Bin-Packing Pods in Managed Kubernetes

Vinay Suryadevara, Jianfei Hu

KubeCon + CloudNativeCon Europe, 2024

ClickHouse

Speakers

Vinay Suryadevara

Senior Software Engineer
@ ClickHouse

Jianfei Hu

Senior Software Engineer
@ ClickHouse

Agenda

01 Introduction

02 Overview

03 Infrastructure setup

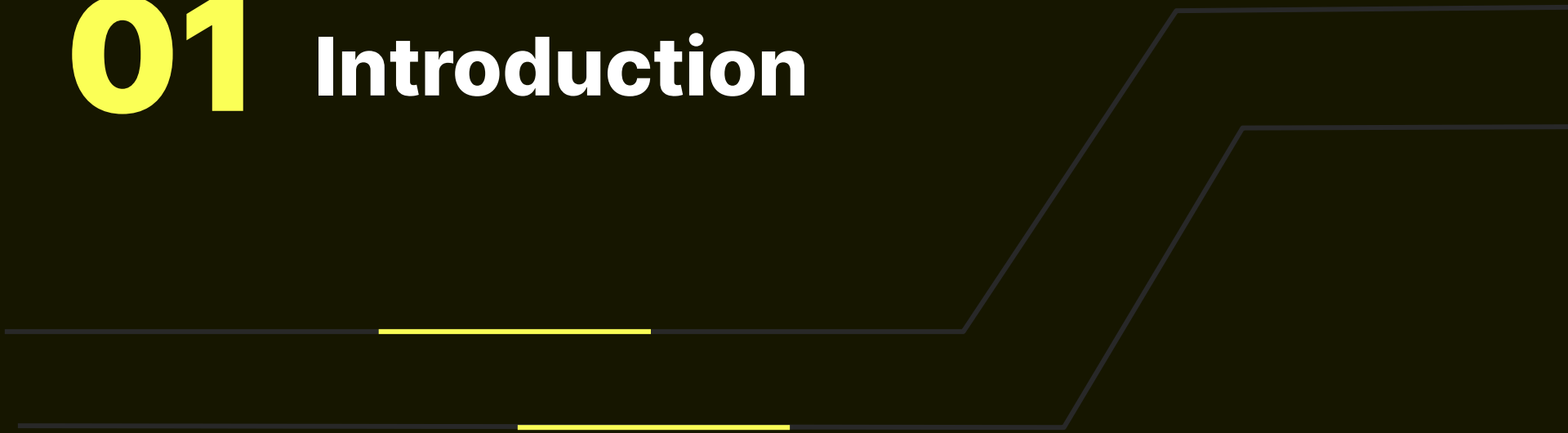
04 Approaches to improve node utilization

05 Rollout & evaluation

06 Summary

07 Q&A

01 Introduction



ClickHouse

What is ClickHouse?

OLAP database

- Analytics use cases
- Aggregations
- Visualizations
- Mostly immutable data

Open source

- Developed since 2009
- OSS 2016
- 33.5 k+ Github stars
- 1400+ contributors
- 530+ releases

Distributed

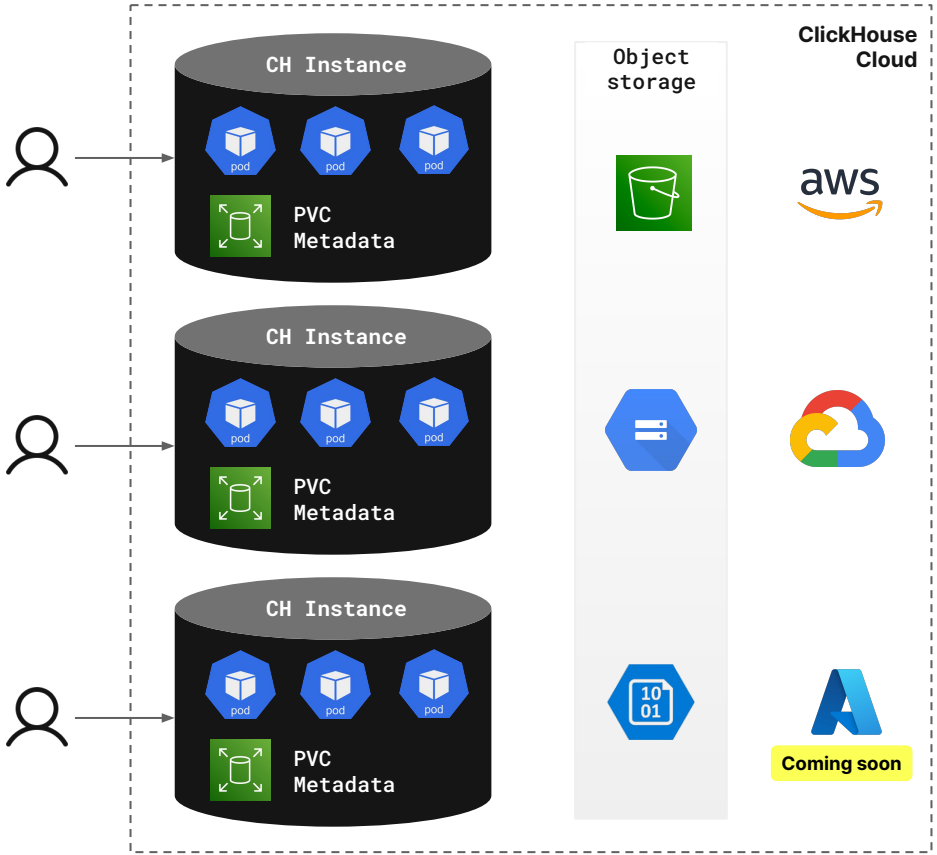
- Replication
- Sharding
- Multi-master
- Cross-region

