# NetApp

# Solution Validation and Use Cases

## NetApp Solutions

NetApp
July 15, 2022

# Table of Contents

# Solution Validation and Use Cases: Red Hat OpenShift with NetApp

The examples provided on this page are solution validations and use cases for Red Hat OpenShift with NetApp.

- Deploy a Jenkins CI/CD Pipeline with Persistent Storage
- Configure Multitenancy on Red Hat OpenShift with NetApp
- Red Hat OpenShift Virtualization with NetApp ONTAP
- Advanced Cluster Management for Kubernetes on Red Hat OpenShift with NetApp

Next: Videos and Demos.

# Deploy a Jenkins CI/CD Pipeline with Persistent Storage: Red Hat OpenShift with NetApp

This section provides the steps to deploy a continuous integration/continuous delivery or deployment (CI/CD) pipeline with Jenkins to validate solution operation.

## Create the resources required for Jenkins deployment

To create the resources required for deploying the Jenkins application, complete the following steps:

1. Create a new project named Jenkins.

# Create Project

Name *

Jenkins

Display Name

Description

Cancel    Create

2. In this example, we deployed Jenkins with persistent storage. To support the Jenkins build, create the PVC. Navigate to Storage > Persistent Volume Claims and click Create Persistent Volume Claim. Select the storage class that was created, make sure that the Persistent Volume Claim Name is jenkins, select the appropriate size and access mode, and then click Create.

## Create Persistent Volume Claim

Edit YAML

**Storage Class**

SC basic ▾

Storage class for the new claim.

**Persistent Volume Claim Name** *

jenkins

A unique name for the storage claim within the project.

**Access Mode** *

◉ Single User (RWO)  ◯ Shared Access (RWX)  ◯ Read Only (ROX)

Permissions to the mounted drive.

**Size** *

100        GiB ▾

Desired storage capacity.

☐ Use label selectors to request storage

Use label selectors to define how storage is created.

Create        Cancel

## Deploy Jenkins with Persistent Storage

To deploy Jenkins with persistent storage, complete the following steps:

1. In the upper left corner, change the role from Administrator to Developer. Click +Add and select From Catalog. In the Filter by Keyword bar, search for jenkins. Select Jenkins Service with Persistent Storage.

Project: jenkins ▼

## Developer Catalog

Add shared apps, services, or source-to-image builders to your project from the Developer Catalog. Cluster admins can install additional apps which will show up here automatical

All Items | All Items

Languages

Databases      jenkins      Group By: None ▼

Middleware

CI/CD

Other

| | Template | | Template | | Template |
| Jenkins | | Jenkins | | Jenkins (Ephemeral) | |
| provided by Red Hat, Inc. | | provided by Red Hat, Inc. | | provided by Red Hat, Inc. | |

**Type**

☑ Operator Backed (0)

☐ Helm Charts (0)

☑ Builder Image (0)

☑ Template (4)

☐ Service Class (0)

Jenkins service, with persistent storage. NOTE: You must have persistent volumes available in...

Jenkins service, with persistent storage. NOTE: You must have persistent volumes available in...

Jenkins service, without persistent storage. WARNING: Any data stored will be lost upon...

Template

Jenkins (Ephemeral)
provided by Red Hat, Inc.

Jenkins service, without persistent storage. WARNING:

2. Click `Instantiate Template`.

# Jenkins
Provided by Red Hat, Inc.      ✕

**Instantiate Template**

**Provider**

Red Hat, Inc.

**Support**

Get support ☑

**Created At**

🌐 May 26, 3:58 am

**Description**

Jenkins service, with persistent storage.

NOTE: You must have persistent volumes available in your cluster to use this template.

## Documentation

https://docs.okd.io/latest/using_images/other_images/jenkins.html ☑

3. By default, the details for the Jenkins application are populated. Based on your requirements, modify the parameters and click Create. This process creates all the required resources for supporting Jenkins on

OpenShift.

