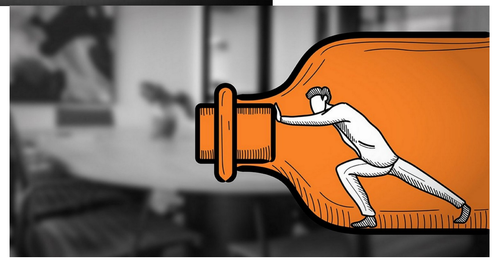
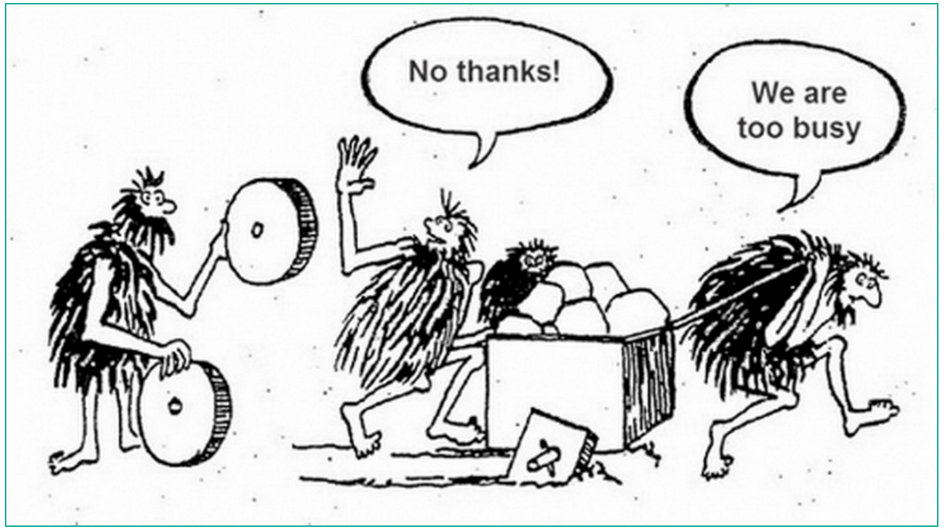
THE SCALE FACTORY

NODELESS SCALING **WITH KARPENTER_**

Marko Bevc




No thanks!
We are too busy

ABOUT ME_

- Head of Consultancy at The Scale Factory (remote-first, Cloud SaaS consultancy, AWS Advanced consulting partner and K8s service provider, we're hiring across EU!)
- Ops background: Senior IT Infrastructure Engineer and System Architect (Linux/Windows automation, network and virtualization)
- Open source contributor, maintainer and supporter
- HashiCorp Ambassador
- Certifications and competencies: AWS, CKA, RHEL, HCTA
- Fan of automation/simplifying things, hiking, cycling and travelling

@_MarkoB

https://www.linkedin.com/in/marko-bevc/

TOPICS **COVERED_**

- Kubernetes and resource scaling
- Autoscaling landscape
- Limitations
- Groupless autoscaling
  - Architecture and advantages
  - Scheduling & bin packing
  - Launching and capacity lifecycle
- Demo
- Conclusions & takeaways

SCALING RESOURCES_

- Scaling types:
  - Horizontal – adding more instances, scaling out
  - Vertical – add more power, scaling up
- Kubernetes resources:
  - Pods – the smallest execution unit
  - Nodes – instances to run Pods on
  - Other: storage, network, etc.

#1

PODS SCALING_

- Where do Pods come from?
  - Direct definition (*kubectl*)
  - Workload resources: *Deployment*, *ReplicaSet*, *StatefulSet*, *DaemonSet*, *Job/CronJob*
- Scaling approaches:
  - HPA | VPA* (*HorizontalPodAutoscaler* | *VerticalPodAutoscaler*)
  - GCP: *MultidimPodAutoscaler*
  - KEDA (K8s Event Driven Autoscaling)
  - Knative (K8s based serverless platform)

#2

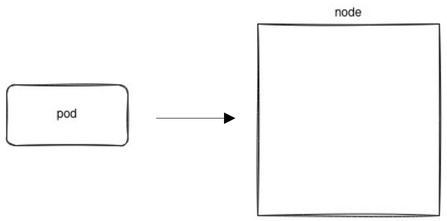
SCALING **NODES_**

- Manual provisioning
- Less maintained projects:
  - Cerebral (deprecated)
  - Escalator (Atlassian)
- Cluster Autoscaler (de-facto)
  - Zalando fork (robust templating, mixing instance types)

