tion for team workspaces, projects, and even specific applications. But ultimately, for a greater isolation of certain environments, namespaces are not enough, and having separate clusters is common. Typically, there is one nonproduction Kubernetes cluster used for some environments (development, testing, and integration testing) and another production Kubernetes cluster to represent performance testing and production environments.

Let's see some of the characteristics of namespaces and how they can help us in different scenarios:

- A namespace is managed as a Kubernetes resource.
- A namespace provides scope for resources such as containers, Pods, Services, or ReplicaSets. The names of resources need to be unique within a namespace, but not across them.
- By default, namespaces provide scope for resources, but nothing isolates those resources and prevents access from one resource to another. For example, a Pod from a development namespace can access another Pod from a production namespace as long as the Pod IP address is known. However, there are Kubernetes plugins that provide networking isolation to achieve true multitenancy across namespaces if desired.
- Some other resources such as namespaces themselves, nodes, and PersistentVolumes do not belong to namespaces and should have unique cluster-wide names.
- Each Kubernetes Service belongs to a namespace and gets a corresponding DNS address that has the namespace in the form of `<service-name>.<namespace-name>.svc.cluster.local`. So the namespace name is in the URI of every Service belonging to the given namespace. That's one reason it is vital to name namespaces wisely.
- ResourceQuotas provide constraints that limit the aggregated resource consumption per namespace. With ResourceQuotas, a cluster administrator can control the number of objects per type that are allowed in a namespace. For example, a developer namespace may allow only five ConfigMaps, five Secrets, five Services, five ReplicaSets, five PersistentVolumeClaims, and ten Pods.
- ResourceQuotas can also limit the total sum of computing resources we can request in a given namespace. For example, in a cluster with a capacity of 32 GB RAM and 16 cores, it is possible to allocate half of the resources—16 GB RAM and 8 cores—for the production namespace, 8 GB RAM and 4 cores for staging environment, 4 GB RAM and 2 cores for development, and the same amount for testing namespaces. The ability of imposing resource constraints on a group of objects by using namespaces and ResourceQuotas is invaluable.