OpenMunich

December 1, 2017

Data & State in a Container World

Andreas Neeb

Solution Architect, Red Hat GmbH

redhat

ABOUT ANEEB

Andreas Neeb,

Chief Architect Financial Services

Red Hat GmbH, aneeb@redhat.com

- Strategist
- Container Guy
- Father & Husband
- Nerd
- Football Fan

Not necessarily always in that order

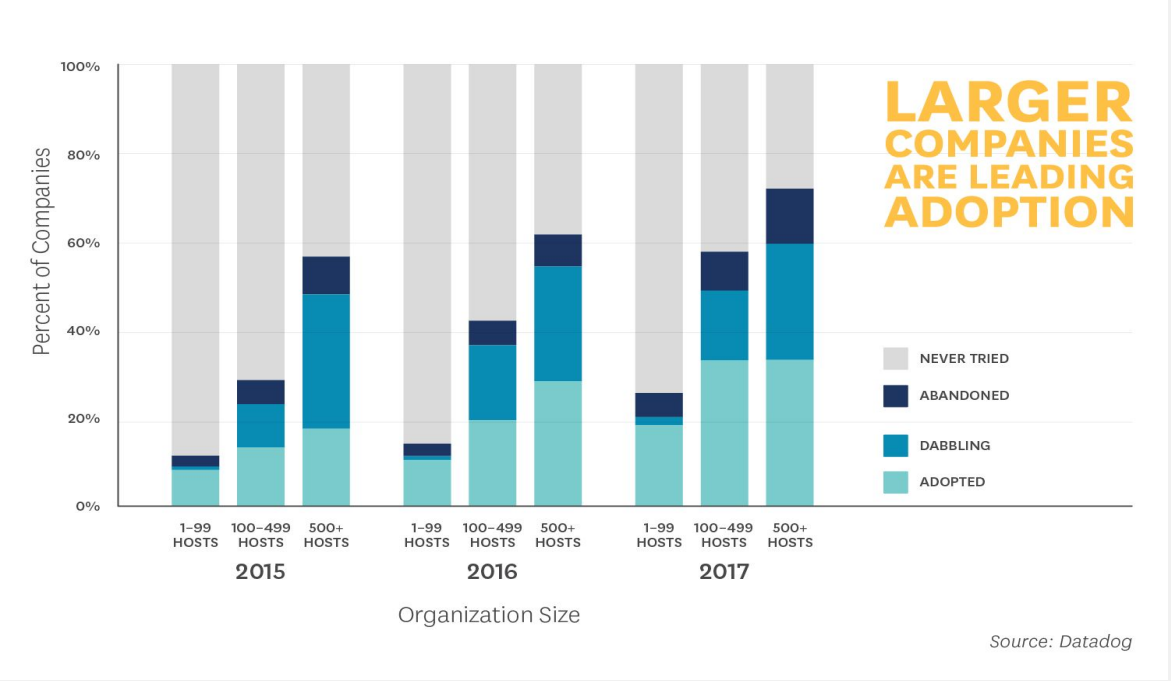
AGENDA

- Intro
- Problem Statement
- Solution Blocks
  - Docker Volumes
  - Kubernetes Persistent Volumes
  - Container Native Storage
- Summary

INTRO

CONTAINER ADOPTION RISING

Container Adoption Status by Infrastructure Size



BUT THERE ARE CHALLENGES

In order to deploy containers, which challenge has been the most difficult to overcome?

Answer Choices	Responses	
Data management	**13.48%**	31
Disaster recovery	**5.65%**	13
Graphical UI	**3.04%**	7
Logging	**4.78%**	11
Multi-cloud or cross-datacenter support	**10.43%**	24
Networking	**9.57%**	22
Persistent storage	**26.09%**	60
Reliability	**6.52%**	15
Scalability	**6.96%**	16
Security	**6.96%**	16
Other (please specify)	**6.52%**	15
Total		**230**

CONTAINER & STATE

What are container images?

Container images are runnable packages that contain your applications and their dependencies. They are lighter than virtual machine images and can be layered with other Container images to re-use common content.

Isolated

Applications run in containers with isolated memory, file-system, and networking resources for maximum stability and security.

Lightweight

Containers include only the minimal runtime requirements for the application, reducing size and simplifying maintenance.

