OpenShift Container Platform 3.5

Scaling and Performance Guide

OpenShift Container Platform 3.5 Scaling and Performance Guide

OpenShift Container Platform 3.5 Scaling and Performance Guide

OpenShift Container Platform 3.5 Scaling and Performance Guide

Legal Notice

Abstract

Scale up your cluster and tune performance in production environments

Table of Contents

8.5. WED APR 12 2017   25

CHAPTER 1. OVERVIEW

This guide provides procedures and examples for how to enhance your OpenShift Container Platform cluster performance and conduct scaling at different levels of an OpenShift Container Platform production stack. It includes recommended practices for building, scaling, and tuning OpenShift Container Platform clusters.

Tuning considerations can vary depending on your cluster setup, and be advised that any performance recommendations in this guide might come with trade-offs.

CHAPTER 2. RECOMMENDED INSTALLATION PRACTICES

2.1. PRE-INSTALLING DEPENDENCIES

A node host will access the network to install any RPMs dependencies, such as **atomic-openshift-***, **iptables**, and **docker**. Pre-installing these dependencies, creates a more efficient install, because the RPMs are only accessed when necessary, instead of a number of times per host during the install.

This is also useful for machines that cannot access the registry for security purposes.

2.2. ANSIBLE INSTALL OPTIMIZATION

The OpenShift Container Platform install method uses Ansible. Ansible is useful for running parallel operations, meaning a fast and efficient installation. However, these can be improved upon with additional tuning options. See the Configuring Ansible section for a list of available Ansible configuration options.

IMPORTANT

Parallel behavior can overwhelm a content source, such as your image registry or Red Hat Satellite server. Preparing your server's infrastructure pods and operating system patches can help prevent this issue.

Run the installer from the lowest-possible latency control node (LAN speeds). Running over a wide area network (WAN) is not advised, neither is running the installation over a lossy network connection.

Ansible provides its own guidance for performance and scaling, including using RHEL 6.6 or later to ensure the version of OpenSSH supports **ControlPersist**, and running the installer from the same LAN as the cluster, but *not* running it from a machine in the cluster.

The following is an example Ansible configuration for large cluster installation and administration that incorporates the recommendations documented by Ansible:

```
# cat /etc/ansible/ansible.cfg
# config file for ansible -- http://ansible.com/
# ==============================================
[defaults]
forks = 20 ❶
host_key_checking = False
remote_user = root
roles_path = roles/
gathering = smart
fact_caching = jsonfile
fact_caching_connection = $HOME/ansible/facts
fact_caching_timeout = 600
log_path = /var/log/ansible.log
nocows = 1
callback_whitelist = profile_tasks

[privilege_escalation]
become = False

[ssh_connection]
```

```
ssh_args = -o ControlMaster=auto -o ControlPersist=600s -o
ServerAliveInterval=60
control_path = %(directory)s/%%h-%%r
pipelining = True       2
timeout = 10
```

1 20 forks is ideal, because larger forks can lead to installations failing.

2 Pipelining reduces the number of connections between control and target nodes, helping to improve installer performance.

> **NOTE**
>
> By default, logging is disabled in Ansible. Ensure logging in the **/etc/ansible/ansible.cfg** file is not commented out.

2.3. NETWORKING CONSIDERATIONS

Network subnets can be changed post-install, but with difficulty. It is much easier to consider the network subnet size prior to installation, because underestimating the size can create problems with growing clusters.

See the Network Optimization topic for recommended network subnetting practices.

CHAPTER 3. RECOMMENDED HOST PRACTICES

3.1. RECOMMENDED PRACTICES FOR OPENSHIFT CONTAINER PLATFORM MASTER HOSTS

In addition to pod traffic, the most-used data-path in an OpenShift Container Platform infrastructure is between the OpenShift Container Platform master hosts and etcd. The OpenShift Container Platform API server (part of the master binary) consults etcd for node status, network configuration, secrets, and more.

