IBM Spectrum Scale

Secure Data Sharing to Applications running in OpenShift

Gero Schmidt (gerosch@de.ibm.com)
Big Data & Analytics Solutions,
IBM Spectrum Scale Development, Germany

IBM

Disclaimer

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision. The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

IBM reserves the right to change product specifications and offerings at any time without notice. This publication could include technical inaccuracies or typographical errors. References herein to IBM products and services do not imply that IBM intends to make them available in all countries.

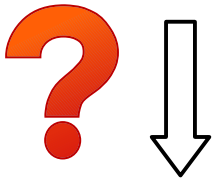
Agenda

How to *provide* & *share* access to *existing data* in IBM Spectrum Scale?

1) ***Accessing*** *existing data* in Spectrum Scale on OpenShift
   - OpenShift (with CNSA & CSI): ***Volume Provisioning***
2) ***Sharing*** access to data in Spectrum Scale on OpenShift
   - OpenShift (with CNSA & CSI): ***Security Context***
3) ***Demo***

How to *provide & share* access to *existing* data in IBM Spectrum Scale

RED HAT OPENSHIFT Container Platform



IBM Spectrum Scale

(1) *Provide access* to data in IBM Spectrum Scale on OpenShift

directories & files

IBM Spectrum Scale
/mnt/fs1/training-data

IBM Spectrum Scale

persistent volumes (PVs)

CNSA CSI

PV → PVC → Pod

Namespace

RED HAT OPENSHIFT Container Platform

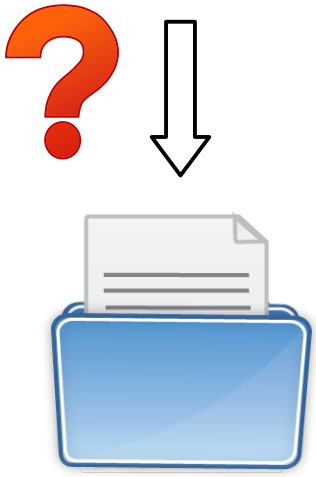
uid, gid, file permissions

namespaces, users, SELinux, container uid/gid

(2) Configure OpenShift *Security Context* for *shared* data access

How to *provide & share* access to *existing* data in IBM Spectrum Scale

RED HAT OPENSHIFT Container Platform

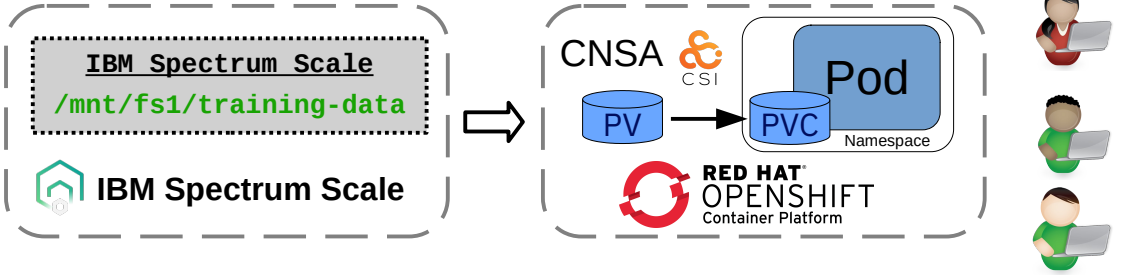


IBM Spectrum Scale

(1) *Provide access* to data in IBM Spectrum Scale on OpenShift

directories & files

persistent volumes (PVs)

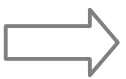


Kubernetes Storage Concepts – Persistent Volumes (PVs)

Dynamic Provisioning

Admin

Admin creates ***StorageClass*** in OpenShift/Kubernetes



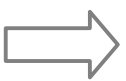
User

User claims volume (PV) from ***StorageClass*** through ***Persistent Volume Claim*** **(PVC)** (*self-service provisioning*)

Static Provisioning

Admin

Admin provisions static ***PVs*** in OpenShift/Kubernetes



User

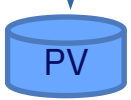
User claims volume (PV) from pool of pre-provisioned **PVs** through ***Persistent Volume Claim*** **(PVC)**

OpenShift: *Dynamic Provisioning (fileset based)*

```
$ oc apply -f scale-sc.yaml
$ cat scale-sc.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
   name: scale-sc
provisioner: spectrumscale.
parameters:
  volBackendFs: "fs0"
  clusterId: "6174907451206
reclaimPolicy: Delete
```

```
$ cat myapp-pvc.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myapp-pvc
spec:
  storageClassName: "scale-sc"
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
```

created ↕ bound



PV (Persistent Volume)

Admin

```
$ cat myapp.yaml
kind: Pod
apiVersion: v1
metadata:
  name: myapp
spec:
  containers:
    - name: demo
      image: alpine:3.8
      command: [ "/bin/sh", "-c", "--" ]
      args: [ while true; …; done;" ]
      volumeMounts:
        - name: vol01
          mountPath: "/data"
  volumes:
    - name: vol01
      persistentVolumeClaim:
        claimName: myapp-pvc
```

- PV is *auto-created* upon user request
- New *empty* volume for the user
- Self-service

User

OpenShift: *Static Provisioning* (basic)

```
$ oc apply -f scale-dev_pv.yaml
$ cat scale-dev_pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: scale-data-pv01
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  csi:
    driver: spectrumscale.csi
    volumeHandle: \
    "clusterID;FSUID;path=/mn
```