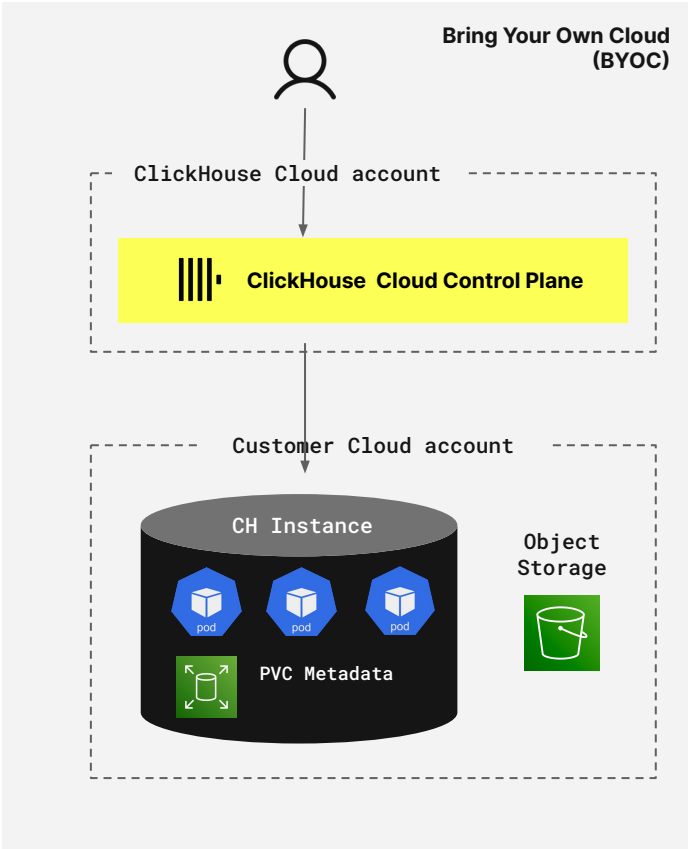
Fast

- Column-oriented storage
- Sparse Indexes
- Data compression
- Vectorized query execution
- Bottom-up design approach

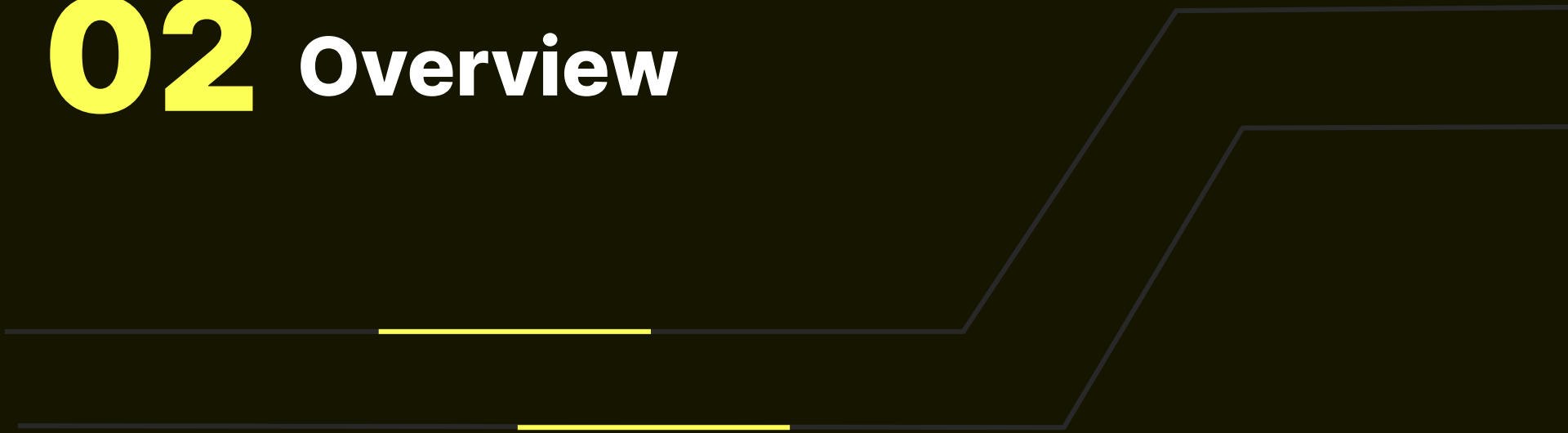
What is ClickHouse **Cloud?**

✔Serverless ✔Idling & Autoscaling ✔Compute & Storage separation





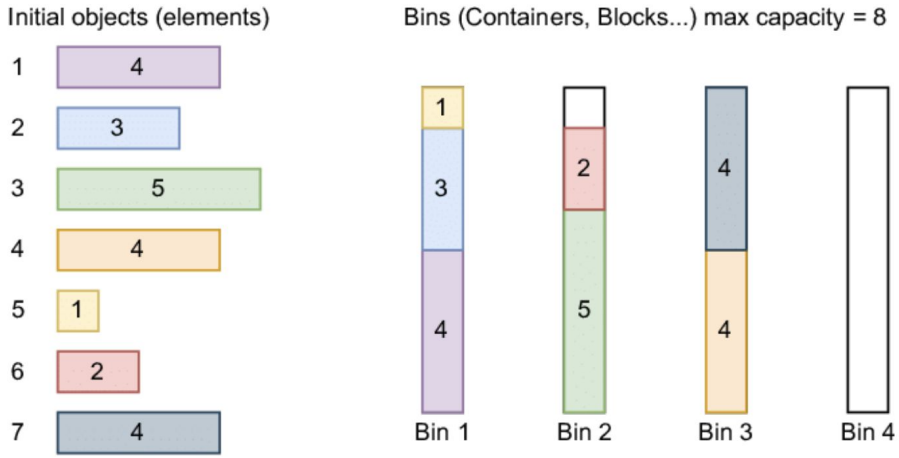
02 Overview



ClickHouse

Bin-packing - What is it?

- **Wikipedia** - “The **bin packing problem** is an optimization problem, in which items of different sizes must be packed into a finite number of bins or containers, each of a fixed given capacity, in a way that minimizes the number of bins used”.
- **Bin-packing in Kubernetes** refers to the process of efficiently scheduling and placing pods onto nodes in the cluster to maximize resource utilization and reduce the number of nodes needed.
- This approach helps in optimizing the use of **CPU, memory**, and other resources across the cluster, leading to cost savings and improved operational efficiency.



Source: https://tinyurl.com/5n6eyske

Overview

Problem

Node resource utilization (CPU/Memory) non-optimal across our fleet.

Higher infrastructure costs.

Solution

Update Kubernetes scheduler scoring strategy to bin-pack pods onto fewer nodes.

Rollout

Multiple Kubernetes clusters, multiple regions, multi-tenant fleet.

Rollout changes to fleet reliably with minimal disruptions.

03 Infrastructure Setup

ClickHouse

Terminology

- **ClickHouse keeper**
  - Similar to Zookeeper.
  - Coordination component for server pods.
- **ClickHouse server**
  - Compute pods for ingest/query.
- **ClickHouse instance/cluster**
  - Custom Resource that consists for Keeper and Server pod StatefulSets and pods.
  - Both keeper and server pods have similar needs/profiles w.r.t bin-packing.
  - Only focusing on Server pods in this talk for simplicity.