Optimize this traffic path by:

- Co-locating master hosts and etcd servers.

- Ensuring an uncongested, low latency LAN communication link between master hosts.

3.2. RECOMMENDED PRACTICES FOR OPENSHIFT CONTAINER PLATFORM NODE HOSTS

The OpenShift Container Platform node configuration file at **/etc/origin/node/node-config.yaml** contains important options, such as the iptables synchronization period, the Maximum Transmission Unit (MTU) of the SDN network, and the proxy-mode.

The node configuration file allows you to pass arguments to the kubelet (node) process. You can view a list of possible options by running **`kubelet --help`**.
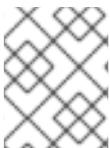
> **NOTE**
>
> Not all kubelet options are supported by OpenShift Container Platform, and are used in the upstream Kubernetes. This means certain options are in limited support.

In the **/etc/origin/node/node-config.yaml** file, two parameters control the maximum number of pods that can be scheduled to a node: **`pods-per-core`** and **`max-pods`**. When both options are in use, the lower of the two limits the number of pods on a node. Exceeding these values can result in:

- Increased CPU utilization on both OpenShift Container Platform and Docker.

- Slow pod scheduling.

- Potential out-of-memory scenarios (depends on the amount of memory in the node).

- Exhausting the pool of IP addresses.

- Resource overcommitting, leading to poor user application performance.

> **NOTE**
>
> 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.

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

```
kubeletArguments:
  pods-per-core:
    - "10"
```

> **NOTE**
>
> Setting **pods-per-core** to 0 disables this limit.

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

```
kubeletArguments:
  max-pods:
    - "250"
```

Using the above example, the default value for **pods-per-core** is **10** and the default value for **max-pods** is **250**. This means that unless the node has 25 cores or more, by default, **pods-per-core** will be the limiting factor.

See the Sizing Considerations section in the installation documentation for the recommended limits for an OpenShift Container Platform cluster. The recommended sizing accounts for OpenShift Container Platform and Docker coordination for container status updates. This coordination puts CPU pressure on the master and docker processes, which can include writing a large amount of log data.

3.3. RECOMMENDED PRACTICES FOR OPENSHIFT ETCD HOSTS

etcd is a distributed key-value store that OpenShift Container Platform uses for configuration.

> **IMPORTANT**
>
> After profiling etcd under OpenShift Container Platform, etcd frequently performs small amounts of storage input and output. Using etcd with storage that handles small read/write operations quickly, such as SSD, is highly recommended.

Optimize communication between etcd and master hosts either by co-locating them on the same host, or providing a dedicated network.

> **NOTE**
>
> etcd hosts are typically memory intensive.

3.4. SCALING HOSTS USING THE TUNED PROFILE

Tuned is a tuning profile delivery mechanism enabled by default in Red Hat Enterprise Linux and other Red Hat products. Tuned customizes Linux settings, such as sysctls, power management, and kernel command line options, to optimize the operating system for different workload performance and scalability requirements.

OpenShift Container Platform leverages the **tuned** daemon and includes Tuned profiles called **atomic-openshift-host** and **atomic-openshift-guest**. These profiles safely increase some of the commonly encountered vertical scaling limits present in the kernel, and are automatically applied to your system during installation.

The Tuned profiles support inheritance between profiles. On an OpenShift Container Platform system, the findings delivered by Tuned will be the union of **throughput-performance** (the default for RHEL) and **atomic-openshift-guest**. Tuned will determine if you are running OpenShift Container Platform on a virtual machine, and, if so, automatically apply **virtual-guest** tuning as well.

To see which Tuned profile is enabled on your system, run:

```
# tuned-adm active
Current active profile: atomic-openshift-node-guest
```

See the Red Hat Enterprise Linux Performance Tuning Guide for more information about Tuned.