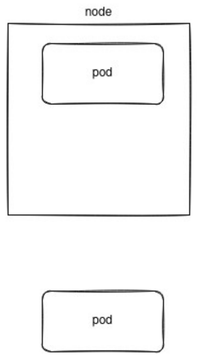
CLUSTER **AUTOSCALER_**

- Industry adopted, open source and vendor neutral
- Automatically adjusts cluster size – cost efficient
- Battle tested: up to ~1000 nodes and 30 pods/node
- Using existing Cloud building blocks – i.e. ASG on AWS
- Challenges: Node Group limitations (AZ, instance type, labels), complex to use, tightly bound to the scheduler, global controller

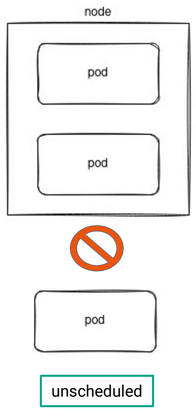
SCHEDULING **ON NODES_**



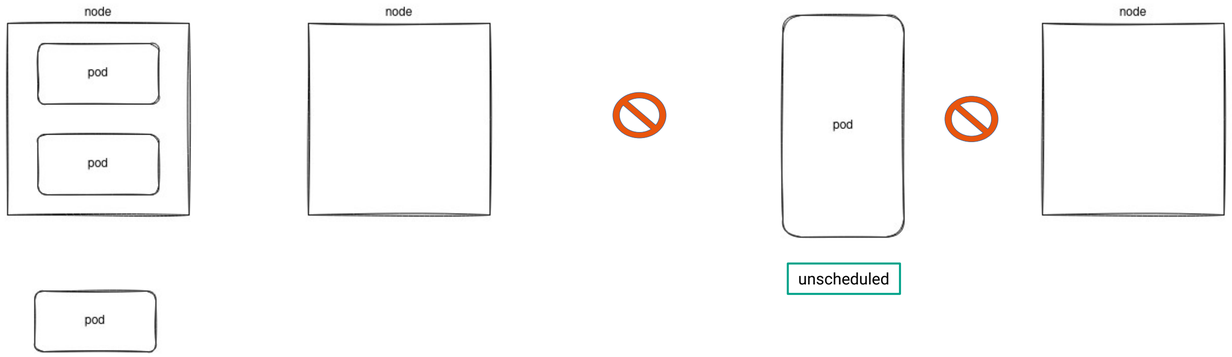
SCHEDULING **ON NODES_**



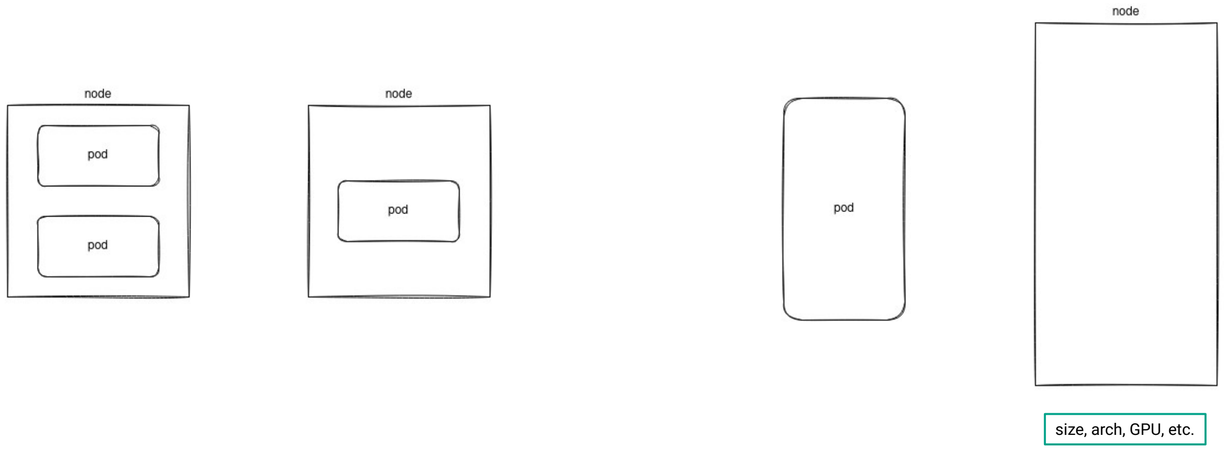
SCHEDULING **ON NODES_**



SCHEDULING **ON NODES_**



SCHEDULING **ON NODES_**



KARPENTER
PROJECT_

- Node lifecycle solution: OSS incubated from **awslabs**, vendor neutral*
- **Removes Node Group** concept → Groupless Node Autoscaling
- Directly provision capacity using:
  - Pod resource requests → instance types
  - Use native Kubernetes labels to track nodes
  - Generating node properties on the fly from scheduling constraints
- Lowers Cloud Provider API utilisation
- Reduces configuration complexity (controller + provisioner(s)CRD)

provisioner.yaml

```
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
metadata:
  name: default
spec:
  labels: # Added to provisioned nodes
    team: a-team
  requirements: # using well-known annotations
    - key: "node.kubernetes.io/instance-type"
      operator: In
      values: ["m5.xlarge", "m5.2xlarge", "c5.xlarge", "c5.2xlarge", "t4g.small", "t4g.xlarge"]
    - key: "topology.kubernetes.io/zone"
      operator: In
      values: ["eu-west-1a", "eu-west-1c"]
    - key: "kubernetes.io/arch"
      operator: In
      values: ["arm64", "amd64"]
    - key: "karpenter.sh/capacity-type" # default to on-demand, not spot
      operator: In
      values: ["on-demand"]
  limits:
    resources:
      cpu: 1000
  provider:
    subnetSelector:
      karpenter.sh/discovery: ${CLUSTER_NAME}
    securityGroupSelector:
      karpenter.sh/discovery: ${CLUSTER_NAME}
    instanceProfile: KarpenterNodeInstanceProfile-${CLUSTER_NAME}
  ttlSecondsAfterEmpty: 30
```

