Workshop “Kubernetes”

kubernetes

AT Computing
(Vijfhart Group)

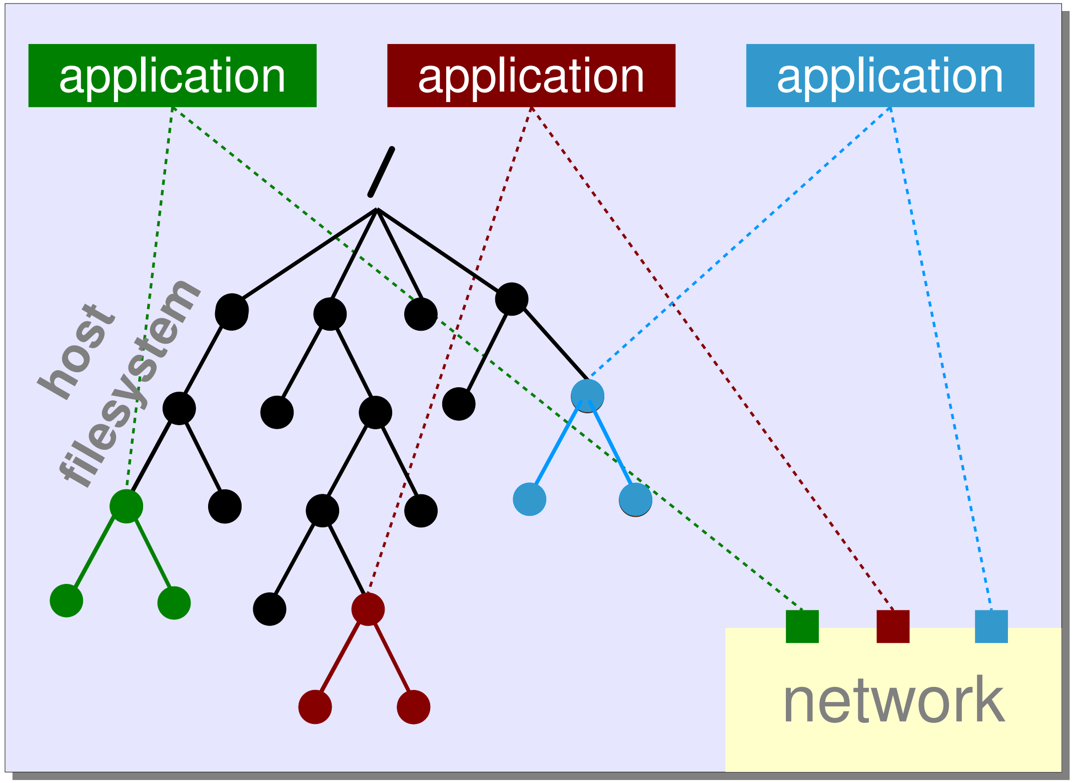
Introduction containers (1)

Conventional production environment

- all applications use same ‘ecosystem’
  - filesystem
  - additional storage
  - network
  - process numbers (PIDs)
  - user identities
  - hardware resources

- disadvantages
  - laborius (de)installation of application
  - compromised application might
    - access all data
    - manipulate entire network stack
    - compromise other applications
  - application might overload resources

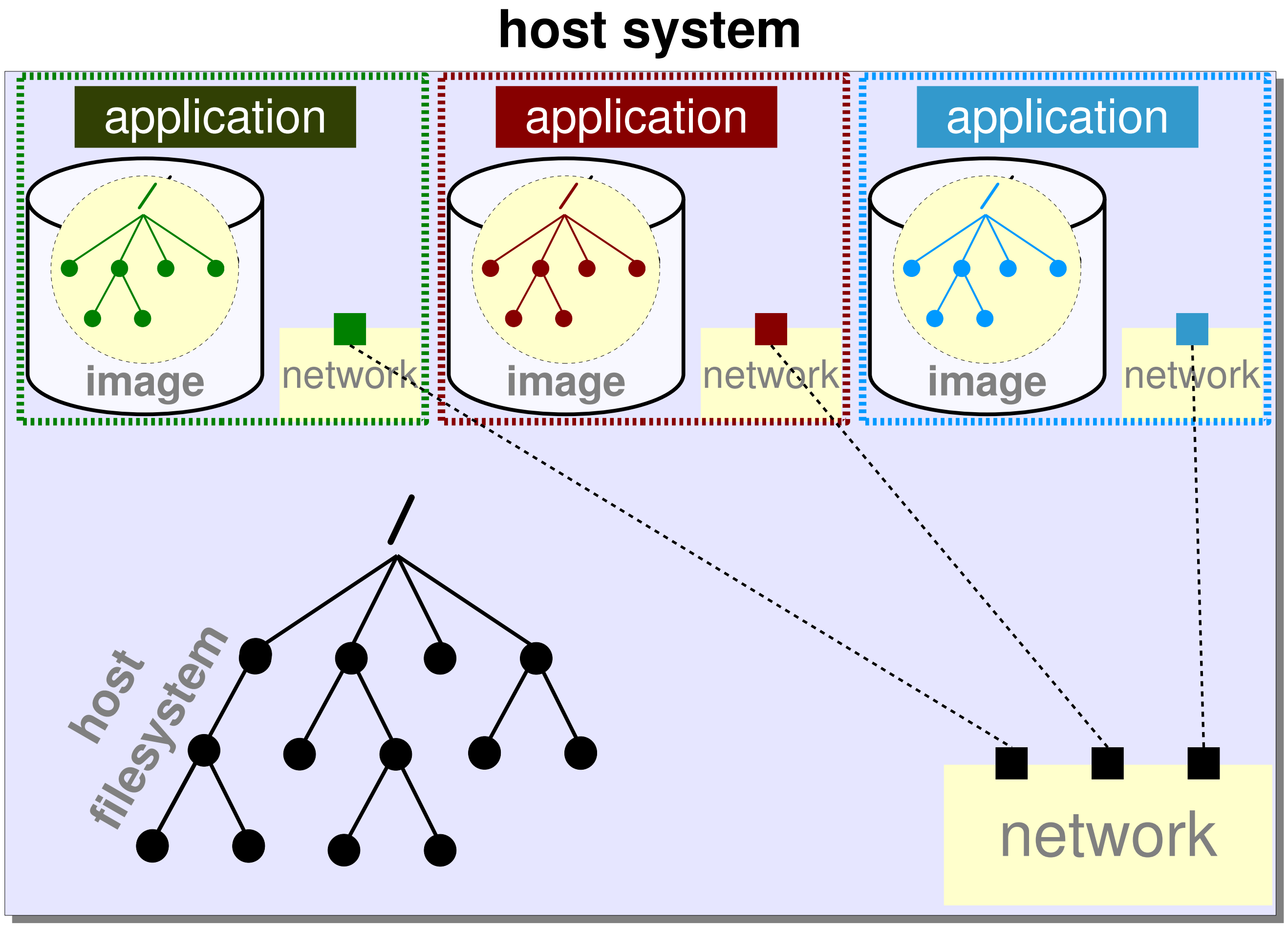
host system



Introduction containers (2)

Container: isolated ecosystem for application

- image with own mini-filesystem containing
  - programs
  - configuration files
  - data files
  - ...
- own additional storage
- own network
- own PID numbers
- own user identities
- cpu/memory/disk limitations



Introduction containers (3)

Image

- needed to start container on destination host

- contains
  - mini-filesystem
  - metadata

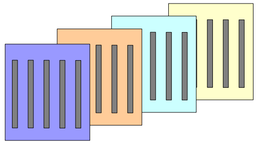
- *developer* (‘dev’)
  - builds image for application
  - ships image to registry

- *operations* (‘ops’)
  - pulls image from registry
  - uses image to activate container
    - in any environment: test, acceptance, production
    - on any operating system: Linux, Windows, macOS
    - on any platform: physical host, virtual machine, cloud

Container summary using Docker

Container summary

Docker container



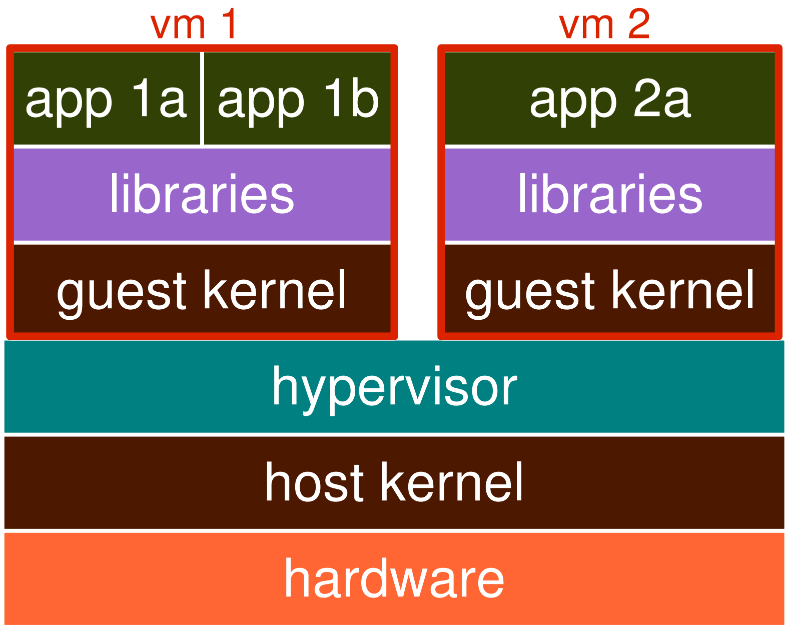
- run application in isolated environment
  - own mini- filesystem
  - own network
  - own private PID numbers
  - own mounted filesystems
  - own users
  - separate root privileges
  - cpu/memory limitation

- lightweight
- simple
- large community (lots of images)