- **Node** - EC2/Azure/GCE VMs.
- **Node utilization** - Sum of **resource requests** for all pods on the node **divided** by **allocatable resources** on the node.
- **Kube-scheduler** - Default pod scheduler for all pods in kubernetes cluster.

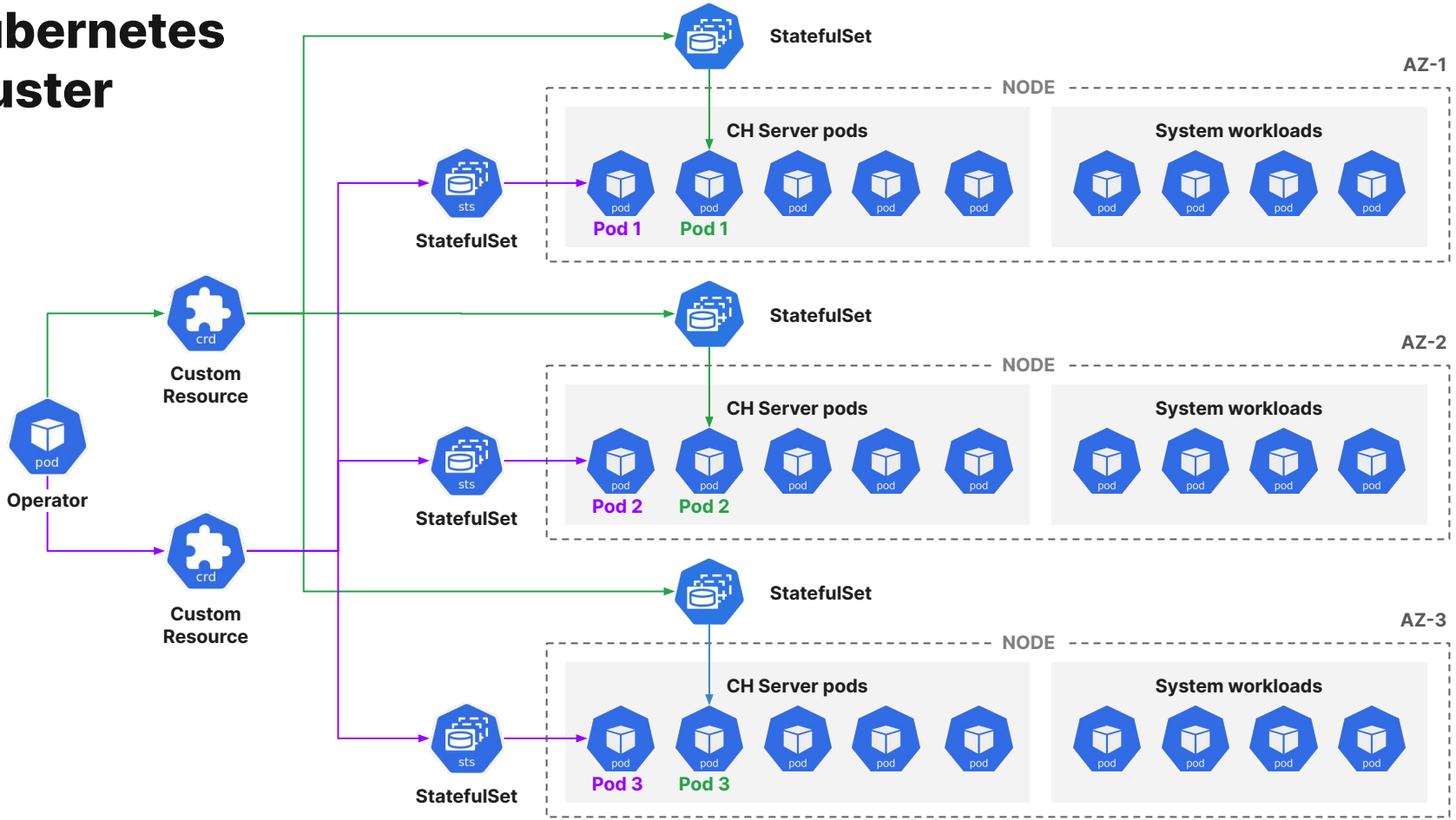
Infrastructure setup

- Managed Kubernetes in each CSP
  - **EKS** (AWS), **GKE** (GCP) and **AKS** (Azure).
- Multi-tenant Kubernetes clusters
  - Pods from 2 different CH instances running on same node.
- Namespaces for isolation
  - Kubernetes NetworkPolicy (**Cilium CNI**) to prevent cross namespace network calls.

Infrastructure setup (contd..)

- **Cluster autoscaler** for dynamic node provisioning.
- **QoS - Guaranteed**.
- **Homogenous workloads**. Same shape for most pods.
- **Overprovisioning** - Reserve extra capacity for workload spikes. Low priority preemptable pods scheduled on buffer nodes.
- **Weekly release** schedule of upgrades to all CH pods. Periodic rescheduling of pods.

Kubernetes cluster



Problem - Infrastructure utilization/cost

- Node utilization across the fleet is around 40% - 60%
- Low utilization means higher number of nodes and higher cost to run the product
- Factors for utilization:
  - Idling, ClickHouse clusters scaled to 0 when receiving no requests for last 30 minutes
  - Autoscaling
  - Service creation, stop, and termination
- **Goal** - Increase node resource utilization across the fleet

04 Approaches to improve node utilization

ClickHouse

Solution requirements

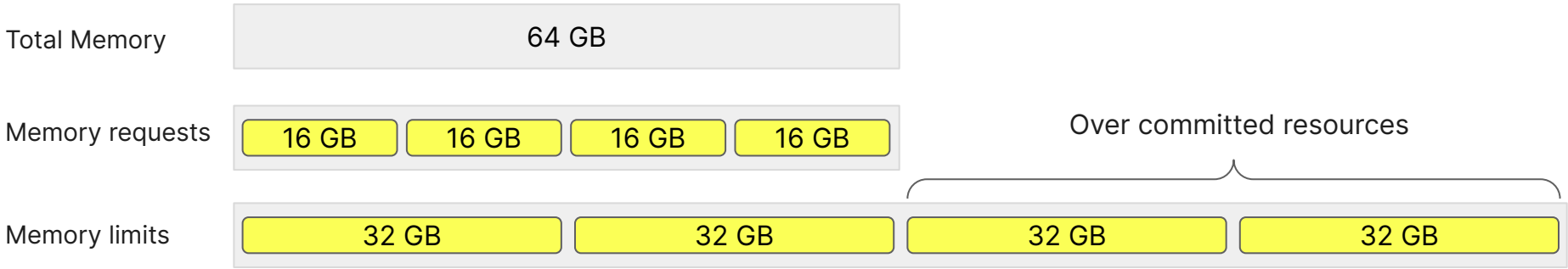
1. **Increase node resource utilization across fleet**

2. **Minimize disruption to customer**
   a. Long running insert queries do not want to be interrupted.
   b. No increase in instance provisioning/cold-start time for CH instances.

