DEEP DIVE INTO DOCKER STORAGE

VIVIAN ZHANG (张芸)
@EMC

WHO AM I?

- From EMC Labs China
- Focus on cloud computing
  - OpenStack
  - Cloudfoundry
  - Docker
  - Storage
- Contact info
  - WeChat @vinoyun
  - Twitter @VivianZhang11
  - Email: vivian.zhang@emc.com

AGENDA

- Docker Storage Mechanism
- Persistent Data
- Flocker
- EMC ScaleIO and Docker
- Q&A

DOCKER STORAGE MECHANISM

WE ALL KNOW THAT

- Docker is lightweight
- Docker is fast

WE ALL KNOW THAT

- Docker is lightweight
- Docker is fast

BUT HOW?

LET'S START FROM A SIMPLE SAMPLE

```
root@atsg201:~# docker run -it ubuntu bash
root@c397370ee243:/# apt-get install python
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
```

```
root@c397370ee243:/# python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

WHAT HAPPENED?

- Create a Docker container with its own:
  - file system
  - network stack
  - process space
  - etc.
- Start with a bash process
- Install Python, and run it

WHAT DID NOT HAPPEN?

- Not make a full copy of the Ubuntu image
- Not modify the Ubuntu image itself
- Not affect any other container

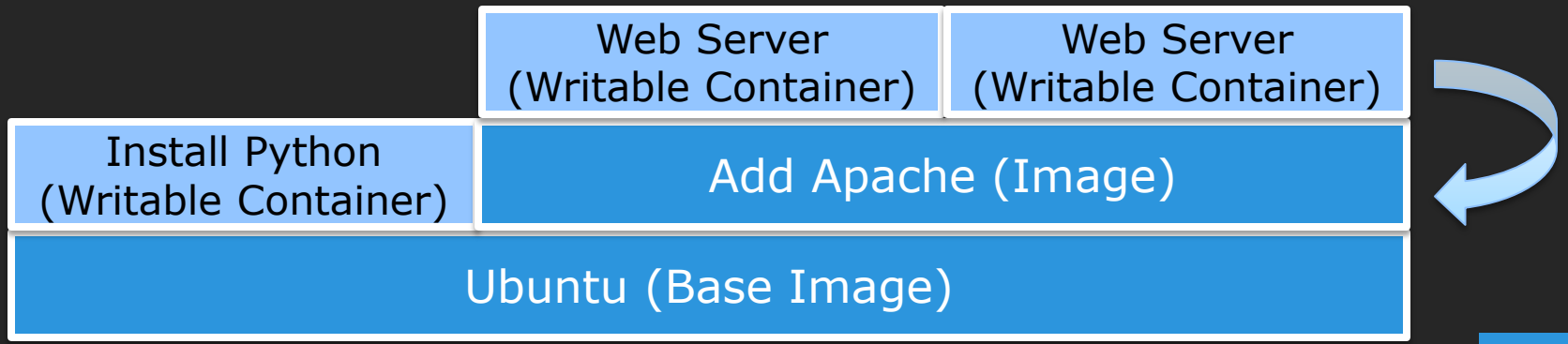
STORAGE FEATURES

- Just track of changes between this image and our containers.

STORAGE FEATURES

- Just track of changes between this image and our containers.

Layer + Copy-on-Write

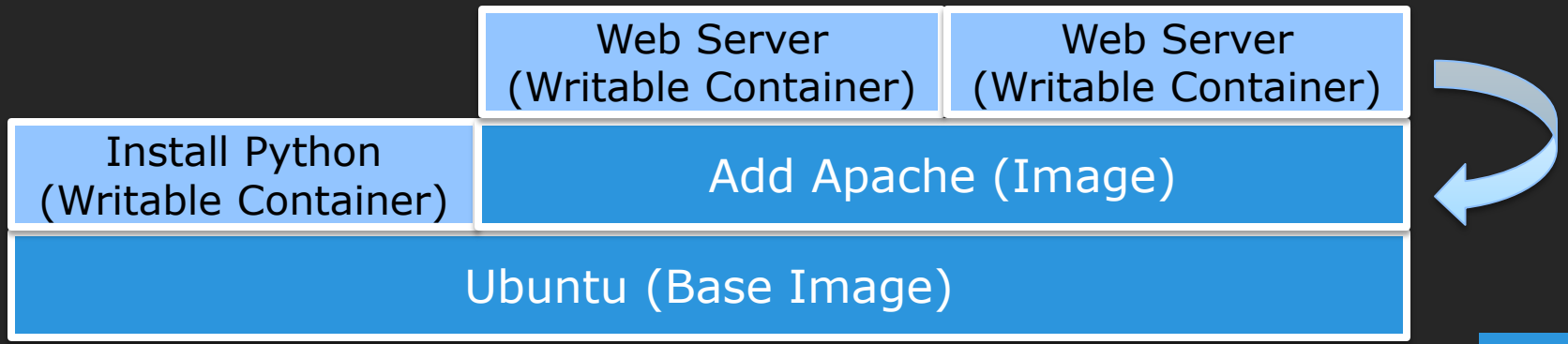


STORAGE FEATURES

- Just track of changes between this image and our containers.

Layer + Copy-on-Write

Docker is lightweight!