Container summary – virtual machines vs. containers

Virtual machine

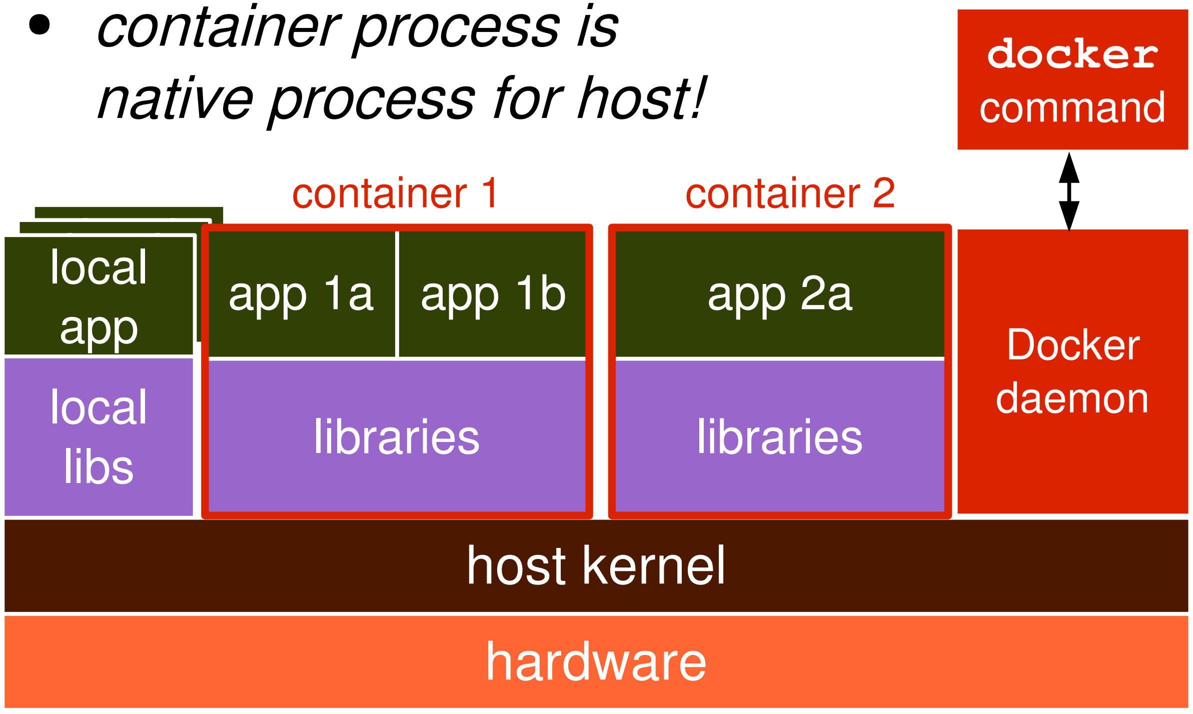
- application
- libraries
- full command set
- full operating system (kernel)



Virtual machines

Docker container

- application
- libraries
- limited command set
- no private kernel
- *container process is native process for host!*



Docker containers

Container summary – run from base image

Start container from base image

- example: run command in base image **ubuntu** from Docker registry

image overruling command

```
$ docker run  ubuntu  ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 08:59 ?        00:00:00 ps -f
$
```

  - container terminates when process in container terminates

- example: run command in base image **ubuntu** interactively

interactive tty

```
$ docker run  -it  ubuntu  bash
root@f8d91cbaca03:/# ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  1 09:12 ?        00:00:00 bash
root        11     1  0 09:12 ?        00:00:00 ps -f
root@f8d91cbaca03:/# exit
$
```

Container summary – build custom image

Build custom image

- specify own modifications in file **`Dockerfile`**

```
$ cat Dockerfile
FROM ubuntu:18.04
RUN  apt-get update && apt-get install -y apache2
COPY index.html /var/www/html/index.html
CMD  ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

```
$ cat index.html
<h1> Message from container! </h1>
```

- build custom image

new image

directory containing **`Dockerfile`**
and other files needed in image

```
$ docker build  -t atcomp/apachetest  .
Successfully built 5d3b567581df
```

- list images

```
$ docker images
REPOSITORY           TAG      IMAGE ID       CREATED              SIZE
atcomp/apachetest    latest   5d3b567581df   About an hour ago    268.1 MB
docker.io/ubuntu     18.04    104bec311bcd   3 months ago         128.9 MB
```

Container summary – run from custom image

Start container from custom image

- run custom container

publish port *detached*: run in background

```
$ docker run  -p 8080:80  -d  atcomp/apachetest
c8eda1f3a6734304195ab3e47280ee77c719fa5c365ba428bc2037e250754c2e
```

SHA256 (often abbreviated with first 48 bits)

- contact webserver via URL **`http://localhost:8080`** (web browser/**`curl`**)

```
$ curl http://localhost:8080
<h1> Message from container! </h1>
```

- show running containers

```
$ docker ps
CONTAINER ID   IMAGE                COMMAND                  .... PORTS
c8eda1f3a673   atcomp/apachetest    "/usr/sbin/apache2ctl"        0.0.0.0:8080->80/tcp
```

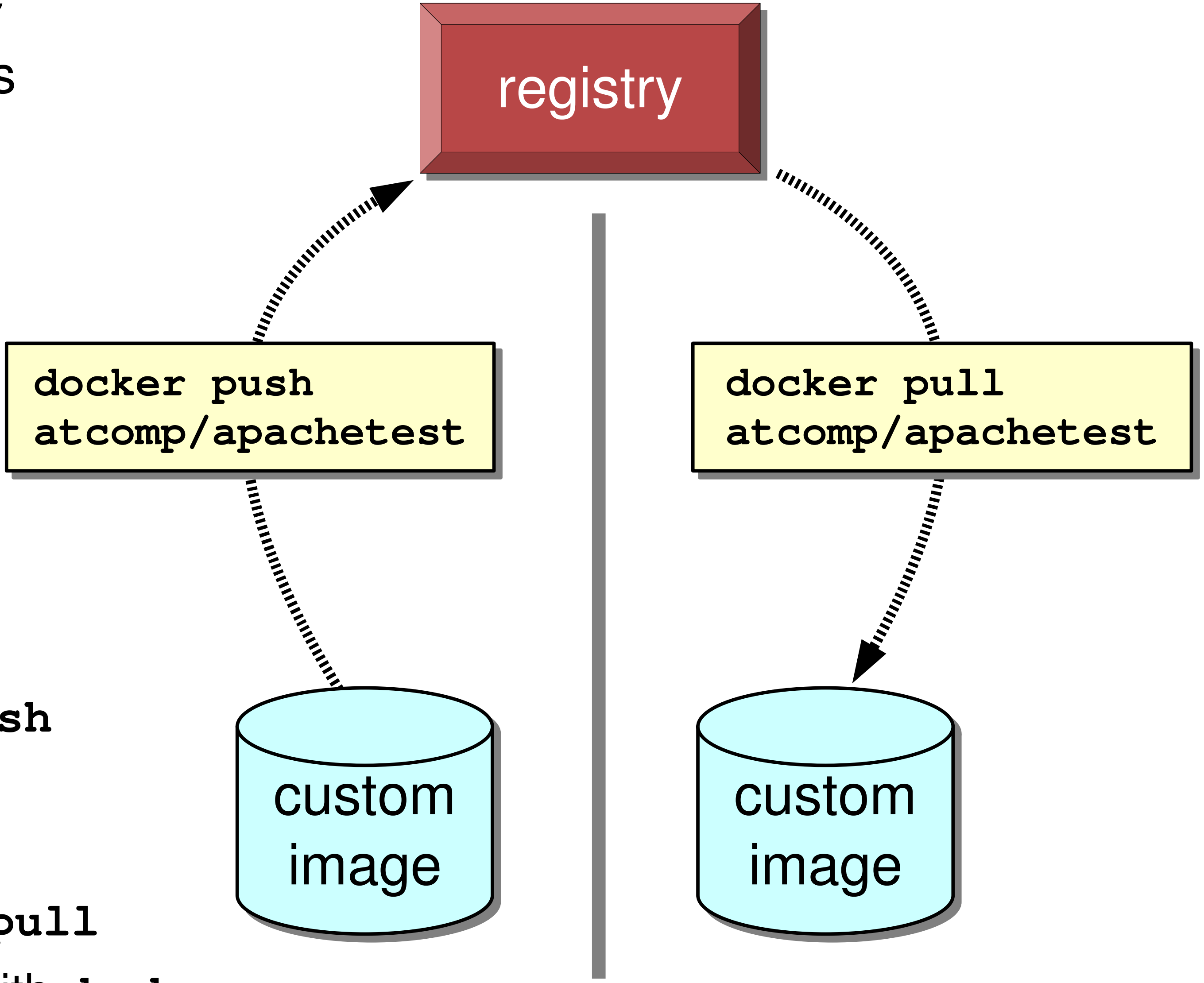
- terminate running container