3. **Multi-Cloud friendly**
   a. As we provide services in all 3 major Cloud providers.

Approach #1 - Over-commit resources

- Set resource requests to be less than limits.
- QoS burstable.
- Can schedule more pods on each node.
- Assumes that not all pods use resources up to limits.



~~Approach #1 - Over-commit resources~~

Why it doesn’t work for us

- ❌ Noisy neighbors can cause service being OOM killed or throttled.
- ❌ QoS guaranteed - We always set resource limit and request. The same on ClickHouse pods to ensure QoS. We would need to change this to QoS burstable.
- ❌ Customer doesn’t get exactly the resources they paid for.

Approach #2 - Tuning cluster AutoScaler

- ***scale-down-utilization-threshold*** - Setting to tune utilization threshold for node scale-down in cluster autoscaler.
- If utilization in node is less than threshold, it is a candidate for scale down.
- Used to identify excess capacity in cluster.

~~Approach #2 - Tuning cluster AutoScaler~~

Why it doesn't work for us

- ❌ Frequent evictions of pods is too disruptive to stateful workloads.
- ❌ Violates requirement of not interrupting long running queries.
- ❌ Not enough fine-grained control on evictions.

Approach #3 - Descheduler

Descheduler, https://github.com/kubernetes-sigs/descheduler

- Counterpart to ***kube-scheduler.***
- Pod evictions based on constraints/plugins.
- HighNodeUtilization strategy plugin.

~~Approach #3 - Descheduler~~

Why it doesn't work for us

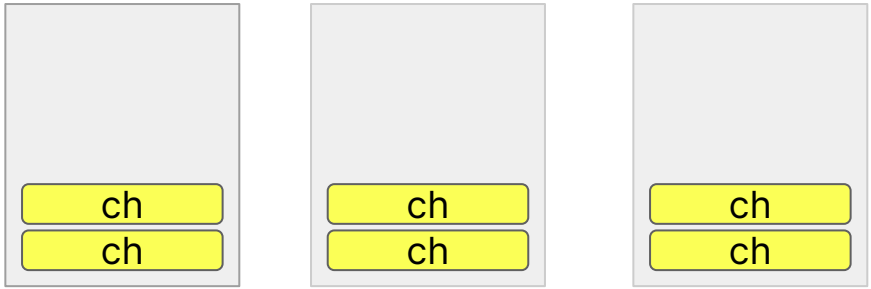
- ❌ Frequent evictions of pods is too disruptive to stateful workloads.
- ❌ Violates requirement of not interrupting long running queries.
- ❌ Not enough fine-grained control on evictions.

Similar problems with solutions #2 & #3

1. Cluster autoscaler and descheduler both work on evicting pods and scaling down nodes but not on scheduling the evicted pods.
2. Frequent evictions of pods which are still serving queries is too disruptive.
3. We decided to focus on optimizing packing during pod scheduling.

Default scheduler behavior

- Kubernetes scheduler does ***LeastAllocated*** scheduling by default. Favors nodes with lower utilization during pod scheduling.
- This makes node scale down unlikely.



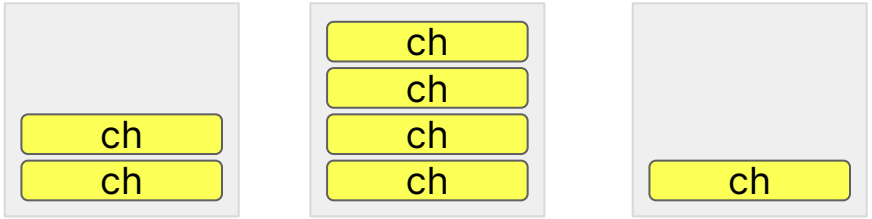
Approach #3 - MostAllocated Scoring Policy

- **KubeSchedulerConfiguration:**
  - Allows for a configurable scoring strategy, CPU, memory, weights, etc.
  - **NodeResourcesFit Plugin**: **MostAllocated**, LeastAllocated(default), RequestedToCapacityRatio.
- **MostAllocated** works for us.

```
profiles:
        - pluginConfig:
            - args:
                apiVersion: kubescheduler.config.k8s.io/v1beta3
                kind: NodeResourcesFitArgs
                scoringStrategy:
                    resources:
                        - name: cpu
                          weight: 1
                        - name: memory
                          weight: 1
                    type: MostAllocated
              name: NodeResourcesFit
          plugins:
            score:
                enabled:
                    - name: NodeResourcesFit
                      weight: 1
          schedulerName: <schedulerName>
```

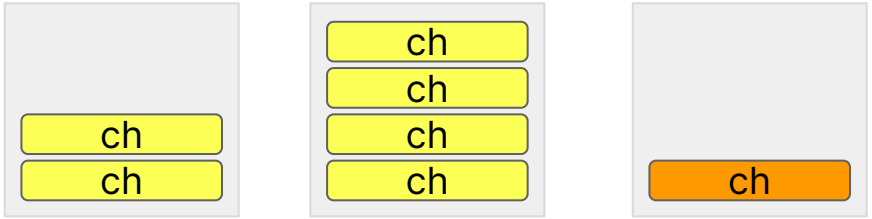
MostAllocated Scoring Policy

- **KubeSchedulerConfiguration:**
  - Allows for a configurable scoring strategy, CPU, memory, weights, etc.
  - **NodeResourcesFit Plugin**: **MostAllocated**, LeastAllocated(default), RequestedToCapacityRatio.
- **MostAllocated** works for us.



