8
OpenShift Troubleshooting, Performance and Best Practices

From now on we understand that the concepts explained in *Chapter 5 - OpenShift Deployment* provided the foundation for you to initiate your first contact with an OpenShift cluster. In this chapter, we will give some tips on how to perform a health check on a cluster, dive into some root cause analysis, and also details on how to make a cluster run according to the best practices.

This chapter covers:

- Things that can crash a cluster
- Troubleshooting reference guide - how to start?
- Understanding misleading error messages.

Things that can crash a cluster

Every time we start learning about some technology it is common to be twice as careful with the installation, configuration, or adjustment to be as thorough as possible. For this, you can go through a specialized reading or also leave through trial and error within the documentation that is not always very clear.

OpenShift is a great and disrupting technology, but you will navigate thru a puzzle of different aspects related to storage, compute, network, and others. Obviously, in the official documentation or even in quick internet searches you will find commands to start from scratch, but in many situations, even with the necessary command and parameters, it is difficult to come from troubleshooting to its solution.

Currently, OpenShift has an automatic recovery system, but this is usually not enough to ensure a stable environment. For this self-healing to take place successfully, many prerequisites need to be checked on the cluster first. So before we understand what can potentially crash, let's understand how this self-adjustment mechanism works.

Operators

In the world of technology, there are many roles played and some of them are linked to the administration of the infrastructure. There are several names for this role, the most common still being the *sysadmin*, who operates the servers and services of an IT infrastructure. Likewise, OpenShift has *operators* that are nothing more than **applications designed to monitor platform behavior and maintain an operation**.

How do the operators work? Operators are assigned to fulfill a single task of maintaining the application and all its components according to a standard. Understand that operators are not the same for all applications, that is, operators are unique, each one with its own parameter definitions, configurations that are required, optional, and others.

The operator parameters contract is described in the **CRD - Custom Resource Definition**. CRD is a definition that extends a Kubernetes API functionality, giving more flexibility to the cluster to store a collection of objects of a certain type. Once a CRD is defined, you can create a **CR - Custom Resource** that will allow you to add their custom API into the cluster.

Operators are a tool for keeping a cluster or application healthy, so why should we care about learning about OpenShift troubleshooting if it fixes itself? Indeed, operators are a powerful tool, but as we mentioned earlier, OpenShift is a big puzzle and the pieces need to fit together perfectly for it to work properly. Although it is reigned by Operators that are somehow prepared to maintain its integrity, failures can occur and the role of the cluster-admin and their experience in solving problems will help keep these operators all healthy.

In the next sections, we'll go deeper into the main components of OpenShift and what aspects to be concerned about.

Etcd

Etcd is a distributed key-value service responsible for storing the state of an OpenShift cluster. Through it, all objects contained in the cluster are shown in a key-value format, so it is important to consider at least 3 (three) important factors in this service, which is the heart of the control plane's operation.

1. Etcd is **highly sensitive** to infrastructure's **latency** and **bandwidth**.
2. Etcd needs to be distributed on all Masters nodes, that is, to be high available an OpenShift cluster infrastructure demands this service be distributed on 3 Master nodes.
3. Unlike many high availability services, in which you have a main and a secondary server, with etcd this concept is based on member **quorum** and **leadership**.

Red Hat made the etcd complexity easier by establishing the number of Master nodes to be 3 as default and also using a cluster operator that manages etcd and reports any issue in it, however, even though you must understand how etcd works to be able to troubleshoot if any complex issue occurs. Go ahead to learn how the quorum and leader-based etcd algorithm works.

How do the quorum and leader-based scheme work?

An etcd cluster works on the concept of **leader** and **followers**, which is known as **Raft Distributed Consensus** protocol. This protocol implements an algorithm based on a *leader election* to establish a distributed consensus among all members of an etcd cluster. Once members are added to an etcd cluster and a leader is elected, the process only requires sending periodic heartbeats to confirm that the leader still responds within a suitable latency time.

In case of an unanswered heartbeat time frame, the members start a new election to guarantee the cluster resilience, self-healing, and continuity of service.

It is recommended that an etcd cluster has an odd number of nodes so that the following formula guarantees the tolerance of a given number of failing members, to this we give the name of **quorum**:

Quorum = (n/2)+1, where "n" represents the number of members.

A cluster must always have at least the quorum number of members working to be working properly. For the sake of clarity, let's check out some scenarios:

Scenario 1: 3-member cluster, all up and running:

Comentado [RP1]: Man, a formula é essa que vc mesmo colocou aqui, nao entendi o que esta errado no cluster, vamos abrir um meet hoje a noite..?

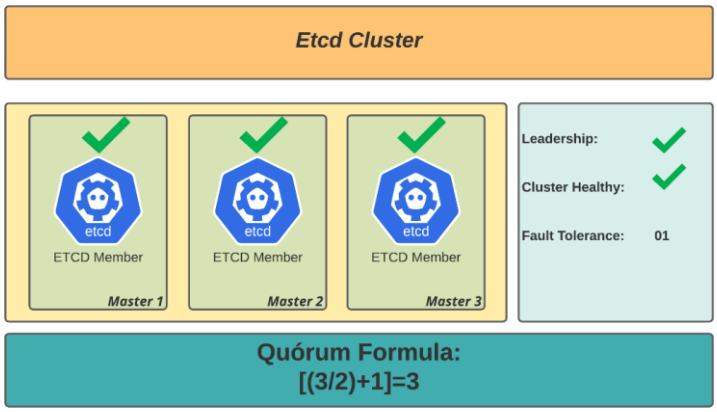
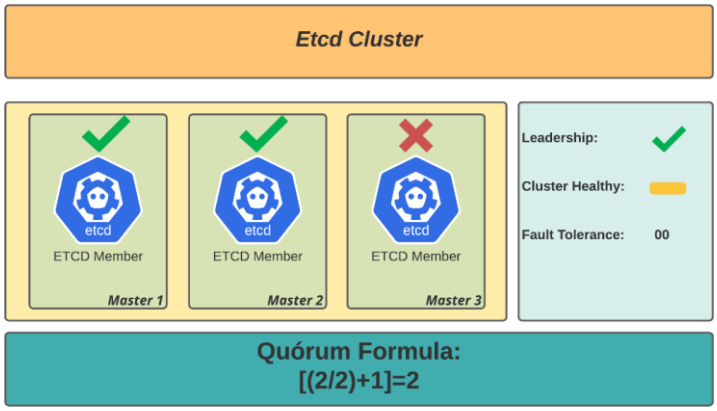


Figure 8.1 - Healthy Etcd Cluster (3-node member health)

Analysis: Quorum is ok as there are a majority of working members, leadership assured, so the cluster is healthy.

