

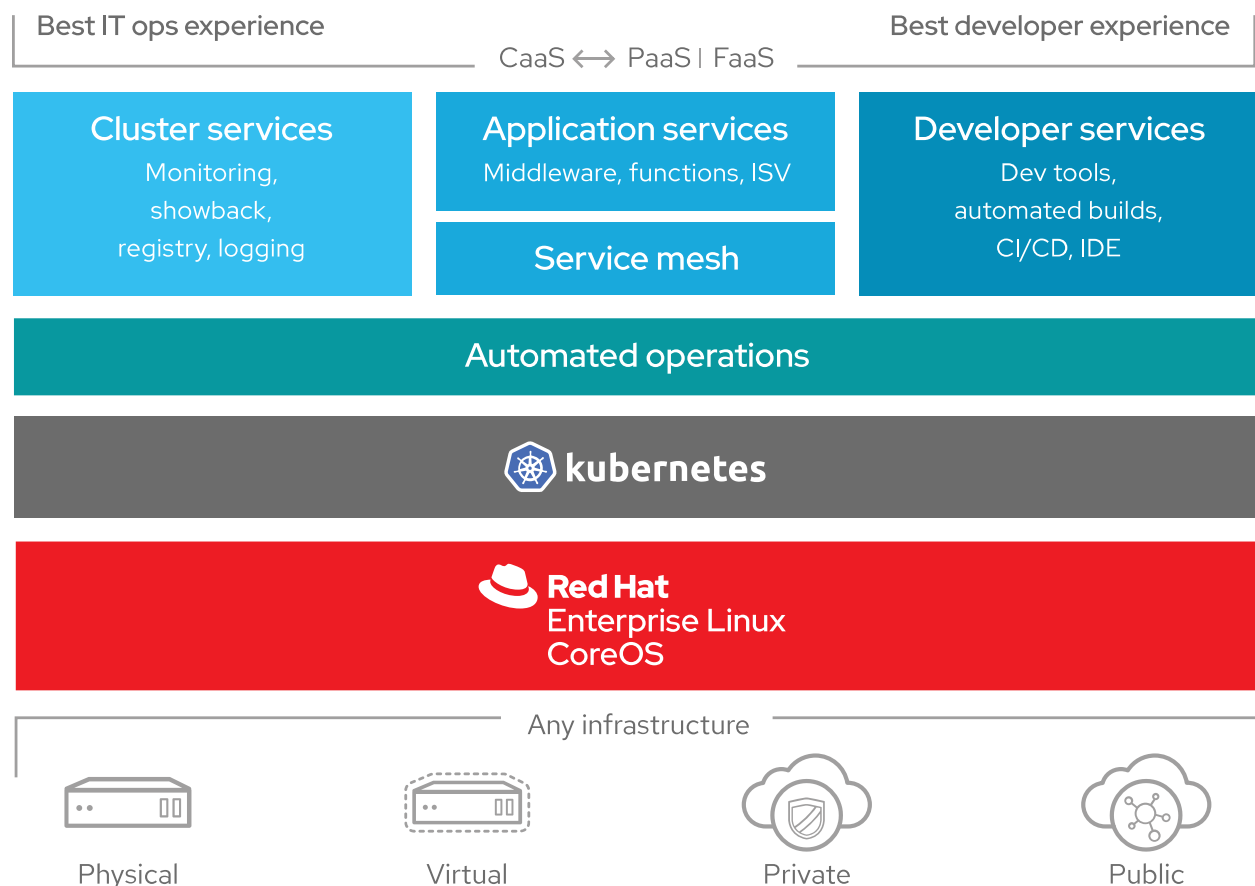
OpenShift

Red Hat® OpenShift® is a hybrid cloud, enterprise Kubernetes application platform.

OpenShift is a cloud development Platform as a Service (PaaS) developed by Red Hat. It is an open source development platform, which enables the developers to develop and deploy their applications on cloud infrastructure. It is very helpful in developing cloud-enabled services.

Openshift origin is built around a core of Docker container, using Kubernetes for cluster containers orchestration.

