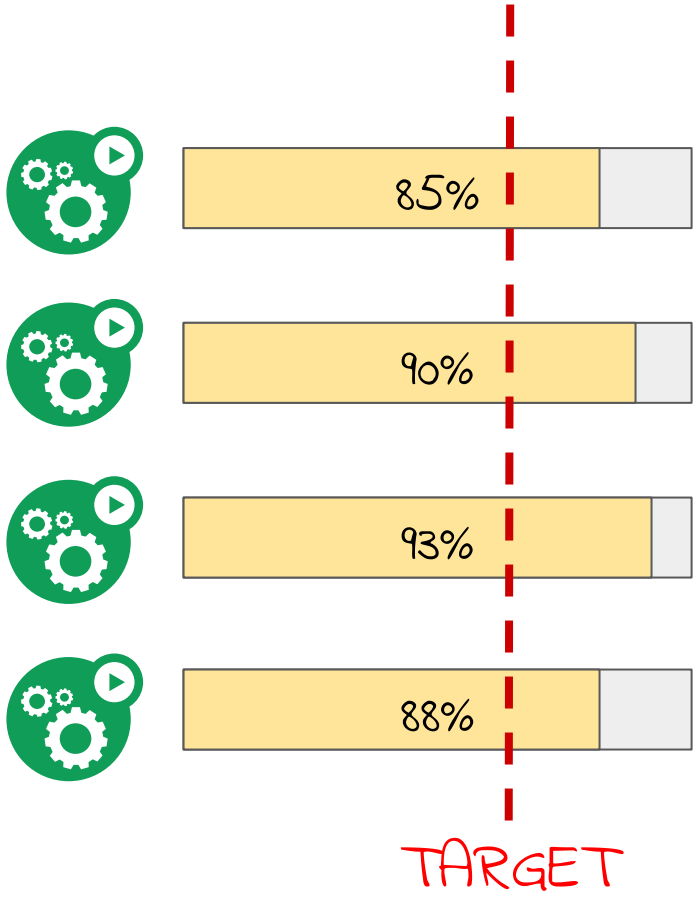
Horizontal Pod Autoscaler

- Based on load-like metrics
  - CPU usage
  - Custom metrics like:
    - Query per second
    - Queue length

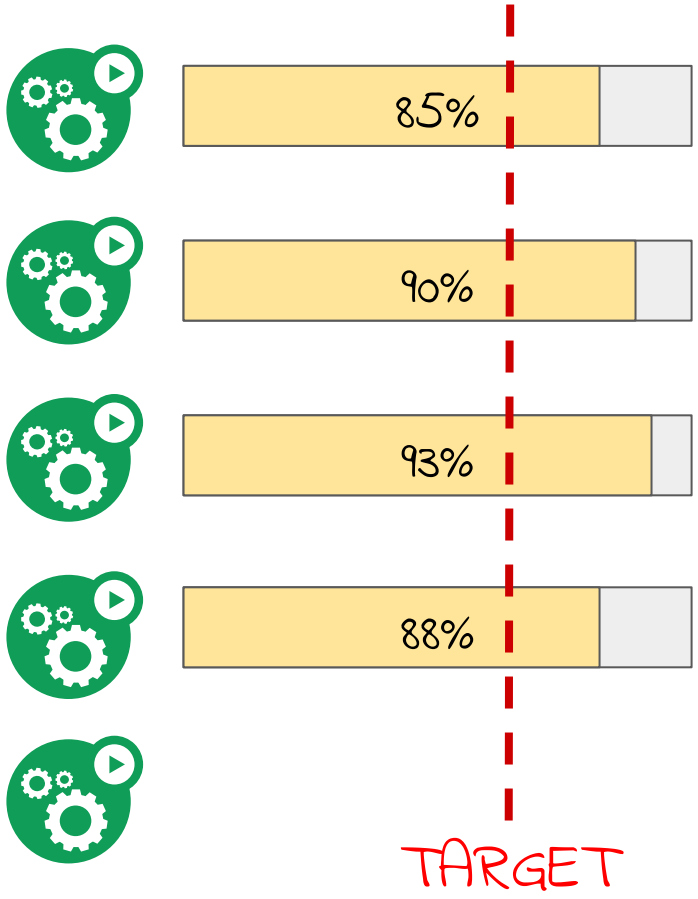
- Adds and removes pod replicas to keep per pod load close to the desired target value.