Scenario 2: 3-member cluster, 2 (two) members working:



Comentado [GF2R1]: a formula sim pecorito, esta correta... o q esta errado é o resultado variar nos 3 centarios... sempre vai ser igual a 2 pq em todos os cenarios ha 3 membros, ou seja:

Quorum = (n/2)+1, where "n" represents the number of members.

Nos 3 cenários n =3, entao nos 3 cenarios o resultado é:

Quorum = (n/2)+1 = (3/2) + 1 = 2 (o arredondamento é sempre pra baixo)

Comentado [GF3R1]: o q muda é apenas a tolerancia a falhas nos 3 cenarios... entende?

Comentado [GF4]: Pecorito, tem um probleminha conceitual aqui com a formula... um etcd com 3 membros sempre vai ter o quorum = 2 ((3/2) + 1 = 2), isso significa que e possivel perder ate um node e ainda se mantem o quorum... o calculo do quorum em si nao muda conforme os membros estarem saudaveis ou nao... o que muda e a resiliencia, que voce colocou ali como “fault tolerance”... mas a formula de quorum nao muda conforme o numero de membros do cluster, estejam eles funcionando ou nao... entao na verdade a formula q esta em verde vai ser sempre 2 em todos os diagramas.

Comentado [RP5R4]: perai.... ali nao estou alterando a fórmula... estou demonstrando o calculo... olha na primeira figura temos um cluster com 3 etcds... logo o "n" é igual a 3.... já na segunda temos apenas 2 etcds entao o "n" é igual a 2.... só estou aplicando a formula... nao alterei a formula em nenhum momento...]

Comentado [GF6R4]: nao nao... nao é isso q eu quis dizer... o cluster em si em todos os diagramas tem 3 membros (3-member cluster), apenas o numero de membros q esta online q muda... por isso q eu comentei q ...

Comentado [GF7R4]: se ainda houver duvida nessa questao me manda msg q eu te explico melhor...

Comentado [RP8R4]: eu entendi man... a questao é que só estou aplicando a formula...nenhum momento mudei a quorum formula...coloquei um comentario mais acima, mas se ainda assim achar que estamos divergentes eu arranco ...

Comentado [GF9R4]: essa hj estou bem corrido por conta do SKO... acho q consigo amanha... vou procurar na sua agenda e mandar um invite entao, ok? ...

Comentado [GF10]: Sempre coloque legenda nas imagens

Comentado [RP11R10]: feito

Figure 8.2 - Healthy Etcd Cluster (2-node member health, risk of outage)

Analysis: Quorum: Ok as there are a majority of working members, leadership assured, degradation risk in case of disruption of one more node, but the cluster is healthy.

Scenario 3: 3-member cluster, 1 (one) member working:

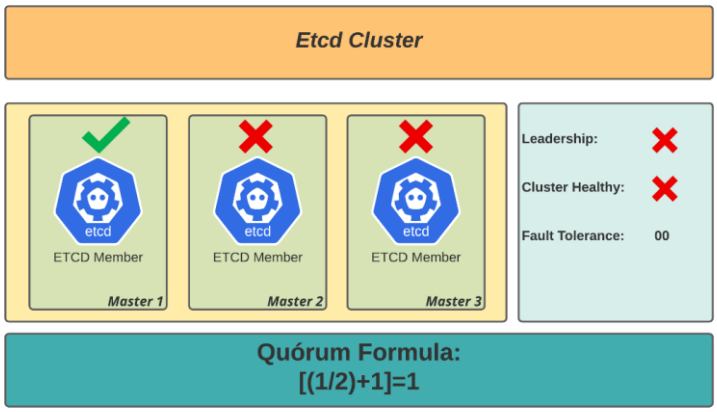


Figure 8.3 - Degraded Etcd Cluster (1-node member health, unhealthy cluster)

Analysis: There is no quorum, as the majority of members are down, so there is no longer possible to elect a new leader and the cluster is degraded.

Troubleshooting Etcd

As mentioned earlier, OpenShift is managed by its operators that can provide standardization, self-healing, and activities metrics, but this is not always enough to keep the cluster fully functional.

Situations such as scenarios 2 and 3 can occur due to different factors related to any infrastructure layer. It is important to carry out an in-depth analysis to restore the cluster to a functional state. Here's an approach to troubleshoot etcd.

For troubleshooting, it is important to consider what kind of disruption the cluster has been hit, if we are still able to use the OpenShift API to access the nodes you can use the first approach. Otherwise, if the API is unavailable you must see the second scenario.

Scenario 01 – Etcd Degraded

In cases when it is still possible to execute **oc** or **kubectl** commands, use the **rsh** command to the etcd pod and perform the following steps thru **etcdctl** is the quickest approach.

An etcd degradation can occur for many different reasons, such as storage or network failures, live migration of the master nodes, or even manipulations in the Operating System which may cause immediate disruption of the cluster.