Portable

Move applications and all of their runtime requirements across systems.

What are container images?

Container images are runnable packages that contain your applications and their dependencies. They are lighter than virtual machine images and can be layered with other Container images to re-use common content.

Isolated

Applications run in containers with isolated memory, file-system, and networking resources for maximum stability and security.

Lightweight

Containers include only the minimal runtime requirements for the application, reducing size and simplifying maintenance.

Portable

Move applications and all of their runtime requirements across systems.

BLAH

What are container images? Take 2

What are container images? Take 2

It is a tarball, really!

What are container images? Take 3

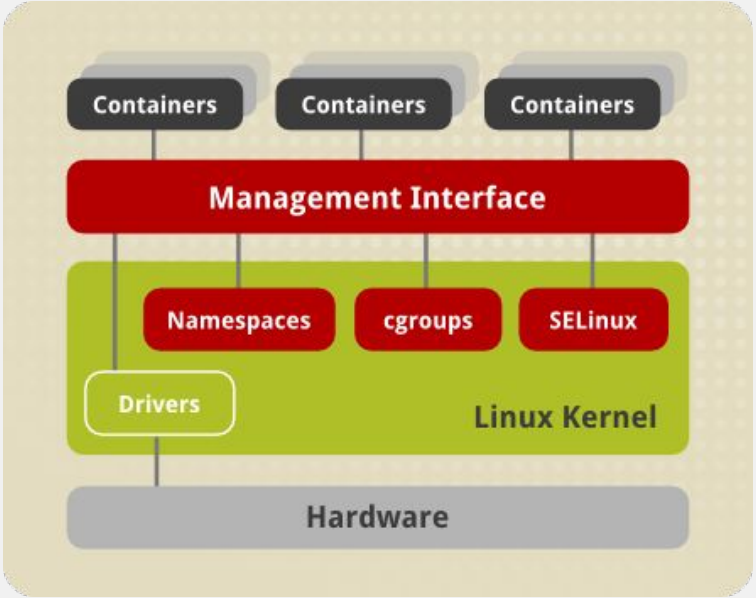
What are container images? Take 2

Well, actually it is a tarball of tarballs.

What are container images? Take 4

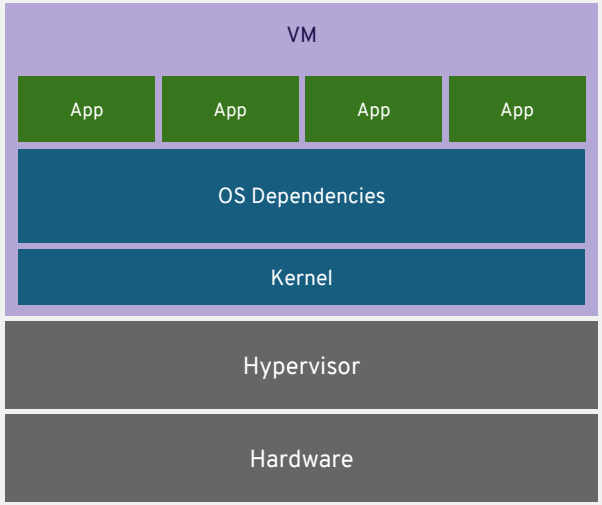
What are container images? Take 2

Well, actually it is a tarball of tarballs boosted by some kernel magic.