Openshift origin includes also a functional Web application and a CLI interface to build up and manage your applications.



Basic Elements

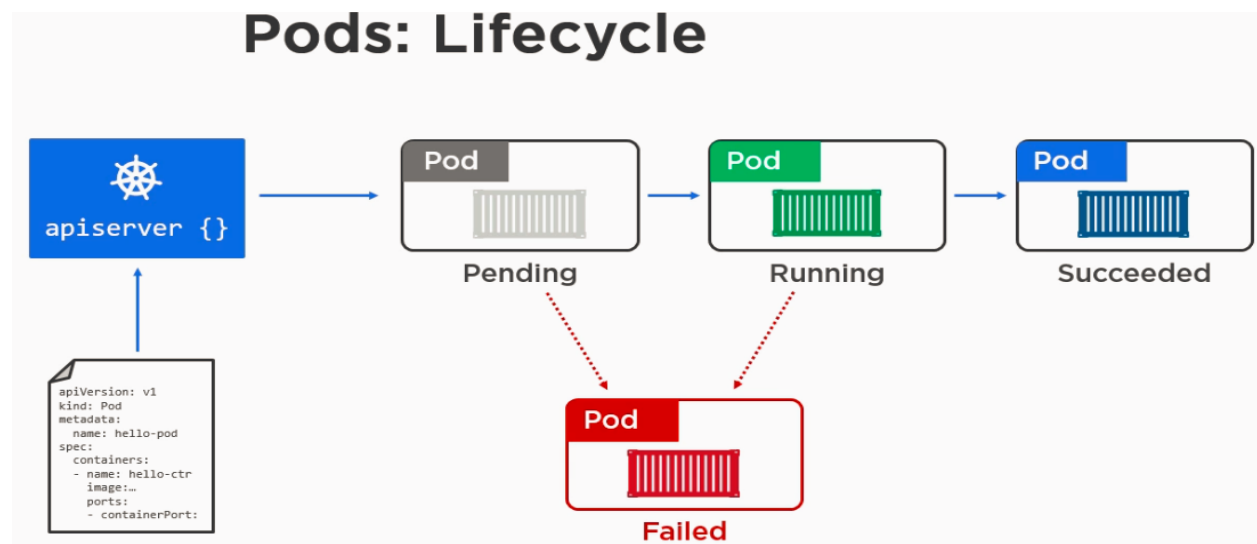
Here are some basic elements you should know:

- **Cluster** – Represents and manages Nodes. Nodes are physical or virtual server(s).
- **Node** – A node is a worker machine in Kubernetes, previously known as a minion. A node may be a VM or physical machine, depending on the cluster.
- **Pod** – A Pod is the basic building block of Kubernetes, the smallest and simplest unit in the Kubernetes object model that you create or deploy. A Pod represents a running process on your cluster.
- **Service** – A Kubernetes Service is an abstraction that defines a logical set of Pods and a policy by which to access them, which is sometimes called a micro-service. The set of Pods targeted by a Service is (usually) determined by a Label Selector. (See below for why you might want a Service without a selector).
- **Controllers** – Abstract elements that interact with basic elements (Jobs, Deployment, etc.)

Pods

Pod can be defined as a collection of container and its storage inside a node of OpenShift (Kubernetes) cluster.

In general, we have two types of pod starting from a single container pod to multi-container pod.



Namespaces

A Kubernetes namespace provides a mechanism to scope resources in a cluster. In OpenShift, a **project** is a Kubernetes namespace with additional annotations.

Namespaces provide a unique scope for:

- Named resources to avoid basic naming collisions.
- Delegated management authority to trusted users.
- The ability to limit community resource consumption.

Each project scopes its own set of:

Objects	Pods, services, replication controllers, etc.
Policies	Rules for which users can or cannot perform actions on objects.
Constraints	Quotas for each kind of object that can be limited.
Service accounts	Service accounts act automatically with designated access to objects in the project.

Replication controllers

A **replication controller** ensures that a specified number of replicas of a pod are running at all times. If pods exit or are deleted, the replication controller acts to instantiate more up to the defined number. Likewise, if there are more running than desired, it deletes as many as necessary to match the defined amount.

