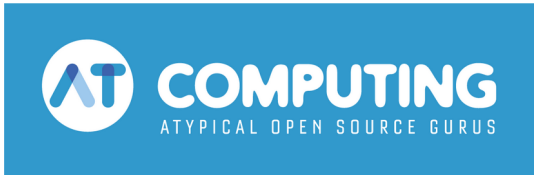
Kubernetes 101 – an introduction

Gerlof Langeveld

www.atcomputing.nl



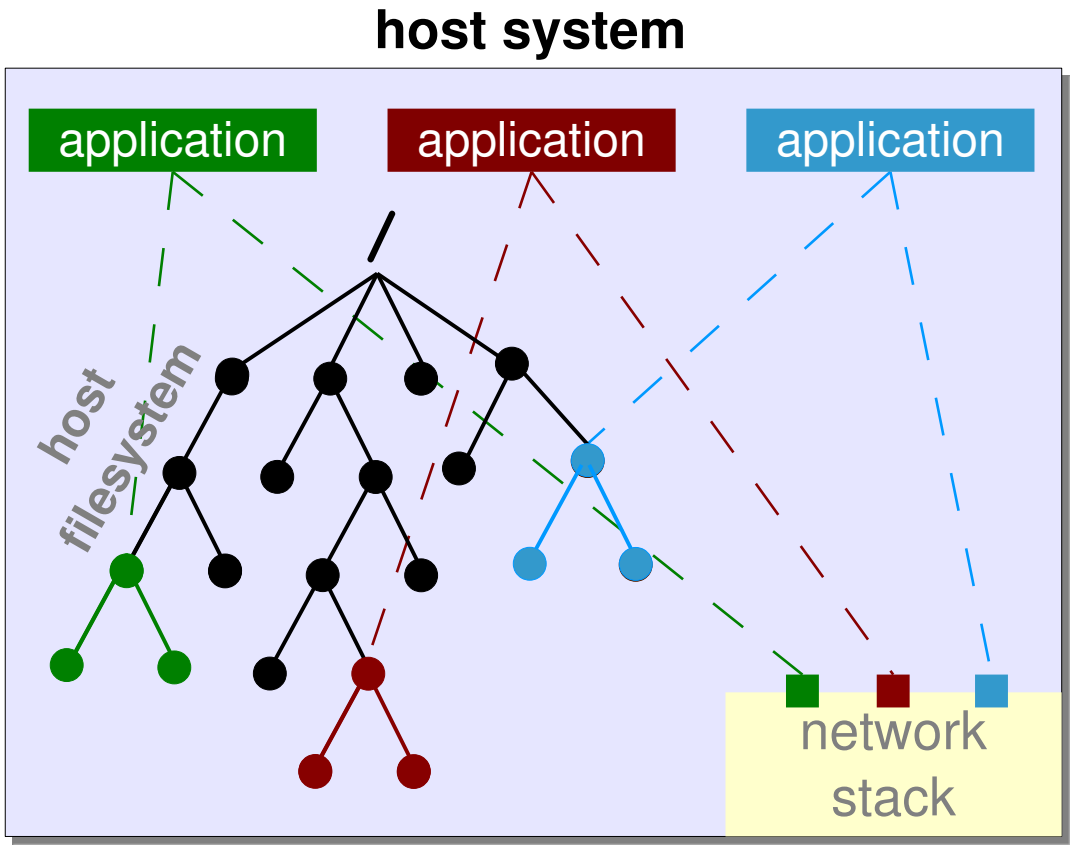

```
git clone https://github.com/atcomputing/k8soverview
```

Container intro – conventional ecosystem

Conventional production environment

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

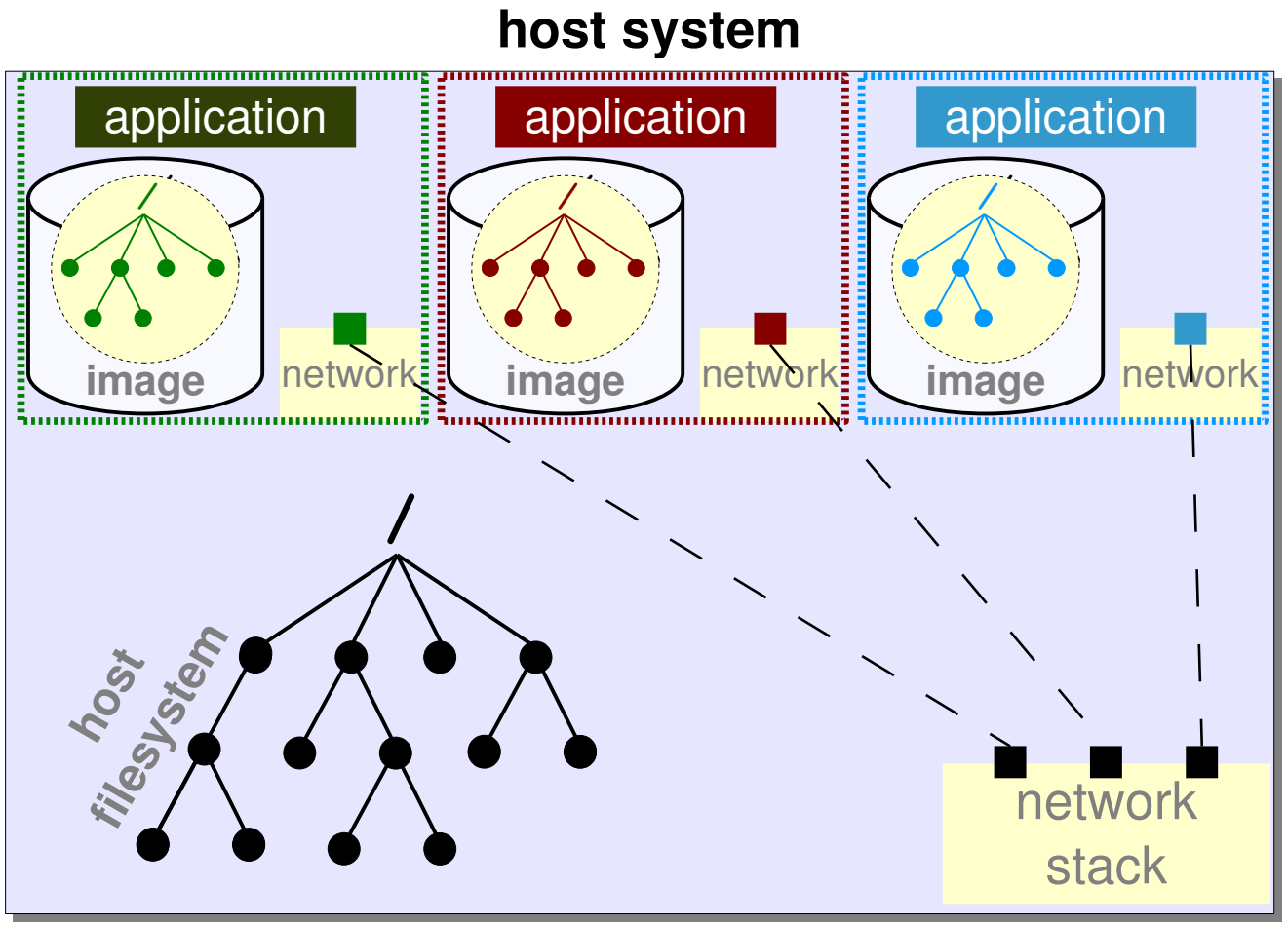
- disadvantages
  - laborius (de)installation of application
  - compromised application might
    - access all data
    - manipulate entire network stack
    - compromise other applications
  - application might overload resources



