KubeCon CloudNativeCon

North America 2024

Does My K8s Application Need CPR?

Performance Evaluation of a Multi-Cluster Workload Management Application

Braulio Dumba & Ezra Silvera, IBM Research

Agenda

- Motivation
- KubeStellar: Manage workloads across multi-clusters
- Testing framework & Experiments examples
- Lessons learned & Conclusion

Motivation

- Using multiple k8s clusters becomes common
  - Isolation: environments (dev, prod) , teams, etc..
  - Compliance with enterprise security or data governance requirements
  - Access to heterogeneous resources (GPUs, special HW, etc.)

- Multiple solutions already exist (e.g., KubeStellar, Karmada, OCM, etc.)

- Performance is a key factor for **managing workloads** across clusters
  - Users expect timely application deployment & status updates
  - Critical for adoption & success

Performance Evaluation Challenges

- Multi-cluster workload management applications are complex
  - Solutions include many components in different locations
  - Require multiple infrastructure configurations
- Most of the popular performance evaluation tools target single cluster
  - Might need to extend/enhance existing tools
- Scale-testing: need to provision many **clusters** → resource consuming
  - Emulation may not be enough for all test cases
- Such solutions offer a large number of configurable settings
  - Significantly increases the number of test options

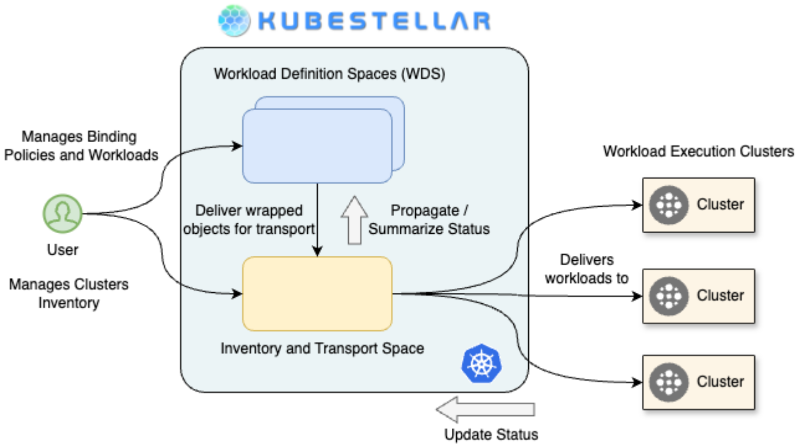
KubeStellar: Background

- A CNCF Sandbox project
- Deploy, configure and manage workloads across multiple K8s clusters
  - Use “native” K8s interfaces
  - Policy-based placement, workload customization, status summarization
- Find more info: https://kubestellar.io/

KUBESTELLAR

KubeStellar: Architecture Overview

- Based on "spaces" abstraction for isolation and multi-tenancy
  - Space: behaves like a regular k8s cluster. Exposes k8s API end-point
- Pluggable transport framework
  - Propagate resources from WDS to the execution cluster(s)
  - Currently using OCM (https://open-cluster-management.io/)
- Customize resources deployed to WEC
- Collect & summarize statuses from WECs


KUBESTELLAR
Workload Definition Spaces (WDS)
Manages Binding Policies and Workloads
User
Manages Clusters Inventory
Deliver wrapped objects for transport
Propagate / Summarize Status
Inventory and Transport Space
Workload Execution Clusters
Cluster
Delivers workloads to
Cluster
Cluster
Update Status

WEC: Workload Execution cluster
WDS: Workload Distribution Space,
ITS: Inventory and Transport Space

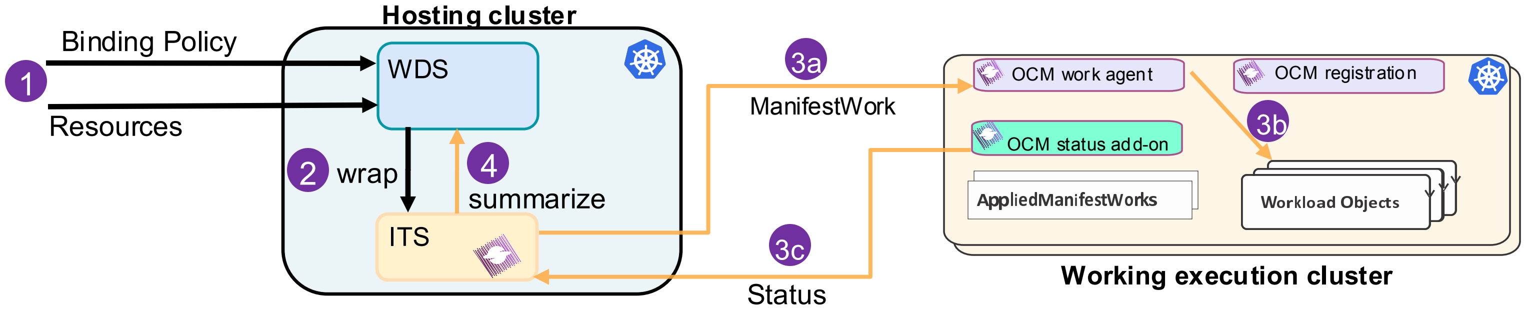
KubeStellar: Binding Policy (BP)