A replication controller configuration consists of:

1. The number of replicas desired (which can be adjusted at runtime).
2. A pod definition to use when creating a replicated pod.
3. A selector for identifying managed pods.

Builds

In OpenShift, build is a process of transforming images into containers. It is the processing which converts the source code to an image. This build process works on pre-defined strategy of building source code to image.

The build processes multiple strategies and sources.

Build Strategies

- **Source to Image** – This is basically a tool, which helps in building reproducible images. These images are always in a ready stage to run using the Docker run command.
- **Docker Build** – This is the process in which the images are built using Docker file by running simple Docker build command.
- **Custom Build** – These are the builds which are used for creating base Docker images.

Routes

In OpenShift, routing is a method of exposing the service to the external world by creating and configuring externally reachable hostname. Routes and endpoints are used to expose the service to the external world, from where the user can use the name connectivity (DNS) to access defined application.

In OpenShift, routes are created by using routers which are deployed by OpenShift admin on the cluster. Routers are used to bind HTTP (80) and https (443) ports to external applications.

Following are the different kinds of protocol supported by routes –

- HTTP
- HTTPS
- TLS and web socket

Templates

Templates are defined as a standard object in OpenShift which can be used multiple times. It is parameterized with a list of placeholders which are used to create multiple objects. This can be used to create anything, starting from a pod

to networking, for which users have authorization to create. A list of objects can be created, if the template from CLI or GUI interface in the image is uploaded to the project directory.

Authentication

In OpenShift, while configuring master and client structure, master comes up with an inbuilt feature of OAuth server. OAuth server is used for generating tokens, which is used for authentication to the API.

Since, OAuth comes as a default setup for master, we have the Allow All identity provider used by default. Different identity providers are present which can be configured at **/etc/openshift/master/master-config.yaml**.

There are different types of identity providers present in OAuth.

- Allow All
- Deny All
- HTTPasswd
- LDAP
- Basic Authentication

Authorization

Authorization is a feature of OpenShift master, which is used to validate for validating a user. This means that it checks the user who is trying to perform an action to see if the user is authorized to perform that action on a given project. This helps the administrator to control access on the projects.

Authorization policies are controlled using –

- Rules
- Roles
- Bindings

Evaluation of authorization is done using –

- Identity
- Action
- Bindings

Using Policies –

- Cluster policy
- Local policy

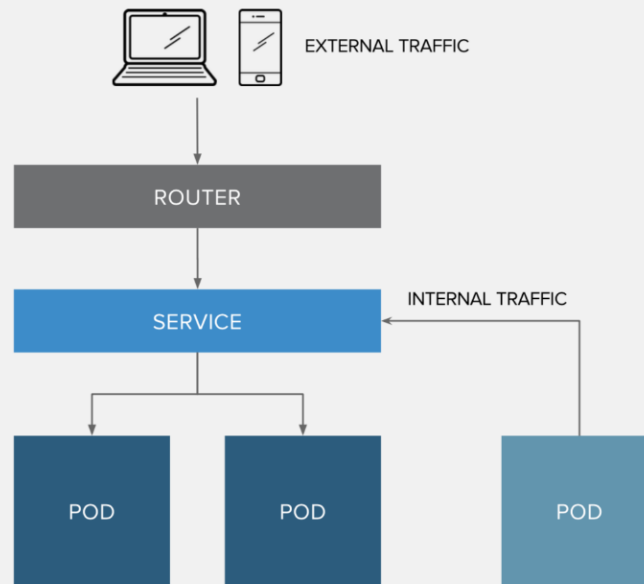
Service –. It's an abstracted layer on top of the pod, which provides a single IP and DNS name through which pods can be accessed. Service helps in managing the load balancing configuration and to scale the pod very easily.

Load Balancing:

Load Balancing functionality is built into OpenShift by default. OpenShift provides its load balancing through its concept of service abstraction.