As mentioned before, run the following commands to open a terminal in an etcd pod and identify the failure:

```
$ oc project openshift-etcd

$ oc get pods -n openshift-etcd | grep -v etcd-quorum-guard |
grep etcd

etcd-ocp-master-0 3/3 Pending 0 14m

etcd-ocp-master-1 3/3 CrashLoopBackOff 6 17m

etcd-ocp-master-3 2/3 Running 0 9m11s
```

Notice that in the previous output 02 (two) of 03 (three) masters are getting issues, so let's *rsh* to master-3, perform a backup, and recreate the etcd nodes.

```
oc rsh etcd-ocp-master-3

Defaulting container name to etcdctl.

Use 'oc describe pod/etcd-ocp-master-3 -n openshift-etcd' to
see all of the containers in this pod.

sh-4.4# etcdctl member list -w table

+------+------+------+------+------+------+

| ID | STATUS | NAME | PEER ADDRS | CLIENT ADDRS | IS LEARNER
+------+------+------+------+------+------+

| 5bdacda4e0f48d91 | failed | ocp-master-1 |
https://192.168.200.235:2380 | https://192.168.200.235:2379 |
false |
```

```
| b50b656cba1b0122 | started | ocp-master-3 | https://192.168.200.14:2380 | https://192.168.200.14:2379 | false |

| cdc9d2f71033600a | failed | ocp-master-0 | https://192.168.200.234:2380 | https://192.168.200.234:2379 | false |

+------+------+------+------+------+------+

sh-4.4# etcdctl endpoint status -w table

+------+------+------+------+------+------+

| ENDPOINT | ID | VERSION | DB SIZE | IS LEADER | IS LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |

+------+------+------+------+------+------+

| https://192.168.200.14:2379 | b50b656cba1b0122 | 3.4.9 | 136 MB | true | false | 281 | 133213554 | 133213554 | |

| https://192.168.200.234:2379 | cdc9d2f71033600a | 3.4.9 | 137 MB | false | false | 281 | 133213554 | 133213554 | |

| https://192.168.200.235:2379 | 5bdacda4e0f48d91 | 3.4.9 | 136 MB | false | false | 281 | 133213554 | 133213554 | |

+------+------+------+------+------+------+
```

Once the IDs of the failed members are identified, the next step is to remove the etcd members, leaving only the one that is running as part of the cluster. To do so, run first the backup of etcd:

```
/usr/local/bin/cluster-backup.sh /home/core/assets/backup
```

Note

The etcd backup will be saved in the home directory of the **core** user in the same node as the etcd pod is running. It is highly recommended you copy it to an off-node location, therefore, you will avoid the risk of losing access to the node and, as such, lose the etcd backup file and put the cluster recovery at risk, which can be nearly incorrigible.

Now that you have already identified the problem with the etcd cluster move to the *how to solve* section and perform the steps there to recover your cluster.

Method 02 – Cluster API down

If the OpenShift API is down, it is important to perform any steps with much more caution to avoid an irreversible loss to the cluster. In such a scenario, you can't use the rsh command, get logs using oc logs, and use any oc or kubectl command, as any of them use the OpenShift API, which makes the troubleshooting and solution much more difficult and complex.

Due to that, there must be regular etcd backups in place before the cluster malfunctions. If there is no previous backup, the first step is to perform a direct backup on the node that is in operation.

- Run :

```
$ ssh -i ~/.ssh/id_rsa core@ocp-master-3
```

- Checkout **etcd** state running **crictl** command:

```
$ sudo crictl | grep -i etcd
```

- Get the **ETCD** pod ID and run the **crictl exec** statement to identify the cluster's node state.

```
$ crictl exec bd077a3f1b211 etcdctl member list -w table

+---+---+---+---+---+---+

| ID | STATUS | NAME | PEER ADDRS | CLIENT ADDRS | IS
LEARNER |

+---+---+---+---+---+---+

| 9e715067705c0f7c | unknown | ocp-master-4 |
https://192.168.200.15:2380 | https://192.168.200.15:2379
| false |

| b50b656cba1b0122 | started | ocp-master-3 |
https://192.168.200.14:2380 | https://192.168.200.14:2379
| false |

| cdc9d2f71033600a | failed  | ocp-master-0 |
https://192.168.200.234:2380 |
https://192.168.200.234:2379 | false |

+---+---+---+---+---+---+
```

- Note that the etcd members are unreachable except for one that is in the started state. Run a backup by going to the node and running the backup command as follows:

```
sudo /usr/local/bin/cluster-backup.sh
/home/core/assets/backup
```

- With the pod id of **etcd**, get by **crictl**, run the command to identify the cluster nodes and their state.

```
$ crictl exec bd077a3f1b211 etcdctl endpoint status -w
table

+-------+-------+-------+-------+-------+-------+

| ENDPOINT | ID | VERSION | DB SIZE | IS LEADER | IS
LEARNER | RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX |
ERRORS |

+-------+-------+-------+-------+-------+-------+

| https://192.168.200.14:2379 | b50b656cba1b0122 | 3.4.9
| 136 MB | true | false | 491 | 133275501 | 133275501 | |

| https://192.168.200.15:2379 | 9e715067705c0f7c | 3.4.9
| 137 MB | false| false | 491 | 133275501 | 133275501 | |

+-------+-------+-------+-------+-------+-------+
```

At this stage it is possible to draw some conclusions, we understand that there is no minimum quorum to keep the cluster up, so the API became unavailable.

How to solve it?

Formerly, on OpenShift version 3, the usual solution for both cases was reestablishing the etcd cluster using a backup. Now with OpenShift 4, it is easier to provision new Master nodes, in your infrastructure though. That said, if your issue is the first scenario (API is still available) and your installation is the Installer-Provisioned Infrastructure (IPI) method, we suggest you take the following steps to recreate the problematic master node, using a new node name.

1. Get the yaml machine descriptor for any current master node by running the command below:

```
oc get machine <master-node> \
    -n openshift-machine-api \
```

Comentado [GF12]: Aqui nao deveriam haver 3 endpoints?

Comentado [RP13R12]: não necessariamente... esse caso eu mesmo fiz num cliente, sao outputs reais.... o que aconteceu... o master-0 já era uma vmmm que tinha morrido, mas para o etcd member list ele existe embora esteja com status de unknown.... ja no endpoint status ele vai até os nodes que tem na lista e traz o resultado... o cluster tava todo f$rrado... entao ele nem conseguia alcançar o endpoint que estava unknown para lista.....saca? até pq a vmm nem existia mais... num ia conseguir chegar mesmo....essa é a sacada, mostrar um caos no etcd para ensinar como resolver de fato!!!