- Defines “what goes where"
- Associates a subset of workload objects in the WDS with a subset of WECs
  - Defined by the user in the WDS
  - Performance implications
- Allows customization of how workload objects are down-synced

```
apiVersion:
control.kubestellar.io/v1alpha1
kind: BindingPolicy
metadata:
  name: nginx
spec:
  clusterSelectors:
  - matchLabels:
      location-group: edge
  downsync:
  - objectSelectors:
    - matchLabels:
        app.kubernetes.io/name: nginx
…....
```

KubeStellar Example: Deploy Workloads

1. User:
    - Deploys workload/resources into WDS
    - Defines the binding policy (desired placement)
2. KS: Transport controller pushes resources into the inventory space (ITS)
    a. Wraps resources into a container object (e.g., ManifestWork)
3. Transport mechanism (OCM)
    a. Distributes wrapped resources into the WEC
    b. Unwraps & deploys resources in the WEC
    c. Status is returned through the KS transport status plugin
4. KS: Propagates status into WDS (user facing objects)

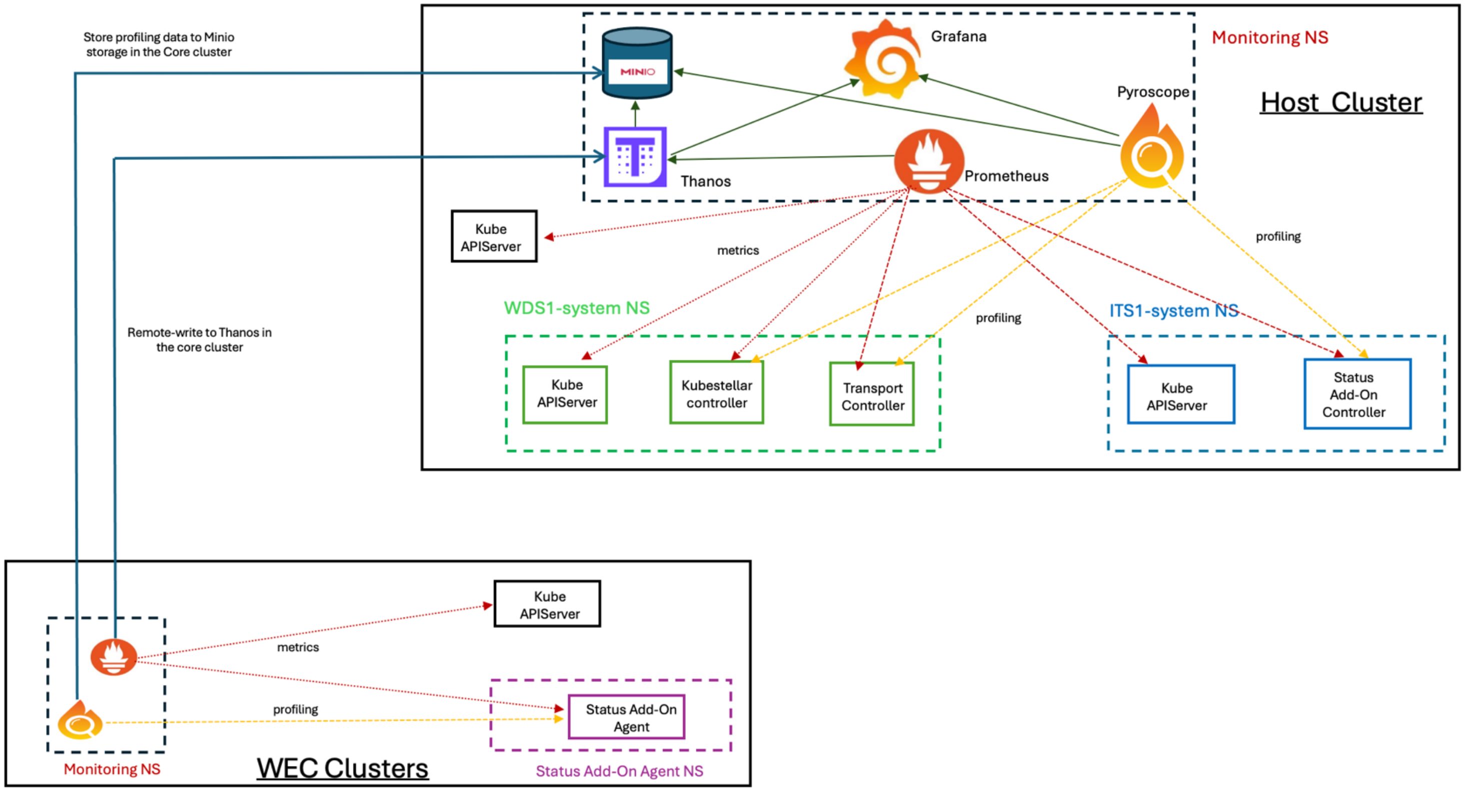


Open-Source Tools

Leverage existing tools when possible

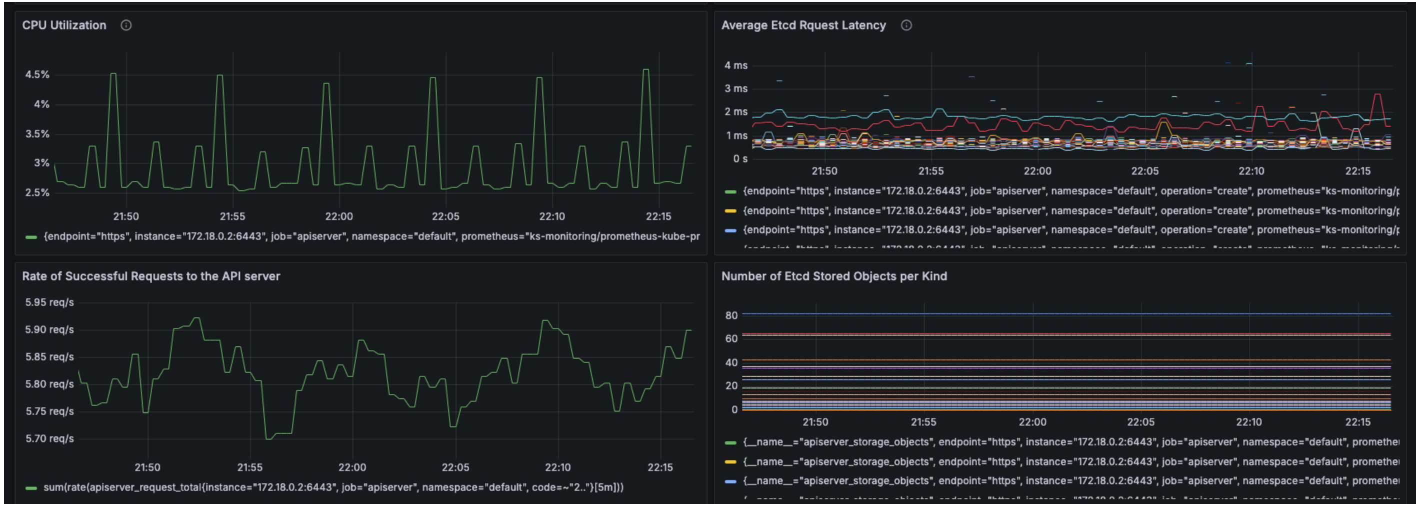
- **ClusterLoader2:** Kubernetes cluster performance test tool
- **Prometheus/Thanos, Pyroscope & Grafana:** Monitoring, profiling, visualization
- **Kind**: Running Kubernetes clusters using Docker container "nodes"
- **Kwok**: Kubernetes WithOut Kubelet, simulating any number of nodes and maintain pods on those nodes
- **Kube-burner**: Performance and scale test orchestration toolset. Mimic production workloads used to stress Kubernetes clusters

Framework Architecture: Monitoring Setup



Framework Architecture: Metrics Collection