```
$ docker stop c8eda1f3a673
c8eda1f3a673
```

Container summary – registries

Docker registry: image store

- server(s) containing images
  - multiple versions
- possibilities
  - Docker Hub (100,000+)
  - in-company registry
- images can be
  - *pushed*
    - stored with **`docker push`**
  - *pulled*
    - explicitly with **`docker pull`**
    - implicitly on initial use with **`docker run`**
    - implicitly on initial use with **`FROM`** in **`Dockerfile`**



Workshop “Kubernetes”

kubernetes

Introduction

What is Kubernetes?

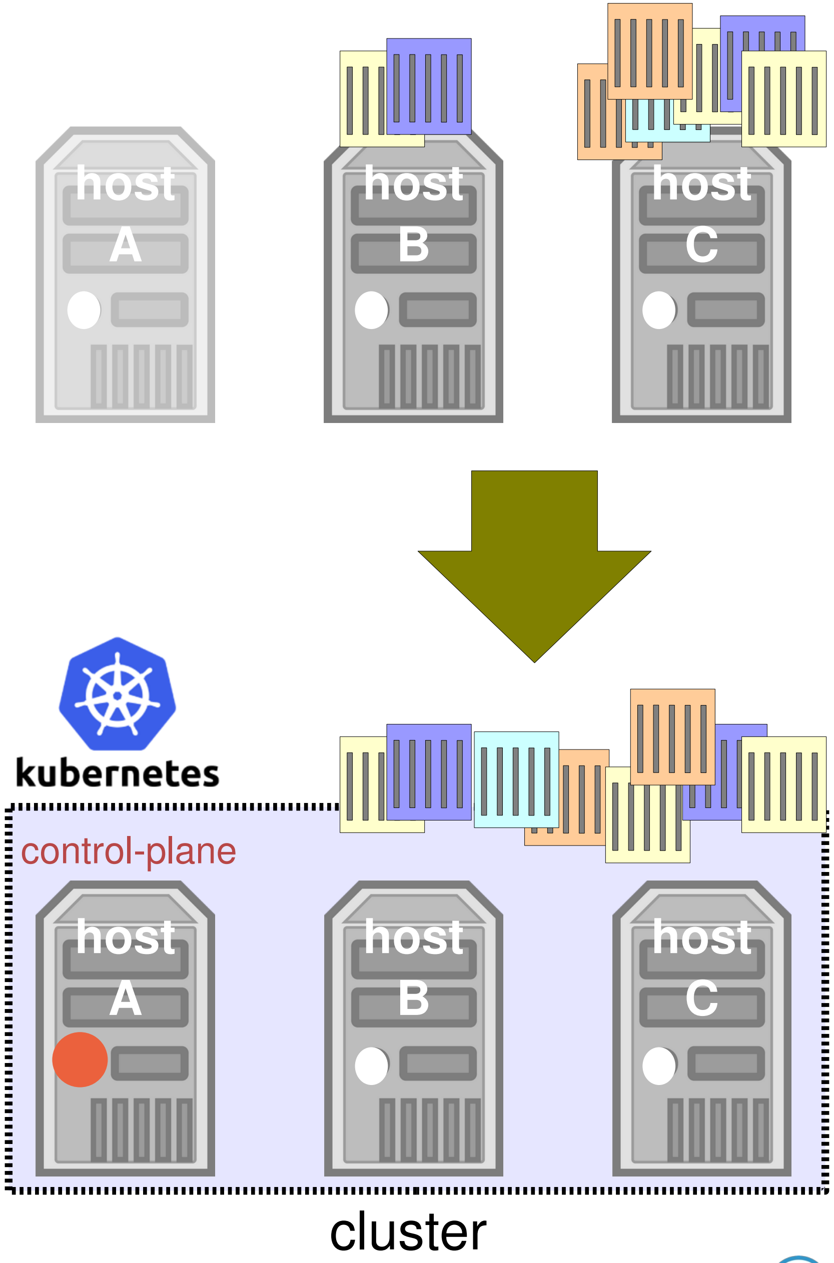
Kubernetes (‘helmsman’) a.k.a. K8s

- combines various hosts into cluster
  - scalability
  - reliability (failover)
- orchestrates containers
  - activate
  - monitor
  - terminate
- can manage various container implementations, like *containerd*, *CRI-O*, *Docker*, ...

Kubernetes orchestration

Kubernetes orchestration

- introduced in 2014, inspired by Google's Borg
- maintained by *Cloud Native Computing Foundation* (CNCF)
- open source
- concept
  - cluster needs at least one *control-plane node,* formerly called *master node*
  - other hosts in cluster are *worker nodes*
    - only run container instances

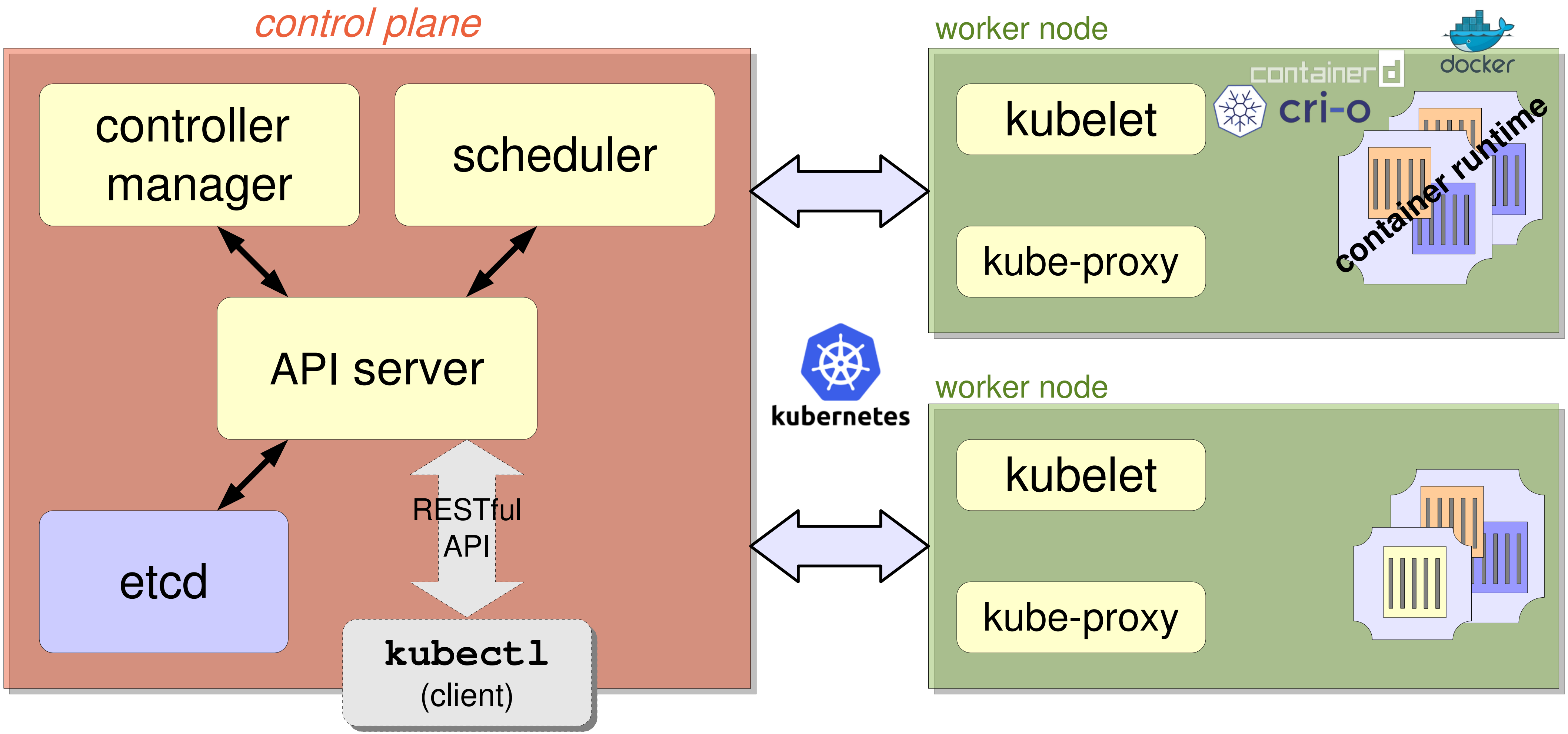


User interface

User interfaces

- command line interface: **`kubectl`** ***subcommand*** ***object*** **[options]**
  - subcommands
    - create, modify and delete objects: **`create`**, **`apply`**, **`delete`**
    - query current state of objects: **`get`**, **`describe`**
    - other actions, like: **`logs`**, **`exec`**, ...
  - overview of subcommands: **`kubectl`**
    more info about subcommand: **`kubectl`** ***subcommand*** **`--help`**
- graphical user interface
  - also valid for cloud implementations, like
    - Google Kubernetes Engine (GKE)
    - Amazon Elastic Kubernetes Service (EKS)
    - Azure Kubernetes Service (AKS)

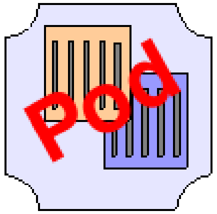
Kubernetes architecture



- *API server*: controls entire cluster
- *scheduler*: schedules containers (via pods) on nodes
- *controller manager*: controls required number of replicas
- *etcd*: distributed key-value store to maintain current cluster state

- *kubelet*:
  - pod startup & monitoring
  - resource management
- *kube-proxy*: container exposure to network & load balancing

Pods



Pod

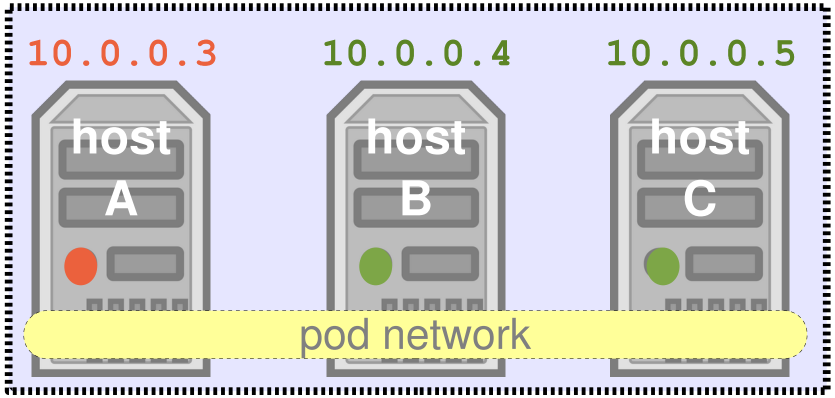
- conceptually smallest unit, running one application

- consists of one or more closely related containers
  - typically one container running primary application
  - additional containers supporting primary application (if needed)

- containers in same pod
  - run on same worker node

  - share *storage volumes*
    - allows persistent data
    - allows containers to access same storage

  - share *network*
    - same IP address on pod network, same ports
    - allows containers to communicate via `localhost`



Pods – startup, status and termination

Pod lifetime

- activate pod specifying command line parameters

```
$ kubectl run apatest --restart=Never --image=atcomp/apachetest --port=80
pod/apatest created
```

or

- activate pod via manifest file

```
$ kubectl apply -f apa.yml
pod/apatest created
```

apa.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: apatest
spec:
  containers:
  - name: apacont
    image: atcomp/apachetest
    ports:
    - containerPort: 80
  restartPolicy: Never
```

- verify pod status and delete pod

```
$ kubectl get pods
NAME     READY   STATUS    RESTARTS   AGE
apatest 1/1     Running   0          82s

$ kubectl get pods -o wide
NAME     READY  STATUS    RESTARTS AGE  IP           NODE ....
apatest 1/1    Running  0        83s  10.244.1.254 hostb

$ kubectl delete pod/apatest
pod "apatest" deleted
```

Pod completion

After container termination

- pod is ‘completed’ but not removed

```
$ kubectl run testpod --restart=Never --image=ubuntu -- sleep 10
pod/testpod created

$ kubectl get pods
NAME       READY   STATUS      RESTARTS   AGE
testpod    1/1     Running     0          6s

$ kubectl get pods                                   after a while...
NAME       READY   STATUS      RESTARTS   AGE
testpod    0/1     Completed   0          17s
```

  - logs can still be watched
  - events can still be verified

- pods are not ‘self-healing’
  - *controller* needed to restart completed pod (covered later)