Comentado [GF14R12]: ok... entendido... faz sentido...

```
    -o yaml \
    > new-master-machine.yaml
```

2. The yaml file should look like the following:

new-master-machine.yaml

```
apiVersion: machine.openshift.io/v1beta1
kind: Machine
metadata:
  finalizers:
  - machine.machine.openshift.io
  labels:
    machine.openshift.io/cluster-api-cluster: ocp-sgw5f
(.. omitted ..)
  name: ocp-master-4
  namespace: openshift-machine-api
  selfLink:
/apis/machine.openshift.io/v1beta1/namespaces/openshift-
machine-api/machines/ocp-master-4
spec:
  metadata: {}
  providerSpec:
    value:
      apiVersion: vsphereprovider.openshift.io/v1beta1
      credentialsSecret:
        name: vsphere-cloud-credentials
      diskGiB: 120
      kind: VSphereMachineProviderSpec
(.. omitted ..)
```

3. Make required changes in the yaml file to provision the new master:

a. Remove the following sections or fields:

   i. Entire `status`, `metadata.annotations` and `metadata.generation` sections.

   ii. Delete `metadata.resourceVersion`, `metadata.uid` and `spec.providerId` fields.

b. Change the `metadata.name` field to a new name (e.g.: <clustername>-<clusterid>-master-3)

c. Update also the node name in the `metadata.selfLink` field.

4. Delete the problematic master node using the command below:

```
$ oc delete machine <problematic-master-node-name> -n
openshift-machine-api
```

5. Use the command below to monitor the deletion process and certify it has been deleted:

```
$ oc get machines -n openshift-machine-api -o wide
```

6. As soon as the problematic master has been deleted you can now provision a new one using the yaml we prepared previously. To do so run the following command:

```
oc apply –f new-master-machine.yaml
```

Note

Repeat this procedure if necessary changing only the `metadata.name` and `metadata.selfLink` fields for each problematic master nodes in your cluster.

After the new master nodes have been provisioned, observe the following steps to verify if the etcd cluster is healthy.

1. Check if all etcd pods are running. You must see three pods running:

```
$ oc get pods -n openshift-etcd | grep -v etcd-quorum-
guard | grep etcd
```

2. There are some cases in which the etcd pod is not deployed automatically with the master provisioning. If you don't see three pods running you may run the following command to force the etcd operator to deploy the etcd in the new node.

```
$ oc patch etcd cluster -p='{"spec":
{"forceRedeploymentReason": "recovery-'"$( date --rfc-
3339=ns )"'"}}' --type=merge
```

3. Now let's check from inside the etcd cluster if it is working as expected. To do so run the following command to open a bash inside one of the etcd pods.

```
# Get the name of one etcd pod

$ oc get pods -n openshift-etcd | grep -v etcd-quorum-
guard | grep etcd

$ oc rsh <etcd-pod-name> -n openshift-etcd
```

4. Now check the cluster member list:

```
etcdctl member list -w table
```

5. In some cases, you will still see the problematic etcd node that we already removed still as part of the cluster members. Therefore, if the previous command shows more than three members, use the command below to remove the nonfunctional etcd members:

```
$ etcdctl remove <member-id>
```

ETCD Performance Analysis

Kubernetes clusters are highly sensitive to latency and throughput, due to that some precautions are necessary to have a stable cluster and also with great performance. OpenShift is a platform designed for high availability, and, as such, the expected **Etcd** use and consumption is traffic intensive. It is important, then, to follow some best practices to have a stable cluster. Let's look at some recommended configurations.

Storage

Etcd is disk usage is intensive, so it is recommended to use SSD disks for fast write/read response time. Regarding response times, we could say that 50 sequential IOPS would be a minimum requirement, but from our experience, the OpenShift usage grows really fast, so we recommend you consider disks that can deliver at least 500 concurrent IOPS, to maintain the cluster's health and stability. However, notice that some providers do not publish the sequential IOPS but only the sequential IOPS. In such cases, consider that concurrent IOPS can be 10x than sequential IOPS.

Below is an example of how to measure the performance of the etcd disks using a customized version of the **fio** tool. In the Openshift cluster run the debug command to get access to a master node.

```
$ oc debug node/master1.ocp.hybridcloud.com
```

As soon as the command is executed, the following message will be displayed. Execute the `chroot` command after the shell is to be able to execute commands in privileged mode:

```
Starting pod/ocp-master1hybridcloud-debug ...
 To use host binaries, run `chroot /host`
 chroot /host


Pod IP: 172.19.10.4
 If you don't see a command prompt, try pressing enter.
 sh-4.4# chroot /host
 sh-4.4#
```

Create the container as indicated below. After the etcd-perf container starts it will automatically run the performance checks.

```
sh-4.4# podman run --volume /var/lib/etcd:/var/lib/etcd:Z
quay.io/openshift-scale/etcd-perf Trying to pull
quay.io/openshift-scale/etcd-perf:latest...


Getting image source signatures
(.. omitted ..)

------- Running fio ------{

"fio version" : "fio-3.7",

"timestamp" : 1631814461,

"timestamp_ms" : 1631814461780,

"time" : "Thu Sep 16 17:47:41 2021",

"global options" : {

"rw" : "write",
 "ioengine" : "sync",

"fdatasync" : "1",
 "directory" : "/var/lib/etcd",

"size" : "22m", [1]
```

```
"bs" : "2300" }, [2]
(.. omitted ..)
"write" : {
"io_bytes" : 23066700,
"io_kbytes" : 22526,
"bw_bytes" : 1319077,
"bw" : 1288,          [3]
"iops" : 573.511752, [4]
(.. omitted ..)
"read_ticks" : 3309,
"write_ticks" : 29285,
"in_queue" : 32594,
"util" : 98.318751
} ]
}
---------
99th percentile of fsync is 5406720 ns
99th percentile of the fsync is within the recommended
threshold - 10 ms, the disk can be used to host etcd [5]
```

[1] Chunk Size - 22MB (megabytes) is usually enough to analyze performance results.

[2] Instead of using 4k block size, etcd use small chunks of 2.3k block size, so it guarantees performance including with small writing fragmentation.