CHAPTER 4. OPTIMIZING COMPUTE RESOURCES

4.1. OVERCOMMITTING

You can use overcommit procedures so that resources such as CPU and memory are more accessible to the parts of your cluster that need them.

Note that when you overcommit, there is a risk that another application may not have access to the resources it requires when it needs them, which will result in reduced performance. However, this may be an acceptable trade-off in favor of increased density and reduced costs. For example, development, quality assurance (QA), or test environments may be overcommited, whereas production might not be.

OpenShift Container Platform implements resource management through the compute resource model and quota system. See the documentation for more information about the OpenShift resource model.

For more information and strategies for overcommiting, see the Overcommitting documentation in the Cluster Administration Guide.

4.2. IMAGE CONSIDERATIONS

4.2.1. Using a Pre-deployed Image to Improve Efficiency

You can create a base OpenShift Container Platform image with a number of tasks built-in to improve efficiency, maintain configuration consistency on all node hosts, and reduce repetitive tasks. This is known as a pre-deployed image.

For example, because every node requires the **ose‑pod** image in order to run pods, each node has to periodically connect to the Docker registry in order to pull the latest image. This can become problematic when you have 100 nodes attempting this at the same time, and can lead to resource contention on the image registry, waste of network bandwidth, and increased pod launch times.

To build a pre-deployed image:

- Create an instance of the type and size required.

- Ensure a dedicated storage device is available for Docker local image or container storage, separate from any persistent volumes for containers.

- Fully update the system, and ensure Docker is installed.

- Ensure the host has access to all yum repositories.

- Set up thin-provisioned LVM storage.

- Pre-seed your commonly used images (such as the **rhel7** base image), as well as OpenShift Container Platform infrastructure container images (**ose-pod**, **ose-deployer**, etc.) into your pre-deployed image.

Ensure that pre-deployed images are configured for any appropriate cluster configurations, such as being able to run on OpenStack, or AWS, as well as any other cluster configurations.

4.2.2. Pre-pulling Images

To efficiently produce images, you can pre-pull any necessary container images to all node hosts. This means the image does not have to be initially pulled, which saves time and performance over slow connections, especially for images, such as S2I, metrics, and logging, which can be large.

This is also useful for machines that cannot access the registry for security purposes.

Alternatively, you can use a local image instead of the default of a specified registry. To do this:

1. Pull from local images by setting the **imagePullPolicy** parameter of a pod configuration to **IfNotPresent** or **Never**.

2. Ensure that all nodes in the cluster have the same images saved locally.

> **NOTE**
>
> Pulling from a local registry is suitable if you can control node configuration. However, it will not work reliably on cloud providers that do not replace nodes automatically, such as GCE. If you are running on Google Container Engine (GKE), there will already be a *.dockercfg* file on each node with Google Container Registry credentials.

4.3. DEBUGGING OPENSHIFT CONTAINER PLATFORM USING THE RHEL TOOLS CONTAINER

Red Hat distributes a **rhel-tools** container, which:

- Allow users to deploy minimal footprint container hosts by moving packages out of the base distribution and into this support container.

- Provide debugging capabilities for Red Hat Enterprise Linux 7 Atomic Host, which has an immutable packet tree. **rhel-tools** includes utilities such as tcpdump, sosreport, git, gdb, perf, and many more common system administration utilities.

Use the **rhel-tools** container with the following:

```
# atomic run rhel7/rhel-tools
```

See the RHEL Tools Container documentation for more information.

CHAPTER 5. OPTIMIZING STORAGE

5.1. OVERVIEW

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.

5.2. GENERAL STORAGE GUIDELINES

The following table lists the available persistent storage technologies for OpenShift Container Platform.