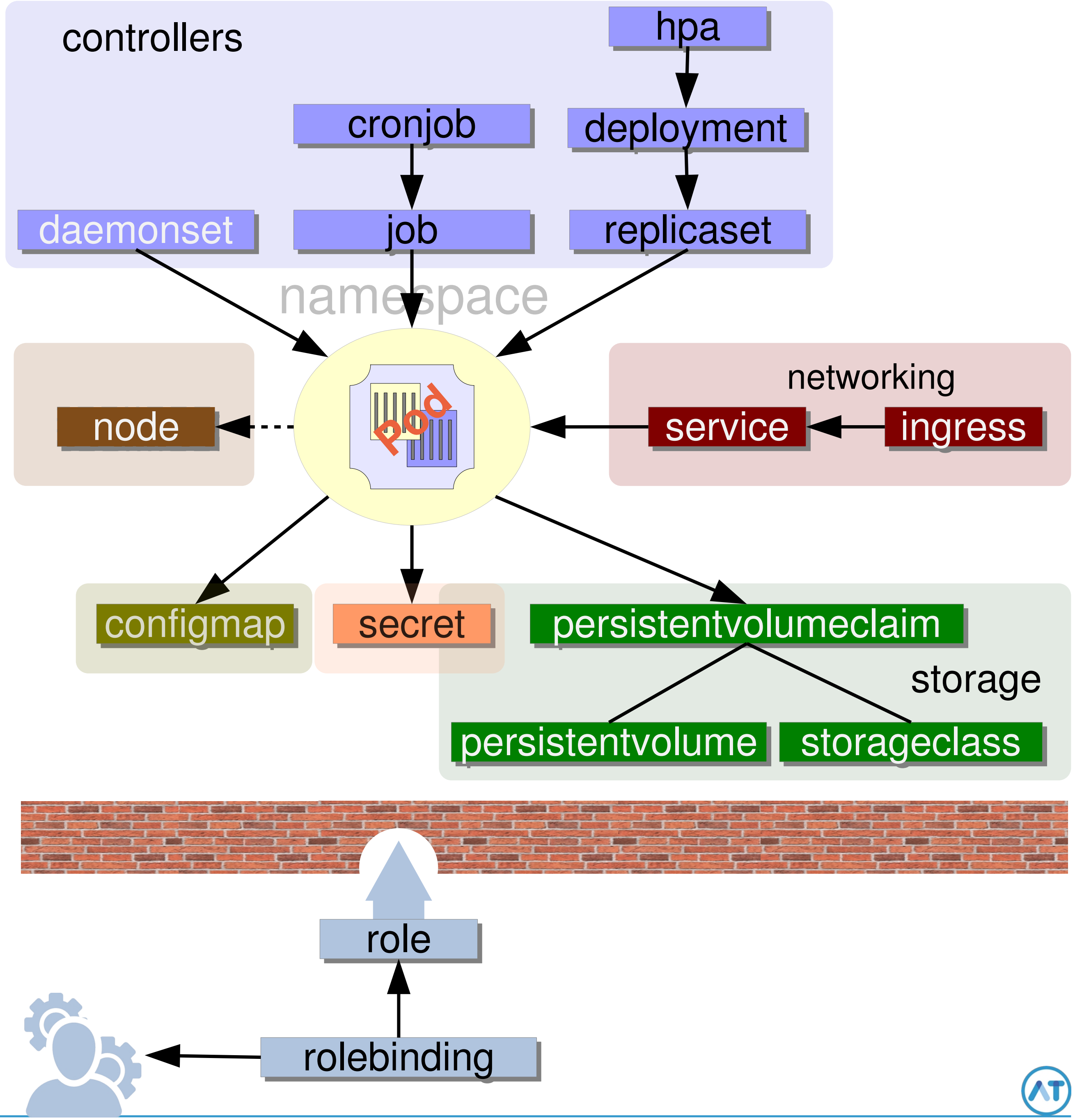
- interactive pod (**–it**)

```
$ kubectl run testpod -it --restart=Never --image=ubuntu -- bash
```

Kubernetes objects (1)

Kubernetes concept

- numerous object types
- every object has
  - type ('kind')
  - unique name
- object refers to other objects by using
  - labels
    (most references *to* pods)
    or
  - object names
    (most references *from* pods)



Kubernetes objects (2)

Kubernetes object

- has *desired state* and *actual state* (current state)
  - control plane continuously tries to match *actual state* to *desired state*
  - desired state defined by **`spec`** in manifest file
- manifest consists of

`apiVersion:`	version of API
`kind:`	type of object
`metadata:`	object name (must be unique) optional labels for selection
`spec:`	specification of desired state, depending on type

apa.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: apatest
  labels:
    app: webserver
spec:
  containers:
  - name: apacont
    image: atcomp/apachetest
    ports:
    - containerPort: 80
  restartPolicy: Never
```

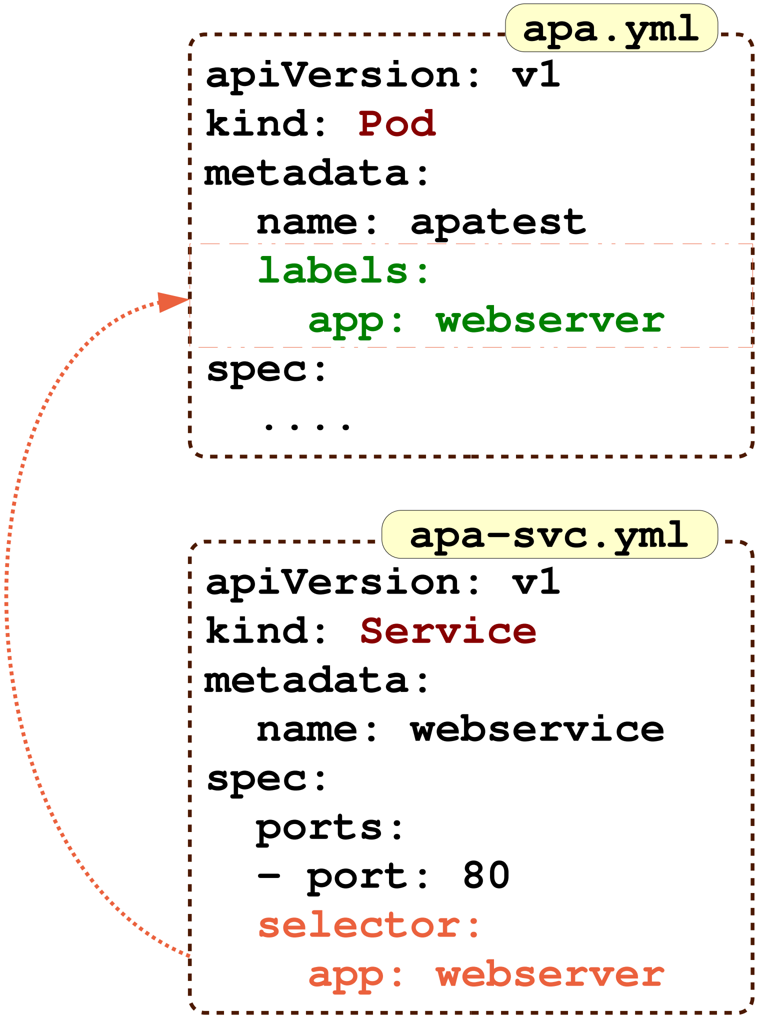
Labels

Object labels

- every object has unique name
- additionally, *labels* can be assigned to be used for
  - selection on command line with **-l** flag

```
$ kubectl get pods    -l app=webserver

$ kubectl delete all -l app=webserver
```

  - refer from one object to another object
    e.g. to assign **Service** object to **Pod** object



apa.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: apatest
  labels:
    app: webserver
spec:
  ....
```

apa-svc.yml

```
apiVersion: v1
kind: Service
metadata:
  name: webservice
spec:
  ports:
  - port: 80
  selector:
    app: webserver
```

Namespaces

Namespaces: subdivide cluster into various virtual clusters

- per project, per application, per developer team, per department, per ....

```
$ kubectl create ns devel
namespace/devel created

$ kubectl get ns
NAME              STATUS    AGE
default           Active    278d
devel             Active    14s
....
```

```
$ kubectl apply -f apa.yaml -n devel             in namespace devel

$ kubectl get pods -n devel                      in namespace devel
NAME        READY     STATUS      RESTARTS     AGE
apatest     1/1       Running     0            48s
```

- allow
  - separate scope for object names
  - limitation on resource utilization (cpu, memory, storage, number of pods, ...)

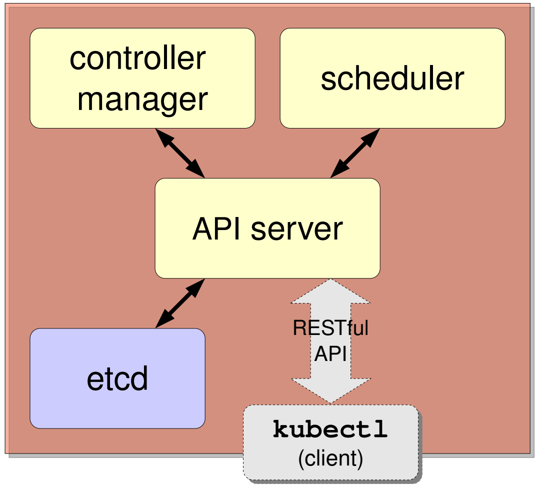
kubernetes

Controllers

Controllers – an introduction

Controllers

- pod ('naked pod', 'bare pod') is not self-healing
- pods usually started under supervision of *controller*
  - create, manage, monitor and restart pods
  - provide rolling updates
- part of *controller manager* in control plane
- manifest file of controller requires
  - controller-specific definitions
  - template of pod (to be controlled)



```
apiVersion: apps/v1
kind: somecontroller
metadata:
  name: sleeper
  ....
spec:
  controller-specific stuff...
  template:
    metadata:
      labels:
        app: webserver
    spec:
      containers:
      - name: snorecont
        image: ubuntu
        command: ["sleep", "60"]
```

pod template

Controller types

Controller types

- *ReplicaSet* (rs) – preferably combined with Deployment
  - pod replicas and restart of failing pods

- *Deployment* – recommended
  - pod replicas by using *ReplicaSet*
  - rolling updates and rollbacks

Deployment – create

Deployment object

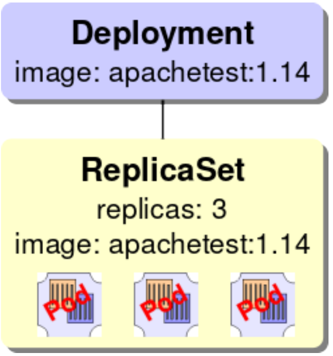
- definition implies ReplicaSet and Pod
- create deployment

```
$ kubectl apply -f apa-deploy.yml
deployment.apps/apadep created

$ kubectl get deployment
NAME     DESIRED   CURRENT   UP-TO-DATE   AVAILABLE ...
apadep   3         3         3            3

$ kubectl get rs
NAME                DESIRED   CURRENT   READY    AGE
apadep-7b6fc56c77   3         3         3        18s

$ kubectl get pods
NAME                       READY  STATUS    RESTARTS ...
apadep-7b6fc56c77-6ldcv    1/1    Running   0
apadep-7b6fc56c77-bqxhr    1/1    Running   0
apadep-7b6fc56c77-hk7qp    1/1    Running
```

