3

Multi-Tenant Considerations

As with almost any software, things get more difficult as you scale in number and size. In the previous chapter, we looked at the most important aspects related to the architecture of an OpenShift cluster. In this chapter, we are going to cover some things you should understand when working with multiple environments on one or more clusters.

In this chapter, we will cover the following topics:

- What is multitenancy?
- Handling multiple tenants
- Multitenancy on OpenShift
- Multi-tenant strategies

What is multitenancy?

Multitenancy is the ability to provide services for multiple users or groups (also known as **tenants**) using a single platform instance. A platform architecture can be designed to be single- or multi-tenant:

- In a single-tenant platform architecture, there is no isolation within an instance of the platform. This means that there is no separation in an instance and that, as such, there is no way to isolate objects for users or groups. In this case, to achieve multitenancy, you need to provision a separate platform instance for every tenant.

- In a multi-tenant platform architecture, it is possible to isolate objects and data between different tenants. Therefore, you can protect each tenant's data and objects, thus providing enough privacy and security, even by using a single platform instance.

Depending on the architectural design and how OpenShift is used, you can have both types of platforms (single- or multi-tenant). In this chapter, you will learn how to define the best way to consider this while designing your OpenShift clusters.

Handling multiple tenants

There are many different ways to work with multiple tenants on OpenShift, with the most obvious one being to have a single cluster for each tenant. However, this is not always possible or the best option: having dedicated hardware and a platform for every tenant can be costly, difficult to maintain, and not efficient. With shared clusters, multiple workloads from different tenants share the same computing capacity, enabling more efficient computing usage.

OpenShift can provide isolation for objects, computing, network, and other hardware resources for each tenant, ensuring they are isolated from each other. In the next section, we are going to look at the different types of isolation and how to utilize them.

Multitenancy in OpenShift

When it comes to multitenancy on OpenShift, several objects are involved. The following table shows some important resources that provide multi-tenant capabilities, all of which we are going to cover in this chapter:

Kubernetes resources	Limit Kubernetes resources between different applications, tenants, and users.	Namespaces and role-based access control (RBAC).
Computing	Limit computing resources for each tenant.	ResourceQuota, nodeSelector, Taint, and Tolerations.
Network	Limit network traffic between the different applications of each tenant.	NetworkPolicy and ingress sharding.
Storage	Limit which kind and amount of storage resources are allowed for each tenant.	ResourceQuota and StorageClass.

The following diagram illustrates how these resources are combined to provide isolation and enable multitenancy:

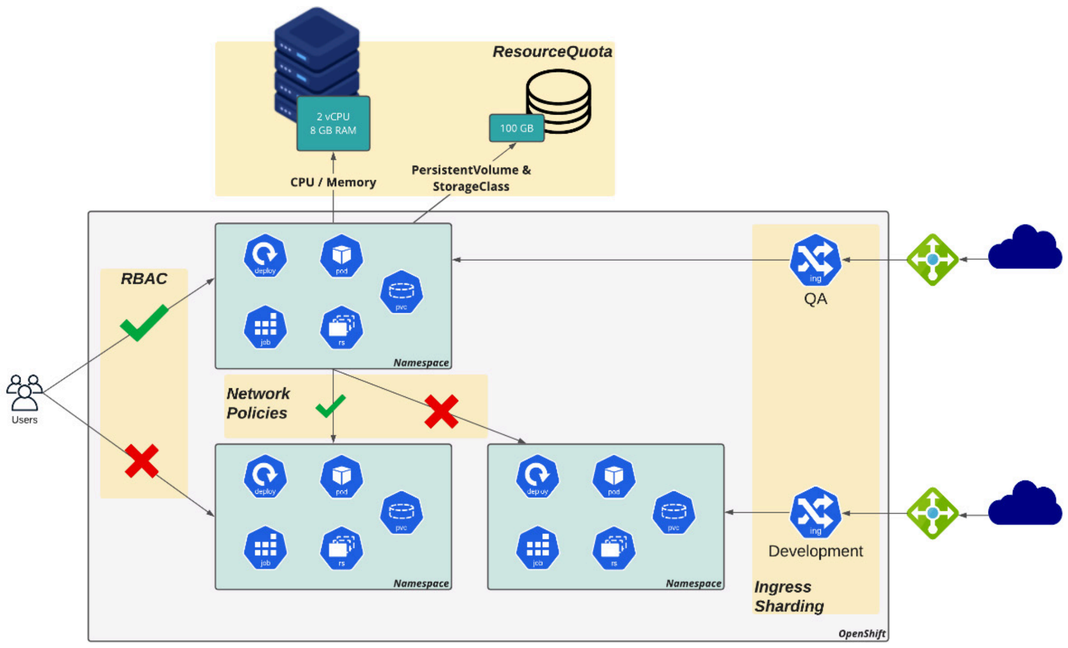

Figure 3.1 – Multitenancy and isolation

Now, let's look at how these objects are used to provide multitenancy capabilities on OpenShift. We are going to use all of them while covering practical examples from ***Chapter 5***, *OpenShift Development*, onward, so don't worry about how to use them for now. Instead, focus on understanding how they provide isolation capabilities to enable multitenancy.

Namespaces

Namespaces provide some level of isolation. Using namespaces, you can define a limited space for the following:

- **Kubernetes workloads**: **Pods**, **Secrets**, **ConfigMaps**, **Deployments**, and so on.
- **Access control**: Isolate the namespace resource's access by giving the appropriate **roles** (permissions) to the users or groups.
- **Limit resource consumption**: It is possible to limit the number of resources that are consumed by a namespace using **ResourceQuotas**.

Role-based access control

Permission control over resources on OpenShift is done using **roles** and **RoleBindings**. **Roles** are a set of actions (such as **`get`**, **`list`**, **`watch`**, **`create`**, **`upgrade`**, and so on) that are permitted over resources (such as **Pods**, **Services**, **Deployments**, **Jobs**, and so on), while **RoleBindings** (or **ClusterRoleBinding**) are how you bind a role to a subject (groups, users, or **ServiceAccounts**).

You are required to use roles and RoleBindings to give users the right permissions to the right namespaces, according to the tenants and the separation logic you want to implement.

ResourceQuotas

ResourceQuotas allows a cluster administrator to constraint a namespace to a limited set of resources. It can limit computing resources and/or the number of objects. It is a crucial thing to consider when you're using shared clusters to ensure there's a limited capacity for each tenant. Without ResourceQuotas, only one namespace can consume the entire capacity of a worker node, for instance.

nodeSelectors, taints, and tolerations

Through **nodeSelectors**, you can dedicate workers for a specific reason or tenant. With nodeSelectors, it is possible to isolate physical compute resources for each tenant: in a 10-node cluster, you can have, for instance, five nodes for QA and the other five for development. **Taints** and **tolerations** are different ways of doing this: while with nodeSelectors, you instruct a Pod to be scheduled in a defined set of nodes that contain a certain label, with taints, you instruct a worker to *repeal* Pods that do not contain a certain toleration to run in it.

NetworkPolicy

A NetworkPolicy provides a standard way to isolate network traffic between Pods and namespaces. It works like a firewall in which you define ingress and/or egress policies to accept/deny traffic flows between different Pods and namespaces.

Ingress/router sharding

On OpenShift, you can create multiple ingress controllers, which will allow you to isolate ingress traffic between different tenants.

Multi-tenant strategies

It is important to understand that there is no physical isolation when using the multi-tenant objects listed previously – isolation is defined and implemented by the software. However, it is possible to provide a physical level of isolation by using different multi-tenant strategies, as you will see now. The best strategy depends on the requirements you have in your company; some companies care more about having an efficient use of computing resources, while others don't care about spending more resources in favor of the most secure isolation strategy possible.

