Lesson 1 : Container Image Identity and Tags

Relate container image tags to their identifier hashes, and identify container images from pods and containers on Kubernetes nodes.

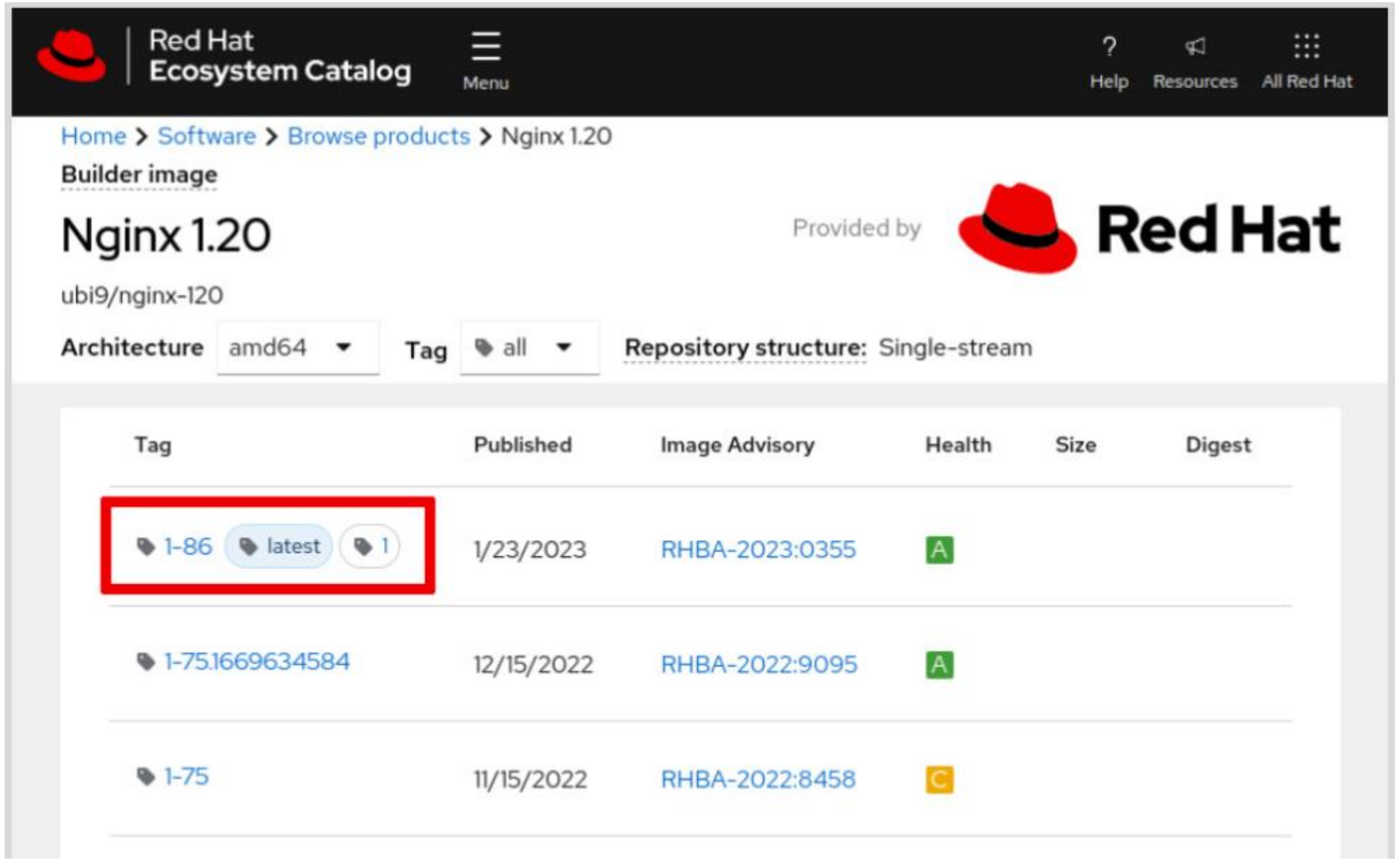
Image : Naming convention

registry_name/user_name/image_name:tag

- registry_name: FQDN of registry server hosting the image.
- user_name: namespace or user or organization the image belongs to
- image_name: unique name in user namespace
- tag: image version. If image name does not provide image tag, latest tag is assumed

Tags

- Multi tags can refer to same image version
- Following ubi9/nginx-120 image has
  - 4 versions: 1, 1-86, 1-75.1669634584 and 1-75
  - 1-86, latest and 1 tags points to same image version
  - Latest and 1 tags are floating tags



Home > Software > Browse products > Nginx 1.20

Builder image

Nginx 1.20

Provided by Red Hat

ubi9/nginx-120

Architecture amd64 Tag all Repository structure: Single-stream

Tag	Published	Image Advisory	Health	Size	Digest
1-86 latest 1	1/23/2023	RHBA-2023:0355	A		
1-75.1669634584	12/15/2022	RHBA-2022:9095	A		
1-75	11/15/2022	RHBA-2022:8458	C		

Floating Tags

- can point to different image versions over time
- Issues faced: Floating tag can be re-assign to new image version without notice
  - Developer stick to latest unaware of the underlying version.
- Example 1:
  - Mon : image:v1-java → latest
  - Tues : image:v2-php → latest → pull by developer1 : App1 deploy image:v2-php to node1
  - Wed : image:v3-php
  - Thu : Image:v4-java → latest
  - Fri : node1 fails
    - ❖ OpenShift relocates App1 to another healthy node - node2
    - ❖ On node2, it pull image with latest tag, thereby use the new version Image:v4-java which different from original version
    - ❖ Developer1 not aware of this

Floating Tags

- can point to different image versions over time
- Issues faced: Floating tag can be re-assign to new image version without notice
  - Developer stick to latest unaware of the underlying version.
- Example 2:
  - Mon : image:v1-java → latest
  - Tues : image:v2-php → latest → pull by developer1 : App1 deploy image:v2-php to node1
  - Wed : image:v3-php
  - Thu : Image:v4-java → latest
  - Fri : AutoScaler add new pods on node3, node4
    - ❖ On node3 and node4, it pull image with latest tag, thereby use the new version Image:v4-java which different from first pod
    - ❖ Conflicting runtime may complicate and crashes data