Container intro – isolated ecosystem

Container: isolated ecosystem for application

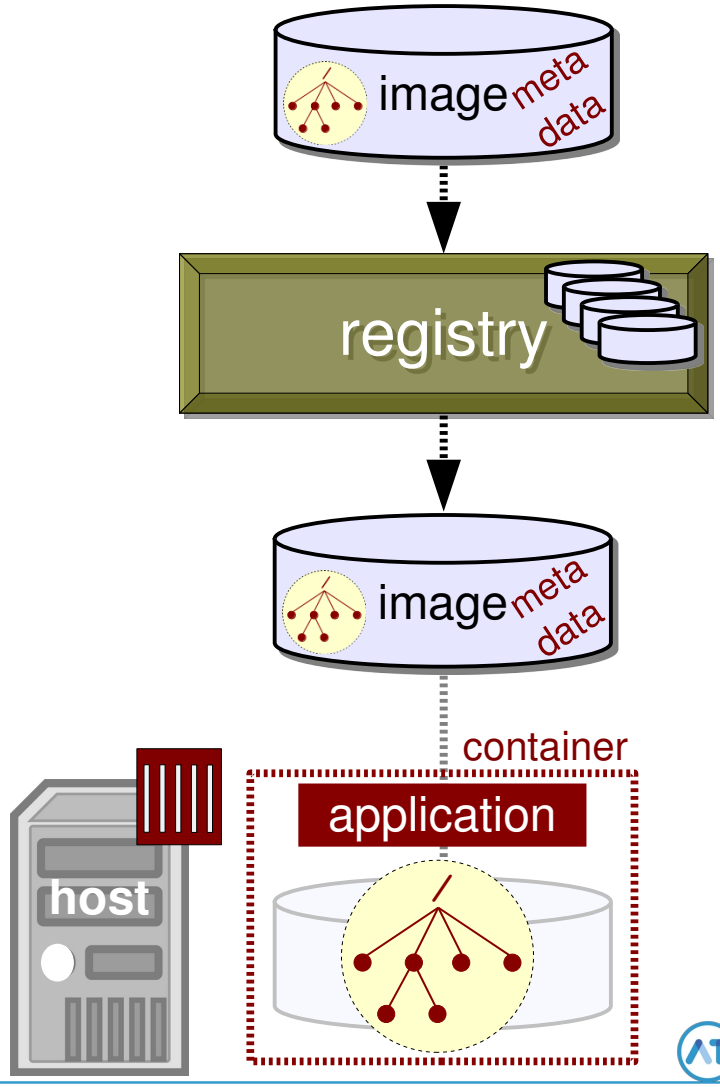
- own mini filesystem containing
  - program executables
  - libraries
  - configuration files
  - data files
  - ...
- own network
- own PID numbers
- own user identities
- cpu/memory/disk guarantees and restrictions



Container intro – container image

Image

- needed to start container on destination host
- contains
  - mini filesystem
  - metadata, e.g. application to be started
- images stored in *registry*
- *developer* (‘dev’)
  - builds image for application
  - pushes image to registry
- *operations* (‘ops’)
  - pulls image from registry
  - uses image to activate container



Container intro – build custom image

Build custom image Docker example

- specify base image and 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    204MB
ubuntu              18.04    5d2df19066ac   3 weeks ago          63.1MB
```

Container intro – run from custom image

Use custom image to run container Docker example

- run custom container

publish port *detached*: run in background

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

- contact webserver via URL **`http://localhost:8080`**
  by web browser or command **`curl`**

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

- push image to registry

