



Enhancing KubeVirt workload scheduling patterns

Simone Tiraboschi - *Red Hat*
FOSDEM 2025

CONTROVERSIAL



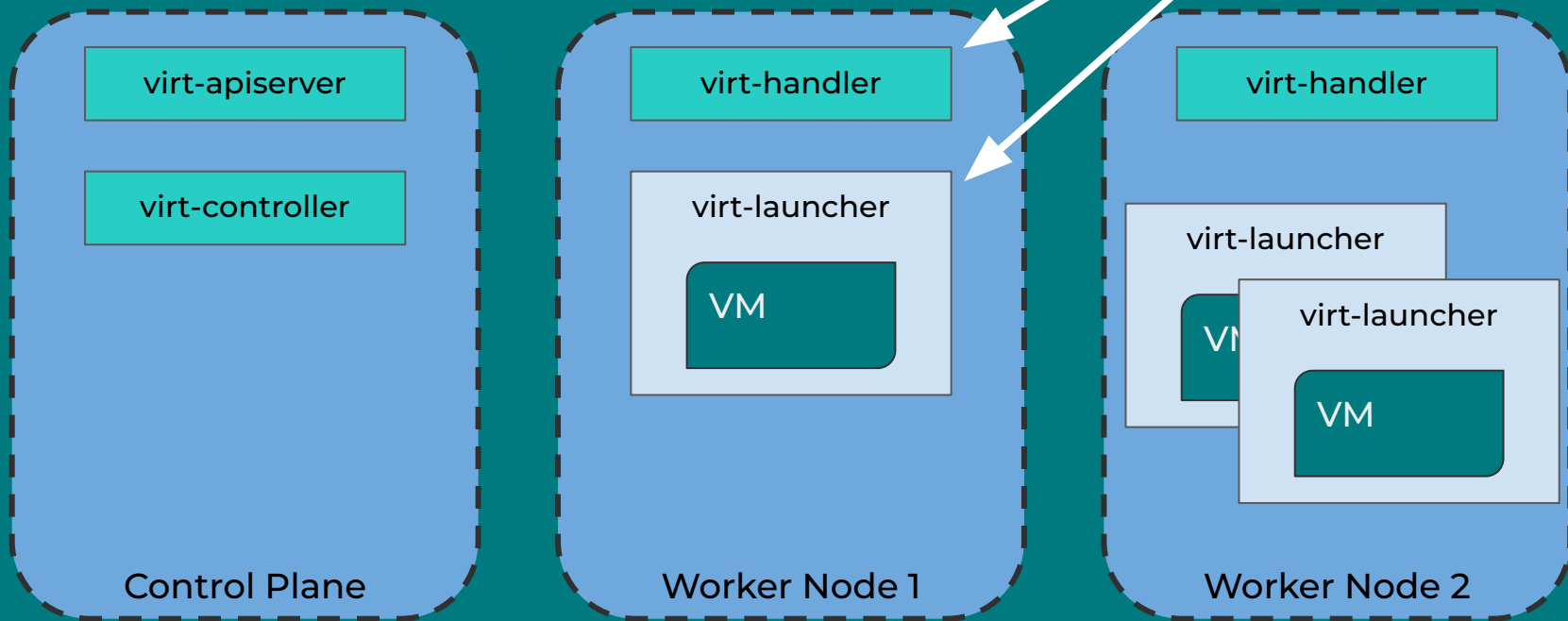
Scheduling is the process of matching workload to Nodes.

By default, the scheduler used is kube-scheduler.



Architecture

Pods!

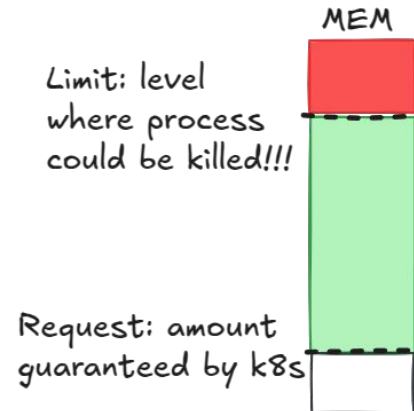
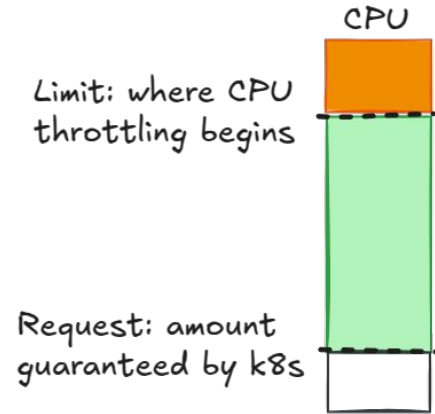




K8s workload resources

Request and limit

- **Request:** amount of a resource allowed to be used, with a strong guarantee of availability
 - CPU (seconds/second), RAM (bytes)
 - Scheduler will not overcommit requests
- **Limit:** max amount of a resource that can be used, regardless of guarantees
 - **scheduler ignores limits**
- Implications:
 - request < **usage** <= limit: resources might be available
 - **usage** > limit: throttled (CPU) or killed (memory)