## Instantiate Template

**Namespace** *

`(PR) jenkins`  ▼

**Jenkins Service Name**

`jenkins`

The name of the OpenShift Service exposed for the Jenkins container.

**Jenkins JNLP Service Name**

`jenkins-jnlp`

The name of the service used for master/slave communication.

**Enable OAuth in Jenkins**

`true`

Whether to enable OAuth OpenShift integration. If false, the static account 'admin' will be initialized with the password 'password'.

**Memory Limit**

`1Gi`

Maximum amount of memory the container can use.

**Volume Capacity** *

`50Gi`

Volume space available for data, e.g. 512Mi, 2Gi.

**Jenkins ImageStream Namespace**

`openshift`

The OpenShift Namespace where the Jenkins ImageStream resides.

**Disable memory intensive administrative monitors**

`false`

Whether to perform memory intensive, possibly slow, synchronization with the Jenkins Update Center on start. If true, the Jenkins core update monitor and site warnings monitor are disabled.

**Jenkins ImageStreamTag**

`jenkins:2`

Name of the ImageStreamTag to be used for the Jenkins image.

**Fatal Error Log File**

`false`

When a fatal error occurs, an error log is created with information and the state obtained at the time of the fatal error.

**Allows use of Jenkins Update Center repository with invalid SSL certificate**

`false`

Whether to allow use of a Jenkins Update Center that uses invalid certificate (self-signed, unknown CA). If any value other than 'false', certificate check is bypassed. By default, certificate check is enforced.

[Create]  [Cancel]

### Jenkins
INSTANT-APP JENKINS
[View documentation] [Get support]

Jenkins service, with persistent storage.

NOTE: You must have persistent volumes available in your cluster to use this template.

The following resources will be created:

- DeploymentConfig
- PersistentVolumeClaim
- RoleBinding
- Route
- Service
- ServiceAccount

4. The Jenkins pods take approximately 10 to 12 minutes to enter the Ready state.

Project: jenkins  ▼

## Pods

[ Create Pod ]                                        Filter by name...

| 1 Running | 0 Pending | 0 Terminating | 0 CrashLoopBackOff | 1 Completed | 0 Failed | 0 Unknown |

Select all filters                                                           1 of 2 Items

| Name ↑ | Namespace | Status | Ready | Owner | Memory | CPU |
|--------|-----------|--------|-------|-------|--------|-----|
| P jenkins-1-c77n9 | NS jenkins | ⟳ Running | 1/1 | RC jenkins-1 | - | 0.004 cores | ⋮ |

5. After the pods are instantiated, navigate to Networking > Routes. To open the Jenkins webpage, click the URL provided for the jenkins route.

Project: jenkins  ▼

## Routes

[ Create Route ]                                      Filter by name...

| 1 Accepted | 0 Rejected | 0 Pending |    Select all filters              1 Item

| Name ↓ | Namespace | Status | Location | Service |
|--------|-----------|--------|----------|---------|
| RT jenkins | NS jenkins | ✔ Accepted | https://jenkins-jenkins.apps.rhv-ocp-cluster.cie.netapp.com ☑ | S jenkins | ⋮ |

6. Because OpenShift OAuth was used while creating the Jenkins app, click Log in with OpenShift.

7. Authorize the Jenkins service account to access the OpenShift users.

## Authorize Access

Service account jenkins in project jenkins is requesting permission to access your account (kube:admin)

### Requested permissions

☑ **user:info**
   Read-only access to your user information (including username, identities, and group membership)

---

☑ **user:check-access**
   Read-only access to view your privileges (for example, "can I create builds?")

You will be redirected to https://jenkins-jenkins.apps.rhv-ocp-cluster.cie.netapp.com/securityRealm/finishLogin

[ Allow selected permissions ]   [ Deny ]

8. The Jenkins welcome page is displayed. Because we are using a Maven build, complete the Maven installation first. Navigate to Manage Jenkins > Global Tool Configuration, and then, in the Maven subhead, click Add Maven. Enter the name of your choice and make sure that the Install Automatically option is selected. Click Save.