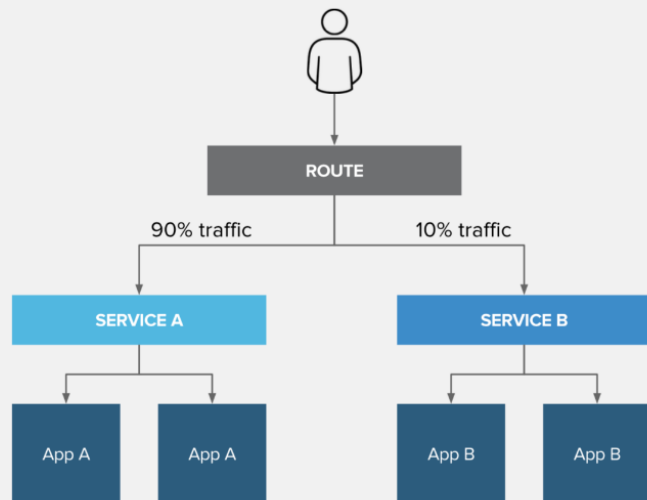
All incoming traffic first comes through a Router, which is responsible for exposing the service externally for users.

ROUTE EXPOSES SERVICES EXTERNALLY



ROUTE SPLIT TRAFFIC

Split Traffic Between Multiple Services For A/B Testing, Blue/Green and Canary Deployments



OpenShift Container Registry

OpenShift Container Platform includes the *OpenShift Container Registry*, a private registry that runs integrated with the platform that you can use to manage your container images.

The OpenShift Container Registry provides role-based access controls that allow you to manage who can pull and push which container images.

OpenShift Container Platform also supports integration with other private registries you may already be using.

Persistent Volume Plug-ins

Containers are useful for both stateless and stateful applications. Protecting attached storage is a key element of securing stateful services.

A **PersistentVolume** (PV) object represents a piece of existing networked storage in the cluster that has been provisioned by an administrator. It is a resource in the cluster just like a node is a cluster resource. PVs are volume plug-ins like **Volumes**, but have a lifecycle independent of any individual **pod** that uses the PV.

Types of Persistent Volumes

OpenShift Container Platform supports the following **PersistentVolume** plug-ins:

- [NFS](#)
- HostPath
- [GlusterFS](#)
- [OpenShift Container Storage \(OCS\) File](#)
- [OpenShift Container Storage \(OCS\) Block](#)
- [Ceph RBD](#)

- [OpenStack Cinder](#)
- [AWS Elastic Block Store \(EBS\)](#)
- [GCE Persistent Disk](#)
- [iSCSI](#)
- [Fibre Channel](#)
- [Azure Disk](#)
- [Azure File](#)
- [VMWare vSphere](#)
- [Local](#)

The access modes are:

Access Mode	CLI Abbreviation	Description
ReadWriteOnce	RWO	The volume can be mounted as read-write by a single node.
ReadOnlyMany	ROX	The volume can be mounted read-only by many nodes.
ReadWriteMany	RWX	The volume can be mounted as read-write by many nodes.

Note: A volume can only be mounted using one access mode at a time, even if it supports many.

Recycling Policy

The current recycling policies are:

Recycling Policy	Description
Retain	Manual reclamation
Recycle	Basic scrub (e.g, <code>rm -rf /<volume>/*</code>)

Note: Currently, NFS and HostPath support recycling.

Limits for Projects:

You can place limits within your OpenShift cluster using ResourceQuotas and LimitRanges. These quotas and limits allow you to control pod and container limits, object counts, and compute resources.

Note: Currently, these limits and quotas only apply to projects and not to users.

Creating a quota in a project to limit the number of pods

To create a quota in the "awesomeproject" that limits the number of pods that can be created to a maximum of 10:

1. Create a ***resource-quota.yaml*** file with the following contents:

2. `apiVersion: v1`
3. `kind: ResourceQuota`
4. `metadata:`

5. name: compute-resources

6. spec:

7. hard:

```
  pods: "10"
```

8. Create the quota using the file you just wrote to apply it to the "awesomeproject":

```
$ oc create -f resource-quota.yaml -n awesomeproject
```