Floating Tags

- The distinction between a floating and non-floating tag is not a technical one, but a convention.
- Discouraged, however there is no mechanism can be used to prevent developer from pushing different image version to existing tag.
- Prevention: select image that is guaranteed not to change over time.
  1. Use the SHA (Secure Hash Algorithm) image ID instead of tag when referencing image version
  2. Use OpenShift image stream (IS). Better control over image version.
  3. Don't use floating tag such as "latest" tag

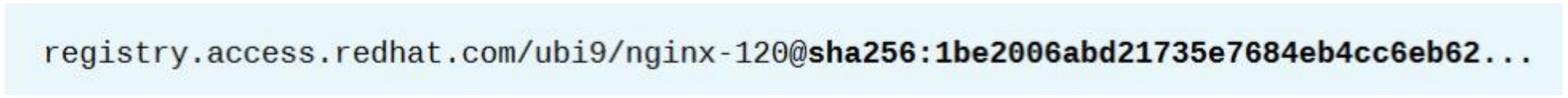
Using SHA Image ID

- is an unique identifier assigned to image in form of computed digest value using SHA algorithm
- is immutable string
- Most secure approach when referring to an image
- To refer; use image-name@SHA-ID instead of image-name:tag

```
$ oc new-app --image
```

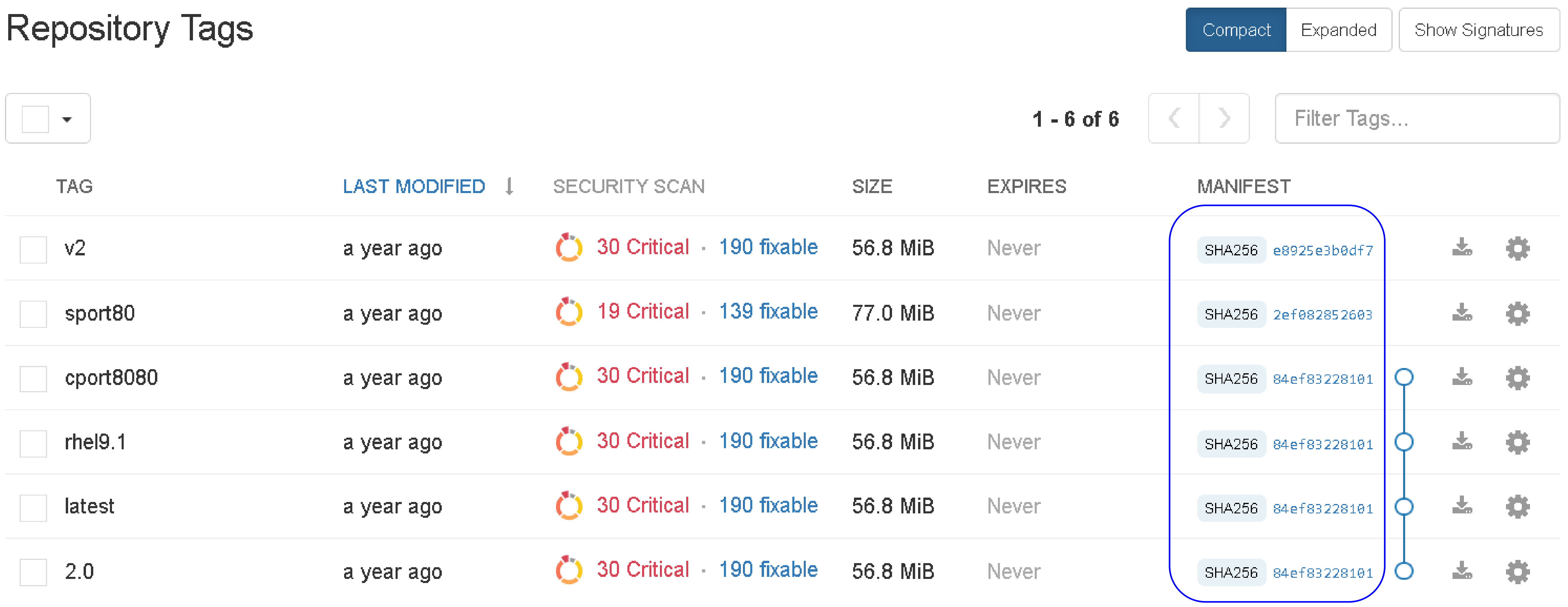
or

```
$ oc run --image
```



```
registry.access.redhat.com/ubi9/nginx-120@sha256:1be2006abd21735e7684eb4cc6eb62...
```