9. You can now create a pipeline to demonstrate the CI/CD workflow. On the home page, click Create New Jobs or New Item from the left-hand menu.



10. On the Create Item page, enter the name of your choice, select Pipeline, and click Ok.



11. Select the Pipeline tab. From the Try Sample Pipeline drop-down menu, select Github + Maven. The code is automatically populated. Click Save.

General    Build Triggers    Advanced Project Options    **Pipeline**

Advanced...

## Pipeline

Definition    Pipeline script ▼

Script

```
1 ▼ node {
2       def mvnHome
3 ▼     stage('Preparation') { // for display purposes
4           // Get some code from a GitHub repository
5           git 'https://github.com/jglick/simple-maven-project-with-tests.git'
6           // Get the Maven tool.
7           // ** NOTE: This 'M3' Maven tool must be configured
8           // **       in the global configuration.
9           mvnHome = tool 'M3'
10      }
11 ▼    stage('Build') {
12          // Run the maven build
13 ▼        withEnv(["MVN_HOME=$mvnHome"]) {
14 ▼            if (isUnix()) {
15                  sh '"$MVN_HOME/bin/mvn" -Dmaven.test.failure.ignore clean package'
16 ▼            } else {
17                  bat(/"%MVN_HOME%\bin\mvn" -Dmaven.test.failure.ignore clean package/)
```

GitHub + Maven ▼

☑ Use Groovy Sandbox

Pipeline Syntax

Save    Apply

12. Click Build Now to trigger the development through the preparation, build, and testing phase. It can take several minutes to complete the whole build process and display the results of the build.

13. Whenever there are any code changes, the pipeline can be rebuilt to patch the new version of software enabling continuous integration and continuous delivery. Click Recent Changes to track the changes from the previous version.

# Configure Multi-tenancy on Red Hat OpenShift with NetApp ONTAP

## Configuring multitenancy on Red Hat OpenShift with NetApp

Many organizations that run multiple applications or workloads on containers tend to deploy one Red Hat OpenShift cluster per application or workload. This allows them to implement strict isolation for the application or workload, optimize performance, and reduce security vulnerabilities. However, deploying a separate Red Hat OpenShift cluster for each application poses its own set of problems. It increases operational overhead having to monitor and manage each cluster on its own, increases cost owing to dedicated resources for different applications, and hinders efficient scalability.

To overcome these problems, one can consider running all the applications or workloads in a single Red Hat OpenShift cluster. But in such an architecture, resource isolation and application security vulnerabilities are one of the major challenges. Any security vulnerability in one workload could naturally spill over into another workload, thus increasing the impact zone. In addition, any abrupt uncontrolled resource utilization by one application can affect the performance of another application, because there is no resource allocation policy by default.

Therefore, organizations look out for solutions that pick up the best in both worlds, for example, by allowing them to run all their workloads in a single cluster and yet offering the benefits of a dedicated cluster for each workload.

One such effective solution is to configure multitenancy on Red Hat OpenShift. Multitenancy is an architecture that allows multiple tenants to coexist on the same cluster with proper isolation of resources, security, and so on. In this context, a tenant can be viewed as a subset of the cluster resources that are configured to be used by a particular group of users for an exclusive purpose. Configuring multitenancy on a Red Hat OpenShift cluster provides the following advantages:

- A reduction in CapEx and OpEx by allowing cluster resources to be shared
- Lower operational and management overhead
- Securing the workloads from cross-contamination of security breaches
- Protection of workloads from unexpected performance degradation due to resource contention

For a fully realized multitenant OpenShift cluster, quotas and restrictions must be configured for cluster resources belonging to different resource buckets: compute, storage, networking, security, and so on. Although we cover certain aspects of all the resource buckets in this solution, we focus on best practices for isolating and securing the data served or consumed by multiple workloads on the same Red Hat OpenShift cluster by configuring multitenancy on storage resources that are dynamically allocated by Astra Trident backed by NetApp ONTAP.