1. After the quota has been in effect for a little while, you can view the usage statistics for the hard limit set on pods.
2. If required, list the quotas defined in the project to see the names of all defined quotas:

```
3. $ oc get quota -n awesomeproject
```

```
4. NAME          AGE
```

```
resource-quota   39m
```

5. Describe the resource quota for which you want statistics:

```
6. $ oc describe quota resource-quota -n awesomeproject
```

```
7. Name:                resource-quota
```

```
8. Namespace:           awesomeproject
```

```
9. Resource      Used  Hard
```

```
10.-----
```

```
    pods      3      10
```

To delete Quota:

```
$ oc delete quota <quota_name>
```

Configuration Options

The [procedure above](#) is just a basic example. The following are references to all the available options for limits and quotas:

This **LimitRange** example explains all the [container limits](#) and [pod limits](#) that you can place within your project:

Example 3.1. Limit Range Object Definition

```
apiVersion: "v1"

kind: "LimitRange"

metadata:

  name: "core-resource-limits" 1
spec:
  limits:
    - type: "Pod"
      max:
        cpu: "2" 2
        memory: "1Gi" 3
      min:
        cpu: "200m" 4
        memory: "6Mi" 5
    - type: "Container"
```

```
max:
  cpu: "2" 6
  memory: "1Gi" 7
min:
  cpu: "100m" 8
  memory: "4Mi" 9
default:
  cpu: "300m" 10
  memory: "200Mi" 11
defaultRequest:
  cpu: "200m" 12
  memory: "100Mi" 13
maxLimitRequestRatio:
  cpu: "10" 14
```

1

The name of the limit range object.

2

The maximum amount of CPU that a pod can request on a node across all containers.

3

The maximum amount of memory that a pod can request on a node across all containers.

4

The minimum amount of CPU that a pod can request on a node across all containers.

5

The minimum amount of memory that a pod can request on a node across all containers.

6

The maximum amount of CPU that a single container in a pod can request.

7

The maximum amount of memory that a single container in a pod can request.

8

The minimum amount of CPU that a single container in a pod can request.

9

The minimum amount of memory that a single container in a pod can request.

10

The default amount of CPU that a container will be limited to use if not specified.

11

The default amount of memory that a container will be limited to use if not specified.

12

The default amount of CPU that a container will request to use if not specified.

13

The default amount of memory that a container will request to use if not specified.

14

The maximum amount of CPU burst that a container can make as a ratio of its limit over request.

To check the amount of used resources (CPU utilization..etc):

Oc get quota

```
oc describe quota <quota-name>
```

Determining Which Roles Users Get by Default

When a user first logs in, there is a default set of permissions that is applied to that user. The scope of permissions that a user can have is controlled by the various types of roles within OpenShift:

- **ClusterRoles**
- **ClusterRoleBindings**
- **Roles** (project-scoped)
- **RoleBindings** (project-scoped)

To view a list of all users that are bound to the project and their roles:

Oc get role bindings

Currently, by default the `system:authenticated` and `system:authenticated:oauth` groups receive the following roles:

Role	Description
shared-resource-viewer	For the <code>openshift</code> project. Allows users to see templates and pull images.
basic-user	For the the entire cluster. Allows users to see their own account, check for information about requesting projects, see which projects they can view, and check their own permissions.
self-provisioner	Allows users to request projects.
system:oauth-token-deleter	Allows users to delete any oauth token for which they know the details.
cluster-status	Allows users to see which APIs are enabled, and basic API server information such as versions.

Role	Description
system:webhook	Allows users to hit the webhooks for a build if they have enough additional information.

To view a list of users and what they have access to across the entire cluster:

```
oc get clusterrolebindings
```

Node Problem Detector:

The Node Problem Detector can detect:

- container runtime issues:
 - unresponsive runtime daemons
- hardware issues:
 - bad CPU
 - bad memory
 - bad disk
- kernel issues:
 - kernel deadlock conditions
 - corrupted file systems
 - unresponsive runtime daemons
- infrastructure daemon issues:

- NTP service outages

```
oc get node <node> -o yaml | grep -B5 KernelDeadlock
```

```
oc get node <node> -o yaml
```