```
$ docker push atcomp/apachetest
```

Kubernetes
basics

What is Kubernetes?

Kubernetes (‘helmsman’) aka. K8s

- combines various hosts into *cluster*
  - scalability
  - reliability (failover)

- orchestrates containers
  - activate
  - monitor
  - terminate

- uses container runtime implementation,
  like *containerd*, *cri-o*, ...

Kubernetes orchestration

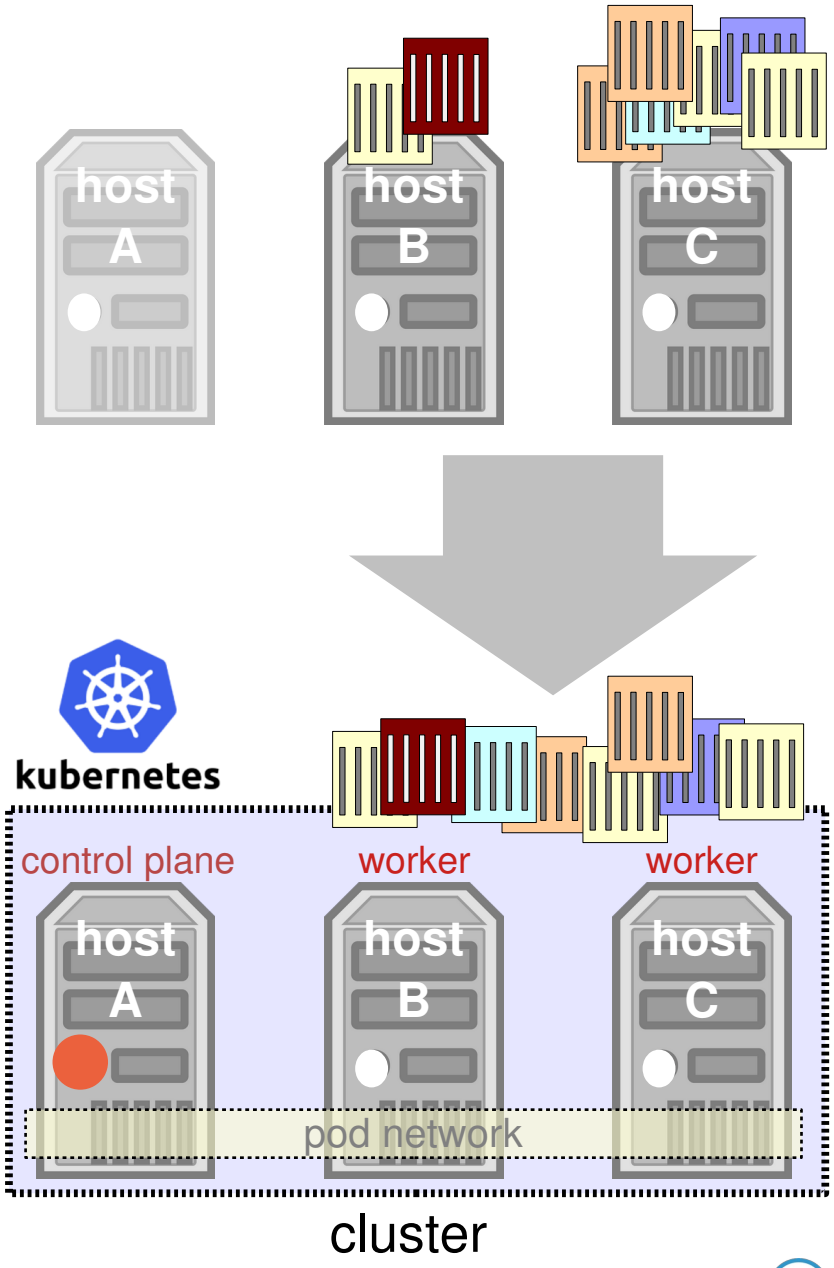
Kubernetes orchestration

- introduced in 2015,
  inspired by Google's Borg

- maintained by
  Cloud Native Computing Foundation (CNCF)

- concept
  - cluster needs at least one *control-plane node,*
    formerly known as *master node*

  - other hosts in cluster are *worker nodes*
    that only run container instances



User interface

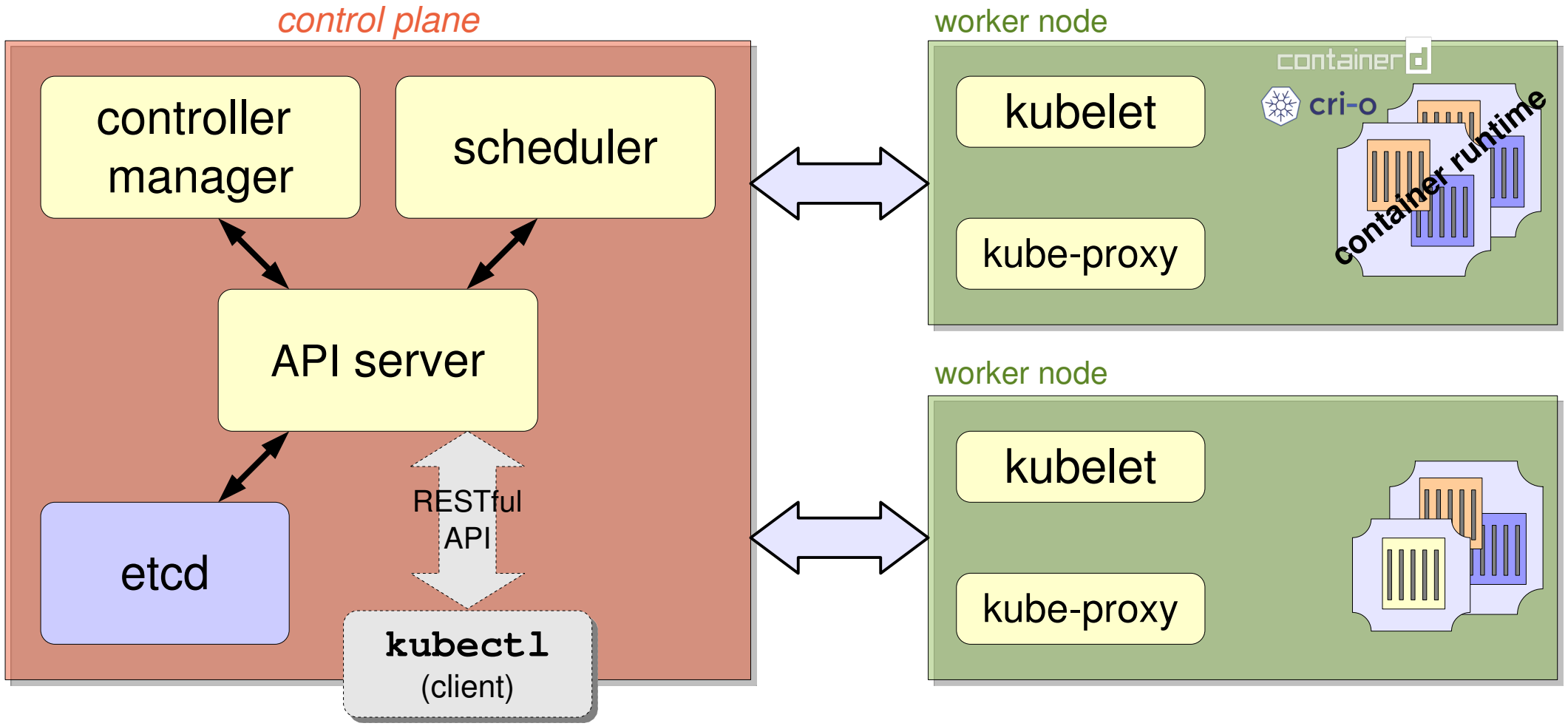
User interfaces

- 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`**, **`scale`**, ...