VM Resources

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: testvm
  ...
spec:
  ...
  template:
    spec:
      ...
    domain:
      cpu:
        cores: 1
        sockets: 4
        threads: 1
      memory:
        guest: 2Gi
```

In Kubernetes, one full core is 1000 of CPU time
CPU is REQUESTed according to CPU overcommit ratio (10 by default):

```
spec:
  configuration:
    developerConfiguration:
      cpuAllocationRatio: 10
```

POD Resources

```
apiVersion: v1
kind: Pod
metadata:
  name: virt-launcher-testvm
spec:
  ...
  containers:
    ...
    name: compute
    resources:
      limits:
        devices.kubevirt.io/kvm: "1"
      requests:
        cpu: 400m
        devices.kubevirt.io/kvm: "1"
        ephemeral-storage: 50M
        memory: 2299Mi
```

We have also VMs with Guaranteed QoS class:
requests = limits
No overcommit

Memory overcommit is disabled by default (*):
The whole guest OS configured memory plus additional overhead for ancillary components is required to avoid getting killed by OOM

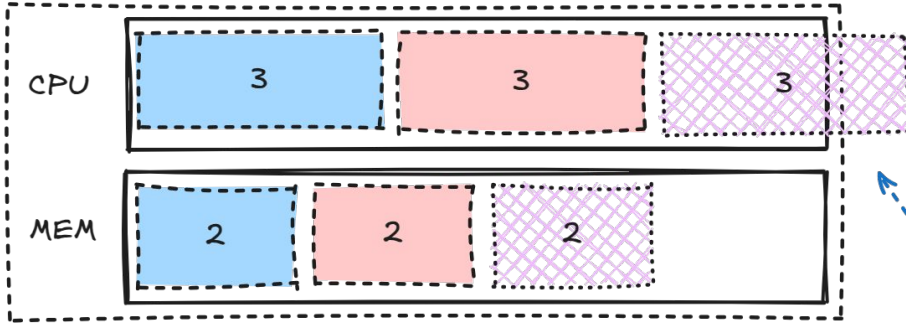
* see: https://kubevirt.io/user-guide/compute/node_overcommit/#overcommit-guest-memory



