- Openshift
  Orchestrierungsservice zur Bereitstellung, Verwaltung und Skalierung von Container-Anwendungen

- Deklaratives System
  Status wird in Resourcen (YAML/JSON) definiert und durch Controller hergestellt
  IaC – Infrastructure as Code (https://blog.nelhage.com/post/declarative-configuration-management)

```
$ oc api-resources -o name --sort-by=name
alertmanagers.monitoring.coreos.com
apiservers.config.openshift.io
apiservices.apiregistration.k8s.io
appliedclusterresourcequotas.quota.openshift.io
authentications.config.openshift.io
authentications.operator.openshift.io
baremetalhosts.metal3.io
bindings
brokertemplateinstances.template.openshift.io
buildconfigs.build.openshift.io
builds.build.openshift.io
builds.config.openshift.io
catalogsources.operators.coreos.com
certificatesigningrequests.certificates.k8s.io
cloudcredentials.operator.openshift.io
clusterautoscalers.autoscaling.openshift.io
clusternetworks.network.openshift.io
clusteroperators.config.openshift.io
...
```

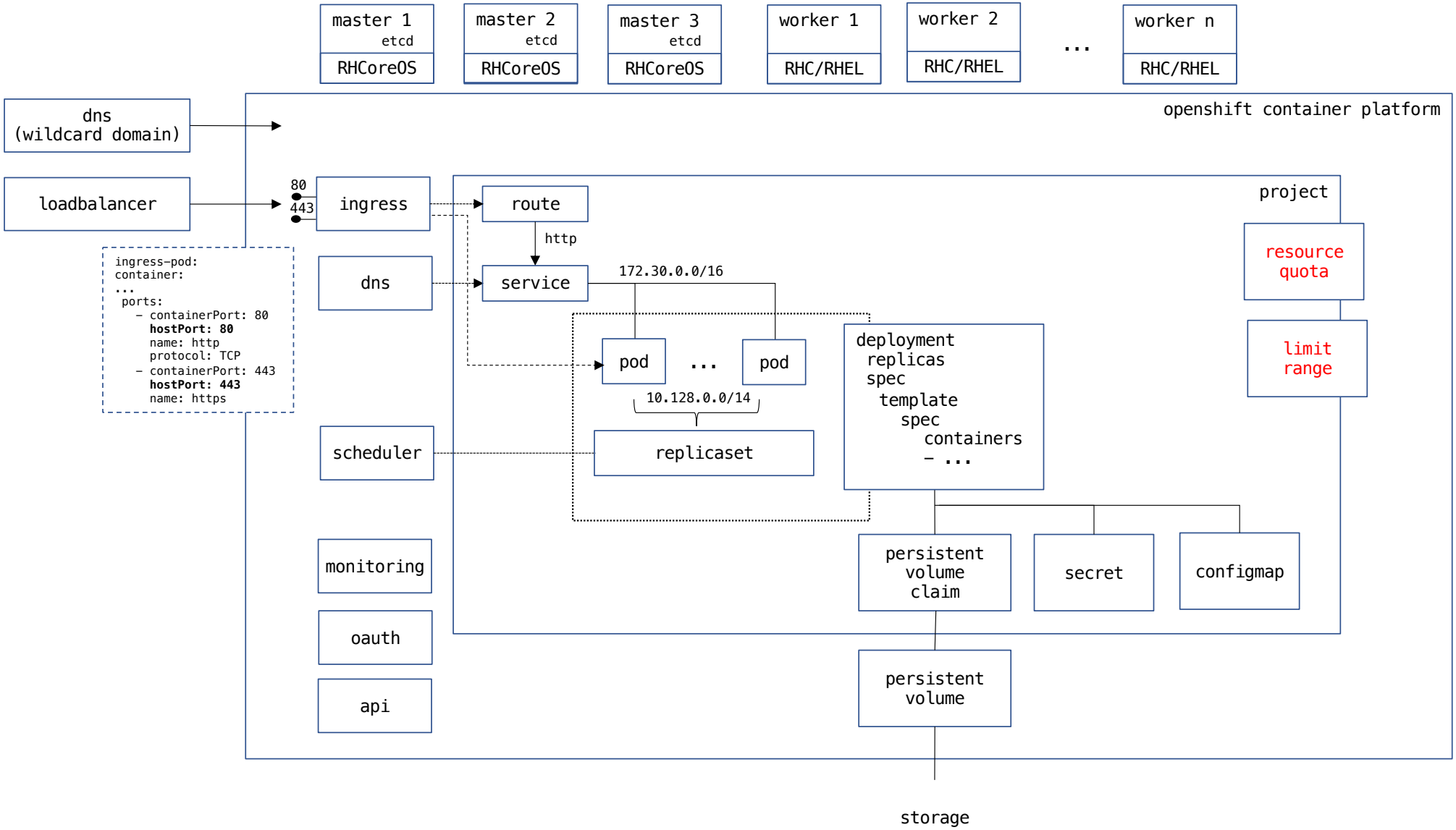
Pod
Replicaset
Deployment
Service (svc)
Route
PersistentVolumeClaim
Secrets
Configmaps

Imagestream
BuildConfig

Node
PersistentVolume

Operator
CustomResourceDefinition

openshift resources

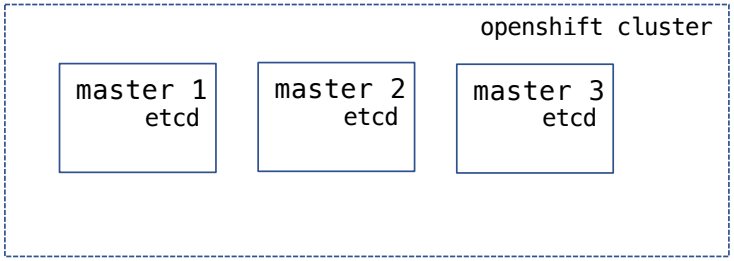

master 1
etcd
RHCoreOS
master 2
etcd
RHCoreOS
master 3
etcd
RHCoreOS
worker 1
RHC/RHEL
worker 2
RHC/RHEL
...
worker n
RHC/RHEL
dns
(wildcard domain)
loadbalancer
openshift container platform
80
443
ingress
route
http
project
resource
quota
ingress-pod:
container:
...
ports:
- containerPort: 80
hostPort: 80
name: http
protocol: TCP
- containerPort: 443
hostPort: 443
name: https
dns
service
172.30.0.0/16
pod
...
pod
10.128.0.0/14
deployment
replicas
spec
template
spec
containers
- ...
limit
range
scheduler
replicaset
monitoring
persistent
volume
claim
secret
configmap
oauth
persistent
volume
api
storage

openshift

Openshift Resources (Manifest)

```
apiVersion: v1
kind: < Resource Type >
metadata:
  name: <name>
  namespace: <namespace>
  annotations:
    ...
  labels:
    app: <application-name>
    ...
spec:
  ...
  selector:
    <key>: <value>
  ...
status:
  ...
```