Table 5.1. Available storage options

Storage type	Description	Examples
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	CNS/CRS GlusterFS [a] iSCSI, Fibre Channel, Ceph RBD, OpenStack Cinder, AWS EBS [a], Dell/EMC Scale.IO, VMware vSphere Volume, GCE Persistent Disk [a], Azure Disk
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.	CNS/CRS GlusterFS [a], RHEL NFS, NetApp NFS [b] , Azure File, Vendor NFS, Vendor GlusterFS [c], Azure File, AWS EFS
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.	CNS/CRS GlusterFS [a], Ceph Object Storage (RADOS Gateway), OpenStack Swift, Aliyun OSS, AWS S3, Google Cloud Storage, Azure Blob Storage, Vendor S3 [c], Vendor Swift [c]

Storage type	Description	Examples

[a] CNS/CRS GlusterFS, Ceph RBD, OpenStack Cinder, AWS EBS, Azure Disk, GCE persistent disk, and VMware vSphere support dynamic persistent volume (PV) provisioning natively in OpenShift Container Platform.

[b] NetApp NFS supports dynamic PV provisioning when using the Trident plugin.

[c] Vendor GlusterFS, Vendor S3, and Vendor Swift supportability and configurability may vary.

NOTE

As of OpenShift Container Platform 3.6.1, Container-Native Storage (CNS) GlusterFS (a hyperconverged or cluster-hosted storage solution) and Container-Ready Storage (CRS) GlusterFS (an externally hosted storage solution) provides interfaces for block, file, and object storage for the purpose of the OpenShift Container Platform registry, logging, and metrics.

5.3. STORAGE RECOMMENDATIONS

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

Table 5.2. Recommended and configurable storage technology

Storage type	ROX [a]	RWX [b]	Registry	Scaled registry	Metrics	Logging	Apps
Block	Yes [c]	No	Configurable	Not configurable	Recommended	Recommended	Recommended
File	Yes [c]	Yes	Configurable	Configurable	Configurable	Configurable	Recommended
Object	Yes	Yes	Recommended	Recommended	Not configurable	Not configurable	Not configurable [d]

[a] ReadOnlyMany

[b] ReadWriteMany

[c] This does not apply to physical disk, VM physical disk, VMDK, loopback over NFS, AWS EBS, and Azure Disk.

[d] 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.

5.3.1. Specific Application Storage Recommendations

5.3.1.1. 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 need to support RWX access mode.

- The storage technology must ensure read-after-write consistency. All NAS storage (excluding CNS/CRS GlusterFS as it uses an object storage interface) 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.

> **WARNING**
>
> Corruption may occur when using NFS to back OpenShift Container Platform registry with production workloads.

5.3.1.2. 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 (excluding CNS/CRS GlusterFS as it uses an object storage interface) are not recommended for OpenShift Container Platform Registry cluster deployment with production workloads.

> **WARNING**
>
> Corruption may occur when using NFS to back OpenShift Container Platform scaled/HA registry with production workloads.

5.3.1.3. Metrics

In an OpenShift Container Platform hosted metrics cluster deployment:

- The preferred storage technology is block storage.

- It is not recommended to use NAS storage (excluding CNS/CRS GlusterFS as it uses a block storage interface from iSCSI) for a hosted metrics cluster deployment with production workloads.

> **WARNING**
>
> Corruption may occur when using NFS to back a hosted metrics cluster deployment with production workloads.

5.3.1.4. 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 (excluding CNS/CRS GlusterFS as it uses a block storage interface from iSCSI) for a hosted metrics cluster deployment with production workloads.

> **WARNING**
>
> Corruption may occur when using NFS to back hosted logging with production workloads.

5.3.1.5. 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.

- NFS does not guarantee read-after-write consistency and is not recommended for applications which require it.

- Applications that depend on writing to the same, shared NFS export may experience issues with production workloads.

5.3.2. Other Specific Application Storage Recommendations

- OpenShift Container Platform Internal **etcd**: For the best etcd reliability, the lowest consistent latency storage technology is preferable.

- 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.