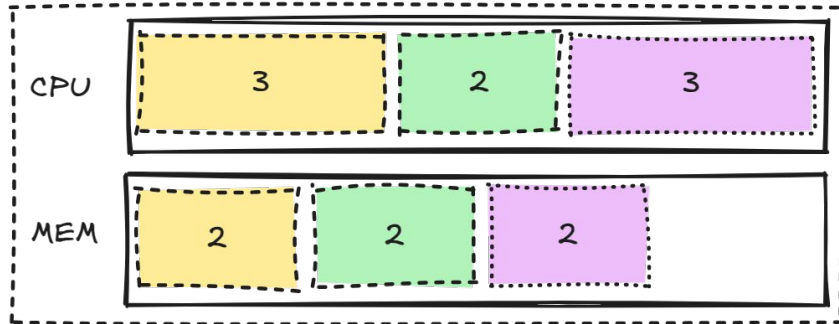


K8s scheduling

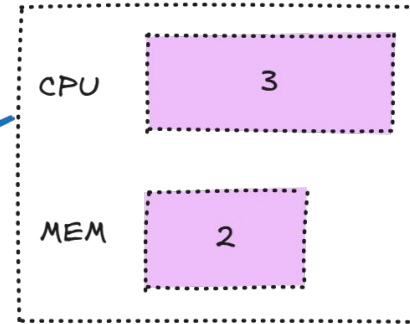
Node A



Node B

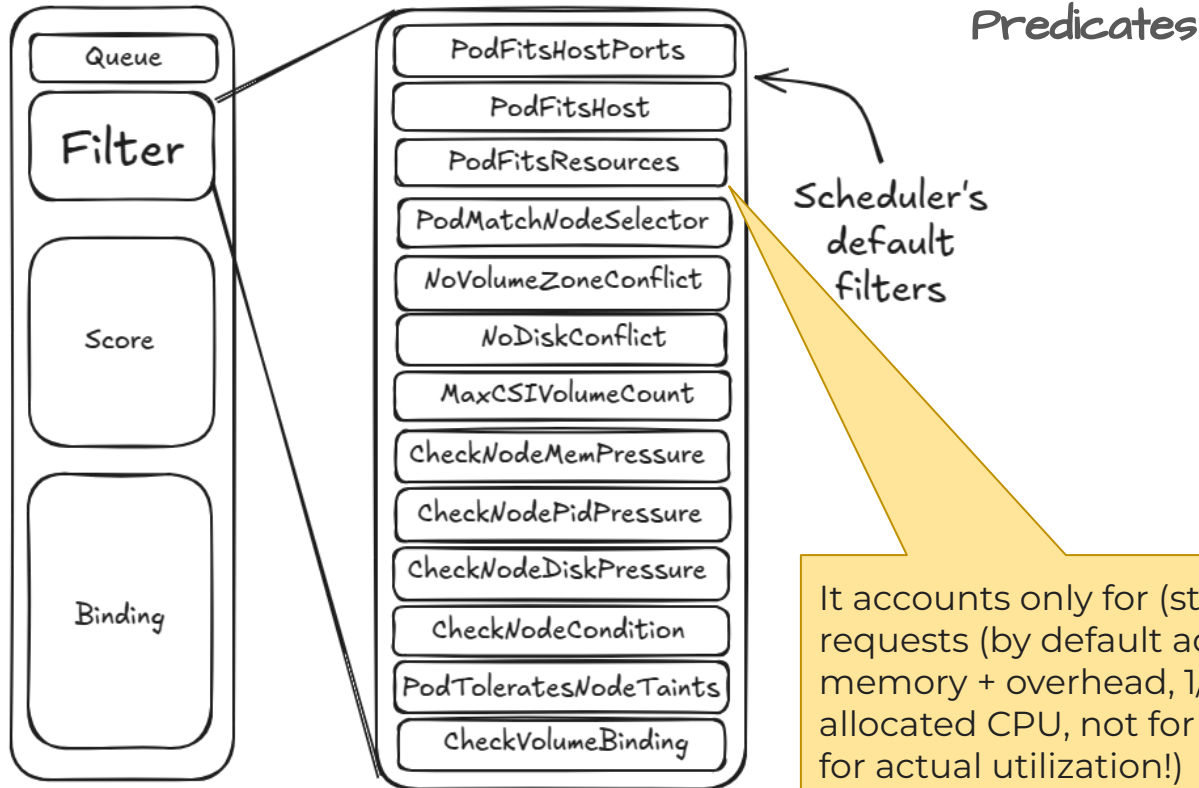


Pod



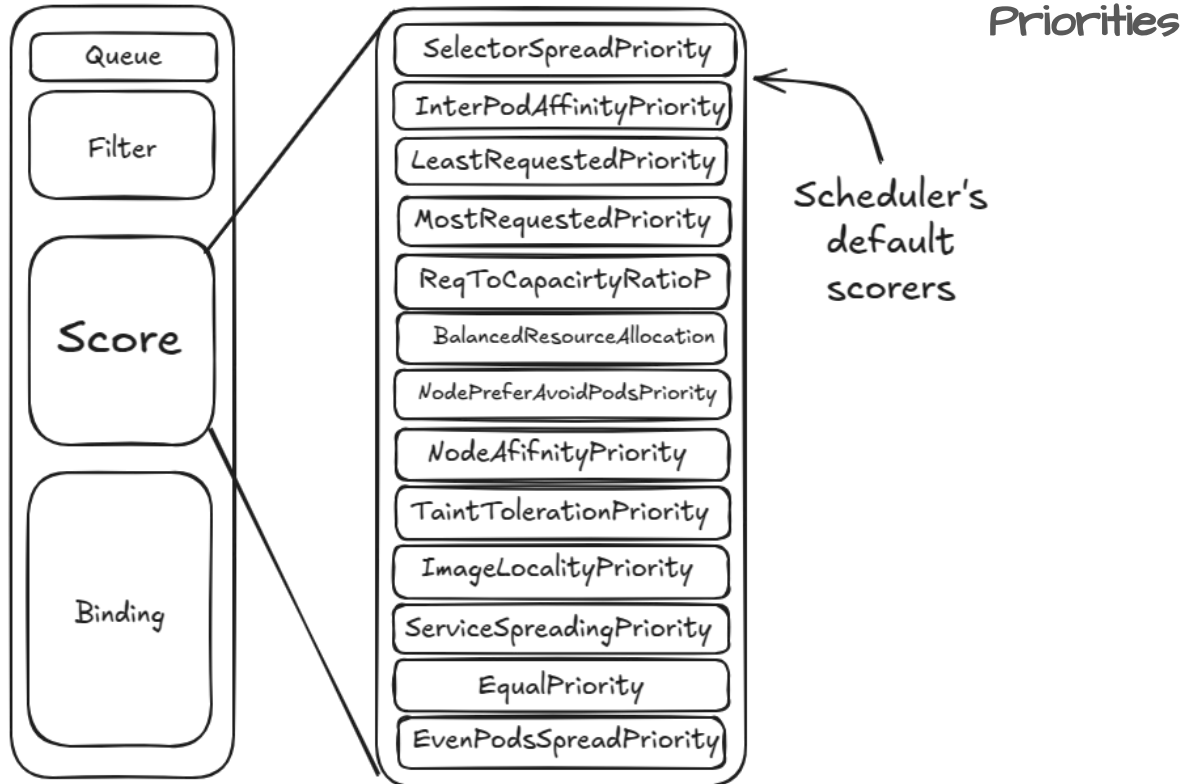


How k8s scheduler works 1/2





How k8s scheduler works 2/2



VMs

- Each VM runs a separate OS, providing stronger isolation
- Require more resources (CPU, RAM, storage) due to running full OS per each VM
- Slower startup time due to OS booting
- Usually stateful with data on local disks
- Can be moved between nodes with live-migration with zero-downtime
- Scaling requires rebooting the VM with a different configuration or hotplugging resources: slower and intensive
- Potentially really long uptime

PODs

- Same kernel OS, isolated user spaces
- (Ideally) lighter
- Fast startup
- Typically stateless, eventually with data on persistent volumes that can be attached
- Cannot be moved between nodes with live-migration but they can be quickly killed and restarted on a different node
- Pods can be "easily scaled" to meet dynamic application requirements
- Supposedly shorter life cycle



VMs Scheduling hints

PODs

- nodeSelector