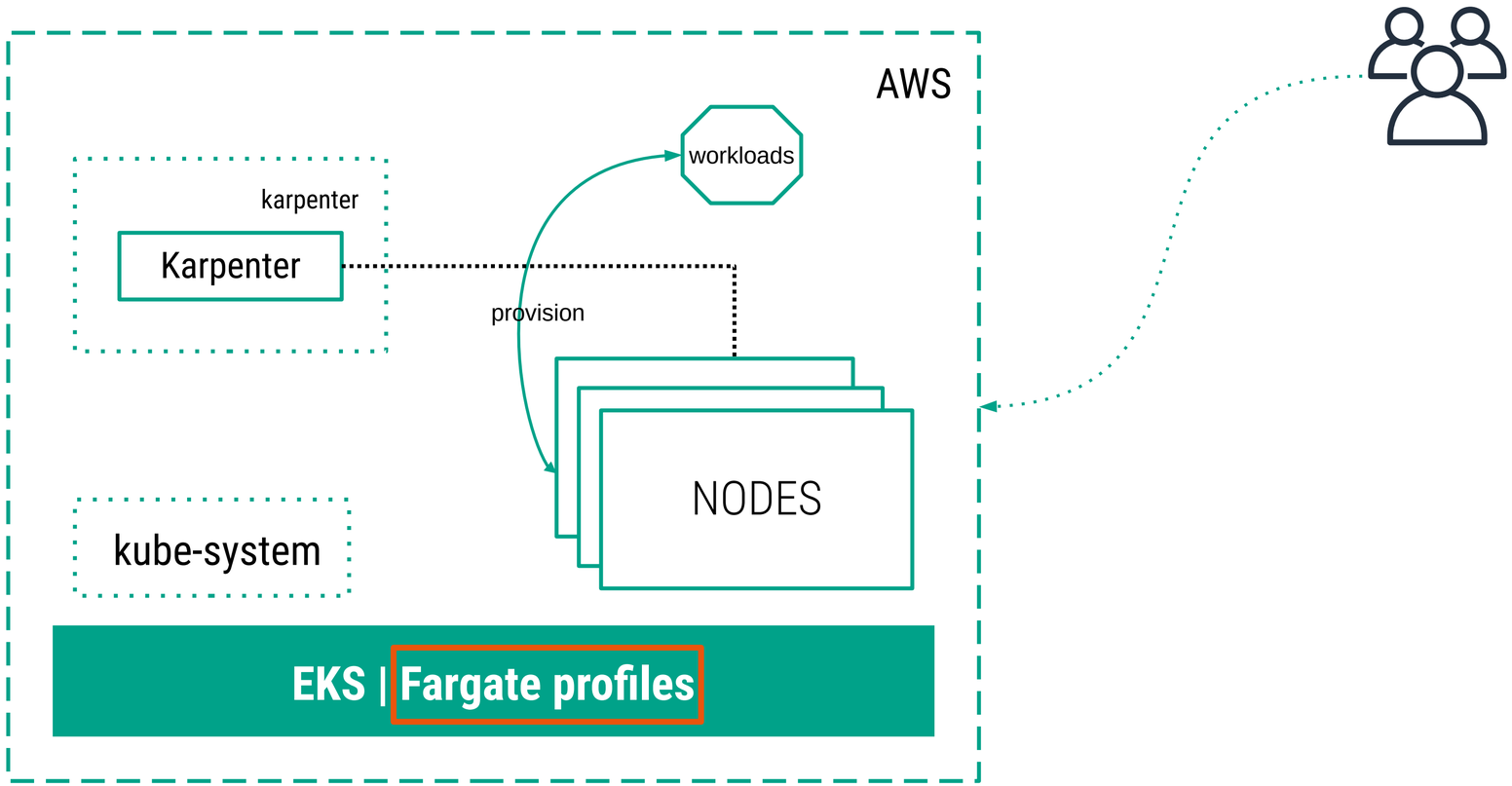
CAPACITY LIFECYCLE_

- Controller loop:
  - Watch and evaluate constraints (resource requests, node selectors, affinities, tolerations, and topology spread constraints)
  - Provision nodes / schedule pods (binding)
  - Remove nodes
- Termination:
  - Removal of empty/underutilised/deleted nodes – cordon & drain
  - Termination grace period
  - Spot rebalance & termination events
  - Instance unhealthy events
  - Expiration TTL – compliance!

LAUNCHING **CAPACITY_**

- Launching capacity: API → EC2 *create-fleet* (*run-instance*)
- Scheduling:
  - Online BinPacking – First Fit Descending
  - Loosely coupled with *kube-scheduler* – cross K8s version
  - Bind pods to the nodes it creates
- Well known labels as provisioning constraints:
  - node.kubernetes.io/instance-type = m5.large
  - topology.kubernetes.io/zone = eu-west-1
  - node.k8s.aws/capacity-type = spot
  - kubernetes.io/os = linux
  - kubernetes.io/arch = amd64

SETTING THE **SCENE_**




TIME FOR
A DEMO!_

CONCLUSIONS_

& TAKEAWAYS

- Capacity planning is **hard**!
- Key advantages:
  - Eliminates Node Group (low touch), full flexibility and direct capacity provisioning – working alongside *kube-scheduler*
  - Lowers complexity & portable
  - Performance: provisioning latency ~1 min → down to 15 sec
- To keep in mind:
  - Improved heuristics, defragmentation / reallocator – Off-line BinPacking (repack inefficiently utilised nodes)
  - Replace spot instances based on markets
  - Currently only supported providers is AWS
  - https://github.com/aws/karpenter/issues

FURTHER **READING_**

- Resources:
  - https://github.com/aws/karpenter/
  - https://kubernetes.io/docs/reference/labels-annotations-taints/
  - https://github.com/kubernetes/autoscaler
  - https://docs.aws.amazon.com/eks/latest/userguide/cluster-autoscaler.html
  - https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/proposals/scalability_tests.md
  - https://blog.kloia.com/karpenter-cluster-autoscaler-76d7f7ec0d0e
  - https://kubernetes.io/docs/concepts/workloads/
  - https://blog.scaleway.com/understanding-kubernetes-autoscaling/
  - https://github.com/zalando-incubator/autoscaler

KEEP IN TOUCH_

Web: https://www.scalefactory.com/
Twitter: @_MarkoB
GitHub: @mbevc1
GitLab: @mbevc1
LinkedIn: https://www.linkedin.com/in/marko-bevc/