- Targets: KS control plane (e.g., WDS & ITS API servers, transport, etc.)
- Single pane of glass to monitor KS hub and remote clusters



{endpoint="https", instance="172.18.0.2:6443", job="apiserver", namespace="default", prometheus="ks-monitoring/prometheus-kube-pr

{endpoint="https", instance="172.18.0.2:6443", job="apiserver", namespace="default", operation="create", prometheus="ks-monitoring/p

{endpoint="https", instance="172.18.0.2:6443", job="apiserver", namespace="default", operation="create", prometheus="ks-monitoring/p

{endpoint="https", instance="172.18.0.2:6443", job="apiserver", namespace="default", operation="create", prometheus="ks-monitoring/p

sum(rate(apiserver_request_total{instance="172.18.0.2:6443", job="apiserver", namespace="default", code=~"2.."}[5m]))

{__name__="apiserver_storage_objects", endpoint="https", instance="172.18.0.2:6443", job="apiserver", namespace="default", promethe

{__name__="apiserver_storage_objects", endpoint="https", instance="172.18.0.2:6443", job="apiserver", namespace="default", promethe

{__name__="apiserver_storage_objects", endpoint="https", instance="172.18.0.2:6443", job="apiserver", namespace="default", promethe

Grafana dashboards to monitor API server, APF and KS controllers

Workload Profile

Used two profiles from Kube-burner reference workloads

Workload benchmark for plain Kubernetes environments

Cluster-density profile
1 deployments, with two pod replicas (pause), mounting 2 secrets, 2 config maps
3 services, the first service points to the TCP/8080 port of the deployments
10 secrets containing a 2048-character random string
10 configmaps

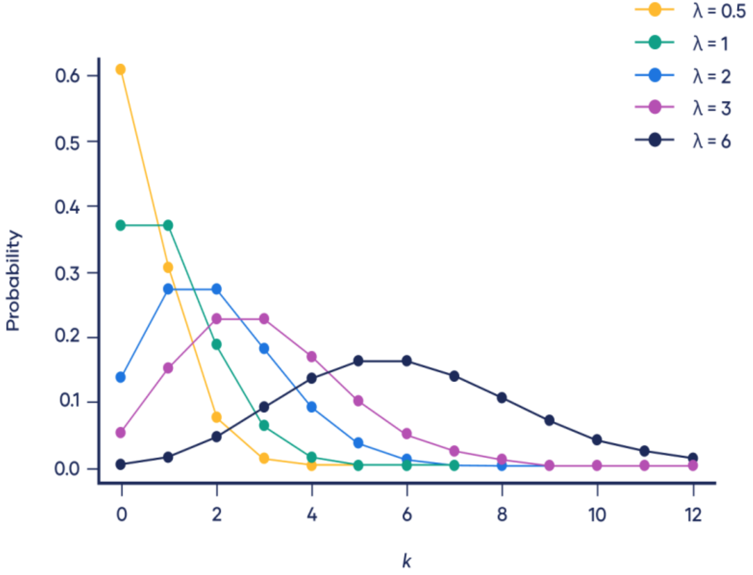
Workload benchmark for OpenShift environments