[3] Bandwidth considering traffic between the node and underlying storage. It is recommended you use at least a network interface of 1GB, for medium and large clusters the recommendation is a 10GB interface.

[4] The recommendation is at least 500 concurrent IOPS, as explained previously.

[5] The report of etcd IO check. In the example, 5.40 ms (milliseconds) demonstrates a reliable performance - that must be under 10 ms.

Besides using etcd-perf to check the disk performance, you could also perfectly use **fio** tool using custom parameters as you need, like block size, chunk size, and so on using the fio

binary tool, which is available using the standard Red Hat package manager (e.g.: `yum/dnf install fio`).

Note

For didactic reasons, we suppressed some results leaving only items that are pertinent for our analysis.

ETCD Sizing

To avoid any problems related to the CPU, you must understand if your cluster is well sized. You must consider some factors to check the cluster sizing, such as the number of customers using the platform, the expected number of requests per second, the amount of storage available for etcd.

First, let's give you some parameters to consider your cluster size:

Cluster Size	Nodes	Clients	Requests/s	Store
Small	Up to 50	Up to 100	Up to 200	Up to 100MB
Medium	Up to 250	Up to 500	Up to 1000	Up to 500MB
Large	Up to 1000	Up to 1500	Up to 10000	Up to 1024MB
Extra Large	Up to 3000	Over 1500	Over 10000	Over 1024MB

The following table demonstrates some use cases using public clouds and on-premises infrastructures according to the amount of CPU, Memory, Disk IOPS, and bandwidth linked to the cluster size.

Cluster Size	Provider	vCPU	Memory	IOPS	Disk Bandwidth (MB/s)
Small	AWS	2	8 GB	3600	56.25
Small	GCP	2	7.5 GB	1500	25
Small	On-premises	4	16 GB	300	N/A
Medium	AWS	4	16 GB	6000	93.75

Comentado [GF15]: Pecorito, qual é a conclusão a respeito desta tabela? A tabela é uma recomendação que estamos fazendo? O que o leitor deve entender ou usar com essa tabela? Acho importante sempre ter um direcionamento quanto ao q for colocado... tem que ter um "começo, meio e fim"... meu feeling é que faltou um "fim" pra essa parte de CPU

Comentado [RP16R15]: ok, corrigido.. lead-in inserido no final da tabela.

Comentado [RP17R15]: troquei também o nome da seçao para "Etcd sizing"....acho que fica mais intuitivo.

Medium	GCP	4	15 GB	4500	75
Medium	On-premises	4	16 GB	300	N/A
Large	AWS	8	32 GB	8000	125
Large	GCP	8	30 GB	7500	125
Large	On-premises	8	16 GB	300	N/A
xLarge	AWS	16	64 GB	16000	250
xLarge	GCP	16	64 GB	15000	250
xLarge	On-premises	16	16 GB	300	N/A

In a conclusion, when you size a cluster you should consider these thresholds because it is already benchmarked by etcd.io and their performance will be acceptable if these recommendations are followed. Further information regarding sizing the etcd cluster can be found at https://etcd.io/docs/v3.5/op-guide/hardware/.

In these previous sections regarding **Etcd**, you saw the ways of checking its performance, troubleshooting the cluster, and also got important information regarding best practices when size a new **Etcd** cluster. We expected you to enjoy the approach and take a closer look in the next section about authentication which will be as far as an interesting theme.

Authentication

Another important aspect of an OpenShift cluster is the user authentication and authorization flow. OpenShift's flexibility and ease-to-use authentication plug-ins are a smart way of setting up users and groups. Instead of simply having a vault of usernames and passwords, OpenShift's authentication service can authenticate a user in a variety of ways, we call it an identity provider. In this way, OpenShift is responsible for trusting the identity provider and allowing or denying authentication according to the provider. In the diagram below you can see how the process of authenticating a user works.

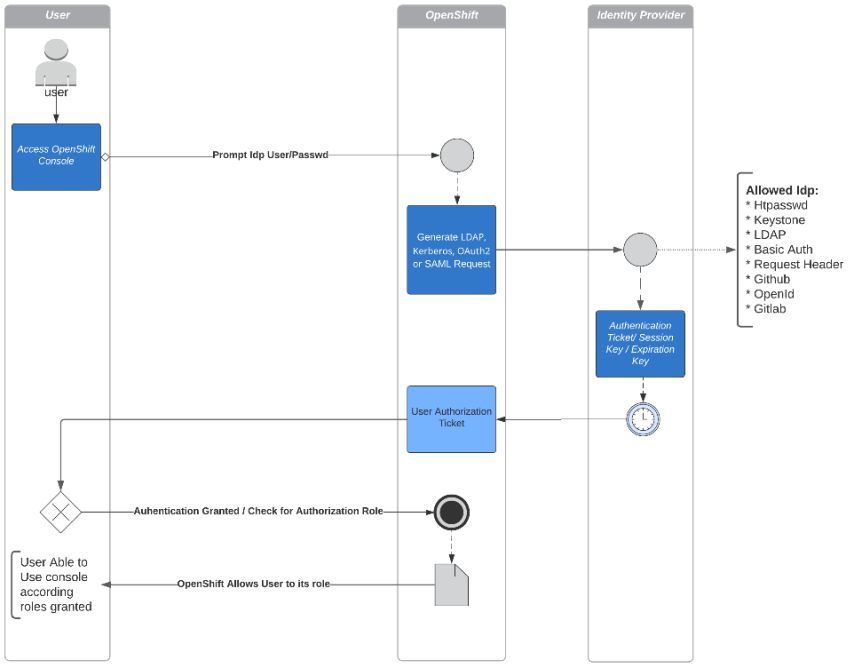


Figure 8.4 – Authentication and Authorization Flow

The identity provider is responsible for notifying OpenShift if the username and password are valid and returning to OpenShift the success or failure authentication status. This process is known by authentication (**AuthN** on some lectures).

As mentioned, we demonstrate the authentication process using the available identities providers, but it is important to stress that OpenShift will not fully trust the authenticated user, initially, it will use the default role of self-provisioner (in case it is enabled on the cluster) or it will simply only allow login to the platform without usage rights.