```
nodeSelector:  
  performance: high
```

- nodeAffinity

```
affinity:  
  nodeAffinity:  
    requiredDuringSchedulingIgnoredDuringExecution:  
      nodeSelectorTerms:  
        - matchExpressions:  
          - key: performance  
            operator: In  
            values:  
              - high
```

- podAffinity/podAntiAffinity

```
podAntiAffinity:  
  requiredDuringSchedulingIgnoredDuringExecution:  
    - labelSelector:  
      matchExpressions:  
        - key: app  
          operator: In  
          values:  
            - cache
```

- nodeSelector

```
nodeSelector:  
  performance: high
```

- nodeAffinity

```
affinity:  
  nodeAffinity:  
    requiredDuringSchedulingIgnoredDuringExecution:  
      nodeSelectorTerms:  
        - matchExpressions:  
          - key: performance  
            operator: In  
            values:  
              - high
```

- podAffinity/podAntiAffinity

```
podAntiAffinity:  
  requiredDuringSchedulingIgnoredDuringExecution:  
    - labelSelector:  
      matchExpressions:  
        - key: app  
          operator: In  
          values:  
            - cache
```



User habits

oVirt

The screenshot shows the oVirt web interface. On the left is a navigation sidebar with options: Dashboard, Compute, Network, Storage, Administration, and Events. The main content area is titled 'Compute > Virtual Machines'. It features a table with columns for Name, Status, Name, Comment, Host, IP Address, and Cluster. A context menu is open over the table, listing actions: Remove, Run, Suspend, Shutdown, Reboot, View Console, Migrate (highlighted with a pink circle), and Create Snapshot. Below the table, a 'Migrate VM(s)' dialog box is open, showing 'Migrate 11 Virtual Machines to the selected Host.' and a 'Destination Host' dropdown set to 'Host 3'. The list of VMs to be migrated includes Virtual Machine 1 through 14, with Virtual Machine 11 highlighted in blue.

Name	Status	Name	Comment	Host	IP Address	Cluster
<input type="checkbox"/>		Virtual Machine 1		Host 1		
<input checked="" type="checkbox"/>		Virtual Machine 2		Host 1		
<input checked="" type="checkbox"/>		Virtual Machine 3		Host 1		
<input checked="" type="checkbox"/>		Virtual Machine 4		Host 1		
<input checked="" type="checkbox"/>		Virtual Machine 5		Host 1		
<input checked="" type="checkbox"/>		Virtual Machine 6		Host 1		
<input checked="" type="checkbox"/>		Virtual Machine 7		Host 1		
<input checked="" type="checkbox"/>		Virtual Machine 8		Host 1		
<input checked="" type="checkbox"/>		Virtual Machine 9		Host 1		
<input checked="" type="checkbox"/>		Virtual Machine 10		Host 1		
<input checked="" type="checkbox"/>		Virtual Machine 11		Host 1		
<input checked="" type="checkbox"/>		Virtual Machine 12		Host 1		
<input type="checkbox"/>		Virtual Machine 13		Host 1		
<input type="checkbox"/>		Virtual Machine 14		Host 1		
<input type="checkbox"/>		Virtual Machine 15		Host 1		

The screenshot shows the 'Migrate VM 99999' dialog box in Proxmox VE. It has a 'Source node' field set to 'prod1' and a 'Target node' dropdown menu set to 'prod2'. The 'Mode' is set to 'Offline'. There is a 'Help' button on the left and a 'Migrate' button on the right.

Proxmox VE

VMware vCenter

Migrate | VMware vCenter Server (Preferred)

The screenshot shows the VMware vCenter migration wizard. It has a progress bar with six steps: 1. Select a migration type, 2. Select a compute resource (highlighted), 3. Select storage, 4. Select networks, 5. Select vMotion priority, and 6. Ready to complete. On the right, a tree view shows the compute resource selection path: vcenter01.dell.lab > Datacenter > Stretched_Cluster01 > esx02.dell.lab (Preferred). Below the tree, it says 'Compatibility checks succeeded.'

Initiate live migration on KubeVirt

```
apiVersion: kubevirt.io/v1
```

```
kind: VirtualMachineInstanceMigration
```

```
metadata:
```

```
  name: migration-job
```

```
  namespace: mynamespace
```

```
spec:
```

```
  vmiName: testvm
```

It's a namespaced object

You can only specify the name of a VM within the VMIM namespace, everything else will be up to the scheduler...