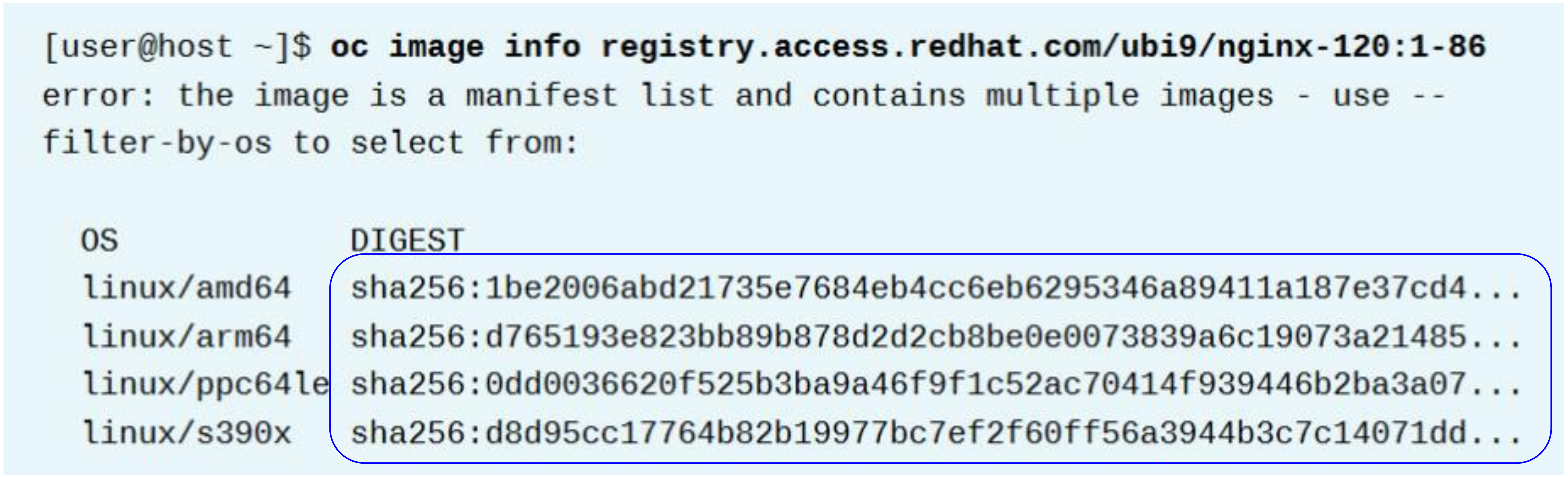
List Tags with the SHA Image ID



Repository Tags

TAG	LAST MODIFIED	SECURITY SCAN	SIZE	EXPIRES	MANIFEST
v2	a year ago	30 Critical · 190 fixable	56.8 MiB	Never	SHA256 e8925e3b0df7
sport80	a year ago	19 Critical · 139 fixable	77.0 MiB	Never	SHA256 2ef082852603
cport8080	a year ago	30 Critical · 190 fixable	56.8 MiB	Never	SHA256 84ef83228101
rhel9.1	a year ago	30 Critical · 190 fixable	56.8 MiB	Never	SHA256 84ef83228101
latest	a year ago	30 Critical · 190 fixable	56.8 MiB	Never	SHA256 84ef83228101
2.0	a year ago	30 Critical · 190 fixable	56.8 MiB	Never	SHA256 84ef83228101

List Tags with the SHA Image ID Tags using **oc image info** command

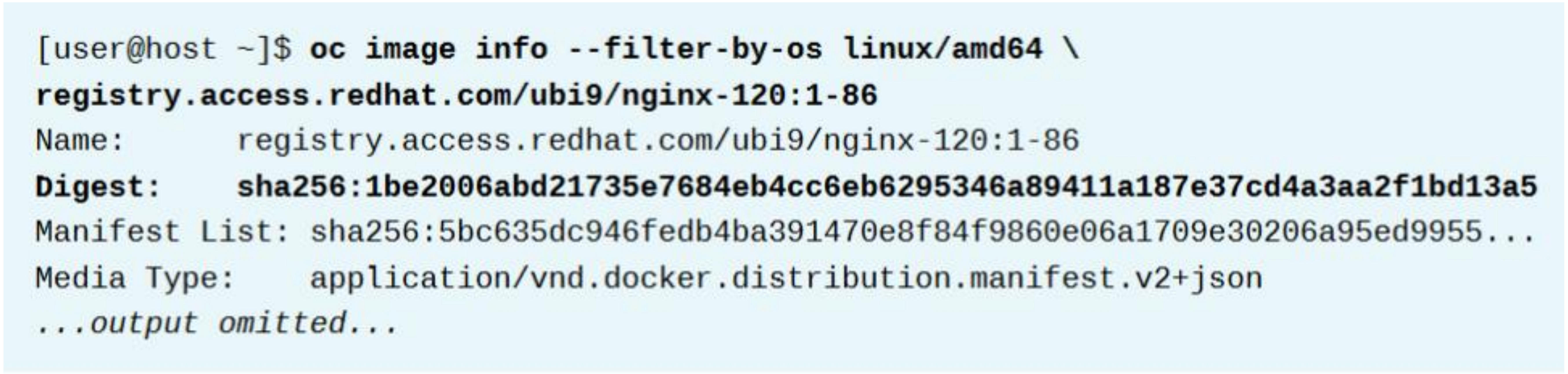


```
[user@host ~]$ oc image info registry.access.redhat.com/ubi9/nginx-120:1-86
error: the image is a manifest list and contains multiple images - use --
filter-by-os to select from:

  OS            DIGEST
  linux/amd64   sha256:1be2006abd21735e7684eb4cc6eb6295346a89411a187e37cd4...
  linux/arm64   sha256:d765193e823bb89b878d2d2cb8be0e0073839a6c19073a21485...
  linux/ppc64le sha256:0dd0036620f525b3ba9a46f9f1c52ac70414f939446b2ba3a07...
  linux/s390x   sha256:d8d95cc17764b82b19977bc7ef2f60ff56a3944b3c7c14071dd...
```

List Tags with the SHA Image ID Tags using oc image info command

- Use **--filter-by-os** option to view details
- Alternatively use **skopeo inspect** command



```
[user@host ~]$ oc image info --filter-by-os linux/amd64 \
registry.access.redhat.com/ubi9/nginx-120:1-86
Name:       registry.access.redhat.com/ubi9/nginx-120:1-86
Digest:     sha256:1be2006abd21735e7684eb4cc6eb6295346a89411a187e37cd4a3aa2f1bd13a5
Manifest List: sha256:5bc635dc946fedb4ba391470e8f84f9860e06a1709e30206a95ed9955...
Media Type:    application/vnd.docker.distribution.manifest.v2+json
...output omitted...
```