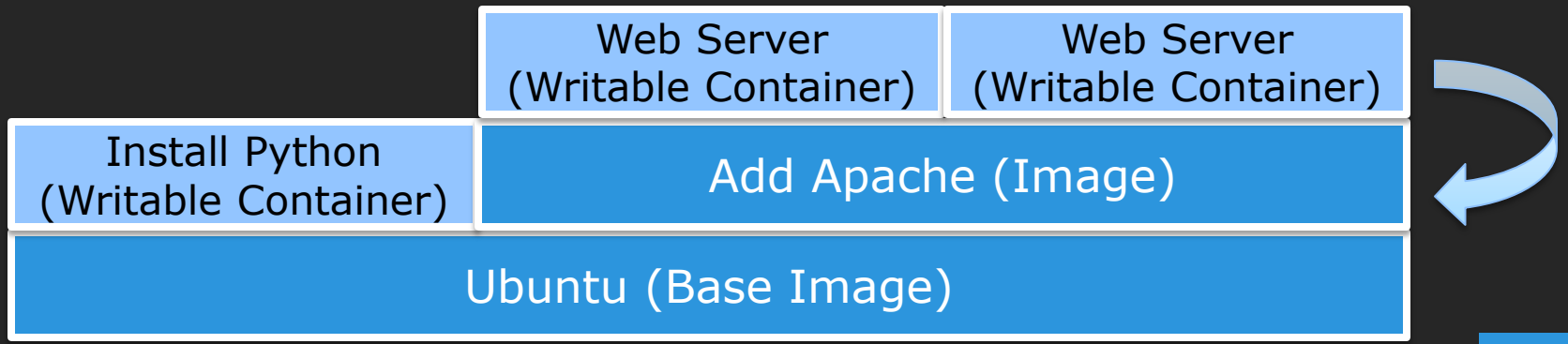


STORAGE FEATURES

- Just track of changes between this image and our containers.

Layer + Copy-on-Write

Docker is lightweight!
Docker is fast!



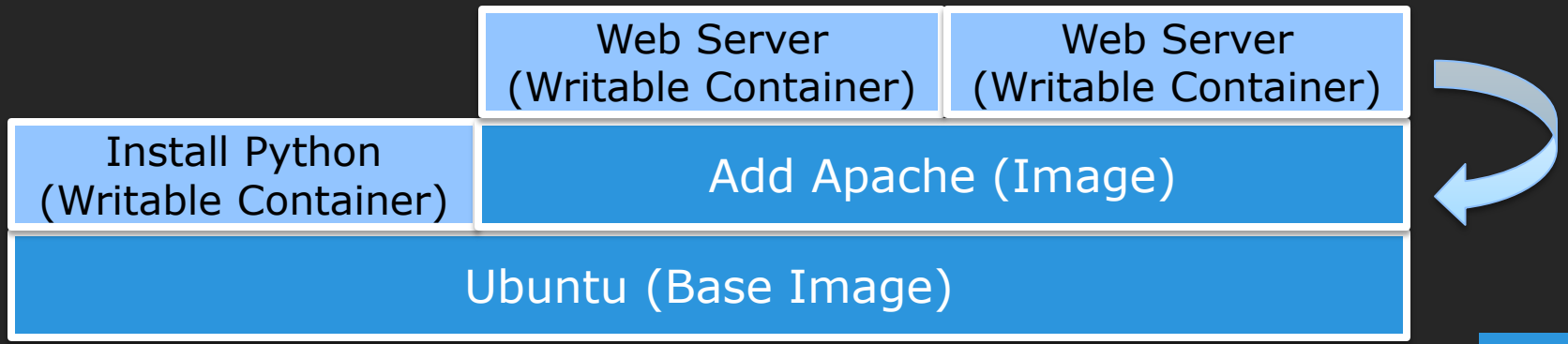
STORAGE FEATURES

- Just track of changes between this image and our containers.

Layer + Copy-on-Write

GraphDB

Docker is lightweight!
Docker is fast!



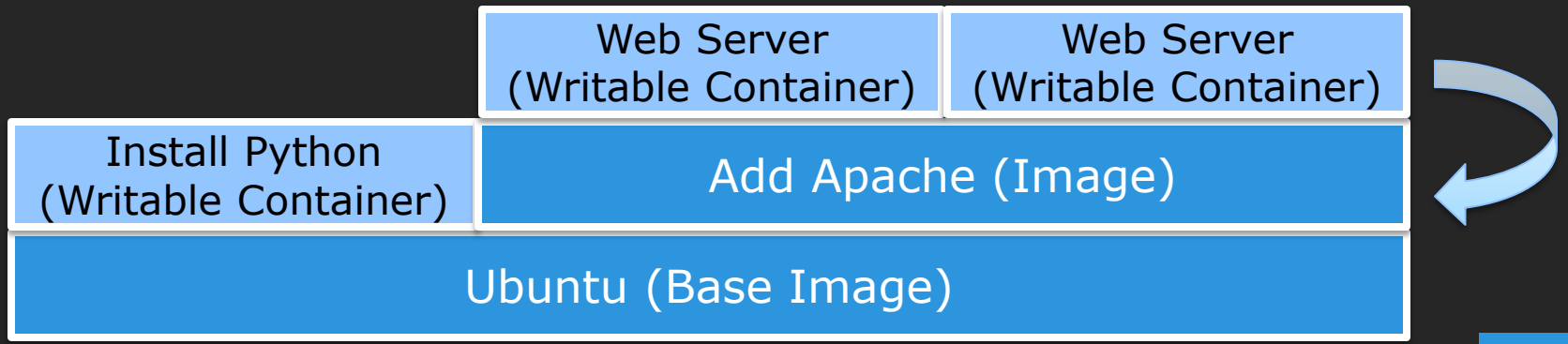
STORAGE FEATURES

- Just track of changes between this image and our containers.

Layer + **Copy-on-Write**

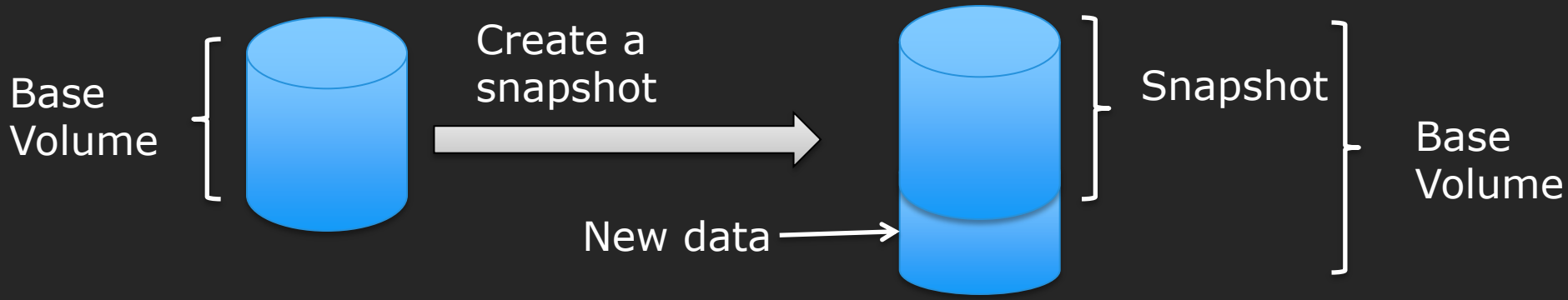
GraphDB | Storage Drivers

Docker is lightweight!
Docker is fast!



COPY ON WRITE

- Snapshot
- LVM, ZFS, BTRFS, AUFS, OverlayFS



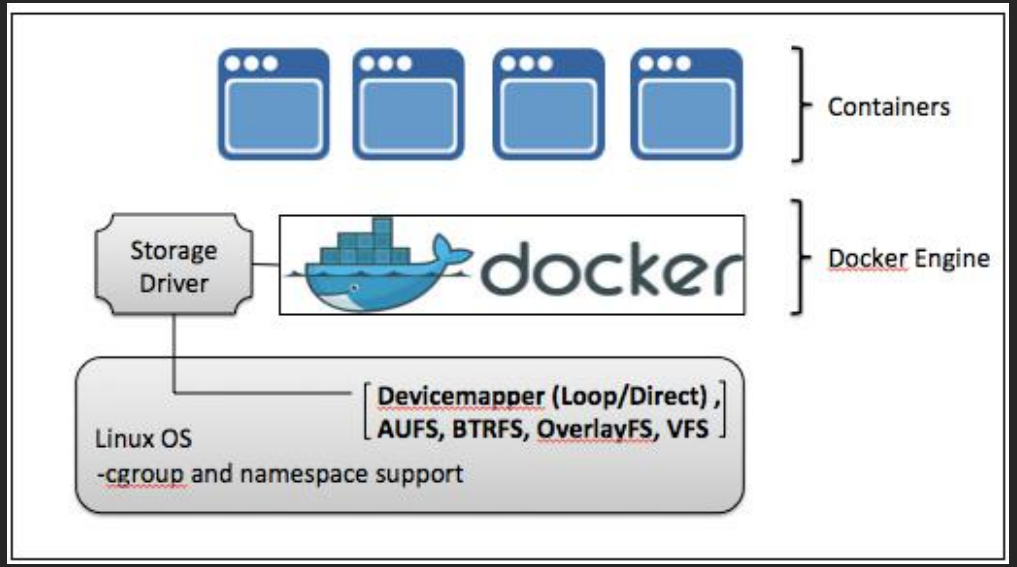
DOCKER STORAGE DRIVERS

- Union Filesystem
  - AUFS: first, but not in Linux mainline
  - OverlayFS: smilar to AUFS, but only two branches
- Block-level Copy-on-Write
  - Device mapper: in Linux kernel 2.6, OpenSuse, CentOS, RHEL
- Copy-on-Write Filesystem
  - BTRFS
  - *ZFS: will be supported in Docker 1.7, released on 18th June*
- Other:
  - VFS: full copy

CONFIGURE DOCKER STORAGE DRIVERS

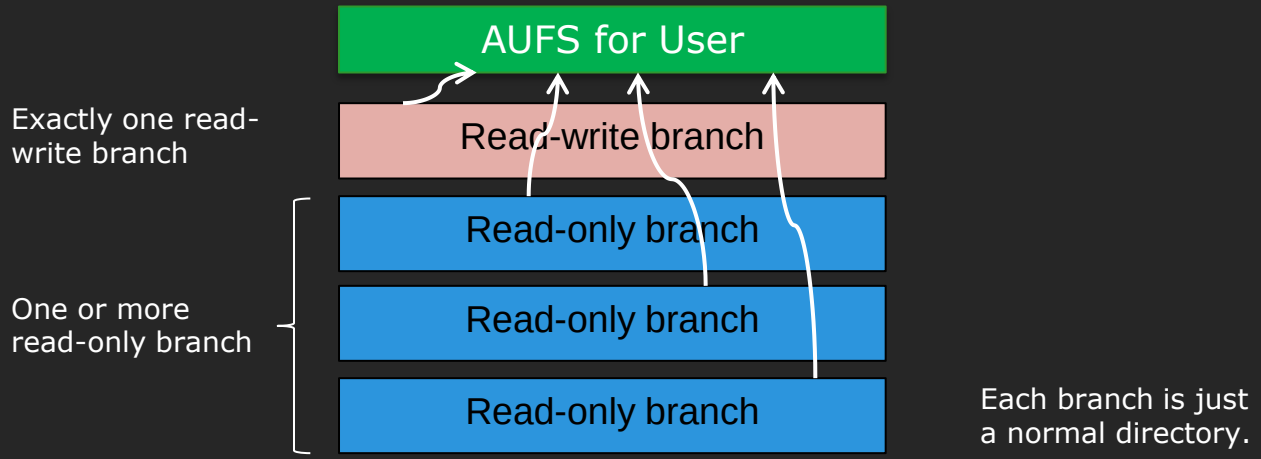
- *docker –s aufs*
- *Configuration file:*

  DOCKER_OPTS="–storage-driver=aufs"



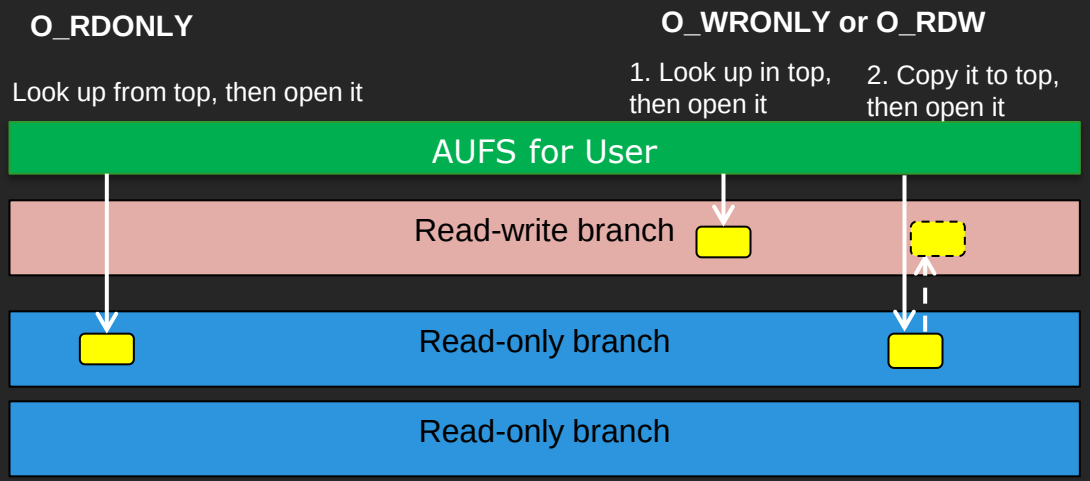
AUFS

Another Union File System



Each branch is just a normal directory.

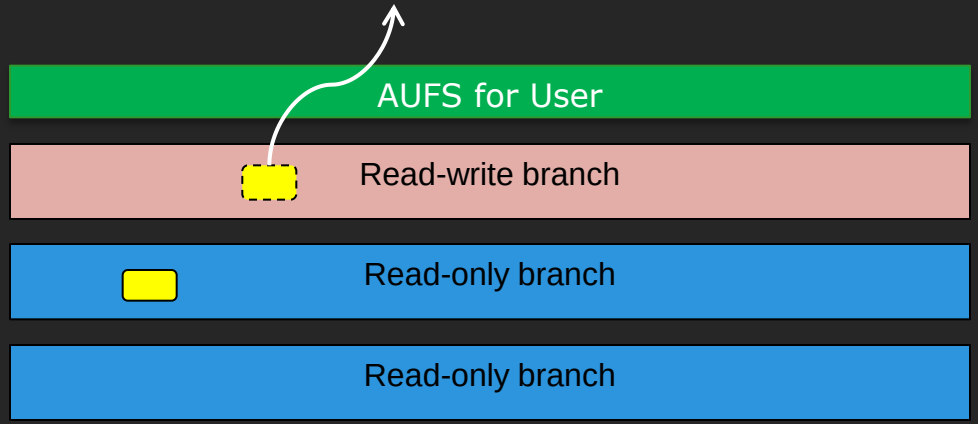
AUFS - OPEN A FILE



AUFS - DELETE A FILE

- A *whiteout* file is created