```
$ virtctl migrate <vmname>
```



A bit of history...

Support specify one node to migrate #32

Closed zhonglin6666 wants to merge 1 commit into `kubevirt:main` from `zhonglin6666`

Conversation 34 Commits 1 Checks 0 Files changed 1



zhonglin6666 commented on Nov 10, 2023

What this PR does / why we need it:

The current migration of virtual machines involves setting node affinity or node selectors, but cannot be done by specifying a fixed node. Now there are currently two ways to migrate:

- Migrate to the specified node through the `virtctl migrate vm node CO` command.
- Migrate to the selected node by scheduler through the `virtctl migrate vm` command.

`kubectl get vmim`

NAME	PHASE	VMI
kubevirt-migrate-vm-2snmc	Succeeded	vm-test

design-proposal: VirtualMachineInstanceMigration - Live migration to a named node #320

Open tiraboschi wants to merge 4 commits into `kubevirt:main` from `tiraboschi:migration_target`

Conversation 178 Commits 4 Checks 0 Files changed 1



tiraboschi commented on Sep 3, 2024 • edited

What this PR does / why we need it:

Adding a design proposal to extend `VirtualMachineInstanceMigration` object with an additional API to let a cluster admin try to trigger a live migration of a VM injecting on the fly and additional `NodeSelector` constraint. The additional `NodeSelector` can only restrict the set of Nodes that are valid target for the migration (eventually down to a single host). All the affinity rules defined on the VM spec are still going to be satisfied.

Which issue(s) this PR fixes (optional, in `fixes #<issue number>` format, will close the issue(s) when PR gets merged):

Fixes <https://issues.redhat.com/browse/CNV-7075>

Special notes for your reviewer:

Something like this was directly proposed/implemented with [kubevirt/kubevirt#10712](https://github.com/kubevirt/kubevirt/pull/10712) getting already discussed there.

area/virtctl dco-signoff: yes

do-not-merge/release-note-label-needed

kind/api-change lifecycle/rotten

needs-ok-to-test size/L

Reviewers

holder101

enp0s3

alicefr

dankenigsberg

vladikr

davidvossell

fabiano

EdDev

aburdenthehand

jobbler

Still in progress? Convert to draft



<https://kubevirt.io/>

It's not a new idea...



Motivation

User stories

- As a cluster administrator:
 - Experienced admins are used to control where their critical workloads are going to be moved to
 - **Habits**
 - **Existing patterns/automation** on/over previous VM management solutions
 - **Planned maintenance** activities on nodes
 - Workload balancing solution doesn't always work as expected
 - I can anticipate load profile changes
 - **Troubleshooting** a node
 - **Validating** a new node migrating there a specific VM
- As a VM owner:
 - I don't want to see my VM object getting amended by another user/admin just for maintenance reasons



Goals

- A user allowed to trigger a live-migration of a VM **limiting** its admitted **target** to a subset of nodes
- The target node that is explicitly required for the actual live migration attempt **should not influence future live migrations** or the placement in case the VM is restarted
- The constraints directly added on the one-off migration can only complement and **limit** constraints already defined on the VM object (**pure AND logic**)
- It's a **one off migration attempt**: it could successfully complete or fail as it can already do today

Proposed design

```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstanceMigration
metadata:
  name: migration-job
  namespace: mynamespace
spec:
  vmiName: testvm
  addedNodeSelector:
    accelerator: gpuenabled123
    kubernetes.io/hostname: "ip-172-20-114-199.example"
```

A node will be a valid target if it will match all the node selector constraints specified on the VM AND additional node selectors specified here