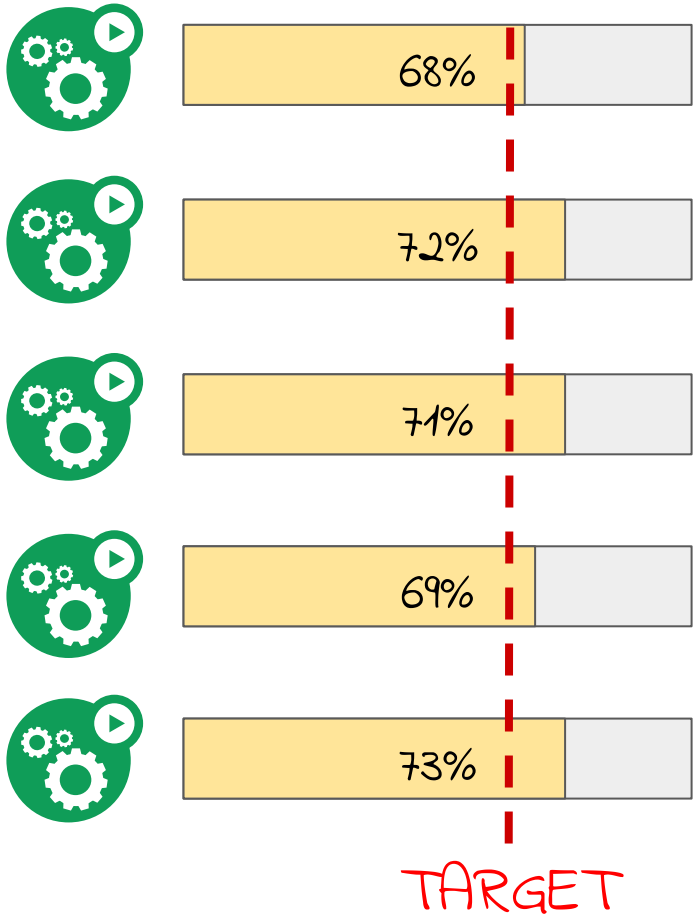
Horizontal Pod Autoscaler



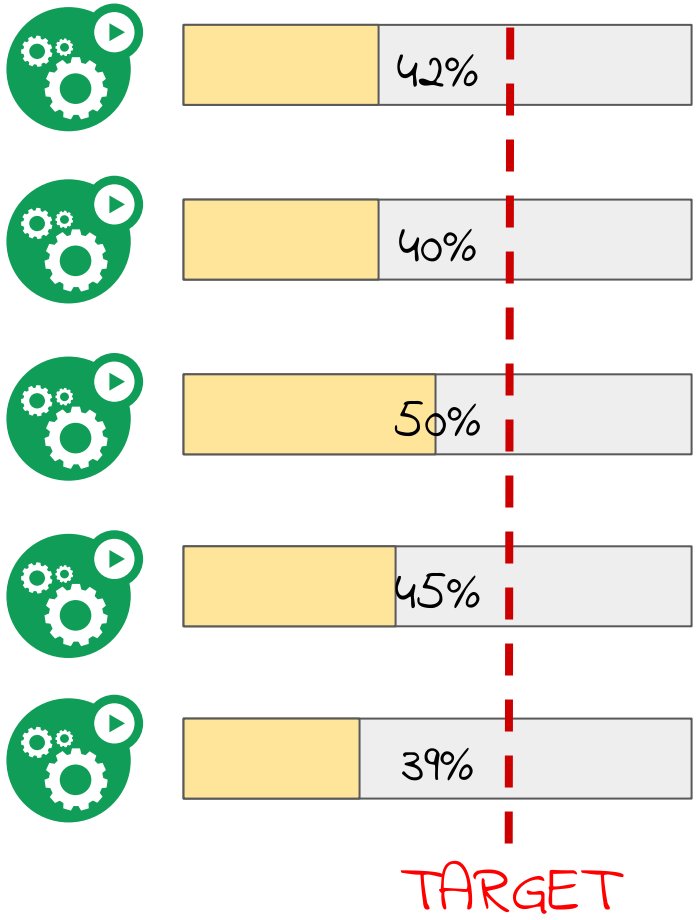
Horizontal Pod Autoscaler



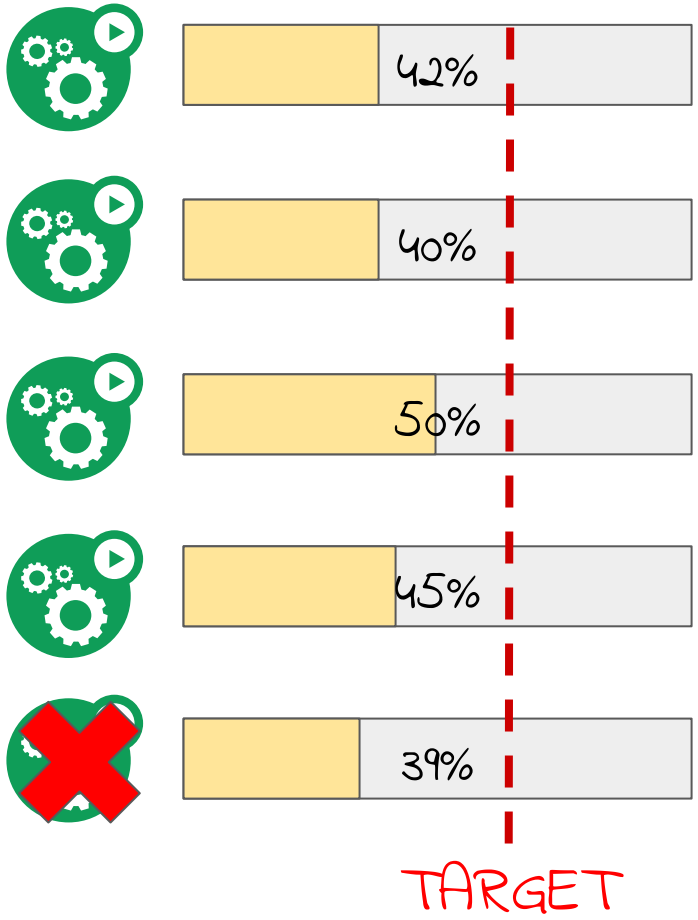
Horizontal Pod Autoscaler



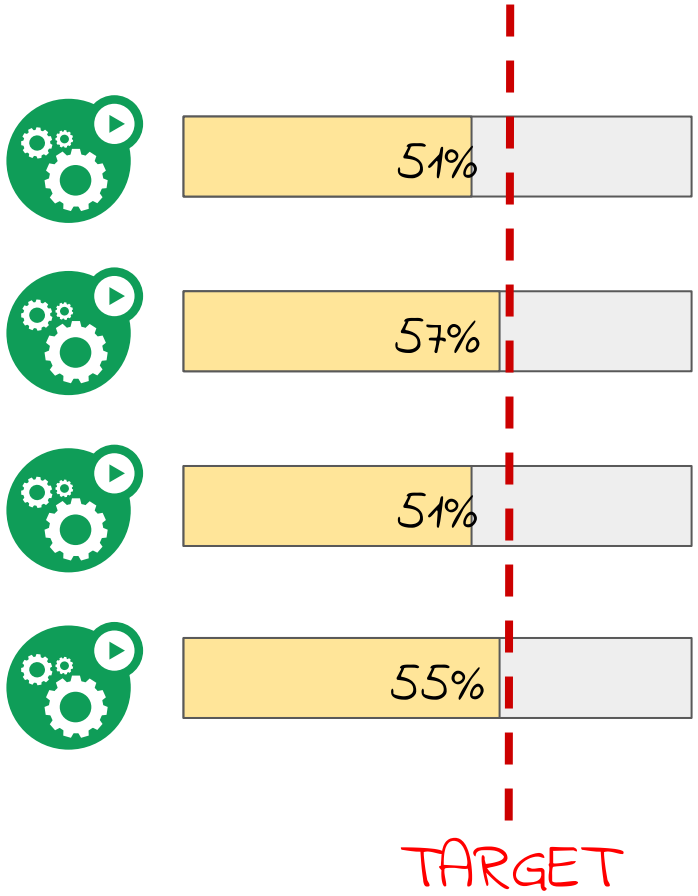
Horizontal Pod Autoscaler



Horizontal Pod Autoscaler

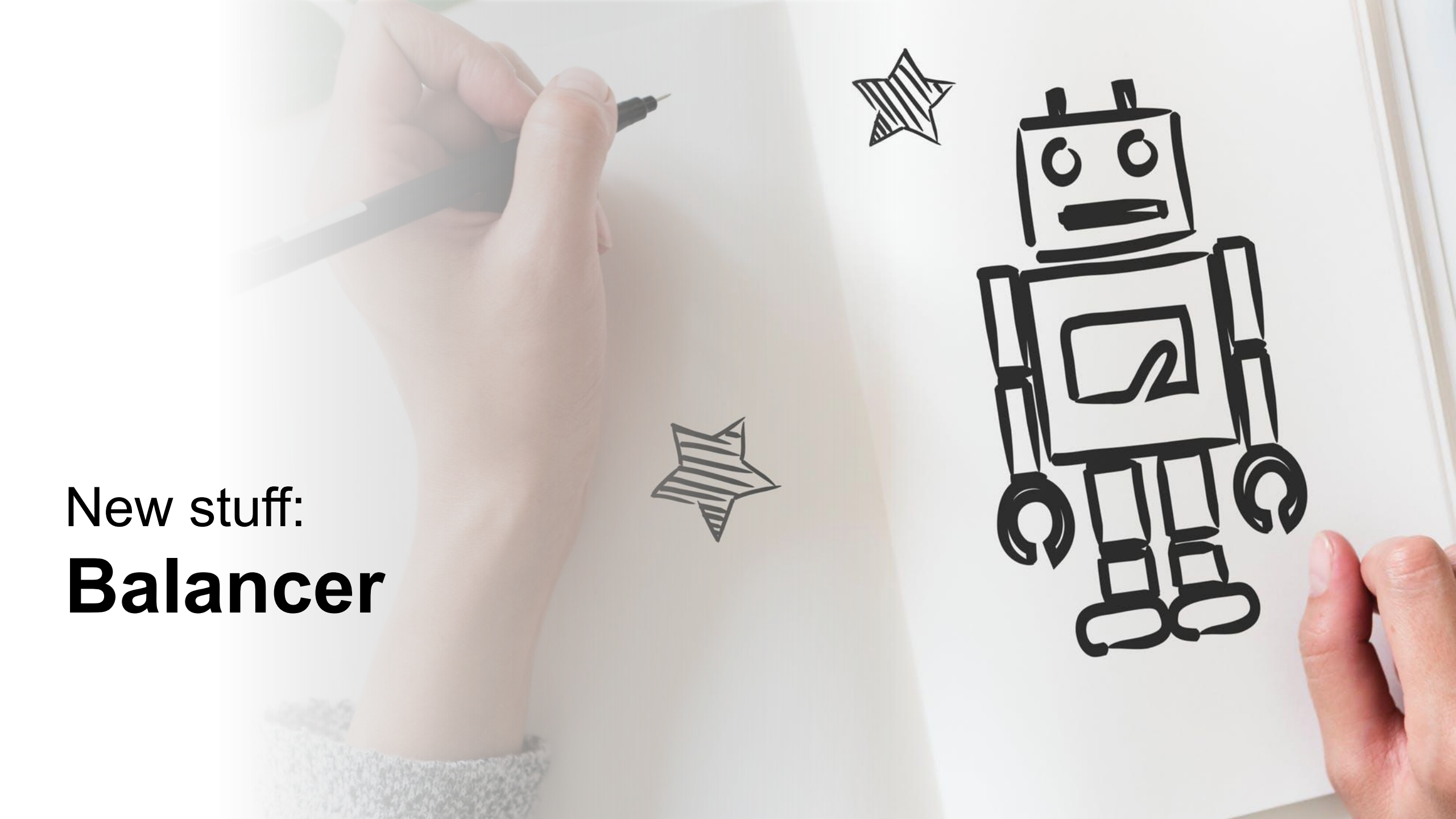


Horizontal Pod Autoscaler



HPA - recent and upcoming changes

- API Graduation to V2
  - v2beta1 went away in 1.25
  - v2beta2 goes away in 1.26
- HPA Controller gained multithreading support (1.26)
  - `--concurrent-horizontal-pod-autoscaler-syncs`
- Improved behavior for targets > 100% (1.26)
- Scale to 0 support for custom metrics (hopefully in 1.26)
- Explicit dry run mode (probably post 1.26)



New stuff:

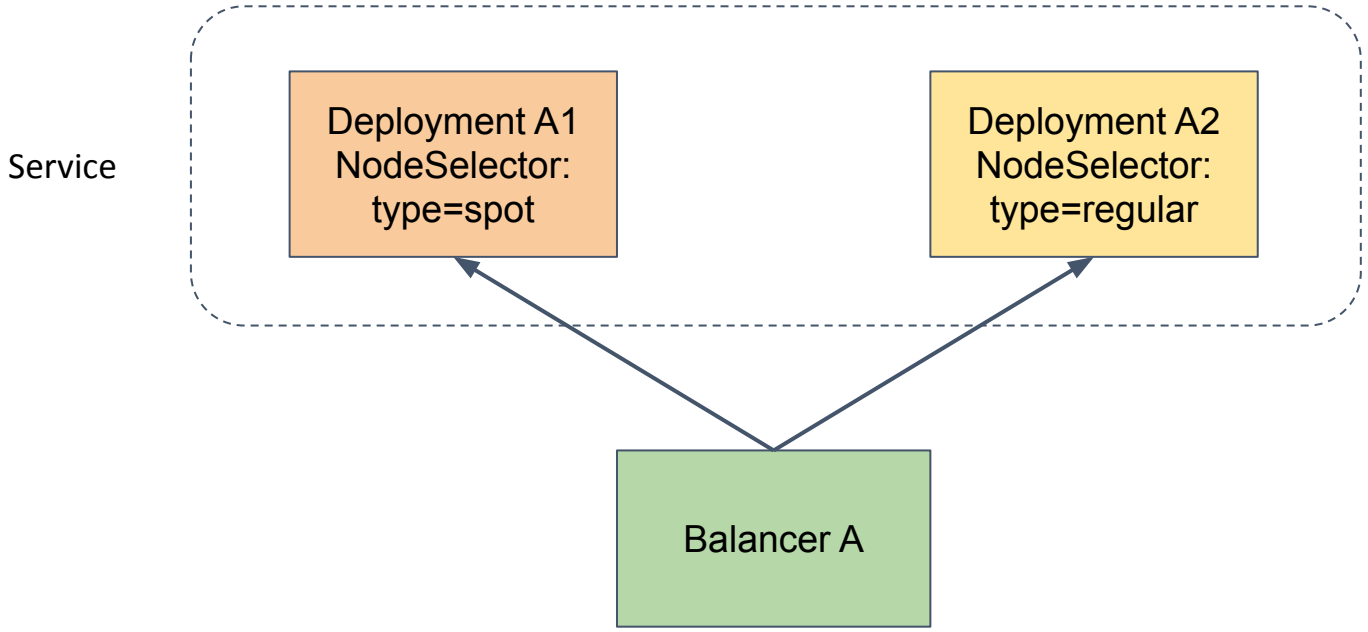
Balancer

Balancer

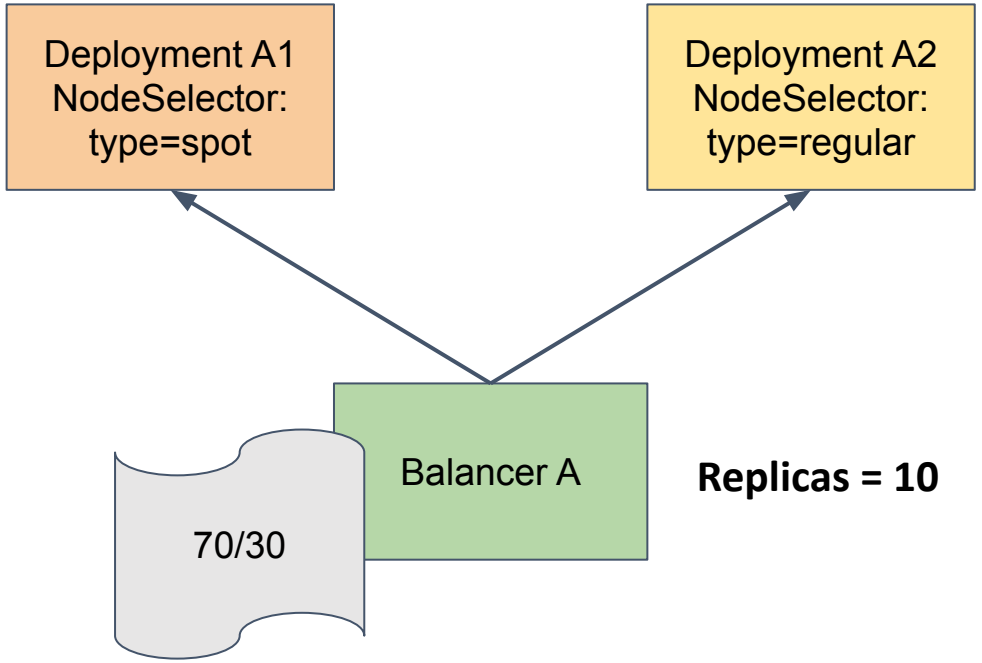
Problem:

- How to ensure equal spreading of pods in 3 zones of a region. With fallback and rebalance?
- How to split pods 70%-30% between spot/preemptible and regular vms? And temporarily use only regulars if spots are not available.
- How to consume nodes with negotiated rates first?
- How to horizontally and vertically autoscale such deployments and make CA work well with them?