Cluster-density-ms profile
1 image stream
4 deployments, each with two pod replicas (pause), mounting 4 secrets, 4 config maps, and 1 downward API volume each
2 services, each pointing to the TCP/8080 and TCP/8443 ports of the first and second deployments
1 edge route pointing to the first service
20 secrets containing a 2048-character random string
10 configmaps containing a 2048-character random string

Workload Generation

- Poisson Workload generator function:
  - Introduce some randomness in the workload

- Extended ClusterLoader2 (CL2)
  - Support poisson distribution tuning set*
  - Add KubeStellar provider**



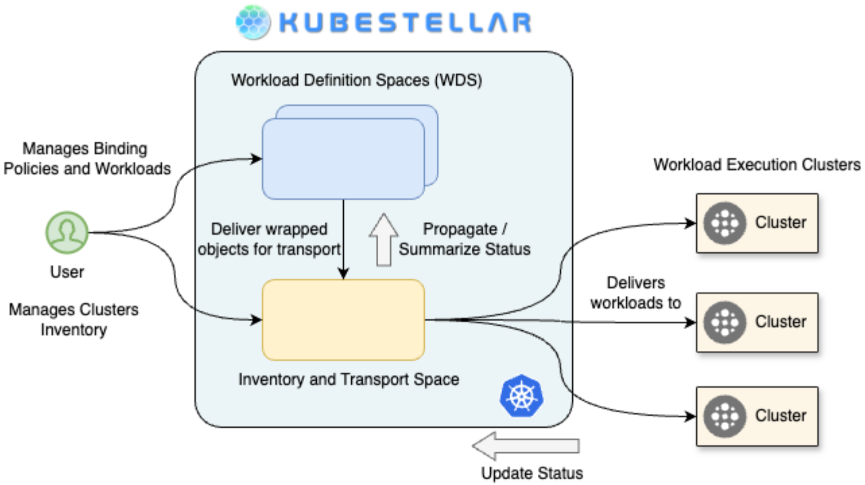
* Merged PR: https://github.com/kubernetes/perf-tests/pull/2633
** Merged PR: https://github.com/kubernetes/perf-tests/pull/2632

[1] reference: https://www.scribbr.com/statistics/poisson-distribution/#:~:text=A%20Poisson%20distribution%20is%20a,the%20mean%20number%20of%20events.

Performance Variables

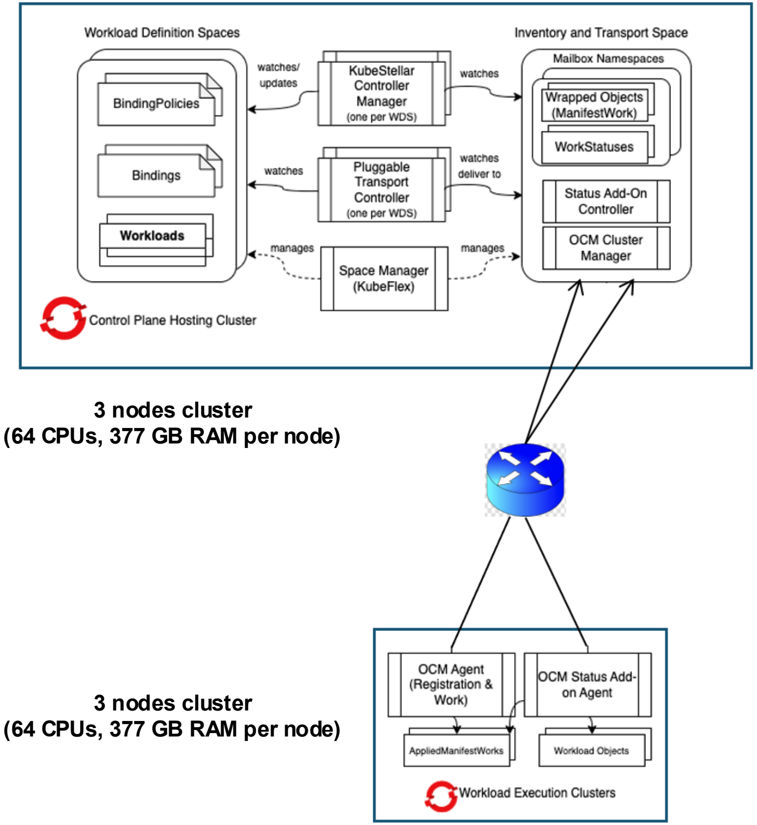
Selected KubeStellar performance parameters:

- **Number of binding Policies**
- **Number of workload execution clusters (WEC)**
- Number of workload description space (with shared ITS)
- Number of workload description space (with dedicated ITS)
- Workload size per binding policy


KUBESTELLAR
Workload Definition Spaces (WDS)
Manages Binding Policies and Workloads
User
Manages Clusters Inventory
Deliver wrapped objects for transport
Propagate / Summarize Status
Inventory and Transport Space
Workload Execution Clusters
Delivers workloads to
Cluster
Cluster
Cluster
Update Status

Experiment 1: Number of Binding Policies

- Setup:
  - 2 OCP clusters
- Workload generator function
  - clusterloader2 **Poisson** tuningSet (alpha=0.5)


Workload Definition Spaces
Inventory and Transport Space
BindingPolicies
Bindings
Workloads
watches/ updates
KubeStellar Controller Manager (one per WDS)
watches
Mailbox Namespaces
Wrapped Objects (ManifestWork)
WorkStatuses
watches
Pluggable Transport Controller (one per WDS)
watches deliver to
Status Add-On Controller
manages
Space Manager (KubeFlex)
manages
OCM Cluster Manager
Control Plane Hosting Cluster
OCM Agent (Registration & Work)
OCM Status Add-on Agent
AppliedManifestWorks
Workload Objects
Workload Execution Clusters