```
$ virtctl migrate <vmname> --addedNodeSelector key1=value1,key2=value2
```

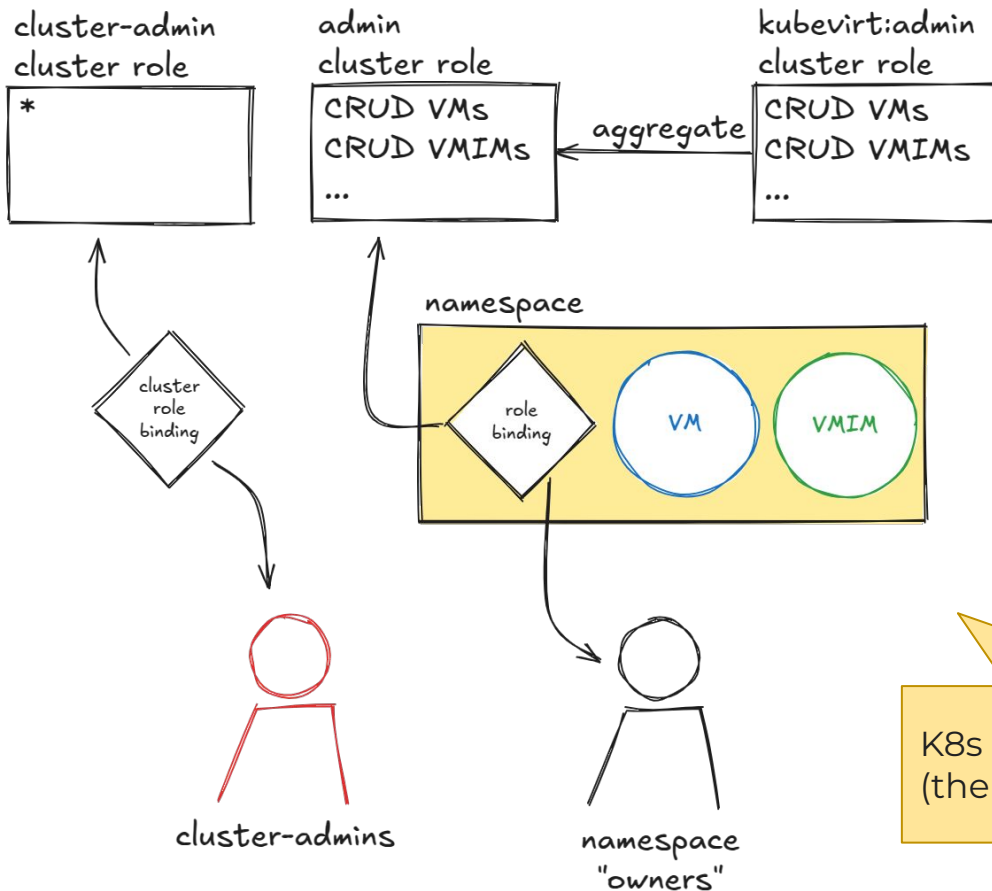


Criticisms and concerns

- **K8s is the scheduler**, the user should not have the control
 - OK, but...
- On K8s we cannot live migrate a pod to a named node
 - OK, but we cannot live migrate a pod at all
 - And without **KubeVirt** we neither have VMs: this is a **VM** specific problem so it should be solved in KubeVirt
- We have other **k8s native paradigms** to "**individually**" address many if not all of the user stories there (e.g. combinations of taints and tolerations, draining and/or cordoning/uncordoning nodes in a specific sequence)
 - Correct but different tasks requires different strategies and they could be **less obvious** for less trained user. We don't have a simply solution to rule them all
- Live migrations are **resource expensive operations**
 - The number of parallel live migrations is capped (by default 5) and we have a single migration queue
 - Migrations are also used for critical infra operations (node drains, upgrades, hotplugging...)
 - "Namespace owners" are currently able to trigger live migrations, we fear that enhancing migration capabilities we could end with users "abusing" that capability
 - But we can make this better...

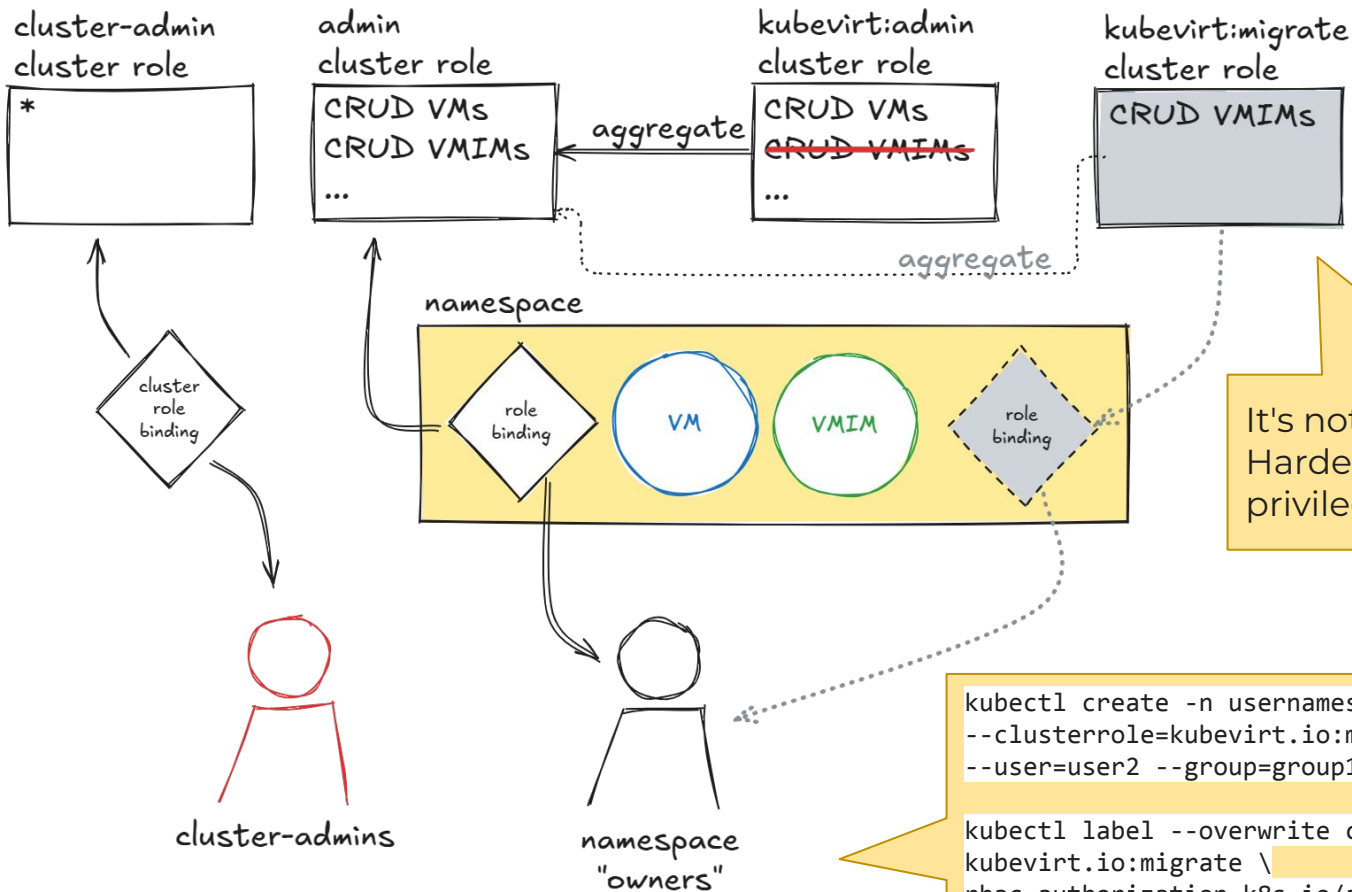


KubeVirt RBAC model



K8s RBAC model is purely additive (there are no "deny" rules).

KubeVirt RBAC model (KubeVirt 1.6)



It's not an API change.
Hardening: principle of least privilege