Some different strategies are as follows:

- Dedicated clusters, one for each tenant

- A shared cluster with no physical separation of resources
- A shared cluster with dedicated worker nodes
- A shared cluster with dedicated worker nodes and ingress controllers

Dedicated clusters

As we have already mentioned, the most obvious strategy is to have a different cluster for each tenant. The following diagram shows an example of providing services for two tenants (QA and Development):

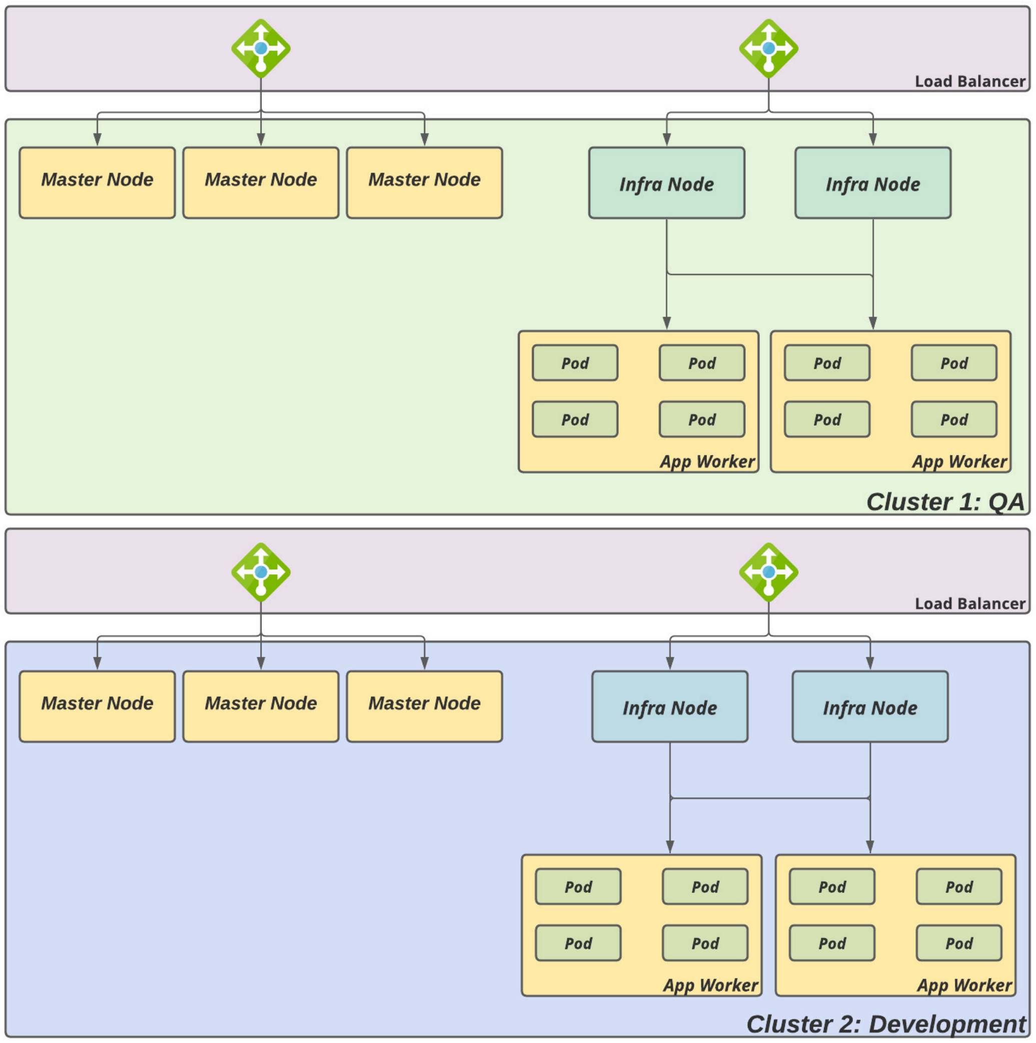

Figure 3.2 – Dedicated clusters

The following table contains a summary of the pros and cons of this strategy, from our point of view:

Pros	Cons
Highest isolation: Physical separation of the workloads, which ensures workloads from different tenants will not be able to communicate inside a cluster. **Computing resources guaranteed:** You don't need (if you don't want) to deal with ResourceQuotas to guarantee computing resources for the tenants.	**Higher resource usage:** For each cluster, you need at least three master nodes and likely some infra-nodes. **Operational overhead:** You may need to manage a large number of clusters.

This type of architecture is usually recommended for companies that have strict requirements for physically isolated environments and don't want to be dependent on multi-tenant software and processes to provide isolation.

Shared clusters, no physical separation

On the other hand, you may decide to have one shared cluster providing services for multiple tenants while using OpenShift objects to enable multitenancy (namespaces, RBAC, ResourceQuotas, and NetworkPolicies). You can see a simple schema of this strategy in the following diagram:

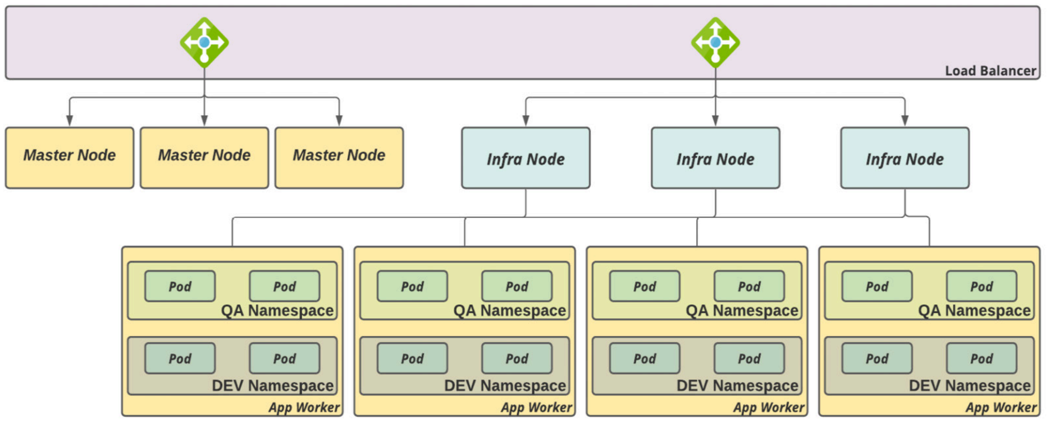

Figure 3.3 – Shared cluster, no physical separation

The following table shows some of the pros and cons of this strategy:

This kind of architecture usually works well for non-production environments, in which some incidents related to performance degradation, for instance, are sometimes tolerated.

Shared clusters, dedicated worker nodes

If you have to provide computing resources that are physically isolated, this may be the right direction to take. In this case, you are going to use the same objects to provide isolation, but you dedicate worker nodes for specific tenants. The following is a simple schema:

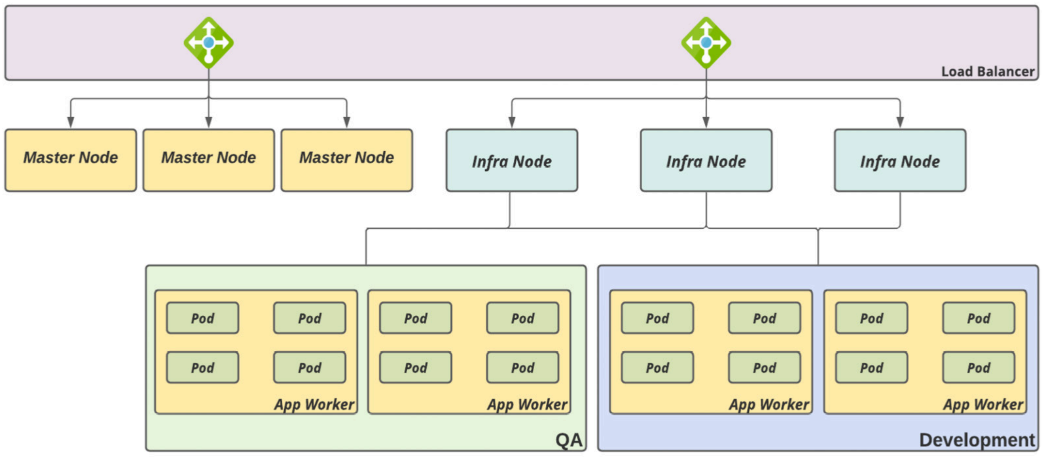

Figure 3.4 – Shared cluster, dedicated worker nodes

This strategy can have the following pros and cons:

Pros	Cons
Computing physical isolation: Dedicated worker nodes for specific tenants and, as such, have physical isolation of computing power.	**Less efficient resource usage:** You may have resources available in one tenant while another suffers from a lack of resources, for instance.
Operational effort: Save operational effort by managing only one (or a few) cluster(s).	**No ingress isolation:** Ingress for all tenants is provided by the same group of ingress instances, which means that a high traffic load in an application from a particular tenant may impact applications from other tenants.

This kind of architecture usually works well for non-production environments, in which some incidents related to performance degradation, for instance, are sometimes tolerated. This architecture also works well for production workloads that don't have requirements for high HTTP(S) throughput and low-latency HTTP(S). The following link provides a capacity baseline for a single OpenShift ingress (**HAProxy**), for comparison: https://docs.openshift.com/container-platform/latest/scalability_and_performance/routing-optimization.html.

Pros	Cons
Lowest resource usage: As you don't need to have dedicated resources for each tenant, you may have more efficient use of computing resources. **Operational effort**: Save operational effort by managing only one (or a few) cluster(s).	**No physical isolation:** Isolation is done based on the namespace and other objects, which is highly dependent on the usage process. A wrong ResourceQuota setup, for instance, can lead to capacity problems; or a mistake in a NetworkPolicy configuration can result in inappropriate allowed/denied network traffic. These problems can be mitigated by automating the creation and management of Namespaces, ResourceQuotas, NetworkPolicies, and others using tools such as Ansible, Jira, Jenkins, ServiceNow, and more. **No ingress isolation:** Ingress for all the tenants is provided by the same group of ingress instances, which means that an application with a high traffic load from a particular tenant may impact applications from other tenants.

Shared clusters, dedicated worker nodes, and ingress controllers

Finally, with this strategy, you can share a cluster among different tenants by providing a higher level of isolation. The following diagram shows this cluster's architecture:

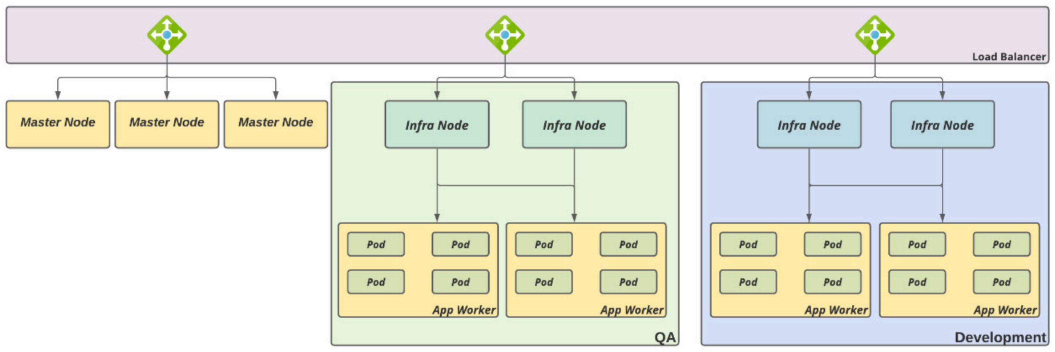

Figure 3.5 – Shared cluster, dedicated worker nodes, and ingress controllers

This strategy can bring the following pros and cons:

Pros	Cons
Computing physical isolation: Have dedicated worker nodes for specific tenants and, as such, have physical isolation of computing power. **Ingress isolation:** Each tenant has ingress instances, which ensure that a high traffic load in one tenant will not impact other tenants. **Operational effort:** Save operational effort by managing only one (or a few) cluster(s).	**Less efficient resource usage:** You may have resources that are available in one tenant while another suffers from a lack of resources. Additional infra-nodes may be required for the ingress of each tenant.

This kind of architecture usually works well for both production and non-production environments. If it's well defined and used, it has the same effect as dedicated clusters. The only difference between this strategy and dedicated clusters is that, in this case, a configuration mistake could lead to an application being deployed in the wrong tenant (by using the incorrect **nodeSelector** in a namespace, for instance).

OpenShift multitenancy checklist

To conclude this chapter, we have decided to add some additional items to the checklist we started building in the previous chapter, as follows:

Are multiple tenants required?	• Yes • No
Define the tenants.	For example, Development, QA, Production Dept A, Production Dept B, and so on.
Level of isolation required between tenants?	• Compute • Network • Ingress
Does the workload require high throughput and low latency for HTTP(s) requests?	• Yes • No
Decision.	• Dedicated clusters • Shared clusters using namespaces, RBAC, ResourceQuotas, and NetworkPolicies • Shared clusters with dedicated worker nodes • Shared clusters with dedicated worker nodes and ingress controllers
Strategy to create and maintain namespaces, RBAC, ResourceQuotas, and NetworkPolicies.	• Manual • Automated using _____________ (specify)
Responsible for creating and maintaining namespaces, RBAC, ResourceQuotas, and NetworkPolicies.	Specify the team(s) that will be responsible for creating and maintaining the namespaces and other objects with the right labels, permissions, quotas, and so on. This ensures data privacy and security between different tenants.

In the next chapter, you will acquire more knowledge about the personas that are usually related to OpenShift, from C-level to operational level, and what skills are required for each role. You will be able to understand what you should expect from each role and prepare yourself and your team to work well with OpenShift.

Summary

In this chapter, we looked at some of the strategies that provide services for multiple tenants with OpenShift clusters. You now understand that we can have dedicated or shared OpenShift clusters to host tenants. You also saw that with shared clusters, you can provide some level of isolation for each tenant by using namespaces, ResourceQuotas, NetworkPolicies, and other objects to provide multitenancy or even have a physical separation of workers and/or ingress; the best option for your use case depends on the requirements of your organization, workloads, and environments.

However, I need to warn you that in the current hybrid cloud world, you will probably need to work with clusters in different providers and regions, which may lead you to have an increasing number of clusters. But don't worry – as we saw in ***Chapter 1***, *Hybrid Cloud Journey and Strategies*, many great tools can help us manage several clusters, such as Red Hat Advanced Cluster Management, Advanced Cluster Security and Quay; we will take a deep dive into these tools by covering practical examples in the last part of this book, which is dedicated only to them.

In the next chapter, we will learn about the personas and skillsets that are usually related to OpenShift, their main duties and responsibilities, and other important factors.

Further reading

If you want to find out more about the concepts that were covered in this chapter, check out the following references:

- *What is multitenancy?* **https://www.redhat.com/en/topics/cloud-computing/what-is-multitenancy**
- *Multi-tenant network isolation (OpenShift documentation):* **https://docs.openshift.com/container-platform/4.8/networking/network_policy/multitenant-network-policy.html**

Previous chapter Next chapter

‹ Chapter 2: Architecture Overview and Definitions

Chapter 4: OpenShift Personas and Skillsets ›