User

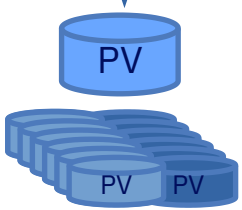
```
$ cat myapp-pvc.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myapp-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
```

```
$ cat myapp.yaml
kind: Pod
apiVersion: v1
metadata:
  name: myapp
spec:
  containers:
    - name: de
      image: alpine:3.8
      command: [ "/bin/sh", "-c", "--" ]
      args: [ while true; …; done;" ]
      volumeMounts:
        - name: vol01
          mountPath: "/data"
  volumes:
    - name: vol01
      persistentVolumeClaim:
        claimName: myapp-pvc
```

- ***Any*** available PV from *pool* is bound (best match)
- No *default StorageClass*
- New *empty* volume for the user
- Self-service alike

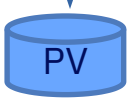
created

Admin



pool

bound



PV (Persistent Volume)

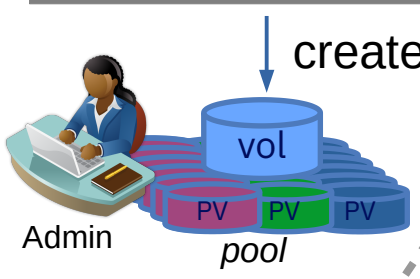
OpenShift: Advanced *Static Provisioning* (*labels & selector*)

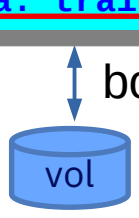
```
$ oc apply -f scale-data-pv01.yaml
$ cat scale-data-pv01.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: scale-data-pv01
  labels:
    dept: ds
    data: train
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  csi:
    driver: spectrumscale.csi
    volumeHandle: \
    "clusterID;FSUID;path=/mn
```

```
$ cat myapp-pvc.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myapp-pvc
spec:
  storageClassName: ""
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  selector:
    matchLabels:
      dept: ds
      data: train
```

```
$ cat myapp.yaml
kind: Pod
apiVersion: v1
metadata:
  name: myapp
spec:
  containers:
    - name: demo
      image: alpi
      command: [ "/bin/sh", "-c", "--" ]
      args: [ while true; …; done;" ]
      volumeMounts:
        - name: vol01
          mountPath: "/data"
  volumes:
    - name: vol01
      persistentVolumeClaim:
        claimName: myapp-pvc
```

- ***Labels*** & ***Selector*** allow *users* to *request* a *specific* PV from pool
- Default *StorageClass* ok
- New volume for user w/access to *existing* data
- Self-service alike





PV (Persistent Volume)

OpenShift: Advanced *Static Provisioning* (*labels*/*selector* + *storage class*)

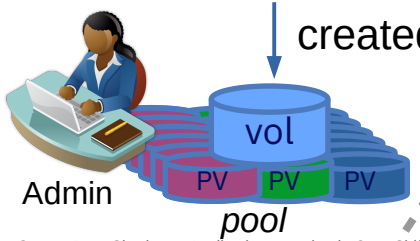
```
$ oc apply -f scale-data-pv01
$ cat scale-data-pv01.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: scale-data-pv01
  labels:
    dept: ds
    data: train
spec:
  storageClassName: static
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  csi:
    driver: spectrumscale.csi
    volumeHandle: \
    "clusterID;FSUID;path=/mnt/fsi/train
```

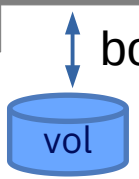
```
$ cat myapp-pvc.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myapp-pvc
spec:
  storageClassName: static
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  selector:
    matchLabels:
      dept: ds
      data: train
```

```
$ cat myapp.yaml
kind: Pod
apiVersion: v1
metadata:
  name: myapp
spec:
  containers:
    - name: dem
      image: al
      command:
      args: [ w
      volumeMounts:
        - name: vol01
          mountPath: "/data"
  volumes:
    - name: vol01
      persistentVolumeClaim:
        claimName: myapp-pvc
```

- ***Labels*** & ***Selector*** allow *users* to *request* a *specific* PV from pool
- *Fake* ***storageClassName*** improves management (use of *resource quotas*)
- Default *StorageClass* ok
- New volume for user w/access to *existing* data
- Self-service alike

User





PV (Persistent Volume)

OpenShift: Advanced *Static Provisioning* (*claimRef*)

```
$ oc apply -f scale-data-pv01.yaml
$ cat scale-data-pv01.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: scale-data-pv01
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  claimRef:
    name: myapp-pvc
    namespace: my-namespace
  csi:
    driver: spectrumscale.csi.
    volumeHandle: \
    "clusterID;FSUID;path=/mnt/fs1/mydata"
```

```
$ cat myapp-pvc.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myapp-pvc
  namespace: my-namespace
spec:
  storageClassName: ""
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
```

```
$ cat myapp.yaml
kind: Pod
apiVersion: v1
metadata:
  name: myapp
spec:
  containers:
    - name: de
      image: a
      command:
      args: [ while true; …; done;" ]
      volumeMounts:
        - name: vol01
          mountPath: "/data"
  volumes:
    - name: vol01
      persistentVolumeClaim:
        claimName: myapp-pvc
```

- ***claimRef:*** PV is *reserved*, can only be bound to a *specific* PVC based on ***PVC name*** & ***namespace***
- Default *StorageClass* ok
- New volume for *user* with access to *selected* data
- Provisioned *on request*

User

Admin

created

(reserved)

bound

- *PVC name*
- *Namespace*

PV (Persistent Volume)