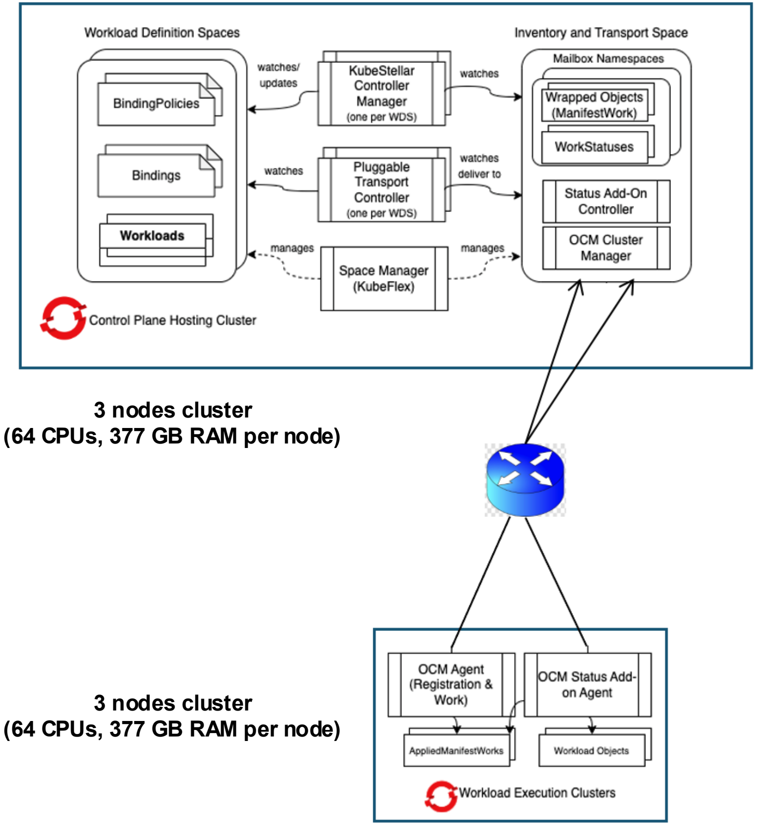
3 nodes cluster
(64 CPUs, 377 GB RAM per node)

3 nodes cluster
(64 CPUs, 377 GB RAM per node)

Experiment 1: Number of Binding Policies

- Setup:
  - 2 OCP clusters
- Workload:
  - cluster-density-ms

Object type	Total #
BindingPolicies	150
Namespaces	150
Image stream	150
Deployments	600
Secrets	3000
Services	300
Configmaps	1500
routes	150


Workload Definition Spaces
BindingPolicies
Bindings
Workloads
watches/
updates
KubeStellar Controller Manager (one per WDS)
watches
Pluggable Transport Controller (one per WDS)
manages
Space Manager (KubeFlex)
manages
watches
watches
deliver to
Inventory and Transport Space
Mailbox Namespaces
Wrapped Objects (ManifestWork)
WorkStatuses
Status Add-On Controller
OCM Cluster Manager
Control Plane Hosting Cluster
OCM Agent (Registration & Work)
OCM Status Add-on Agent
AppliedManifestWorks
Workload Objects
Workload Execution Clusters

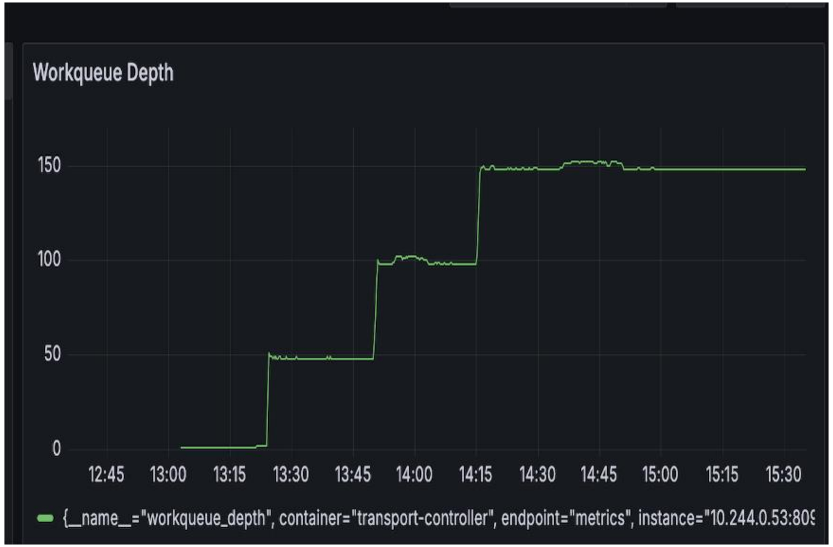
3 nodes cluster
(64 CPUs, 377 GB RAM per node)

3 nodes cluster
(64 CPUs, 377 GB RAM per node)