## Architecture

Although Red Hat OpenShift and Astra Trident backed by NetApp ONTAP do not provide isolation between workloads by default, they offer a wide range of features that can be used to configure multitenancy. To better understand designing a multitenant solution on a Red Hat OpenShift cluster with Astra Trident backed by NetApp ONTAP, let us consider an example with a set of requirements and outline the configuration around it.

Let us assume that an organization runs two of its workloads on a Red Hat OpenShift cluster as part of two projects that two different teams are working on. The data for these workloads reside on PVCs that are dynamically provisioned by Astra Trident on a NetApp ONTAP NAS backend. The organization has a requirement to design a multitenant solution for these two workloads and isolate the resources used for these projects to make sure that security and performance is maintained, primarily focused on the data that serves those applications.

The following figure depicts the multitenant solution on a Red Hat OpenShift cluster with Astra Trident backed by NetApp ONTAP.

**Technology requirements**

1. NetApp ONTAP storage cluster
2. Red Hat OpenShift cluster
3. Astra Trident

**Red Hat OpenShift – Cluster resources**

From the Red Hat OpenShift cluster point of view, the top-level resource to start with is the project. An OpenShift project can be viewed as a cluster resource that divides the whole OpenShift cluster into multiple virtual clusters. Therefore, isolation at project level provides a base for configuring multitenancy.

Next up is to configure RBAC in the cluster. The best practice is to have all the developers working on a single project or workload configured into a single user group in the Identity Provider (IdP). Red Hat OpenShift allows IdP integration and user group synchronization thus allowing the users and groups from the IdP to be imported into the cluster. This helps the cluster administrators to segregate access of the cluster resources dedicated to a project to a user group or groups working on that project, thereby restricting unauthorized access to any cluster resources. To learn more about IdP integration with Red Hat OpenShift, see the documentation here.

**NetApp ONTAP**

It is important to isolate the shared storage serving as a persistent storage provider for a Red Hat OpenShift cluster to make sure that the volumes created on the storage for each project appear to the hosts as if they are

created on separate storage. To do this, create as many SVMs (storage virtual machines) on NetApp ONTAP as there are projects or workloads, and dedicate each SVM to a workload.

**Astra Trident**

After you have different SVMs for different projects created on NetApp ONTAP, you must map each SVM to a different Trident backend. The backend configuration on Trident drives the allocation of persistent storage to OpenShift cluster resources, and it requires the details of the SVM to be mapped to. This should be the protocol driver for the backend at the minimum. Optionally, it allows you to define how the volumes are provisioned on the storage and to set limits for the size of volumes or usage of aggregates and so on. Details concerning the definition of the Trident backends can be found here.

**Red Hat OpenShift – storage resources**

After configuring the Trident backends, the next step is to configure StorageClasses. Configure as many storage classes as there are backends, providing each storage class access to spin up volumes only on one backend. We can map the StorageClass to a particular Trident backend by using the storagePools parameter while defining the storage class. The details to define a storage class can be found here. Thus, there is a one-to-one mapping from StorageClass to Trident backend which points back to one SVM. This ensures that all storage claims via the StorageClass assigned to that project are served by the SVM dedicated to that project only.

Because storage classes are not namespaced resources, how do we ensure that storage claims to storage class of one project by pods in another namespace or project gets rejected? The answer is to use ResourceQuotas. ResourceQuotas are objects that control the total usage of resources per project. It can limit the number as well as the total amount of resources that can be consumed by objects in the project. Almost all the resources of a project can be limited using ResourceQuotas and using this efficiently can help organizations cut cost and outages due to overprovisioning or overconsumption of resources. Refer to the documentation here for more information.

For this use case, we need to limit the pods in a particular project from claiming storage from storage classes that are not dedicated to their project. To do that, we need to limit the persistent volume claims for other storage classes by setting `<storage-class-name>.storageclass.storage.k8s.io/persistentvolumeclaims` to 0. In addition, a cluster administrator must ensure that the developers in a project should not have access to modify the ResourceQuotas.

Next: Configuration.