Use **crictl images** command

--digests display SHA image IDs

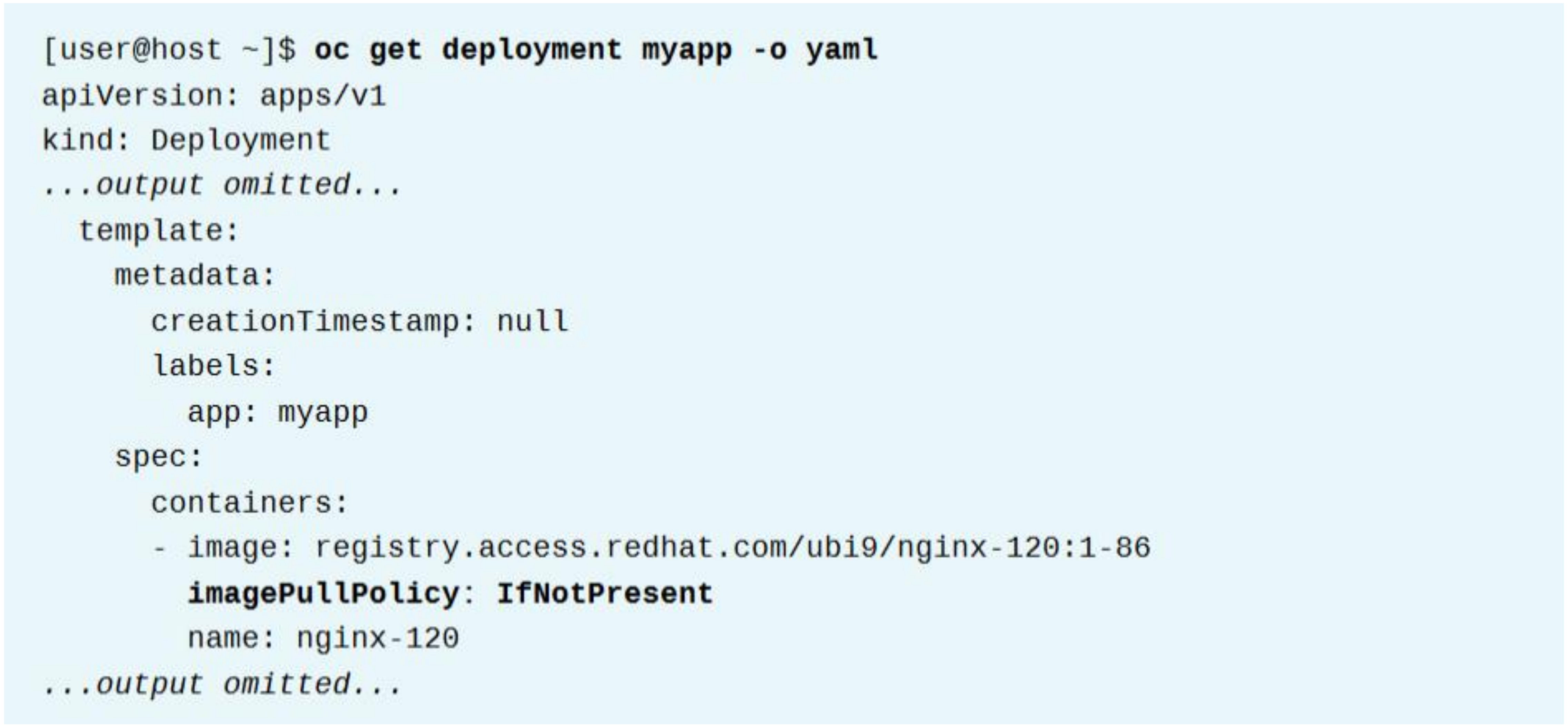
--no-trunc display SHA full strings

```
[user@host ~]$ oc debug node/node-name
Temporary namespace openshift-debug-csn2p is created for debugging node...
Starting pod/node-name-debug ...
To use host binaries, run `chroot /host`
Pod IP: 192.168.50.10
If you don't see a command prompt, try pressing enter.
sh-4.4# chroot /host
sh-4.4# crictl images --digests --no-trunc \
registry.access.redhat.com/ubi9/nginx-120:1-86
IMAGE                                     TAG  DIGEST            IMAGE ID    ...
registry.access.redhat.com/ubi9/nginx-120 1-86 sha256:1be2...13a5 2e68...949e ...
```

Selecting a Pull Policy

- Control when OpenShift pulls/download image to compute node
- **IfNotPresent**
  - If image not on compute node, then OpenShift pull image from registry.
  - If image is already on compute node, because OpenShift pulled the image during a preceding deployment, then OpenShift uses that local image.
- **Always**
  - OpenShift retrieves the SHA ID of the image from the external registry.
  - It compares local image SHA ID to the external image,
    - If SHA ID is same then OpenShift uses local image
    - Else different SHA ID (newly built image), then OpenShift pull the new image.
  - Case use: floating tag
- **Never**
  - OpenShift never pull the image, even if compute node does not have the image locally. Deployment will fails.
  - Case use: To improve speed and avoid relying on external container registry. Developer manually pull image when needed

Example of YAML manifest with the Pull Policy set



```
[user@host ~]$ oc get deployment myapp -o yaml
apiVersion: apps/v1
kind: Deployment
...output omitted...
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: myapp
    spec:
      containers:
      - image: registry.access.redhat.com/ubi9/nginx-120:1-86
        imagePullPolicy: IfNotPresent
        name: nginx-120
...output omitted...
```

Pruning Images from Cluster Nodes

- When deployment/pod are deleted, associated images remain on cluster nodes.
  - Reason: OpenShift can reuse image without having to pull again from external registry. This to conserve network bandwidth.
- Images consume disk space on compute nodes.
- By default: Kubelet agent runs garbage collector to remove older unused images
  - File system storage images is above 85%
  - Stop when file system usage drops below 85%
- Pruning - manual process to quickly removes unused images.

Use **crictl imagefsinfo** command

```
[user@host ~]$ oc debug node/node-name
Temporary namespace openshift-debug-csn2p is created for debugging node...
Starting pod/node-name-debug ...
To use host binaries, run `chroot /host`
Pod IP: 192.168.50.10
If you don't see a command prompt, try pressing enter.
sh-4.4# chroot /host
sh-4.4# crictl imagefsinfo
{
  "status": {
    "timestamp": "1674465624446958511",
    "fsId": {
      "mountpoint": "/var/lib/containers/storage/overlay-images"
    },
    "usedBytes": {
      "value": "1318560"
    },
    "inodesUsed": {
      "value": "446"
    }
  }
}
```

- On compute node
- Retrieve local file system that stores container images

Manually execute the pruning process

```
$ crictl rmi --help
NAME:
   crictl rmi - Remove one or more images

USAGE:
   crictl rmi [command options] IMAGE-ID [IMAGE-ID...]

OPTIONS:
   --all, -a    Remove all images (default: false)
   --prune, -q  Remove all unused images (default: false)
   --help, -h   show help (default: false)
```

```
$ crictl rmi --prune
```

- Recommended to rely on garbage collector to prune unused images, or unused containers

Guided Exercise: Container Image Identity and Tags

You should be able to:

- Inspect container images
- List images of containers that run on compute nodes
- Deploy applications by using image tags or SHA IDs.