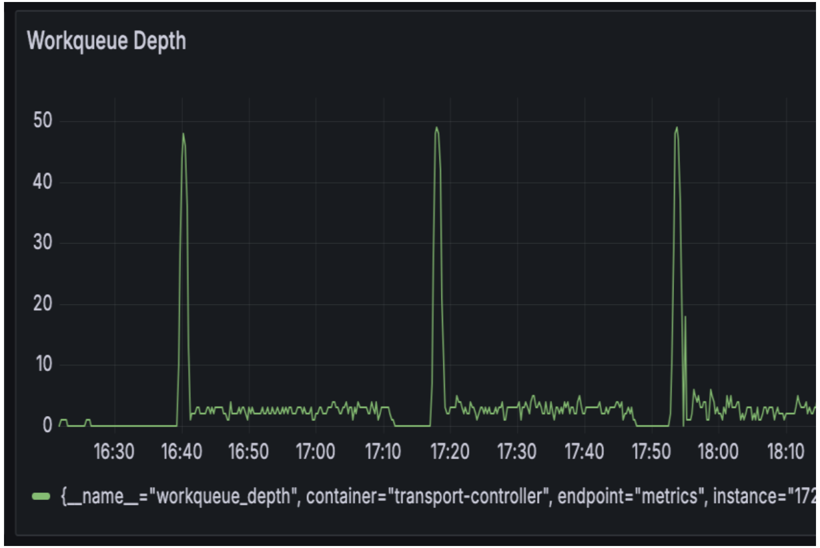
Experiment 1: Number of Binding Policies - Results

Issue detected: controllers fight (component: KS transport controllers)

Before fixes



After fixes



Experiment 2: Long Running Test

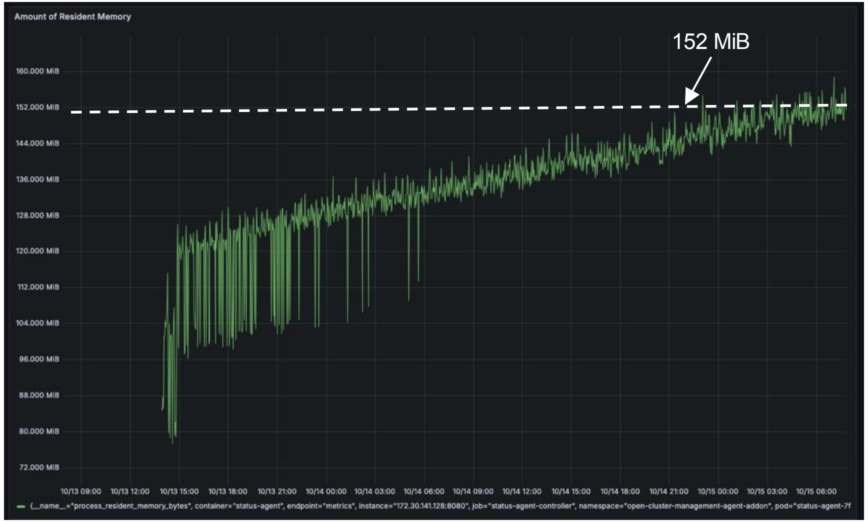
- Setup:
  - 2 OCP clusters

- Long running measurements: 1 binding Policy
  - Workload: pod that sleeps for 20 seconds
  - Workload generator function: clusterloader **RSteppedLoad** tuningSet (burstSize=1 and stepDelay=60 sec)
  - Custom controller: deletes a pod after reaching the completed state – another pod is created after 1 minute with a different name
  - Experiment duration (48 hours): 10/13 - 10/15

Experiment 2: Long Running Test - Results

Issue detected: memory leak (component: status-agent controller)