oc apply

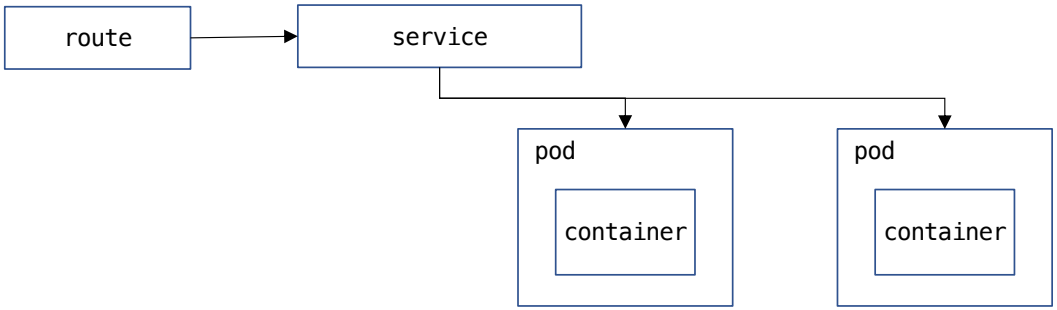



```
apiVersion: v1
kind: Pod
metadata:
  name: webserver
  namespace: do180
  labels:
    app: webserver
spec:
  containers:
  - image: quay.io/danielstraub/webserver:do180
    imagePullPolicy: Always
    ports:
    - containerPort: 8080
      protocol: TCP
  ...
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wildfly
  labels:
    app: wildfly
spec:
  replicas: 2
  selector:
    matchLabels:
      app: wildfly
  template:
    metadata:
      labels:
        app: wildfly
    spec:
      containers:
      - name: wildfly
        image: quay.io/do288/wildfly:latest
        ports:
        - containerPort: 8080
          protocol: TCP
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: wildfly
  labels:
    app: wildfly
spec:
  type: ClusterIP
  selector:
    app: wildfly
  ports:
  - name: http
    protocol: TCP
    port: 8080
    targetPort: 8080
```

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: wildfly
  labels:
    app: wildfly
spec:
  host: sample.apps.eu46.prod.nextcle.com
  to:
    kind: Service
    name: wildfly
  tls:
    termination: edge
```



application

Declarative :

```
$ ls
deployment.yml route.yml service.yml

$ oc apply -f .
deployment.apps/wildfly created
route.route.openshift.io/wildfly created
service/wildfly created
```

Imperative :

```
$ oc new-app <container-image | git-repository>
--> Found container image 9a9e908 (9 days old) from quay.io for "quay.io/do288/wildfly"

    * An image stream tag will be created as "wildfly:latest" that will track this image

--> Creating resources ...
    imagestream.image.openshift.io "wildfly" created
    deployment.apps "wildfly" created
    service "wildfly" created
--> Success
```

- oc login –u <user> -p <password> <api-server-url>

- oc new-project <name>

- oc create –f <resource-yml>
  oc apply –f <resource-yml>

- oc status

- oc get <resource-type> [<resource-name>]

  - oc get pods
  - oc get deployment
  - oc get svc <service>
  - oc get events

- oc describe <resource-type> <resource-name>

- oc expose svc <service-name>

- oc logs <podname>

- oc exec –it <podname> -- <program>
- oc rsh <podname>
- oc cp <pod>:<locatio> <location>

- oc port-forward <podname> <local-port>:<remote-port>

- oc new-app <☺anything☺>

- oc delete <resource-type> <resource-name>

- oc rollout latest deployment <deployment-name>

https://docs.openshift.com/container-platform/4.15/cli_reference/openshift_cli/developer-cli-commands.html

```
$ oc create deployment --image=quay.io/danielstraub/toolbox -o yaml toolbox -- bash -c 'sleep infitity'
apiVersion: apps/v1
kind: Deployment
metadata:
  name: toolbox
  labels:
    app: toolbox
spec:
  replicas: 1
  selector:
    matchLabels:
      app: toolbox
  template:
    metadata:
    labels:
        app: toolbox
    spec:
      containers:
      - command:
        - bash
        - -c
        - sleep infitity
        image: quay.io/danielstraub/toolbox
        name: toolbox
```

```
$ oc create service clusterip webserver --tcp=80:8080 -o yaml
apiVersion: v1
kind: Service
metadata:
  name: webserver
  labels:
    app: webserver
spec:
  ports:
  - name: 80-8080
    port: 80
    protocol: TCP
    targetPort: 8080
  selector:
    app: webserver
  type: ClusterIP

$ oc create route edge --hostname do180.<wildcard-doamin> --service webserver --insecure-policy=Redirect webserver -o yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: webserver
  labels:
    app: webserver
spec:
  host: do180.apps.eu410.prod.nextcle.com
  port:
    targetPort: http
  tls:
    insecureEdgeTerminationPolicy: Redirect
    termination: edge
  to:
    name: webserver
```

```
oc create route -help

Available Commands:
  edge        Create a route that uses edge TLS termination
  passthrough Create a route that uses passthrough TLS termination
  reencrypt   Create a route that uses reencrypt TLS termination
```

```
/
├── bin
├── etc
├── lib
├── opt
├── proc
├── usr
└── var
    └── lib
        └── container
            └── storage

/
systemd─┬─agetty
        ├─auditd───{auditd}
        ├─chronyd
        ├─crond
        ├─dbus-daemon
        ├─irqbalance
        ├─ksmtuned───sleep
        ├─polkitd───6*[{polkitd}]
        ├─rpc.idmapd
        ├─rpc.mountd
        ├─rpc.statd
        ├─rpcbind
        ├─rsyslogd───2*[{rsyslogd}]
        ├─sshd─── sshd───bash
        ├─systemd-journal
        ├─systemd-logind
        ├─systemd-machine
        ├─systemd-udevd
        └─ < app >
```