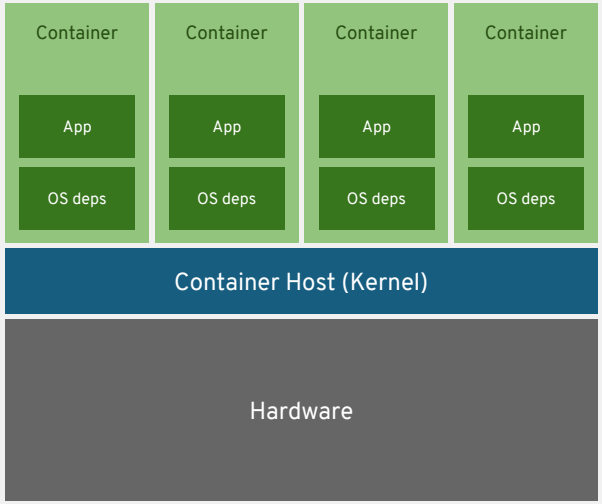
VIRTUAL MACHINES VS. CONTAINERS

VIRTUAL MACHINES



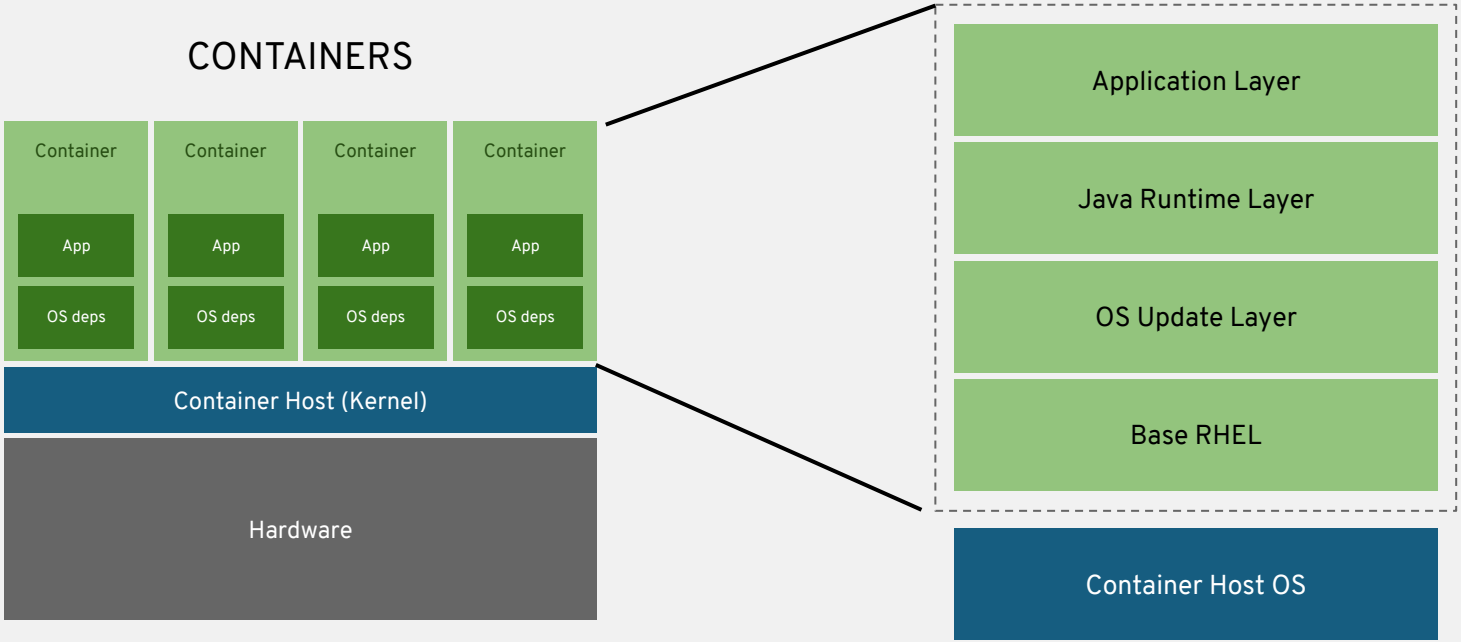
virtual machines are isolated
apps are not