5.4. CHOOSING A DOCKER GRAPH DRIVER

Docker stores images and containers in a graph driver (a pluggable storage backend), such as Device Mapper, Overlay, and Btrfs. Each have advantages and disadvantages. For example, Overlay is faster than Device Mapper at starting and stopping containers, but is not POSIX compliant because of the architectural limitations of a union file system, and does not yet support SELinux.

For more information about Overlay, including supportability and usage caveats, see the RHEL 7.3 Release Notes.

In production environments, using a LVM thin pool on top of regular block devices (not loop devices) for container images and container root file systems storage is recommended.

> **NOTE**
>
> Using a Loop device back-end can affect performance issues. While you can still continue to use it, Docker logs a warning message. For example:
>
> ```
> devmapper: Usage of loopback devices is strongly discouraged
> for production use.
> Please use `--storage-opt dm.thinpooldev` or use `man docker`
> to refer to
> dm.thinpooldev section.
> ```

To ease Docker backend storage configuration, use the **docker-storage-setup** utility, which automates much of the configuration details:

1. If you had a separate disk drive dedicated to Docker storage (for example, */dev/xvdb*), add the following to the */etc/sysconfig/docker-storage-setup* file:

   ```
   DEVS=/dev/xvdb
   VG=docker_vg
   ```

2. Restart the **docker-storage-setup** service:

   ```
   # systemctl restart docker-storage-setup
   ```

   After the restart, **docker-storage-setup** sets up a volume group named **docker_vg** and creates a thin pool logical volume. Documentation for thin provisioning on RHEL is available in the LVM Administrator Guide. View the newly created volumes with the **lsblk** command:

   ```
   # lsblk /dev/xvdb
   NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
   xvdb 202:16 0 20G 0 disk
   └─xvdb1 202:17 0 10G 0 part
     ├─docker_vg-docker--pool_tmeta 253:0 0 12M 0 lvm
   ```

```
| └─docker_vg-docker--pool 253:2 0 6.9G 0 lvm
└─docker_vg-docker--pool_tdata 253:1 0 6.9G 0 lvm
  └─docker_vg-docker--pool 253:2 0 6.9G 0 lvm
```

> **NOTE**
>
> Thin-provisioned volumes are not mounted and have no file system (individual containers do have an XFS file system), thus they will not show up in "df" output.

3. To verify that Docker is using a LVM thin pool, and to monitor disk space utilization, use the **docker info** command. The **Pool Name** will correspond with the **VG** you specified in */etc/sysconfig/docker-storage-setup*:

```
# docker info | egrep -i 'storage|pool|space|filesystem'
Storage Driver: devicemapper
 Pool Name: docker_vg-docker--pool
 Pool Blocksize: 524.3 kB
 Backing Filesystem: xfs
 Data Space Used: 62.39 MB
 Data Space Total: 6.434 GB
 Data Space Available: 6.372 GB
 Metadata Space Used: 40.96 kB
 Metadata Space Total: 16.78 MB
 Metadata Space Available: 16.74 MB
```

By default, a thin pool is configured to use 40% of the underlying block device. As you use the storage, LVM automatically extends the thin pool up to 100%. This is why the **Data Space Total** value does not match the full size of the underlying LVM device. This auto-extend technique was used to unify the storage approach taken in both Red Hat Enterprise Linux and Red Hat Atomic Host, which only uses a single partition.

In development, Docker in Red Hat distributions defaults to a loopback mounted sparse file. To see if your system is using the loopback mode:

```
# docker info|grep loop0
 Data file: /dev/loop0
```

> **IMPORTANT**
>
> Red Hat strongly recommends using the Device Mapper storage driver in thin pool mode for production workloads.

Overlay is also supported for Docker use cases as of Red Hat Enterprise Linux 7.2, and provides faster start up time and page cache sharing, which can potentially improve density by reducing overall memory utilization.