```
root@atsg201:~# docker run ubuntu rm /etc/shadow
root@atsg201:~# ls -la /var/lib/docker/aufs/diff/$(docker ps --no-trunc -lq)/etc
total 8
drwxr-xr-x 2 root root 4096 Jun  2 13:09 .
drwxr-xr-x 5 root root 4096 Jun  2 13:09 ..
-r--r--r-- 2 root root    0 Jun  2 13:09 .wh.shadow
```



AUFS - IN PRACTICE

- AUFS mountpoint

```
root@atsg201:~# ls /var/lib/docker/aufs/mnt/$(docker ps  --no-trunc -q)
bin   data  etc   lib    media  opt   root  sbin  sys  usr
boot  dev   home  lib64  mnt    proc  run   srv   tmp  var
```

- AUFS branches

```
root@atsg201:~# ls /var/lib/docker/aufs/diff/$(docker ps  --no-trunc -q)
data  etc  tmp  usr  var
```

- All writes go to /var/lib/docker

```
root@atsg201:~# df -h /var/lib/docker
Filesystem                    Size  Used Avail Use% Mounted on
/dev/mapper/atsgxxx--vg-root   14G   14G     0 100% /
```

PERFORMANCE

- Mount is fast, so creation of containers is quick
- Read/write access has native speeds
- Open is slow in two scenarios:
  - Very large files (log files, databases ...)
  - Many directories with many layers

COMPARISON OF STORAGE DRIVERS (*)

	Union Filesystems (AUFS, overlayfs)	Copy-on-write block devices	Snapshotting filesystems
Provisioning	Superfast Supercheap	Average Cheap	Fast Cheap
Changing small files	Superfast Supercheap	Fast Costly	Fast Cheap
Changing large files	Slow (first time) Inefficient (copy-up!)	Fast Cheap	Fast Cheap
Diffing	Superfast	Slow	Superfast
Memory usage	Efficient	Inefficient (at high densities)	Inefficient (but may improve)
Drawbacks	Random quirks AUFS not mainline Overlayfs+Docker is WIP	Higher disk usage Great performance (except diffing)	ZFS not mainline BTRFS not as nice
Bottom line	**Ideal for PAAS, CI/CD, high density things**	**Works everywhere, but slow and inefficient**	**This is the future (Probably!)**

* From Jerome@Docker

BEST PRACTICE (*)

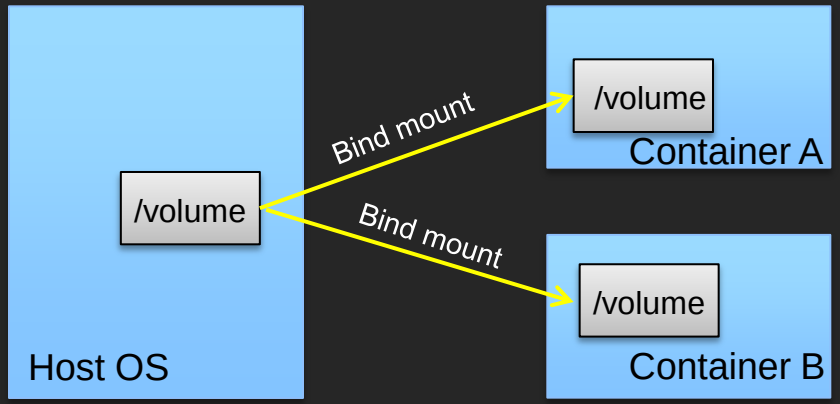
- PaaS or other high-density environment:
  - AUFS
  - OverlayFS
- Big writable files:
  - BTRFS
  - Device mapper

PERSISTENT DATA

WHY WE NEED PERSISTENT DATA BYPASS FILE SYSTEM?

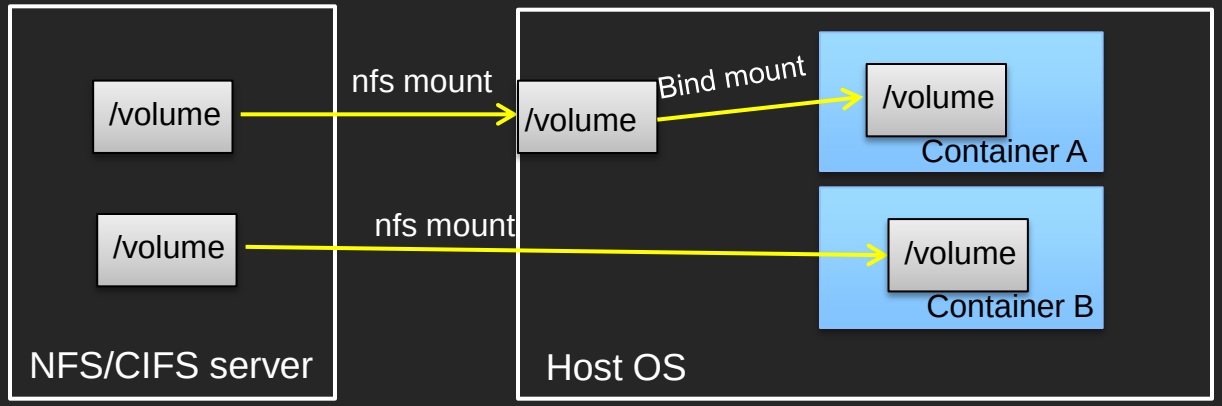
- Share and reuse data between host and container, or among containers:
  - System library directories
  - Configuration files
  - Log
- Separate data from application for later use:
  - Replication, archive, analyze
- Available while:
  - The container is killed
  - The image is upgraded

Host directories, files

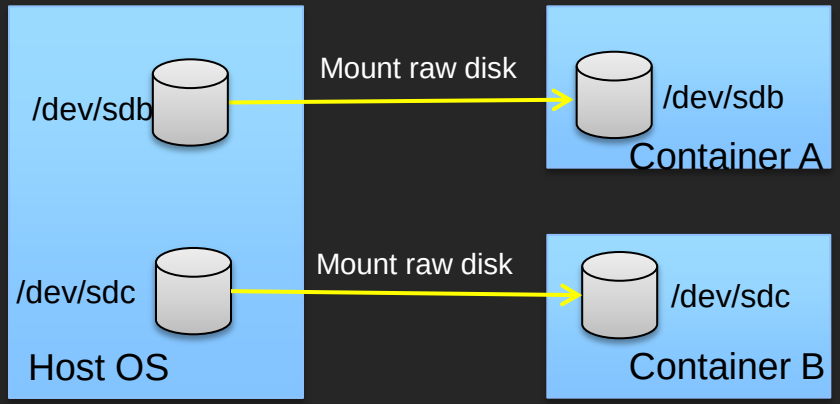


-v, --volume=[]

ATTACHED STORAGE (NFS/CIFS)

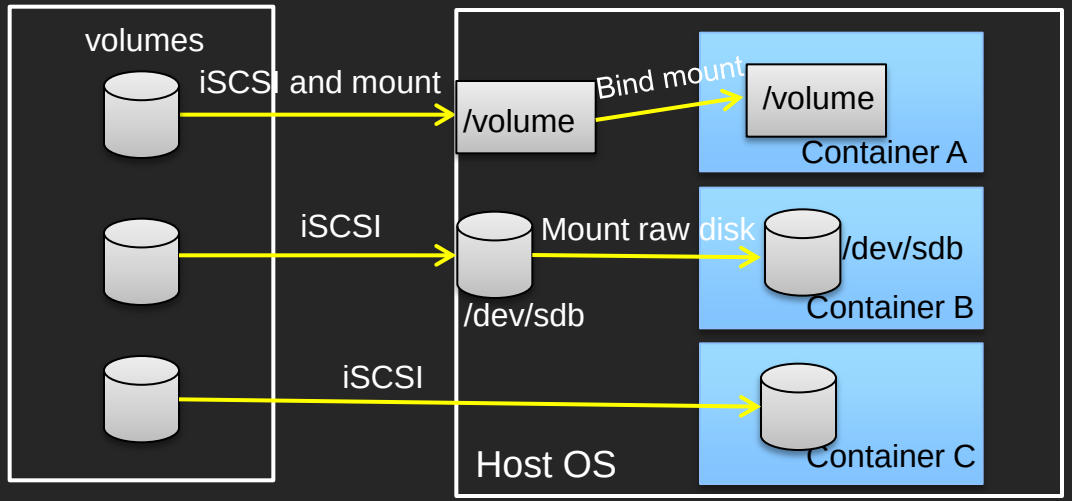


Host Disks

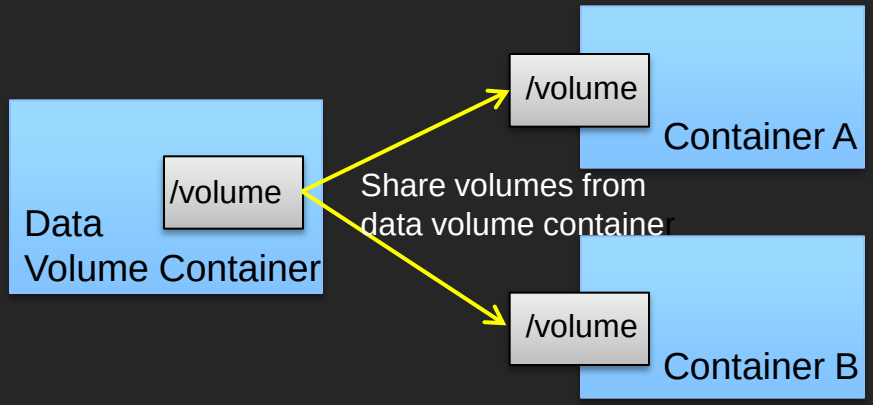


--device=/dev/sdb

ATTACHED STORAGE (ISCSI)



DATA VOLUME CONTAINER



--volumes-from=[]

PROBLEMS

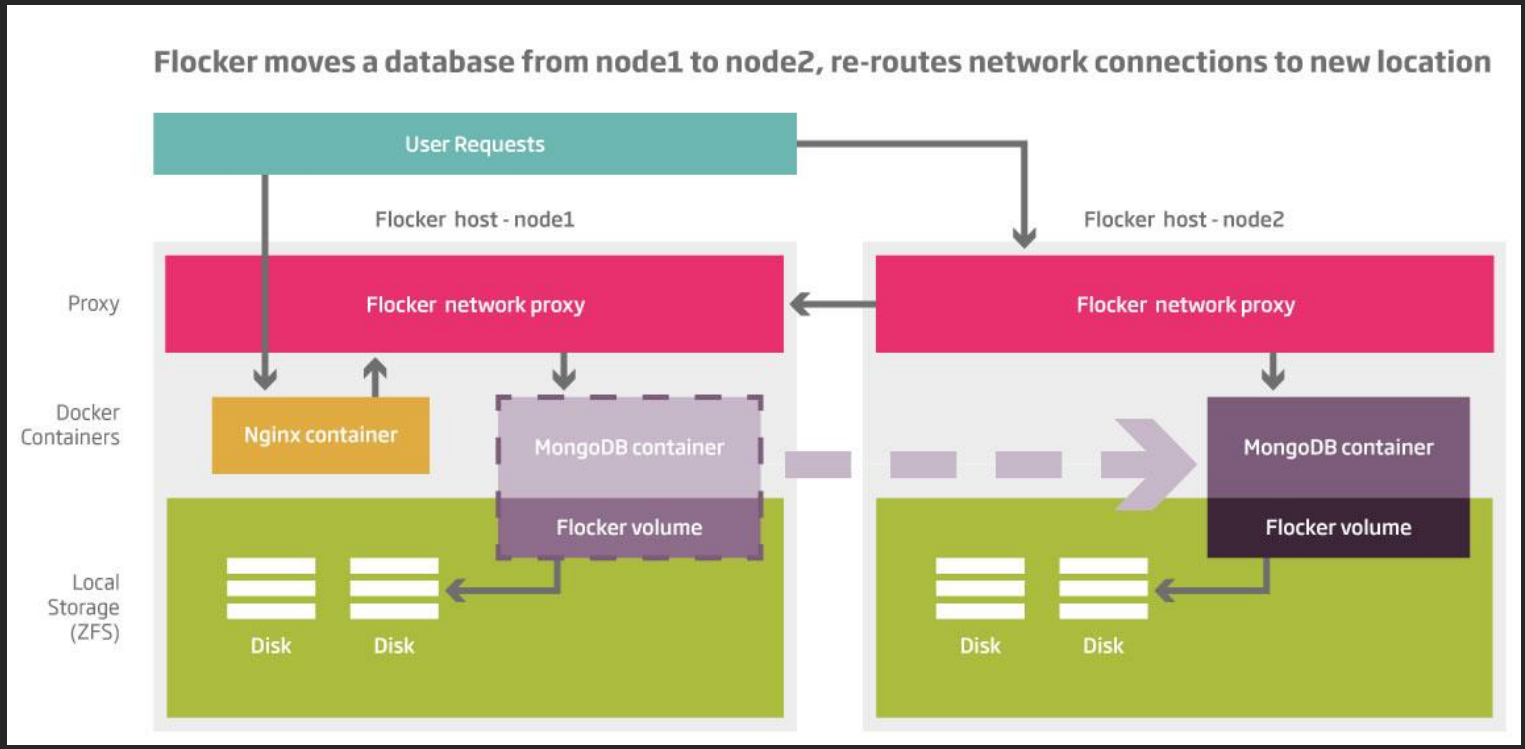
- Portability!

- Data couldn’t be portable as containers.
- Stateful applications (database, key-value store, quenes) need more persistent storage.

FLOCKER - OPEN SOURCE SOLUTION

FLOCKER INTRODUCTION

- By ClusterHQ, open source
- Flocker is a data volume manager and multi-host Docker cluster management tool.
- Move your Docker containers and their data together between Linux hosts:
  - Databases
  - queues
  - key-value stores
- Flocker lets you run microservices apps with database containers and move them around between servers.


Flocker moves a database from node1 to node2, re-routes network connections to new location
User Requests
Flocker host - node1
Flocker host - node2
Proxy
Flocker network proxy
Flocker network proxy
Docker Containers
Nginx container
MongoDB container
MongoDB container
Flocker volume
Flocker volume
Local Storage (ZFS)
Disk
Disk
Disk
Disk

DEPLOY AN APP

fig.yml

```
web:
  image: clusterhq/flask
  links:
   - "redis:redis"
  ports:
   - "80:80"
redis:
  image: dockerfile/redis
  ports:
   - "6379:6379"
  volumes: ["/data"]
```

deployement-node1.yml

```
"version": 1
"nodes":
  "172.16.255.250": ["web", "redis"]
  "172.16.255.251": []
```