apa-deploy.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apadep
  labels:
    app: webserver
spec:
  replicas: 3
  selector:
    matchLabels:
      app: apapod
  template:
    metadata:
      labels:
        app: apapod
    spec:
      containers:
      - name: apacont
        image: atcomp/apachetest:1.14
        ports:
        - containerPort: 80
```

pod template



Deployment – rolling update

Rolling update

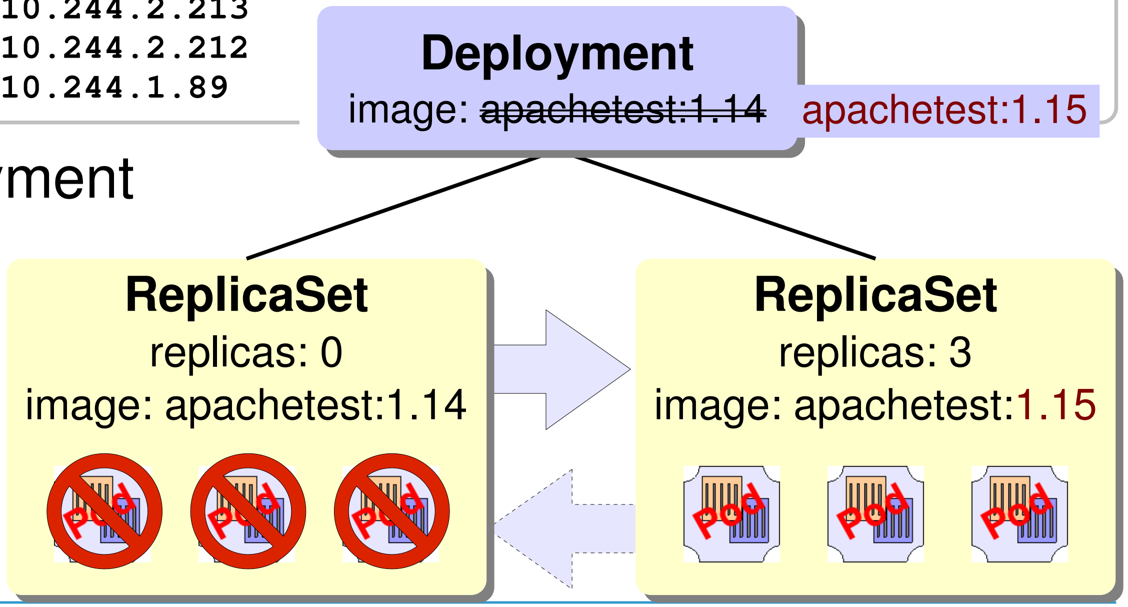
- executed when pod specification changes, like new image

```
$ kubectl set image deployment/apadep  apacont=atcomp/apachetest:1.15
deployment.apps/apadep image updated
```

```
$ kubectl get all -o wide
NAME                    READY  UP-TO-DATE AVAILABLE CONTAINERS IMAGES                 ...
deployment.apps/apadep  3/3    3          3         apacont    atcomp/apachetest:1.15

NAME                           DESIRED CURRENT READY CONTAINERS IMAGES                 ...
replicaset/apadep-5b4f756b5c   3       3       3     apacont    atcomp/apachetest:1.15
replicaset/apadep-7b6fc56c77   0       0       0     apacont    atcomp/apachetest:1.14

NAME                          READY STATUS   IP            ...
pod/apadep-5b4f756b5c-5kvrx   1/1   Running  10.244.2.213
pod/apadep-5b4f756b5c-rflpn   1/1   Running  10.244.2.212
pod/apadep-5b4f756b5c-z5zbj   1/1   Running  10.244.1.89
```

- creates new replicaset within deployment
  - pod replicas replaced one-by-one
  - original replicaset preserved for rollback

```
$ kubectl rollout undo \
          deployment/apadep
```



Deployment – scaling

Deployment – scaling

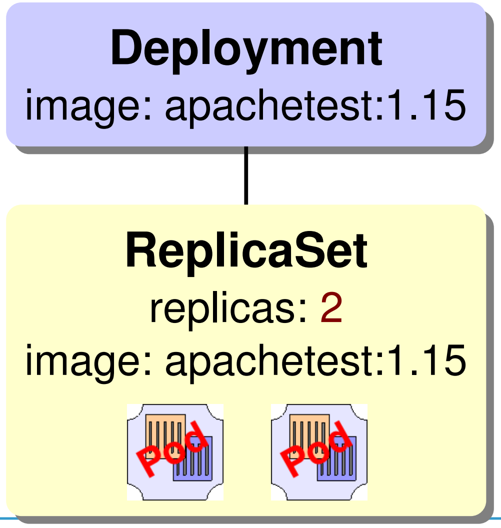
- modify number of replicas

```
# kubectl scale --replicas=2 deployment/apadep
deployment.extensions/apadep scaled
```

```
$ kubectl get all -o wide
NAME                          READY   STATUS    RESTARTS   AGE     IP            NODE
pod/apadep-5b4f756b5c-5kvrx   1/1     Running   3          4m20s   10.244.2.213  hostc
pod/apadep-5b4f756b5c-rflpn   1/1     Running   3          4m19s   10.244.1.212  hostb

NAME                     DESIRED CURRENT UP-TO-DATE AVAILABLE CONTAINERS   IMAGES
deployment.apps/apadep   2       2       2          2         apacont      atcomp/apachetest:1.15

NAME                                DESIRED   CURRENT   READY   CONTAINERS   IMAGES
replicaset.apps/apadep-5b4f756b5c   2         2         2       apacont      atcomp/apachetest:1.15
replicaset.apps/apadep-7b6fc56c77   0         0         0       apacont      atcomp/apachetest:1.14
```

- replicas can even be scaled to 0 (temporarily no pods)

- autoscaling possible with HPA controller
  - minimum and maximum number of replicas
  - target CPU time utilization per replica



Deployment
image: apachetest:1.15

ReplicaSet
replicas: 2
image: apachetest:1.15

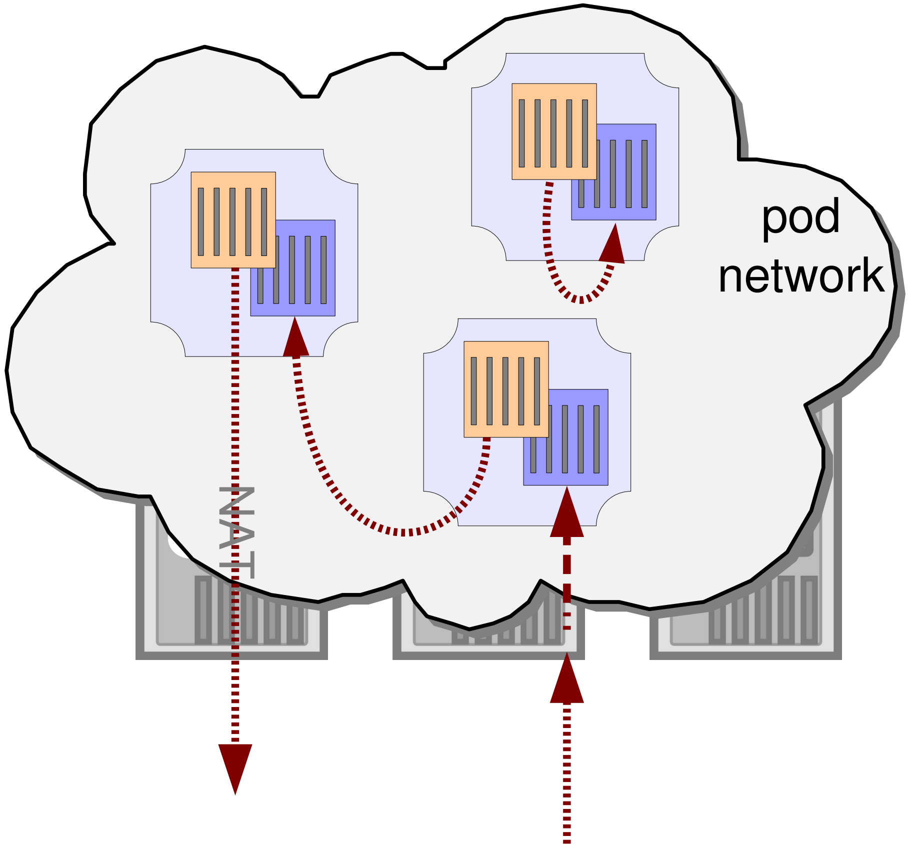
kubernetes

Networking

Kubernetes networking

Communication possibilities

- container-to-container in same pod
  - via `localhost` (loopback interface)
  - port numbers may conflict between applications
- pod-to-external
  - via Network Address Translation (NAT)
- pod-to-pod
  - what is IP address of destination pod?
- external-to-pod
  - what is IP address of destination pod?



Services – IP address of pod

Example: access via dynamic IP address

- create deployment with 2 pod replicas

```
$ kubectl apply -f apa-deploy.yml
deployment.apps/apadep created

$ kubectl get pods -o wide
NAME                      READY ... IP            NODE
apadep-8654d77c94-928tw  1/1         10.244.1.130  hostb
apadep-8654d77c94-ggv8c  1/1         10.244.2.64   hostc

$ curl 10.244.1.130
<h1> Message from container! </h1>
```

- disadvantages
  - no access from outside cluster
  - no load balancing
  - when pod terminates, it probably gets another IP address after restart

```
$ kubectl delete pod/apadep-....-928tw
$ kubectl get pods -o wide
NAME                      READY    IP            NODE
apadep-8654d77c94-7br4b   1/1      10.244.1.131  hostb
apadep-8654d77c94-ggv8c   1/1      10.244.2.64   hostc
```

apa-deploy.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apadep
  labels:
    app: webserver
spec:
  replicas: 2
  selector:
    matchLabels:
      app: apapod
  template:
    metadata:
      labels:
        app: apapod
    spec:
      containers:
      - name: apacont
        image: atcomp/apachetest:1.14
        ports:
        - containerPort: 80
```