```
kubectl create -n usernamespace rolebinding kvmigrate \
--clusterrole=kubevirt.io:migrate --user=user1 \
--user=user2 --group=group1

kubectl label --overwrite clusterrole \
kubevirt.io:migrate \
rbac.authorization.k8s.io/aggregate-to-admin=true
```

KubeVirt Razor: "If something is useful for Pods, we should not implement it only for VMs".

But this is a VM specific topic...



Alternatives: 1 - amend VM node affinity

1. set a (temporary?) nodeSelector/nodeAffinity on the VM
2. wait for it to be propagated to the VMI due to LiveUpdate rollout strategy
3. trigger a live migration with existing APIs (no need for any code change)
4. wait for the migration to complete
5. (eventually) remove the (temporary?) nodeSelector to let the VM be freely migrate to any node in the future



- Imperative flow
- Error prone
- It has still to be somehow "orchestrated"
- It can mess with devops/laC approaches



Alternatives: 2 - configure a secondary scheduler for VMs

Trimaran: Load-aware scheduling plugins

Trimaran is a collection of load-aware scheduler plugins described in [Trimaran: Real Load Aware Scheduling](#).

Currently, the collection consists of the following plugins.

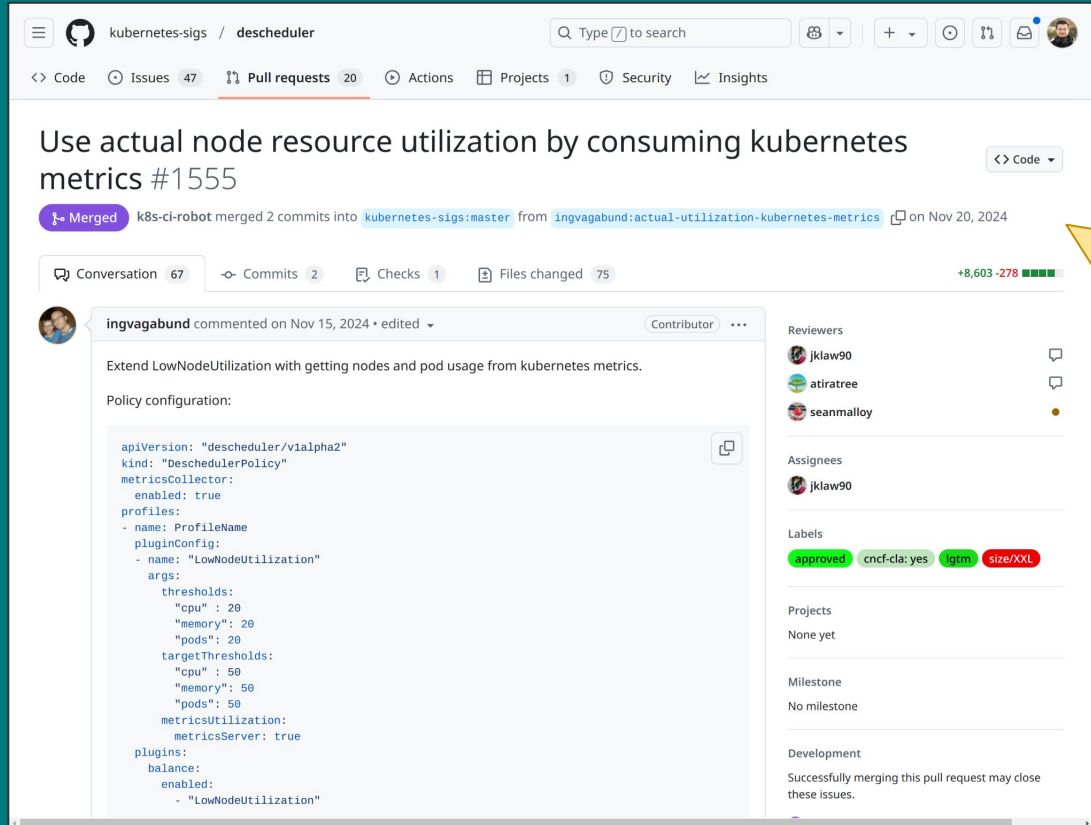
- **TargetLoadPacking** : Implements a packing policy up to a configured CPU utilization, then switches to a spreading policy among the hot nodes. (Supports CPU resource.)
- **LoadVariationRiskBalancing** : Equalizes the risk, defined as a combined measure of average utilization and variation in utilization, among nodes. (Supports CPU and memory resources.)
- **LowRiskOverCommitment** : Evaluates the performance risk of overcommitment and selects the node with the lowest risk by taking into consideration (1) the resource limit values of pods (limit-aware) and (2) the actual load (utilization) on the nodes (load-aware). Thus, it provides a low risk environment for pods and alleviate issues with overcommitment, while allowing pods to use their limits.

The Trimaran plugins utilize a [load-watcher](#) to access resource utilization data via metrics providers. Currently, the `load-watcher` supports three metrics providers: [Kubernetes Metrics Server](#), [Prometheus Server](#), and [SignalFx](#).

- Yes, but...
 - The cluster admin should deploy the **custom scheduler**
 - **Each individual VM** should be configured to be scheduled by the secondary scheduler and VM owners or devops flows could revert it
- Still only about scheduling according to static **reservation** (by default 1/10 of allocated cores), not actual utilization
- It's **not going to continuously rebalance the cluster** according to changes in the usage patterns



Alternatives: 3 - use Kube Descheduler for automatic workload rebalancing



The screenshot shows a GitHub pull request titled "Use actual node resource utilization by consuming kubernetes metrics #1555". The pull request is merged and was submitted by k8s-ci-robot on Nov 20, 2024. The description of the pull request is: "Extend LowNodeUtilization with getting nodes and pod usage from kubernetes metrics." Below the description is a code block showing the policy configuration for the Descheduler. The configuration includes a metricsCollector, profiles for LowNodeUtilization, and various thresholds and target thresholds for CPU, memory, and pods. The pull request also shows reviewers (jklaw90, atirtree, seanmalloy) and an assignee (jklaw90). The pull request is labeled as approved, cncf-cla:yes, lgtn, and sizeXXX.