```
Container: namespace mnt, pid, net, user, uts, ipc

  /
  ├── bin
  ├── etc
  ├── lib
  ├── opt
  ├── proc
  ├── usr
  └── var

  /
  └─ < app >
```

ip: 10.88.0.x

cgroups : mem, cpu

selinux : context security

secomp : syscalls

eth0

cni0: 10.88.0.1

KERNEL

rootfull container

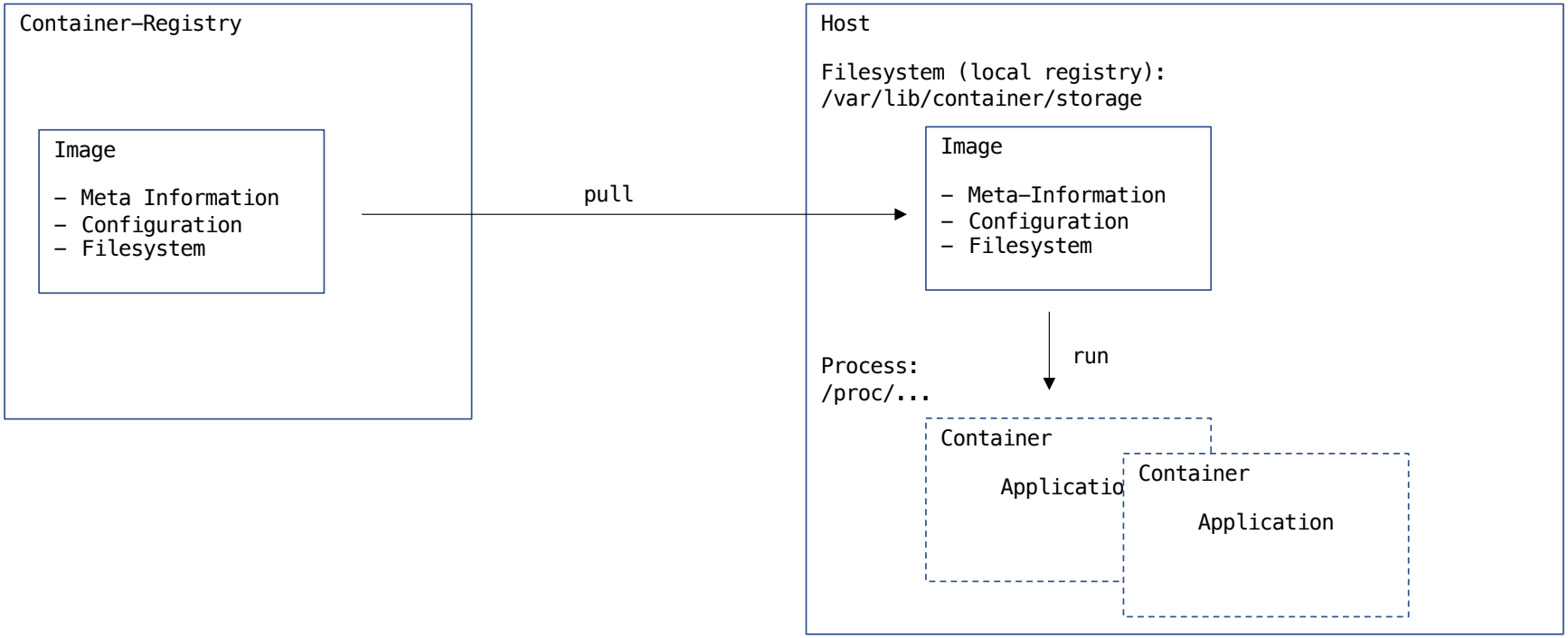
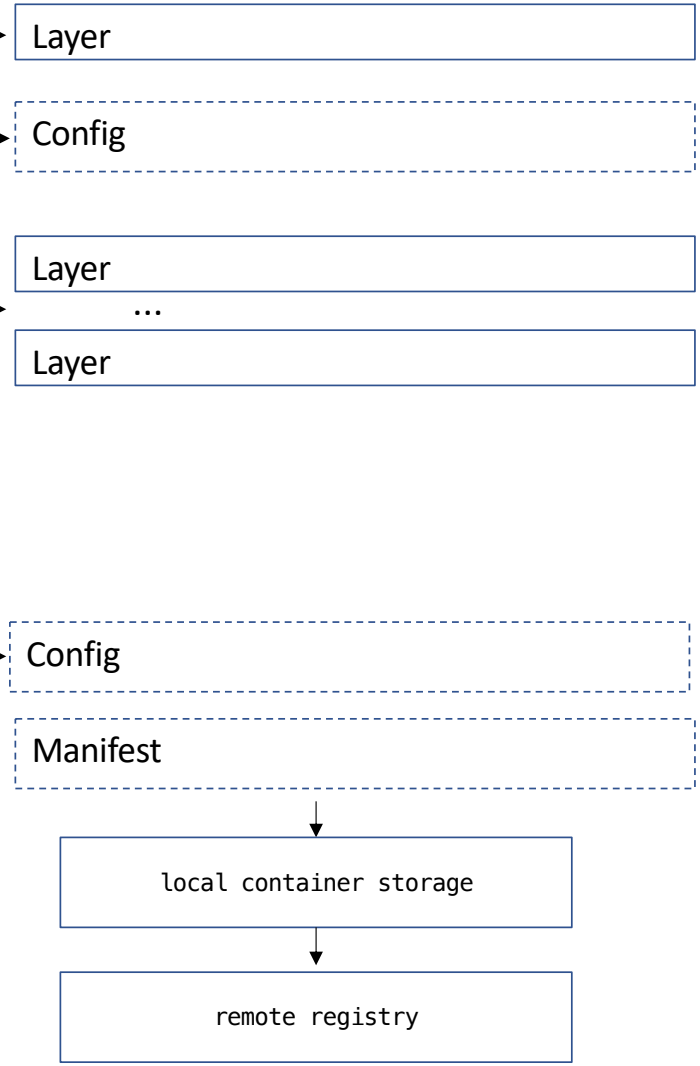


image and container

podman build - Containerfile

```
FROM ubi9/ubi
LABEL version=.... maintainer=
MAINTAINER daniel
ENV key=value
ARG version

ADD  http://repos/app-$version.tar /opt/app/
COPY webapp.war /opt/tomcat/webapps
RUN yum install -y tomcat  && \
         useradd tomcat && \
         chown -R 1000:1000 /opt/tomcat && \
         yum cleanup && rm /tmp/*


USER 1000
WORKDIR /opt/tomcat
VOLUMES /opt/tomcat/logs
EXPOSE [ 8080, 8001 ]
ENTRYPOINT [ "bin/sh -c" ]
CMD [ "bin/catalina.sh", "start" ]
```




```
podman build -t my-tomcat .  <- Build-Dir


podman push <registry>/<name>:<tag>
```

build image

Container in Openshift:

- beliebige User-Id RUN chmod - R 0770 ....
- Group-Id 0 (root) RUN chgrp –R 0
- Ports > 1024

```
apiVersion: project.openshift.io/v1
kind: Project
metadata:
  annotations:
    openshift.io/sa.scc.mcs: s0:c26,c15
    openshift.io/sa.scc.supplemental-groups: 1000680000/10000
    openshift.io/sa.scc.uid-range: 1000680000/10000
```

```
# oc exec pgadmin-778c479f79-tfbqn -- id
uid=1000680000(1000680000) gid=0(root) groups=0(root),1000680000

# ls -al /mnt/nfs/apps/pgadmin
-rw-r--r-- 1 1000680000 root 124K Nov 27 01:03 access_log
-rw-r--r-- 1 1000680000 root  853 Nov 27 00:44 config_local.py
-rw-r--r-- 1 1000680000 root 1.2K Nov 27 00:46 error_log
```

NFS-Mount →

https://cloud.redhat.com/blog/a-guide-to-openshift-and-uids

Verwenden einer externen Container Registry - Authentifizierung

```
$ oc get serviceaccounts
NAME       SECRETS
default    2
...
```

```
$ oc create secret docker-registry <secret-name> --docker-server <registry> --docker-username <user> --docker-password <password>

$ oc secrets link <serviceaccount-name> <secret-name> --for pull
```

Verwenden einer externen Container Registry - Secret von auth.json