CONTAINERS

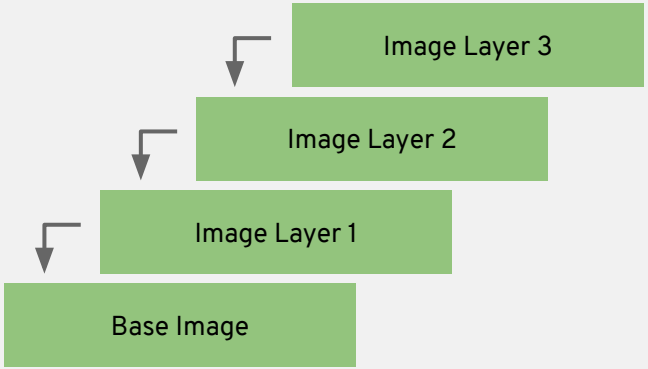


containers are isolated
so are the apps

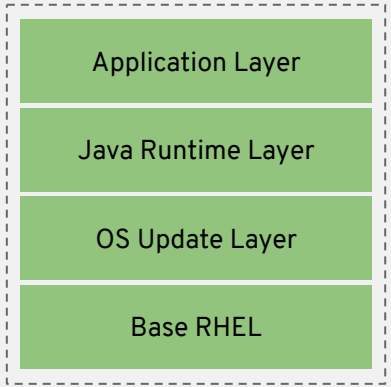
CONTAINER IMAGE LAYERING



CONTAINER IMAGE LAYERING

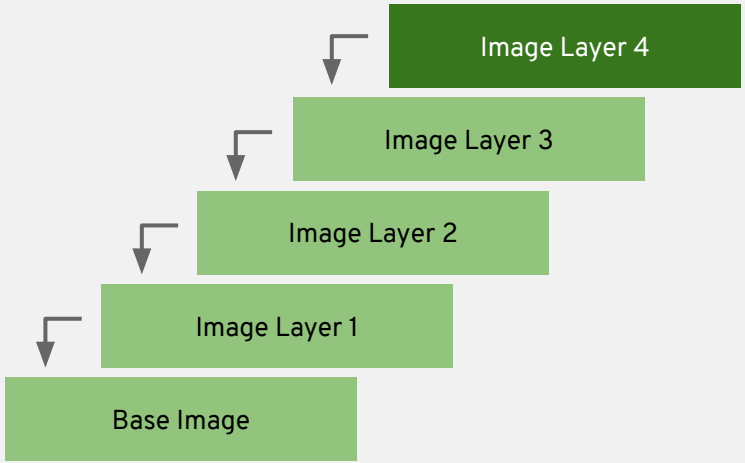


Container Image Layers

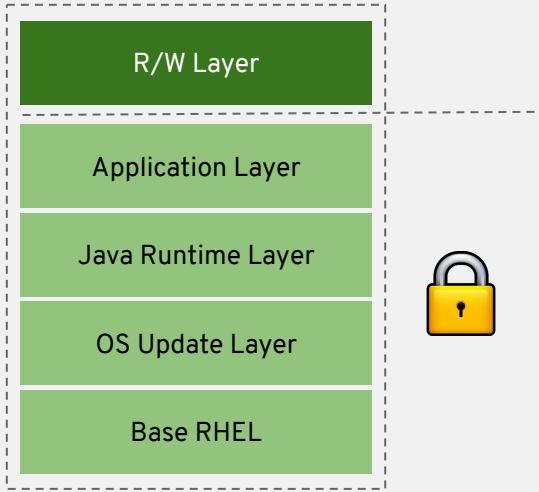


Example Container Image

CONTAINER IMAGE LAYERING



Container Image Layers



Example Container Image

Demo - plain Container w/o Volume

```
docker build -t my-httpd --build-arg IMAGE_VERSION=1.0 .
docker run --name httpd -d -p 8080:8080 my-httpd
docker exec httpd cat image-version
docker cp html/index.html httpd:/var/www/html/

docker build -t my-httpd --build-arg IMAGE_VERSION=2.0 .
docker stop httpd && docker rm httpd
docker run --name httpd -d -p 8080:8080 my-httpd
docker exec httpd cat image-version
```

Demo - plain Container w/o Volume

```
docker build -t my-httpd --build-arg IMAGE_VERSION=1.0 .
docker run --name httpd -d -p 8080:8080 my-httpd
docker exec httpd cat image-version
docker cp html/index.html httpd:/var/www/html/

docker build -t my-httpd --build-arg IMAGE_VERSION=2.0 .
docker stop httpd && docker rm httpd
docker run --name httpd -d -p 8080:8080 my-httpd
docker exec httpd cat image-version
```

BORING

Summary

- Container are stateless / volatile per se and therefore - on its own - not suited for stateful (aka any!) application
- This is by design and enables “immutable infrastructure”
- Changes to a Container require a full rebuild and deployment which in turn result of a loss of temporary r/w layer

DOCKER VOLUMES

DOCKER VOLUMES

Volumes are directories (or files) that are outside of the default Union File System and exist as normal directories and files on the host filesystem.