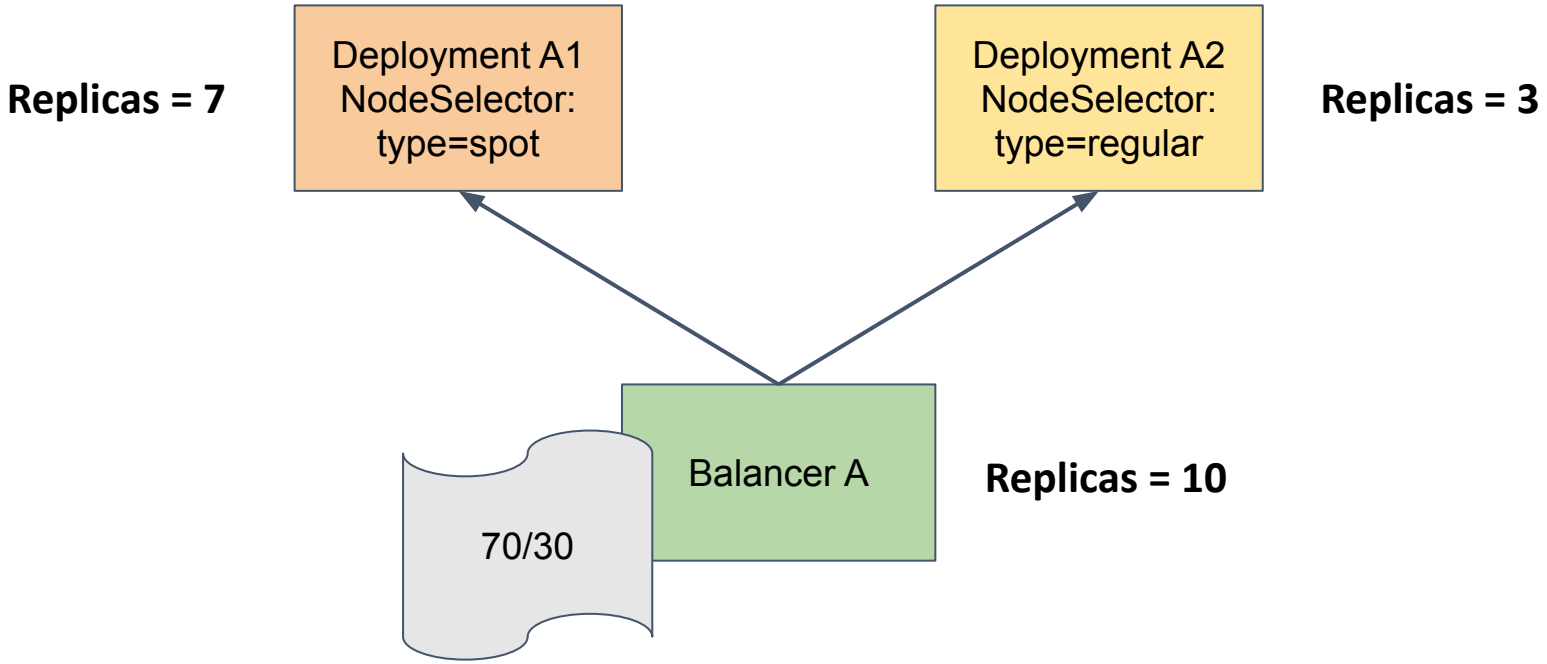
Balancer


Service
Deployment A1
NodeSelector:
type=spot
Deployment A2
NodeSelector:
type=regular
Balancer A

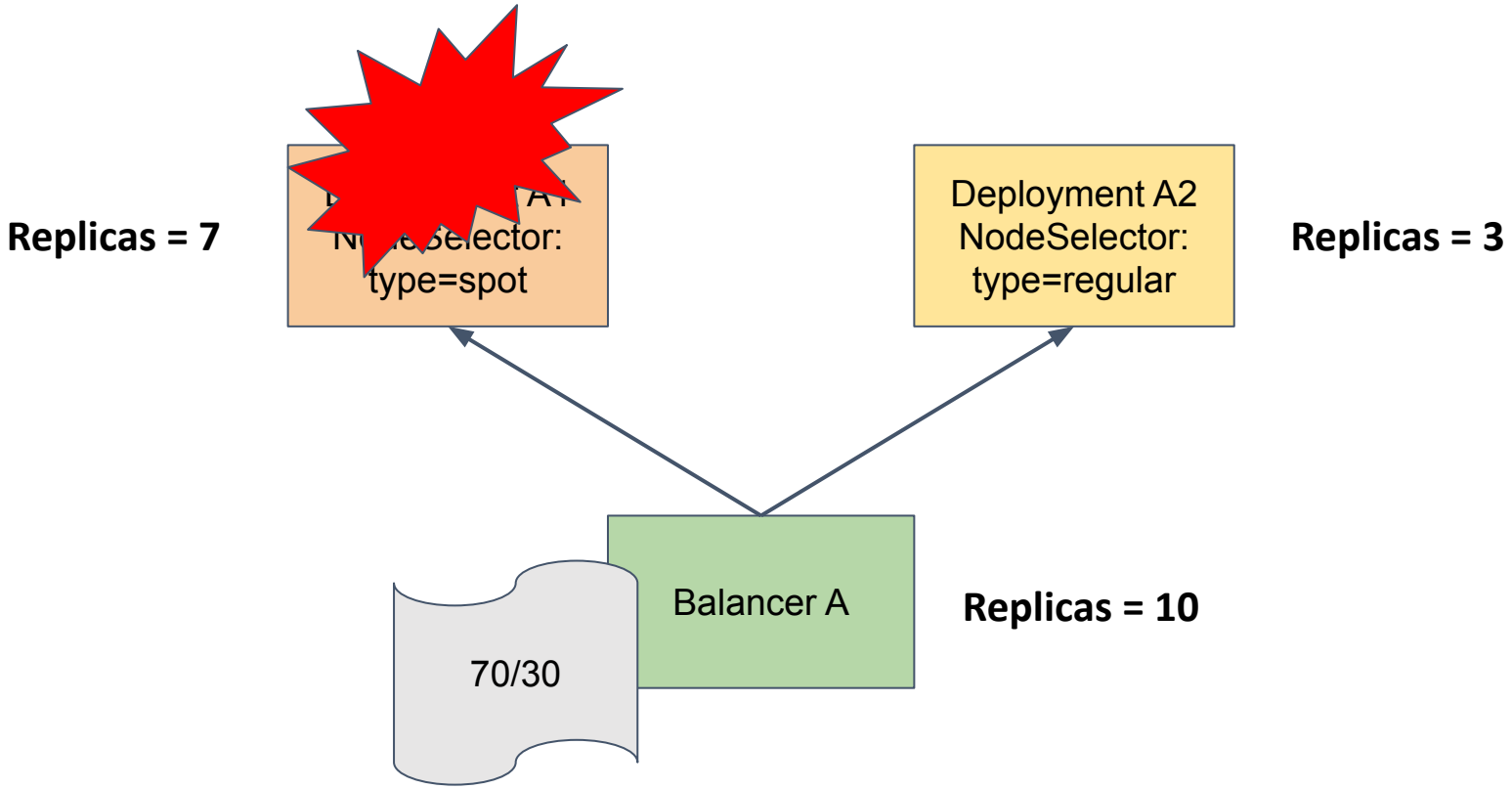
Balancer


Deployment A1
NodeSelector:
type=spot
Deployment A2
NodeSelector:
type=regular
Balancer A
70/30
Replicas = 10

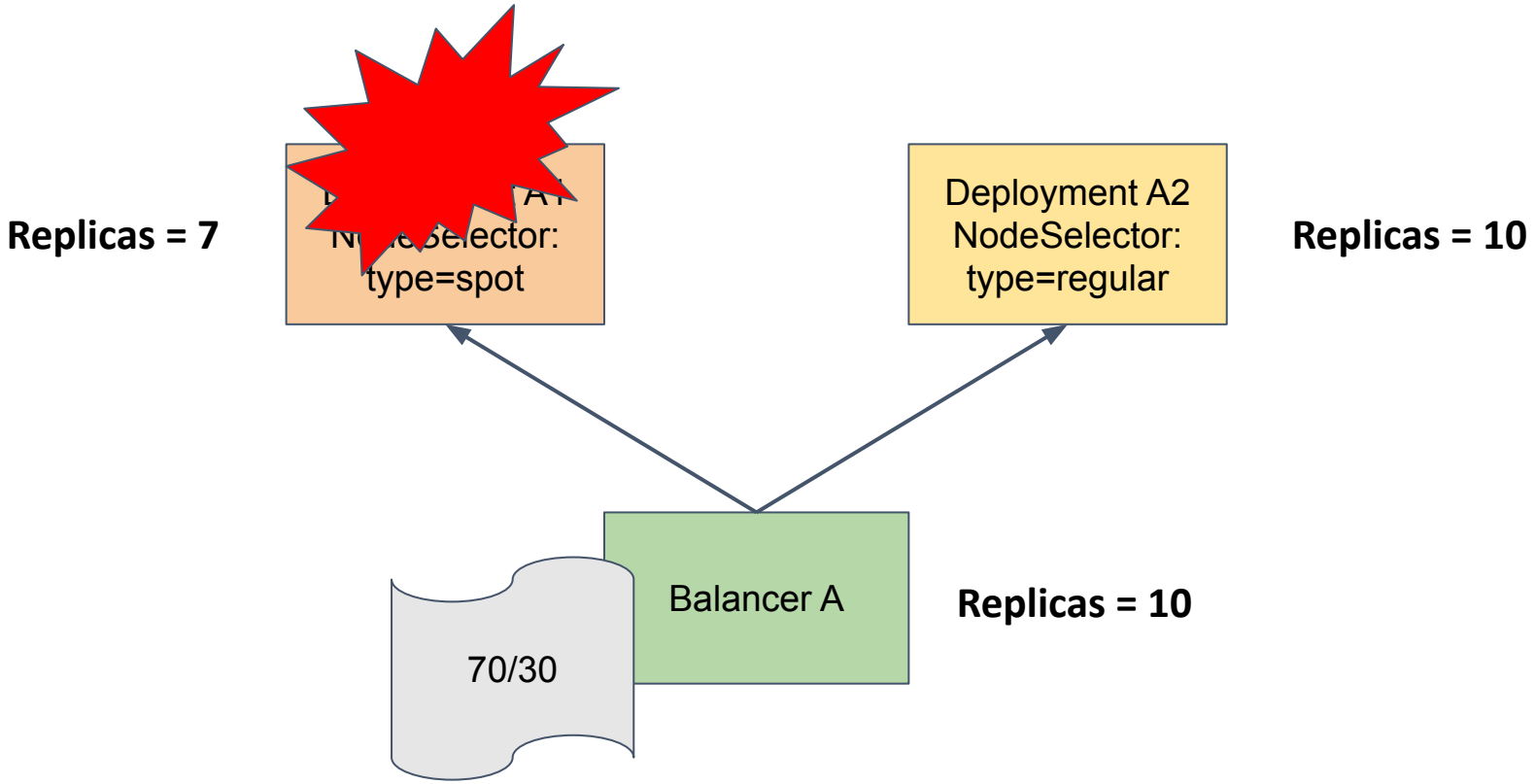
Balancer


Replicas = 7
Deployment A1
NodeSelector:
type=spot
Deployment A2
NodeSelector:
type=regular
Replicas = 3
Balancer A
Replicas = 10
70/30