Provisioning PVs for *shared data access* across *different namespaces*

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: home-ns1
spec:
  capaci
    stor
  access
    - Re
  csi:
    driv
    volu
"8358383
  claimR
    name
    name
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: home-ns2
spec:
  capaci
    stor
  accessl
    - Re
  csi:
    driv
    volu
"8358383
  claimR
    name
    name
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: home-ns3
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  csi:
    driver: spectrumscale.csi.ibm.com
    volumeHandle:
"835838342966509310;099B6A7A:5EB99721;path=/mnt/fs1/ocp-home"
  claimRef:
    name: home
    namespace: nmspc3
```

PV

PV

PV

Admin

user #1

user #2

user #3

Pod Pod namespace #1 (project #1) "*nmspc1*" PVC

Pod Pod namespace #2 (project #2) "*nmspc2*" PVC

Pod Pod namespace #3 (project #3) "*nmspc3*" PVC

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
```

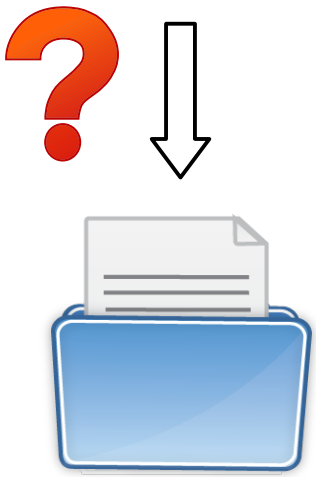
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: home
spec:
  storageClassName: ""
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
```

OpenShift: *Static Provisioning* use cases

- **Basic static volume provisioning:**
  - Provide a ***pool of empty persistent volumes*** at a chosen path in IBM Spectrum Scale to be claimed by users. PV is only matched based on ***size*** and ***access mode*** of a user's PVC request.
- **Advanced static volume provisioning (1): `labels/selector & storageClass: ""`**
  - Provide ***access to existing data*** in a ***shared*** directory in IBM Spectrum Scale to ***multiple users***. Users can claim a pre-provisioned PV with specific data from a *pool* of PVs through ***labels*** that characterize the PV and its data. A PVC is bound to a PV based on its *labels* and *access mode*. Multiple PVs need to be created if claimed by different user *namespaces*. Use of an empty *storageClassName* in the PVC is mandatory to *skip dynamic provisioning* through a default storage class.
- **Advanced static volume provisioning (2): `labels/selector & storageClass: "static"`**
  - Provide ***access to existing data*** in a ***shared*** directory in IBM Spectrum Scale to ***multiple users***. Users can claim a pre-provisioned PV with specific data from a *pool of PVs* through ***labels*** and a (fake) ***storage class***. The *labels* characterize the PV and its data. The fake ***storageClassName*** like "*static*" improves volume management, allows to *restrict access* by *namespace* through *storage resource quotas*, and *skips dynamic provisioning* through a default storage class. A PVC is bound to a PV based on its *labels*, *storage class* and *access mode*. Multiple PVs need to be created if claimed by different user *namespaces*.
- **Advanced static volume provisioning (3): `claimRef: PVC-name & namespace` *`(preferred, selective control)`***
  - Provide ***access to existing data*** in a ***selected*** directory in IBM Spectrum Scale to a ***selected user*** (i.e. ***namespace***) by enforcing that the provisioned PV can only be bound to a specific PVC request in the specified ***namespace*** with the specified PVC ***name***. For ***shared access*** the admin needs to create one PV per user *namespace* on request.

How to *provide & share* access to *existing* data in IBM Spectrum Scale

RED HAT OPENSHIFT Container Platform



IBM Spectrum Scale

(2) Configure OpenShift *Security Context* for *shared* data access

directories & files

persistent volumes (PVs)

POSIX

IBM Spectrum Scale
/mnt/fs1/training-data

IBM Spectrum Scale

CNSA & CSI

PV → PVC

Pod

Namespace

RED HAT OPENSHIFT Container Platform

uid, gid, file permissions

namespaces, users, SELinux, container uid/gid

OpenShift: *Security Context Constraints* (SCC) & Pod *security context*

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: restricted
groups:
- system:authenticated
fsGroup:
  type: MustRunAs
runAsUser:
  type: MustRunAsRange
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
[...]
```

SGID

UID

s0:c27,c24

addl GIDs

Admin

Pre-allocated default values from *namespace*

```
kind: Namespace
metadata:
  annotations:
    openshift.io/requester: user1
    openshift.io/sa.scc.mcs: s0:c27,c24
    openshift.io/sa.scc.supplemental-groups: 1000750000/10000
    openshift.io/sa.scc.uid-range: 1000750000/10000
  name: nmspc1
```

Admin

Volumes: **/data** IBM Spectrum Scale CSI volume
/data2 *EmptyDir {}*

User process in container (uid/gid):

File permissions & SELinux label on mounted volume:

```
# oc rsh test-pod
sh-4.4$ id
uid=1000750000(1000750000) gid=0(root) groups=0(root),1000750000

sh-4.4$ ls -dlZ /data*
drwxrwxr-x. 5 root  root system_u:object_r:container_file_t:s0:c27,c24 4096 Sep 27 13:16 /data
drwxrwsrwx. 2 root 1000750000 system_u:object_r:container_file_t:s0:c27,c24 Sep 27 13:16 /data2
```