```
$ oc create secret generic quayio --from-file .dockerconfigjson=/run/user/1000/containers/auth.json --type kubernetes.io/dockerconfigjson
```

```
apiVersion: v1
kind: Secret
metadata:
  name: quayio
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: ewogICJhdXRocyI6IHsKICAgICJyZWdpc3 ...
```

Serviceaccount 'imagePullSecrets' :

```
$ oc secrets link <serviceaccount-name> <secret-name> --for pull
```

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
imagePullSecrets:
- name: default-dockercfg-4sdrk
- name: quayio
...
```

oder im Deployment verwenden:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pgadmin
spec:
  replicas: 1
  template:
    spec:
      imagePullSecrets:
      - name: quayio
      containers:
      - name: pgadmin
        image: registry.connect.redhat.com/crunchydata/crunchy-pgadmin4
```

Deployment-Strategien

- Rolling Updates : Pods werden der Reihe nach aktualisiert
- Recreate: existierende Pods werden beendet und neue gestartet

DeploymentConfig: DEPRECATED ab 4.15 !

- Pre/Mid/Post – Lifecycle Hooks

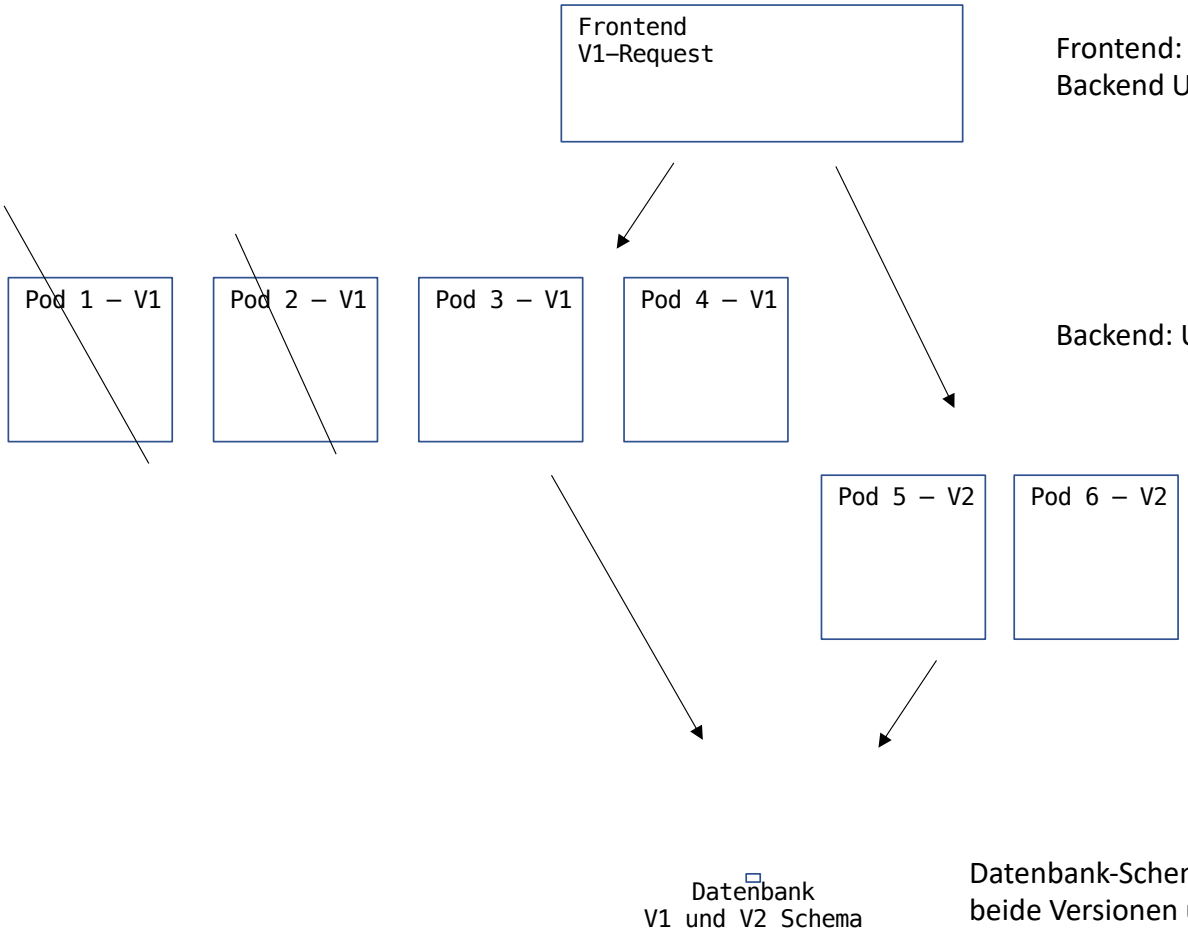
Beenden eines Pods:

- SIGTERM: Pod soll keine neuen Verbindungen annehmen und bestehenden Aktionen beenden
- SIGKILL: nach `terminationGracePeriodSeconds` (30s) wird der Pod beendet

```
kind: Deployment
metadata:
  name: ...
spec:
  revisionHistoryLimit: 3   (default: 10)
  replicas: 4
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1             ← max. 5 Pods aktiv
      maxUnavailable: 1
  ...
  template:
    spec:
      containers:
      - ...
      terminationGracePeriodSeconds: 30
```

```
oc rollout SUBCOMMAND deployment <name>

  cancel      Cancel the in-progress deployment
  history     View rollout history
  latest      Start a new rollout for deployment config with latest state
  pause       Mark the provided resource as paused
  restart     Restart a resource
  resume      Resume a paused resource
  retry       Retry the latest failed rollout
  status      Show the status of the rollout
  undo        Undo a previous rollout
```

N-1 Abwärtskompatibilität bei Rolling-Update:



Frontend: behält V1 bis Backend Update fertig ist

Backend: Update von V1 auf V2

Datenbank-Schema muss auch beide Versionen unterstützen

A/B Deployment Strategy:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: service-a
spec:
ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: http
  selector:
    app.kubernetes.io/instance: deployment-a
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: service-b
spec:
ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: http
  selector:
    app.kubernetes.io/instance: deployment-b
```

```yaml
kind: Route
metadata:
  name: <name>
spec:
  host: <host>
  to:
    kind: Service
    name: service-a
    weight: 80
  alternateBackends:
  - kind: Service
    name: service-b
    weight: 20
```

Secrets:

- Passwörter, Token, Zertifikate ...
- typisiert: basic-auth, dockerfg, tls, opaque
- Inhalte sind base64-decodiert, nicht verschlüsselt

ConfigMap:

- generische Key-Value Daten

→ max. Größe 1 MB
→ nur innerhalb eines Project (NS) sichtbar

```
apiVersion: v1
kind: Secret
metadata:
  name: ...
  namespace: ...
data:
  password: MTIzNDU2
type: Opaque
```

```
# echo MTIzNDU2 | base64 -d
123456
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: ...
  namespace: ...
binaryData:
  keystore: |
    7oAMCAQICCF7Dt6ZDf6TgMA0GCSqGSIb3DQEBBQUAMEI1ZSQUla
    MTEQMA4GA1UECwwHU ...
data:
  HOME: /usr/share/nginx
  default.conf: |
    server {
      listen 8181 default_server;
      server_name _;
      location / {
         root   /usr/share/nginx/html;
         index  index.html index.htm;
      }
    }
```

```
$ oc create configmap <cm-name> --from-literal FOO=BAR

$ oc create configmap <cm-name> --from-file <path>

$ oc create secret docker-registry quayio --docker-server quay.io --docker-username <user> --docker-password <password>
```

Secrets: Verwendung als Umgebungs-Variable

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-env-pod
spec:
  containers:
  - name: mycontainer
    image: redis
    env:
    - name: SECRET_USERNAME
      valueFrom:
        secretKeyRef:
           name: mysecret
           key: username
    - name: SECRET_PASSWORD
      valueFrom:
        secretKeyRef:
           name: mysecret
           key: password
```

ConfigMap: Verwendung als Konfigurations-Dateien

```
apiVersion: apps/v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    container: nginx
    volumeMounts:
    - mountPath: /etc/nginx/conf.d
      name: config
  volumes:
   - name: config
     configMap:
       name: nginx-config
```

```
apiVersion: apps/v1
kind: Pod
metadata:
  name: wildfly-standalone-xml
spec:
  containers:
  - name: wildfly
    container: nginx
    volumeMounts:
    - mountPath: /opt/wildfly/standalone/configuation/standalone.xml
      name: standalone-xml
      subPath: standalone.xml
  volumes:
   - name: standalone-xml
     configMap:
       name: standalone-xml
```

```
$ oc set env deployment/<deployment-name> --from cm/<cm-name>

$ oc set volume deployment/<deployment-name> -add -t configmap -m /etc/nginx/conf.d --name config --configmap-name <cm-name>
```

Volumes

```
node

/
├── bin
├── etc
├── lib        mount
├── mnt      ──────────►  container
├── opt                   /
├── proc                  ├── bin
├── usr                   ├── etc
└── var                   ├── lib
                          ├── mnt
                          ├── opt
                          ├── proc
                          ├── usr
                          └── var

podman –v <local>:<container>
```

```
kind: Pod
...
spec:
  containers:
     ...
     volumeMounts:
       – mountPath: <path_container_fs>
         name: <name>
      ...
  volumes:
     – name: <volume>
       <volume–type>:
          <volume–attributes>
```

→ https://kubernetes.io/docs/concepts/storage/volumes/

Volume=Types
- emptyDir
- hostPath (system:openshift:scc:hostmount–anyuid !)
- configMap
- secret
- persistentVolumeClaim

...

Persistence

Administrator erzeugt PersistentVolume

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-data
spec:
  accessModes:
    - ReadWriteMany
  capacity:
    storage: 10Gi
  nfs:
    path: /mnt/nfs/data
    server: 10.0.0.1
  persistentVolumeReclaimPolicy: Retain
```

- automatisiertes PV-Management mit storageClass/Provisioner

Anwendung erstellt Anforderung

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: html-data
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
```

und verwendet dieses im Deployment / Pod

```
kind: Deployment
...
    containers:
      - name: webserver
        ...
        volumeMounts:
         - mountPath: /usr/share/ngnix/html
           name: html
    volumes:
      - name: html
        persistentVolumeClaim:
          claimName: html-data
```

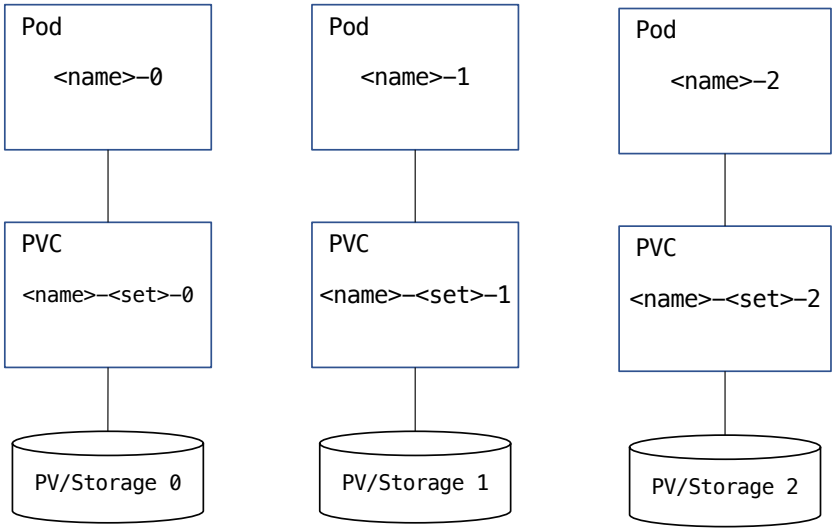
StatefulSet

```yaml
apiVersion: v1
kind: Service
metadata:
  name: <svc-name>
spec:
  clusterIP: None
  ports:
  - name:
    ...
  selector:
    app: <name>
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: <name>
spec:
  serviceName: <svc-name>
  replicas: 3
  ...
  template:
    metadata:
      labels:
        app: <name>
    spec:
      containers:
      - name: ...
        ...
        volumeMounts:
        - name: <pvc-name>
          mountPath: ...
  volumeClaimTemplates:
  - metadata:
      name: <pvc-name>
    spec:
      accessModes:
      - ReadWriteMany
      resources:
        requests:
          storage: xxGi
```

Headless-Service (keine Cluster-IP, kein Loadbalancing)

Host-Name/DNS A-Record für jeden Pod: <name-#>.<svc-name>.<project>.svc.cluster.local

CNAME <svc-name>.<project>.svc.cluster.local + SVR-Records für jeden Pod



(compute) Resources :

- Memory: number of bytes (quantity suffixes: E, P, T, G, M, k | Ei, Pi, Ti, Gi, Mi, Ki)
- CPU : millicores (m) … fractions of *time* of a single CPU (not the fraction of number of CPUs).