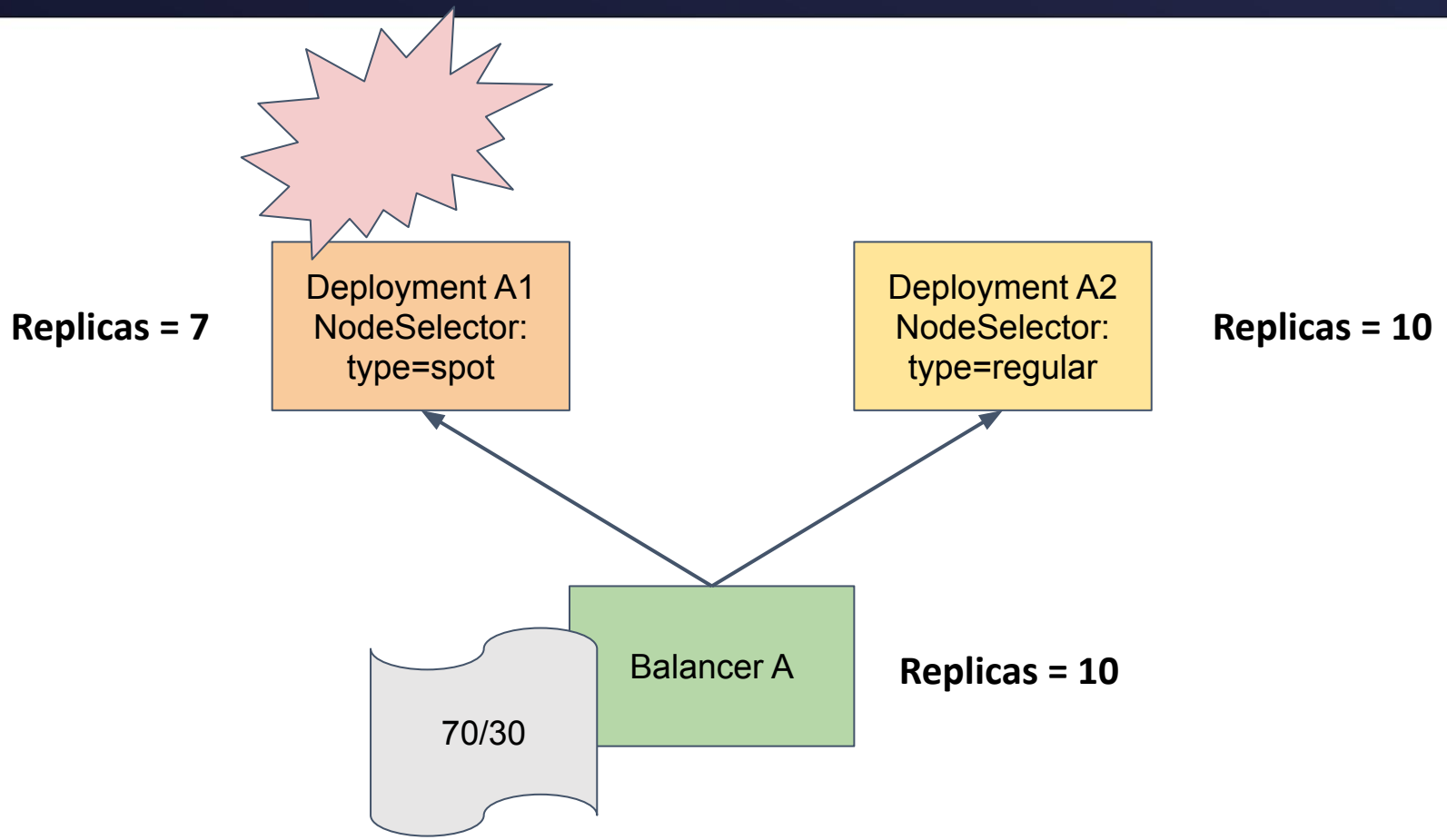
Balancer


Replicas = 7
type=spot
Deployment A2
NodeSelector:
type=regular
Replicas = 3
Balancer A
Replicas = 10
70/30

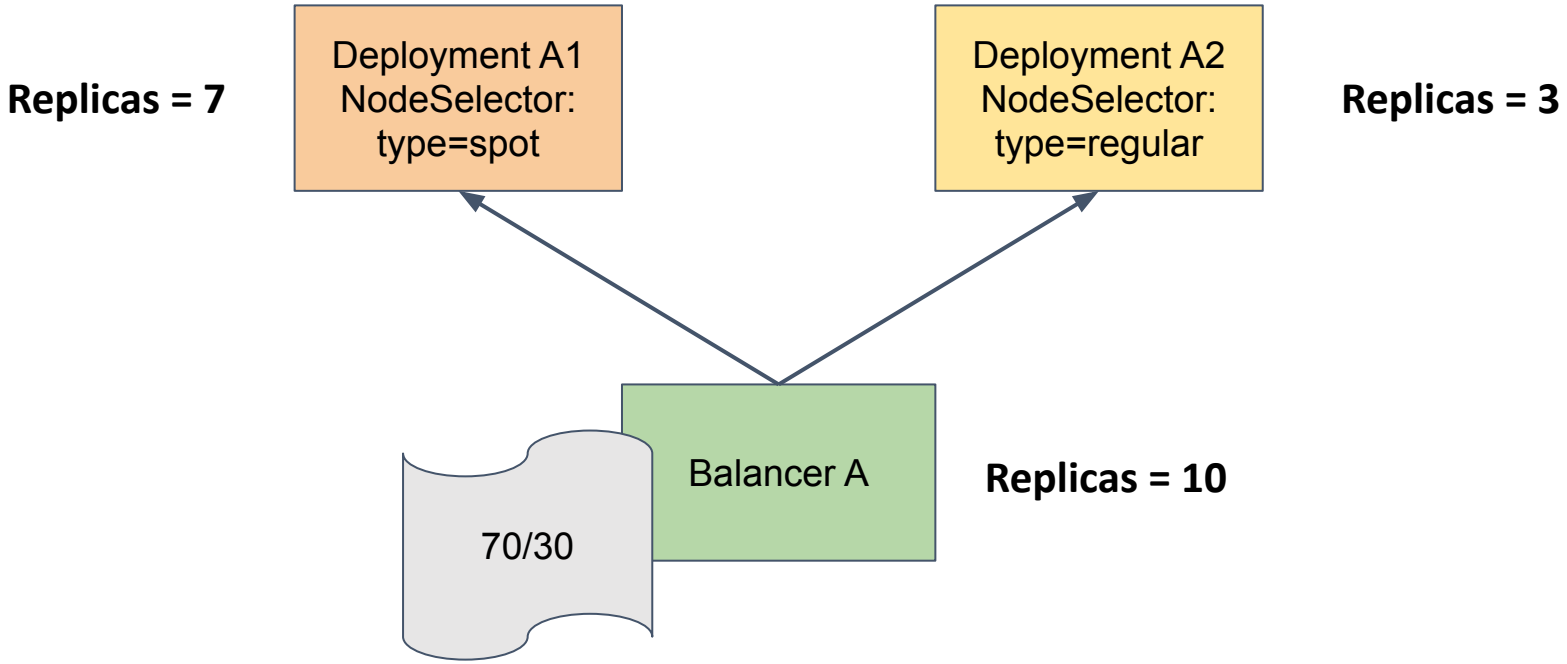
Balancer


Replicas = 7
type=spot
Deployment A2
NodeSelector:
type=regular
Replicas = 10
Balancer A
Replicas = 10
70/30

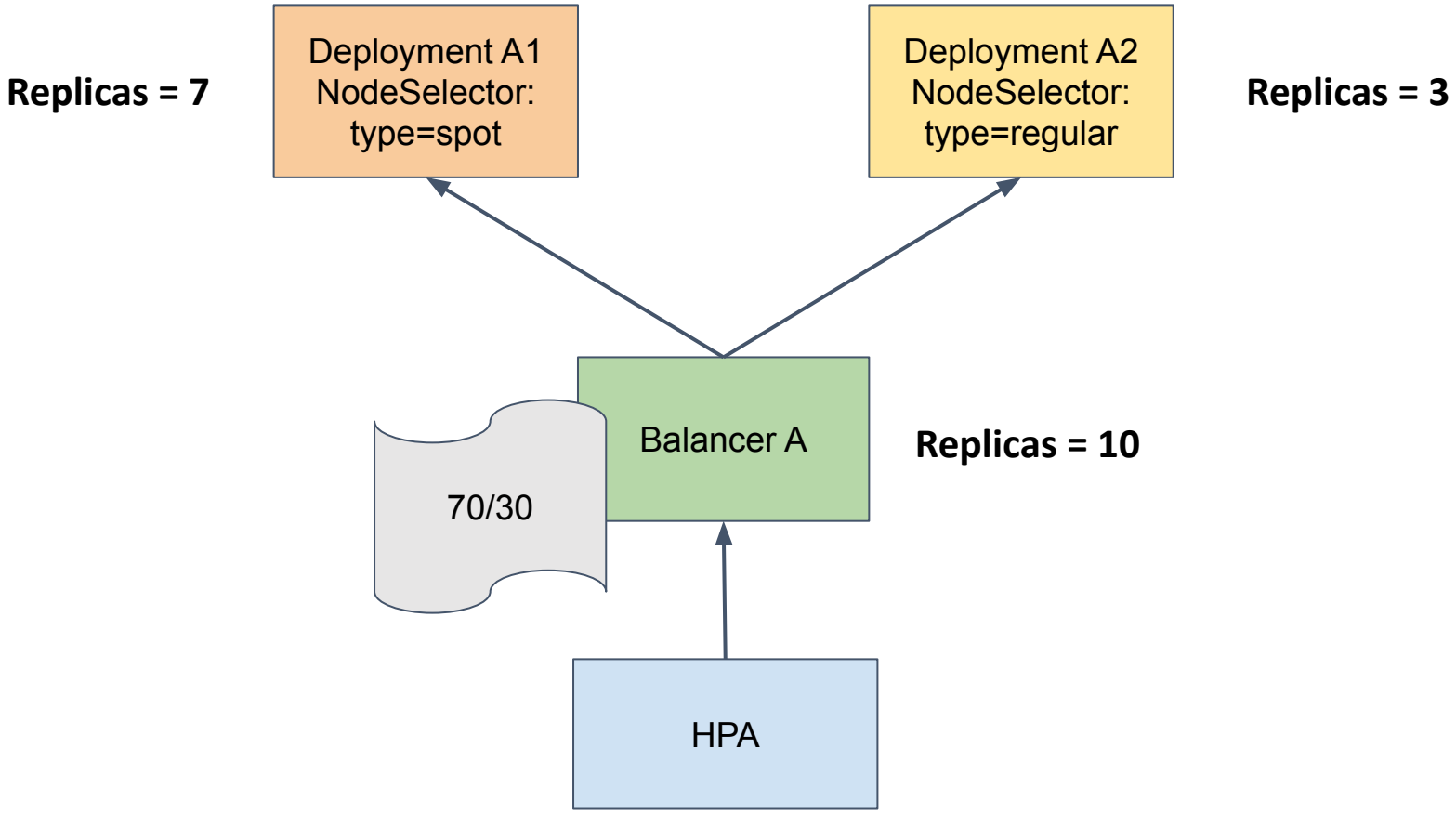
Balancer


Replicas = 7
Deployment A1
NodeSelector:
type=spot
Deployment A2
NodeSelector:
type=regular
Replicas = 10
Balancer A
Replicas = 10
70/30