# Configuration

For any multitenant solution, no user can have access to more cluster resources than is required. So, the entire set of resources that are to be configured as part of the multitenancy configuration is divided between cluster-admin, storage-admin, and developers working on each project.

The following table outlines the different tasks to be performed by different users:

| Role | Tasks |
|------|-------|
| **Cluster-admin** | Create projects for different applications or workloads |
| | Create ClusterRoles and RoleBindings for storage-admin |
| | Create Roles and RoleBindings for developers assigning access to specific projects |
| | [Optional] Configure projects to schedule pods on specific nodes |
| **Storage-admin** | Create SVMs on NetApp ONTAP |
| | Create Trident backends |
| | Create StorageClasses |
| | Create storage ResourceQuotas |
| **Developers** | Validate access to create or patch PVCs or pods in assigned project |
| | Validate access to create or patch PVCs or pods in another project |
| | Validate access to view or edit Projects, ResourceQuotas, and StorageClasses |

**Configuration**

**Prerequisites**

- NetApp ONTAP cluster
- Red Hat OpenShift cluster
- Trident installed on the cluster
- Admin workstation with tridentctl and oc tools installed and added to $PATH
- Admin access to ONTAP
- Cluster-admin access to OpenShift cluster
- Cluster is integrated with Identity Provider
- Identity provider is configured to efficiently distinguish between users in different teams

**Configuration: cluster-admin tasks**

The following tasks are performed by the Red Hat OpenShift cluster-admin:

1. Log into Red Hat OpenShift cluster as the cluster-admin.
2. Create two projects corresponding to different projects.

```
oc create namespace project-1
oc create namespace project-2
```

3. Create the developer role for project-1.

```
cat << EOF | oc create -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: project-1
  name: developer-project-1
rules:
  - verbs:
      - '*'
    apiGroups:
      - apps
      - batch
      - autoscaling
      - extensions
      - networking.k8s.io
      - policy
      - apps.openshift.io
      - build.openshift.io
      - image.openshift.io
      - ingress.operator.openshift.io
      - route.openshift.io
      - snapshot.storage.k8s.io
      - template.openshift.io
    resources:
      - '*'
  - verbs:
      - '*'
    apiGroups:
      - ''
    resources:
      - bindings
      - configmaps
      - endpoints
      - events
      - persistentvolumeclaims
      - pods
      - pods/log
      - pods/attach
      - podtemplates
      - replicationcontrollers
```

```
        - services
        - limitranges
        - namespaces
        - componentstatuses
        - nodes
    - verbs:
        - '*'
      apiGroups:
        - trident.netapp.io
      resources:
        - tridentsnapshots
  EOF
```

> (i) The role definition provided in this section is just an example. Developer roles must be defined based on end-user requirements.

4. Similarly, create developer roles for project-2.

5. All OpenShift and NetApp storage resources are usually managed by a storage admin. Access for storage administrators is controlled by the trident operator role that is created when Trident is installed. In addition to this, the storage admin also requires access to ResourceQuotas to control how storage is consumed.

6. Create a role for managing ResourceQuotas in all projects in the cluster to attach it to storage admin.

```
cat << EOF | oc create -f -
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: resource-quotas-role
rules:
  - verbs:
      - '*'
    apiGroups:
      - ''
    resources:
      - resourcequotas
  - verbs:
      - '*'
    apiGroups:
      - quota.openshift.io
    resources:
      - '*'
EOF
```

7. Make sure that the cluster is integrated with the organization's identity provider and that user groups are synchronized with cluster groups. The following example shows that the identity provider has been integrated with the cluster and synchronized with the user groups.

```
$ oc get groups
NAME                      USERS
ocp-netapp-storage-admins    ocp-netapp-storage-admin
ocp-project-1                ocp-project-1-user
ocp-project-2                ocp-project-2-user
```

8. Configure ClusterRoleBindings for storage admins.

```
cat << EOF | oc create -f -
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: netapp-storage-admin-trident-operator
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: ocp-netapp-storage-admins
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: trident-operator
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: netapp-storage-admin-resource-quotas-cr
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: ocp-netapp-storage-admins
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: resource-quotas-role
EOF
```

ⓘ For storage admins, two roles must be bound: trident-operator and resource-quotas.