MostAllocated Scoring Policy

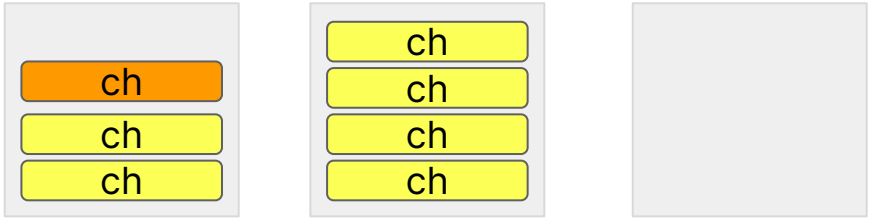
- **KubeSchedulerConfiguration:**
  - Allows for a configurable scoring strategy, CPU, memory, weights, etc.
  - **NodeResourcesFit Plugin**: **MostAllocated**, LeastAllocated(default), RequestedToCapacityRatio.
- **MostAllocated** works for us.



A ClickHouse pod is about to be stopped/idled/restarted

MostAllocated Scoring Policy

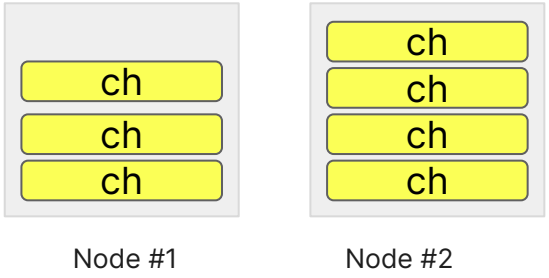
- **KubeSchedulerConfiguration:**
  - Allows for a configurable scoring strategy, CPU, memory, weights, etc.
  - **NodeResourcesFit Plugin**: **MostAllocated**, LeastAllocated(default), RequestedToCapacityRatio.
- **MostAllocated** works for us.



Leaving this node empty; new nodes scheduled onto the most allocated one.

MostAllocated Scoring Policy

- **KubeSchedulerConfiguration:**
  - Allows for a configurable scoring strategy, CPU, memory, weights, etc.
  - **NodeResourcesFit Plugin**: **MostAllocated**, LeastAllocated(default), RequestedToCapacityRatio.
- **MostAllocated** works for us.



Cluster Autoscaler can scale down the node.

MostAllocated: Requirements Check

- **Requirement 1: Increase node utilization across fleet.**
  - Higher number of pods/node → better utilization, lower number of nodes.
- **Requirement 2: Low impact to customer.**
  - Scheduler scoring policy update will not trigger pod restart.
  - Policy will take effect in subsequent scheduling decisions.
- **Requirement 3: Multi-Cloud friendly.**
  - Kubernetes native solution. Applies to all clouds.

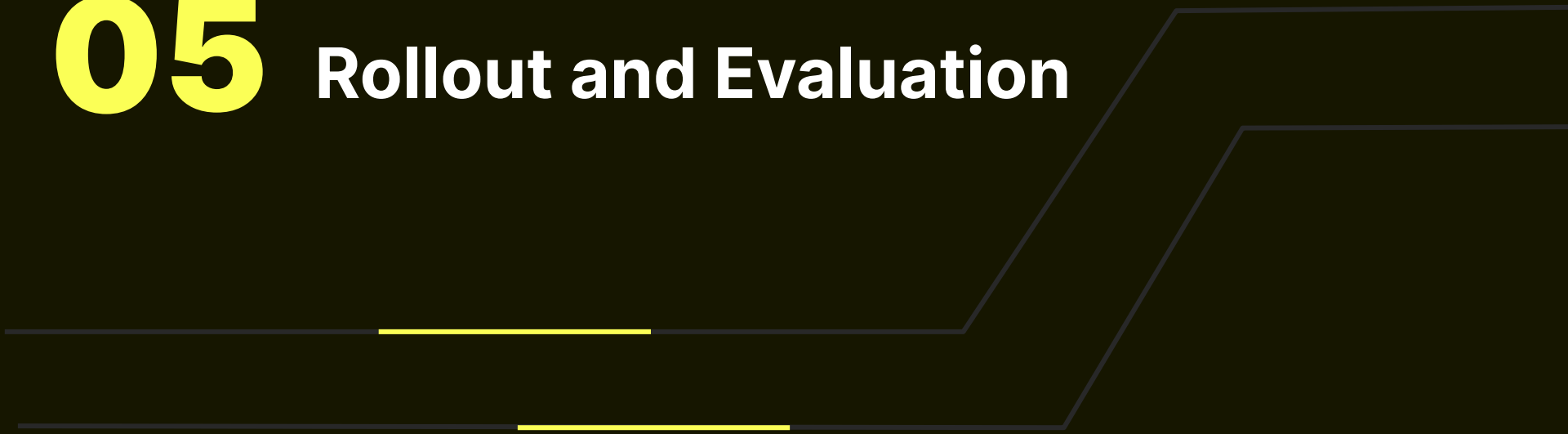
Not so fast! ✋

- **EKS** (**AWS**) and **AKS** (**Azure**) do not support the needed setting updates for the Kubernetes control plane.
- **GCP** offers an ***optimized_utilization*** setting that helps save costs.
- How do we set MostAllocated scheduler policy in Managed Kubernetes?

Secondary scheduler

- Natively supported by Kubernetes.
- Runs in parallel to kube-scheduler. Non CH server pods can still be scheduled by default kube-scheduler.
- Kube-scheduler image with ***MostAllocated*** scoring strategy.
- Deployed in ***kube-system*** namespace to prevent evictions/scale-down.

05 Rollout and Evaluation



ClickHouse

Rollout process

- Create deployment of scheduler.
- Update ClickHouse scheduler.
- Smaller clusters first.
- Fleet wide restart (as scheduler is part of PodSpec).
- Evaluation and measurement.