- Changes to volumes are applied directly to the host filesystem
- Volumes can be shared across containers
- Volumes have a separate lifecycle
  - Volumes survive a container / image delete
  - Update to images don’t affect volumes

Demo - Container with Volume

```
docker volume create --name www
docker stop httpd && docker rm httpd
docker run --name httpd -d -p 8080:8080 -v www:/var/www my-httpd
docker exec httpd cat image-version
docker cp html/index.html httpd:/var/www/html/

docker build -t my-httpd --build-arg IMAGE_VERSION=3.0 .
docker stop httpd && docker rm httpd
docker run --name httpd -d -p 8080:8080 -v www:/var/www my-httpd
docker exec httpd cat image-version
```

DOCKER VOLUMES

Data now lives outside the container but on the host

- Not directly portable, Container can’t fail over
- Data lifecycle tied to host

Possible solutions

- Manual copy
- Synch data (nfs/glusterfs/rsync)
- Shared Storage Mountpoint

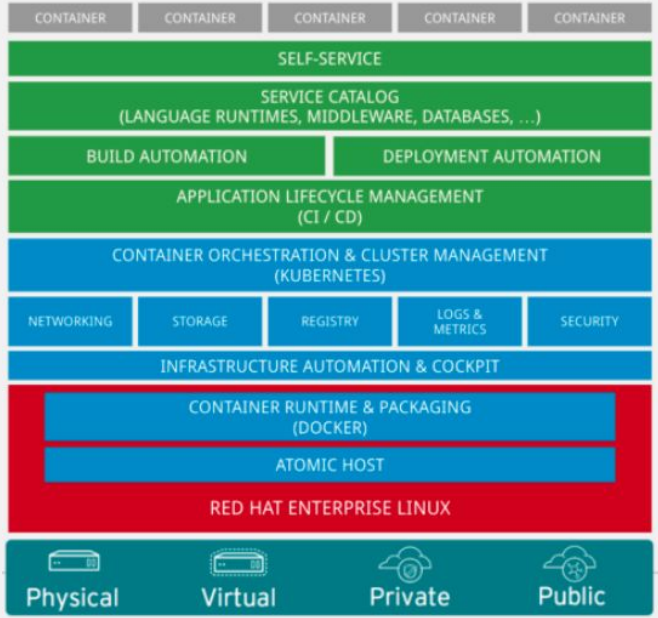
KUBERNETES PERSISTENT VOLUMES

KUBERNETES / OPENSHIFT

OpenShift is Red Hat's Enterprise Kubernetes Distribution

Solves all kind of nasty problems when managing container at scale, ie.

- Orchestration
- Service Discovery
- High Availability
- Networking
- Security
- Logging & Metering
- Storage