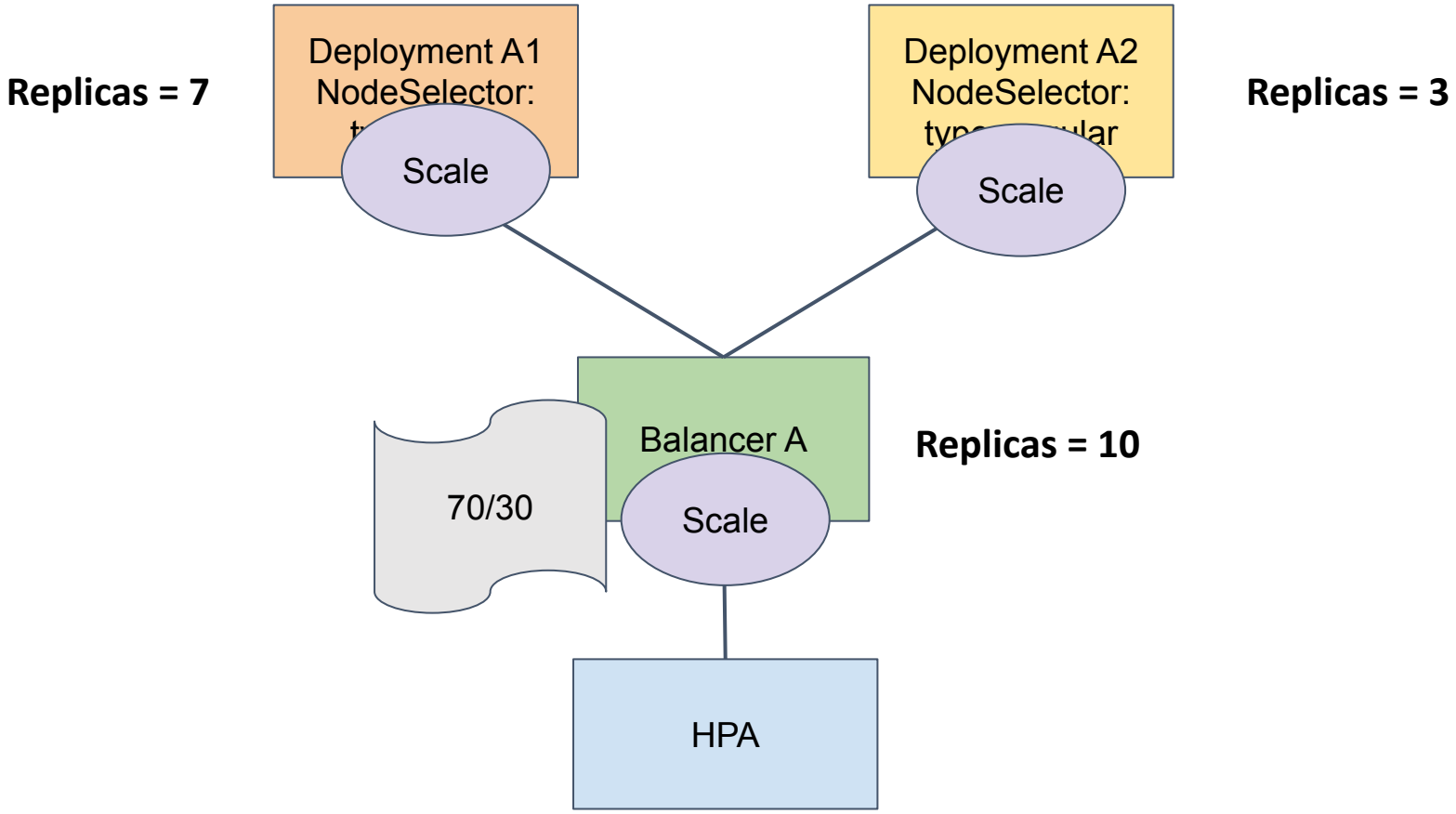
Balancer


Replicas = 7
Deployment A1
NodeSelector:
type=spot
Deployment A2
NodeSelector:
type=regular
Replicas = 3
Balancer A
Replicas = 10
70/30

Balancer


Replicas = 7
Deployment A1
NodeSelector:
type=spot
Deployment A2
NodeSelector:
type=regular
Replicas = 3
Balancer A
Replicas = 10
70/30
HPA

Balancer

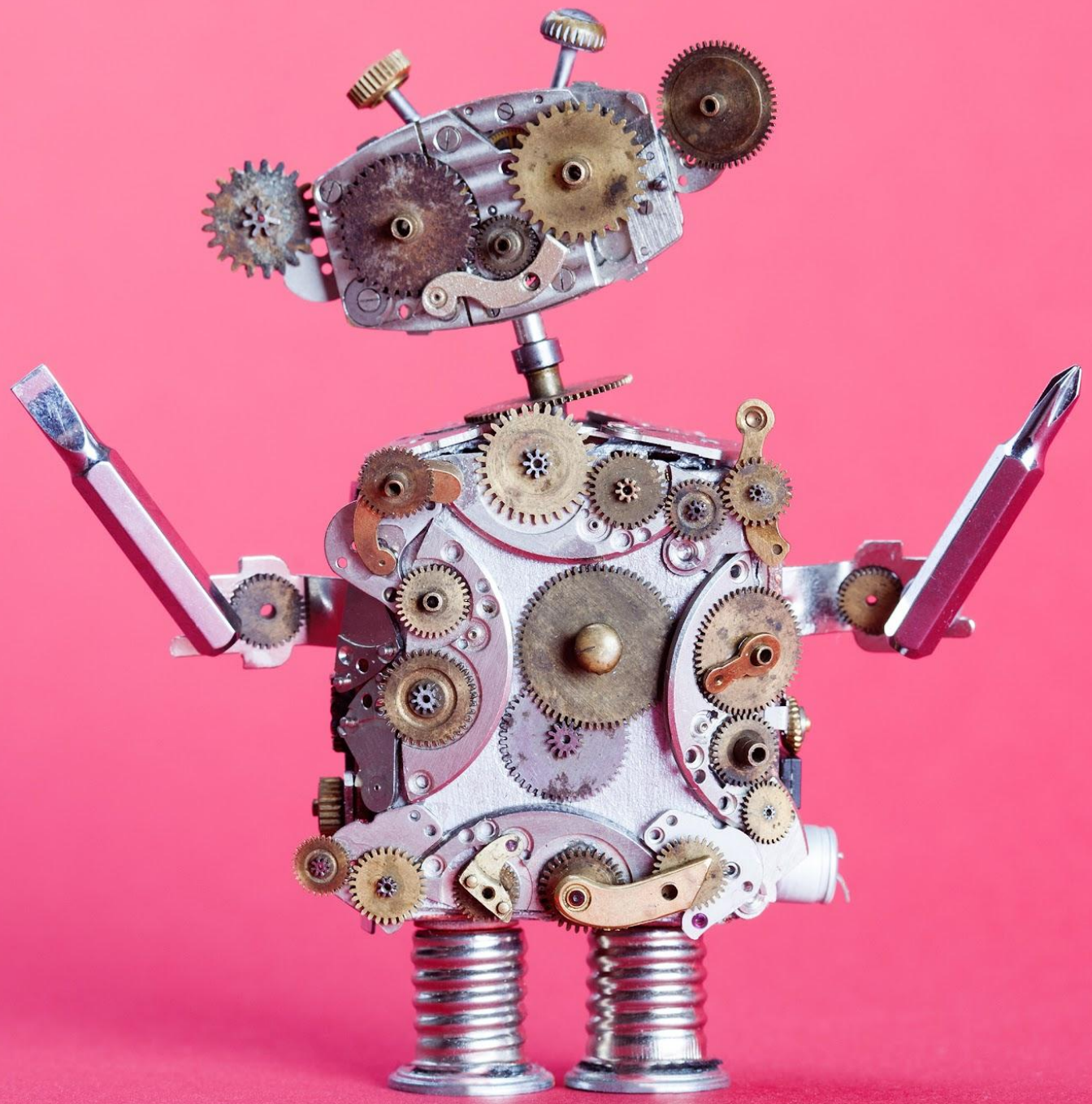

Replicas = 7
Deployment A1
NodeSelector:
Scale
Deployment A2
NodeSelector:
Scale
Replicas = 3
Balancer A
Scale
70/30
Replicas = 10
HPA

Balancer

Status:

- SIG-internal KEP merged
- Largely already coded in a private Google-internal repository
- To be open sourced in November
- Alpha release this year

Vertical Pod Autoscaler

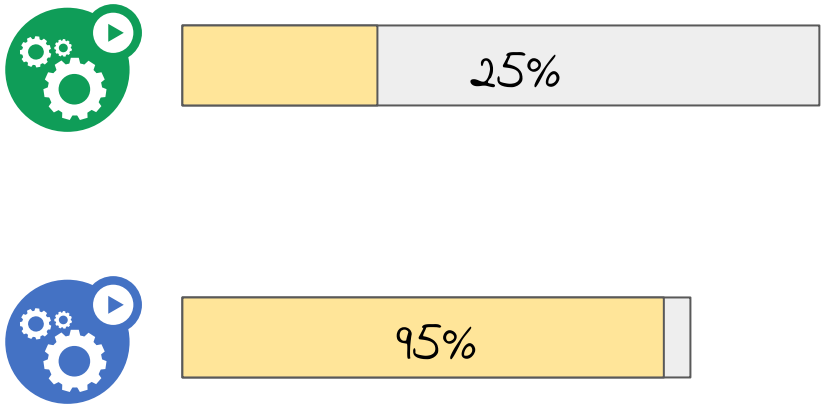


Vertical Pod Autoscaler

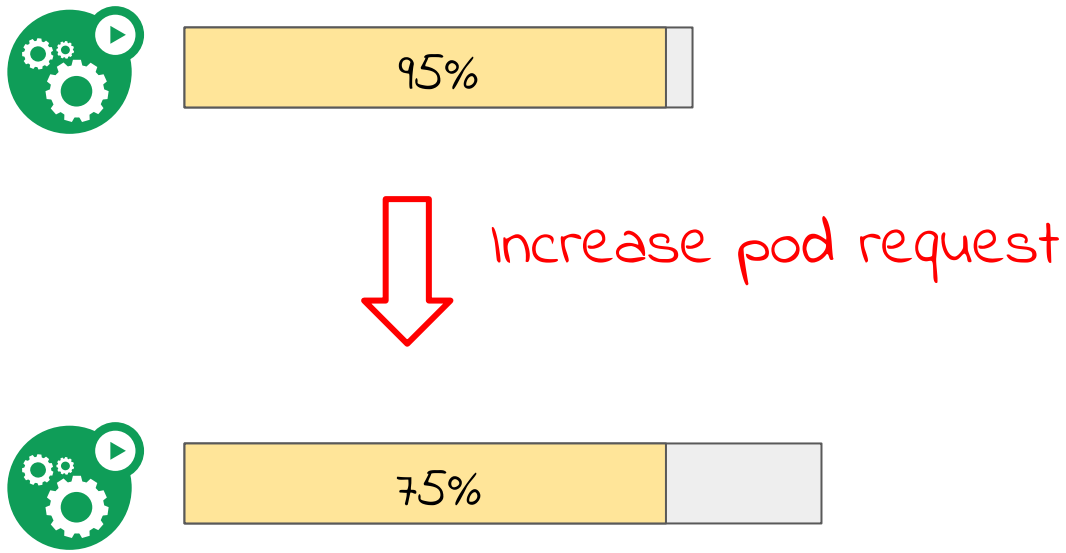
- Pod size should not be to big or too small.