```
you@laptop:~$ flocker-deploy deployment-node1.yml fig.yml
```

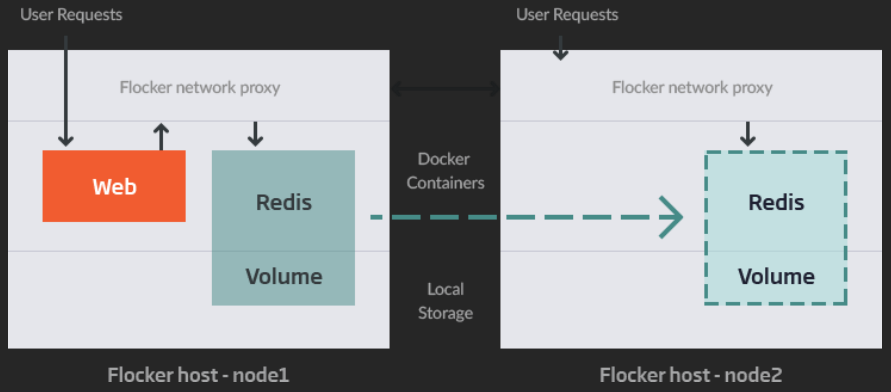
MIGRATE A CONTAINER

deployement-node2.yml

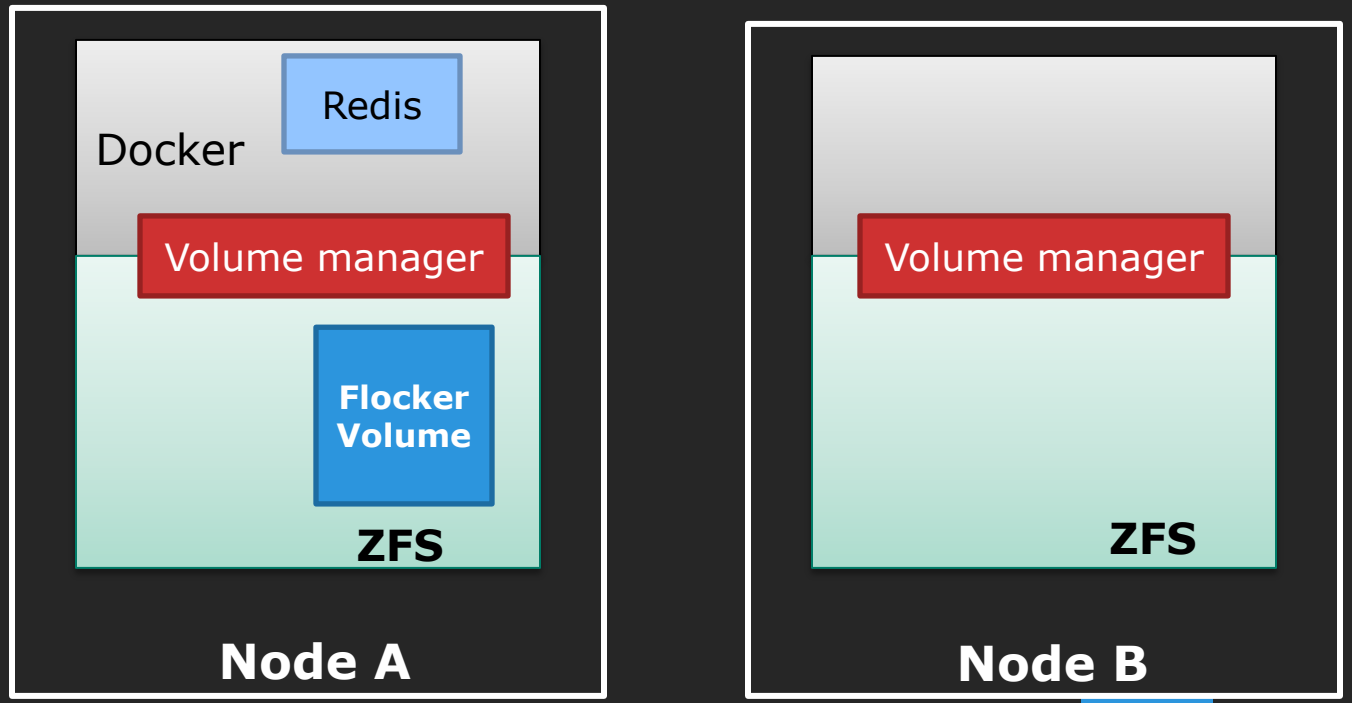
```
"version": 1
"nodes":
  "172.16.255.250": ["web"]
  "172.16.255.251": ["redis"]
```