Lesson 2 : Update Application Image and Setting

Update applications with minimal downtime by using deployment strategies

Application Code, Configuration, and Data

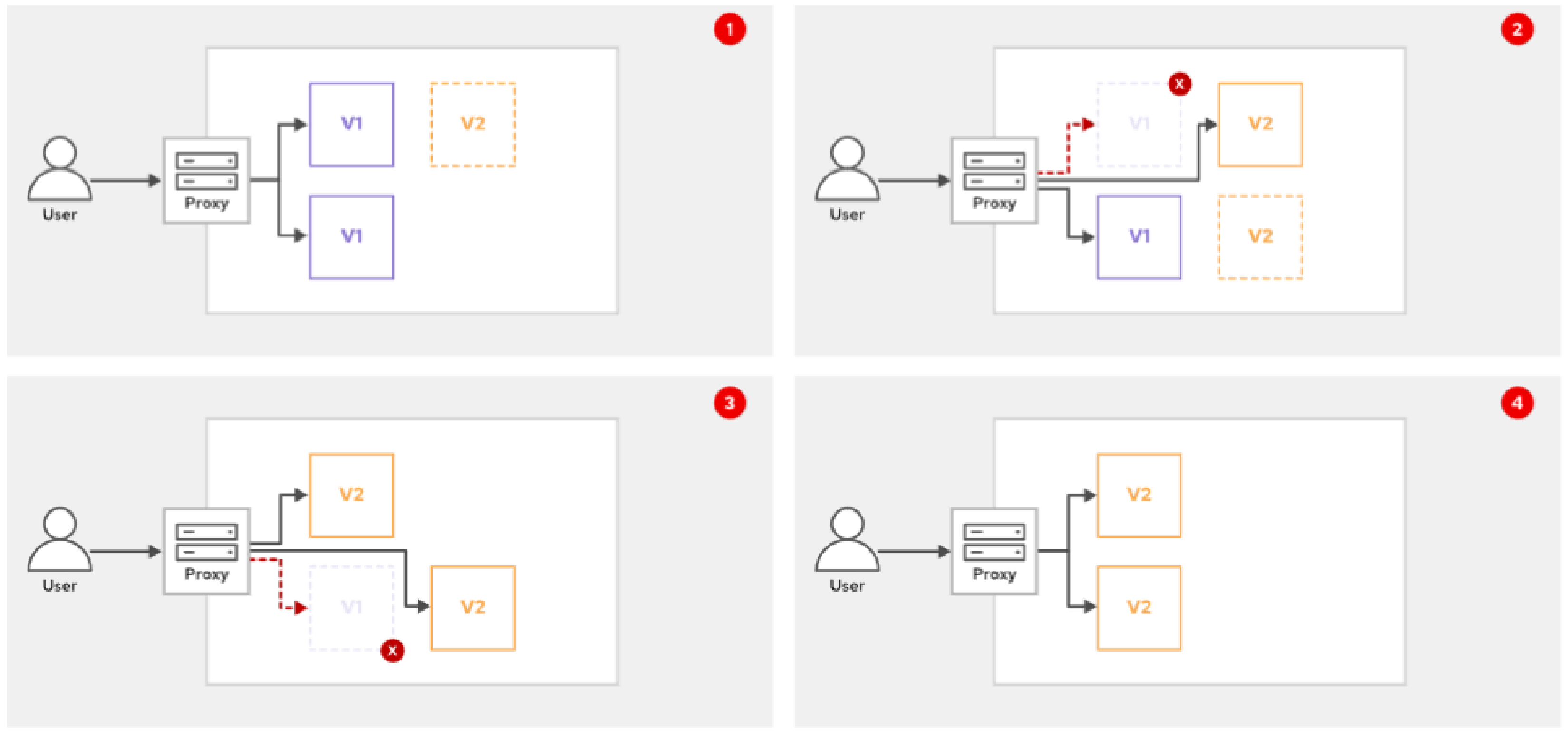
- Common Practices:
  - Applications decouple code, configuration and data from application.
  - Each deployment, applications loads configuration from external source.
  - Advantage: Deploying application to different environment without source code changes.
- OpenShift provides:
  - Configuration map, secret and volume resources to externalized configuration and data.
  - Image Stream to externalized application source codes.
- Use of CI/CD pipeline to automatically build image from change source code, then push images to container registry.
- Use configMaps or Secrets to update configuration of application.

Deployment Strategies

- New versions to users may introduce bugs or reduce application performance
- Above risks application failures which result in:
  - Disruption to services or transactions → Business disrupt and violation of SLA → business lost in revenue
- Testing and Validation stages in CI/CD pipelines to reduce or mitigate risks
- Use of deployment strategy:
  - RollingUpdate
  - Recreate