- Based on resource data
  - CPU usage
  - Memory usage
  - OOM events

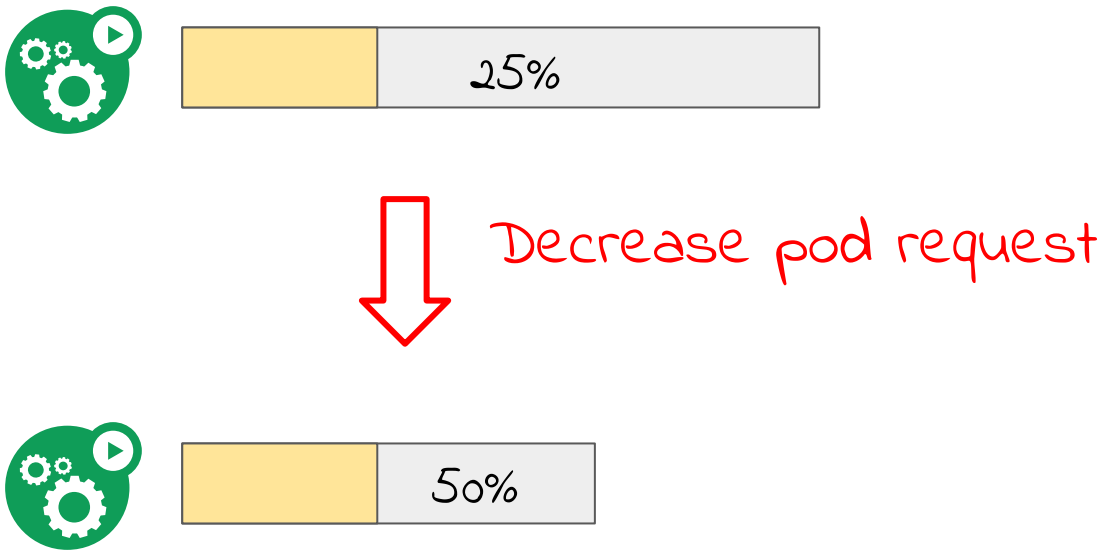
- Recommends pod sizes to keep real usage well within the requested pod capacity.



Vertical Pod Autoscaler



Vertical Pod Autoscaler



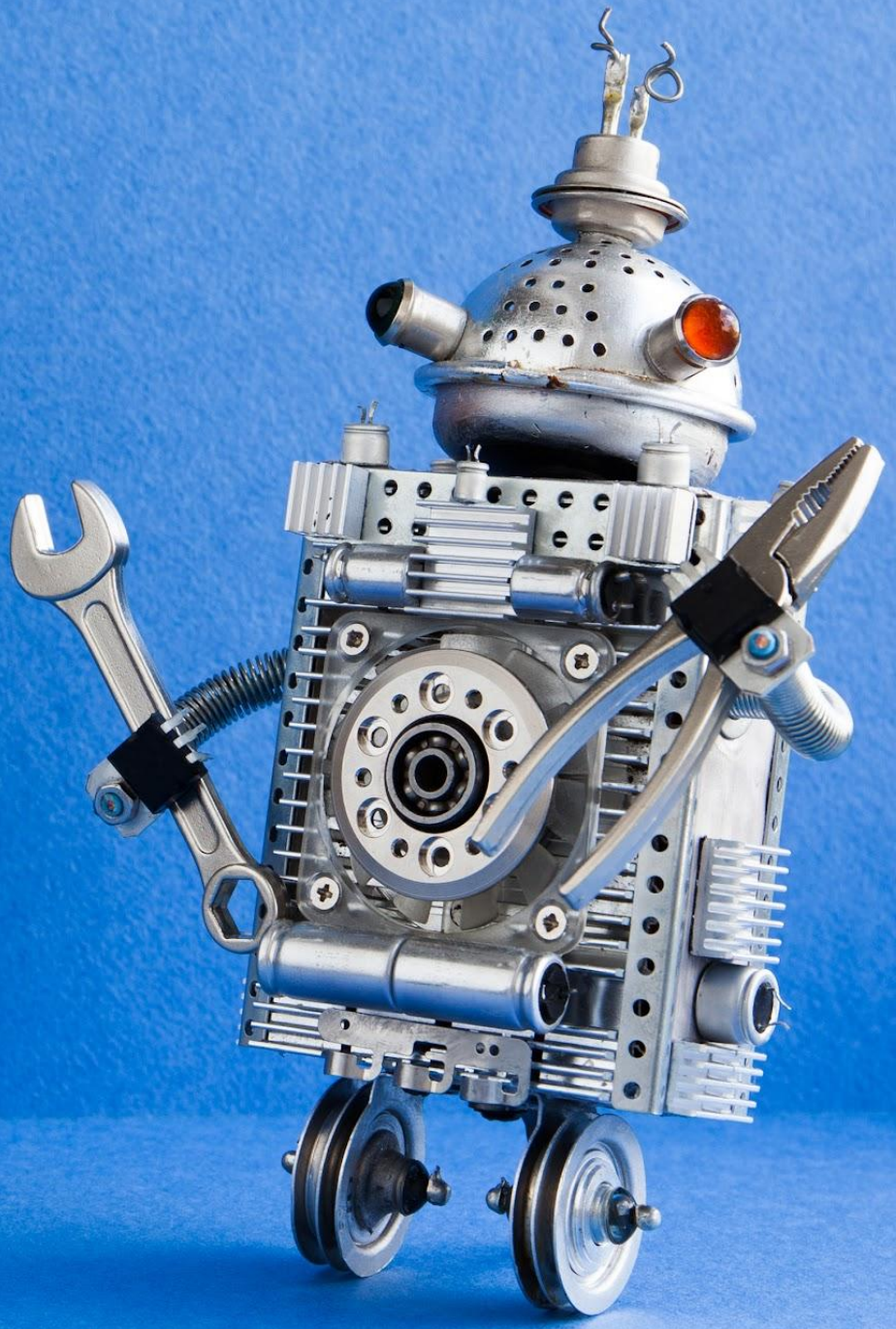
VPA - recent and upcoming changes

- Pluggable/multiple recommender support

- Configurable:
  - limit for minimum number replicas to start actuation
  - percentile used by recommendations

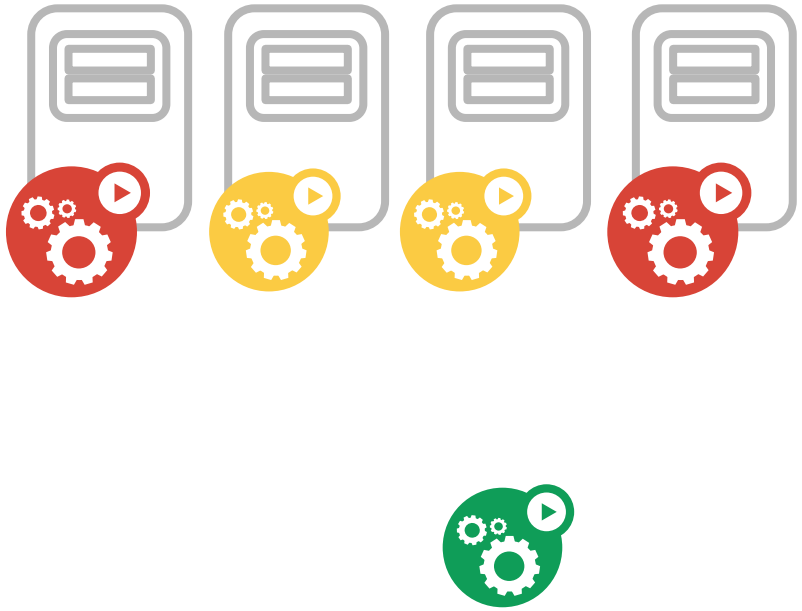
- Coming (hopefully) soon:
  - Fixed ratio between CPU/Mem
  - Limit for the direction of updates
  - In place updates for VPA (once k/k/#102884 is merged)

Cluster Autoscaler

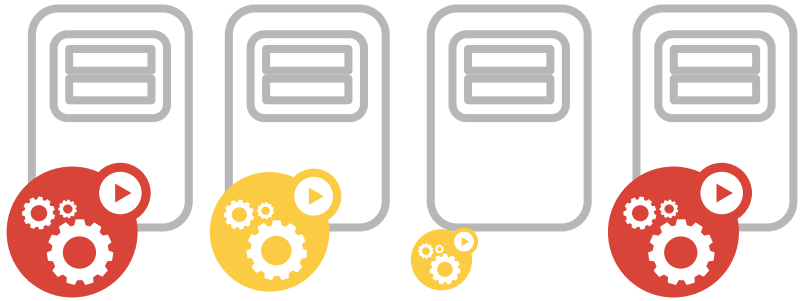


Cluster Autoscaler

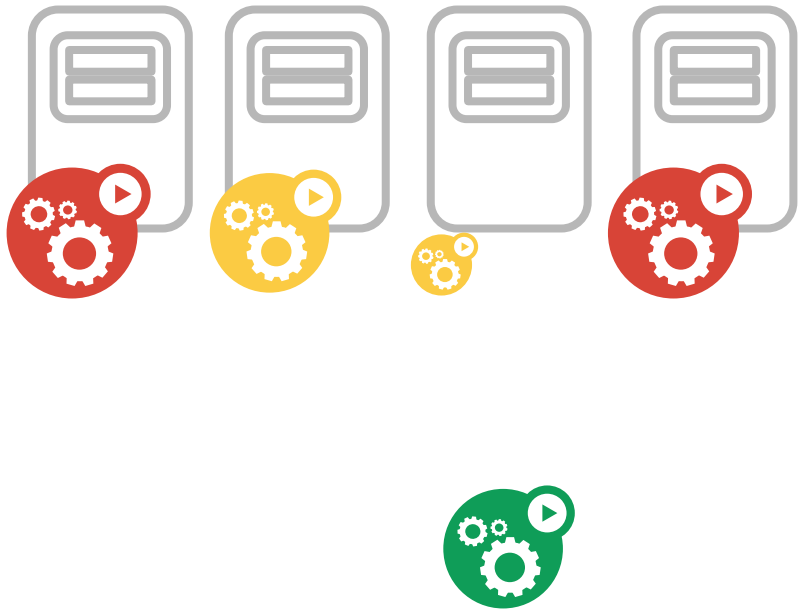
- Provides nodes for pods that don't have a place to run.
- Removes underutilised nodes.
- Uses scheduling simulations and declared pod requests.