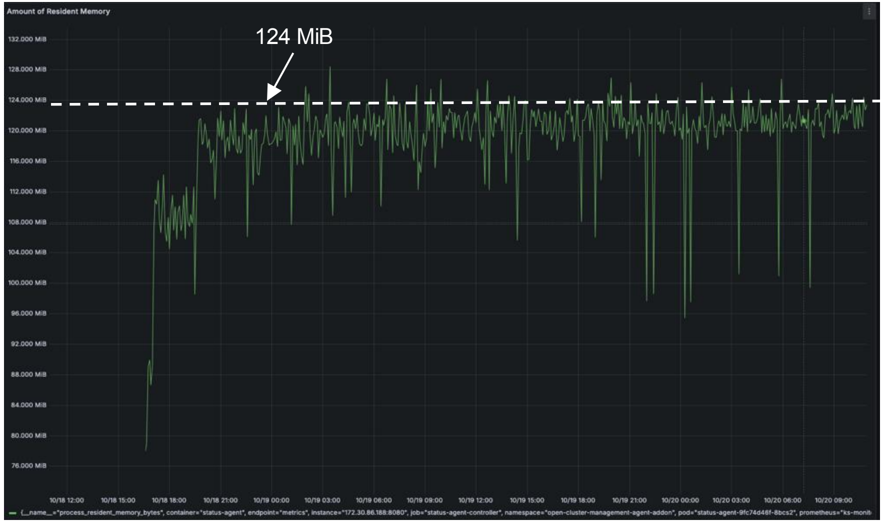
Before fixes



Amount of Resident Memory

152 MiB

After fixes

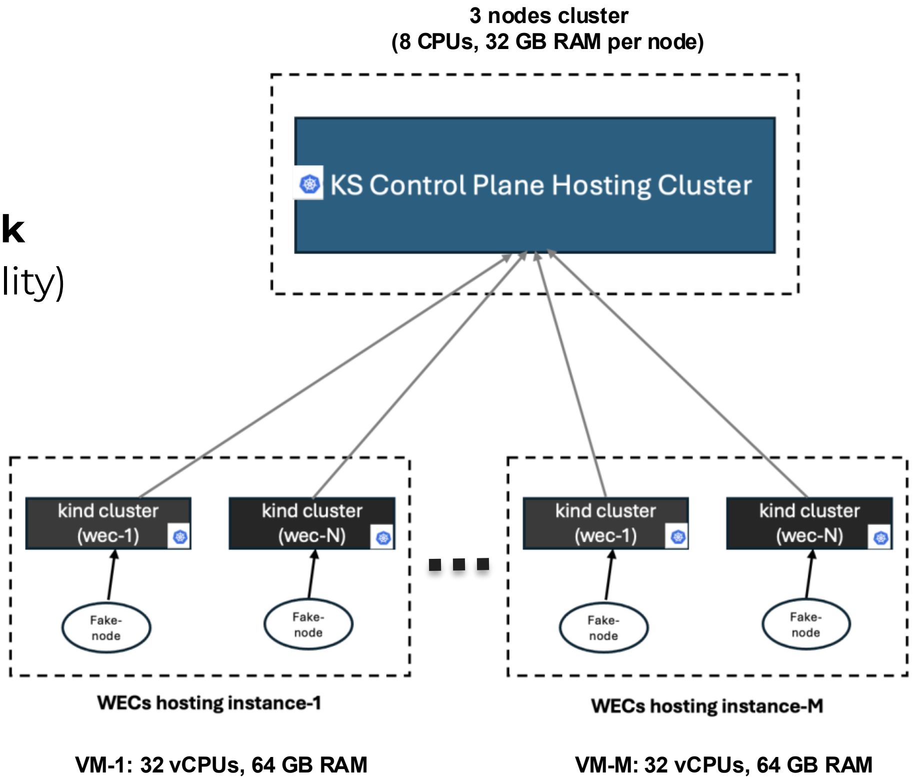


Amount of Resident Memory

124 MiB

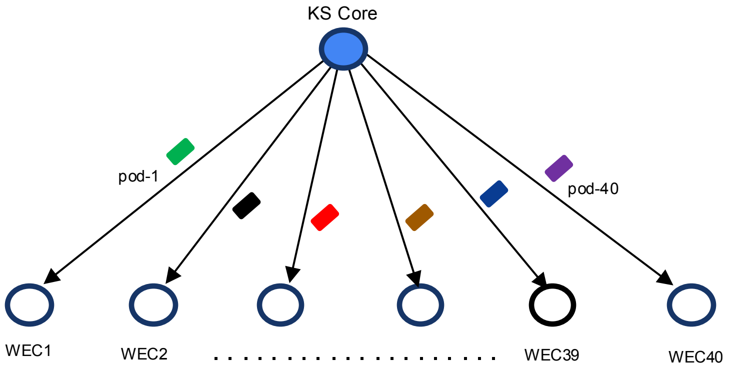
Experiment 3: Number of WECs

- Setup:
    - Ubuntu 24.04 VMs (M = 5)
    - **100 kind clusters** (N = 20)
    - Emulation of nodes & pods with **kwok**
    - Using Ansible automation (repeatability)

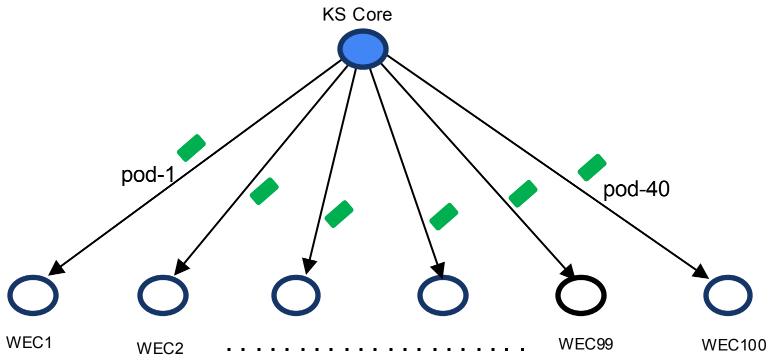


Experiment 3: Number of WECs (Cont.)

- Measurements: E2E & down-sync latencies, resource utilization, etc.
- Workload: kwok fake pod



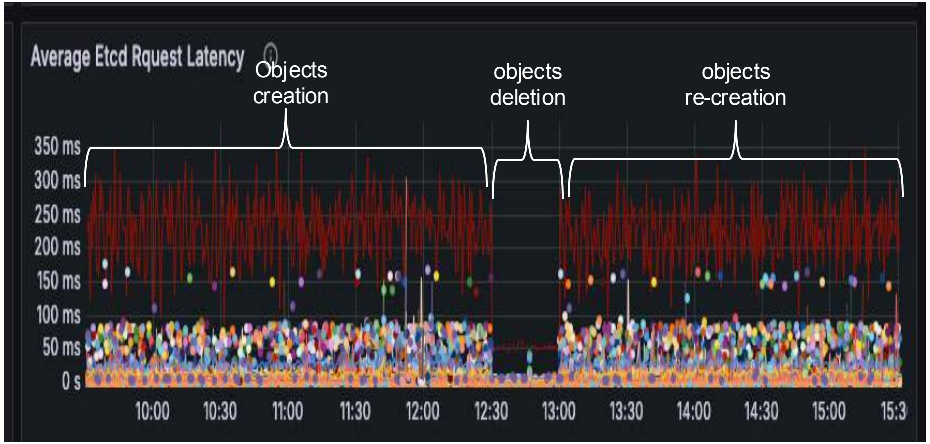
1-1 Deployment
(1 binding Policy per cluster)