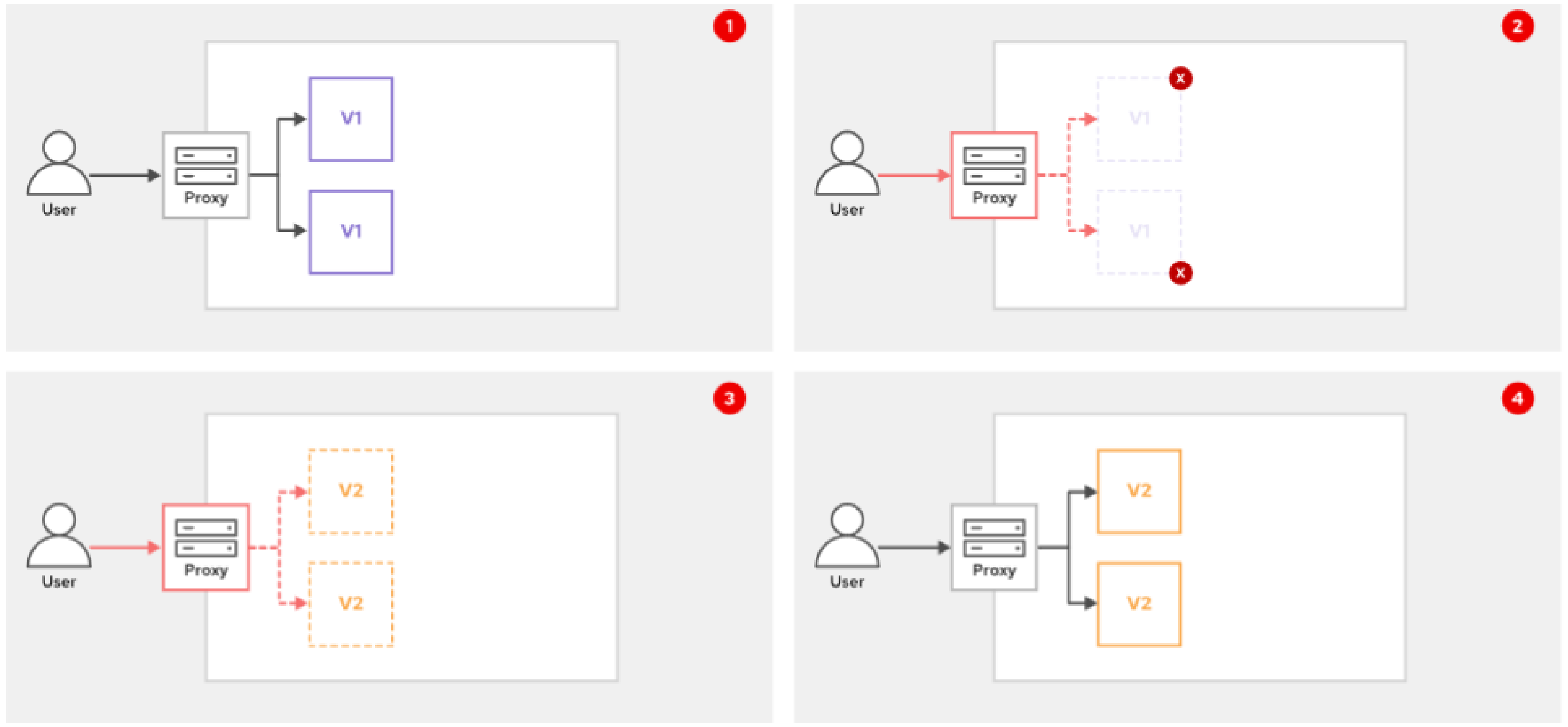
Rolling Update Strategy

- Updating version of application in stages
- Replaces one instance after another until all instances are replaced.
- Drawbacks: requires compatibility between versions in deployment



Recreate Strategy

- All instances are killed first
- Then replaced with new ones.
- Drawbacks: Causes downtime; because no instances are available to fulfil requests



Comparisons

Strategy	Advantage	Disadvantage	Use case
Rolling Update	Slower. Eliminates downtime Uses fewest resources: replaces pods without infrastructure overheads	Two version exists simultaneously at one point. This causes transient incompatibilities issues	When different version of applications can run at same time. Applications that need HA: example medical systems
Recreates	Faster when dealing with large number of pods Wont face two version incompatibilities	Long downtime Sudden surged in node's resources consumption	When application cannot have different simultaneously running code version. Data Migrations or data transformations

Defining the strategy in YAML manifest

```
apiVersion: apps/v1
kind: Deployment
metadata:
...output omitted...
spec:
  progressDeadlineSeconds: 600
  replicas: 10
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: myapp2
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 50%
    type: RollingUpdate
  template:
...output omitted...
```

- maxSurge: indicate how many pods OpenShift can be create above normal number of replicas
- maxUnavailable: indicates how many pods OpenShift can remove below normal number of replicas
- Both parameters can use percentage or number of pods
- Recommended to configure readiness probe

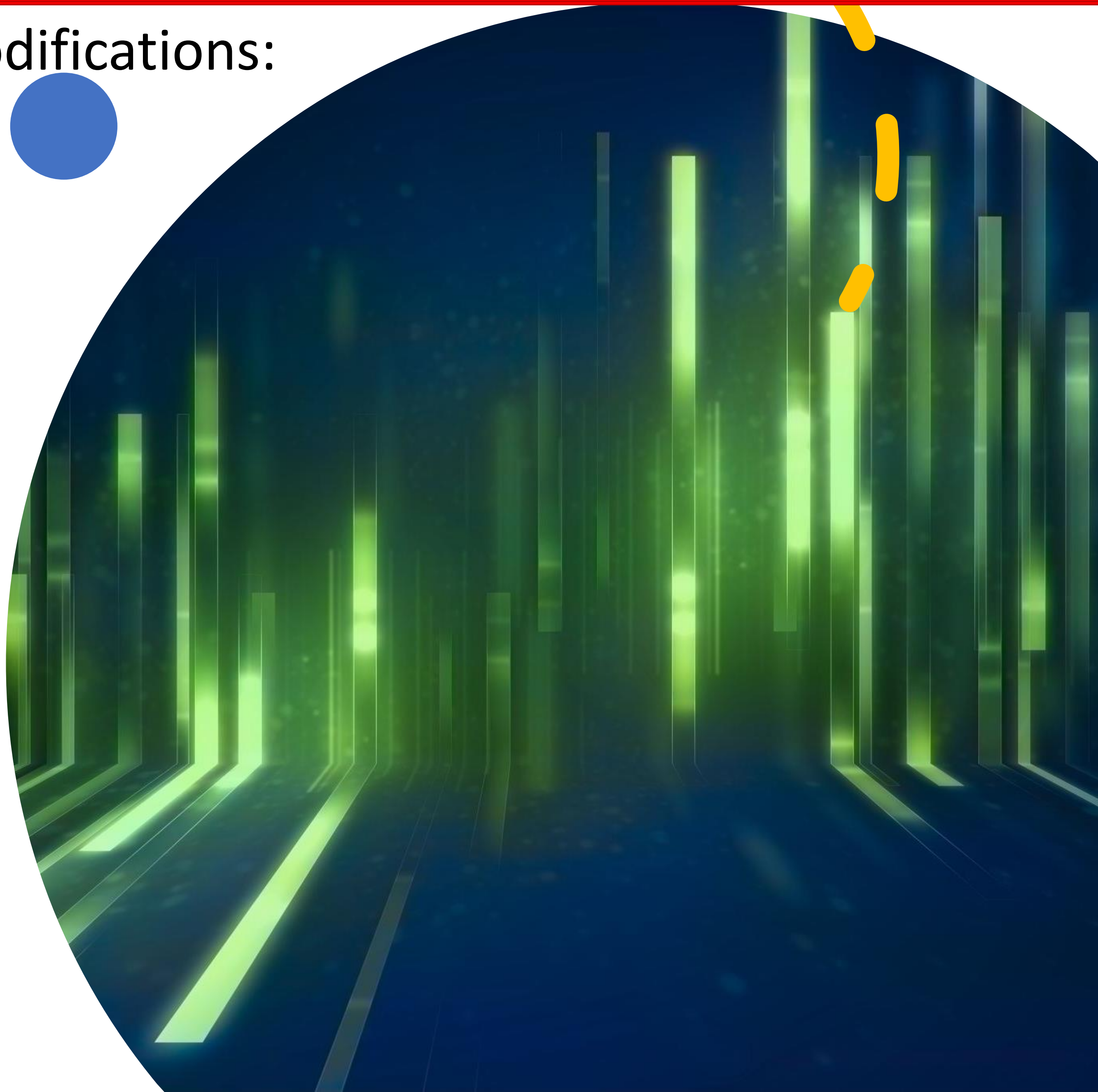
Rolling out Applications

- OpenShift rolls out application for each modifications:
  - Modify image version
  - Injecting environment variables
  - Configure readiness probes and so on
- Prevent multiple deployments

01 Pause rollout

02 Apply all modifications

03 Resume rollout



Rolling out Applications

- Use the `oc rollout pause` command to pause the rollout of the `myapp` deployment:

```
[user@host ~]$ oc rollout pause deployment/myapp
```

- Apply all your modifications to the `Deployment` object. The following example modifies the image, an environment variable, and the readiness probe.

```
[user@host ~]$ oc set image deployment/myapp \
nginx-120=registry.access.redhat.com/ubi9/nginx-120:1-86
[user@host ~]$ oc set env deployment/myapp NGINX_LOG_TO_VOLUME=1
[user@host ~]$ oc set probe deployment/myapp --readiness --get-url http://:8080
```

- Resume the rollout:

```
[user@host ~]$ oc rollout resume deployment/myapp
```

OpenShift rolls out the application to apply all your modifications to the `Deployment` object.