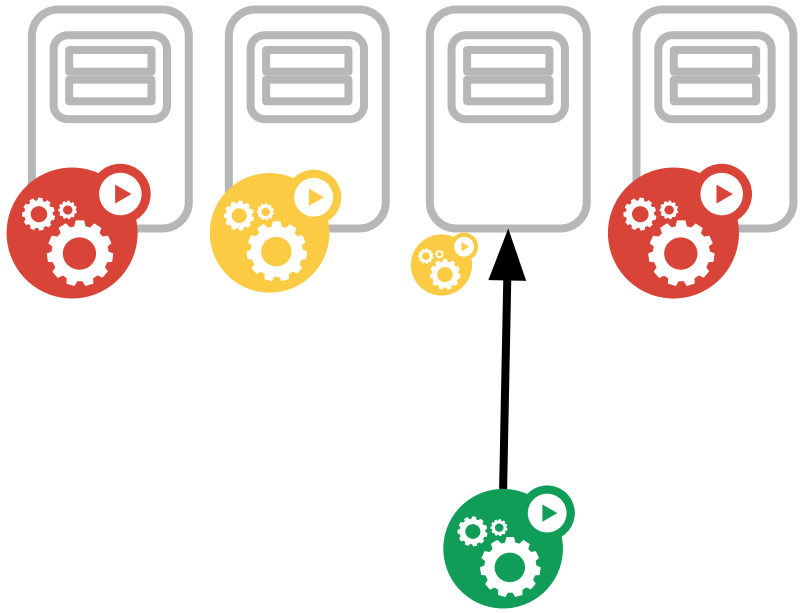
Cluster Autoscaler



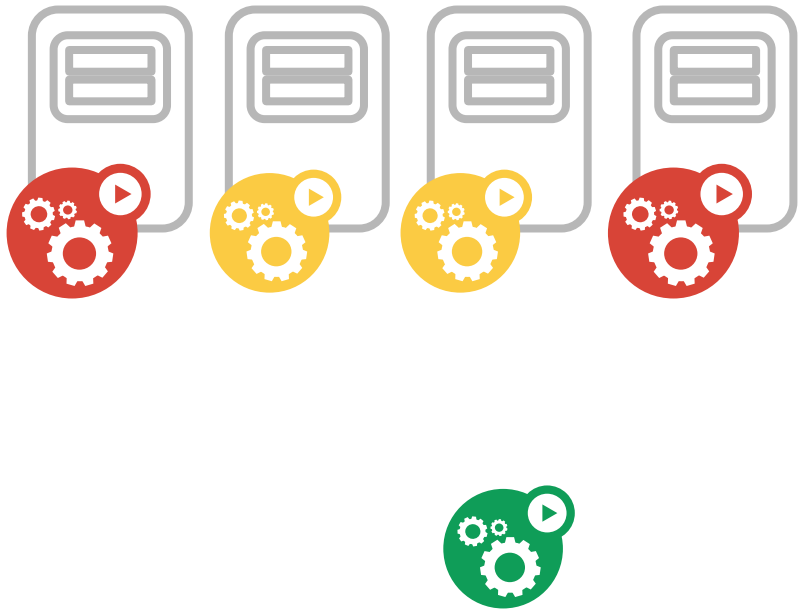
Cluster Autoscaler



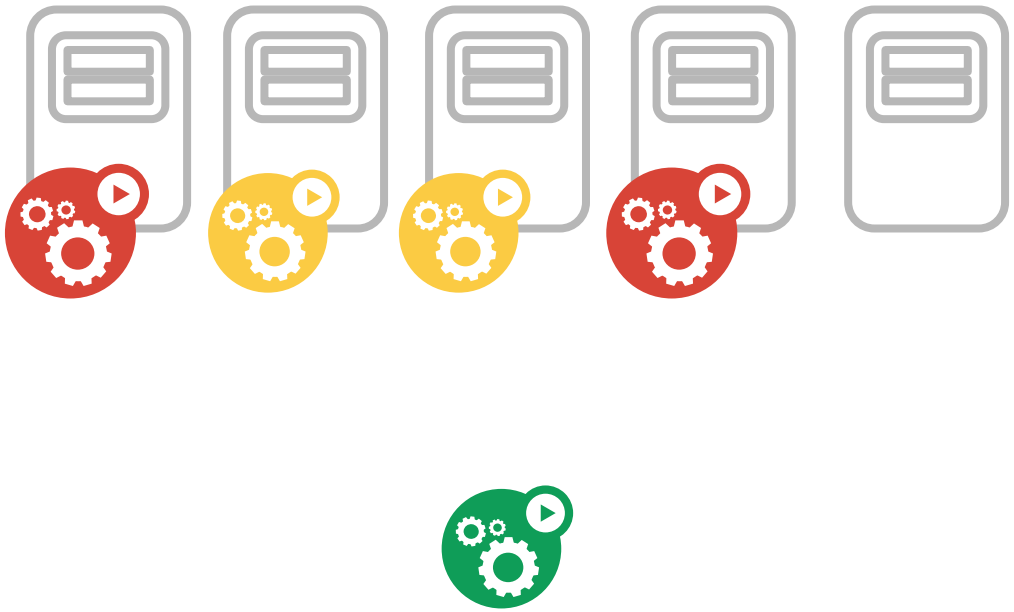
Cluster Autoscaler



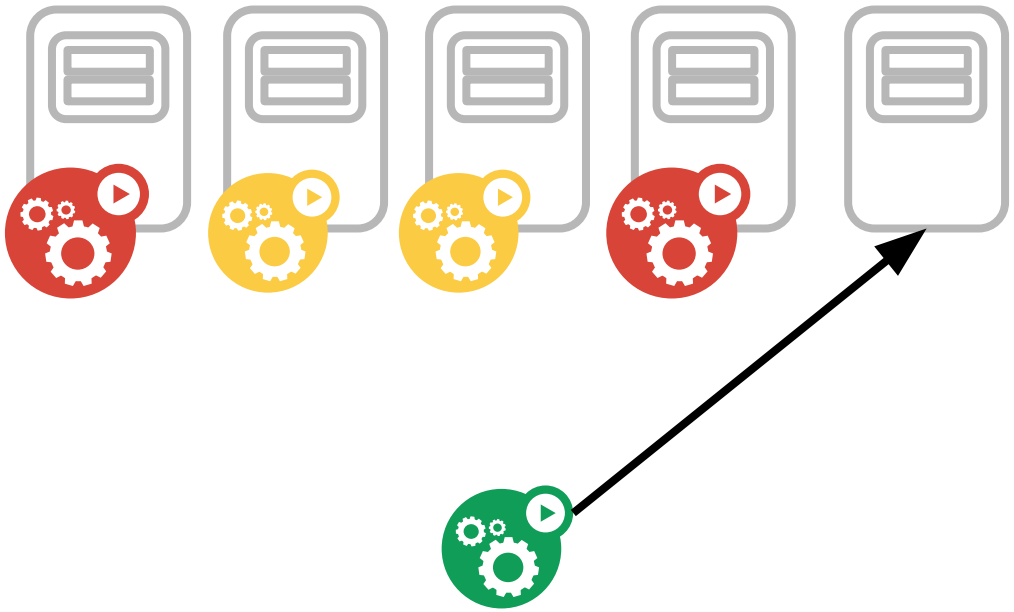
Cluster Autoscaler



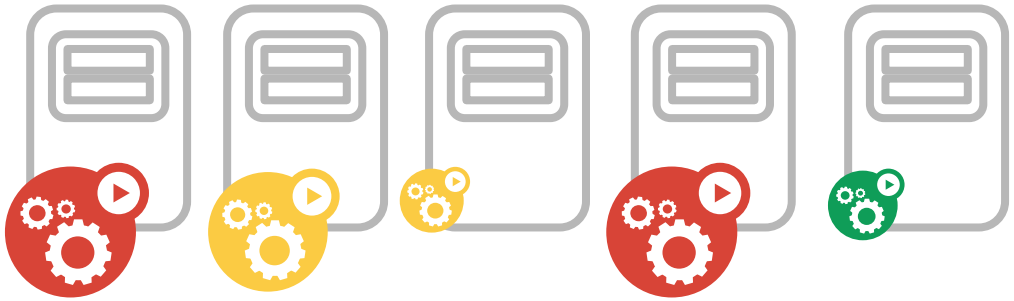
Cluster Autoscaler



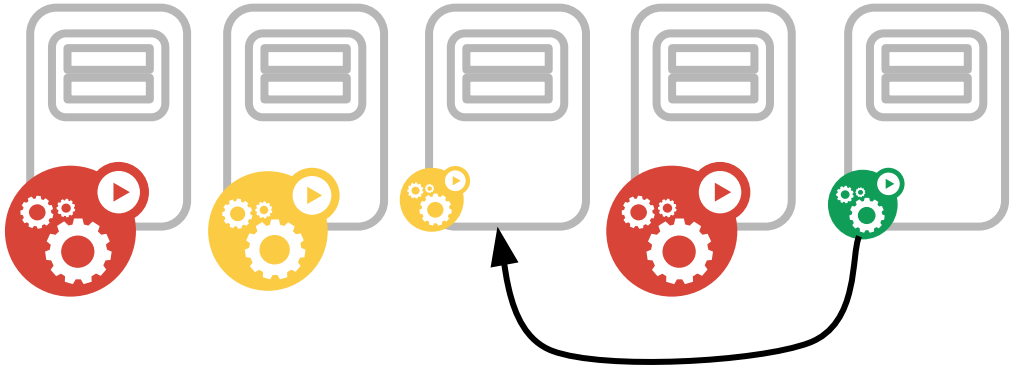
Cluster Autoscaler



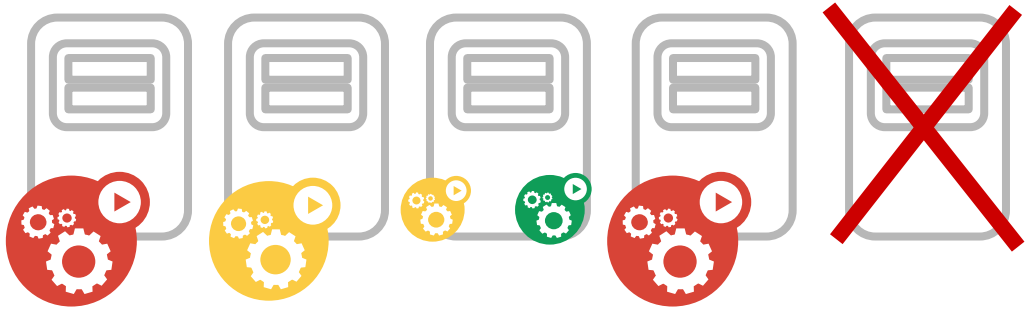
Cluster Autoscaler



Cluster Autoscaler



Cluster Autoscaler



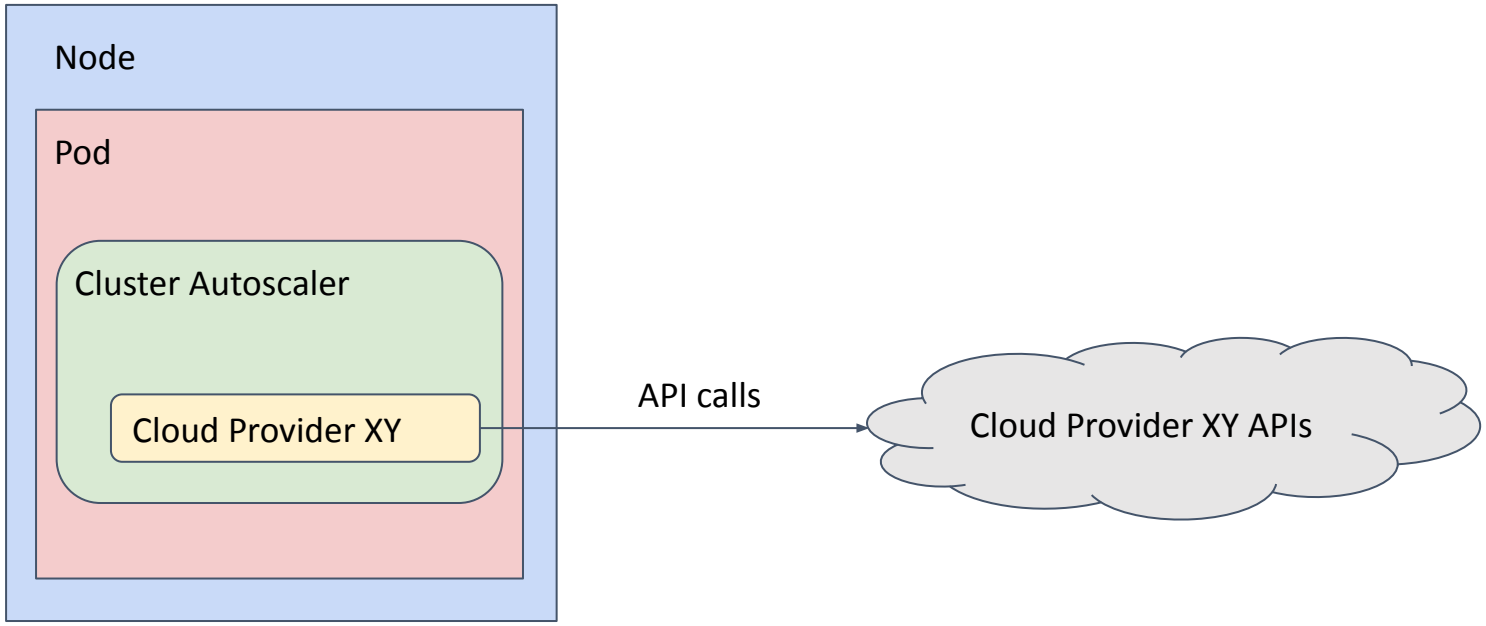
CA - recent and upcoming changes

- Parallel scale down (hopefully 1.26)
- Scale down not blocking on pending pods (1.24)
- gRPC:
  - cloud provider (1.25)
  - expander (1.24)
- Better batch use cases support - Kqueue integration (1.27+)

External gRPC Cloud Provider

What is a cloud provider in the Cluster Autoscaler (CA)?