```
$ kubectl get nodes
NAME      STATUS   ROLES           AGE   VERSION
hosta     Ready    control-plane   75m   v1.26.0
hostb     Ready    <none>          40m   v1.26.0
hostc     Ready    <none>          40m   v1.26.0
```

- graphical user interface
  - also valid for cloud implementations, like
    - Google Kubernetes Engine (GKE)
    - Amazon Elastic Kubernetes Service (EKS)
    - Azure Kubernetes Service (AKS)

Kubernetes architecture

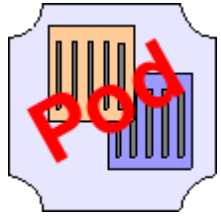


- *API server*: controls entire cluster
- *scheduler*: schedules containers (via pods) on nodes
- *controller manager*: controls required number of replicas
- *etcd*: distributed key-value store to maintain current cluster state

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

Pods – introduction

Pod



- conceptually smallest unit, wrapping one or more containers
  - typically one container running primary application, with other supporting containers
  - containers in pod share same IP address on *internal* pod network
- run pod manually

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

- or run pod via *manifest* file (YAML syntax)

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

- status:

```
$ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
apatest   1/1     Running   0          8s

$ kubectl get pods -o wide
    ...   AGE   IP            NODE     ....
          8s    10.244.1.5    hostb

$ curl 10.244.1.5
<h1> Message from container! </h1>
```

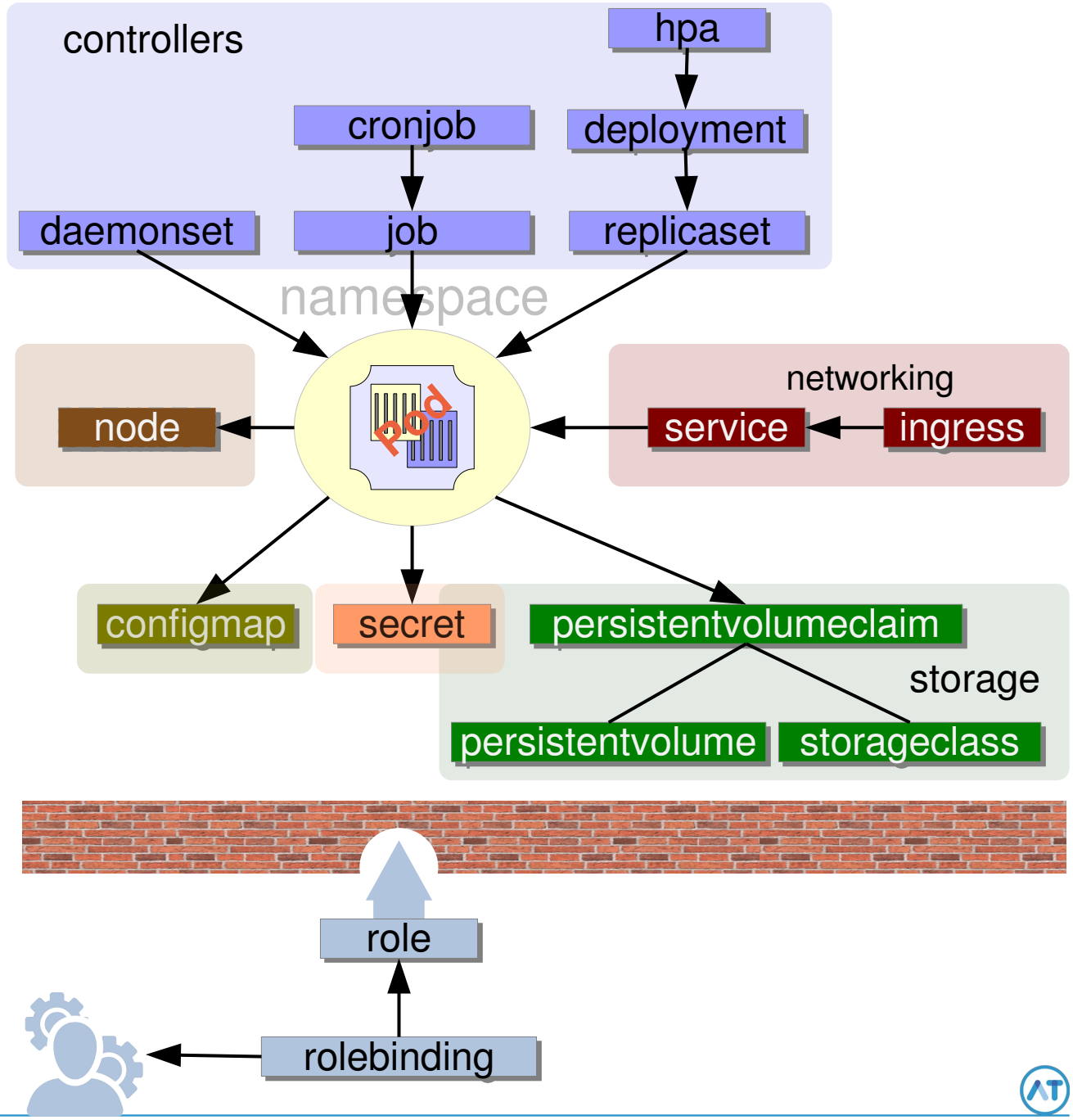
apa.yml

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

Kubernetes objects

Kubernetes concept

- numerous object types, like *pod*, *node*, *service*, *deployment*, ...
- 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)



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
```

  - references between objects
    e.g. to assign **`Service`** object to **`Pod`** objects

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 develop
namespace/develop created

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

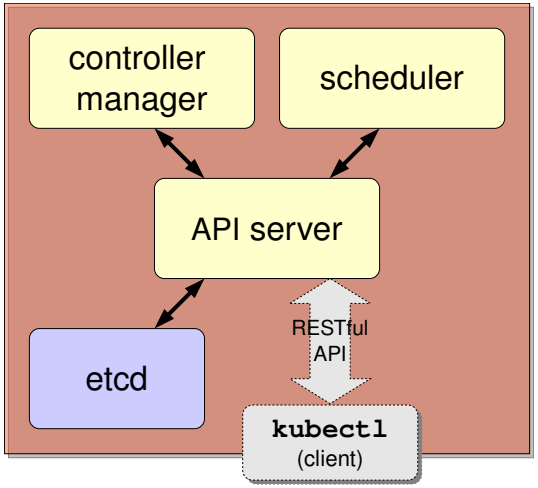
```
$ kubectl apply -f apa.yaml -n develop      in namespace develop

$ kubectl get pods -n develop               in namespace develop
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, ...)

Controllers

Controller types

Controllers



- pod ('naked pod', 'bare pod') not self-healing
  - pods usually started under supervision of *controller*
  - controllers are part of *controller manager*

- various controller types, like
  - *ReplicaSet* (rs)
    - maintains pod replicas and restarts failing pods
  - *Deployment* (deploy)

    maintains pod replicas by using ReplicaSet

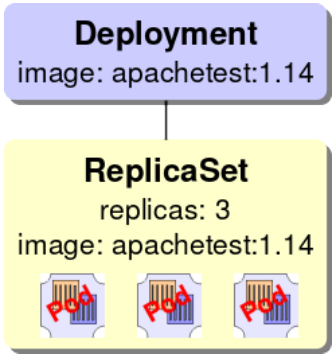
    provides rolling updates and rollbacks

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     READY  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    Runn
apadep-7b6fc56c77-hk7qp       1/1    Runn
```