Rolling out Applications

- Similarly; only this time set number of replicas to zero

```
[user@host ~]$ oc create deployment myapp2 \
--image registry.access.redhat.com/ubi9/nginx-120:1-86 --replicas 0
[user@host ~]$ oc get deployment/myapp2
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
myapp2    0/0     0            0           9s
```

- Apply the configuration to the `Deployment` object. The following example adds a readiness probe.

```
[user@host ~]$ oc set probe deployment/myapp2 --readiness --get-url http://:8080
```

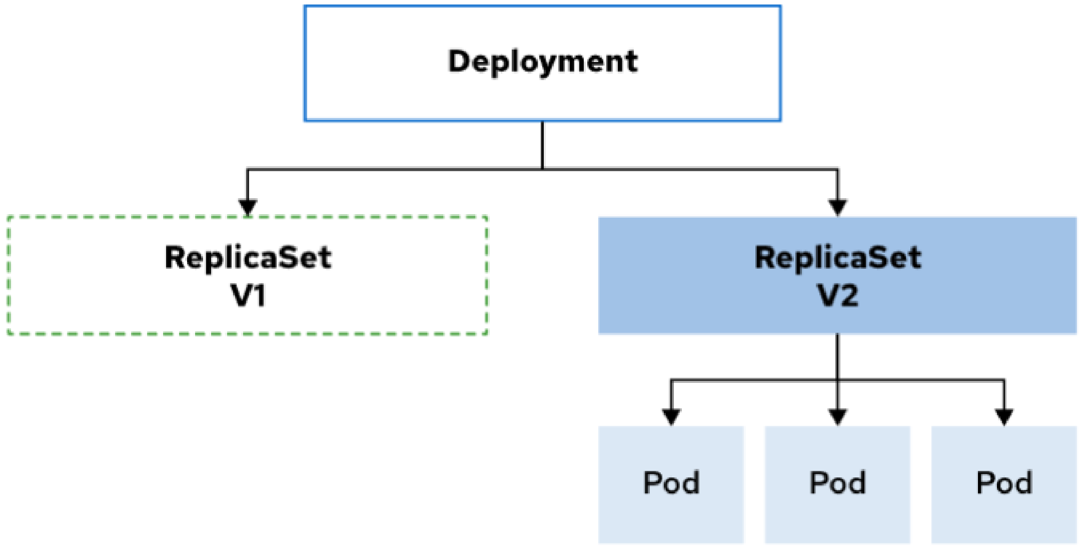
- Scale up the deployment. OpenShift rolls out the application.

```
[user@host ~]$ oc scale deployment/myapp2 --replicas 10
[user@host ~]$ oc get deployment/myapp2
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
myapp2    10/10   10           10          18s
```

ReplicaSets

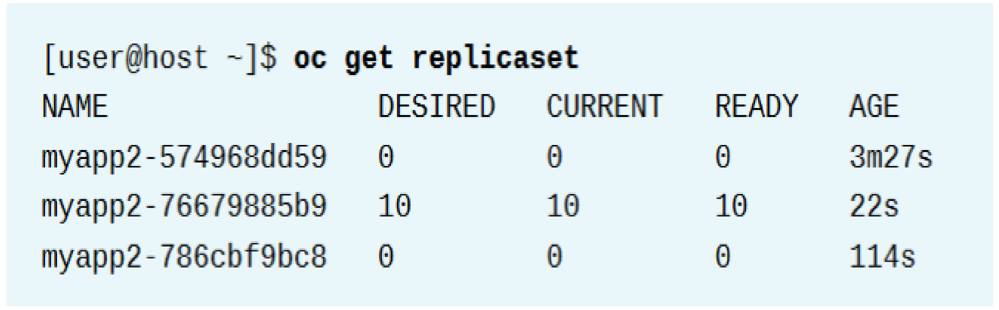
- OpenShift deploy ReplicaSet to create and monitor pods.
- If pod fails, ReplicaSet deploys new one.
- When deploying new pods, ReplicaSet always copy definition from Deployment object.
- When Deployment updated:
  - Create new ReplicaSet object
  - Copy definition from updated Deployment to the new ReplicaSet
  - Old completed ReplicaSet is kept for rollback purposes
- Several ReplicaSets coexists and coordinate rollout of new application versions

Lifecycle of ReplicaSets



- Old ReplicaSet does not run any pods.
- Current ReplicaSet v2 manages 3 replicated pods.
- Do not directly change or delete ReplicaSets.
- Deployment
  - Specifies ReplicaSet copy to keep
  - Automatically Delete extra ReplicaSet
- If deployment is deleted, all associated ReplicaSets will be deleted

Monitoring ReplicaSets during rolling update

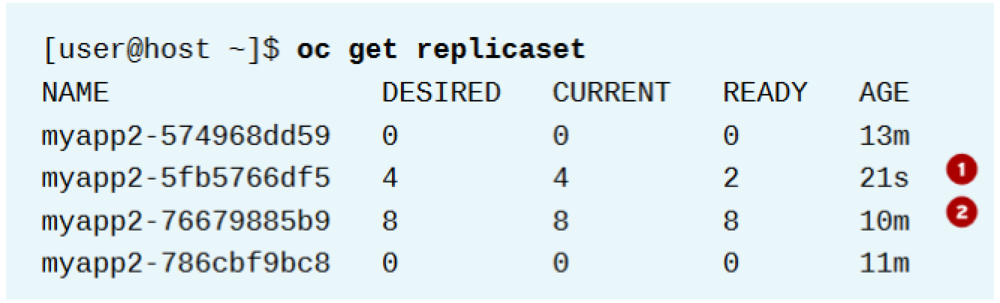


```
[user@host ~]$ oc get replicaset
NAME                DESIRED    CURRENT    READY    AGE
myapp2-574968dd59   0          0          0        3m27s
myapp2-76679885b9   10         10         10       22s
myapp2-786cbf9bc8   0          0          0        114s
```