```
you@laptop:~$ flocker-deploy deployment-node2.yml fig.yml
```

After a few seconds

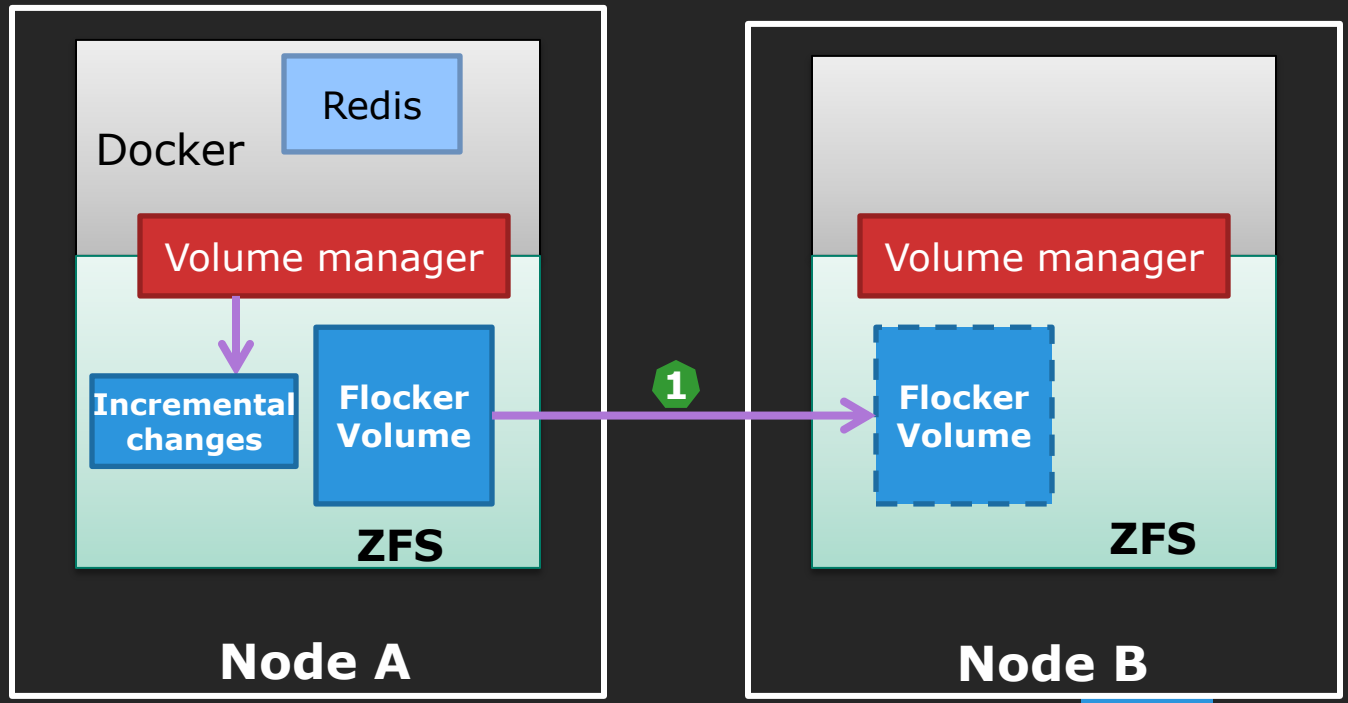


MINIMAL DOWNTIME VOLUME MIGRATION



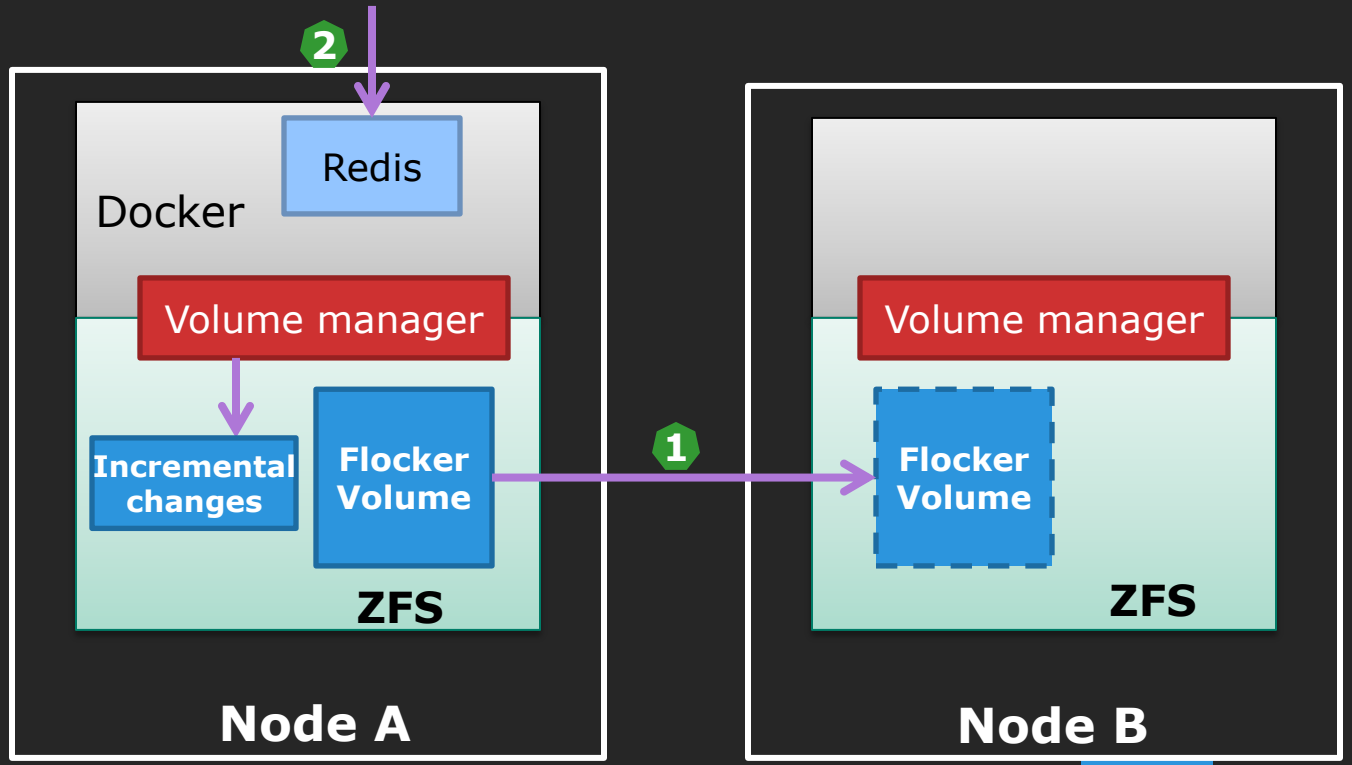
MINIMAL DOWNTIME VOLUME MIGRATION

1. Push full data, add new data to incremental changes



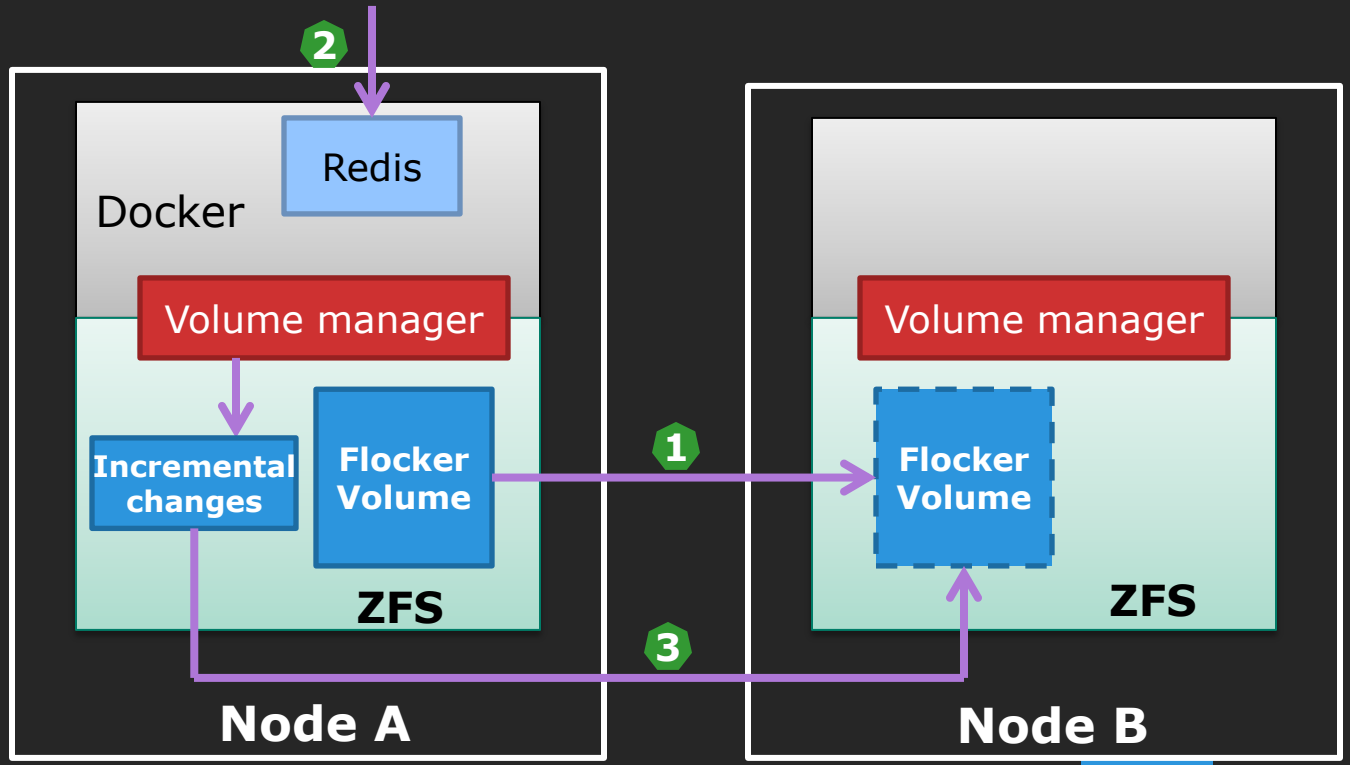
MINIMAL DOWNTIME VOLUME MIGRATION

1. Push full data, add new data to incremental changes
2. Shutdown Redis



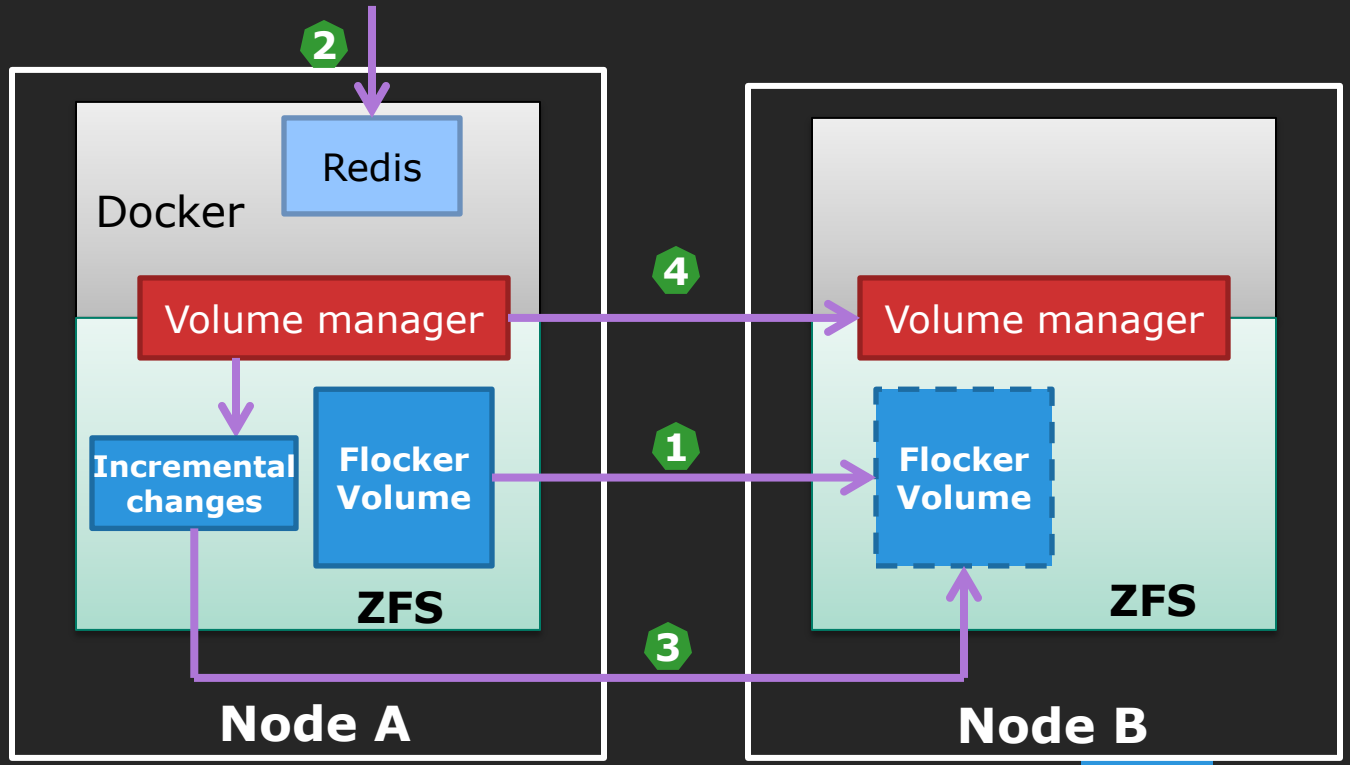
MINIMAL DOWNTIME VOLUME MIGRATION