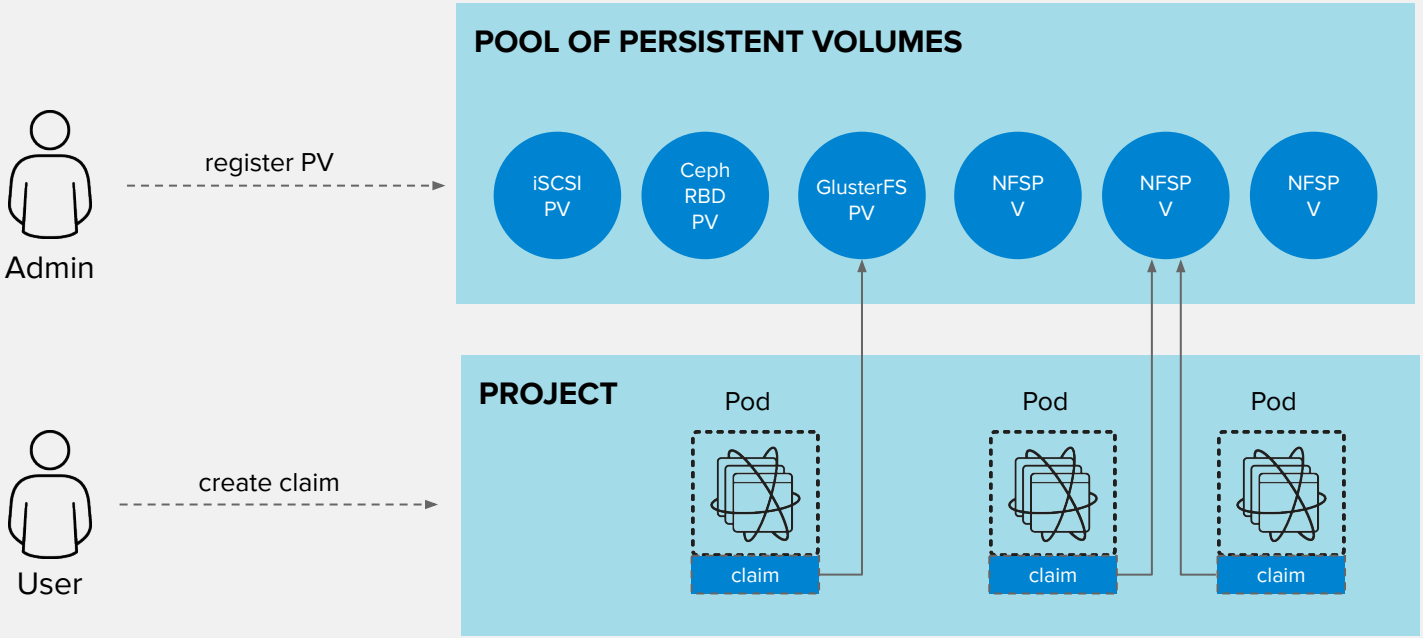


PERSISTENT STORAGE

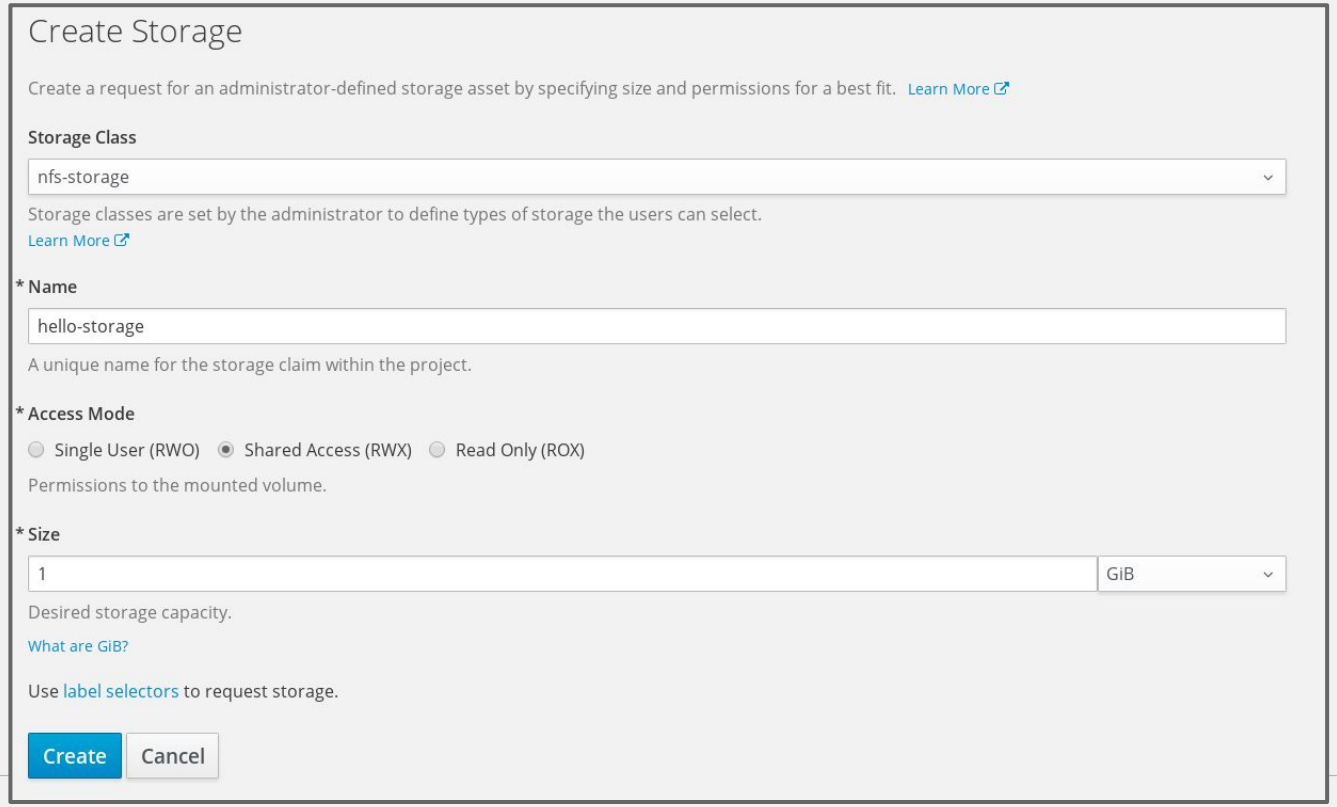
- Persistent Volume (PV) is tied to a piece of network storage
- Provisioned by an administrator (static or dynamically)
- Allows admins to describe storage and users to request storage
- Assigned to pods based on the requested size, access mode, labels and type