9. Create RoleBindings for developers binding the developer-project-1 role to the corresponding group (ocp-project-1) in project-1.

```
cat << EOF | oc create -f -
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: project-1-developer
  namespace: project-1
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: ocp-project-1
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: developer-project-1
EOF
```

10. Similarly, create RoleBindings for developers binding the developer roles to the corresponding user group in project-2.

**Configuration: Storage-admin tasks**

The following resources must be configured by a storage administrator:

1. Log into the NetApp ONTAP cluster as admin.

2. Navigate to Storage > Storage VMs and click Add. Create two SVMs, one for project-1 and the other for project-2, by providing the required details. Also create a vsadmin account to manage the SVM and its resources.

# Add Storage VM                                                    ✕

STORAGE VM NAME

project-1-svm

## Access Protocol

| ✔ SMB/CIFS, NFS | iSCSI |
|---|---|

☐ Enable SMB/CIFS

☑ Enable NFS

☑ Allow NFS client access
Add at least one rule to allow NFS clients to access volumes in this storage VM. ⑦

EXPORT POLICY
Default

RULES

| Rule Index | Clients | Access Protocols | Read-Only R... | Read/Wr |
|---|---|---|---|---|
| | 10.61.181.0/24 | Any | Any | Any |

＋ Add

DEFAULT LANGUAGE ⑦

| c.utf_8 | ⌄ |
|---|---|

NETWORK INTERFACE
Use multiple network interfaces when client traffic is high.

### K8s-Ontap-01

| IP ADDRESS | SUBNET MASK | GATEWAY | BROADCAST DOMAIN |
|---|---|---|---|
| 10.61.181.224 | 24 | Add optional gateway | Default-4 ⌄ |

3. Log into the Red Hat OpenShift cluster as the storage administrator.

4. Create the backend for project-1 and map it to the SVM dedicated to the project. NetApp recommends using the SVM's vsadmin account to connect the backend to SVM instead of using the ONTAP cluster administrator.

```
cat << EOF | tridentctl -n trident create backend -f
{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "backendName": "nfs_project_1",
    "managementLIF": "172.21.224.210",
    "dataLIF": "10.61.181.224",
    "svm": "project-1-svm",
    "username": "vsadmin",
    "password": "NetApp123"
}
EOF
```

(i) We are using the ontap-nas driver for this example. Use the appropriate driver when creating the backend based on the use case.

(i) We assume that Trident is installed in the trident project.

5. Similarly create the Trident backend for project-2 and map it to the SVM dedicated to project-2.

6. Next, create the storage classes. Create the storage class for project-1 and configure it to use the storage pools from backend dedicated to project-1 by setting the storagePools parameter.

```
cat << EOF | oc create -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: project-1-sc
provisioner: csi.trident.netapp.io
parameters:
  backendType: ontap-nas
  storagePools: "nfs_project_1:.*"
EOF
```

7. Likewise, create a storage class for project-2 and configure it to use the storage pools from backend dedicated to project-2.

8. Create a ResourceQuota to restrict resources in project-1 requesting storage from storageclasses dedicated to other projects.

```
cat << EOF | oc create -f -
kind: ResourceQuota
apiVersion: v1
metadata:
  name: project-1-sc-rq
  namespace: project-1
spec:
  hard:
    project-2-sc.storageclass.storage.k8s.io/persistentvolumeclaims: 0
EOF
```

9. Similarly, create a ResourceQuota to restrict resources in project-2 requesting storage from storageclasses dedicated to other projects.