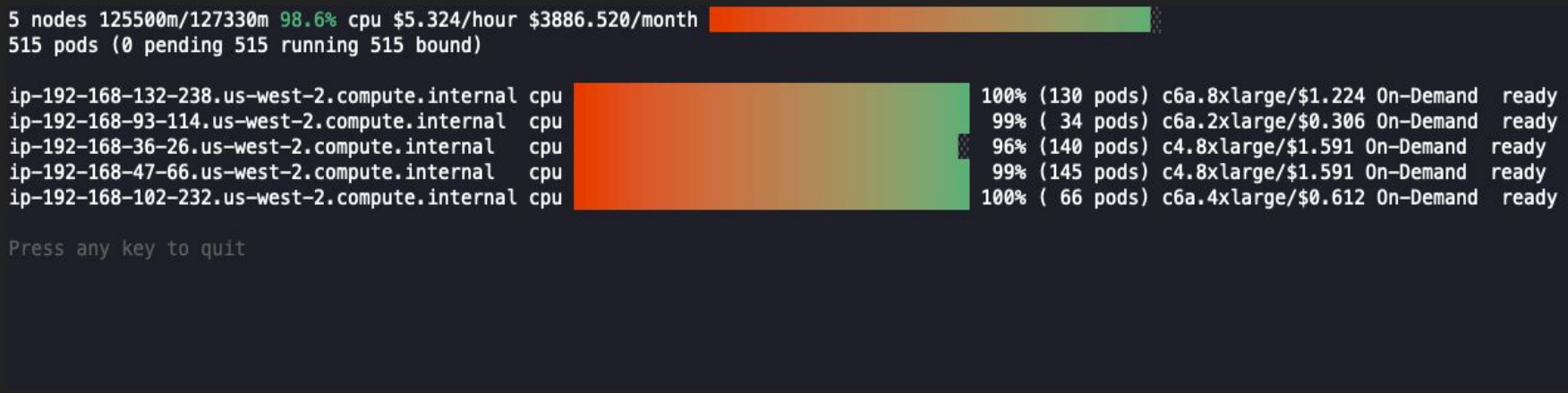
MostAllocated Scheduler Setup

- Deployment with 3 pods.
  - Same upstream kube-scheduler with MostAllocated scoring policy.
  - 3 pods for HA setup with leader election enabled

- Basic reliability test.
  - Constantly schedule lots of pods.
  - Restart leader scheduler.
  - Secondary scheduler take over quickly.

Measurement tooling, ad-hoc analysis

- EKS node viewer, https://github.com/awslabs/eks-node-viewer
  - Visualize CPU/memory utilization and cost.



Measurement tooling, final evaluation

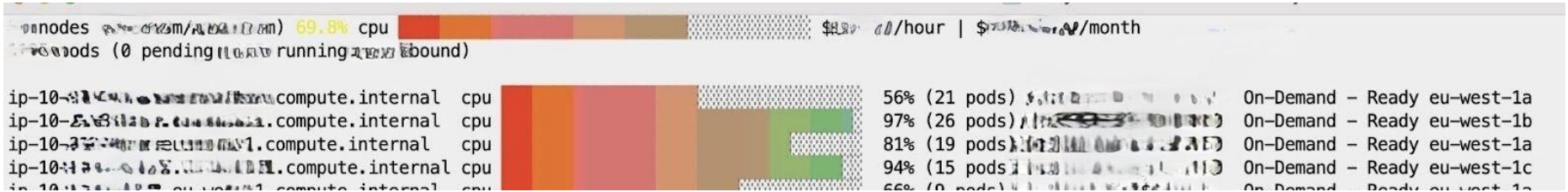
- Internal data warehouse.
  - Also runs on ClickHouse Cloud!
  - Importing AWS cost and usage data.
  - SuperSet as UI layer for on-demand analysis.

Rollout: start from small cluster first

- No significant savings.
- Need to have some minimal regardless.
  - One node for each machine type at each AZ.
- But gain confidence for production safety.

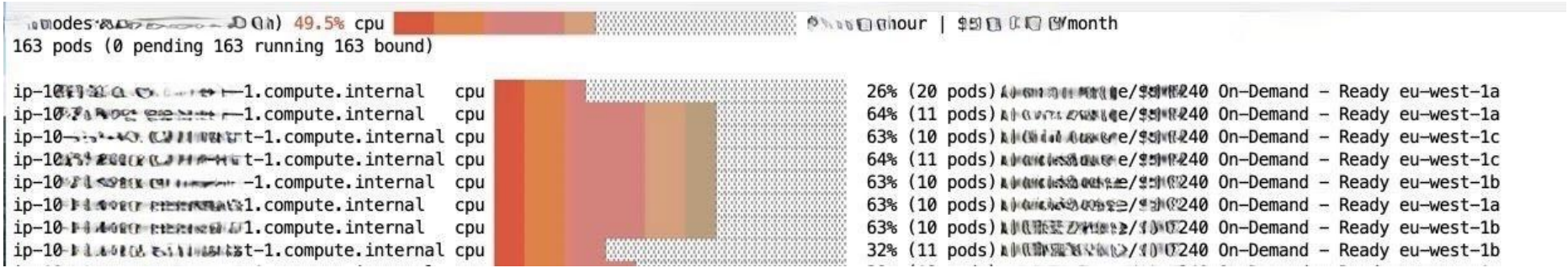
Continue with larger cluster (**before the rollout**)

- Some larger cluster has more significant savings.