CHAPTER 6. ROUTING AND NETWORK OPTIMIZATION

6.1. SCALING OPENSHIFT CONTAINER PLATFORM HAPROXY ROUTER

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

On an public cloud instance of size 4 vCPU/16GB RAM, a single HAProxy router is able to handle between 7000-32000 HTTP keep-alive requests depending on encryption, page size, and the number of connections used. For example, when using TLS edge or re-encryption terminations with large page sizes and a high numbers of connections, expect to see results in the lower range. With HTTP keep-alive, a single HAProxy router is capable of saturating 1 Gbit NIC at page sizes as small as 8 kB.

The table below shows HTTP keep-alive performance on such a public cloud instance with a single HAProxy and 100 routes:

Encryption	Page size	HTTP(s) requests per second
none	1kB	15435
none	4kB	11947
edge	1kB	7467
edge	4kB	7678
passthrough	1kB	25789
passthrough	4kB	17876
re-encrypt	1kB	7611
re-encrypt	4kB	7395

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:

Number of applications	Application type
5-10	static file/web server or caching proxy
100-1000	applications generating dynamic content

In general, HAProxy can saturate about 5-1000 applications, depending on the technology in use. The number will typically be lower for applications serving only static content.

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

6.2. OPTIMIZING NETWORK PERFORMANCE

The OpenShift SDN uses OpenvSwitch, virtual extensible LAN (VXLAN) tunnels, OpenFlow rules, and iptables. This network can be tuned by using jumbo frames, network interface cards (NIC) offloads, multi-queue, and ethtool settings.

VXLAN provides benefits over VLANs, such as an increase in networks from 4096 to over 16 million, and layer 2 connectivity across physical networks. This allows for all pods behind a service to communicate with each other, even if they are running on different systems.

VXLAN encapsulates all tunneled traffic in user datagram protocol (UDP) packets. However, this leads to increased CPU utilization. Both these outer- and inner-packets are subject to normal checksumming rules to guarantee data has not been corrupted during transit. Depending on CPU performance, this additional processing overhead can cause a reduction in throughput and increased latency when compared to traditional, non-overlay networks.

Cloud, VM, and bare metal CPU performance can be capable of handling much more than one Gbps network throughput. When using higher bandwidth links such as 10 or 40 Gbps, reduced performance can occur. This is a known issue in VXLAN-based environments and is not specific to containers or OpenShift Container Platform. Any network that relies on VXLAN tunnels will perform similarly because of the VXLAN implementation.

If you are looking to push beyond one Gbps, you can:

- Use Native Container Routing. This option has important operational caveats that do not exist when using OpenShift SDN, such as updating routing tables on a router.

- Evaluate network plug-ins that implement different routing techniques, such as border gateway protocol (BGP).

- Use VXLAN-offload capable network adapters. VXLAN-offload moves the packet checksum calculation and associated CPU overhead off of the system CPU and onto dedicated hardware on the network adapter. This frees up CPU cycles for use by pods and applications, and allows users to utilize the full bandwidth of their network infrastructure.

VXLAN-offload does not reduce latency. However, CPU utilization is reduced even in latency tests.

6.2.1. Optimizing the MTU for Your Network

There are two important maximum transmission units (MTUs): the network interface card (NIC) MTU and the SDN overlay's MTU.

The NIC MTU must be less than or equal to the maximum supported value of the NIC of your network. If you are optimizing for throughput, pick the largest possible value. If you are optimizing for lowest latency, pick a lower value.

The SDN overlay's MTU must be less than the NIC MTU by 50 bytes at a minimum. This accounts for the SDN overlay header. So, on a normal ethernet network, set this to 1450. On a jumbo frame ethernet network, set this to 8950.

NOTE

This 50 byte overlay header is relevant to the OpenShift SDN. Other SDN solutions might require the value to be more or less.