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

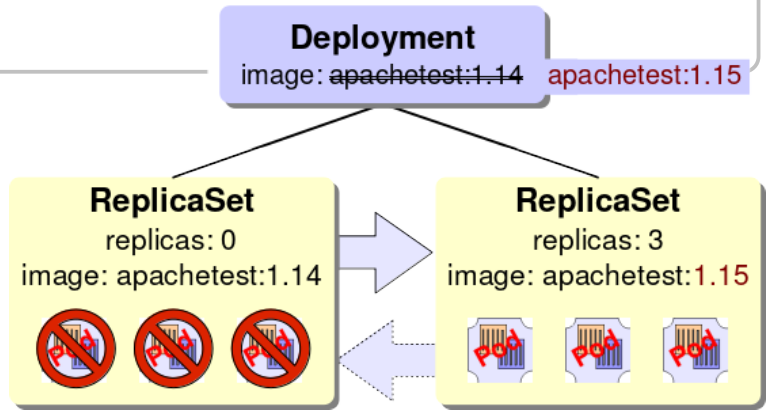
- modify deployment manifest file and apply:
  #replicas, environment vars, image version, ...

first image **apachetest:1.14** has been modified to **apachetest:1.15**

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

```
$ 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
```

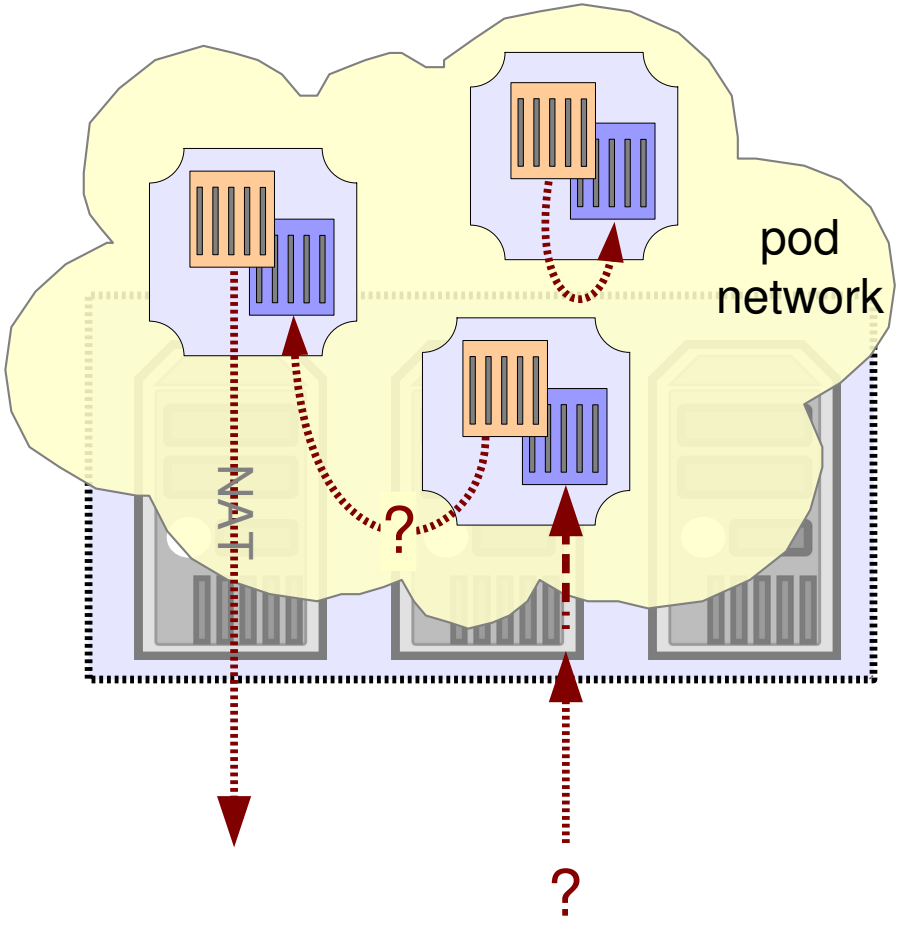
Networking

Kubernetes networking

Communication possibilities

- every pod gets unique IP address on pod network, dynamically assigned

- communication possibilities
  - container-to-container in same pod
    - via **`localhost`** (loopback interface)
  - 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?



Networking – 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 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

```yaml
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:1.15
        ports:
        - containerPort: 80
```