Services – internal access

Service object

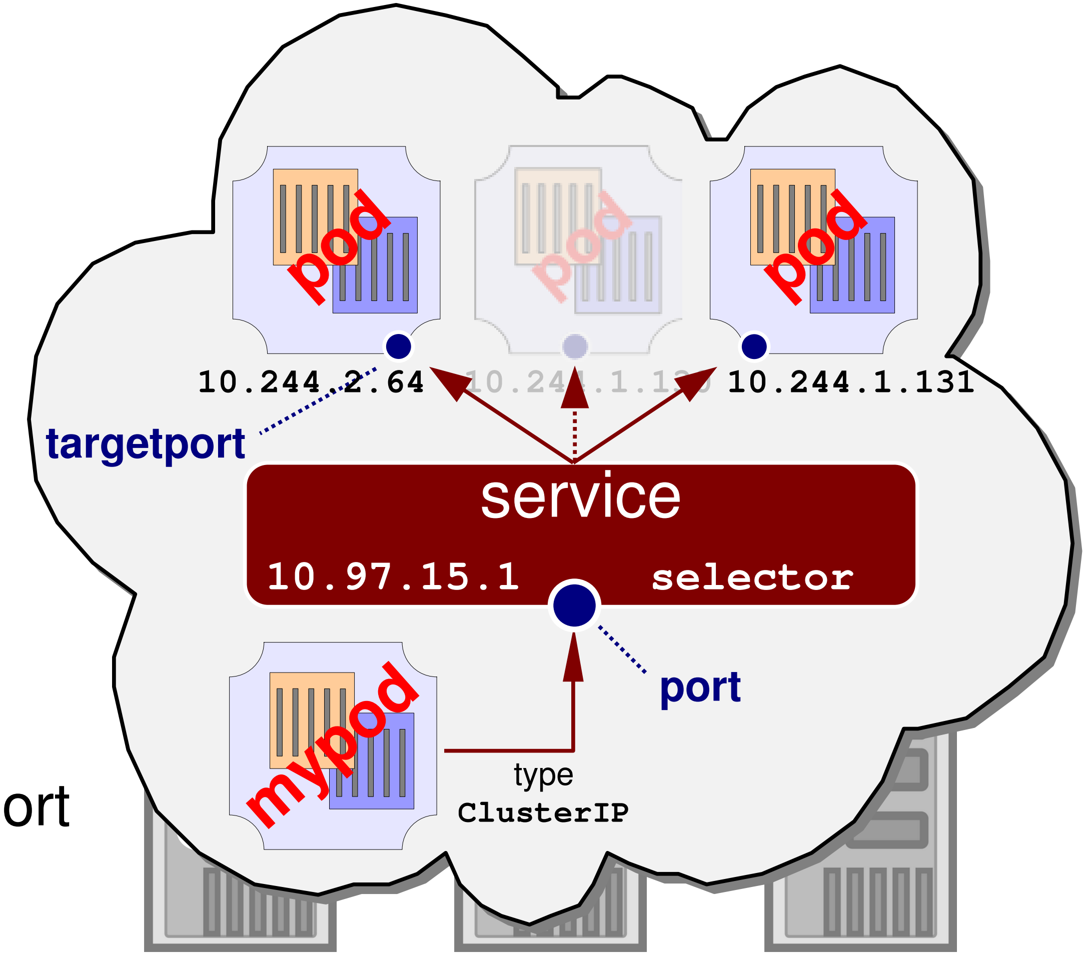
- gets static virtual IP (VIP) address, though dynamically assigned
  - stable IP address to reach mortal pods
  - load balancing

- contains selector
  - refers to label of pod to attach to
  - endpoint object created per target port

- accessibility determined by *type*

  `ClusterIP`: routable within cluster (default) for internal access

  `NodePort`: static port on every node in cluster for external access

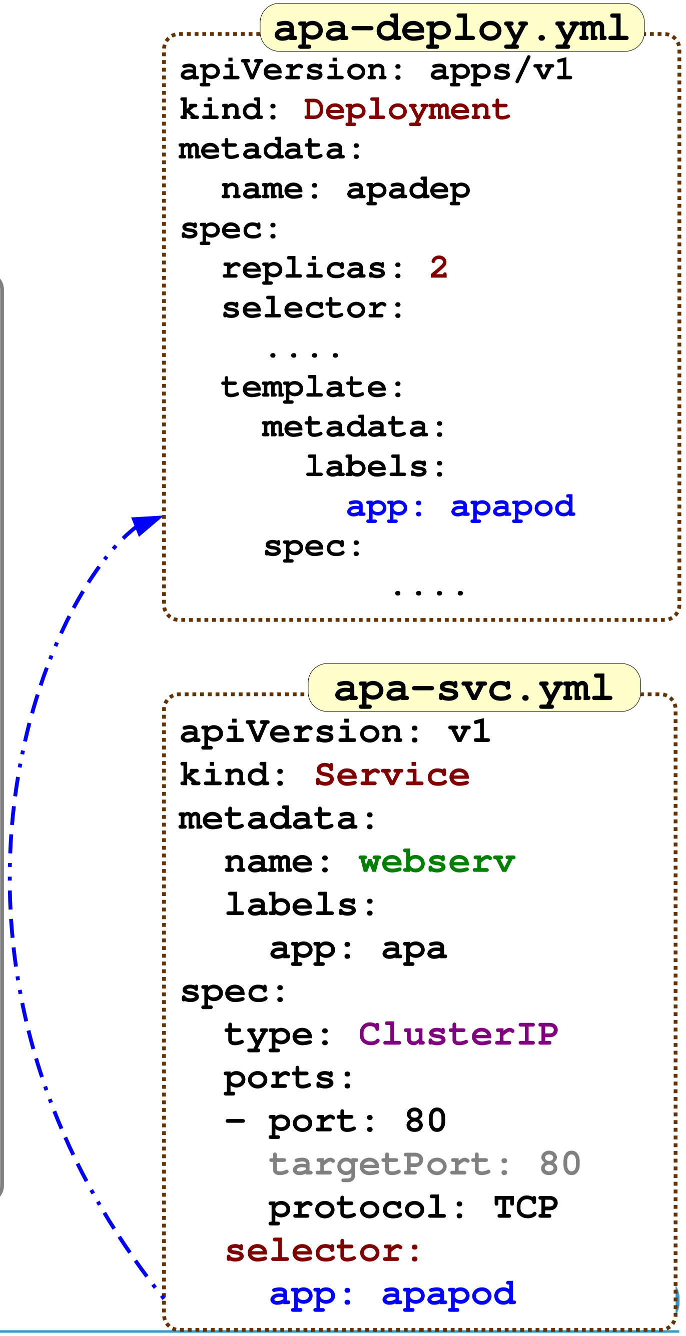


Services – internal: setup `ClusterIP`

Example: add *internal* service for webserver

- create service referring to label **`app=apapod`**
- type **`ClusterIP`** to provide internal access

```
$ kubectl apply -f apa-svc.yml
service/webserv created

$ kubectl get svc
NAME         TYPE         CLUSTER-IP     .... PORT(S)    ....
webserv      ClusterIP    10.97.15.1          80/TCP

$ kubectl get ep
NAME          ENDPOINTS                            AGE
webserv       10.244.1.131:80,10.244.2.64:80       67m

$ kubectl get all
NAME                           DESIRED  CURRENT  UP-TO-DATE  AVAILABLE AGE
deployment.apps/apadep         2        2        2           2         10m

NAME                                        DESIRED  CURRENT  READY    AGE
replicaset.apps/apadep-8654d77c94           2        2        2        10m

NAME                                  READY    STATUS      RESTARTS       AGE
pod/apadep-8654d77c94-7br4b           1/1      Running     0              10m
pod/apadep-8654d77c94-ggv8c           1/1      Running     0              10m
```



apa-deploy.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apadep
spec:
  replicas: 2
  selector:
    ....
  template:
    metadata:
      labels:
        app: apapod
    spec:
      ....
```

apa-svc.yml

```
apiVersion: v1
kind: Service
metadata:
  name: webserv
  labels:
    app: apa
spec:
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
  selector:
    app: apapod
```

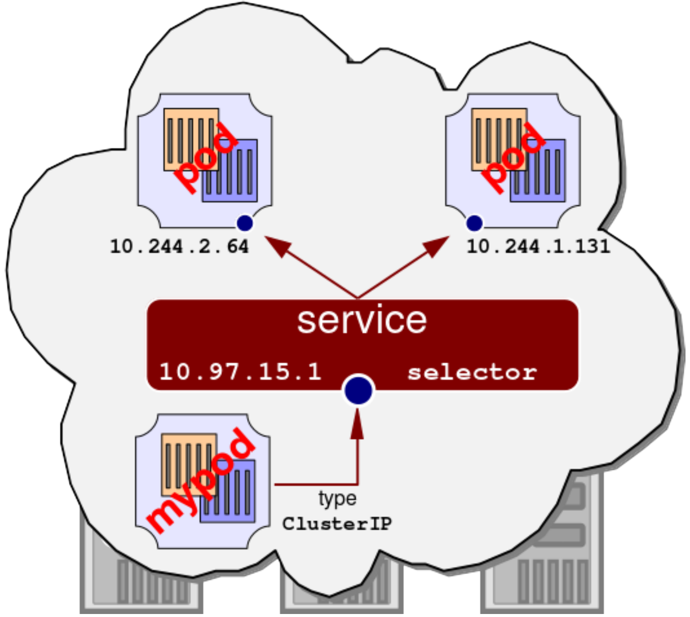
Services – internal: discovery

Service discovery by other pods

- via internal DNS
- maintains record for every service:

  service[.***ns***.svc.cluster.local]

- preferred (always available)!



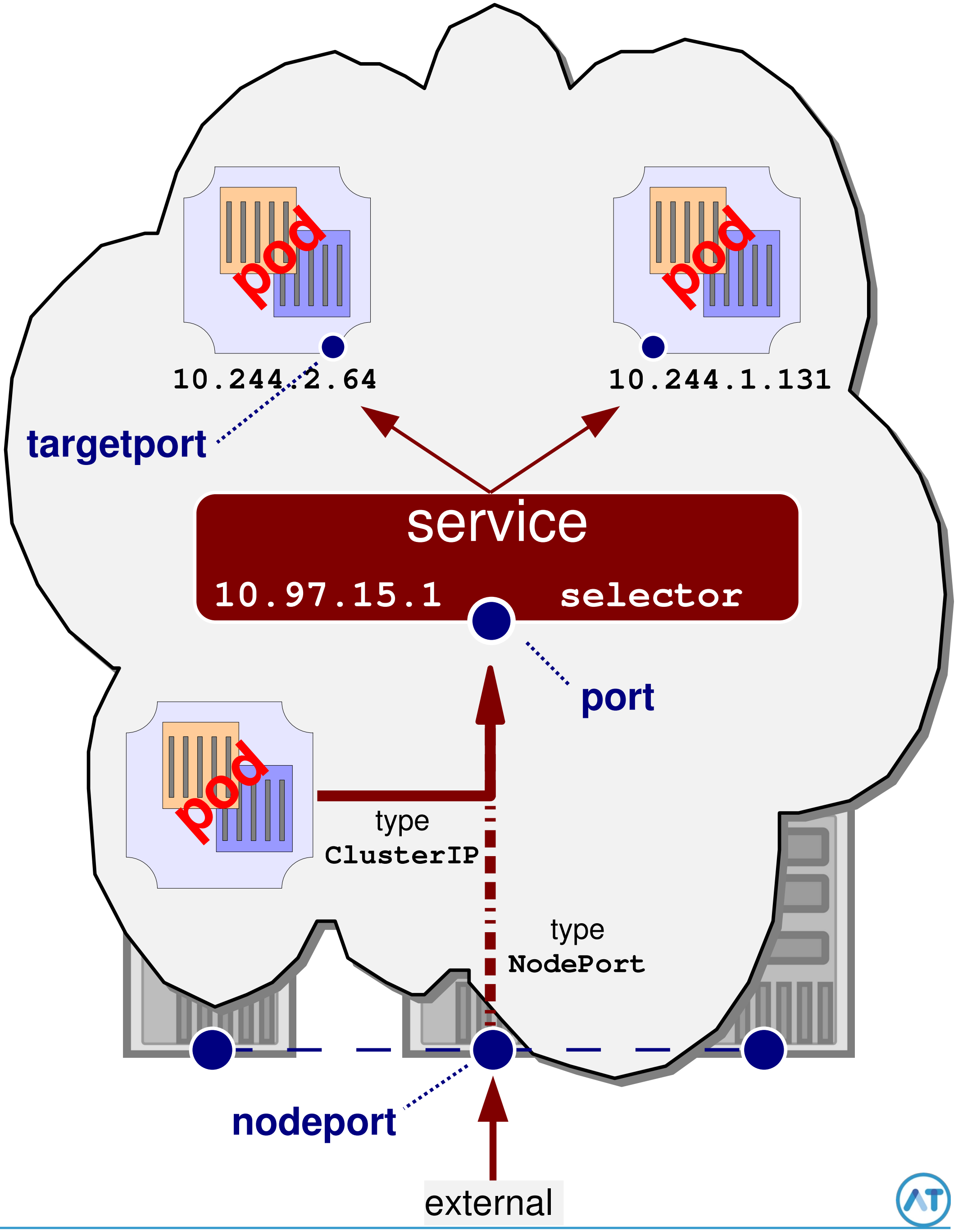
```
$ kubectl run mypod -it --restart=Never --image=atcomp/nwubuntu
root@mypod:/# cat /etc/resolv.conf
nameserver 10.96.0.10
search default.svc.cluster.local svc.cluster.local cluster.local