SGID

SGID

Non-privileged User

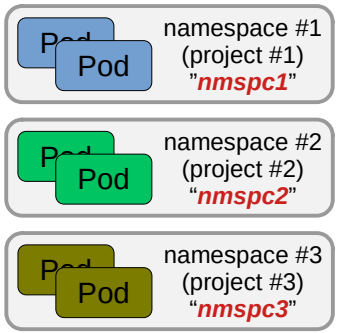
OpenShift: A user can have multiple *namespaces* (aka *projects*)

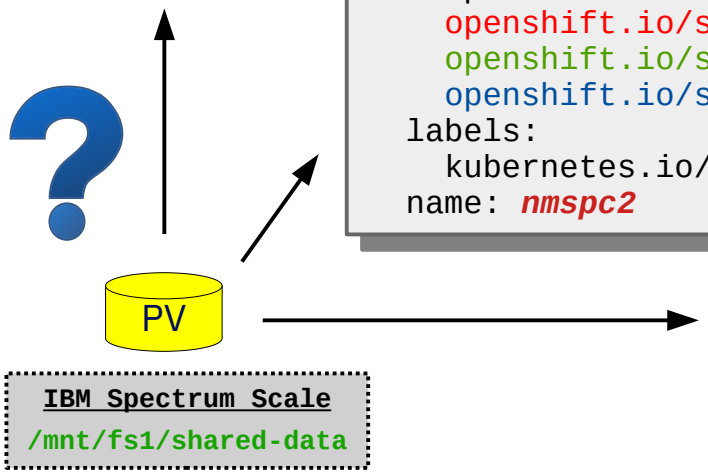
```
kind: Namespace
metadata:
  annotations:
    openshift.io/requester: user1
    openshift.io/sa.scc.mcs: s0:c27,c24
    openshift.io/sa.scc.supplemental-groups: 1000750000/10000
    openshift.io/sa.scc.uid-range: 1000750000/10000
  labels:
    kubernetes.i
  name: nmspc1
```

```
kind: Namespace
metadata:
  annotations:
    openshift.io/requester: user1
    openshift.io/sa.scc.mcs: s0:c28,c7
    openshift.io/sa.scc.supplemental-groups: 1000770000/10000
    openshift.io/sa.scc.uid-range: 1000770000/10000
  labels:
    kubernetes.io/me
  name: nmspc2
```

```
kind: Namespace
metadata:
  annotations:
    openshift.io/requester: user1
    openshift.io/sa.scc.mcs: s0:c29,c4
    openshift.io/sa.scc.supplemental-groups: 1000820000/10000
    openshift.io/sa.scc.uid-range: 1000820000/10000
  labels:
    kubernetes.io/metadata.name: nmspc3
  name: nmspc3
```

User



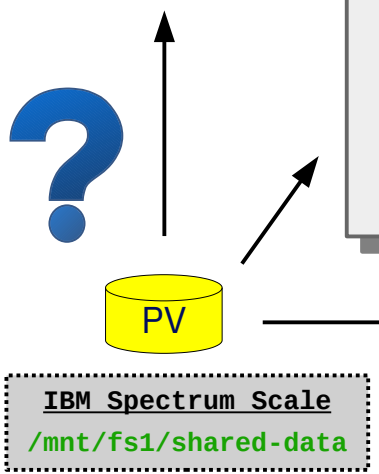


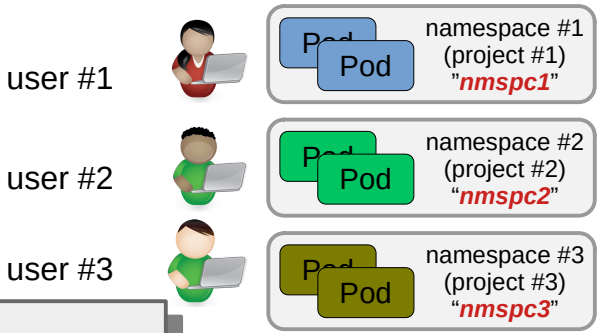
OpenShift: Multiple users have their own *namespaces* (aka *projects*)

```
kind: Namespace
metadata:
  annotations:
    openshift.io/requester: user1
    openshift.io/sa.scc.mcs: s0:c27,c24
    openshift.io/sa.scc.supplemental-groups: 1000750000/10000
    openshift.io/sa.scc.uid-range: 1000750000/10000
  labels:
    kubernetes.i
  name: nmspc1
```

```
kind: Namespace
metadata:
  annotations:
    openshift.io/requester: user2
    openshift.io/sa.scc.mcs: s0:c28,c7
    openshift.io/sa.scc.supplemental-groups: 1000770000/10000
    openshift.io/sa.scc.uid-range: 1000770000/10000
  labels:
    kubernetes.io/me
  name: nmspc2
```

```
kind: Namespace
metadata:
  annotations:
    openshift.io/requester: user3
    openshift.io/sa.scc.mcs: s0:c29,c4
    openshift.io/sa.scc.supplemental-groups: 1000820000/10000
    openshift.io/sa.scc.uid-range: 1000820000/10000
  labels:
    kubernetes.io/metadata.name: nmspc3
  name: nmspc3
```





OpenShift: Multiple users have their own *namespaces* (aka *projects*)

```
kind: Namespace
metadata:
  annotations:
    openshift.io/requester: user1
    openshift.io/sa.scc.mcs: s0:c27,c24
    openshift.io/sa.scc.supplemental-groups: 1000750000/10000
    openshift.io/sa.scc.uid-range: 1000750000/10000
  labels:
    kubernetes.io/metadata.name: nmspc1
```

```
kind: Namespace
metadata:
  annotations:
    openshift.io/requester: user2
    openshift.io/sa.scc.mcs: s0:c28,c7
    openshift.io/sa.scc.supplemental-groups: 1000770000/10000
    openshift.io/sa.scc.uid-range: 1000770000/10000
  labels:
    kubernetes.io/metadata.name: nmspc2
```

```
kind: Namespace
metadata:
  annotations:
    openshift.io/requester: user3
    openshift.io/sa.scc.mcs: s0:c29,c4
    openshift.io/sa.scc.supplemental-groups: 1000820000/10000
    openshift.io/sa.scc.uid-range: 1000820000/10000
  labels:
    kubernetes.io/metadata.name: nmspc3
```