Before rolling update:

Only second ReplicaSets is active and monitoring 10 pods

The inactive ReplicaSets represent previous version of the Deployment object



```
[user@host ~]$ oc get replicaset
NAME                DESIRED    CURRENT    READY    AGE
myapp2-574968dd59   0          0          0        13m
myapp2-5fb5766df5   4          4          2        21s   1
myapp2-76679885b9   8          8          8        10m   2
myapp2-786cbf9bc8   0          0          0        11m
```

After rolling update:

During rolling update, two ReplicaSets are active

1. new ReplicaSet is updating 4 new pods; however only 2 is ready to accept client requests

2. old ReplicaSet is scaling down from 10 to 8 pods

Managing Rollout

- If new version of application does not work
  - then rollback using old preserved ReplicaSet

```
[user@host ~]$ oc rollout undo deployment/myapp2
```

  - This is similar to **oc rollback** command but only works with DeploymentConfig
- Check the rollback status

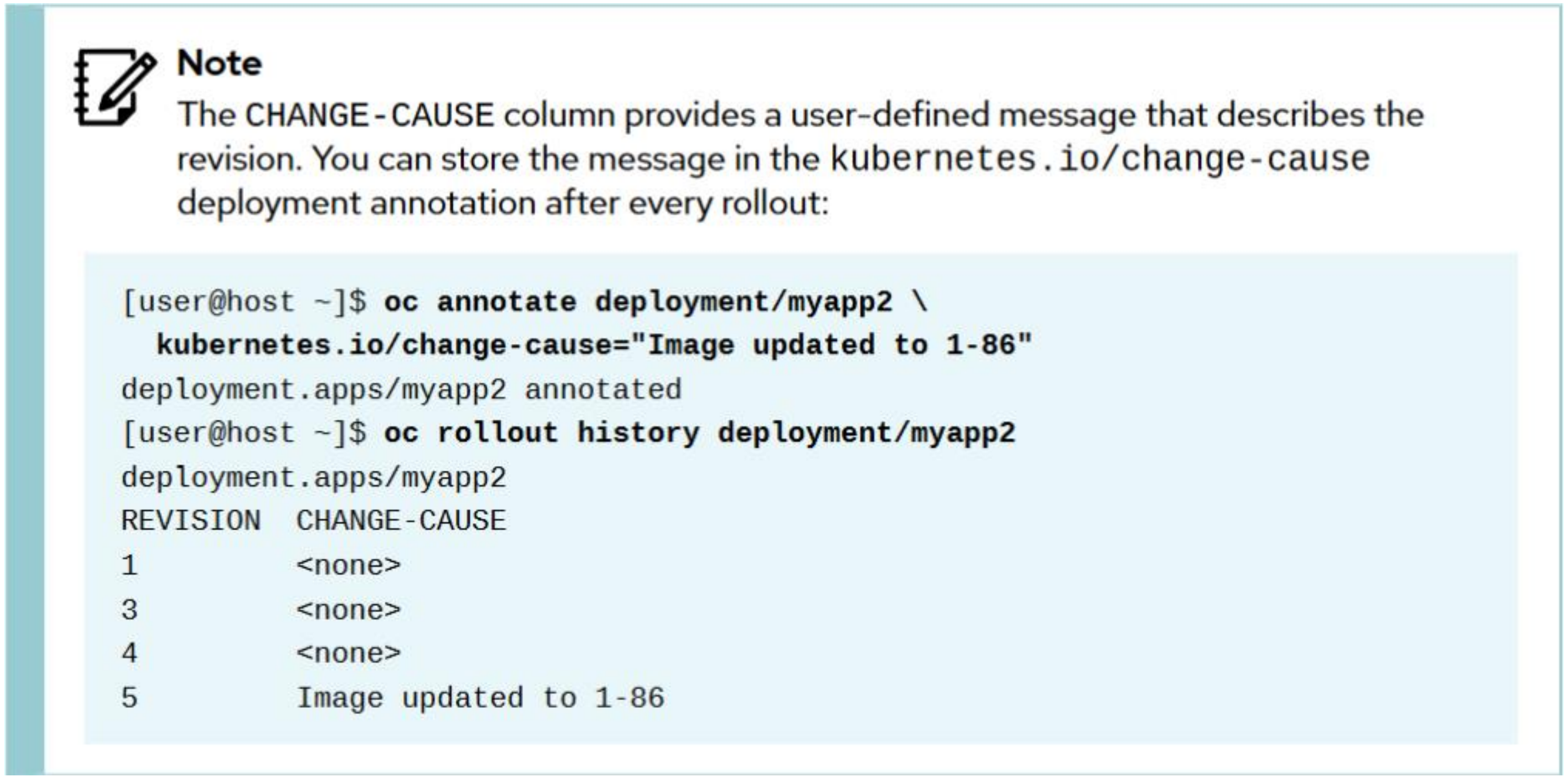
```
[user@host ~]$ oc rollout status deployment/myapp2
deployment "myapp2" successfully rolled out
```

- Default oc rollout undo rolls back to preceding deployment version
- If need to rollback to earlier revision, list available version

```
[user@host ~]$ oc rollout history deployment/myapp2
deployment.apps/myapp2
REVISION  CHANGE-CAUSE
1         <none>
3         <none>
4         <none>
5         <none>
```

```
[user@host ~]$ oc rollout undo deployment/myapp2 --to-revision 1
```

The CHANGE-CAUSE column



Note

The CHANGE-CAUSE column provides a user-defined message that describes the revision. You can store the message in the `kubernetes.io/change-cause` deployment annotation after every rollout:

```
[user@host ~]$ oc annotate deployment/myapp2 \
  kubernetes.io/change-cause="Image updated to 1-86"
deployment.apps/myapp2 annotated
[user@host ~]$ oc rollout history deployment/myapp2
deployment.apps/myapp2
REVISION  CHANGE-CAUSE
1         <none>
3         <none>
4         <none>
5         Image updated to 1-86
```

Lab: Update Application Image and Settings

You should be able to:

- Pause, update, and resume a deployment.
- Roll back a failing application.