OpenShift Container Platform 4.4

Scalability and performance

Scaling your OpenShift Container Platform cluster and tuning performance in production environments
Scaling your OpenShift Container Platform cluster and tuning performance in production environments
Abstract

This document provides instructions for scaling your cluster and optimizing the performance of your OpenShift Container Platform environment.
9.4. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO APPLICATION REQUIREMENTS  

CHAPTER 10. OPTIMIZING STORAGE  
10.1. AVAILABLE PERSISTENT STORAGE OPTIONS  
10.2. RECOMMENDED CONFIGURABLE STORAGE TECHNOLOGY  
10.2.1. Specific application storage recommendations  
10.2.1.1. Registry  
10.2.1.2. Scaled registry  
10.2.1.3. Metrics  
10.2.1.4. Logging  
10.2.1.5. Applications  
10.2.2. Other specific application storage recommendations  

CHAPTER 11. OPTIMIZING ROUTING  
11.1. BASELINE ROUTER PERFORMANCE  
11.2. ROUTER PERFORMANCE OPTIMIZATIONS  

CHAPTER 12. WHAT HUGE PAGES DO AND HOW THEY ARE CONSUMED BY APPLICATIONS  
12.1. WHAT HUGE PAGES DO  
12.2. HOW HUGE PAGES ARE CONSUMED BY APPS  
12.3. CONFIGURING HUGE PAGES
CHAPTER 1. RECOMMENDED PRACTICES FOR INSTALLING LARGE CLUSTERS

Apply the following practices when installing large clusters or scaling clusters to larger node counts.

1.1. RECOMMENDED PRACTICES FOR INSTALLING LARGE SCALE CLUSTERS

When installing large clusters or scaling the cluster to larger node counts, set the cluster network cidr accordingly in your `install-config.yaml` file before you install the cluster:

```yaml
networking:
  clusterNetwork:
    - cidr: 10.128.0.0/14
      hostPrefix: 23
    machineCIDR: 10.0.0.0/16
    networkType: OpenShiftSDN
  serviceNetwork:
    - 172.30.0.0/16
```

The default clusterNetwork cidr 10.128.0.0/14 cannot be used if the cluster size is more than 500 nodes. It must be set to 10.128.0.0/12 or 10.128.0.0/10 to get to larger node counts beyond 500 nodes.
CHAPTER 2. RECOMMENDED HOST PRACTICES

This topic provides recommended host practices for OpenShift Container Platform.

2.1. RECOMMENDED NODE HOST PRACTICES

The OpenShift Container Platform node configuration file contains important options. For example, two parameters control the maximum number of pods that can be scheduled to a node: `podsPerCore` and `maxPods`.

When both options are in use, the lower of the two values limits the number of pods on a node. Exceeding these values can result in:

- Increased CPU utilization.
- Slow pod scheduling.
- Potential out-of-memory scenarios, depending on the amount of memory in the node.
- Exhausing the pool of IP addresses.
- Resource overcommitting, leading to poor user application performance.

**IMPORTANT**

In Kubernetes, a pod that is holding a single container actually uses two containers. The second container is used to set up networking prior to the actual container starting. Therefore, a system running 10 pods will actually have 20 containers running.

`podsPerCore` sets the number of pods the node can run based on the number of processor cores on the node. For example, if `podsPerCore` is set to 10 on a node with 4 processor cores, the maximum number of pods allowed on the node will be 40.

```
kubeletConfig:
podsPerCore: 10
```

Setting `podsPerCore` to 0 disables this limit. The default is 0. `podsPerCore` cannot exceed `maxPods`.

`maxPods` sets the number of pods the node can run to a fixed value, regardless of the properties of the node.

```
kubeletConfig:
maxPods: 250
```

2.2. CREATE A KUBELETCONFIG CRD TO EDIT KUBELET PARAMETERS

The kubelet configuration is currently serialized as an ignition configuration, so it can be directly edited. However, there is also a new kubelet-config-controller added to the Machine Config Controller (MCC). This allows you to create a KubeletConfig custom resource (CR) to edit the kubelet parameters.

**Procedure**
1. Run:

   ```
   $ oc get machineconfig
   ```

   This provides a list of the available machine configuration objects you can select. By default, the two kubelet-related configs are `01-master-kubelet` and `01-worker-kubelet`.

2. To check the current value of max Pods per node, run:

   ```
   # oc describe node <node-ip> | grep Allocatable -A6
   ```

   Look for **value: pods: <value>**.

   For example:

   ```
   # oc describe node ip-172-31-128-158.us-east-2.compute.internal | grep Allocatable -A6
   Allocatable:
   attachable-volumes-aws-ebs:  25
   cpu:                         3500m
   hugepages-1Gi:               0
   hugepages-2Mi:               0
   memory:                      15341844Ki
   pods:                        250
   ```

3. To set the max Pods per node on the worker nodes, create a custom resource file that contains the kubelet configuration. For example, `change-maxPods-cr.yaml`:

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: KubeletConfig
   metadata:
     name: set-max-pods
   spec:
     machineConfigPoolSelector:
       matchLabels:
         custom-kubelet: large-pods
     kubeletConfig:
       maxPods: 500
   ```

   The rate at which the kubelet talks to the API server depends on queries per second (QPS) and burst values. The default values, **50** for `kubeAPIQPS` and **100** for `kubeAPIBurst`, are good enough if there are limited pods running on each node. Updating the kubelet QPS and burst rates is recommended if there are enough CPU and memory resources on the node:

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: KubeletConfig
   metadata:
     name: set-max-pods
   spec:
     machineConfigPoolSelector:
       matchLabels:
         custom-kubelet: large-pods
     kubeletConfig:
       maxPods: <pod_count>
       kubeAPIBurst: <burst_rate>
       kubeAPIQPS: <QPS>
   ```
a. Run:

```bash
$ oc label machineconfigpool worker custom-kubelet=large-pods
```

b. Run:

```bash
$ oc create -f change-maxPods-cr.yaml
```

c. Run:

```bash
$ oc get kubeletconfig
```

This should return `set-max-pods`.

Depending on the number of worker nodes in the cluster, wait for the worker nodes to be rebooted one by one. For a cluster with 3 worker nodes, this could take about 10 to 15 minutes.

4. Check for `maxPods` changing for the worker nodes:

```bash
$ oc describe node
```

a. Verify the change by running:

```bash
$ oc get kubeletconfigs set-max-pods -o yaml
```

This should show a status of `True` and `type:Success`

**Procedure**

By default, only one machine is allowed to be unavailable when applying the kubelet-related configuration to the available worker nodes. For a large cluster, it can take a long time for the configuration change to be reflected. At any time, you can adjust the number of machines that are updating to speed up the process.

1. Run:

```bash
$ oc edit machineconfigpool worker
```

2. Set `maxUnavailable` to the desired value.

```yaml
spec:
  maxUnavailable: <node_count>
```

**IMPORTANT**

When setting the value, consider the number of worker nodes that can be unavailable without affecting the applications running on the cluster.

**2.3. MASTER NODE SIZING**
The master node resource requirements depend on the number of nodes in the cluster. The following master node size recommendations are based on the results of control plane density focused testing.

<table>
<thead>
<tr>
<th>Number of worker nodes</th>
<th>CPU cores</th>
<th>Memory (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>100</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td>250</td>
<td>16</td>
<td>64</td>
</tr>
</tbody>
</table>

**IMPORTANT**

Because you cannot modify the master node size in a running OpenShift Container Platform 4.4 cluster, you must estimate your total node count and use the suggested master size during installation.

**NOTE**

In OpenShift Container Platform 4.4, half of a CPU core (500 millicore) is now reserved by the system by default compared to OpenShift Container Platform 3.11 and previous versions. The sizes are determined taking that into consideration.

### 2.4. RECOMMENDED ETCD PRACTICES

For large and dense clusters, etcd can suffer from poor performance if the keyspace grows excessively large and exceeds the space quota. Periodic maintenance of etcd including defragmentation needs to be done to free up space in the data store. It is highly recommended that you monitor Prometheus for etcd metrics and defragment it when needed before etcd raises a cluster-wide alarm that puts the cluster into a maintenance mode, which only accepts key reads and deletes. Some of the key metrics to monitor are `etcd_server_quota_backend_bytes` which is the current quota limit, `etcd_mvcc_db_total_size_in_use_in_bytes` which indicates the actual database usage after a history compaction, and `etcd_debugging_mvcc_db_total_size_in_bytes` which shows the database size including free space waiting for defragmentation.

### 2.5. ADDITIONAL RESOURCES

- OpenShift Container Platform cluster maximums
CHAPTER 3. RECOMMENDED CLUSTER SCALING PRACTICES

Apply the following best practices to scale the number of worker machines in your OpenShift Container Platform cluster. You scale the worker machines by increasing or decreasing the number of replicas that are defined in the worker MachineSet.