Pod
PVC

user #1
user #2
user #3

Pod
Pod
namespace #1 (project #1) "*nmspc1*"

Pod
Pod
namespace #2 (project #2) "*nmspc2*"

Pod
Pod
namespace #3 (project #3) "*nmspc3*"

?

SELinux s0:c27,c24

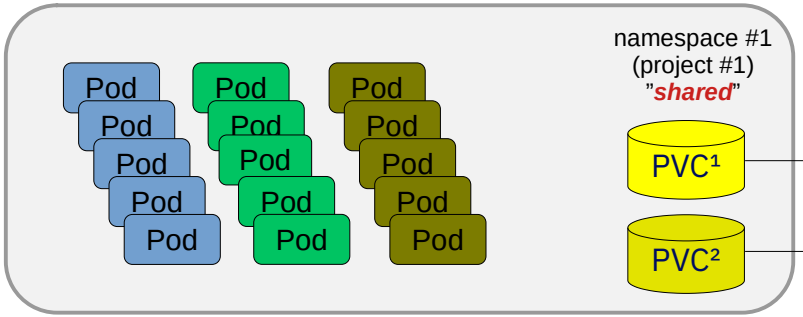
SELinux s0:c28,c7

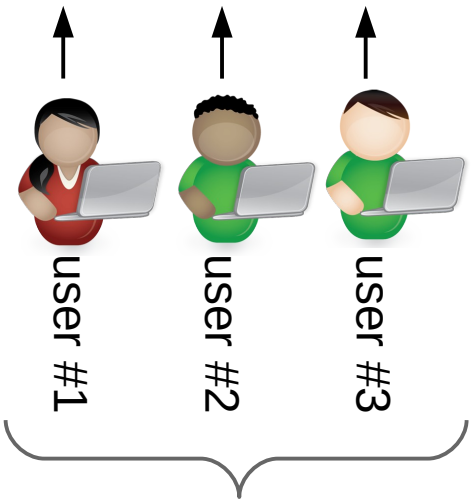
SELinux s0:c29,c4

PV
PV
PV

IBM Spectrum Scale
/mnt/fs1/shared-data
s0:c29,c4

Using a *shared namespace* for collaboration on *shared data*





shared namespace

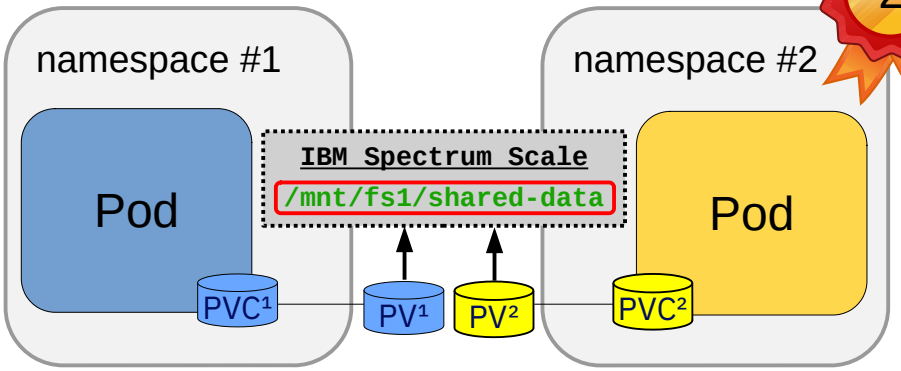
```
kind: Namespace
metadata:
  annotations:
    openshift.io/requester: ocpadmin
    openshift.io/sa.scc.mcs: s0:c27,c4
    openshift.io/sa.scc.supplemental-groups: 1000710000/10000
    openshift.io/sa.scc.uid-range: 1000710000/10000
  labels:
    kubernetes.io/metadata.name: shared
  name: shared
```


```
$ oc adm policy add-role-to-user admin|edit|view user1 -n namespace
```

Sharing data access across *different namespaces* (w/o SCCs)

Namespace #1:

```
$ oc get ns mynamespace1 -o yaml
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    openshift.io/sa.scc.mcs: s0:c26,c0
    openshift.io/sa.scc.supplemental-groups: 1000650000/10000
    openshift.io/sa.scc.uid-range: 1000650000/10000
  name: mynamespace1
```



Namespace #2:

```
$ oc get ns mynamespace2 -o yaml
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    openshift.io/sa.scc.mcs: s0:c27,c19
    openshift.io/sa.scc.supplemental-groups: 1000670000/10000
    openshift.io/sa.scc.uid-range: 1000670000/10000
  name: mynamespace2
```

Editing the *namespace* annotations requires a ***privileged*** user!