**Validation**

To validate the multitenant architecture that was configured in the previous steps, complete the following steps:

**Validate access to create PVCs or pods in assigned project**

1. Log in as ocp-project-1-user, developer in project-1.
2. Check access to create a new project.

```
oc create ns sub-project-1
```

3. Create a PVC in project-1 using the storageclass that is assigned to project-1.

```
cat << EOF | oc create -f -
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: test-pvc-project-1
  namespace: project-1
  annotations:
    trident.netapp.io/reclaimPolicy: Retain
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: project-1-sc
EOF
```

4. Check the PV associated with the PVC.

```
oc get pv
```

5. Validate that the PV and its volume is created in an SVM dedicated to project-1 on NetApp ONTAP.

```
volume show -vserver project-1-svm
```

6. Create a pod in project-1 and mount the PVC created in previous step.

```
cat << EOF | oc create -f -
kind: Pod
apiVersion: v1
metadata:
  name: test-pvc-pod
  namespace: project-1
spec:
  volumes:
    - name: test-pvc-project-1
      persistentVolumeClaim:
       claimName: test-pvc-project-1
  containers:
    - name: test-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: test-pvc-project-1
EOF
```

7. Check if the pod is running and whether it mounted the volume.

```
oc describe pods test-pvc-pod -n project-1
```

**Validate access to create PVCs or pods in another project or use resources dedicated to another project**

1. Log in as ocp-project-1-user, developer in project-1.
2. Create a PVC in project-1 using the storageclass that is assigned to project-2.

```
cat << EOF | oc create -f -
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: test-pvc-project-1-sc-2
  namespace: project-1
  annotations:
    trident.netapp.io/reclaimPolicy: Retain
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: project-2-sc
EOF
```

3. Create a PVC in project-2.

```
cat << EOF | oc create -f -
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: test-pvc-project-2-sc-1
  namespace: project-2
  annotations:
    trident.netapp.io/reclaimPolicy: Retain
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: project-1-sc
EOF
```

4. Make sure that PVCs `test-pvc-project-1-sc-2` and `test-pvc-project-2-sc-1` were not created.

```
oc get pvc -n project-1
oc get pvc -n project-2
```

5. Create a pod in project-2.

```
cat << EOF | oc create -f -
kind: Pod
apiVersion: v1
metadata:
  name: test-pvc-pod
  namespace: project-1
spec:
  containers:
    - name: test-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
EOF
```

**Validate access to view and edit Projects, ResourceQuotas, and StorageClasses**

1. Log in as ocp-project-1-user, developer in project-1.

2. Check access to create new projects.

```
oc create ns sub-project-1
```

3. Validate access to view projects.

```
oc get ns
```

4. Check if the user can view or edit ResourceQuotas in project-1.

```
oc get resourcequotas -n project-1
oc edit resourcequotas project-1-sc-rq -n project-1
```

5. Validate that the user has access to view the storageclasses.

```
oc get sc
```

6. Check access to describe the storageclasses.

7. Validate the user's access to edit the storageclasses.

```
oc edit sc project-1-sc
```

**Scaling: Adding more projects**

In a multitenant configuration, adding new projects with storage resources requires additional configuration to make sure that multitenancy is not violated. For adding more projects in a multitenant cluster, complete the following steps:

1. Log into the NetApp ONTAP cluster as a storage admin.

2. Navigate to `Storage` → `Storage VMs` and click `Add`. Create a new SVM dedicated to project-3. Also create a vsadmin account to manage the SVM and its resources.

# Add Storage VM

✕

STORAGE VM NAME

project-3-svm

## Access Protocol

| ✓ SMB/CIFS, NFS | iSCSI |

☐ Enable SMB/CIFS

☑ Enable NFS

    ☑ Allow NFS client access
    Add at least one rule to allow NFS clients to access volumes in this storage VM. ⑦

EXPORT POLICY
Default

RULES

| Rule Index | Clients | Access Protocols | Read-Only R... | Read/Wr |
|---|---|---|---|---|
|  | 10.61.181.0/24 | Any | Any | Any |

➕ Add

DEFAULT LANGUAGE ⑦

c.utf_8 ⌄

NETWORK INTERFACE
Use multiple network interfaces when client traffic is high.