1. Push full data, add new data to incremental changes
2. Shutdown Redis
3. Push changes



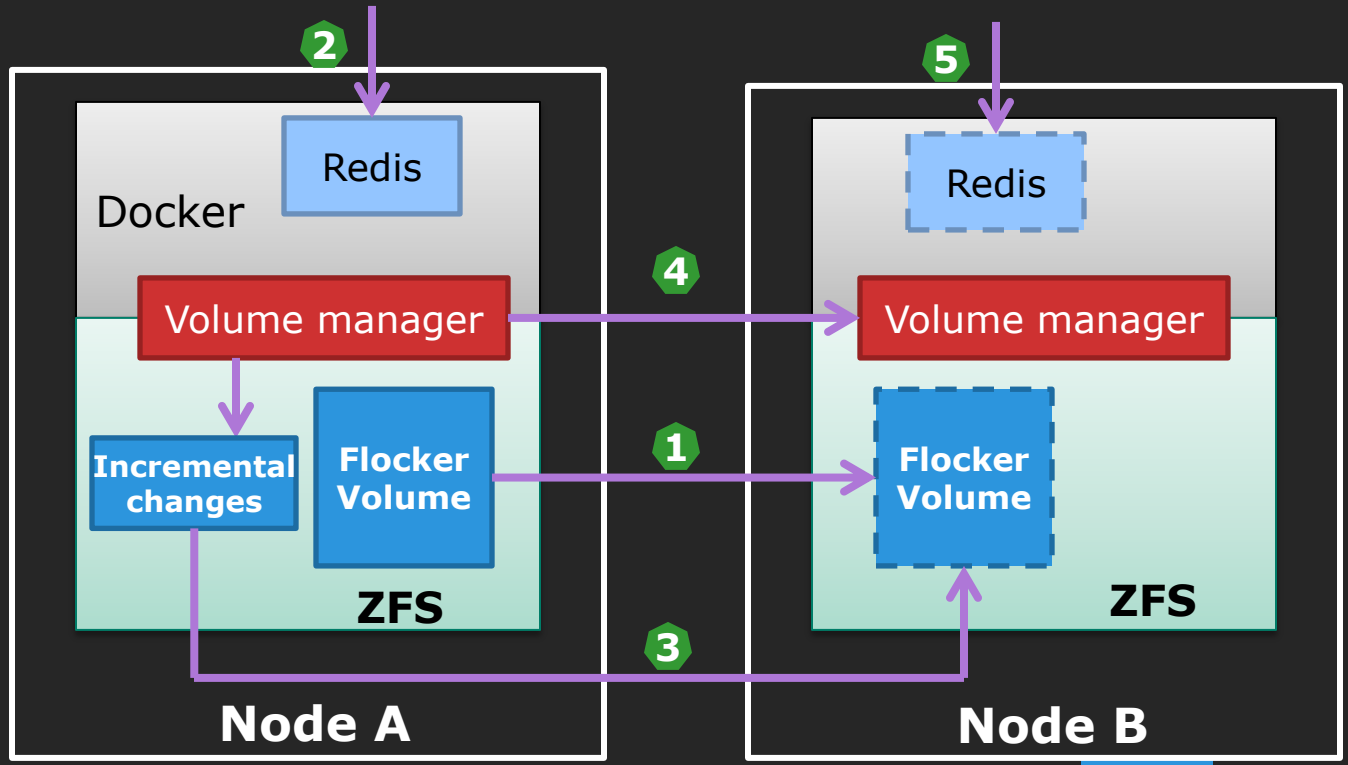
MINIMAL DOWNTIME VOLUME MIGRATION

1. Push full data, add new data to incremental changes
2. Shutdown Redis
3. Push changes
4. Hand off



MINIMAL DOWNTIME VOLUME MIGRATION

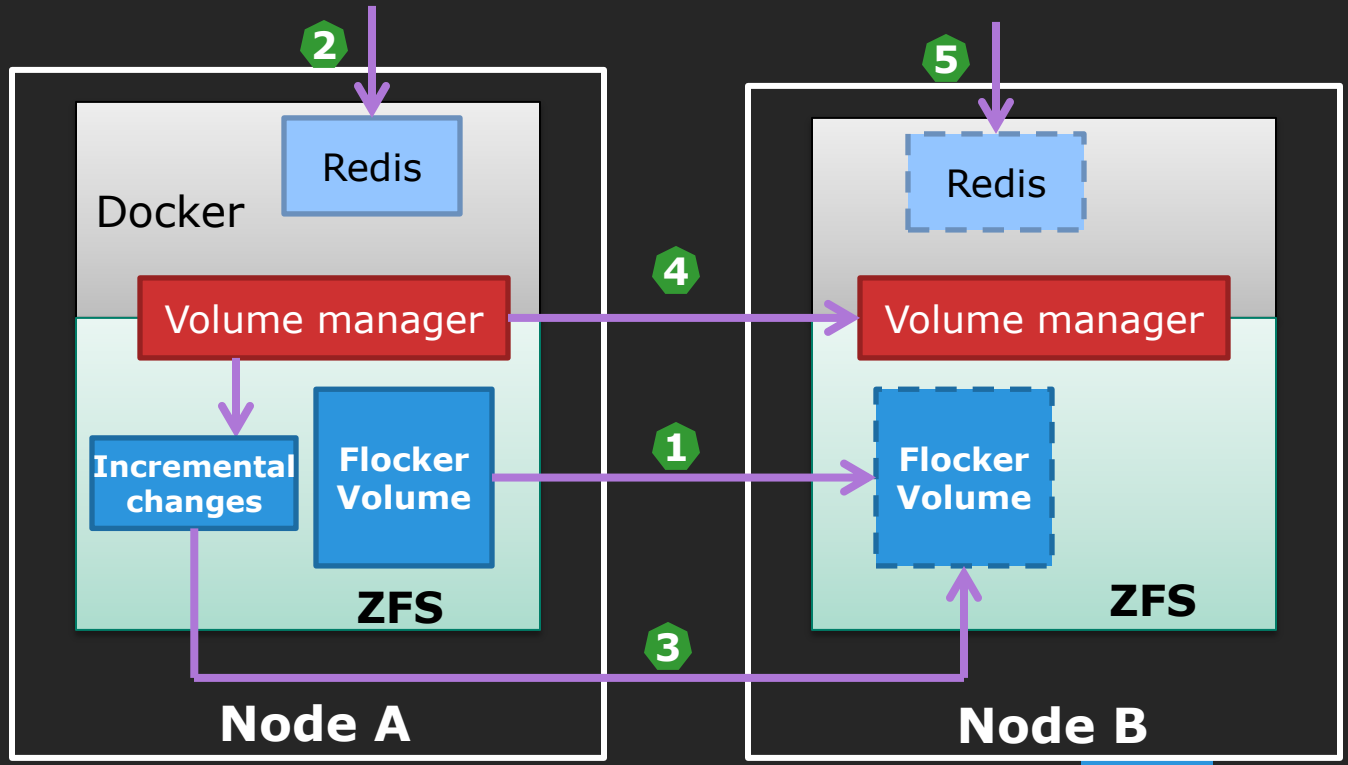
1. Push full data, add new data to incremental changes
2. Shutdown Redis
3. Push changes
4. Hand off
5. Start new Redis



MINIMAL DOWNTIME VOLUME MIGRATION

1. Push full data, add new data to incremental changes