Networking – services

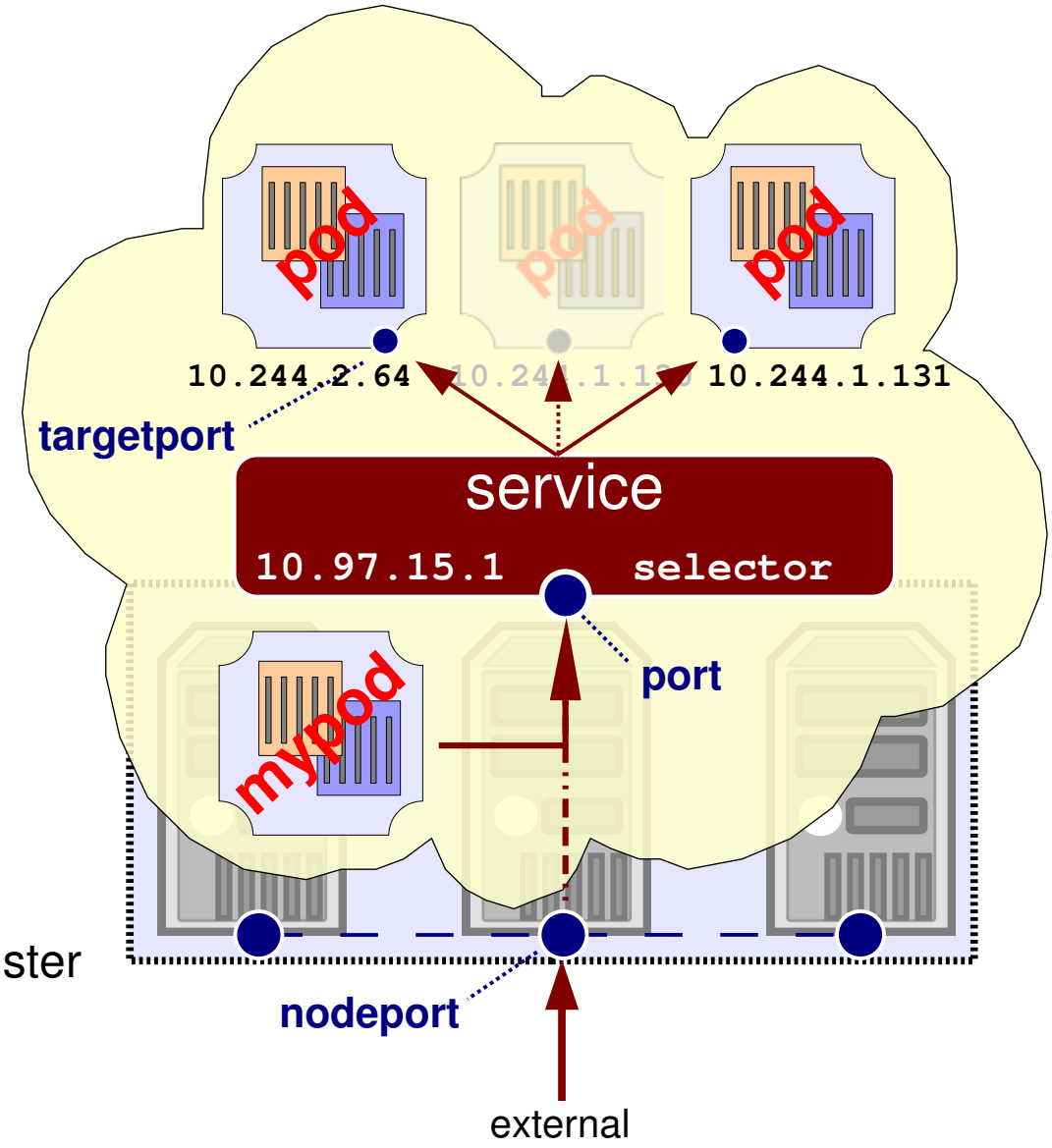
Service object

- separate object
- gets *static* virtual IP address, though dynamically assigned
- attaches pods by using selector referring to pod label
- accessibility determined by *type*

  `ClusterIP`: pod-to-pod communication

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

- load balancing to attached pods



Networking – setup `ClusterIP` service

Example: add *internal* service for webserver

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

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

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

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

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

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

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

apa-deploy.yml

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

apa-svc.yml

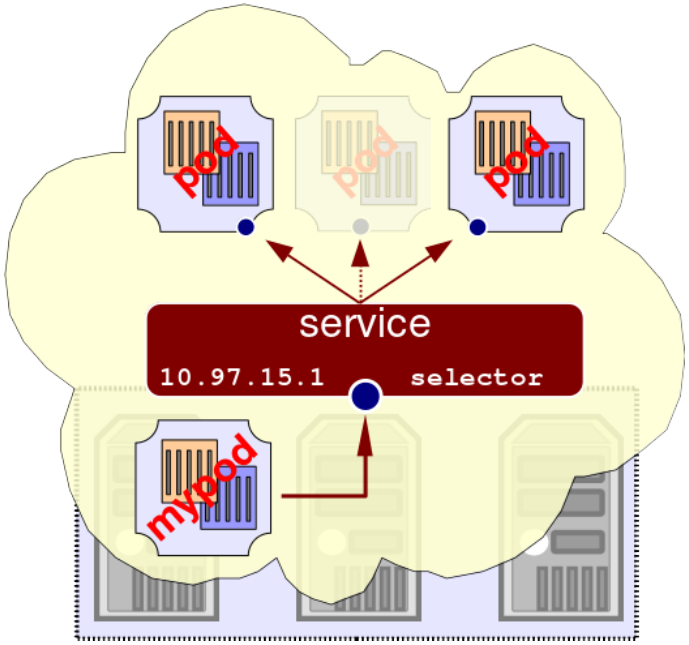
```
apiVersion: v1
kind: Service
metadata:
  name: webserv
  labels:
    app: apa
spec:
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
  selector:
    app: apapod
```

Networking – service discovery

Service discovery by other pods

- via internal DNS
- maintains record for every service:

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



- example: activate interactive pod (flags **-it**)

```
$ kubectl run mypod -it --restart=Never --image=atcomp/nwubuntu
root@mypod:/# host webserv
webserv.default.svc.cluster.local has address 10.97.15.1

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

Networking – setup **NodePort** service

Example: add *external* service for webserver

- type **NodePort** to provide external access

- creates port on every node in cluster in range 30000-32767
  - can be specified with keyword **nodePort**

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

- access from outside cluster

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

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