For authorization to occur (known as **AuthZ**) it is necessary to inform the appropriate role for each user, explicitly which role/clusterole will be assigned to the user. This can be accomplished through the inclusion of a group or direct assignment to the user.

To better understand which rolebinding would fit the user rights, you should consider the default rolebindings already existent in every OpenShift cluster:

- **admin** - This is a project admin, it allows every project scoped resources including the ability to create RoleBindings using ClusterRoles. It does not allow the modification of quota, limits, or cluster resources.
- **edit** - This is a project editor, it allows usage and manipulation at every project scoped resources, cannot change authorization objects.
- **view** - This is a project viewer, it allows the inspection of project scoped resources, works like a read-only rolebinding. Secrets inspections are not allowed.
- **cluster-admin** - This is the equivalent of the root user on *nix-like OS, it allows total control of any resource in the entire cluster.
- **cluster-reader** - This is useful for allowing users especially those that realize monitoring on the cluster, this rolebinding does not allow users under this role to escalate or manipulate objects on the cluster.

The role only is assigned after the first user successful login.

We showed that the authentication process is a bit demystified and you can realize the process is composed of authN & authZ. It is important to give the proper permission for each user, the previous diagram showed in a quick point of view how the steps of that authentication process work.

Troubleshooting reference guide – how to start?

In this section, you will see some approaches when starting to troubleshoot your OpenShift cluster. Due to the power of the **oc cli**, you will have different ways to succeed in almost any troubleshooting in your OpenShift cluster. Along with your training, you will gain the experience you need to take a step further in using and troubleshooting your OpenShift/Kubernetes cluster.

Describe objects

As we have mentioned, **oc cli** is a powerful tool to help OpenShift users to do a lot of operations, and also make some troubleshooting. One of the first step on troubleshooting is to getting some answers from objects.

- Choose an object from cluster (deployment/deployment config / service/pod) and execute describe.

```
oc describe pod sso-10-qm2hc
```

- Check output in the events session of the object and you will see events that offending object to work.

```
Events:
  Type     Reason          Age                From
Message
  ----     ------          ----               ----
-------
  Normal   Scheduled       90s                default-
scheduler  Successfully assigned rhsso/sso-10-qm2hc to ocp-
hml4.hybridcloud.com
  Normal   AddedInterface  89s                multus
Add eth0 [10.242.22.12/23] from openshift-sdn
  Normal   Pulling         39s (x3 over 89s)  kubelet
Pulling image "image-registry.openshift-image-
registry.svc:5000/rhsso/sso74-custom"
  Warning  Failed          33s (x3 over 83s)  kubelet
Failed to pull image "image-registry.openshift-image-
registry.svc:5000/rhsso/sso74-custom": rpc error: code =
Unknown desc = pinging container registry image-
registry.openshift-image-registry.svc:5000: Get
"https://image-registry.openshift-image-
registry.svc:5000/v2/": dial tcp 10.244.109.169:5000: connect:
no route to host
  Warning  Failed          33s (x3 over 83s)  kubelet
Error: ErrImagePull
  Normal   BackOff         8s (x4 over 83s)   kubelet
Back-off pulling image "image-registry.openshift-image-
registry.svc:5000/rhsso/sso74-custom"
  Warning  Failed          8s (x4 over 83s)   kubelet
Error: ImagePullBackOff
```

In this case, **oc describe** detected quickly the error related to image and you can act on deployment in order to correct it and make the application up and running.

Events

Another parameter on the **oc cli** that helps a lot during investigarion is **oc get events.** This is very useful to demonstrate a log of tasks recently executed. This tasks could be successful messages or even error messages. Events can be execute cluster-wide or scoped, check the sample below of events logs

```
 oc get events -n openshift-image-registry

LAST SEEN   TYPE      REASON              OBJECT
MESSAGE

35m         Normal    Scheduled           pod/cluster-image-
registry-operator-7456697c64-88hxc    Successfully assigned
openshift-image-registry/cluster-image-registry-operator-
7456697c64-88hxc to ocp-master2.hybridcloud.com

35m         Normal    AddedInterface      pod/cluster-image-
registry-operator-7456697c64-88hxc    Add eth0
[10.242.0.37/23] from openshift-sdn

35m         Normal    Pulled              pod/cluster-image-
registry-operator-7456697c64-88hxc    Container image
"quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:6a78c524aab5bc95c671811b2c76d59a6c2d394c8f9ba3f2a92
bc05a780c783a" already present on machine

(...omitted...)
```

In the previous sample, you see some entries on event logs of namespace openshift-image-registry, and you could identify any errors if it occurs.

Pod Logs

The first step of any troubleshooting in an OpenShift cluster must start with pod logs. Regularly, pod logs bring important information related to the scheduler, pod affinity/anti-affinity, container images, and persistent volumes.

There are several ways to check the pod logs:

Comentado [GF18]: Pecorito, aqui acho q antes de entrar em logs de pods poderia exercitar verificacoes mais basicas, como rodar um oc describe ou ver a aba events, oc status e outras dessas verificacoes iniciais que a gente faz antes de efetivamente entrar nos logs da aplicacao.... O que acha?

Comentado [RP19R18]: done!

- Common way inside the namespace:

```
oc project mynamespace

oc logs mypod
```

- From any namespace:

```
oc -n mynamespace logs mypod
```

- To check logs of a specific container inside a pod:

```
oc -n mynamespace logs -f mypod -c kube_proxy
```

- You can also check the logs using the OpenShift Console UI. To do so, access the `Logs` tab of desired namespace and pod:

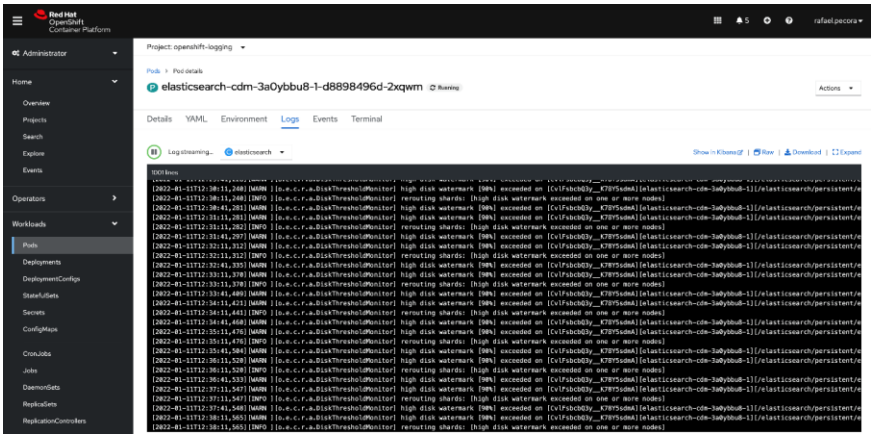


Figure 8.5 - Pod Logs example (OpenShift Console GUI)

You may also have issues during the application deployment. See next what you can check to find evidence of deployment problems.