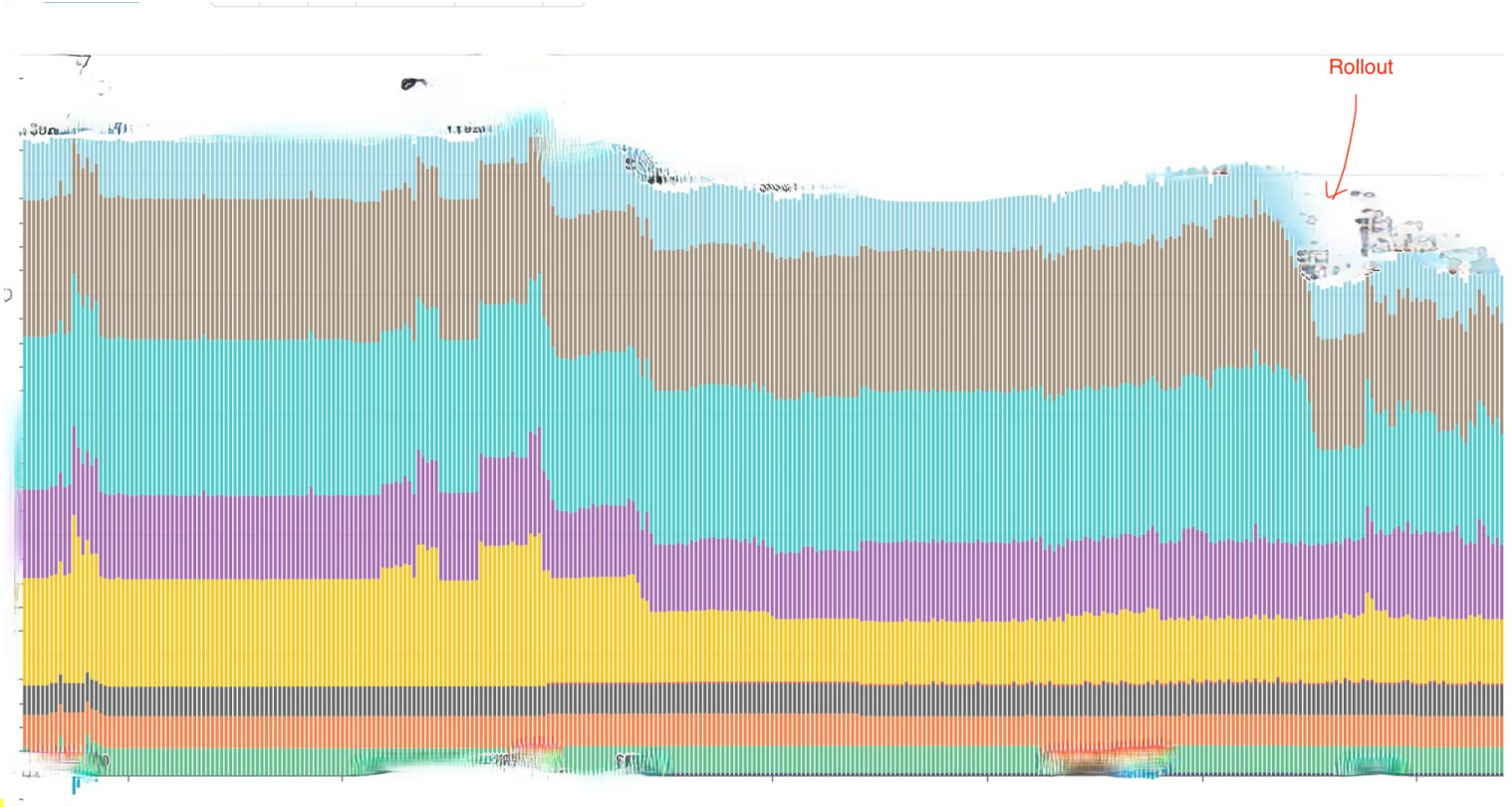


Continue with larger cluster (**after the rollout**)

- Resource utilization went up from 50 % to 70%.
- Nodes are reclaimed by cluster autoscaler.
  - 10% - 15% cost savings.

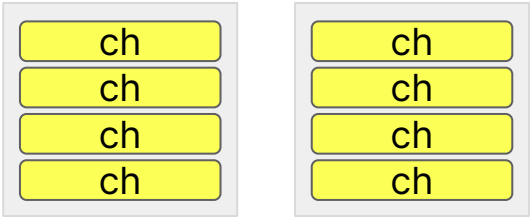


Now, it’s time to look at cost savings (data warehouse)



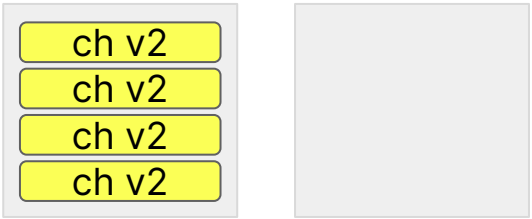
Utilization Over Time

- High utilization after most allocated scheduler repacked entire fleet.
- Utilization dropped when services being idle, or terminated.



Utilization Over Time

- High utilization after most allocated scheduler repacked entire fleet.
- Utilization dropped when services being idle, or terminated.
- **Regular release restarts all pods, repacking the fleet.**



Findings during the rollout

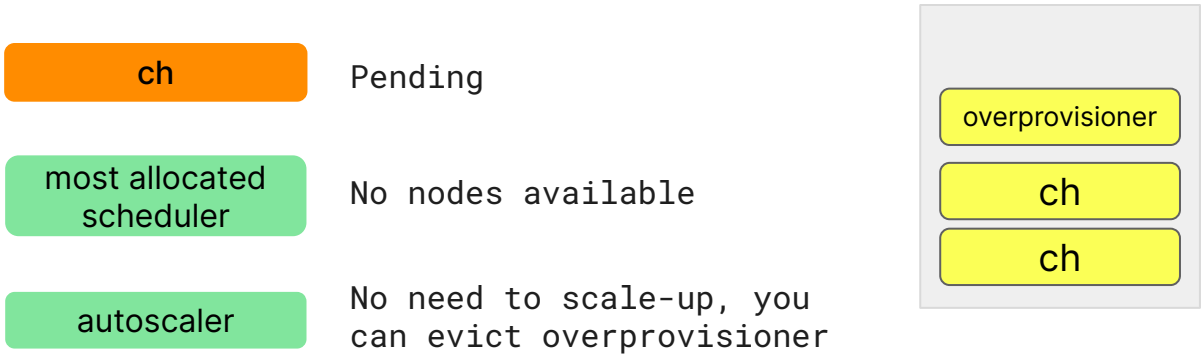
- System workloads preventing node scaling down.
- Unschedulable pods mystery.
- Checking cold start time.

System Workloads

- System utility workloads:
  - A large node is not scaled down because of system workloads running.
  - Using ephemeral volume prevent cluster autoscaler from reclaiming the node.
  - Examples, ArgoCD controller, CoreDNS, EBS CSI controller.
- Add `safe-to-evict` label.

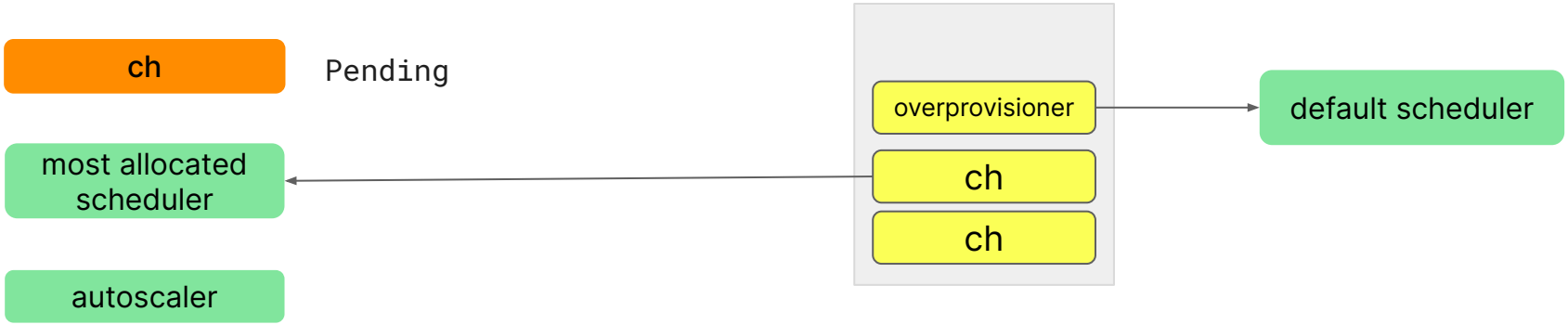
Unschedulable pods

- Pods are stuck in pending state.
- Cluster autoscaler and most allocated scheduler think differently.