2. Shutdown Redis
3. Push changes
4. Hand off
5. Start new Redis

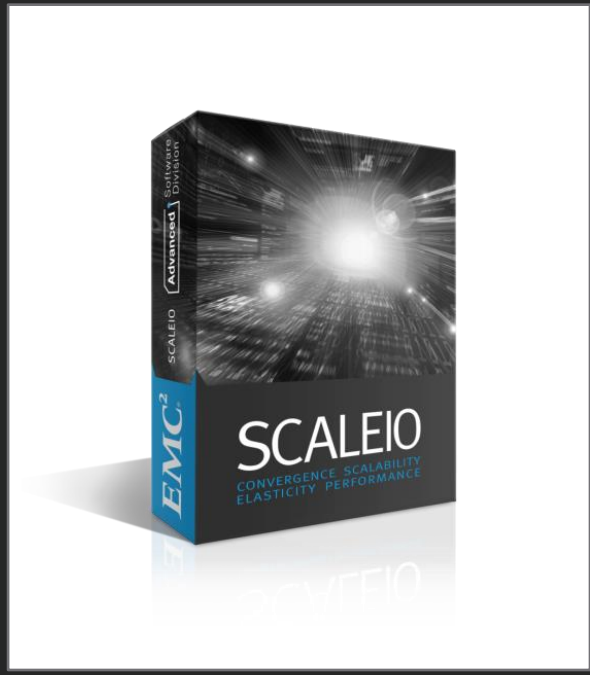
Core technology: Copy-on-write Snapshot



EMC SCALEIO AND DOCKER -- CONVERGED INFRASTRUCTURE FOR DATA PERSISTENCE

WHAT IS SCALEIO

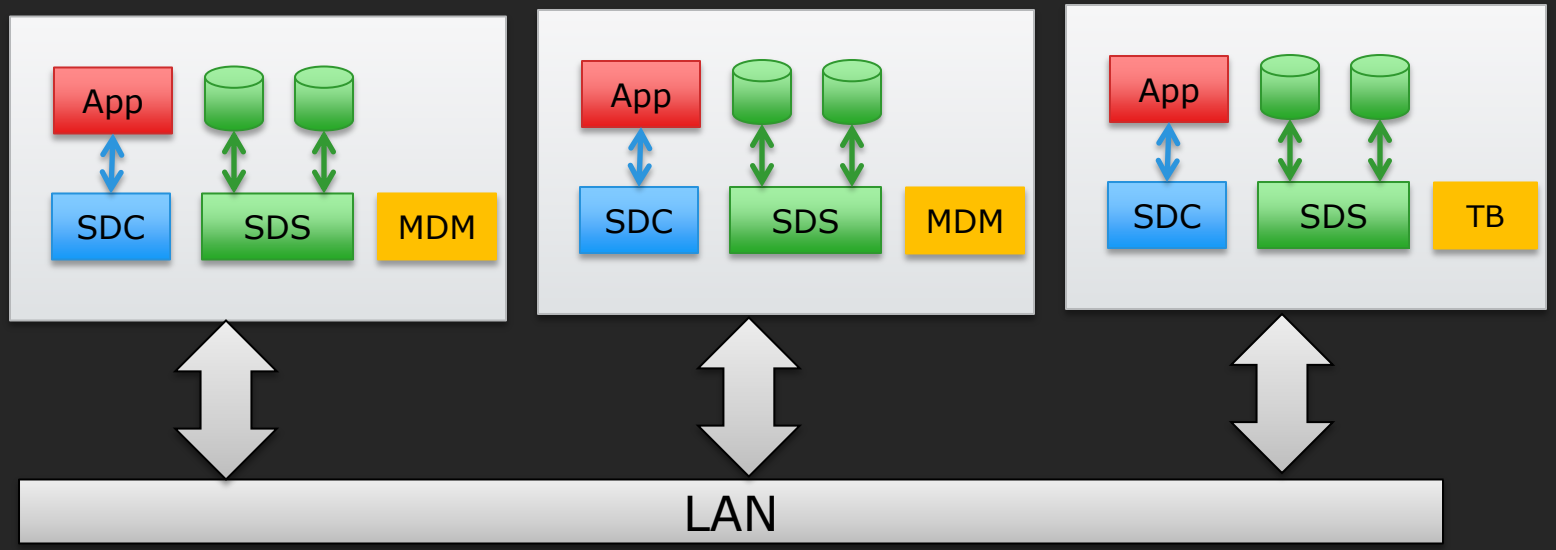
- Elastic converged infrastructure
- Distributed software-only solution
- Using servers' local disks and LAN/IB
- Create a virtual SAN
- Similar to VMWare vSAN



KEY BENEFITS

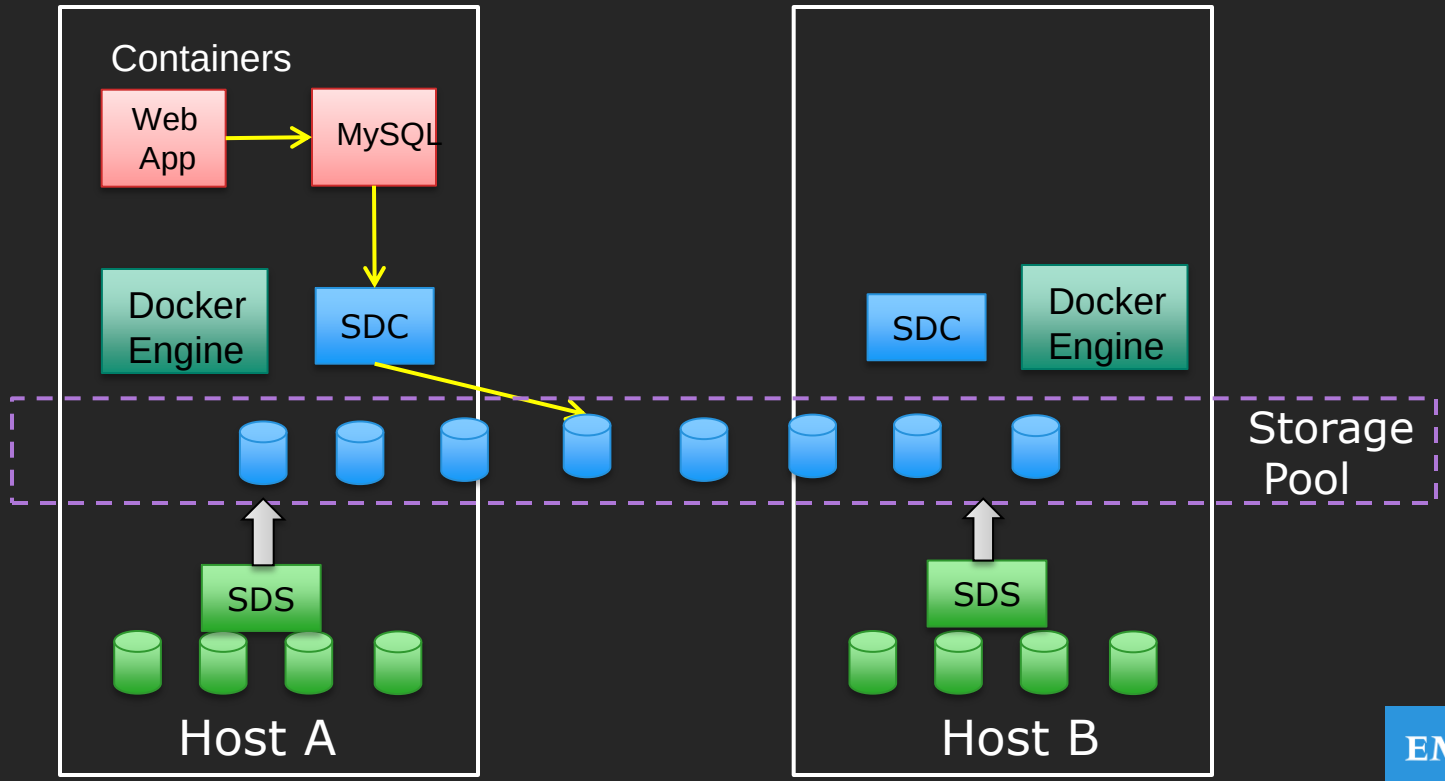
- Convergence of storage and compute