Set SELinux context on *namespace #2* ...

```
$ oc edit ns mynamespace2
namespace/mynamespace2 edited

$ oc get ns mynamespace2 -o yaml
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    openshift.io/sa.scc.mcs: s0:c26,c0
    openshift.io/sa.scc.supplemental-groups: 1000670000/10000
    openshift.io/sa.scc.uid-range: 1000670000/10000
  name: mynamespace2
```

Admin

… for *shared data access* in *namespace #2*

OpenShift: *Custom SCC* & Pod/Container *securityContext*

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: shared-scc
runAsUser:
  type: MustRunAsRange
  uidRangeMin: 5000
  uidRangeMax: 5499
seLinuxContext:
  type: MustRunAs
  seLinuxOptions:
    level: "s0:c26,c0"
fsGroup:
  type: MustRunAs
  ranges:
  - min: 5000
    max: 6000
supplementalGroups:
  type: MustRunAs
  ranges:
  - min: 5000
    max: 6000
```

UID

s0:c26,c0

SGID

addl GIDs

Admin

Custom *securityContext* in pod/container

```
spec:
  securityContext:
    runAsUser: 5001
    runAsGroup: 5001
    seLinuxOptions:
      level: "s0:c26,c0"
    fsGroup: 5500
    supplementalGroups: [5002, 5003]
```

Non-privileged User

```
/data:
drwxr-xr-x. root root .
drwxr-xr-x. root root ..
drwxrwxrwx. root root scratch
drwxrwx---. root 5500 shared
drwxr-x---. 5001 5001 user1
drwxr-x---. 5002 5002 user2
drwxr-x---. 5003 5003 user3
drwx------. 5004 5004 user4
```

Volumes: **/data** IBM Spectrum Scale CSI volume
/data2 *EmptyDir {}*

User process in container (uid/gid):

File permissions on mounted volumes:

```
# oc rsh test-pod
sh-4.4$ id
uid=5001(5001) gid=5001 groups=5001,5002,5003,5500

sh-4.4$ ls -dlZ /data*
drwxr-xr-x. 8 root root system_u:object_r:container_file_t:s0:c0,c26 4096 Sep 30 08:32 /data
drwxrwsrwx. 2 root 5500 system_u:object_r:container_file_t:s0:c0,c26    6 Sep 30 08:32 /data2
```

Non-privileged User

SGID

SGID

Using *SCCs* to *share data access: service account (role & rolebinding)*

```yaml
apiVersion: v1
kind: SecurityContextConstraints
metadata:
  name: shared-scc
allowPrivilegedContainer: false
runAsUser:
  type: MustRunAsRange
  uidRangeMin: 5000
  uidRangeMax: 5499
seLinuxContext:
  type: MustRunAs
  seLinuxOptions:
    level: "s0:c26,c0"
fsGroup:
  type: MustRunAs
  ranges:
  - min: 5000
    max: 6000
supplementalGroups:
  type: MustRunAs
  ranges:
  - min: 5000
    max: 6000
```

Admin

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: shared
  namespace: target-namespace1
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: use-shared-scc
  namespace: target-namespace1
rules:
- apiGroups:
  - security.openshift.io
  resourceNames:
  - "shared-scc"
  resources:
  - securitycontextconstraints
  verbs:
  - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: use-shared-scc
  namespace: target-namespace1
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: use-shared-scc
subjects:
  - kind: ServiceAccount
    name: shared
    namespace: target-namespace1
```

Admin

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  containers:
    - name: test
      image: ...
      securityContext:
        runAsUser: 5001
        runAsGroup: 5001
      command: [ "/bin/sh", "-c", "--" ]
      args: [ "... " ]
      volumeMounts:
        - name: vol1
          mountPath: "/data"
  serviceAccountName: shared
  securityContext:
    fsGroup: 5500
    supplementalGroups: [5002, 5003]
  volumes:
    - name: vol1
      persistentVolumeClaim:
        claimName: home
```

Using *SCCs* to *share data access:* OpenShift *users & groups*

```
apiVersion: v1
kind: SecurityContextConstraints
metadata:
  name: shared-scc
allowPrivilegedContainer: false
runAsUser:
  type: MustRunAsRange
  uidRangeMin: 5000
  uidRangeMax: 5499
seLinuxContext:
  type: MustRunAs
  seLinuxOptions:
    level: "s0:c26,c0"
fsGroup:
  type: MustRunAs
  ranges:
  - min: 5000
    max: 6000
supplementalGroups:
  type: MustRunAs
  ranges:
  - min: 5000
    max: 6000
```

Admin

```
# oc adm groups new homegrp

# oc adm groups add-users \
  homegrp user1 user2

# oc adm policy \
  add-scc-to-group \
  shared-scc homegrp

# oc adm policy \
  add-scc-to-user \
  shared-scc user3
```

Admin

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  containers:
    - name: test
      image: ...
      securityContext:
        runAsUser: 5001
        runAsGroup: 5001
      command: [ "/bin/sh", "-c", "--" ]
      args: [ "... " ]
      volumeMounts:
        - name: vol1
          mountPath: "/data"
  securityContext:
    fsGroup: 5500
    supplementalGroups: [5002, 5003]
  volumes:
    - name: vol1
      persistentVolumeClaim:
        claimName: home
```

Using *SCCs* to *share data access:* OpenShift *users & groups*


Admin

Assign OpenShift *users* to *groups*

```
# oc adm groups new my-group

# oc adm groups add-users my-group user1 user2

# oc get groups
NAME      USERS
my-group  user1, user2

# oc delete group my-group
```

Check user/group bindings to SCC

```
$ oc describe clusterrolebinding.rbac \
    | grep -A7 scc:my-scc
Name:         system:openshift:scc:my-scc
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  system:openshift:scc:shared-scc
Subjects:
  Kind  Name     Namespace
  ----  ----     ---------
  User  user3
  Group my-group
```