Diagnostics Tool

The **oc adm diagnostics** command runs a series of checks for error conditions in the host or cluster. Specifically, it:

- Verifies that the default registry and router are running and correctly configured.
- Checks **ClusterRoleBindings** and **ClusterRoles** for consistency with base policy.
- Checks that all of the client configuration contexts are valid and can be connected to.
- Checks that SkyDNS is working properly and the pods have SDN connectivity.
- Validates master and node configuration on the host.
- Checks that nodes are running and available.
- Analyzes host logs for known errors.
- Checks that systemd units are configured as expected for the host.
- To use the diagnostics tool, preferably on a master host and as cluster administrator, run:

```
# oc adm diagnostics
```

Available diagnostics include:

Diagnostic Name	Purpose
AggregatedLogging	Check the aggregated logging integration for proper configuration and operation.
AnalyzeLogs	Check systemd service logs for problems. Does not require a configuration file to check against.
ClusterRegistry	Check that the cluster has a working container image registry for builds and image streams.
ClusterRoleBindings	Check that the default cluster role bindings are present and contain the expected subjects according to base policy.
ClusterRoles	Check that cluster roles are present and contain the expected permissions according to base policy.
ClusterRouter	Check for a working default router in the cluster.
ConfigContexts	Check that each context in the client configuration is complete and has connectivity to its API server.
DiagnosticPod	Creates a pod that runs diagnostics from an application standpoint, which checks that DNS within the pod is working as expected and the credentials for the default service account authenticate correctly to the master API.
EtcdWriteVolume	Check the volume of writes against etcd for a time period and classify them by operation and key. This diagnostic only runs if specifically requested, because it does not run as quickly as other diagnostics and can increase load on etcd.
MasterConfigCheck	Check this host's master configuration file for problems.
MasterNode	Check that the master running on this host is also running a node to verify that it is a member of the cluster SDN.

Diagnostic Name	Purpose
MetricsApiProxy	Check that the integrated Heapster metrics can be reached via the cluster API proxy.
NetworkCheck	<p>Create diagnostic pods on multiple nodes to diagnose common network issues from an application or pod standpoint. Run this diagnostic when the master can schedule pods on nodes, but the pods have connection issues. This check confirms that pods can connect to services, other pods, and the external network.</p> <p>If there are any errors, this diagnostic stores results and retrieved files in a local directory (<i>/tmp/openshift/</i>, by default) for further analysis. The directory can be specified with the --network-logdir flag.</p>
NodeConfigCheck	Checks this host's node configuration file for problems.
NodeDefinitions	Check that the nodes defined in the master API are ready and can schedule pods.
RouteCertificateValidation	Check all route certificates for those that might be rejected by extended validation.
ServiceExternalIPs	Check for existing services that specify external IPs, which are disallowed according to master configuration.
UnitStatus	Check systemd status for units on this host related to OKD. Does not require a configuration file to check against.

Limit Range:

A limit range, defined by a **LimitRange** object, enumerates compute resource constraints in a project at the pod, container, image, image stream, and persistent volume claim level, and specifies the amount of resources that a pod, container, image, image stream, or persistent volume claim can consume.

Note: For CPU and Memory limits, if you specify a **max** value, but do not specify a **min** limit, the resource can consume CPU/memory resources greater than **max** value`.