K8s-Ontap-01

| IP ADDRESS | SUBNET MASK | GATEWAY | BROADCAST DOMAIN |
|---|---|---|---|
| 10.61.181.228 | 24 | Add optional gateway | Default-4 ⌄ |

3. Log into the Red Hat OpenShift cluster as cluster admin.

4. Create a new project.

```
oc create ns project-3
```

5. Make sure that the user group for project-3 is created on IdP and synchronized with the OpenShift cluster.

```
oc get groups
```

6. Create the developer role for project-3.

```
cat << EOF | oc create -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: project-3
  name: developer-project-3
rules:
  - verbs:
      - '*'
    apiGroups:
      - apps
      - batch
      - autoscaling
      - extensions
      - networking.k8s.io
      - policy
      - apps.openshift.io
      - build.openshift.io
      - image.openshift.io
      - ingress.operator.openshift.io
      - route.openshift.io
      - snapshot.storage.k8s.io
      - template.openshift.io
    resources:
      - '*'
  - verbs:
      - '*'
    apiGroups:
      - ''
    resources:
      - bindings
      - configmaps
      - endpoints
      - events
      - persistentvolumeclaims
      - pods
      - pods/log
      - pods/attach
      - podtemplates
      - replicationcontrollers
      - services
```

```
        - limitranges
        - namespaces
        - componentstatuses
        - nodes
   - verbs:
        - '*'
     apiGroups:
        - trident.netapp.io
     resources:
        - tridentsnapshots
EOF
```

> (i) The role definition provided in this section is just an example. The developer role must be defined based on the end-user requirements.

7. Create RoleBinding for developers in project-3 binding the developer-project-3 role to the corresponding group (ocp-project-3) in project-3.

```
cat << EOF | oc create -f -
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: project-3-developer
  namespace: project-3
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: ocp-project-3
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: developer-project-3
EOF
```

8. Login to the Red Hat OpenShift cluster as storage admin

9. Create a Trident backend and map it to the SVM dedicated to project-3. NetApp recommends using the SVM's vsadmin account to connect the backend to the SVM instead of using the ONTAP cluster administrator.

```
cat << EOF | tridentctl -n trident create backend -f
{
    "version": 1,
    "storageDriverName": "ontap-nas",
    "backendName": "nfs_project_3",
    "managementLIF": "172.21.224.210",
    "dataLIF": "10.61.181.228",
    "svm": "project-3-svm",
    "username": "vsadmin",
    "password": "NetApp!23"
}
EOF
```

(i)    We are using the ontap-nas driver for this example. Use the appropriate driver for creating the backend based on the use-case.

(i)    We assume that Trident is installed in the trident project.

10. Create the storage class for project-3 and configure it to use the storage pools from backend dedicated to project-3.

```
cat << EOF | oc create -f -
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: project-3-sc
provisioner: csi.trident.netapp.io
parameters:
  backendType: ontap-nas
  storagePools: "nfs_project_3:.*"
EOF
```

11. Create a ResourceQuota to restrict resources in project-3 requesting storage from storageclasses dedicated to other projects.

```
cat << EOF | oc create -f -
kind: ResourceQuota
apiVersion: v1
metadata:
  name: project-3-sc-rq
  namespace: project-3
spec:
  hard:
    project-1-sc.storageclass.storage.k8s.io/persistentvolumeclaims: 0
    project-2-sc.storageclass.storage.k8s.io/persistentvolumeclaims: 0
EOF
```

12. Patch the ResourceQuotas in other projects to restrict resources in those projects from accessing storage from the storageclass dedicated to project-3.

```
oc patch resourcequotas project-1-sc-rq -n project-1 --patch
'{"spec":{"hard":{ "project-3-
sc.storageclass.storage.k8s.io/persistentvolumeclaims": 0}}}'
oc patch resourcequotas project-2-sc-rq -n project-2 --patch
'{"spec":{"hard":{ "project-3-
sc.storageclass.storage.k8s.io/persistentvolumeclaims": 0}}}'
```