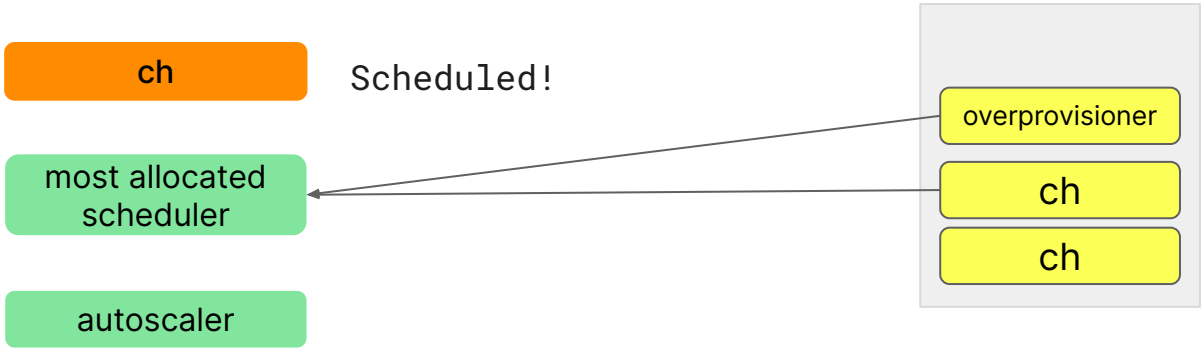
Unschedulable pods

- Kubernetes uses different queues for different scheduler
  - *When Pods are created, they go to a queue and wait to be scheduled. The scheduler picks a Pod from the queue and tries to schedule it on a Node. If no Node is found that satisfies all the specified requirements of the Pod, preemption logic is triggered for the pending Pod*



Unschedulable pods

- Kubernetes uses different queues for different scheduler
  - *When Pods are created, they go to a queue and wait to be scheduled. The scheduler picks a Pod from the queue and tries to schedule it on a Node. If no Node is found that satisfies all the specified requirements of the Pod, preemption logic is triggered for the pending Pod*
- **Solution, use most allocated scheduler for over-provisioner**



Cold-start time

- In theory, cold-start time can increase.
  - Clusters are more packed, when new pods are scheduled, more likely to request for a new node.
- Measurement:
  - Confirmed not happened.
  - Over-provisioner buffer the scheduling.

Other Cloud Providers

- GCP, optimized-utilization profile, achieves similar effect.
- Azure, deploy customized scheduler similar to EKS.

Other Cloud Providers

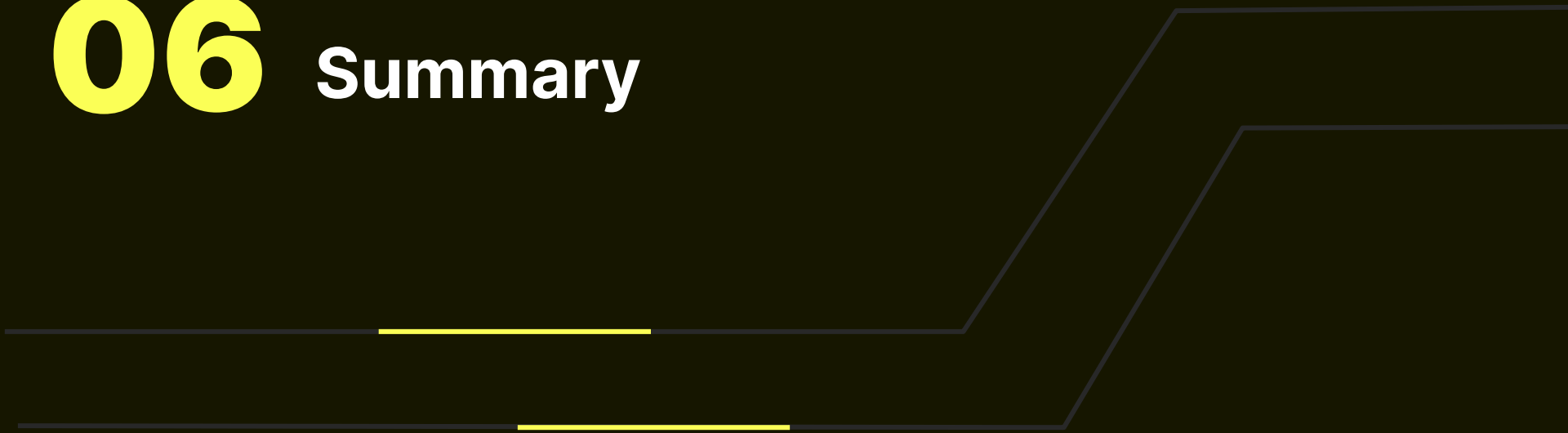
GCP

Optimized-utilization profile, achieves similar effect.

Azure

Deploy customized scheduler similar to EKS

06 Summary



ClickHouse

Summary

- ✅ Improve resource utilization for a multi-tenant Kubernetes hosting environment.
- ✅ Minimize the disruption to the workloads.
- ✅ Evaluated a few approaches.
  - Kubernetes scheduler configuration + most allocated strategy.
  - Addressed a few issues during the rollout.
- ✅ The rollout improves 15 - 20% resource utilization.

A few things to remember if you want to bin-pack

- Scheduler scoring strategy.
- QoS guaranteed to avoid noisy neighbor.
- Re-pack the fleet periodically.
- Overprovision to avoid node provisioning delay.
- As cluster becomes bigger, fragmentation issue can be mitigated naturally.

Thank you!

- Blog “Saving Millions of Dollars by Bin-Packing ClickHouse Pods in AWS EKS”.
- Fantastic Ordinals and How to Avoid Them: Auto-Scaling Challenges in a Cloud Database - Manish Gill, **Thursday March 21, 4:30 PM. This room.**
- We are #hiring!