3.1. RECOMMENDED PRACTICES FOR SCALING THE CLUSTER

When scaling up the cluster to higher node counts:

- Spread nodes across all of the available zones for higher availability.
- Scale up by no more than 25 to 50 machines at once.
- Consider creating new MachineSets in each available zone with alternative instance types of similar size to help mitigate any periodic provider capacity constraints. For example, on AWS, use m5.large and m5d.large.

**NOTE**

Cloud providers might implement a quota for API services. Therefore, gradually scale the cluster.

The controller might not be able to create the machines if the replicas in the MachineSets are set to higher numbers all at one time. The number of requests the cloud platform, which OpenShift Container Platform is deployed on top of, is able to handle impacts the process. The controller will start to query more while trying to create, check, and update the machines with the status. The cloud platform on which OpenShift Container Platform is deployed has API request limits and excessive queries might lead to machine creation failures due to cloud platform limitations.

Enable machine health checks when scaling to large node counts. In case of failures, the health checks monitor the condition and automatically repair unhealthy machines.

3.2. MODIFYING A MACHINESET

To make changes to a MachineSet, edit the MachineSet YAML. Then, remove all machines associated with the MachineSet by deleting each machine or scaling down the MachineSet to 0 replicas. Then, scale the replicas back to the desired number. Changes you make to a MachineSet do not affect existing machines.

If you need to scale a MachineSet without making other changes, you do not need to delete the machines.

**NOTE**

By default, the OpenShift Container Platform router pods are deployed on workers. Because the router is required to access some cluster resources, including the web console, do not scale the worker MachineSet to 0 unless you first relocate the router pods.

Prerequisites

- Install an OpenShift Container Platform cluster and the oc command line.
- Log in to oc as a user with cluster-admin permission.

**Procedure**

1. Edit the MachineSet:
   
   ```
   $ oc edit machineset <machineset> -n openshift-machine-api
   ```

2. Scale down the MachineSet to 0:
   
   ```
   $ oc scale --replicas=0 machineset <machineset> -n openshift-machine-api
   ```
   Or:
   
   ```
   $ oc edit machineset <machineset> -n openshift-machine-api
   ```
   Wait for the machines to be removed.

3. Scale up the MachineSet as needed:
   
   ```
   $ oc scale --replicas=2 machineset <machineset> -n openshift-machine-api
   ```
   Or:
   
   ```
   $ oc edit machineset <machineset> -n openshift-machine-api
   ```
   Wait for the machines to start. The new Machines contain changes you made to the Machineset.

### 3.3. ABOUT MACHINEHEALTHCHECKS


To monitor machine health, you create a resource to define the configuration for a controller. You set a condition to check for, such as staying in the NotReady status for 15 minutes or displaying a permanent condition in the node-problem-detector, and a label for the set of machines to monitor.

**NOTE**

You cannot apply a MachineHealthCheck to a machine with the master role.

The controller that observes a MachineHealthCheck resource checks for the status that you defined. If a machine fails the health check, it is automatically deleted and a new one is created to take its place. When a machine is deleted, you see a machine deleted event. To limit disruptive impact of the machine deletion, the controller drains and deletes only one node at a time. If there are more unhealthy machines than the maxUnhealthy threshold allows for in the targeted pool of machines, remediation stops so that manual intervention can take place.

To stop the check, you remove the resource.

### 3.4. SAMPLE MACHINEHEALTHCHECK RESOURCE

The MachineHealthCheck resource resembles the following YAML file:
MachineHealthCheck

apiVersion: machine.openshift.io/v1beta1
kind: MachineHealthCheck
metadata:
  name: example
  namespace: openshift-machine-api
spec:
  selector:
    matchLabels:
      machine.openshift.io/cluster-api-machine-role: <role>
      machine.openshift.io/cluster-api-machine-type: <role>
      machine.openshift.io/cluster-api-machineset: <cluster_name>-<label>-<zone>
  unhealthyConditions:
    - type: "Ready"
      timeout: "300s"
      status: "False"
    - type: "Ready"
      timeout: "300s"
      status: "Unknown"
  maxUnhealthy: "40%"

1. Specify the name of the MachineHealthCheck to deploy.
2, 3. Specify a label for the machine pool that you want to check.
4. Specify the MachineSet to track in `<cluster_name>-<label>-<zone>` format. For example, `prod-node-us-east-1a`.
5. Specify the amount of unhealthy machines allowed in the targeted pool of machines. This can be set as a percentage or an integer.

**NOTE**

The **matchLabels** are examples only; you must map your machine groups based on your specific needs.

### 3.5. CREATING A MACHINEHEALTHCHECK RESOURCE

You can create a MachineHealthCheck resource for all MachinePools in your cluster except the master pool.

**Prerequisites**

- Install the `oc` command line interface.

**Procedure**

1. Create a `healthcheck.yml` file that contains the definition of your MachineHealthCheck.

2. Apply the `healthcheck.yml` file to your cluster:

   $ oc apply -f healthcheck.yml
CHAPTER 4. USING THE NODE TUNING OPERATOR

Learn about the Node Tuning Operator and how you can use it to manage node-level tuning by orchestrating the tuned daemon.

4.1. ABOUT THE NODE TUNING OPERATOR

The Node Tuning Operator helps you manage node-level tuning by orchestrating the tuned daemon. The majority of high-performance applications require some level of kernel tuning. The Node Tuning Operator provides a unified management interface to users of node-level sysctls and more flexibility to add custom tuning specified by user needs. The Operator manages the containerized tuned daemon for OpenShift Container Platform as a Kubernetes DaemonSet. It ensures the custom tuning specification is passed to all containerized tuned daemons running in the cluster in the format that the daemons understand. The daemons run on all nodes in the cluster, one per node.

Node-level settings applied by the containerized tuned daemon are rolled back on an event that triggers a profile change or when the containerized tuned daemon is terminated gracefully by receiving and handling a termination signal.

The Node Tuning Operator is part of a standard OpenShift Container Platform installation in version 4.1 and later.

4.2. ACCESSING AN EXAMPLE NODE TUNING OPERATOR SPECIFICATION

Use this process to access an example Node Tuning Operator specification.

Procedure

1. Run:

   $ oc get Tuned/default -o yaml -n openshift-cluster-node-tuning-operator

The default CR is meant for delivering standard node-level tuning for the OpenShift Container Platform platform and any custom changes to the default CR will be overwritten by the Operator. For custom tuning, create your own tuned CRs. Newly created CRs will be combined with the default CR and custom tuning applied to OpenShift Container Platform nodes based on node or Pod labels and profile priorities.

While in certain situations the support for Pod labels can be a convenient way of automatically delivering required tuning, this practice is discouraged and strongly advised against, especially in large-scale clusters. The default tuned CR ships without Pod label matching. If a custom profile is created with Pod label matching, then the functionality will be enabled at that time. The Pod label functionality might be deprecated in future versions of the Node Tuning Operator.

4.3. DEFAULT PROFILES SET ON A CLUSTER

The following are the default profiles set on a cluster.

```yaml
apiVersion: tuned.openshift.io/v1
default
kind: Tuned
metadata:
  name: default
```
namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - name: "openshift"
    data:
      [main]
      summary=Optimize systems running OpenShift (parent profile)
      include=${f:virt_check:virtual-guest:throughput-performance}
      [selinux]
      avc_cache_threshold=8192
      [net]
      nf_contrack_hashsize=131072
      [sysctl]
      net.ipv4.ip_forward=1
      kernel.pid_max=>4194304
      net.netfilter.nf_contrack_max=1048576
      net.ipv4.conf.all.arp_announce=2
      net.ipv4.neigh.default.gc_thresh1=8192
      net.ipv4.neigh.default.gc_thresh2=32768
      net.ipv4.neigh.default.gc_thresh3=65536
      net.ipv6.neigh.default.gc_thresh1=8192
      net.ipv6.neigh.default.gc_thresh2=32768
      net.ipv6.neigh.default.gc_thresh3=65536
      vm.max_map_count=262144
      [sysfs]
      /sys/module/nvme_core/parameters/io_timeout=4294967295
      /sys/module/nvme_core/parameters/max_retries=10
  - name: "openshift-control-plane"
    data:
      [main]
      summary=Optimize systems running OpenShift control plane
      include=openshift
      [sysctl]
      # ktune sysctl settings, maximizing i/o throughput
      #
      # Minimal preemption granularity for CPU-bound tasks:
      # (default: 1 msec# (1 + ilog(ncpus)), units: nanoseconds)
      kernel.sched_min_granularity_ns=10000000
      # The total time the scheduler will consider a migrated process
      # "cache hot" and thus less likely to be re-migrated
      # (system default is 500000, i.e. 0.5 ms)
      kernel.sched_migration_cost_ns=5000000
      # SCHED_OTHER wake-up granularity.
      #
      # Preemption granularity when tasks wake up. Lower the value to
      # improve wake-up latency and throughput for latency critical tasks.
      kernel.sched_wakeup_granularity_ns=4000000
  - name: "openshift-node"
    data:

4.4. CUSTOM TUNING SPECIFICATION

The custom resource (CR) for the operator has two major sections. The first section, profile:, is a list of tuned profiles and their names. The second, recommend:, defines the profile selection logic.