Assign SSCs to OpenShift *users* or *groups*

```
# oc adm policy add-scc-to-group my-scc my-group
# oc adm policy remove-scc-from-group my-scc my-group

# oc adm policy add-scc-to-user  my-scc user3
# oc adm policy remove-scc-from-user  my-scc user3
```

Further references: *Syncing LDAP groups (https://docs.openshift.com/container-platform/4.10/authentication/ldap-syncing.html)*

Using *SCCs* to *share data access:* Admission process

```
apiVersion: v1
kind: SecurityContextConstraints
metadata:
  name: shared-scc
allowPrivilegedContainer: false
runAsUser:
  type: MustRunAsRange
  uidRangeMin: 5000
  uidRangeMax: 5499
seLinuxContext:
  type: MustRunAs
  seLinuxOptions:
    level: "s0:c26,c0"
fsGroup:
  type: MustRunAs
  ranges:
  - min: 5000
    max: 6000
supplementalGroups:
  type: MustRunAs
  ranges:
  - min: 5000
    max: 6000
```

Admin

Pod

SCC ↔ securityContext

Admission process evaluates ***all involved SCCs*** from

- ***user***
- ***group***
- ***service account***

and selects ***one*** that best matches the requested ***pod securityContext*** or rejects the pod if no match is found.

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  containers:
    - name: test
      image: ...
      securityContext:
        runAsUser: 5001
        runAsGroup: 5001
        seLinuxOptions:
          level: "s0:c26,c0"
      command: [ "/bin/sh", "-c", "--" ]
      args: [ "... " ]
      volumeMounts:
        - name: vol1
          mountPath: "/data"
  securityContext:
    fsGroup: 5500
    supplementalGroups: [5002, 5003]
  volumes:
    - name: vol1
      persistentVolumeClaim:
        claimName: home
```

user with access to higher privileged SCCs (e.g SCC "anyuid")

Using SCCs to *share data access*: Debugging

Custom SCC with *service account*

```
# oc get pod test -o yaml | grep serviceAccount:
  serviceAccount: shared
# oc get pod test -o yaml | grep scc
    openshift.io/scc: shared-scc
```

Custom SCC with *users & groups*

```
# oc get pod test -o yaml | grep serviceAccount:
  serviceAccount: default
# oc get pod test -o yaml | grep scc
    openshift.io/scc: shared-scc
```