Deployment Logs

In some cases, a pod did not start or remains in constant crashing which makes it difficult to get any log, since the pod is not in Running, CrashingLoopBack, ContainerInitializing, ContainerCreation, and so on.

In such cases, you can check the **deployment** or **deploymentconfig** logs, which will help you to identify deployment misconfigurations.

Similar to pod logs, deployment logs are accessible running the `oc logs` command. For `deployments` run the following:

```
oc -n mynamespace logs deployment/mydeploypods
```

For `deploymentconfigs` use this one:

```
oc -n  mynamespace logs dc/mydeploypods
```

Usually, you will not find exactly the issue root cause and solution to be applied, but it will give a good indication for you to find why it is failing, such as dependent components that are missing on the solution, like images not available, security context constraint with incorrect permissions, absence of configmaps, secrets, and serviceaccounts.

Operator Logs

As we already covered in this book, OpenShift uses several Operators to deploy and monitor critical features for the platform. That said, operators bring helpful logs to identify some configuration issues and instability with the platform. Those logs are stored in namespaces starting with the name **openshift-***, which is a standard for most tools included with OpenShift.

The reason to maintain several operator pods apart of project pods are related to some affinity/anti-affinity rules, taints/toleration strategies that user can apply to the cluster. Operator pods are the watchdog to maintain the namespaces' health watching to CRD (custom resource definition), its standards, liveness, and readiness and prevent the OpenShift's critical namespaces to suffer undesirable changes.

One of the main benefits of Operators for the cluster's stability is the ability to maintain the desired state of the operator namespace's objects. In other words, if any unwanted changes are made by mistake directly into the operator's objects will be reverted by the operator itself. Every change in the operator's objects needs to be done through the operator itself, by editing ConfigMaps or Custom Resources objects, according to the operator's specification. That also means that any changes are checked and confirmed by operators before they are effectively applied.

At the next subsection you will see a further information to oc cli, to check operators functions you must start with:

- List Cluster Operators

```
oc get co
```

- Choose some cluster operator to verify:

Comentado [GF20]: Acho q aqui faltou efetivamente botar o procedimento de checagem dos logs, sugiro adicionar um oc logs clusteroperator XXX, etc.

Comentado [RP21R20]: comecei com oc describe e falei que o proximo passo seria oc logs... senao ficará repetitivo de acordo com a explicacao de oc dada na seçao anterior, ok?

```
oc describe co <clusteroperatorName>

Example:

oc describe co storage
```

- Check into status output for the eventually error messages:

```
(...ommittted...)

Status:

  Conditions:

    Last Transition Time:  2021-08-26T14:51:59Z

    Message:               All is well

    Reason:                AsExpected

    Status:                False

    Type:                  Degraded

    Last Transition Time:  2021-08-26T14:51:59Z

    Message:               All is well

    Reason:                AsExpected

    Status:                False

    Type:                  Progressing

    Last Transition Time:  2021-08-26T14:51:59Z

    Message:
DefaultStorageClassControllerAvailable: No default
StorageClass for this platform

    Reason:                AsExpected

    Status:                True

    Type:                  Available

    Last Transition Time:  2021-08-26T14:52:00Z

    Message:               All is well

    Reason:                AsExpected
```

```
    Status:                 True
    Type:                   Upgradeable
(...ommittted...)
```

In the previous example, only a warning to set up a default StorageClass for a cluster. No critical issues were found at storage. In case of any issue, you start troubleshooting on events and log pods of the desired namespace

Oc cli

Oc is a **CLI tool** for OpenShift that gives powerful control to the Openshift API. You can debug, operate and manage cluster. The oc cli has some powerful options for troubleshooting clusters, using oc or kubectl you can run scoped or cluster commands according to the logged user. Some useful and powerful commands to troubleshooting are:

- High verbosity logs:

```
$ oc -n <namespaceName> logs <podName> -v 8
```

- Cluster Events:

```
$ oc get events
```

- Namespaced Events:

```
$ oc -n <namespaceName> get events
```

- Execute a single command inside the pod (double dash needed):

```
$ oc exec mypod -- date
```

- Execute a single command inside pod to a specific container (double dash needed):

```
$ oc exec mypod -c httpd-container -- date
```

- Create iterative commands spawning a pseudo-terminal:

```
$ oc exec mypod -i -t -- ls -t /usr
```

- Similar o exec, rsh open a terminal inside pod:

```
$ oc -n <namespaceName> rsh  <podName>
```

Comentado [GF22]: Nao seria “oc cli”? odo e o cli usado para dev...

Comentado [RP23R22]: vero, vi aqui... realmente, me confundi com o termo. atualizei e coloquei oc cli.

All of the previous commands help you to identify problems in a cluster or even an application, furthermore, you can investigate directly the node using **oc debug**.

```
$ oc debug node/<nodeName>
```

The **oc debug** gives you a non-root privilege access and you cannot run many OS commands without escalation, to do so we recommend you to **chroot** it.

```
$ chroot /host /bin/bash
```

After that, you can regularly use OS shell commands.

As you can see, OpenShift has a lot of debugging commands to help you identify cluster-wide or scoped issues, it is not recommended but it is possible to directly **ssh** on nodes. This kind of approach requires good knowledge of *Red Hat CoreOS* systems, **podman,** and **crio** commands to avoid node disruption.