Multiple custom tuning specifications can co-exist as multiple CRs in the operator’s namespace. The existence of new CRs or the deletion of old CRs is detected by the Operator. All existing custom tuning specifications are merged and appropriate objects for the containerized tuned daemons are updated.

Profile data

The profile: section lists tuned profiles and their names.

```
profile:
  - name: tuned_profile_1
    data: |
      # Tuned profile specification
      [main]
      summary=Description of tuned_profile_1 profile
      
      [sysctl]
      net.ipv4.ip_forward=1
      # ... other sysctl's or other tuned daemon plugins supported by the containerized tuned
      # ...

  - name: tuned_profile_n
    data: |
      # Tuned profile specification
      [main]
      summary=Description of tuned_profile_n profile

      # tuned_profile_n profile settings
```

Recommended profiles
The **profile** selection logic is defined by the **recommend** section of the CR:

```
recommend:
  - match: # optional; if omitted, profile match is assumed unless a profile with a higher matches first
    <match> # an optional array
    priority: <priority> # profile ordering priority, lower numbers mean higher priority (0 is the highest priority)
    profile: <tuned_profile_name> # e.g. tuned_profile_1
    # ...
  - match:
    <match>
    priority: <priority>
    profile: <tuned_profile_name> # e.g. tuned_profile_n
```

If `<match>` is omitted, a profile match (for example, `true`) is assumed.

`<match>` is an optional array recursively defined as follows:

```
- label: <label_name> # node or pod label name
  value: <label_value> # optional node or pod label value; if omitted, the presence of <label_name> is enough to match
  type: <label_type> # optional node or pod type ("node" or "pod"); if omitted, "node" is assumed
  <match> # an optional <match> array
```

If `<match>` is not omitted, all nested `<match>` sections must also evaluate to `true`. Otherwise, `false` is assumed and the profile with the respective `<match>` section will not be applied or recommended.

Therefore, the nesting (child `<match>` sections) works as logical AND operator. Conversely, if any item of the `<match>` array matches, the entire `<match>` array evaluates to `true`. Therefore, the array acts as logical OR operator.

**Example**

```
- match:
  - label: tuned.openshift.io/elasticsearch
    match:
      - label: node-role.kubernetes.io/master
        - label: node-role.kubernetes.io/infra
type: pod
  priority: 10
  profile: openshift-control-plane-es
- match:
  - label: node-role.kubernetes.io/master
    - label: node-role.kubernetes.io/infra
  priority: 20
  profile: openshift-control-plane
- priority: 30
  profile: openshift-node
```

The CR above is translated for the containerized tuned daemon into its `recommend.conf` file based on the profile priorities. The profile with the highest priority (10) is `openshift-control-plane-es` and, therefore, it is considered first. The containerized tuned daemon running on a given node looks to see if there is a pod running on the same node with the `tuned.openshift.io/elasticsearch` label set. If not, the
entire `<match>` section evaluates as `false`. If there is such a pod with the label, in order for the `<match>` section to evaluate to `true`, the node label also needs to be `node-role.kubernetes.io/master` or `node-role.kubernetes.io/infra`.

If the labels for the profile with priority `10` matched, `openshift-control-plane-es` profile is applied and no other profile is considered. If the node/pod label combination did not match, the second highest priority profile (`openshift-control-plane`) is considered. This profile is applied if the containerized tuned pod runs on a node with labels `node-role.kubernetes.io/master` or `node-role.kubernetes.io/infra`.

Finally, the profile `openshift-node` has the lowest priority of `30`. It lacks the `<match>` section and, therefore, will always match. It acts as a profile catch-all to set `openshift-node` profile, if no other profile with higher priority matches on a given node.

### 4.5. CUSTOM TUNING EXAMPLE

The following CR applies custom node-level tuning for OpenShift Container Platform nodes with label `tuned.openshift.io/ingress-node-label` set to any value. As an administrator, use the following command to create a custom tuned CR.

**Example**

```bash
oc create -f- <<_EOF_
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: ingress
  namespace: openshift-cluster-node-tuning-operator
spec:
_profile:
_EOF_
```
Custom profile writers are strongly encouraged to include the default tuned daemon profiles shipped within the default Tuned CR. The example above uses the default `openshift-control-plane` profile to accomplish this.

### 4.6. SUPPORTED TUNED DAEMON PLUG-INS

Excluding the `[main]` section, the following Tuned plug-ins are supported when using custom profiles defined in the `profile:` section of the Tuned CR:

- audio
- cpu
- disk
- eepc_she
- modules
- mounts
- net
- scheduler
- scsi_host
- selinux
- sysctl
- sysfs
- usb
- video
- vm

There is some dynamic tuning functionality provided by some of these plug-ins that is not supported. The following Tuned plug-ins are currently not supported:
- bootloader
- script
- systemd

See Available Tuned Plug-ins and Getting Started with Tuned for more information.
CHAPTER 5. USING CLUSTER LOADER

Cluster Loader is a tool that deploys large numbers of various objects to a cluster, which creates user-defined cluster objects. Build, configure, and run Cluster Loader to measure performance metrics of your OpenShift Container Platform deployment at various cluster states.

5.1. INSTALLING CLUSTER LOADER

Procedure

1. To pull the container image, run:

   $ sudo podman pull registry.svc.ci.openshift.org/ocp/4.4:tests

5.2. RUNNING CLUSTER LOADER

Prerequisites

- The repository will prompt you to authenticate. The registry credentials allow you to access the image, which is not publicly available. Use your existing authentication credentials from installation.

Procedure

1. Execute Cluster Loader using the built-in test configuration, which deploys five template builds and waits for them to complete:

   $ sudo podman run -v ${LOCAL_KUBECONFIG}:/root/.kube/config:z -i registry.svc.ci.openshift.org/ocp/4.4:tests /bin/bash -c 'export KUBECONFIG=/root/.kube/config && openshift-tests run-test "[Feature:Performance][Serial][Slow] Load cluster should load the cluster [Suite:openshift]"'

   Alternatively, execute Cluster Loader with a user-defined configuration by setting the environment variable for VIPERCONFIG:


   In this example, ${LOCAL_KUBECONFIG} refers to the path to the kubeconfig on your local file system. Also, there is a directory called ${LOCAL_CONFIG_FILE_PATH}, which is mounted into the container that contains a configuration file called test.yaml. Additionally, if the test.yaml references any external template files or podspec files, they should also be mounted into the container.

5.3. CONFIGURING CLUSTER LOADER

The tool creates multiple namespaces (projects), which contain multiple templates or Pods.
5.3.1. Example Cluster Loader configuration file

Cluster Loader’s configuration file is a basic YAML file:

```
provider: local
ClusterLoader:
  cleanup: true
projects:
  - num: 1
    basename: clusterloader-cakephp-mysql
    tuning: default
    ifexists: reuse
    templates:
      - num: 1
        file: cakephp-mysql.json

  - num: 1
    basename: clusterloader-dancer-mysql
    tuning: default
    ifexists: reuse
    templates:
      - num: 1
        file: dancer-mysql.json

  - num: 1
    basename: clusterloader-django-postgresql
    tuning: default
    ifexists: reuse
    templates:
      - num: 1
        file: django-postgresql.json

  - num: 1
    basename: clusterloader-nodejs-mongodb
    tuning: default
    ifexists: reuse
    templates:
      - num: 1
        file: quickstarts/nodejs-mongodb.json

  - num: 1
    basename: clusterloader-rails-postgresql
    tuning: default
    templates:
      - num: 1
        file: rails-postgresql.json

tuningsets: 2
  - name: default
    pods:
      stepping: 3
      stepsizes: 5
      pause: 0 s
      rate_limit: 4
      delay: 0 ms
```
Optional setting for end-to-end tests. Set to local to avoid extra log messages.

The tuning sets allow rate limiting and stepping, the ability to create several batches of Pods while pausing in between sets. Cluster Loader monitors completion of the previous step before continuing.

Stepping will pause for $M$ seconds after each $N$ objects are created.

Rate limiting will wait $M$ milliseconds between the creation of objects.

This example assumes that references to any external template files or podspec files are also mounted into the container.

## IMPORTANT

If you are running Cluster Loader on Microsoft Azure, then you must set the AZURE_AUTH_LOCATION variable to a file that contains the output of terraform.azure.auto.tfvars.json, which is present in the installer directory.