Cloud specific logic hardcoded inside CA used to interact with specific cloud providers

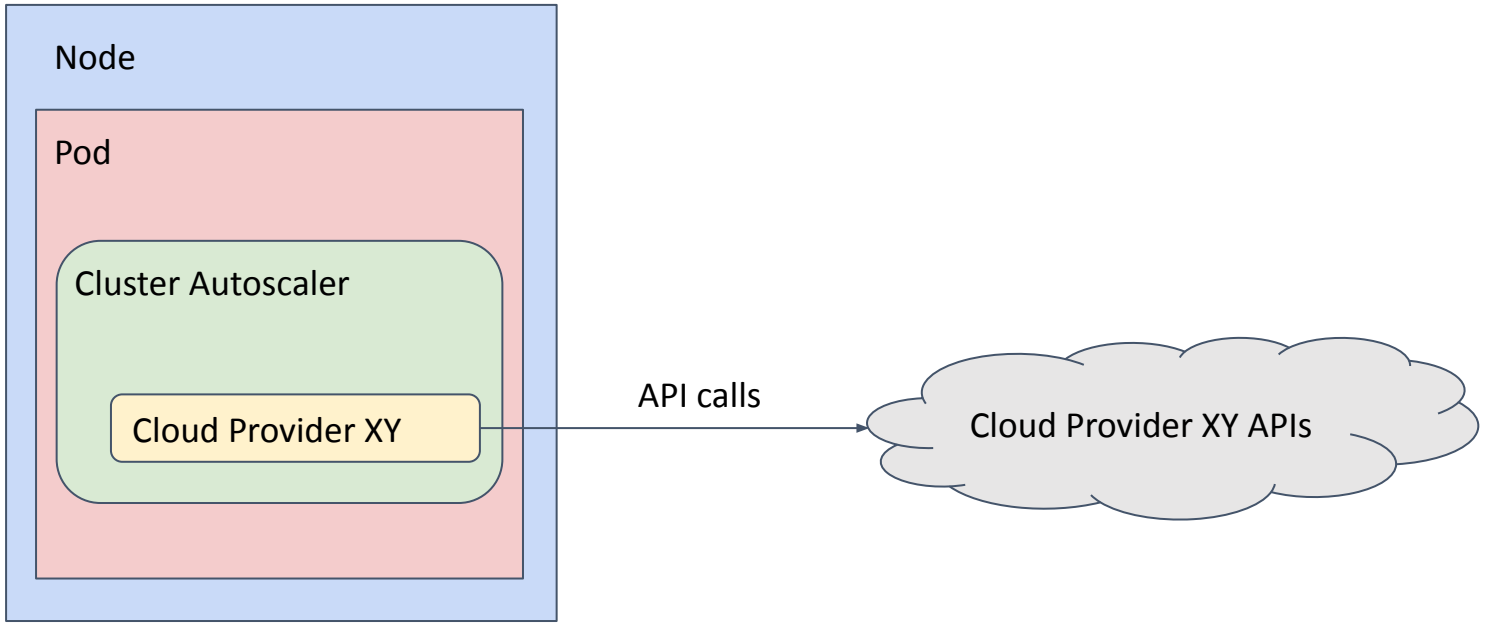


External gRPC Cloud Provider

How to implement a cloud provider in CA?

- Fork CA code
- Implement cloud provider logic as `CloudProvider` and **NodeGroup** go interfaces
- Add it to the cloud provider builder

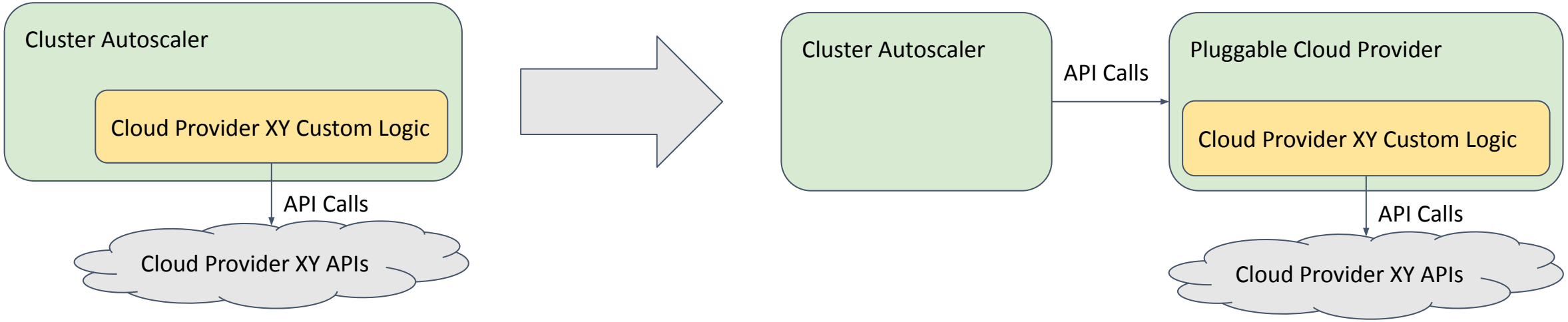
contributing rules: no new dependencies



External gRPC Cloud Provider

Pluggable Cloud Provider - why?

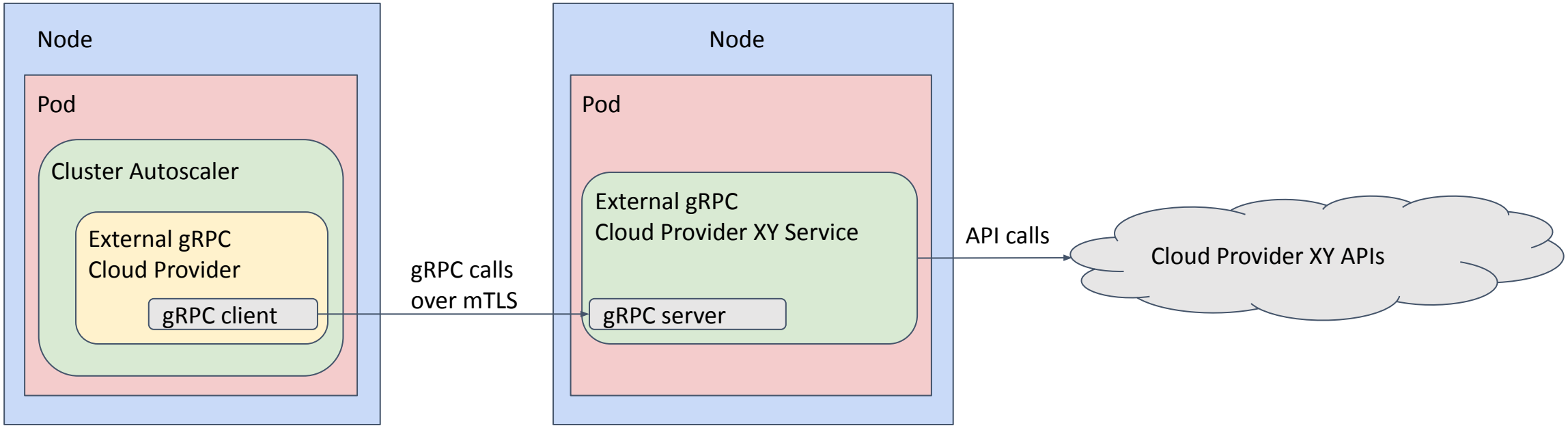
- No need to fork CA
- Decouple CA core and cloud provider dependencies
- No need to follow CA development lifecycle



External gRPC Cloud Provider

Pluggable Cloud Provider over gRPC

- Out-of-tree cloud provider support
- Custom cloud provider logic in an external service, exposed as gRPC APIs
- CA acts as gRPC client, implemented as a classic in-tree cloud provider



External gRPC Cloud Provider

How to create an external gRPC cloud provider service? - general requirements

- CA assumes nodes belong to *node groups*, all nodes within a group:
  - Must be of the same machine type
  - Have the same set of labels and taints
  - Are located in the same availability zone
- There must be a way to delete a specific node
- There must be a way to match a Kubernetes node to the instance it is running on, e.g. `ProviderId` field

External gRPC Cloud Provider

How to create an external gRPC cloud provider service? - implementation

- Create a standalone service with your language of choice
- Implement the server side of the `CloudProvider` gRPC service defined in [`externalgrpc.proto`]
- Fill it with your cloud provider logic
- Expose the gRPC endpoint with mTLS

```
service CloudProvider {
 // CloudProvider specific RPC functions
 rpc NodeGroups
 rpc NodeGroupForNode
 rpc PricingNodePrice //optional
 rpc PricingPodPrice  //optional
 rpc GPULabel
 rpc GetAvailableGPUTypes
 rpc Cleanup
 rpc Refresh

 // NodeGroup specific RPC functions
 rpc NodeGroupTargetSize
 rpc NodeGroupIncreaseSize
 rpc NodeGroupDeleteNodes
 rpc NodeGroupDecreaseTargetSize
 rpc NodeGroupNodes
 rpc NodeGroupTemplateNodeInfo //optional
 rpc NodeGroupGetOptions       //optional
}
```

External gRPC Cloud Provider

Caveats

- gRPC calls are cached on the CA side, but performances for very large clusters have not been tested yet

- The **CloudProvider** gRPC service mimics the **`CloudProvider`** and **`NodeGroup`** go interfaces with some differences:
    - **`NodeInfo`** does not return information about initial pods for a node (e.g. static pods), calculations when scaling from 0 nodes could be wrong if such pods exist
    - **`GetResourceLimiter`** function (unpopular) has not been implemented
    - Deprecated functions have not been implemented

External gRPC Cloud Provider

Learn more

- Design Proposal (PR)
- Implementation (PR)
- README
- Protobuf definition file
- Example Code

More
about SIG-Autoscaling

Meetings

Monday 10:30 EDT

https://zoom.us/j/944410904

Slack

#sig-autoscaling

kubernetes.slack.com

Github

https://github.com/kubernetes/autoscaler


Thank You!


Time for
Questions!