To configure the MTU, edit the node configuration file at */etc/origin/node/node-config.yaml*, and edit the following:

```
networkConfig:
  mtu: 1450          1
  networkPluginName: "redhat/openshift-ovs-subnet"     2
```

1 Maximum transmission unit (MTU) for the pod overlay network.

2 Set to **redhat/openshift-ovs-subnet** for the **ovs-subnet** plug-in, **redhat/openshift-ovs-multitenant** for the **ovs-multitenant** plug-in, or **redhat/openshift-ovs-networkpolicy** for the **ovs-networkpolicy** plug-in. This can also be set to any other CNI-compatible plug-in as well.

6.3. CONFIGURING NETWORK SUBNETS

OpenShift Container Platform provides IP address management for both pods and services. The default values allow for:

- Maximum cluster size of 1024 nodes

- Each of the 1024 nodes has a /23 allocated to it (510 usable IP addresses for pods)

- Provides 65,536 IP addresses for services.

Under most circumstances, these networks cannot be changed after deployment. Thus it is important to plan ahead for growth.

Restrictions for resizing networks are document here: Configuring SDN documentation.

If you would like to plan for a larger environment, here are some example values to consider adding to the **[OSE3:vars]** section in your Ansible inventory file:

```
[OSE3:vars]
osm_cluster_network_cidr=10.128.0.0/10
```

This will allow for 8192 nodes, each with 510 usable IP addresses.

See the supportability limits in the OpenShift Container Platform documentation for node/pod limits for the version of software you are installing.

6.4. OPTIMIZING IPSEC

Because encrypting and decrypting node hosts uses CPU power, performance is affected both in throughput and CPU usage on the nodes when encryption is enabled, regardless of the IP security system being used.

IPSec encrypts traffic at the IP payload level, before it hits the NIC, protecting fields that would otherwise be used for NIC offloading. This means that some NIC acceleration features may not be usable when IPSec is enabled and will lead to increased throughput and CPU usage.

CHAPTER 7. SCALING CLUSTER METRICS

7.1. OVERVIEW

OpenShift Container Platform exposes metrics that can be collected and stored in back-ends by Heapster. As an OpenShift Container Platform administrator, you can view containers and components metrics in one user interface. These metrics are also used by horizontal pod autoscalers in order to determine when and how to scale.

This topic provides information for scaling the metrics components.

> **NOTE**
>
> Autoscaling the metrics components, such as Hawkular and Heapster, is not supported by OpenShift Container Platform.

7.2. RECOMMENDATIONS FOR OPENSHIFT CONTAINER PLATFORM VERSION 3.5

- Run metrics pods on dedicated OpenShift Container Platform infrastructure nodes.

- Use persistent storage when configuring metrics. Set **USE_PERSISTENT_STORAGE=true**.

- Keep the **METRICS_RESOLUTION=30** parameter in OpenShift Container Platform metrics deployments. Using a value lower than the default value of **30** for **METRICS_RESOLUTION** is not recommended. When using the Ansible metrics installation procedure, this is the **openshift_metrics_resolution** parameter.

- Closely monitor OpenShift Container Platform nodes with host metrics pods to detect early capacity shortages (CPU and memory) on the host system. These capacity shortages can cause problems for metrics pods.

- In OpenShift Container Platform version 3.5 testing, test cases up to 25,000 pods were monitored in a OpenShift Container Platform cluster.

7.3. CAPACITY PLANNING FOR CLUSTER METRICS

In tests performed with with 210 and 990 OpenShift Container Platform nodes, where 10500 pods and 11000 pods were monitored respectively, the Cassandra database grew at the speed shown in the table below:

Table 7.1. Cassandra Database storage requirements based on number of nodes/pods in the cluster

Number of Nodes	Number of Pods	Cassandra Storage growth speed	Cassandra storage growth per day	Cassandra storage growth per week
210	10500	500 MB per hour	15 GB	75 GB
990	11000	1 GB per hour	30 GB	210 GB

In the above calculation, approximately 20 percent of the expected size was added as overhead to ensure that the storage requirements do not exceed calculated value.