### 5.3.2. Configuration fields

**Table 5.1. Top-level Cluster Loader Fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cleanup</td>
<td>Set to true or false. One definition per configuration. If set to true, cleanup deletes all namespaces (projects) created by Cluster Loader at the end of the test.</td>
</tr>
<tr>
<td>projects</td>
<td>A sub-object with one or many definition(s). Under projects, each namespace to create is defined and projects has several mandatory subheadings.</td>
</tr>
<tr>
<td>tuningsets</td>
<td>A sub-object with one definition per configuration. tuningsets allows the user to define a tuning set to add configurable timing to project or object creation (Pods, templates, and so on).</td>
</tr>
<tr>
<td>sync</td>
<td>An optional sub-object with one definition per configuration. Adds synchronization possibilities during object creation.</td>
</tr>
</tbody>
</table>

**Table 5.2. Fields under projects**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>An integer. One definition of the count of how many projects to create.</td>
</tr>
</tbody>
</table>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>basename</td>
<td>A string. One definition of the base name for the project. The count of identical namespaces will be appended to <strong>basename</strong> to prevent collisions.</td>
</tr>
<tr>
<td>tuning</td>
<td>A string. One definition of what tuning set you want to apply to the objects, which you deploy inside this namespace.</td>
</tr>
<tr>
<td>ifexists</td>
<td>A string containing either <strong>reuse</strong> or <strong>delete</strong>. Defines what the tool does if it finds a project or namespace that has the same name of the project or namespace it creates during execution.</td>
</tr>
<tr>
<td>configmaps</td>
<td>A list of key-value pairs. The key is the ConfigMap name and the value is a path to a file from which you create the ConfigMap.</td>
</tr>
<tr>
<td>secrets</td>
<td>A list of key-value pairs. The key is the secret name and the value is a path to a file from which you create the secret.</td>
</tr>
<tr>
<td>pods</td>
<td>A sub-object with one or many definition(s) of Pods to deploy.</td>
</tr>
<tr>
<td>templates</td>
<td>A sub-object with one or many definition(s) of templates to deploy.</td>
</tr>
</tbody>
</table>

**Table 5.3. Fields under pods and templates**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num</td>
<td>An integer. The number of Pods or templates to deploy.</td>
</tr>
<tr>
<td>image</td>
<td>A string. The docker image URL to a repository where it can be pulled.</td>
</tr>
<tr>
<td>basename</td>
<td>A string. One definition of the base name for the template (or pod) that you want to create.</td>
</tr>
<tr>
<td>file</td>
<td>A string. The path to a local file, which is either a PodSpec or template to be created.</td>
</tr>
<tr>
<td>parameters</td>
<td>Key-value pairs. Under <strong>parameters</strong>, you can specify a list of values to override in the pod or template.</td>
</tr>
</tbody>
</table>
### Table 5.4. Fields under `tuningsets`

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>A string. The name of the tuning set which will match the name specified when defining a tuning in a project.</td>
</tr>
<tr>
<td>pods</td>
<td>A sub-object identifying the <code>tuningsets</code> that will apply to Pods.</td>
</tr>
<tr>
<td>templates</td>
<td>A sub-object identifying the <code>tuningsets</code> that will apply to templates.</td>
</tr>
</tbody>
</table>

### Table 5.5. Fields under `tuningsets pods` or `tuningsets templates`

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stepping</td>
<td>A sub-object. A stepping configuration used if you want to create an object in a step creation pattern.</td>
</tr>
<tr>
<td>rate_limit</td>
<td>A sub-object. A rate-limiting tuning set configuration to limit the object creation rate.</td>
</tr>
</tbody>
</table>

### Table 5.6. Fields under `tuningsets pods` or `tuningsets templates`, `stepping`

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stepsize</td>
<td>An integer. How many objects to create before pausing object creation.</td>
</tr>
<tr>
<td>pause</td>
<td>An integer. How many seconds to pause after creating the number of objects defined in <code>stepsize</code>.</td>
</tr>
<tr>
<td>timeout</td>
<td>An integer. How many seconds to wait before failure if the object creation is not successful.</td>
</tr>
<tr>
<td>delay</td>
<td>An integer. How many milliseconds (ms) to wait between creation requests.</td>
</tr>
</tbody>
</table>

### Table 5.7. Fields under `sync`

| Field   | Description |
Field | Description
---|---
server | A sub-object with `enabled` and `port` fields. The boolean `enabled` defines whether to start an HTTP server for pod synchronization. The integer `port` defines the HTTP server port to listen on (9090 by default).
running | A boolean. Wait for Pods with labels matching `selectors` to go into **Running** state.
succeeded | A boolean. Wait for Pods with labels matching `selectors` to go into **Completed** state.
selectors | A list of selectors to match Pods in **Running** or **Completed** states.
timeout | A string. The synchronization timeout period to wait for Pods in **Running** or **Completed** states. For values that are not 0, use units: [ns|us|ms|s|m|h].

### 5.4. KNOWN ISSUES

- Cluster Loader fails when called without configuration. ([BZ#1761925](#))

- If the IDENTIFIER parameter is not defined in user templates, template creation fails with error: unknown parameter name "IDENTIFIER". If you deploy templates, add this parameter to your template to avoid this error:

```json
{
   "name": "IDENTIFIER",
   "description": "Number to append to the name of resources",
   "value": "1"
}
```

If you deploy Pods, adding the parameter is unnecessary.
CHAPTER 6. USING CPU MANAGER

CPU Manager manages groups of CPUs and constrains workloads to specific CPUs.

CPU Manager is useful for workloads that have some of these attributes:

- Require as much CPU time as possible.
- Are sensitive to processor cache misses.
- Are low-latency network applications.
- Coordinate with other processes and benefit from sharing a single processor cache.

6.1. SETTING UP CPU MANAGER

Procedure

1. Optional: Label a node:

   ```
   # oc label node perf-node.example.com cpumanager=true
   ```

2. Edit the MachineConfigPool of the nodes where CPU Manager should be enabled. In this example, all workers have CPU Manager enabled:

   ```
   # oc edit machineconfigpool worker
   ```

3. Add a label to the worker MachineConfigPool:

   ```
   metadata:
   creationTimestamp: 2019-xx-xxx
   generation: 3
   labels:
   custom-kubelet: cpumanager-enabled
   ```

4. Create a KubeletConfig, cpumanager-kubeletconfig.yaml, custom resource (CR). Refer to the label created in the previous step to have the correct nodes updated with the new KubeletConfig. See the machineConfigPoolSelector section:

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: KubeletConfig
   metadata:
   name: cpumanager-enabled
   spec:
   machineConfigPoolSelector:
   matchLabels:
   custom-kubelet: cpumanager-enabled
   kubeletConfig:
   cpuManagerPolicy: static
   cpuManagerReconcilePeriod: 5s
   ```

5. Create the dynamic KubeletConfig:
This adds the CPU Manager feature to the KubeletConfig and, if needed, the Machine Config Operator (MCO) reboots the node. To enable CPU Manager, a reboot is not needed.

6. Check for the merged KubeletConfig:

```

"ownerReferences": [
  {
    "apiVersion": "machineconfiguration.openshift.io/v1",
    "kind": "KubeletConfig",
    "name": "cpumanager-enabled",
    "uid": "7ed5616d-6b72-11e9-aae1-021e1ce18878"
  }
],
```

7. Check the worker for the updated kubelet.conf:

```
# oc debug node/perf-node.example.com
sh-4.2# cat /host/etc/kubernetes/kubelet.conf | grep cpuManager

  cpuManagerPolicy: static  ①
cpuManagerReconcilePeriod: 5s  ②
```

① ② These settings were defined when you created the KubeletConfig CR.

8. Create a Pod that requests a core or multiple cores. Both limits and requests must have their CPU value set to a whole integer. That is the number of cores that will be dedicated to this Pod:

```
# cat cpumanager-pod.yaml
apiVersion: v1
kind: Pod
metadata:
generateName: cpumanager-
spec:
  containers:
  - name: cpumanager
    image: gcr.io/google_containers/pause-amd64:3.0
    resources:
      requests:
        cpu: 1
        memory: "1G"
      limits:
        cpu: 1
        memory: "1G"
    nodeSelector:
      cpumanager: "true"
```

9. Create the Pod:

```
# oc create -f cpumanager-pod.yaml
```
10. Verify that the Pod is scheduled to the node that you labeled:

```
# oc describe pod cpumanager
Name:               cpumanager-6cqz7
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:  perf-node.example.com/xxx.xx.xx.xxx
...
Limits:
  cpu:     1
  memory:  1G
Requests:
  cpu:     1
  memory:  1G
...
QoS Class:       Guaranteed
Node-Selectors:  cpumanager=true
```

11. Verify that the cgroups are set up correctly. Get the process ID (PID) of the pause process:

```
# cd /sys/fs/cgroup/cpuset/kubepods.slice/kubepods-pod69c01f8e_6b74_11e9_ac0f_0a2b62178a22.slice
# for i in `ls cpuset.cpus tasks` ; do echo -n "$i " ; cat $i ; done
cpuset.cpus 1
tasks 32706
```

Pods of quality of service (QoS) tier Guaranteed are placed within the kubepods.slice. Pods of other QoS tiers end up in child cgroups of kubepods:

```
# cd /sys/fs/cgroup/cpuset/kubepods.slice/kubepods-pod69c01f8e_6b74_11e9_ac0f_0a2b62178a22.slice/crio-b5437308f1a574c542bdf08563b865c0345c8f8c0b0a655612c.scope
# for i in `ls cpuset.cpus tasks` ; do echo -n "$i " ; cat $i ; done
cpuset.cpus 1
tasks 32706
```

12. Check the allowed CPU list for the task:

```
# grep ^Cpus_allowed_list /proc/32706/status
Cpus_allowed_list:    1
```

13. Verify that another pod (in this case, the pod in the burstable QoS tier) on the system cannot run on the core allocated for the Guaranteed pod:

```
# cat /sys/fs/cgroup/cpuset/kubepods.slice/kubepods-burstabled.slice/kubepods-burstabled-podc494a073_6b77_11e9_98c0_06bba5c387ea.slice/crio-c56982f57b75a2420947f0afc6cafe7534c5734efc34157525fa9abbf99e3849.scope/cpuset.cpus
0

# oc describe node perf-node.example.com
```
This VM has two CPU cores. You set `kube-reserved` to 500 millicores, meaning half of one core is subtracted from the total capacity of the node to arrive at the Node Allocatable amount. You can see that Allocatable CPU is 1500 millicores. This means you can run one of the CPU Manager pods since each will take one whole core. A whole core is equivalent to 1000 millicores. If you try to schedule a second pod, the system will accept the pod, but it will never be scheduled:
CHAPTER 7. USING TOPOLOGY MANAGER

Topology Manager is a Kubelet component that collects hints from CPU Manager and Device Manager to align pod CPU and device resources on the same non-uniform memory access (NUMA) node.

Topology Manager uses topology information from collected hints to decide if the pod can be accepted or rejected from the node, based on the configured Topology Manager policy and Pod resources requested.

Topology Manager is useful for workloads that desire to use hardware accelerators to support latency-critical execution and high throughput parallel computation.

NOTE

Topology Manager is an alpha feature in OpenShift Container Platform.

7.1. SETTING UP TOPOLOGY MANAGER

Prequisites

- Configure the CPU Manager policy to be static. Refer to Using CPU Manager in the Scalability and Performance section.

Procedure

1. Enable the LatencySensitive FeatureGate

   # oc edit featuregate/cluster

2. Add Feature Set: LatencySensitive to the spec:

   # oc describe featuregate/cluster

   Name:         cluster
   Namespace:    <none>
   Labels:       <none>
   Annotations:  release.openshift.io/create-only: true
   API Version:  config.openshift.io/v1
   Kind:         FeatureGate
   Metadata:
   Creation Timestamp: 2019-10-30T15:06:41Z
   Generation:          2
   Resource Version:    7773803
   Self Link:           /apis/config.openshift.io/v1/featuregates/cluster
   UID:                 b00204ab-cc5e-4ca5-ad93-b9bdd738c1de
   Spec:
   Feature Set:  LatencySensitive
   Events:         <none>

3. Configure the Topology Manager policy with KubeletConfig.
   The example YAML file below shows a single-numa-node policy specified.

   apiVersion: machineconfiguration.openshift.io/v1
7.2. TOPOLOGY MANAGER POLICIES

Topology Manager works on Nodes and Pods that meet the following conditions:

- The Node’s CPU Manager Policy is configured as **static**.
- The Pods are in the **Guaranteed** QoS class.

When the above conditions are met, Topology Manager will align CPU and device requests for the Pod.

Topology Manager supports 4 allocation policies. These policies are set via a Kubelet flag, `--topology-manager-policy`. The policies are:

- **none** (default)
- **best-effort**
- **restricted**
- **single-numa-node**

### 7.2.1. none policy

This is the default policy and does not perform any topology alignment.

### 7.2.2. best-effort policy

For each container in a Guaranteed Pod with the best-effort topology management policy, kublet calls each Hint Provider to discover their resource availability. Using this information, the Topology Manager stores the preferred NUMA Node affinity for that container. If the affinity is not preferred, Topology Manager will store this and admit the pod to the node anyway.

### 7.2.3. restricted policy

For each container in a Guaranteed Pod with the restricted topology management policy, kublet calls each Hint Provider to discover their resource availability. Using this information, the Topology Manager stores the preferred NUMA Node affinity for that container. If the affinity is not preferred, Topology
Manager will reject this pod from the node. This will result in a pod in a Terminated state with a pod admission failure.

### 7.2.4. single-numa-node

For each container in a Guaranteed Pod with the single-numa-node topology management policy, kublet calls each Hint Provider to discover their resource availability. Using this information, the Topology Manager determines if a single NUMA Node affinity is possible. If it is, the pod will be admitted to the node. If this is not possible then the Topology Manager will reject the pod from the node. This will result in a pod in a Terminated state with a pod admission failure.

### 7.3. POD INTERACTIONS WITH TOPOLOGY MANAGER POLICIES

The example Pod specs below help illustrate Pod interactions with Topology Manager.

The following Pod runs in the **BestEffort** QoS class because no resource requests or limits are specified.

```yaml
spec:
  containers:
  - name: nginx
    image: nginx
```

The next Pod runs in the **Burstable** QoS class because requests are less than limits.

```yaml
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
      requests:
        memory: "100Mi"
```

If the selected policy is anything other than **none**, Topology Manager would not consider either of these Pod specifications.

The last example Pod below runs in the Guaranteed QoS class because requests are equal to limits.

```yaml
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
        cpu: "2"
        example.com/device: "1"
      requests:
        memory: "200Mi"
        cpu: "2"
        example.com/device: "1"
```
Topology Manager would consider this Pod. The Topology Manager consults the CPU Manager static policy, which returns the topology of available CPUs. Topology Manager also consults Device Manager to discover the topology of available devices for example.com/device.

Topology Manager will use this information to store the best Topology for this container. In the case of this Pod, CPU Manager and Device Manager will use this stored information at the resource allocation stage.
CHAPTER 8. SCALING THE CLUSTER MONITORING OPERATOR

OpenShift Container Platform exposes metrics that the Cluster Monitoring Operator collects and stores in the Prometheus-based monitoring stack. As an administrator, you can view system resources, containers and components metrics in one dashboard interface, Grafana.

8.1. PROMETHEUS DATABASE STORAGE REQUIREMENTS

Red Hat performed various tests for different scale sizes.

Table 8.1. Prometheus Database storage requirements based on number of nodes/pods in the cluster

<table>
<thead>
<tr>
<th>Number of Nodes</th>
<th>Number of Pods</th>
<th>Prometheus storage growth per day</th>
<th>Prometheus storage growth per 15 days</th>
<th>RAM Space (per scale size)</th>
<th>Network (per tsdb chunk)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>1800</td>
<td>6.3 GB</td>
<td>94 GB</td>
<td>6 GB</td>
<td>16 MB</td>
</tr>
<tr>
<td>100</td>
<td>3600</td>
<td>13 GB</td>
<td>195 GB</td>
<td>10 GB</td>
<td>26 MB</td>
</tr>
<tr>
<td>150</td>
<td>5400</td>
<td>19 GB</td>
<td>283 GB</td>
<td>12 GB</td>
<td>36 MB</td>
</tr>
<tr>
<td>200</td>
<td>7200</td>
<td>25 GB</td>
<td>375 GB</td>
<td>14 GB</td>
<td>46 MB</td>
</tr>
</tbody>
</table>

Approximately 20 percent of the expected size was added as overhead to ensure that the storage requirements do not exceed the calculated value.

The above calculation is for the default OpenShift Container Platform Cluster Monitoring Operator.

NOTE

CPU utilization has minor impact. The ratio is approximately 1 core out of 40 per 50 nodes and 1800 pods.

Lab environment

In a previous release, all experiments were performed in an OpenShift Container Platform on OpenStack environment:

- Infra nodes (VMs) - 40 cores, 157 GB RAM.
- CNS nodes (VMs) - 16 cores, 62 GB RAM, NVMe drives.

IMPORTANT

Currently, OpenStack environments are not supported for OpenShift Container Platform 4.4.
Recommendations for OpenShift Container Platform

- Use at least three infrastructure (infra) nodes.
- Use at least three `openshift-container-storage` nodes with non-volatile memory express (NVMe) drives.

### 8.2. CONFIGURING CLUSTER MONITORING

**Procedure**

To increase the storage capacity for Prometheus:

1. Create a YAML configuration file, `cluster-monitoring-config.yml`. For example:

```yaml
apiVersion: v1
kind: ConfigMap
data:
  config.yaml: |
    prometheusOperator:
      baseImage: quay.io/coreos/prometheus-operator
    prometheusConfigReloaderBaseImage: quay.io/coreos/prometheus-config-reloader
    configReloaderBaseImage: quay.io/coreos/configmap-reload
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    prometheusK8s:
      retention: {{PROMETHEUS_RETENTION_PERIOD}}
      baseImage: openshift/prometheus
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      volumeClaimTemplate:
        spec:
          storageClassName: gp2
          resources:
            requests:
              storage: {{PROMETHEUS_STORAGE_SIZE}}
    alertmanagerMain:
      baseImage: openshift/prometheus-alertmanager
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      volumeClaimTemplate:
        spec:
          storageClassName: gp2
          resources:
            requests:
              storage: {{ALERTMANAGER_STORAGE_SIZE}}
    nodeExporter:
      baseImage: openshift/prometheus-node-exporter
    kubeRbacProxy:
      baseImage: quay.io/coreos/kube-rbac-proxy
    kubeStateMetrics:
      baseImage: quay.io/coreos/kube-state-metrics
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    grafana:
```
A typical value is `PROMETHEUS_RETENTION_PERIOD=15d`. Units are measured in time using one of these suffixes: s, m, h, d.

A typical value is `PROMETHEUS_STORAGE_SIZE=2000Gi`. Storage values can be a plain integer or as a fixed-point integer using one of these suffixes: E, P, T, G, M, K. You can also use the power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki.

A typical value is `ALERTMANAGER_STORAGE_SIZE=20Gi`. Storage values can be a plain integer or as a fixed-point integer using one of these suffixes: E, P, T, G, M, K. You can also use the power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki.

2. Set the values like the retention period and storage sizes.

3. Apply the changes by running:

   ```sh
   $ oc create -f cluster-monitoring-config.yml
   ```
CHAPTER 9. PLANNING YOUR ENVIRONMENT ACCORDING TO OBJECT MAXIMUMS

Consider the following tested object maximums when you plan your OpenShift Container Platform cluster.

These guidelines are based on the largest possible cluster. For smaller clusters, the maximums are lower. There are many factors that influence the stated thresholds, including the etcd version or storage data format.

In most cases, exceeding these numbers results in lower overall performance. It does not necessarily mean that the cluster will fail.

9.1. OPENSShift CONTAINER PLATFORM TESTED CLUSTER MAXIMUMS FOR MAJOR RELEASES


<table>
<thead>
<tr>
<th>Maximum type</th>
<th>3.x tested maximum</th>
<th>4.x tested maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Nodes</td>
<td>2,000</td>
<td>2,000</td>
</tr>
<tr>
<td>Number of Pods (^a)</td>
<td>150,000</td>
<td>150,000</td>
</tr>
<tr>
<td>Number of Pods per node</td>
<td>250</td>
<td>500 (^b)</td>
</tr>
<tr>
<td>Number of Pods per core</td>
<td>There is no default value.</td>
<td>There is no default value.</td>
</tr>
<tr>
<td>Number of Namespaces (^c)</td>
<td>10,000</td>
<td>10,000</td>
</tr>
<tr>
<td>Number of Builds</td>
<td>10,000 (Default pod RAM 512 Mi) - Pipeline Strategy</td>
<td>10,000 (Default pod RAM 512 Mi) - Source-to-Image (S2I) build strategy</td>
</tr>
<tr>
<td>Number of Pods per namespace (^d)</td>
<td>25,000</td>
<td>25,000</td>
</tr>
<tr>
<td>Number of Services (^e)</td>
<td>10,000</td>
<td>10,000</td>
</tr>
<tr>
<td>Number of Services per Namespace</td>
<td>5,000</td>
<td>5,000</td>
</tr>
<tr>
<td>Number of Back-ends per Service</td>
<td>5,000</td>
<td>5,000</td>
</tr>
<tr>
<td>Number of Deployments per Namespace (^d)</td>
<td>2,000</td>
<td>2,000</td>
</tr>
</tbody>
</table>
The Pod count displayed here is the number of test Pods. The actual number of Pods depends on the application’s memory, CPU, and storage requirements.

This was tested on a cluster with 100 worker nodes with 500 Pods per worker node. The default maxPods is still 250. To get to 500 maxPods, the cluster must be created with a hostPrefix of 22 in the install-config.yaml file and maxPods set to 500 using a custom KubeletConfig. The maximum number of Pods with attached Persistent Volume Claims (PVC) depends on storage backend from where PVC are allocated. In our tests, only OpenShift Container Storage v4 (OCS v4) was able to satisfy the number of Pods per node discussed in this document.

When there are a large number of active projects, etcd might suffer from poor performance if the keyspace grows excessively large and exceeds the space quota. Periodic maintenance of etcd, including defragmentation, is highly recommended to free etcd storage.

There are a number of control loops in the system that must iterate over all objects in a given namespace as a reaction to some changes in state. Having a large number of objects of a given type in a single namespace can make those loops expensive and slow down processing given state changes. The limit assumes that the system has enough CPU, memory, and disk to satisfy the application requirements.

Each Service port and each Service back-end has a corresponding entry in iptables. The number of back-ends of a given Service impact the size of the endpoints objects, which impacts the size of data that is being sent all over the system.

### 9.2. OPENSHIFT CONTAINER PLATFORM TESTED CLUSTER MAXIMUMS

<table>
<thead>
<tr>
<th>Maximum type</th>
<th>3.x tested maximum</th>
<th>4.x tested maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Nodes</td>
<td>2,000</td>
<td>2,000</td>
</tr>
<tr>
<td>Number of Pods [a]</td>
<td>150,000</td>
<td>150,000</td>
</tr>
<tr>
<td>Number of Pods per node</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>Number of Pods per core</td>
<td>There is no default value.</td>
<td>There is no default value.</td>
</tr>
<tr>
<td>Number of Namespaces [b]</td>
<td>10,000</td>
<td>10,000</td>
</tr>
<tr>
<td>Number of Builds</td>
<td>10,000 (Default pod RAM 512 Mi) - Pipeline Strategy</td>
<td>10,000 (Default pod RAM 512 Mi) - Pipeline Strategy</td>
</tr>
<tr>
<td>Number of Pods per Namespace [c]</td>
<td>25,000</td>
<td>25,000</td>
</tr>
</tbody>
</table>

[a] The Pod count displayed here is the number of test Pods. The actual number of Pods depends on the application’s memory, CPU, and storage requirements.

[b] This was tested on a cluster with 100 worker nodes with 500 Pods per worker node. The default maxPods is still 250. To get to 500 maxPods, the cluster must be created with a hostPrefix of 22 in the install-config.yaml file and maxPods set to 500 using a custom KubeletConfig. The maximum number of Pods with attached Persistent Volume Claims (PVC) depends on storage backend from where PVC are allocated. In our tests, only OpenShift Container Storage v4 (OCS v4) was able to satisfy the number of Pods per node discussed in this document.

[c] When there are a large number of active projects, etcd might suffer from poor performance if the keyspace grows excessively large and exceeds the space quota. Periodic maintenance of etcd, including defragmentation, is highly recommended to free etcd storage.

[d] There are a number of control loops in the system that must iterate over all objects in a given namespace as a reaction to some changes in state. Having a large number of objects of a given type in a single namespace can make those loops expensive and slow down processing given state changes. The limit assumes that the system has enough CPU, memory, and disk to satisfy the application requirements.

[e] Each Service port and each Service back-end has a corresponding entry in iptables. The number of back-ends of a given Service impact the size of the endpoints objects, which impacts the size of data that is being sent all over the system.
<table>
<thead>
<tr>
<th>Maximum type</th>
<th>4.1 tested maximum</th>
<th>4.2 tested maximum</th>
<th>4.3 tested maximum</th>
<th>4.4 tested maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Services [d]</td>
<td>10,000</td>
<td>10,000</td>
<td>10,000</td>
<td>10,000</td>
</tr>
<tr>
<td>Number of Services per Namespace</td>
<td>5,000</td>
<td>5,000</td>
<td>5,000</td>
<td>5,000</td>
</tr>
<tr>
<td>Number of Back-ends per Service</td>
<td>5,000</td>
<td>5,000</td>
<td>5,000</td>
<td>5,000</td>
</tr>
<tr>
<td>Number of Deployments per Namespace [c]</td>
<td>2,000</td>
<td>2,000</td>
<td>2,000</td>
<td>2,000</td>
</tr>
</tbody>
</table>

[a] The Pod count displayed here is the number of test Pods. The actual number of Pods depends on the application’s memory, CPU, and storage requirements.

[b] When there are a large number of active projects, etcd might suffer from poor performance if the keyspace grows excessively large and exceeds the space quota. Periodic maintenance of etcd, including defragmentation, is highly recommended to free etcd storage.

[c] There are a number of control loops in the system that must iterate over all objects in a given namespace as a reaction to some changes in state. Having a large number of objects of a given type in a single namespace can make those loops expensive and slow down processing given state changes. The limit assumes that the system has enough CPU, memory, and disk to satisfy the application requirements.

[d] Each service port and each service back-end has a corresponding entry in iptables. The number of back-ends of a given service impact the size of the endpoints objects, which impacts the size of data that is being sent all over the system.

In OpenShift Container Platform 4.4, half of a CPU core (500 millicore) is reserved by the system compared to OpenShift Container Platform 3.11 and previous versions.

### 9.3. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO TESTED CLUSTER MAXIMUMS

**IMPORTANT**

Oversubscribing the physical resources on a node affects resource guarantees the Kubernetes scheduler makes during pod placement. Learn what measures you can take to avoid memory swapping.

Some of the tested maximums are stretched only in a single dimension. They will vary when many objects are running on the cluster.

The numbers noted in this documentation are based on Red Hat’s test methodology, setup, configuration, and tunings. These numbers can vary based on your own individual setup and environments.

While planning your environment, determine how many pods are expected to fit per node:
Required Pods per Cluster / Pods per Node = Total Number of Nodes Needed

The current maximum number of pods per node is 250. However, the number of pods that fit on a node is dependent on the application itself. Consider the application’s memory, CPU, and storage requirements, as described in How to plan your environment according to application requirements.

Example scenario

If you want to scope your cluster for 2200 pods per cluster, you would need at least five nodes, assuming that there are 500 maximum pods per node:

$$\frac{2200}{500} = 4.4$$

If you increase the number of nodes to 20, then the pod distribution changes to 110 pods per node:

$$\frac{2200}{20} = 110$$

Where:

Required Pods per Cluster / Total Number of Nodes = Expected Pods per Node

9.4. HOW TO PLAN YOUR ENVIRONMENT ACCORDING TO APPLICATION REQUIREMENTS

Consider an example application environment:

<table>
<thead>
<tr>
<th>Pod type</th>
<th>Pod quantity</th>
<th>Max memory</th>
<th>CPU cores</th>
<th>Persistent storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>apache</td>
<td>100</td>
<td>500 MB</td>
<td>0.5</td>
<td>1 GB</td>
</tr>
<tr>
<td>node.js</td>
<td>200</td>
<td>1 GB</td>
<td>1</td>
<td>1 GB</td>
</tr>
<tr>
<td>postgresql</td>
<td>100</td>
<td>1 GB</td>
<td>2</td>
<td>10 GB</td>
</tr>
<tr>
<td>JBoss EAP</td>
<td>100</td>
<td>1 GB</td>
<td>1</td>
<td>1 GB</td>
</tr>
</tbody>
</table>

Extrapolated requirements: 550 CPU cores, 450GB RAM, and 1.4TB storage.

Instance size for nodes can be modulated up or down, depending on your preference. Nodes are often resource overcommitted. In this deployment scenario, you can choose to run additional smaller nodes or fewer larger nodes to provide the same amount of resources. Factors such as operational agility and cost-per-instance should be considered.

<table>
<thead>
<tr>
<th>Node type</th>
<th>Quantity</th>
<th>CPUs</th>
<th>RAM (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes (option 1)</td>
<td>100</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>Nodes (option 2)</td>
<td>50</td>
<td>8</td>
<td>32</td>
</tr>
</tbody>
</table>
Some applications lend themselves well to overcommitted environments, and some do not. Most Java applications and applications that use huge pages are examples of applications that would not allow for overcommitment. That memory cannot be used for other applications. In the example above, the environment would be roughly 30 percent overcommitted, a common ratio.
CHAPTER 10. OPTIMIZING STORAGE

Optimizing storage helps to minimize storage use across all resources. By optimizing storage, administrators help ensure that existing storage resources are working in an efficient manner.