```
# oc apply -f pod.yaml
Error from server (Forbidden): error when creating "pod.yaml": pods "test" is forbidden:
unable to validate against any security context constraint:
[
provider "anyuid": Forbidden: not usable by user or serviceaccount,
provider restricted:
  .spec.securityContext.fsGroup: Invalid value: []int64{5500}: 5500 is not an allowed group,
  spec.containers[0].securityContext.runAsUser: Invalid value: 5001: must be in ranges: [1000750000, 1000759999],
provider "shared-scc": Forbidden: not usable by user or serviceaccount,
provider "nonroot": Forbidden: not usable by user or serviceaccount,
provider "hostmount-anyuid": Forbidden: not usable by user or serviceaccount,
provider "machine-api-termination-handler": Forbidden: not usable by user or serviceaccount,
provider "hostnetwork": Forbidden: not usable by user or serviceaccount,
provider "hostaccess": Forbidden: not usable by user or serviceaccount,
provider "spectrum-scale-csiaccess": Forbidden: not usable by user or serviceaccount,
provider "node-exporter": Forbidden: not usable by user or serviceaccount,
provider "ibm-spectrum-scale-privileged": Forbidden: not usable by user or serviceaccount,
provider "privileged": Forbidden: not usable by user or serviceaccount
]
```

Summary – *What did we learn?*

(I) ***Advanced static volume provisioning*** → provide access to ***existing data*** in IBM Spectrum Scale

- ***labels/selector*** → *any* user can claim from pool, self-service, no control over namespace/SELinux context
- ***claimRef*** → *selective control* of PV binding by PVC *name* & *namespace* (ensure proper SELinux context)

(II) Set proper **OpenShift *security context*** to securely provide ***shared data access***

(1) Shared data access using a *shared namespace* for multiple users

→ Collaboration (everyone has access to everything; define *roles* as *view/edit/admin*)

(2) Shared data access by *modifying namespace annotation* (SELinux MCS label)

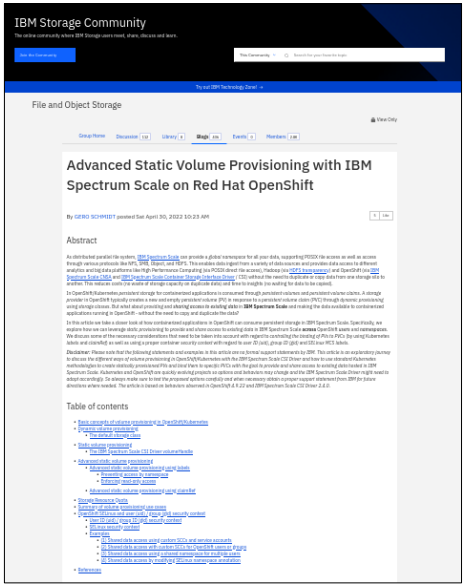
→ E.g. same user/app on different *sites* accessing same data

(3) Shared data access using *custom SCCs* and *service accounts* → Application deployments

(4) Shared data access with *custom SCCs* for OpenShift *users* or *groups* → Developers w/shared data

- Need to trust the user! User can do harm when not applying the correct SELinux MCS label by error.
- Requires careful planning of custom SCCs with regard to *uid/gid* permissions and *SELinux MCS labels*.
- SElinux context *may* become less of an issue with future changes or options in Kubernetes/CSI driver.

References



Blog post:
https://community.ibm.com/community/user/storage/blogs/gero-schmidt1/2022/04/01/advanced-static-volume-provisioning-on-ocp

Pdf:
https://ibm.box.com/v/AdvancedStaticProvisioning

Demo

(1) ***Static volume provisioning*** for

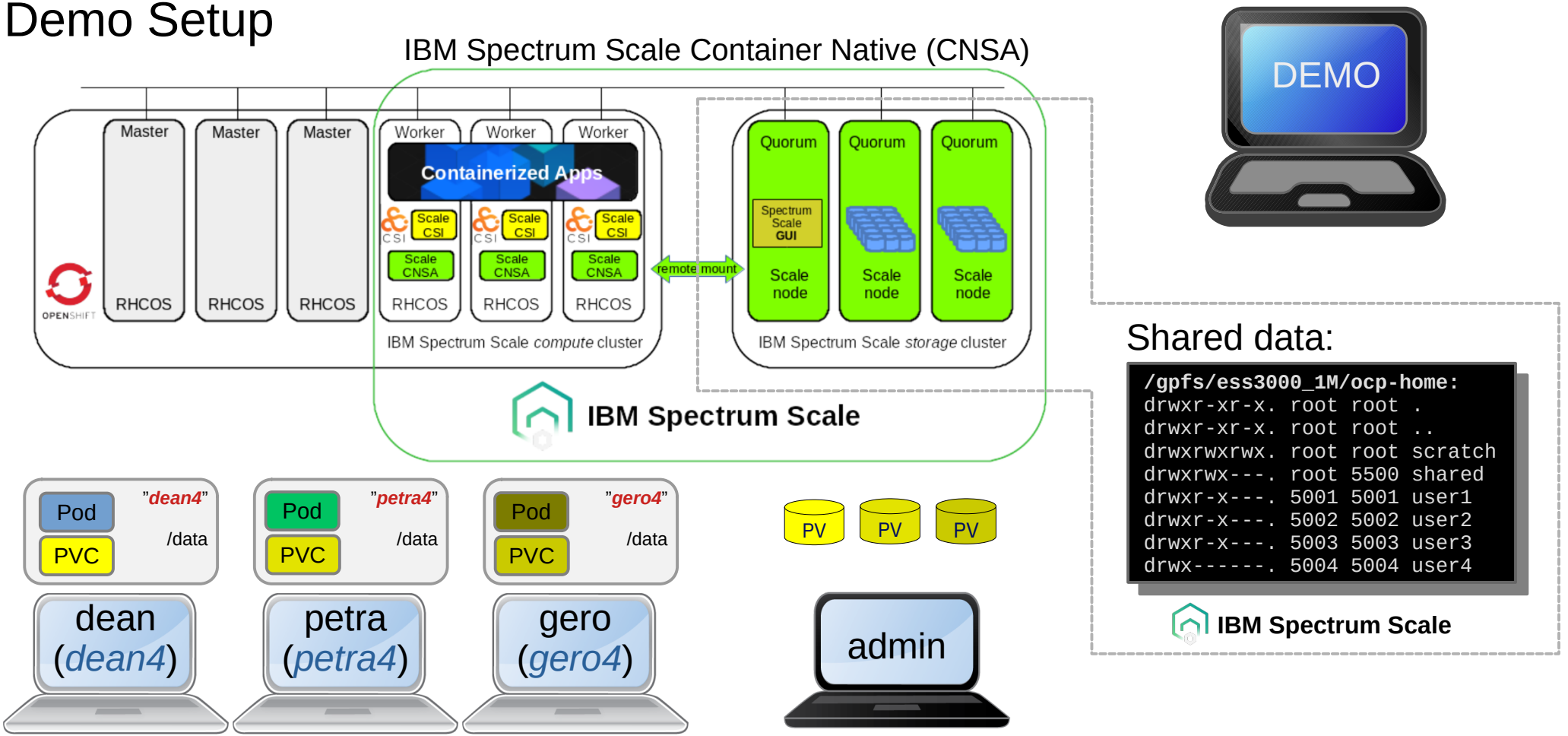
accessing *existing data* in IBM Spectrum Scale

(2) Using a ***custom SCC*** *&* ***securityContext*** *for*

securely providing *shared data access across namespaces*

The demo recording can be found here: *https://youtu.be/IUvn4YpFzRY*

Demo Setup


IBM Spectrum Scale Container Native (CNSA)
Master
Master
Master
Worker
Worker
Worker
Containerized Apps
Scale CSI
Scale CNSA
RHCOS
OPENSHIFT
IBM Spectrum Scale compute cluster
remote mount
Quorum
Spectrum Scale GUI
Scale node
IBM Spectrum Scale storage cluster
IBM Spectrum Scale
DEMO
Shared data:
/gpfs/ess3000_1M/ocp-home:
drwxr-xr-x. root root .
drwxr-xr-x. root root ..
drwxrwxrwx. root root scratch
drwxrwx---. root 5500 shared
drwxr-x---. 5001 5001 user1
drwxr-x---. 5002 5002 user2
drwxr-x---. 5003 5003 user3
drwx------. 5004 5004 user4
IBM Spectrum Scale
Pod
PVC
"dean4"
/data
"petra4"
"gero4"
PV
dean (dean4)
petra (petra4)
gero (gero4)
admin

Thank you for being a valued member of our IBM Spectrum Scale Community!