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "core-resource-limits" 1
spec:
  limits:
    - type: "Pod"
      max:
        cpu: "2" 2
        memory: "1Gi" 3
      min:
        cpu: "200m" 4
        memory: "6Mi" 5
    - type: "Container"
      max:
        cpu: "2" 6
        memory: "1Gi" 7
      min:
        cpu: "100m" 8
        memory: "4Mi" 9
      default:
        cpu: "300m" 10
        memory: "200Mi" 11
      defaultRequest:
        cpu: "200m" 12
        memory: "100Mi" 13
      maxLimitRequestRatio:
        cpu: "10" 14
```

-
- 1 The name of the limit range object.
 - 2 The maximum amount of CPU that a pod can request on a node across all containers.
 - 3 The maximum amount of memory that a pod can request on a node across all containers.
 - 4 The minimum amount of CPU that a pod can request on a node across all containers. Not setting a `min` value or setting `0` is unlimited allowing the pod to consume more than the `max` value.
 - 5 The minimum amount of memory that a pod can request on a node across all containers. Not setting a `min` value or setting `0` is unlimited allowing the pod to consume more than the `max` value.
 - 6 The maximum amount of CPU that a single container in a pod can request.
 - 7 The maximum amount of memory that a single container in a pod can request.
 - 8 The minimum amount of CPU that a single container in a pod can request. Not setting a `min` value or setting `0` is unlimited allowing the pod to consume more than the `max` value.
 - 9 The minimum amount of memory that a single container in a pod can request. Not setting a `min` value or setting `0` is unlimited allowing the pod to consume more than the `max` value.
 - 10 The default amount of CPU that a container will be limited to use if not specified.
 - 11 The default amount of memory that a container will be limited to use if not specified.
 - 12 The default amount of CPU that a container will request to use if not specified.
 - 13 The default amount of memory that a container will request to use if not specified.
 - 14 The maximum amount of CPU burst that a container can make as a ratio of its limit over request.

Garbage Collection:

The OpenShift Container Platform node performs two types of garbage collection:

- Container garbage collection: Removes terminated containers.
- Image garbage collection: Removes images not referenced by any running pods.

Container Garbage Collection

Container garbage collection is enabled by default and happens automatically in response to eviction thresholds being reached.

The node tries to keep any container for any pod accessible from the API. If the pod has been deleted, the containers will be as well.

Containers are preserved as long the pod is not deleted and the eviction threshold is not reached.

If the node is under disk pressure, it will remove containers and their logs will no longer be accessible via **oc logs**

The policy for container garbage collection is based on three node settings:

Setting	Description
minimum-container-ttl-duration	The minimum age that a container is eligible for garbage collection. The default is 0 . Use 0 for no limit. Values for this setting can be specified using unit suffixes such as h for hour, m for minutes, s for seconds.
maximum-dead-containers-per-container	The number of instances to retain per pod container. The default is 1 .
maximum-dead-containers	The maximum number of total dead containers in the node. The default is -1 , which means unlimited.

Note: The **maximum-dead-containers** setting takes precedence over the **maximum-dead-containers-per-container** setting when there is a conflict.

Container Garbage Collection Settings

kubeletArguments:

minimum-container-ttl-duration:

- **"10s"**

maximum-dead-containers-per-container:

- "2"

maximum-dead-containers:

- "240"

Detecting Containers for Deletion

Each spin of the garbage collector loop goes through the following steps:

1. Retrieve a list of available containers.
2. Filter out all containers that are running or are not alive longer than the **minimum-container-ttl-duration** parameter.
3. Classify all remaining containers into equivalence classes based on pod and image name membership.
4. Remove all unidentified containers (containers that are managed by kubelet but their name is malformed).
5. For each class that contains more containers than the **maximum-dead-containers-per-container** parameter, sort containers in the class by creation time.
6. Start removing containers from the oldest first until the **maximum-dead-containers-per-container** parameter is met.
7. If there are still more containers in the list than the **maximum-dead-containers** parameter, the collector starts removing containers from each class so the number of containers in each one is not greater than the average number of containers per class, or **<all_remaining_containers>/<number_of_classes>**.
8. If this is still not enough, sort all containers in the list and start removing containers from the oldest first until the **maximum-dead-containers** criterion is met.

Image Garbage Collection

Image garbage collection relies on disk usage as reported by **cAdvisor** on the node to decide which images to remove from the node. It takes the following settings into consideration:

Setting	Description
image-gc-high-threshold	The percent of disk usage (expressed as an integer) which triggers image garbage collection. The default is 85 .
image-gc-low-threshold	The percent of disk usage (expressed as an integer) to which image garbage collection attempts to free. Default is 80 .

Image Garbage Collection Settings

kubeletArguments:

image-gc-high-threshold:

- "85"

image-gc-low-threshold:

- "80"

Detecting Images for Deletion

Two lists of images are retrieved in each garbage collector run:

1. A list of images currently running in at least one pod
2. A list of images available on a host

Verify the Application is Running


If your DNS is correctly configured, then your new application can be accessed using a web browser. If you cannot access your application, then speak with your system administrator.

To view your new application:

Configuring Automated Builds

You forked the source code for this application from the OpenShift Container Platform GitHub repository. Therefore, you can use a webhook to automatically trigger a rebuild of your application whenever you push code changes to your forked repository.

To set up a webhook for your application:

1. From the Web Console, navigate to the project containing your application.
2. Click the **Browse** tab, then click **Builds**.
3. Click your build name, then click the **Configuration** tab.
4. Click  next to **GitHub webhook URL** to copy your webhook payload URL.
5. Navigate to your forked repository on GitHub, then click **Settings**.
6. Click **Webhooks & Services**.
7. Click **Add webhook**.
8. Paste your webhook URL into the **Payload URL** field.
9. Click **Add webhook** to save.

GitHub now attempts to send a ping payload to your OpenShift Container Platform server to ensure that communication is successful. If you see a green check mark appear next to your webhook URL, then it is correctly configured. Hover your mouse over the check mark to see the status of the last delivery.

The next time you push a code change to your forked repository, your application will automatically rebuild.

Network Namespaces

OpenShift Container Platform uses software-defined networking (SDN) to provide a unified cluster network that enables communication between containers across the cluster.

Using network namespaces, you can isolate pod networks.

Each pod gets its own IP and port range to bind to, thereby isolating pod networks from each other on the node.

Pods from different projects cannot send packets to or receive packets from pods and services of a different project. You can use this to isolate developer, test and production environments within a cluster.

SDN Network Plugins

- OpenShift Container Platform supports the Kubernetes **Container Network Interface (CNI)** as the interface between the OpenShift Container Platform and Kubernetes.
- Software defined network (SDN) plug-ins match network capabilities to your networking needs.
- Additional plug-ins that support the CNI interface can be added as needed.

Flannel SDN

- **flannel** is a virtual networking layer designed specifically for containers. OpenShift Container Platform can use it for networking containers instead of the default software-defined networking (SDN) components.
- This is useful if running OpenShift Container Platform within a cloud provider platform that also relies on SDN, such as OpenStack, and you want to avoid encapsulating packets twice through both platforms.

Other SDN's: NSX-T SDN, Nuage SDN

OpenShift Container Platform Cluster Limits

Limit Type	3.7 Limit	3.9 Limit	3.10 Limit	3.11 Limit
Number of nodes [1]	2,000	2,000	2,000	2,000
Number of pods [2]	120,000	120,000	150,000	150,000
Number of pods per node	250	250	250	250
Number of pods per core	10 is the default value. The maximum supported value is the number of pods per node.	10 is the default value. The maximum supported value is the number of pods per node.	There is no default value. The maximum supported value is the number of pods per node.	There is no default value. The maximum supported value is the number of pods per node.
Number of namespaces	10,000	10,000	10,000	10,000
Number of builds: Pipeline Strategy	N/A	10,000 (Default pod RAM 512Mi)	10,000 (Default pod RAM 512Mi)	10,000 (Default pod RAM 512Mi)
Number of pods per namespace [3]	3,000	3,000	3,000	25,000
Number of services [4]	10,000	10,000	10,000	10,000

Limit Type	3.7 Limit	3.9 Limit	3.10 Limit	3.11 Limit
Number of services per namespace	N/A	N/A	5,000	5,000
Number of back-ends per service	5,000	5,000	5,000	5,000
Number of deployments per namespace [3]	2,000	2,000	2,000	2,000

Planning Your Environment According to Application Requirements

Consider an example application environment:

Pod Type	Pod Quantity	Max Memory	CPU Cores	Persistent Storage
apache	100	500MB	0.5	1GB
node.js	200	1GB	1	1GB
postgresql	100	1GB	2	10GB
JBoss EAP	100	1GB	1	1GB

Note:

1. Clusters with more than the stated limit are not supported. Consider splitting into multiple clusters.
2. 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.
3. There are a number of control loops in the system that need to 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.
4. 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.