10.1. AVAILABLE PERSISTENT STORAGE OPTIONS

Understand your persistent storage options so that you can optimize your OpenShift Container Platform environment.

Table 10.1. Available storage options

<table>
<thead>
<tr>
<th>Storage type</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
</table>
| Block        | - Presented to the operating system (OS) as a block device  
- Suitable for applications that need full control of storage and operate at a low level on files bypassing the file system  
- Also referred to as a Storage Area Network (SAN)  
- Non-shareable, which means that only one client at a time can mount an endpoint of this type | AWS EBS and VMware vSphere support dynamic persistent volume (PV) provisioning natively in OpenShift Container Platform. |
| File         | - Presented to the OS as a file system export to be mounted  
- Also referred to as Network Attached Storage (NAS)  
- Concurrency, latency, file locking mechanisms, and other capabilities vary widely between protocols, implementations, vendors, and scales. | RHEL NFS, NetApp NFS [a], and Vendor NFS |
| Object       | - Accessible through a REST API endpoint  
- Configurable for use in the OpenShift Container Platform Registry  
- Applications must build their drivers into the application and/or container. | AWS S3 |

[a] NetApp NFS supports dynamic PV provisioning when using the Trident plug-in.
10.2. RECOMMENDED CONFIGURABLE STORAGE TECHNOLOGY

The following table summarizes the recommended and configurable storage technologies for the given OpenShift Container Platform cluster application.

Table 10.2. Recommended and configurable storage technology

<table>
<thead>
<tr>
<th>Storage type</th>
<th>ROX [a]</th>
<th>RWX [b]</th>
<th>Registry</th>
<th>Scaled registry</th>
<th>Metrics [c]</th>
<th>Logging</th>
<th>Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
<td>Yes [d]</td>
<td>No</td>
<td>Configurable</td>
<td>Not configurable</td>
<td>Recommended</td>
<td>Recommended</td>
<td>Recommended</td>
</tr>
<tr>
<td>File</td>
<td>Yes [d]</td>
<td>Yes</td>
<td>Configurable</td>
<td>Configurable [e]</td>
<td>Configurable [f]</td>
<td>Configurable</td>
<td>Recommended</td>
</tr>
<tr>
<td>Object</td>
<td>Yes</td>
<td>Yes</td>
<td>Recommended</td>
<td>Recommended</td>
<td>Not configurable</td>
<td>Not configurable</td>
<td>Not configurable [g]</td>
</tr>
</tbody>
</table>

[a] ReadOnlyMany
[b] ReadWriteMany
[c] Prometheus is the underlying technology used for metrics.
[d] This does not apply to physical disk, VM physical disk, VMDK, loopback over NFS, AWS EBS, and Azure Disk.
[e] For metrics, using file storage with the ReadWriteMany (RWX) access mode is unreliable. If you use file storage, do not configure the RWX access mode on any PersistentVolumeClaims that are configured for use with metrics.
[f] For logging, using any shared storage would be an anti-pattern. One volume per elasticsearch is required.
[g] Object storage is not consumed through OpenShift Container Platform’s PVs/persistent volume claims (PVCs). Apps must integrate with the object storage REST API.

NOTE

A scaled registry is an OpenShift Container Platform registry where three or more pod replicas are running.

10.2.1. Specific application storage recommendations
IMPORTANT

Testing shows issues with using the NFS server on RHEL as storage backend for the container image registry. This includes the OpenShift Container Registry and Quay, Cassandra for metrics storage, and Elasticsearch for logging storage. Therefore, using NFS to back PVs used by core services is not recommended.

Other NFS implementations on the marketplace might not have these issues. Contact the individual NFS implementation vendor for more information on any testing that was possibly completed against these OpenShift core components.

10.2.1.1. Registry

In a non-scaled/high-availability (HA) OpenShift Container Platform registry cluster deployment:

- The preferred storage technology is object storage followed by block storage. The storage technology does not have to support RWX access mode.

- The storage technology must ensure read-after-write consistency. All NAS storage are not recommended for OpenShift Container Platform Registry cluster deployment with production workloads.

- While hostPath volumes are configurable for a non-scaled/HA OpenShift Container Platform Registry, they are not recommended for cluster deployment.

10.2.1.2. Scaled registry

In a scaled/HA OpenShift Container Platform registry cluster deployment:

- The preferred storage technology is object storage. The storage technology must support RWX access mode and must ensure read-after-write consistency.

- File storage and block storage are not recommended for a scaled/HA OpenShift Container Platform registry cluster deployment with production workloads.

- All NAS storage are not recommended for OpenShift Container Platform Registry cluster deployment with production workloads.

10.2.1.3. Metrics

In an OpenShift Container Platform hosted metrics cluster deployment:

- The preferred storage technology is block storage.

IMPORTANT

Testing shows significant unrecoverable corruptions using file storage and, therefore, file storage is not recommended for use with metrics.

There are file storage implementations in the marketplace that might not have these issues. Contact the individual storage vendor for more information on any testing that was possibly completed against these OpenShift core components.

10.2.1.4. Logging
In an OpenShift Container Platform hosted logging cluster deployment:

- The preferred storage technology is block storage.
- It is not recommended to use NAS storage for a hosted metrics cluster deployment with production workloads.

**IMPORTANT**

Testing shows issues with using the NFS server on RHEL as storage backend for the container image registry. This includes Elasticsearch for logging storage. Therefore, using NFS to back PVs used by core services is not recommended.

Other NFS implementations on the marketplace might not have these issues. Contact the individual NFS implementation vendor for more information on any testing that was possibly completed against these OpenShift core components.

10.2.1.5. Applications

Application use cases vary from application to application, as described in the following examples:

- Storage technologies that support dynamic PV provisioning have low mount time latencies, and are not tied to nodes to support a healthy cluster.
- Application developers are responsible for knowing and understanding the storage requirements for their application, and how it works with the provided storage to ensure that issues do not occur when an application scales or interacts with the storage layer.

10.2.2. Other specific application storage recommendations

- OpenShift Container Platform Internal **etcd**: For the best **etcd** reliability, the lowest consistent latency storage technology is preferable.
- It is highly recommended that you use **etcd** with storage that handles small read/write operations quickly, such as SSD. Ceph, NFS, and spinning disks are not recommended.
- OpenStack Cinder: OpenStack Cinder tends to be adept in ROX access mode use cases.
- Databases: Databases (RDBMSs, NoSQL DBs, etc.) tend to perform best with dedicated block storage.
CHAPTER 11. OPTIMIZING ROUTING

The OpenShift Container Platform HAProxy router scales to optimize performance.

11.1. BASELINE ROUTER PERFORMANCE

The OpenShift Container Platform router is the Ingress point for all external traffic destined for OpenShift Container Platform services.

When evaluating a single HAProxy router performance in terms of HTTP requests handled per second, the performance varies depending on many factors. In particular:

- HTTP keep-alive/close mode
- route type
- TLS session resumption client support
- number of concurrent connections per target route
- number of target routes
- back end server page size
- underlying infrastructure (network/SDN solution, CPU, and so on)

While performance in your specific environment will vary, Red Hat lab tests on a public cloud instance of size 4 vCPU/16GB RAM, a single HAProxy router handling 100 routes terminated by backends serving 1kB static pages is able to handle the following number of transactions per second.

In HTTP keep-alive mode scenarios:

<table>
<thead>
<tr>
<th>Encryption</th>
<th>LoadBalancerService</th>
<th>HostNetwork</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>21515</td>
<td>29622</td>
</tr>
<tr>
<td>edge</td>
<td>16743</td>
<td>22913</td>
</tr>
<tr>
<td>passthrough</td>
<td>36786</td>
<td>53295</td>
</tr>
<tr>
<td>re-encrypt</td>
<td>21583</td>
<td>25198</td>
</tr>
</tbody>
</table>

In HTTP close (no keep-alive) scenarios:

<table>
<thead>
<tr>
<th>Encryption</th>
<th>LoadBalancerService</th>
<th>HostNetwork</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>5719</td>
<td>8273</td>
</tr>
<tr>
<td>edge</td>
<td>2729</td>
<td>4069</td>
</tr>
<tr>
<td>passthrough</td>
<td>4121</td>
<td>5344</td>
</tr>
</tbody>
</table>
Default router configuration with `ROUTER_THREADS=4` was used and two different endpoint publishing strategies (LoadBalancerService/HostNetwork) tested. TLS session resumption was used for encrypted routes. With HTTP keep-alive, a single HAProxy router is capable of saturating 1 Gbit NIC at page sizes as small as 8 kB.

When running on bare metal with modern processors, you can expect roughly twice the performance of the public cloud instance above. This overhead is introduced by the virtualization layer in place on public clouds and holds mostly true for private cloud-based virtualization as well. The following table is a guide on how many applications to use behind the router:

<table>
<thead>
<tr>
<th>Number of applications</th>
<th>Application type</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-10</td>
<td>static file/web server or caching proxy</td>
</tr>
<tr>
<td>100-1000</td>
<td>applications generating dynamic content</td>
</tr>
</tbody>
</table>

In general, HAProxy can support routes for 5 to 1000 applications, depending on the technology in use. Router performance might be limited by the capabilities and performance of the applications behind it, such as language or static versus dynamic content.

Router sharding should be used to serve more routes towards applications and help horizontally scale the routing tier.

### 11.2. ROUTER PERFORMANCE OPTIMIZATIONS

OpenShift Container Platform no longer supports modifying router deployments by setting environment variables such as `ROUTER_THREADS`, `ROUTER_DEFAULT_TUNNEL_TIMEOUT`, `ROUTER_DEFAULT_CLIENT_TIMEOUT`, `ROUTER_DEFAULT_SERVER_TIMEOUT`, and `RELOAD_INTERVAL`.

You can modify the router deployment, but if the Ingress Operator is enabled, the configuration is overwritten.
CHAPTER 12. WHAT HUGE PAGES DO AND HOW THEY ARE CONSUMED BY APPLICATIONS

12.1. WHAT HUGE PAGES DO

Memory is managed in blocks known as pages. On most systems, a page is 4Ki. 1Mi of memory is equal to 256 pages; 1Gi of memory is 256,000 pages, and so on. CPUs have a built-in memory management unit that manages a list of these pages in hardware. The Translation Lookaside Buffer (TLB) is a small hardware cache of virtual-to-physical page mappings. If the virtual address passed in a hardware instruction can be found in the TLB, the mapping can be determined quickly. If not, a TLB miss occurs, and the system falls back to slower, software-based address translation, resulting in performance issues. Since the size of the TLB is fixed, the only way to reduce the chance of a TLB miss is to increase the page size.

A huge page is a memory page that is larger than 4Ki. On x86_64 architectures, there are two common huge page sizes: 2Mi and 1Gi. Sizes vary on other architectures. In order to use huge pages, code must be written so that applications are aware of them. Transparent Huge Pages (THP) attempt to automate the management of huge pages without application knowledge, but they have limitations. In particular, they are limited to 2Mi page sizes. THP can lead to performance degradation on nodes with high memory utilization or fragmentation due to defragmenting efforts of THP, which can lock memory pages. For this reason, some applications may be designed to (or recommend) usage of pre-allocated huge pages instead of THP.

In OpenShift Container Platform, applications in a pod can allocate and consume pre-allocated huge pages.

12.2. HOW HUGE PAGES ARE CONSUMED BY APPS

Nodes must pre-allocate huge pages in order for the node to report its huge page capacity. A node can only pre-allocate huge pages for a single size.

Huge pages can be consumed through container-level resource requirements using the resource name `hugepages-<size>`, where size is the most compact binary notation using integer values supported on a particular node. For example, if a node supports 2048KiB page sizes, it exposes a schedulable resource `hugepages-2Mi`. Unlike CPU or memory, huge pages do not support over-commitment.