- Scale-out to thousands of servers
- Elastic—add/remove servers & capacity "on the fly"
- Performance—massive I/O parallelism

SCALEIO ARCHITECTURE

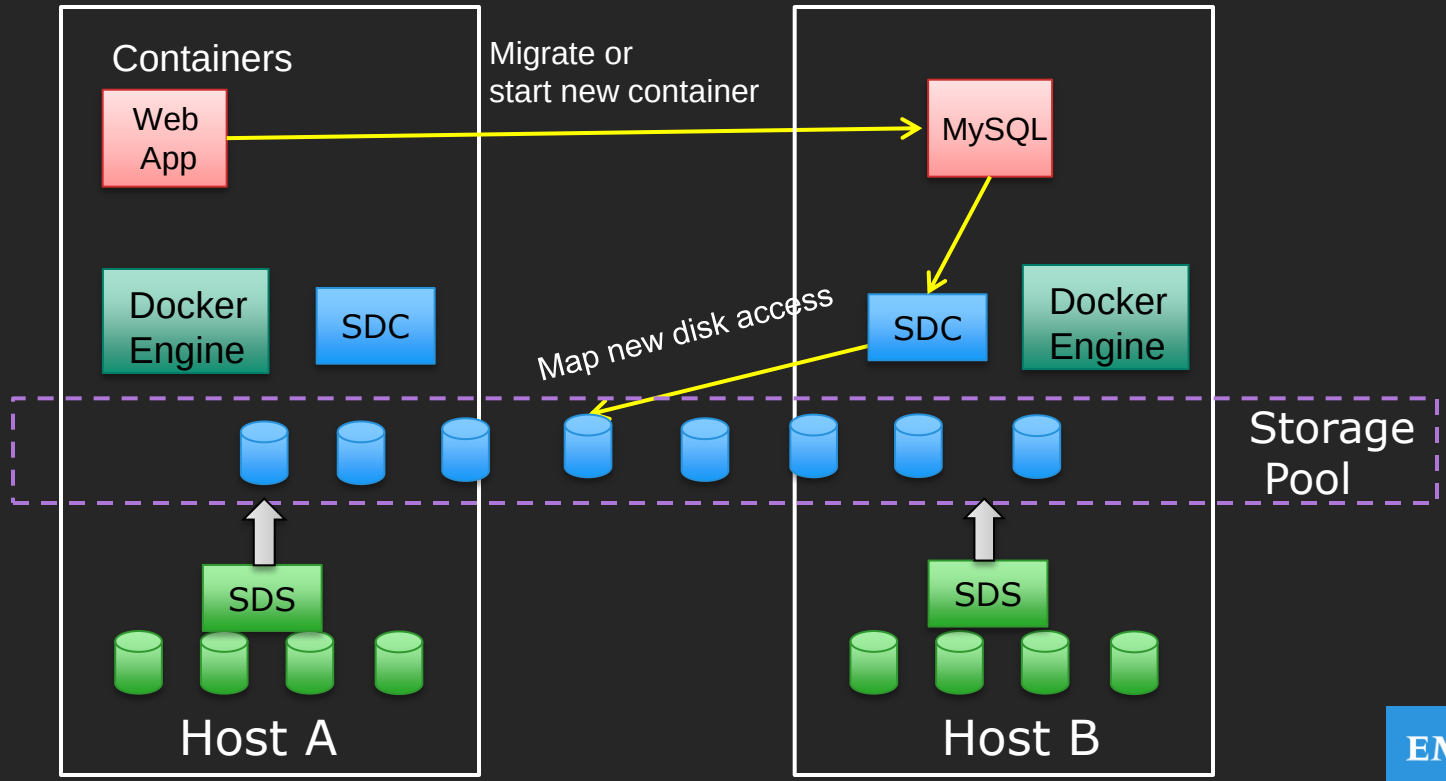


SDC: ScaleIO data client
SDS: ScaleIO data server
MDM: Meta Data Manager
TB: Tier Break

SCALEIO + DOCKER ENV.



MIGRATE THE MYSQL CONTAINER



How ScaleIO benefits Docker?

- Shared storage, no need to migrate data, just need to map the new disk access.
- P2p, load-balance.
- Aggregate local disks from nodes to a global pool.

EMC中国研究院订阅号


EMC²