```
apiVersion: v1
kind: Pod
metadata:
...
spec:
  containers:
  - name: <name>
    resources:
      requests:
        memory: 64Mi
        cpu: 100m
      limits:
        memory: 128Mi
        cpu: 200m
```

Scheduling

Execution
(cgroups)

```
$ oc describe node master01
...
Allocatable:
  cpu:                3500m
  memory:             15268156Ki
Non-terminated Pods: (60 in total)
   CPU Requests CPU Limits Memory Requests Memory Limits
...
Allocated resources:
Resource           Requests      Limits
  --------           --------      ------
  cpu                2397m (68%)   0 (0%)
  memory             9347Mi (62%)  512Mi (3%)
```

resouce usage

Liveness / Readiness / Startup Probes

- liveness : Container wird bei negativen Ergebnis neu gestartet `.spec.containers.livenessProbe`
- readiness: Route/Service wird aktiviert/deaktiviert `.spec.containers.readinessProbe`
- startup: liveness/readiness sind deaktiviert bis startup positiv ist `.spec.containers.startupProbe`
  Container wird bei neg. Startup-Probe sofort beendet

Probes:

```
exec:
  command:
  - cat
  - /tmp/ready
initialDelaySeconds: 5
periodSeconds: 5
```

```
httpGet:
  path: /healthz
  port: healthz-port
  schema: https
  httpHeaders: ...
failureThreshold: 1
periodSeconds: 10

200 <= status < 400
```

```
tcpSocket:
  port: 5432
initialDelaySeconds: 15
periodSeconds: 20
```

- initialDelaySeconds: Zeitdauer bis zur ersten liviness/readiness Probe
- periodSeconds: Intervall zur Ausführung der Proben (default 10 sec)
- timeoutSeconds: max. Timeout bei einer Probe (default 1 sec)
- successThreshold: Schwellwert ab wann aufeinderfolgende positive Proben als Erfolg gewertet werden (default 1)
- failureThreshold: Schwellwert ab wann aufeinderfolgende negative Proben als Ausfall gewertet werden (default 3)

```yaml
kind: Deployment
apiVersion: apps/v1
metadata:
  name: webserver
spec:
  ...
  template:
    spec:
      containers:
      - name: webserver
        image: webserver
        imagePullPolicy: Always
        ports:
        - name: http
          containerPort: 8080
          protocol: TCP
        readinessProbe:
          failureThreshold: 3
          httpGet:
            path: /healthz
            port: http
            scheme: HTTP
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 1
        ...
```

nginx.conf

```
server {
  listen 8080 default_server;
  server_name _;
  location / {
    root   /usr/share/nginx/html;
    index  index.html index.htm;
  }

  location /healthz {
    access_log off;
    return 200;
  }

}
```

Multi Container App / Staging

Template: parametrisierbare Liste von Resource-Definitionen

```yaml
kind: Template
apiVersion: v1
metadata:
  name: rest-sample
objects:
- apiVersion: v1
  kind: Service
  metadata:
    name: ${APP_NAME}
  spec:
    selector:
      app.kubernetes.io/name: ${APP_NAME}
   ...
- apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: ${APP_NAME}
  spec:
    template:
      spec:
        containers:
        - name: ${APP_NAME}
          image: ${IMAGE_NAME}
...
- apiVersion: v1
  kind: Route
...
parameters:
- description: Application Name
  name: APP_NAME
  required: true
- description: Image Name
  name: IMAGE_NAME
  required: true
...
```

```
oc process (TEMPLATE | -f FILENAME) -p APP_NAME=... | oc create -f -

oder bei installiertem Template ( oc create -f template.yml) :

oc new-app <template-name>
```

```
apiVersion: template.openshift.io/v1
kind: Template
metadata:
  name: webserver
  labels:
    template: webserver
labels:
  app: webserver
parameters:
- name: stage
  required: true
- name: version
  value: latest
objects:
- apiVersion: v1
  kind: Deployment
  metadata:
    name: webserver
    namespace: stage-${stage}
    ...
       containers:
        - name: webserver
          image: registry.straubcloud.de/webserver:${version}
...
- apiVersion: v1
  kind: Route
  ...
  spec:
    host: webserver-${stage}.ocp.straubcloud.de
  ...
```

- **Namespace stage-test**

```
$ oc process -f webserver.yml -p stage=test | oc apply -f -
deployment.apps/webserver created
service/webserver created
route.route.openshift.io/webserver created
```

- **Namespace stage-prod**

```
$ oc process -f webserver.yml -p stage=prod -p version=2.0 | oc apply -f -
deployment.apps/webserver created
service/webserver created
route.route.openshift.io/webserver created
```

Helm-Chart: Paket-Manager (Lifecycle + Template-Engine + Dependencies)

```
$ helm create sample
Creating sample

$ tree sample
sample
├── charts
├── Chart.yaml
├── templates
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── NOTES.txt
│   ├── serviceaccount.yaml
│   ├── service.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml
```

helm create

Helm-Chart: Paket-Manager (Lifecycle + Template-Engine + Dependencies)

Chart.yml

```
apiVersion: v1
name: sample
description: Sample Application
version: 1.0
appVersion: 1.0
dependencies:
- name: dep1
  version: ...
  repository: ...
```

values.yml

```
image:
  repository: quay.io/redhat.io/sample
  tag: '2'

service:
  port: 8080

env:
  ...


dep1.key: value
```

Templates:

deployment.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ APP_NAME }}
spec:
  template:
    selector:
      matchLabels:
        {{- include "sample.selectorLabels" . | nindent 6 }}
     spec:
       containers:
         - image: {{.Values.image.repository}}: {{.Values.image.tag}}
 ...
```

Go-Templates:

_helpers.tpl

```
{{- define "sample.selectorLabels" -}}
app.kubernetes.io/name: {{ include "sample.name" . }}
app.kubernetes.io/instance: {{ .Release.Name }}
{{- end }}
...
```

```
helm create
helm dependency update
helm install / upgrade / rollback / uninstall

helm template (lokales processing)
```

Helm: Verwendung unterschiedlicher Stages

```
webserver
├── Chart.yaml
├── stage-prod
│   ├── env.properties
│   └── values.yml
├── stage-test
│   ├── env.properties
│   └── values.yml
├── templates
  ...
└── values.yaml
```

➢ **Namespace stage-test**

```
$ helm install webserver . -f stage-test/values.yml -n stage-test
NAME: webserver
LAST DEPLOYED: ...
NAMESPACE: stage-test
STATUS: deployed
REVISION: 1

$ helm upgrade webserver . -f stage-test/values.yml --set image.tag=1.0
Release "webserver" has been upgraded. Happy Helming!
NAME: webserver
LAST DEPLOYED: ...
NAMESPACE: stage-test
STATUS: deployed
REVISION: 2
```

➢ **Namespace stage-prod**

```
$ helm install webserver . -f stage-prod/values.yml -n stage-prod
NAME: webserver
LAST DEPLOYED: ....
NAMESPACE: stage-prod
STATUS: deployed
REVISION: 1
```

Kustomize: generieren/transformieren von Resourcen (Manifeste mit minimalen Meta-Daten)

kustomization.yml

```
kind: Kustomization
apiVersion: kustomize.config.k8s.io/v1beta1

namespace: sample

resources:
- deployment.yml
- service.yml
- route.yml
- http://...    -> kustomize.yml in Git/Web-Repository


images:
- name: sample
  newName: registry/sample
  newTag: '5'

commonLabels:
  app.kubernetes.io/instance: sample

configMapGenerator:
- name: rest-sample
  literals:
  - LAUNCH_JBOSS_IN_BACKGROUND=1
...
```

resources → https://github.com/hashicorp/go-getter#url-format

deployment.yml

```
apiVersion: apps/v1
metadata:
  name: rest-sample
spec:
  replicas: 1
  template:
    spec:
      containers:
      - name: sample
        image: sample
```

```
$ oc kustomize <kustom-dir>
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app.kubernetes.io/instance: rest-sample
  name: rest-sample
  namespace: sample
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/instance: sample
  template:
    containers:
      image: registry/sample:5
...

$ oc apply -k .
```