```
apiVersion: "descheduler/v1alpha2"
kind: "DeschedulerPolicy"
metricsCollector:
  enabled: true
profiles:
  - name: ProfileName
    pluginConfig:
      - name: "LowNodeUtilization"
        args:
          thresholds:
            "cpu": 20
            "memory": 20
            "pods": 20
          targetThresholds:
            "cpu": 50
            "memory": 50
            "pods": 50
          metricsUtilization:
            metricsServer: true
        plugins:
          balance:
            enabled:
              - "LowNodeUtilization"
```

- Now (nov '24) also **load-aware**, before only based on reservation
- Only about **descheduling**: it will not influence the scheduling of the migration target pod
- Likely the smartest option on large clusters, overkilling on small environments (currently it's only an **optional** component)?

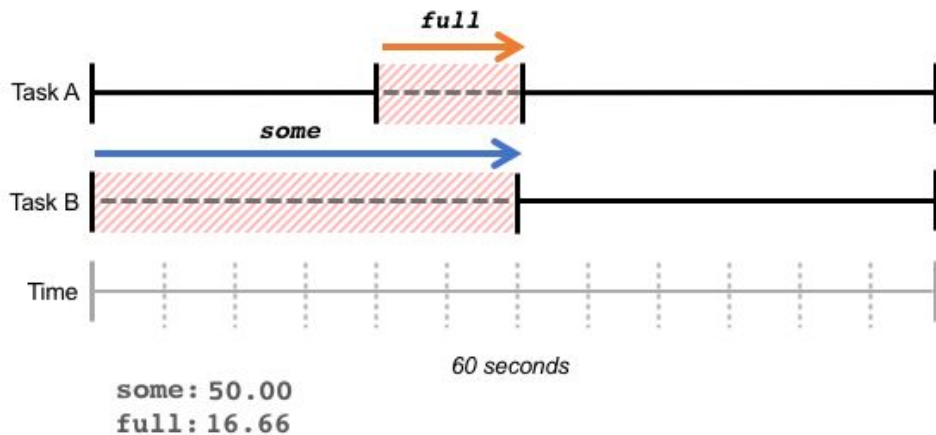




Load aware descheduling: One more thing...

Consume Pressure Stall Information (PSI) metrics

- Novel (\geq Kernel 4.20) canonical pressure metrics for three major resources: memory, CPU, and IO.
- Reported at node and cgroup slice level
- It's not measuring usage but the actual productivity losses caused by resource scarcity



cAdvisor integration is still in progress...

The screenshot shows a GitHub pull request page for 'Add Pressure Stall Information Metrics #3649'. The pull request is open and has 7 commits. The description of the pull request is as follows:

issues: [#3052](#), [#3083](#), [kubernetes/enhancements#4205](#)

This change adds metrics for pressure stall information, that indicate why some or all tasks of a cgroup2 have waited due to resource congestion (cpu, memory, io). The change exposes this information by including the *PSIStats* of each controller in it's stats, i.e. *CPUStats.PSI*, *MemoryStats.PSI* and *DiskStats.PSI*.

The information is additionally exposed as Prometheus metrics. The metrics follow the naming outlined by the prometheus/node-exporter, where stalled eq full and waiting eq some.

```
container_pressure_cpu_stalled_seconds_total
container_pressure_cpu_waiting_seconds_total
container_pressure_memory_stalled_seconds_total
container_pressure_memory_waiting_seconds_total
container_pressure_io_stalled_seconds_total
container_pressure_io_waiting_seconds_total
```

Reviewers: [rexagod](#), [SuperQ](#)

Assignees: No one assigned

Labels: None yet

Projects: None yet

**Questions?
Comments?**

**As users,
make your voice heard!!!**