If the **METRICS_DURATION** and **METRICS_RESOLUTION** values are kept at the default (**7** days and **15** seconds respectively), it is safe to plan Cassandra storage size requirements for week, as in the values above.

> **WARNING**
>
> Because OpenShift Container Platform metrics uses the Cassandra database as a datastore for metrics data, if **USE_PERSISTENT_STORAGE=true** is set during the metrics set up process, **PV** will be on top in the network storage, with NFS as the default. However, using network storage in combination with Cassandra is not recommended, as per the Cassandra documentation.

7.4. SCALING OPENSHIFT CONTAINER PLATFORM METRICS PODS

One set of metrics pods (Cassandra/Hawkular/Heapster) is able to monitor at least 25,000 pods.

CAUTION

Pay attention to system load on nodes where OpenShift Container Platform metrics pods run. Use that information to determine if it is necessary to scale out a number of OpenShift Container Platform metrics pods and spread the load across multiple OpenShift Container Platform nodes. Scaling OpenShift Container Platform metrics heapster pods is not recommended.

7.4.1. Prerequisites

If persistent storage was used to deploy OpenShift Container Platform metrics, then you must create a persistent volume (PV) for the new Cassandra pod to use before you can scale out the number of OpenShift Container Platform metrics Cassandra pods. However, if Cassandra was deployed with dynamically provisioned PVs, then this step is not necessary.

7.4.2. Scaling the Cassandra Components

Cassandra nodes use persistent storage. Therefore, scaling up or down is not possible with replication controllers.

Scaling a Cassandra cluster requires modifying the **openshift_metrics_cassandra_replicas** variable and re-running the deployment. By default, the Cassandra cluster is a single-node cluster.

To scale up the number of OpenShift Container Platform metrics hawkular pods to two replicas, run:

```
# oc scale -n openshift-infra --replicas=2 rc hawkular-metrics
```

Alternatively, update your inventory file and re-run the deployment.

NOTE

If you add a new node to or remove an existing node from a Cassandra cluster, the data stored in the cluster rebalances across the cluster.

To scale down:

1. If remotely accessing the container, run the following for the Cassandra node you want to remove:

   ```
   $ oc exec -it <hawkular-cassandra-pod> nodetool decommission
   ```

   If locally accessing the container, run the following instead:

   ```
   $ oc rsh <hawkular-cassandra-pod> nodetool decommission
   ```

   This command can take a while to run since it copies data across the cluster. You can monitor the decommission progress with **nodetool netstats -H**.

2. Once the previous command succeeds, scale down the **rc** for the Cassandra instance to **0**.

   ```
   # oc scale -n openshift-infra --replicas=0 rc <hawkular-cassandra-
   rc>
   ```

   This will remove the Cassandra pod.

IMPORTANT

If the scale down process completed and the existing Cassandra nodes are functioning as expected, you can also delete the **rc** for this Cassandra instance and its corresponding persistent volume claim (PVC). Deleting the PVC can permanently delete any data associated with this Cassandra instance, so if the scale down did not fully and successfully complete, you will not be able to recover the lost data.

CHAPTER 8. REVISION HISTORY: SCALING AND PERFORMANCE GUIDE

8.1. MON AUG 14 2017

Affected Topic	Description of Change
Routing and Network Optimization	Added the Optimizing the MTU for Your Network section.

8.2. MON JUN 19 2017

Affected Topic	Description of Change
Routing and Network Optimization	Added the Optimizing IPSec section.

8.3. TUE JUN 13 2017

Affected Topic	Description of Change
Recommended Host Practices	Removed recommendation that the local etcd instance be listed first in master configuration.

8.4. TUE MAY 02 2017

Affected Topic	Description of Change
Recommended Host Practices	Included a section on setting max pods per node.

8.5. WED APR 12 2017

OpenShift Container Platform 3.5 Initial Release

Affected Topic	Description of Change
All topics	New guide on recommended practices for building, scaling, and tuning OpenShift Container Platform clusters.