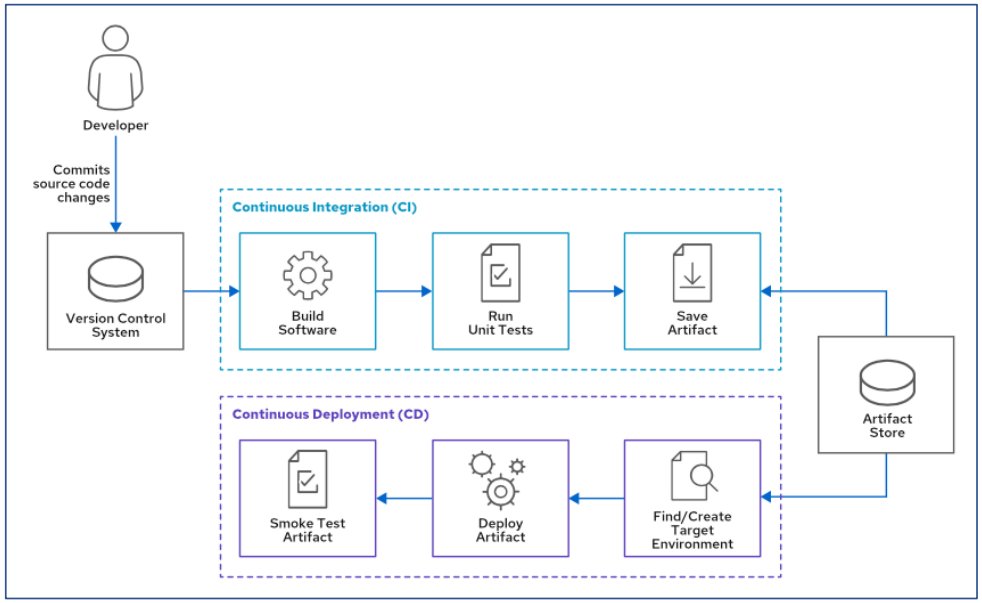
Kustomize Overlays : erzeugen unterschiedlicher Varianten von einer Basis-Vorlage

```
                                        base/kustomization.yml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- deployment.yml
- service.yml
- route.yml
```

```
                                overlays/test/kustomization.yml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- ../../base

namespace: test

images:
- name: sample
  newName: registry/sample
  newTag: '3-SNAPSHOT'
```

```
                          overlays/production/kustomization.yml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
- ../../base

namespace: production

images:
- name: sample
  newName: registry/sample
  newTag: '5'
```

```
$ oc apply -k overlays/test
service/sample configured
deployment.apps/sample configured
route.route.openshift.io/sample configured

$ oc apply -k overlays/production
...
```



Continuos Integration
Continuos Delivery

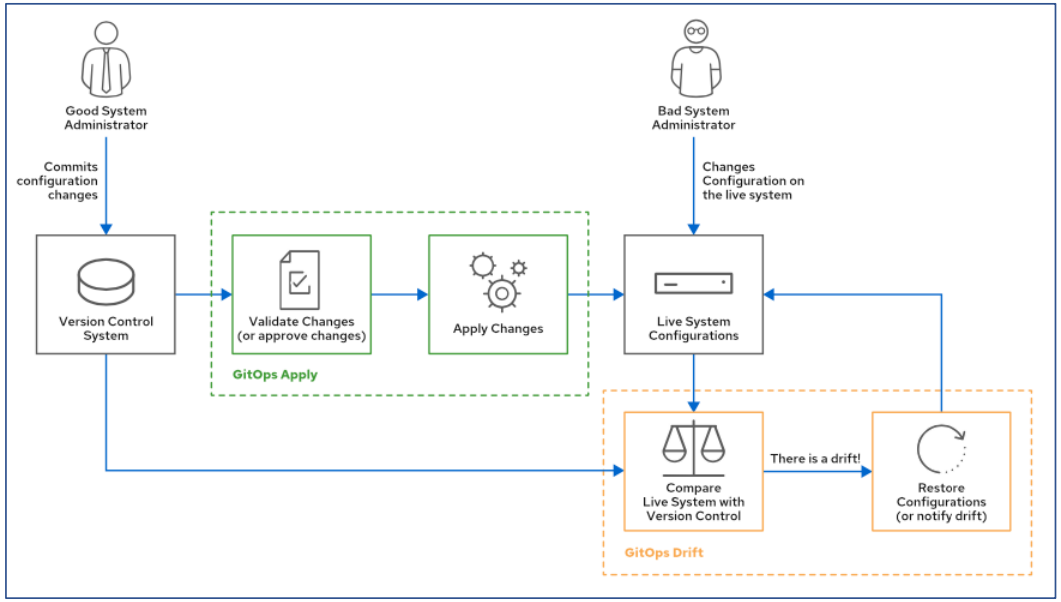
→ Developer
→ running application

Jenkins, CruiseControl, TeamCity, GitLab ...
Kubernetes native (Tekton, Argo CD, ...)

GitOps Workflow

→ Administrators
→ live System

Ansible, Puppet, Terraform ...
ArgoCD, FluxCD, JenkinsX



GitOps – Workflow mit Pipelines:

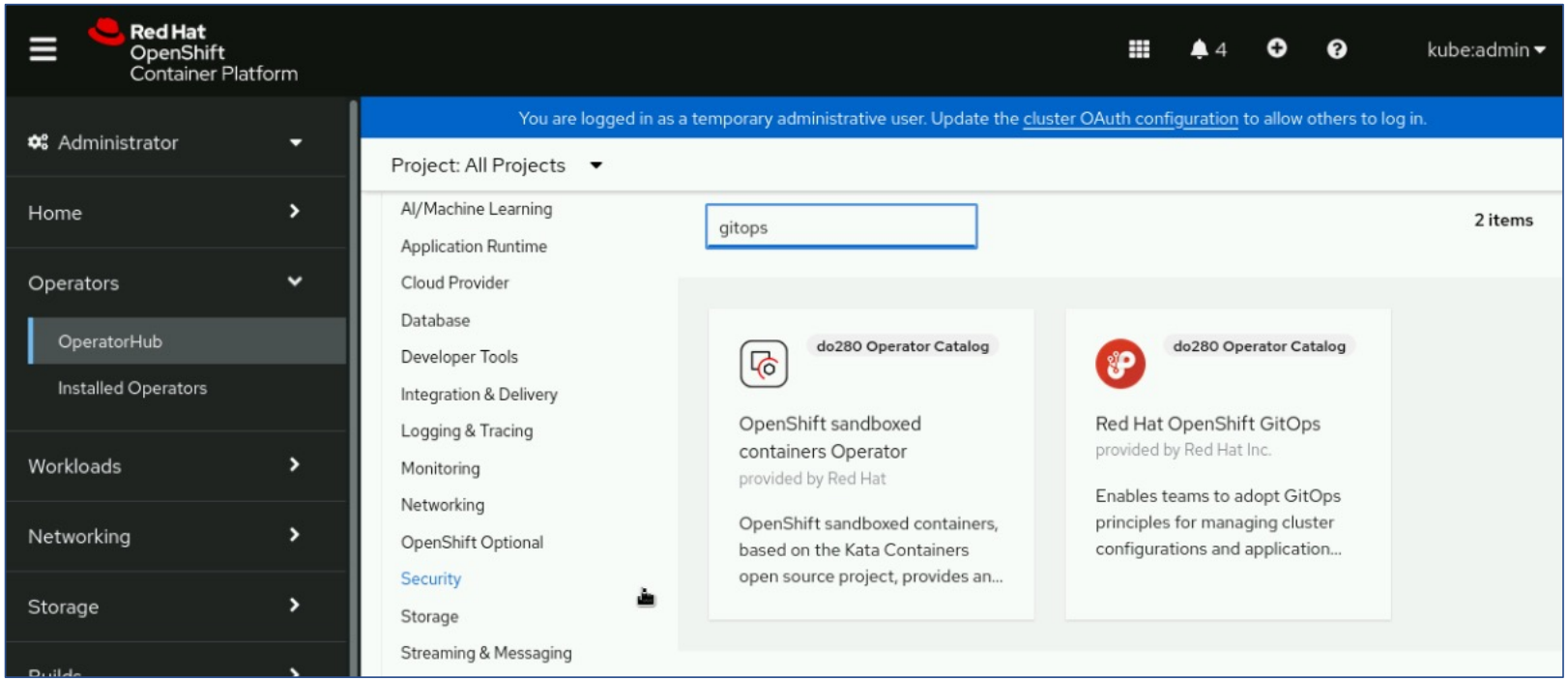
- Apply Pipeline:
  - `validate : oc apply --validate --dry-run [ folder/files from Git ]`
  - `apply    : oc apply`

- Drift Pipeline:
  - `diff     : oc diff [ folder/files from Git ]`
  - `optional/restore: oc apply`

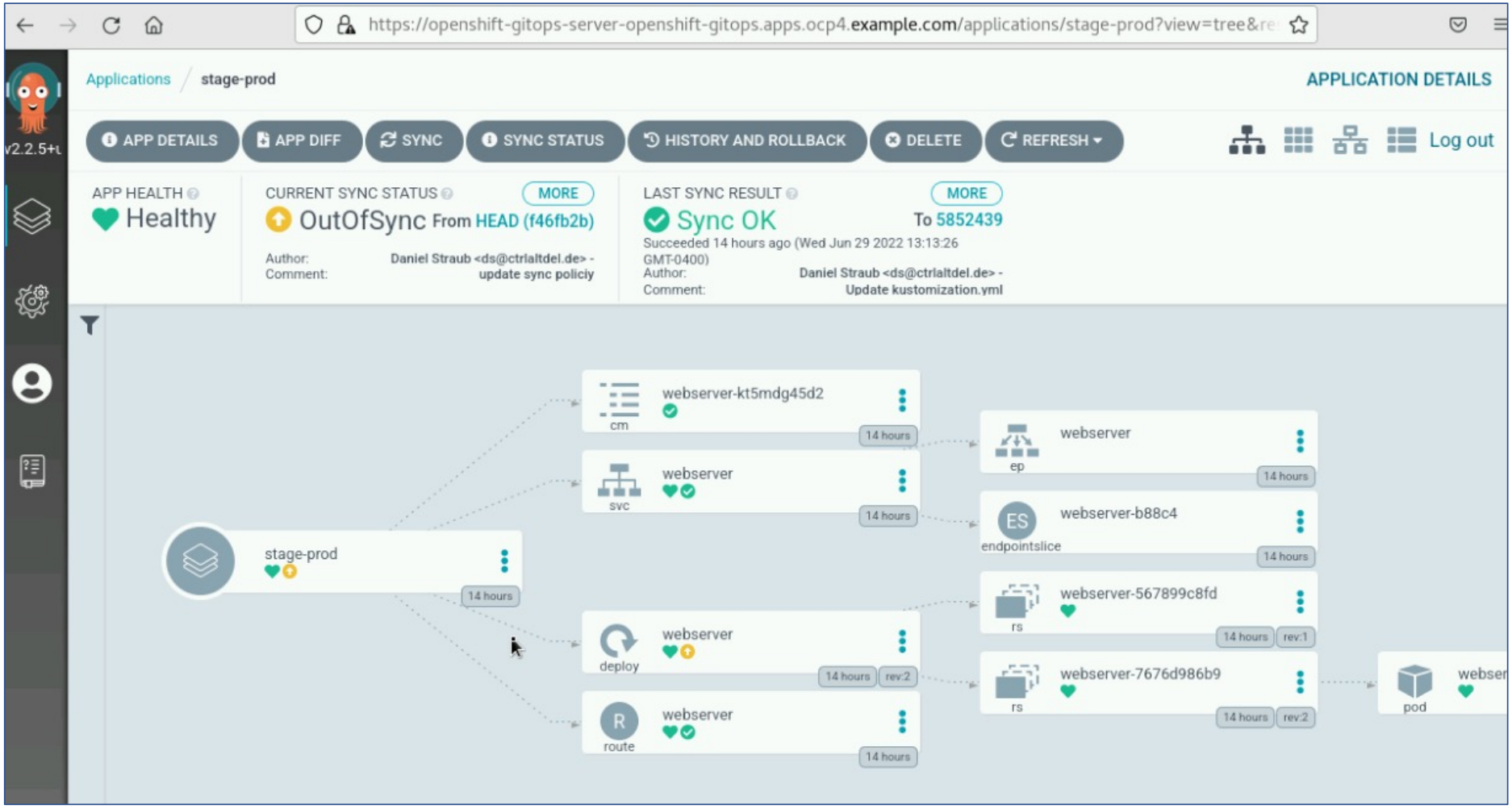
GitOps – Workflow mit ArgoCD / FluxCD:
Abgleich Ist-Zustand (Cluster) mit Kustomize/Helm-Definitionen im Git
Benachrichtigungen, manueller/automatische Synchronisation bei Abweichungen

apps calibre	ssh://git@gitea.apps:10022/ds/calibre.git/overlays/production in-cluster/apps	HEAD	Healthy Synced	⋮
apps pgadmin	ssh://git@gitea.apps:10022/ds/pgadmin.git/overlays/production in-cluster/apps	HEAD	Healthy Synced	⋮
apps postgres	ssh://git@gitea.apps:10022/ds/postgres.git/overlays/production in-cluster/database	HEAD	Healthy Synced	⋮
apps rest-sample	ssh://git@gitea.apps:10022/ds/rest-sample.git/overlays/production in-cluster/sample	HEAD	Healthy OutOfSync	⋮

Red Hat Openshift GitOps - Operator



ArgoCD = Openshift GitOps



Openshift GitOps