NFS	OpenStack Cinder	iSCSI	Azure Disk	AWS EBS	FlexVolume
GlusterFS	Ceph RBD	Fiber Channel	Azure File	GCE Persistent Disk	VMWare vSphere VMDK

PERSISTENT STORAGE



Demo - Kubernetes Persistent Volumes



Additional Kubernetes Options for State

- Environment Variables
- ConfigMaps
- Secrets
- Stateful Sets (Hostname, ID, Ordering, ...)

Kubernetes Persistent Volumes

- Data persisted “outside” of Container Host.
- Abstraction of Storage Backend
- Clear separation of concerns
- Developer Self Service

- Volumes still need to be (pre) provisioned
- Data still sticky to actual storage provider

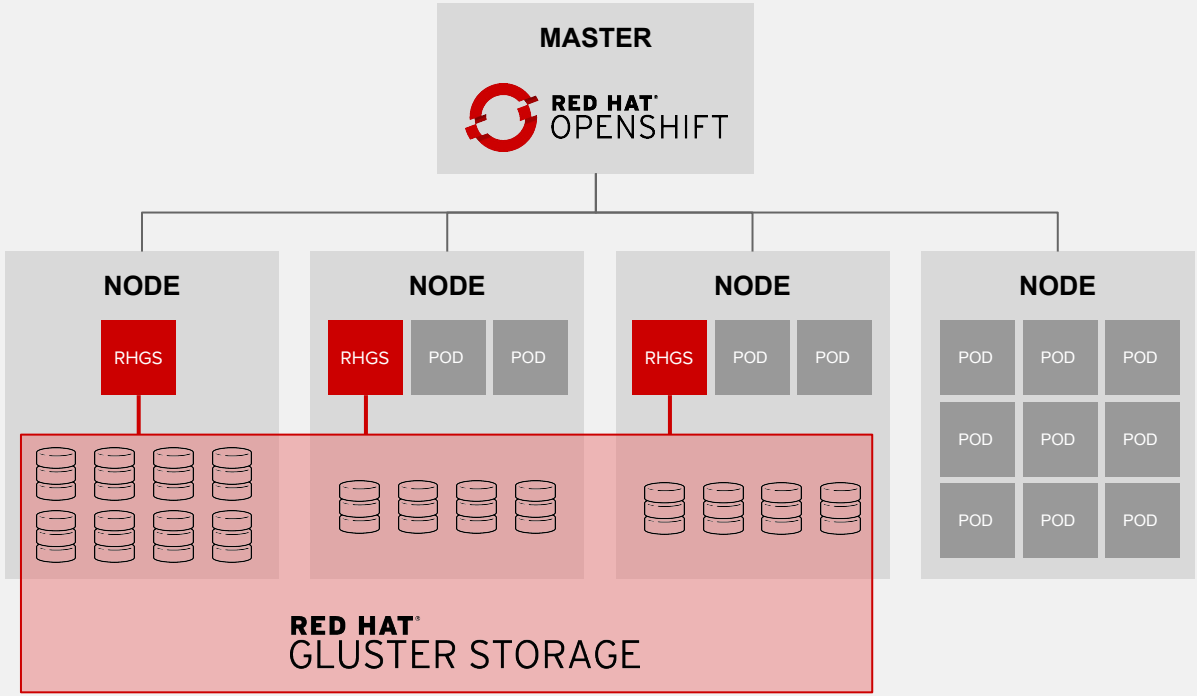
CONTAINER NATIVE STORAGE

CONTAINER NATIVE STORAGE

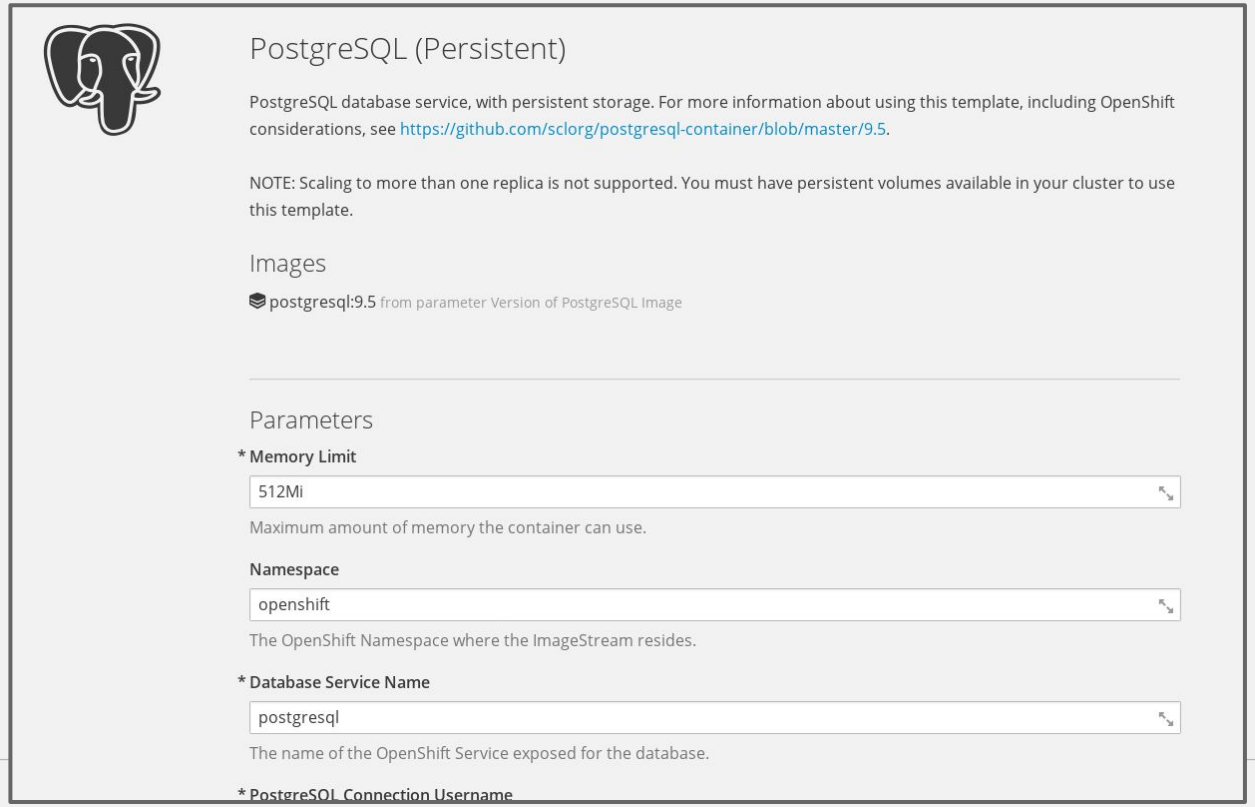
GlusterFS im Container + Heketi (restAPI)

- (Storage)-Hosts have N additional Block Devices (JBOD)
- Containerized daemon create a large raw pool
- Rest-API creates/modifies/removes volumes
- Kubernetes creates a persistent volume per claim “on the fly”

CONTAINER-NATIVE STORAGE



Demo - Container Native Storage



Container Native Storage

- Simplified management. All the Kubernetes awesomeness for storage.
- Improve the storage-provisioning experience for app developers.
- Lower costs. Use existing servers to converge apps and storage.
- Abstract away cloud provider details. Gain portability.

redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos

BACKUP

Demo - Kubernetes Persistent Volumes

```
Container starten
/tmp/hello/hello.conf anpassen
Container loeschen -> container auf anderem host, datei weg

oc get pv | grep nfs
Storage in container mounten (nfs storage class, RWX, 1GB)
/tmp/hello/hello.conf anpassen
Container loeschen -> container auf anderem host, datei da

Scale Container
```

Demo - Container Native Storage

```
Show storage project

oc get pv | wc -l

Provision postgres

oc get pv | wc -l
```