```yaml
apiVersion: v1
kind: Pod
metadata:
generateName: hugepages-volume-
spec:
  containers:
  - securityContext:
    privileged: true
    image: rhel7:latest
    command:
      - sleep
      - inf
    name: example
    volumeMounts:
      - mountPath: /dev/hugepages
        name: hugepage
        resources:
          limits:
```
hugerequests-2Mi: 100Mi
memory: "1Gi"
cpu: "1"

volumes:
- name: hugepage
  emptyDir:
    medium: HugePages

Specify the amount of memory for **hugepages** as the exact amount to be allocated. Do not specify this value as the amount of memory for **hugepages** multiplied by the size of the page. For example, given a huge page size of 2MB, if you want to use 100MB of huge-page-backed RAM for your application, then you would allocate 50 huge pages. OpenShift Container Platform handles the math for you. As in the above example, you can specify **100MB** directly.

### Allocating huge pages of a specific size

Some platforms support multiple huge page sizes. To allocate huge pages of a specific size, precede the huge pages boot command parameters with a huge page size selection parameter `hugepagesz=<size>`. The `<size>` value must be specified in bytes with an optional scale suffix [kKmMgG]. The default huge page size can be defined with the `default_hugepagesz=<size>` boot parameter.

### Huge page requirements

- Huge page requests must equal the limits. This is the default if limits are specified, but requests are not.
- Huge pages are isolated at a pod scope. Container isolation is planned in a future iteration.
- **EmptyDir** volumes backed by huge pages must not consume more huge page memory than the pod request.
- Applications that consume huge pages via `shmget()` with `SHM_HUGETLB` must run with a supplemental group that matches `proc/sys/vm/hugetlb_shm_group`.

### Additional resources

- **Configuring Transparent Huge Pages**

### 12.3. CONFIGURING HUGE PAGES

Nodes must pre-allocate huge pages used in an OpenShift Container Platform cluster. Use the Node Tuning Operator to allocate huge pages on a specific node.

### Procedure

1. Label the node so that the Node Tuning Operator knows on which node to apply the tuned profile, which describes how many huge pages should be allocated:

   ```bash
   $ oc label node <node_using_hugepages> hugepages=true
   ```

2. Create a file with the following content and name it `hugepages_tuning.yaml`:

   ```yaml
   apiVersion: tuned.openshift.io/v1
   kind: Tuned
   ```
Set the `name` parameter value to `hugepages`.

Set the `profile` section to allocate huge pages.

Set the `match` section to associate the profile to nodes with the `hugepages` label.

3. Create the custom `hugepages` tuned profile by using the `hugepages_tuning.yaml` file:

```sh
$ oc create -f hugepages_tuning.yaml
```

4. After creating the profile, the Operator applies the new profile to the correct node and allocates huge pages. Check the logs of a tuned pod on a node using huge pages to verify:

```sh
$ oc logs <tuned_pod_on_node_using_hugepages> \
  -n openshift-cluster-node-tuning-operator | grep 'applied' | tail -n1
```

```
2019-08-08 07:20:41.286 INFO tuned.daemon.daemon: static tuning from profile 'node-hugepages' applied
```