root@mypod:/# host webserv
webserv.default.svc.cluster.local has address 10.97.15.1

root@mypod:/# curl webserv
<h1> Message from container! </h1>
```

Services – external access

External access to pod

- requires service type **`Nodeport`** (implies service type **`ClusterIP`**)

- port range 30000-32767
  - specify with keyword **`nodePort`**
  - dynamically assigned when keyword **`nodePort`** omitted



Services – external: setup `NodePort`

Example: add *external* service for webserver

- create service

```
$ kubectl apply -f apa-svcn.yml
service/webserv created

$ kubectl get svc
NAME        TYPE        CLUSTER-IP     .... PORT(S)    ....
webserv     NodePort    10.97.15.1          80:32123/TCP

$ ss -tl
State  Recv-Q Send-Q Local Address  Foreign Address
LISTEN     0      0     [::]:32123          [::]:*
```

- access from any host outside cluster

```
anyhost$ curl hosta:32123        or hostb or hostc
<h1> Message from container! </h1>
```

- access from inside cluster (similar to `ClusterIP`)

```
$ kubectl run mypod -it --restart=Never
                        --image=atcomp/nwubuntu
root@mypod:/# curl webserv
<h1> Message from container! </h1>
```

apa-deploy.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apadep
  labels:
    app: apa
spec:
  replicas: 2
  selector:
    ....
  template:
    metadata:
      labels:
        app: apa
    spec:
      containers:
        ....
```

apa-svcn.yml

```
apiVersion: v1
kind: Service
metadata:
  name: webserv
  labels:
    app: apa
spec:
  type: NodePort
  ports:
  - port: 80
    nodePort: 32123
    protocol: TCP
  selector:
    app: apa
```

Workshop “Kubernetes”

kubernetes

Storage

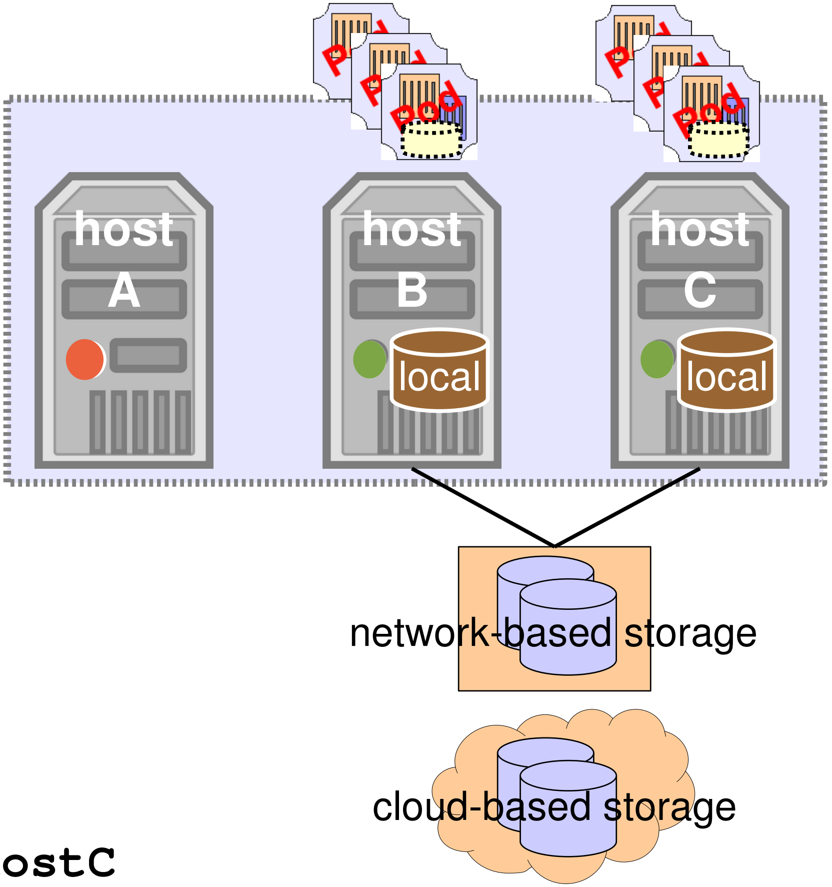
Kubernetes Volumes

Storage

- *stateless pod* – preferred!
  - no need to preserve data
  - container storage (filesystem) on local disk

- *stateful pod*
  - requires persistent volume
  - volume can be mapped on
    - local disk
      - restricted use
    - network storage
      - pod terminated on **`hostB`** might be restarted on **`hostC`**

        and/or
      - pods running on different hosts might share same storage



Volumes

Volumes

- have specific type

- have explicit lifetime
  (life span of pod, permanent, ...)

- in pod manifest

  `spec.volumes:`

  provided volumes of certain type

  `spec.containers.volumeMounts:`

  mount point in container

```
apiVersion: v1
kind: Pod
metadata:
  name: ....
spec:
  containers:
  - name: ....
    image: ....
    volumeMounts:
    - name: myvol
      mountPath: /mnt

  volumes:
  - name: myvol
    volumetype
```

Volumes – shared between pods on same node

Persistent shared volume on node

- persistent storage in filesystem of host
  - notice: only host on which pod is created!
  - read/write access

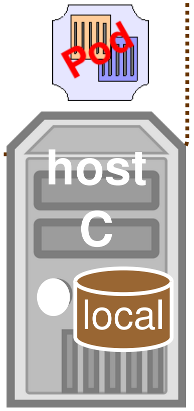
  solution: **`hostPath`** volume

- example: run pod on specific host

```
$ kubectl get nodes --show-labels
NAME    STATUS   ROLES ... LABELS
hosta   Ready    master    kubernetes.io/hostname=hosta
hostb   Ready    <none>    kubernetes.io/hostname=hostb
hostc   Ready    <none>    kubernetes.io/hostname=hostc

$ kubectl apply -f hostpath-pod.yml

$ kubectl get pod/apahost -o wide
NAME      READY   STATUS    RESTARTS  AGE  IP            NODE
apahost   1/1     Running   0         19m  10.244.2.240  hostc

$ curl 10.244.2.240
<H1> Welcome to hostc! </H1>
```

hostpath-pod.yml

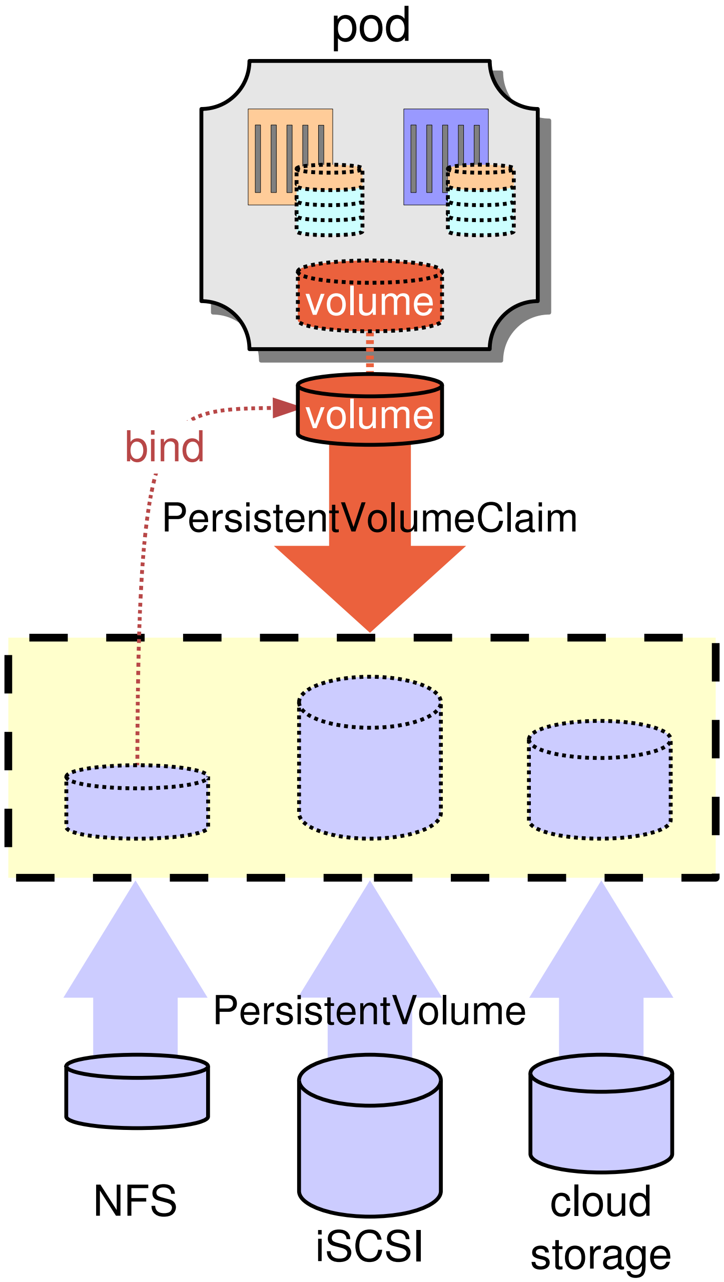
```
apiVersion: v1
kind: Pod
metadata:
  name: apahost
  labels:
    app: webserver
spec:
  nodeSelector:
    kubernetes.io/hostname: hostc
  containers:
  - name: apatest
    image: atcomp/apachetest
    ports:
    - containerPort: 80
    volumeMounts:
    - name: hosthtml
      mountPath: /var/www/html
  volumes:
  - name: hosthtml
    hostPath:
      path: /var/www/html
      type: Directory
```