If any of the previous commands helped you can still use **must-gather** commands which will generate a temporary pod and concatenate all node logs which is useful for the Red Hat engineering support team to analyze and correlate events among the nodes.

The most common way to run **must-gather** is:

```
oc adm must-gather --dest-dir=/local/directory
```

It will create a tar file under the chosen directory with all logs collected which can be very useful to identify problems.

Understanding misleading error messages

Even if you have learned the ways to identify a problem it is not unusual that the error presented does not provide enough information to help you correct the problem.

Thinking about this theme of unintelligible messages, we chose some very common ones to which we suggest some ways to solve the problem related to the error.

ImagePullBackOff

This is a common error and it is related to absence of container image. Check the following approach to being familiarized when this kind of issue would occur on the pods.

```
NAMESPACE                                                  NAME                                READY   STATUS
RESTARTS   AGE
namespace1                                                 e-reader-
backend-6449dd6d5f-tfmqm                 0/1
ImagePullBackOff   0          17h
```

Investigating pod log:

```
$ oc logs e-reader-backend-6449dd6d5f-tfmqm

Error from server (BadRequest): container "e-reader-backend"
in pod "e-reader-backend-6449dd6d5f-tfmqm" is waiting to
start: trying and failing to pull image
```

Looking at the error messages, typically it is linked to the absence of the image in the registry. This means some possible problems, the image, and its tag are not available through the imagestream address to the registry or with incorrect pointing in the deployment/deployment config.

Another correlation would be the node through which the scheduler will provision the pod is not able to download images from an external registry such as redhat.io, quay.io, or similar.

CrashLoopBackOff

This is an error that requires some more knowledge before acting on solving effectively. It occurs because OpenShift fails after checking all the prerequisites to start a pod, so many issues could result in this error message.

```
NAMESPACE                                                  NAME
READY   STATUS             RESTARTS   AGE

3scale                                                     backend-
redis-1-9qs2q                                             0/1
CrashLoopBackOff   211        17h
```

Investigating pod log:

```
$ oc logs backend-redis-1-9qs2q

1:M 11 Jan 13:02:19.042 # Bad file format reading the append
only file: make a backup of your AOF file, then use ./redis-
check-aof --fix <filename>
```

The log usually shows interesting things like a command to solve some mistaken build/deploy, but it can also be a trap to conduct you to wrong correction attempt. It is important to take into account when a pod had persistent volume or it is expecting another pod with persistent volume to start. In the example, the pod log showed you a message to recover some files, so during your experience using a container, you can check the problem and will perceive that a pod can contain more than one container and the error could also be related to another container that is not healthy yet.

In this case, analyze other containers in the same pod and other pods in the same namespace to get the right conclusion and better solution to the case.

Init:0/1

Typically, when you got Init:0/1 it means the pod is waiting for another pod or a condition that hasn't been satisfied yet. We demonstrate below what kinds of conditions can result from this message and how to solve it.

```
NAMESPACE                                          NAME
READY   STATUS             RESTARTS   AGE

3scale                                             backend-
cron-1-zmnpj                                      0/1
Init:0/1           0          17h
```

Investigating pod log:

```
# oc logs backend-cron-1-zmnpj

Error from server (BadRequest): container "backend-cron" in
pod "backend-cron-1-zmnpj" is waiting to start:
PodInitializing
```

This is the most confusing status when you are troubleshooting error messages. Uncertainly, it can be anything wrong in the namespace, so the error message only shows "PodInitializing". You could interpret it as a condition when a pod is waiting to start, meanwhile, this message means that a condition isn't satisfied.

To help you, we have listed some items that must be checked for the pod to finally initialize and upload its containers properly.

- Check serviceaccount, some containers need a specific account name and policy applied to start.

- Check security context constraints (**SCC**), the SCC defines a context that allows/blocks some kind of objects to be used, a common example is a persistent volume that needs at least a restricted **scc** policy to attach a volume to a pod.
- Check other containers into namespace: depending on build strategy it is possible to define a pod to depend on another pod through liveness and readiness probing.

We will discuss in-depth Security Context Constraints (SCC) in *Chapter 10 – Openshift Security*.

Summary

In this chapter we focused on important components of OpenShift, such as Operators and their role to maintain the resilience of a cluster, we also talk about situations that can cause damage to the cluster.

We deep dive into the heart of OpenShift called Etcd, understanding its importance for the cluster, how to prepare it to receive a high volume of traffic, verifying its sizing, performance, and understanding how to perform troubleshooting in case of problems.

We've also discussed a bit about the AuthN & AuthZ (Authentication and Authorization) process so you know the power and flexibility of Openshift in using external identity providers without changing or turning the cluster a harder work.

We finish it with some important troubleshooting tips and tools that will certainly help you in your daily management of clusters and applications.

In the next chapter, we will bring important information about network management in OpenShift. We will discuss and give examples to understand the main differences between a pod network and a service network, as well as understand the difference between North-South and East-West traffic, keep on this interesting reading and learn more in chapter *9 - OpenShift Network*.

Further reading

If you want to look at more information about what we covered in this chapter, check the following references:

- Red Hat Knowledge Base – Etcd Recommendation: https://access.redhat.com/solutions/4770281
- Etcd Quorum Model: https://etcd.io/docs/v3.5/faq/

- Understanding Etcd quorum: http://thesecretlivesofdata.com/raft/
- Etcd Hardware Sizing Recommendation: https://etcd.io/docs/v3.5/op-guide/hardware/
- Etcd Tunning Options: https://etcd.io/docs/v3.5/tuning/
- Etcd Benchmarking thresholds: https://etcd.io/docs/v3.5/benchmarks/
- Kubernetes Authentication flow: https://kubernetes.io/docs/reference/access-authn-authz/authentication/
- Openshift Identity Providers: https://docs.openshift.com/container-platform/4.7/authentication/understanding-identity-provider.html
- Learn more about Security Context Constraints: https://docs.openshift.com/container-platform/4.8/authentication/managing-security-context-constraints.html