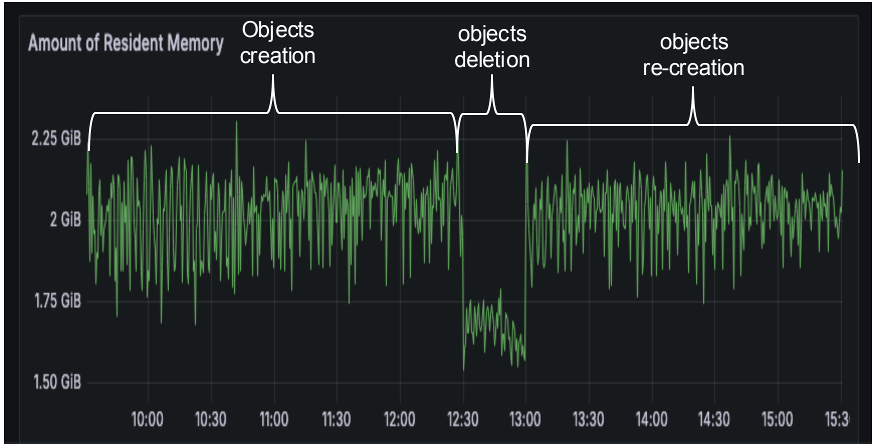


1-Many Deployment
(1 binding Policy for all clusters)

Experiment 3: Number of WECs - Results

Resource Utilization: ITS API Server

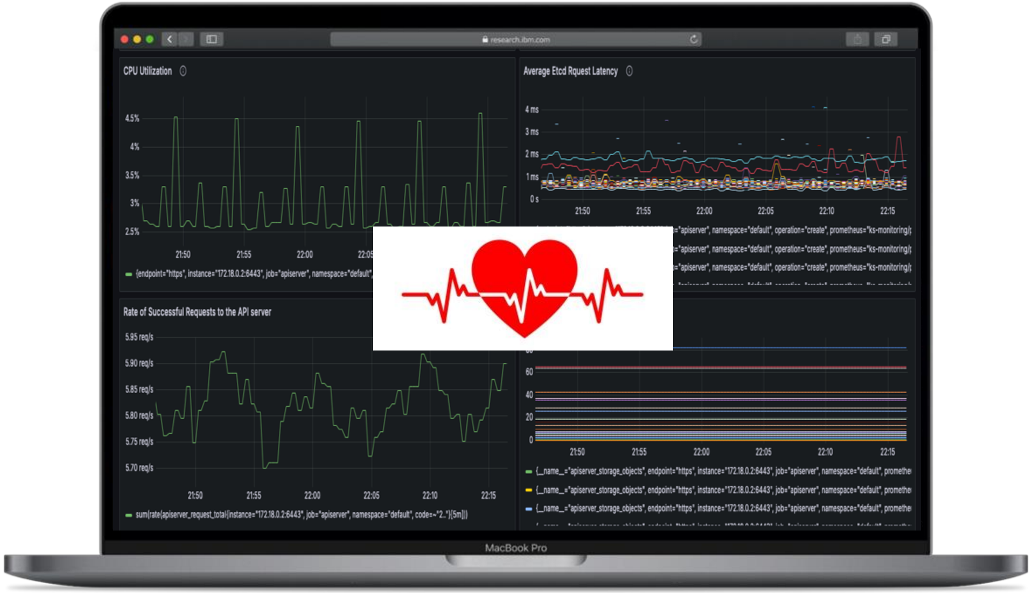




Lessons Learned & Conclusion

- A single pane of glass to analyze performance is extremely helpful
- Define performance variable & stay focused on them
- Long tests are a must – some issues can't be detected without it
- Profiling is your friend
- Leverage existing open-source tools when possible
- Performance analysis should be used to help building usage guidelines/benchmark

So, Does My k8s Application Need CPR?


research.ibm.com
CPU Utilization
4.5%
4%
3.5%
3%
2.5%
21:50
21:55
22:00
22:05
{endpoint="https", instance="172.18.0.2:6443", job="apiserver", namespace="default",
Average Etcd Rquest Latency
4 ms
3 ms
2 ms
1 ms
0 s
21:50
21:55
22:00
22:05
22:10
22:15
="apiserver", namespace="default", operation="create", prometheus="ks-monitoring/
="apiserver", namespace="default", operation="create", prometheus="ks-monitoring/
="apiserver", namespace="default", operation="create", prometheus="ks-monitoring/
Rate of Successful Requests to the API server
5.95 req/s
5.90 req/s
5.85 req/s
5.80 req/s
5.75 req/s
5.70 req/s
21:50
21:55
22:00
22:05
22:10
22:15
sum(rate(apiserver_request_total{instance="172.18.0.2:6443", job="apiserver", namespace="default", code=~"2.."}[5m]))
60
40
20
0
21:50
21:55
22:00
22:05
22:10
22:15
{__name__="apiserver_storage_objects", endpoint="https", instance="172.18.0.2:6443", job="apiserver", namespace="default", promethe
{__name__="apiserver_storage_objects", endpoint="https", instance="172.18.0.2:6443", job="apiserver", namespace="default", promethe
{__name__="apiserver_storage_objects", endpoint="https", instance="172.18.0.2:6443", job="apiserver", namespace="default", promethe
MacBook Pro

No! I can see and hear the **heartbeats** even for my **Multi-Cluster** management app