```
$ kubectl run mypod -it --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: apapod
    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: apapod
```

Storage

Kubernetes Volumes

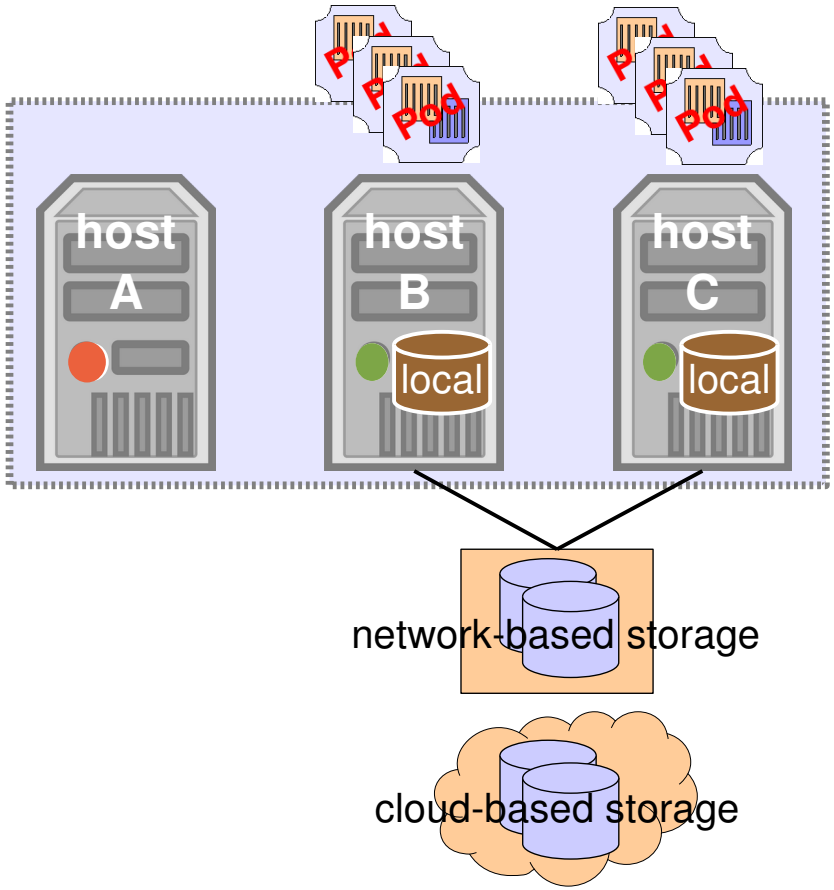
Storage

- *stateless pod* – preferred!
  - no need to preserve data
  - non-persistent storage in mini filesystem

- *stateful pod*
  - requires persistent volume

  - various volume types, like
    - local host disk
      – restricted use

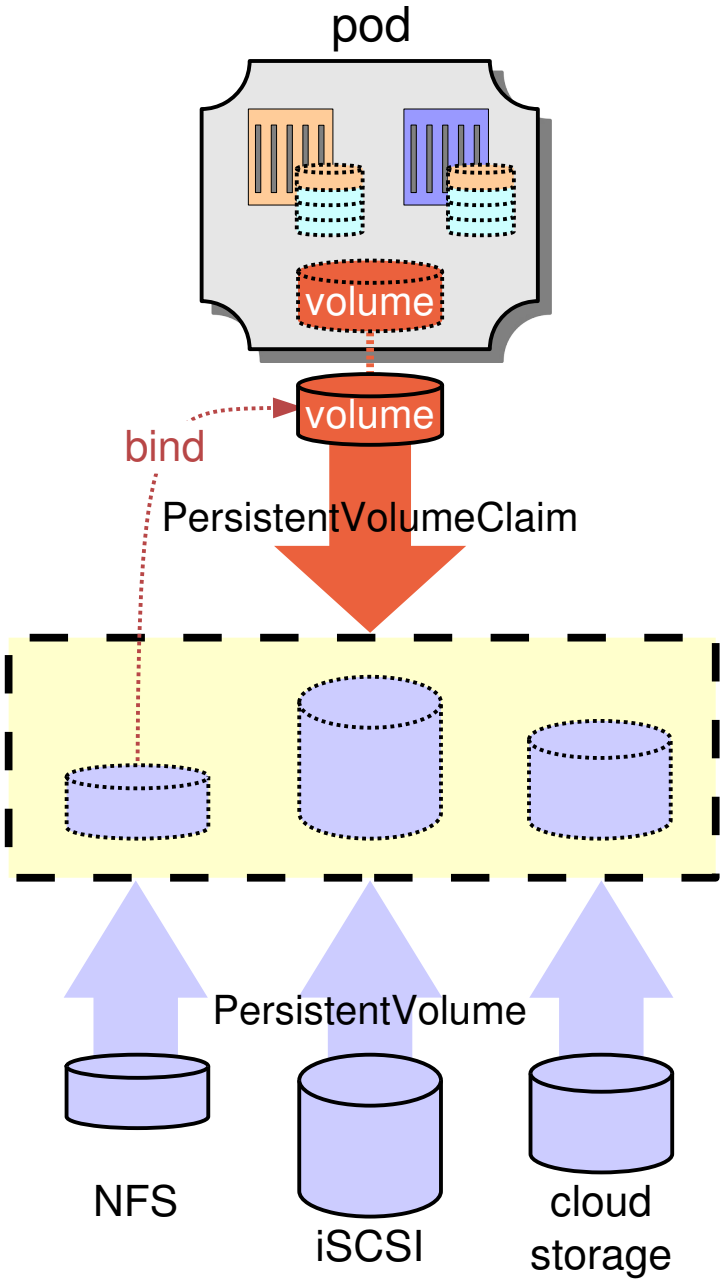
    - network storage
      – pod terminated on **`hostB`** might be restarted on **`hostC`**
      and/or
      – pods running on different hosts might share same storage



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*
      to be mounted in pod
    - specific properties can be defined, like size, access mode, performance, ...
- binding of PVC to PV is 1-to-1 mapping



Persistent Volumes – static provisioning (2)

Example persistent volume

- create PV of 1GiB based on NFS

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

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

pub-pv.yml

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

- request PVC of 500MiB

```
$ 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
```

pub-pvc.yml

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

Persistent Volumes – static provisioning (3)

Example persistent volume – cont'd

- create pod using PVC as volume

example:

```
$ kubectl apply -f pub-pod.yml
pod/testpod created

$ kubectl get pod
NAME      READY   STATUS ...
testpod   0/1     Completed

$ kubectl logs testpod
....
drwxrwxrwx+ ...  4096 Jun 22  2021  Documents
drwxrwxrwx+ ...  4096 Feb 11  2020  Music
drwxrwxrwx+ ...  4096 Dec 17  2020  Photos
```

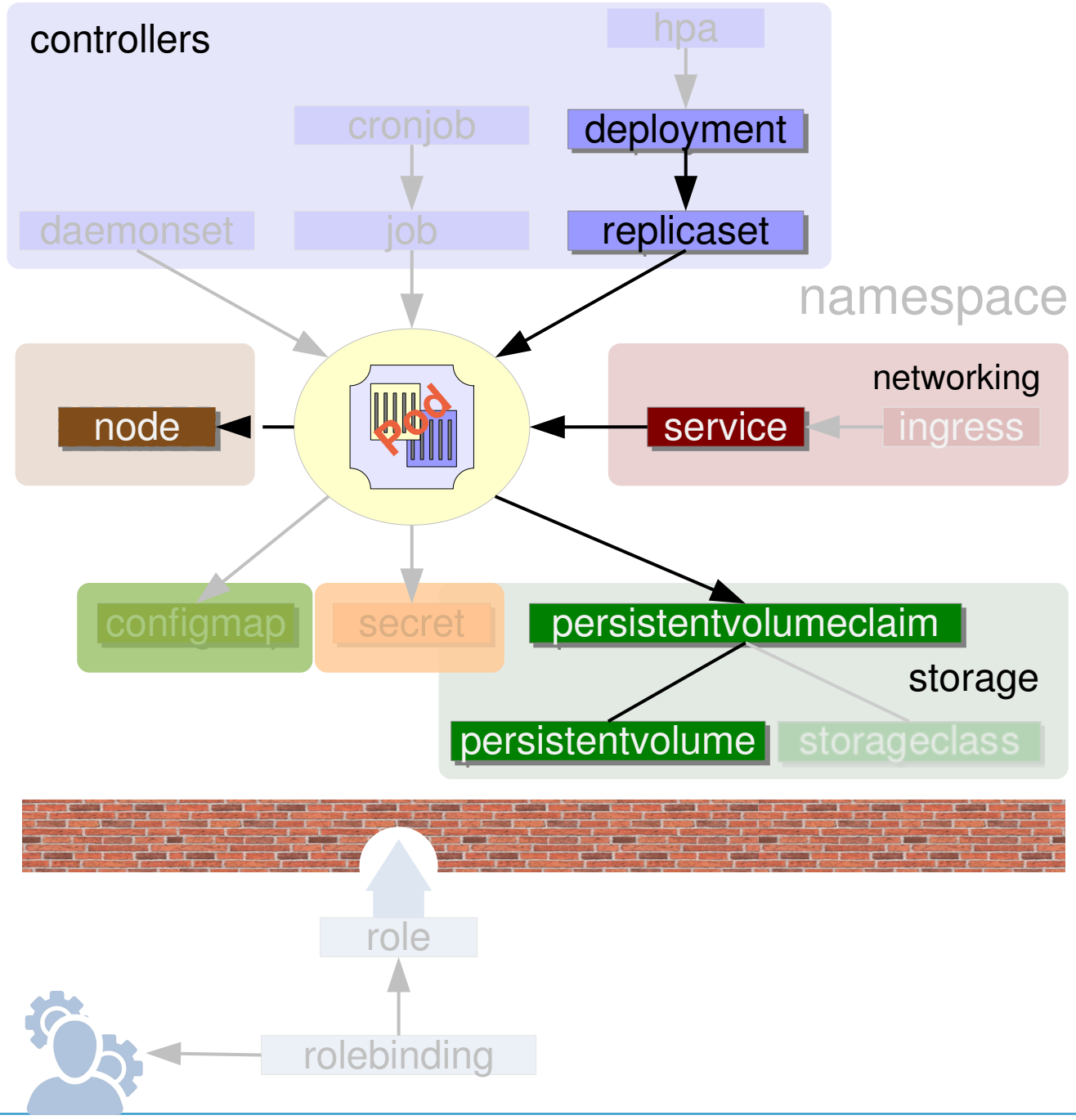
output of `ls -l /public`

pub-pod.yml

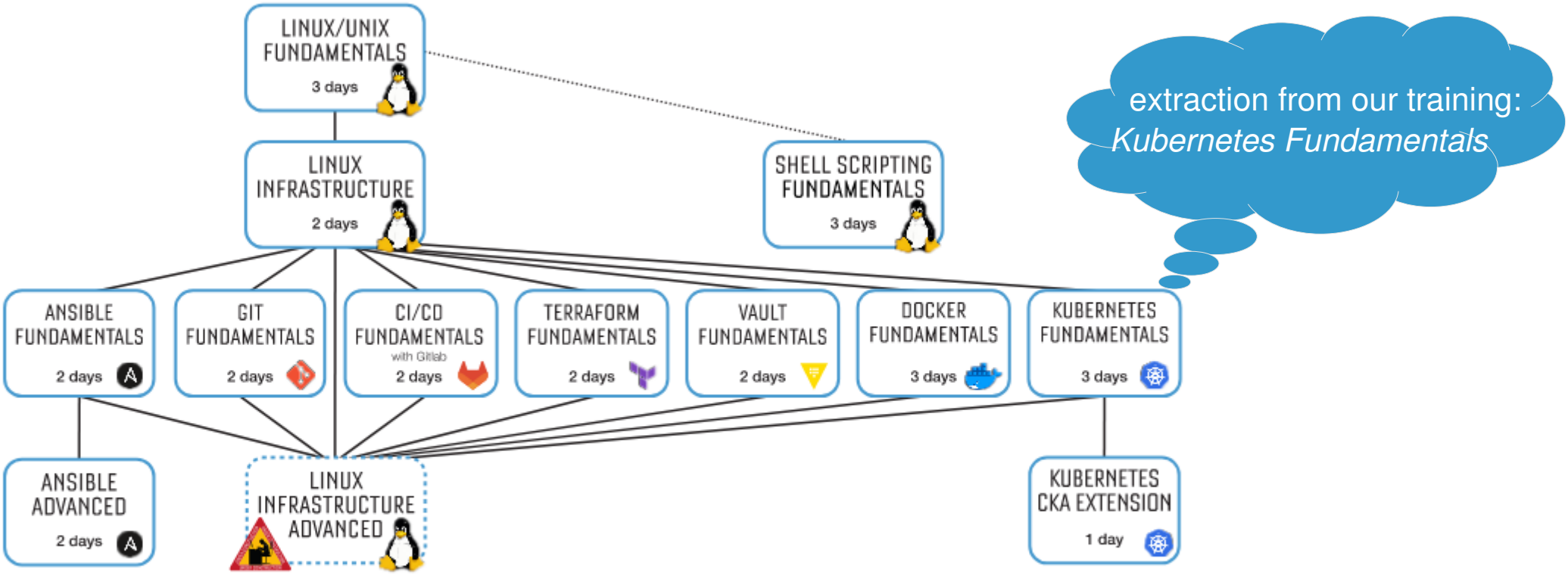
```
apiVersion: v1
kind: Pod
metadata:
  name: pub-pod
spec:
  containers:
  - name: lspub
    image: ubuntu
    command: ["ls", "-l", "/public"]
    volumeMounts:
    - name: pubstore
      mountPath: /public
  restartPolicy: Never

  volumes:
  - name: pubstore
    persistentVolumeClaim:
      claimName: pub-pvc
```

- also possible: *dynamic provisioning*
  - allocate PV dynamically via storage class when creating PVC

Objects used



Kubernetes 101 – an introduction



Questions?

```
git clone https://github.com/atcomputing/k8soverview
```