Persistent Volumes – static provisioning (1)

Persistent volumes – *static allocation*

- managed by
  - **`PersistentVolume`** (PV)
    - *piece of storage* provisioned by administrator
    - example types: NFS, iSCSI
  - **`PersistentVolumeClaim`** (PVC)
    - *request for storage* by user to be mounted in pod
    - specific properties can be defined, like size, access mode, performance, ...
- binding of PVC to PV
  - 1-to-1 mapping
  - PVC state '**`Pending`**' if no suitable PV available



Persistent Volumes – static provisioning (2)

Example persistent volume

- create PV of 1GiB based on NFS
- request PVC of 500MiB

pub-pv.yml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pub-pv
spec:
  storageClassName: nfspool
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 1Gi
  nfs:
    server: nasi
    path: /nfs/Public
    readOnly: false
```

```
$ kubectl apply -f pub-pv.yml
persistentvolume/pub-pv created

$ kubectl get pv
NAME      CAPACITY  ACCESS ...   STATUS      CLAIM
pub-pv    1Gi       RWX          Available
```

pub-pvc.yml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pub-pvc
spec:
  storageClassName: nfspool
  accessModes:
  - ReadWriteMany
  resources:
     requests:
       storage: 500Mi
```

```
$ kubectl apply -f pub-pvc.yml
persistentvolumeclaim/pub-pvc created

$ kubectl get pvc
NAME      STATUS   VOLUME    CAPACITY    ACCESS...
pub-pvc   Bound    pub-pv    1Gi         RWX


$ kubectl get pv
NAME      CAPACITY ACCESS... STATUS    CLAIM
pub-pv    1Gi      RWX       Bound     default/pub-pvc
```

Persistent Volumes – static provisioning (3)

Example persistent volume – cont'd

- create deployment with pod using PVC as volume
- pod pending as long as PVC pending

```
$ kubectl apply -f pub-deploy.yml
deployment/pubdep created

$ kubectl get pod
NAME                          READY   STATUS ...
pubdep-5874f6fbd6-qvndg       1/1     Running

$ kubectl exec -it pubdep-..-qvndg bash
root@pubdep-5874f6fbd6-qvndg:/# ls -l /public
....
drwxrwxrwx+ ...   4096 Jun 22  2018  Documents
drwxrwxrwx+ ...   4096 Feb 11  2016  Music
drwxrwxrwx+ ...   4096 Dec 17  2016  Photos
```

pub-deploy.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pubdep
  labels:
    app: pubdep
spec:
  replicas: 1
  selector:
    matchLabels:
      app: pubpod
  template:
    metadata:
      labels:
        app: pubpod
    spec:
      containers:
      - name: sleepcont
        image: ubuntu
        command: ["sleep", "3600"]
        volumeMounts:
        - name: pubstore
          mountPath: /public
      volumes:
      - name: pubstore
        persistentVolumeClaim:
          claimName: pub-pvc
```

Persistent Volumes – dynamic provisioning

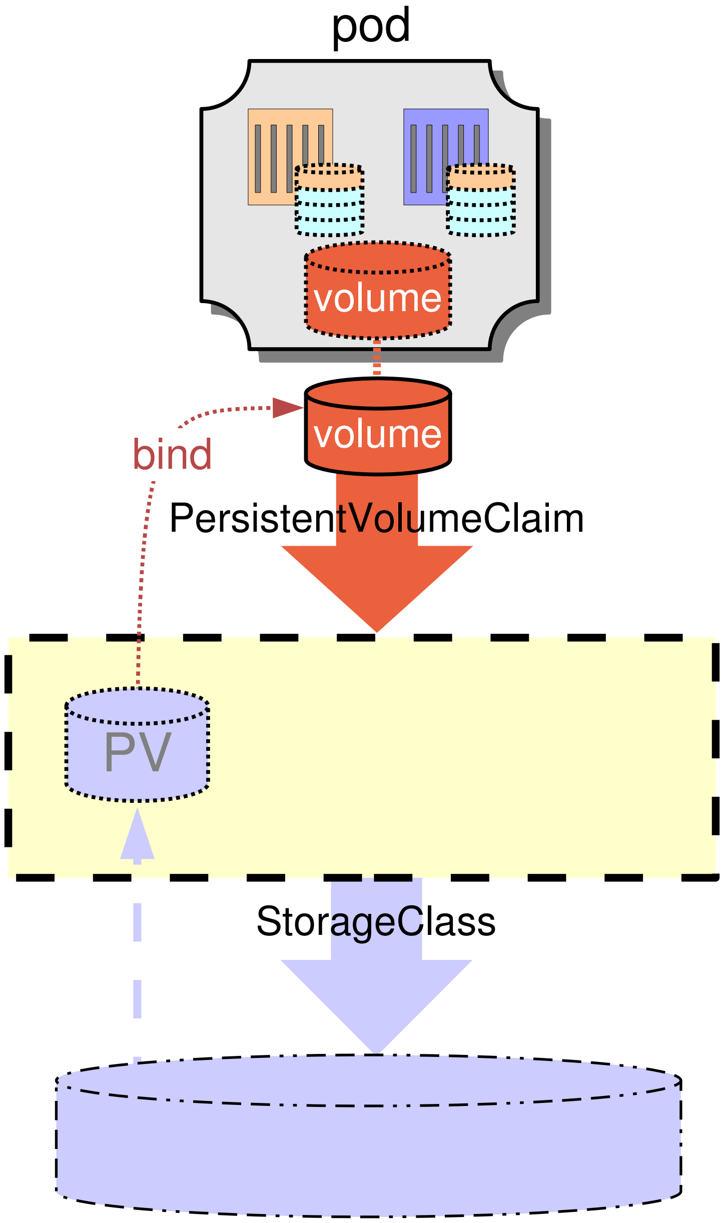
Persistent volumes – *dynamic provisioning*

- managed by **`StorageClass`** (SC)
  - dynamically allocates storage when claim (PVC) issued
  - uses *provisioner* (volume plugin) for allocation

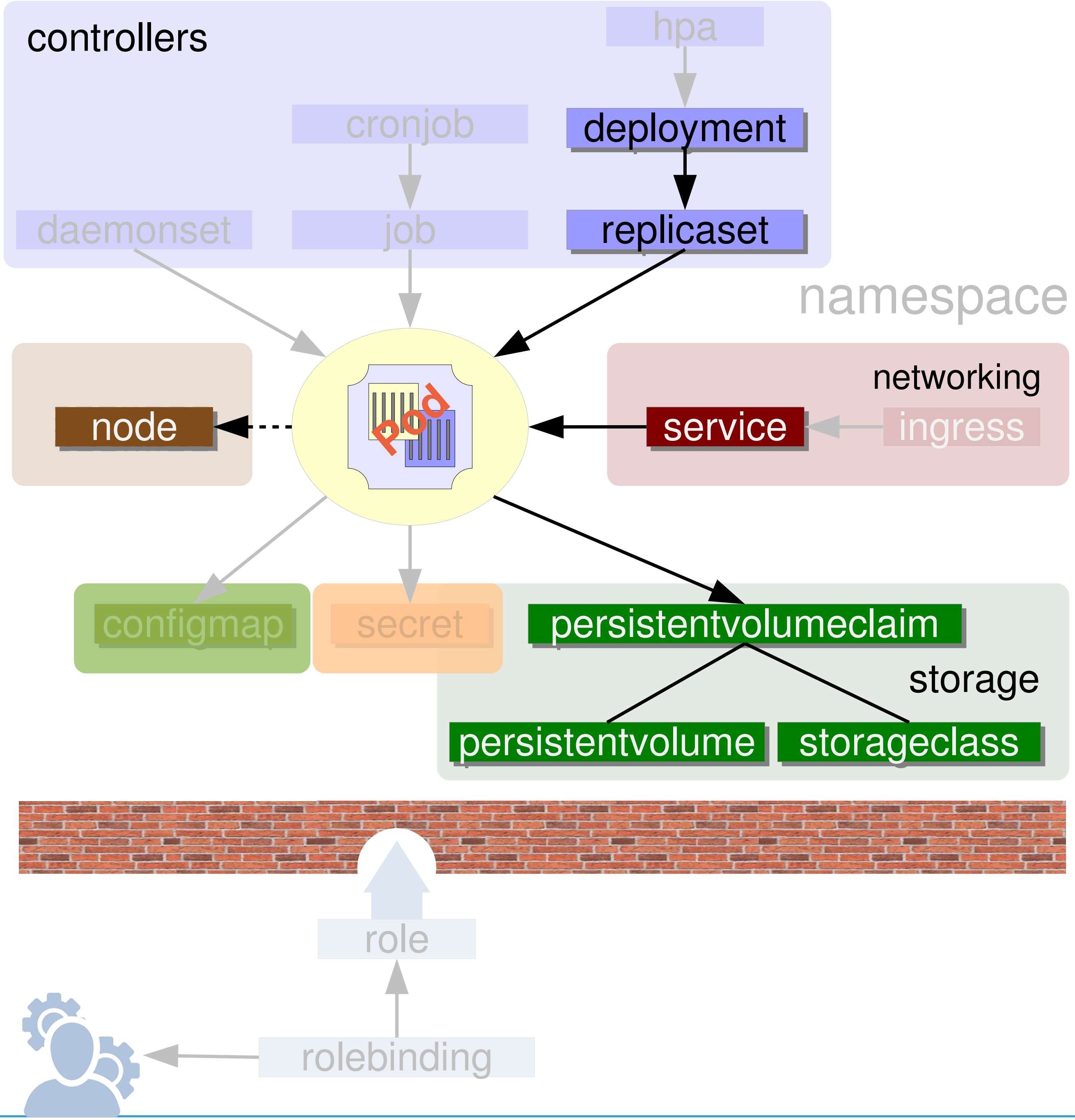
    builtin provisioners (**`kubernetes.io`**):

`gce-pd`	(GCEPersistentDisk)
`aws-ebs`	(AWSElasticBlockStore)
`azure-disk`	(AzureDisk)
`azure-file`	(AzureFile)
`glusterfs`	(Glusterfs)

    and many others....
  - dynamically creates PV object
  - volume initially empty



Objects in this workshop



Workshop “Kubernetes”

Workshop is extraction from the course

“Kubernetes Fundamentals”

(three days)

Questions?