Hi everyone, thanks David for inviting me to speak about a topic that has way too many buzzwords and confusion. So in this talk, rather than give you the One Persistent Storage Solution To Rule Them All, I'm going to give you a framework that you can use to figure out what you need.

Cheryl
@oicheryl

STORAGEOS

A little about me, my name is Cheryl and I'm an ex-Google engineer and spent five years writing C++ on Google Maps before joining StorageOS as product manager.

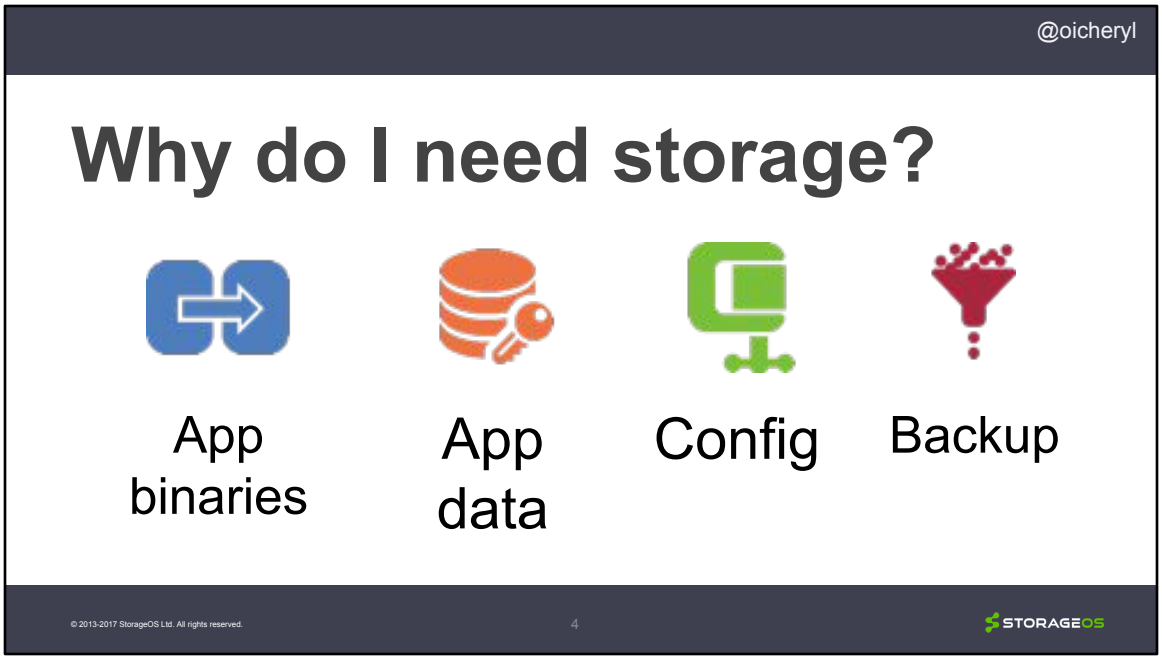
We'll start from the very beginning and ask

Why do I need storage?

why do I need storage? Stupid question right? Well yes, but applications typically have more than one storage requirement.

Application binaries need ephemeral, performance storage.
Application data (what most people think of, like databases, message queues) need dedicated persistent performance, for example block storage for databases, replication for high availability etc., snapshots for point-in-time copies and encryption.
Configuration needs to be shared and persistent, typically filesystem.
For backup, you want it to be cost efficient, so you'd be looking for compression and deduplication, maybe back up to the cloud.

So why is this difficult with containers?



why do I need storage? Stupid question right? Well yes, but applications typically have more than one storage requirement.

Application binaries need ephemeral, performance storage.
Application data for running databases, message queues need dedicated persistent performance, for example block storage, replication for high availability etc., snapshots and encryption.
Configuration files needs to be shared and persistent.
For backup, you want it to be cost efficient, so you'd be looking for compression and deduplication, maybe back up to the cloud.

You probably want more than one storage solution. So why is this difficult with containers?

Why is this tricky with containers?

First of all, does anyone not know what containers are?

Applications have become loosely coupled and stateless
Designed to scale and manage failure – it is no longer economical to remediate state

So why is this difficult with containers?

No pets

First of all, You know the cattle/pet analogy? Don't treat your servers like pets that you have to lovingly name and take care of, treat them like cattle that when they get ill, you take them out back and shoot them.

In other words, you don't want special storage pets, you want ordinary commodity hardware or cloud instances.

@oicheryl

Data follows

7

STORAGEOS

Secondly, data needs to follow containers around. When nodes fail you want to reschedule that container to another node and for data to follow it.

You want to avoid mapping containers to individual hosts - then you've lost your portability and mobility.

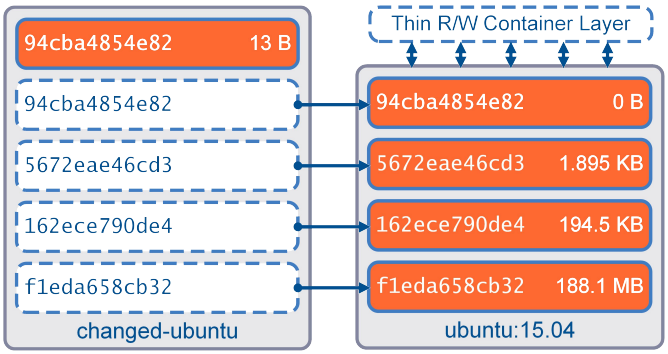
Humans are failable

Thirdly, humans are failable. Don’t rely on someone running through a playbook, they will screw things up.

You want to manage storage through APIs and you want that integrated with Docker and Kubernetes. So let’s take a closer look at Docker.

@oicheryl

Docker container layers



STORAGEOS

Docker containers comprise a layered image and a writable ‘Container layer’. Here, the base image is Ubuntu and the top right is the thin r/w container layer, where new or modified data is stored.
When a container is deleted its writable layer is removed leaving just the underlying image layers behind.

This is good because sharing layers makes images smaller and lack of state makes it easy to move containers around.
This is bad because generally you want your app to do something useful in the real world.

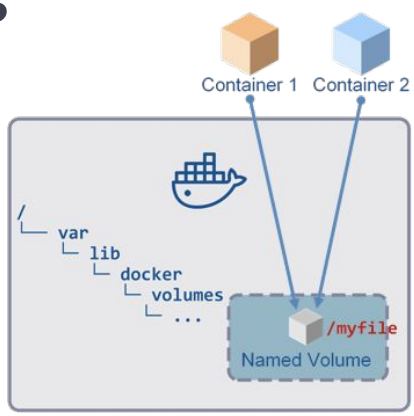
So we can look at options with Docker.

Docker local volumes

```
$ docker volume create --name mydata
```

```
$ docker run --rm -v mydata:/data:rw alpine ash -c \
  "echo hello world > /data/myfile"
```

```
$ sudo cat /var/lib/docker/volumes/mydata/_data/myfile
hello world
```

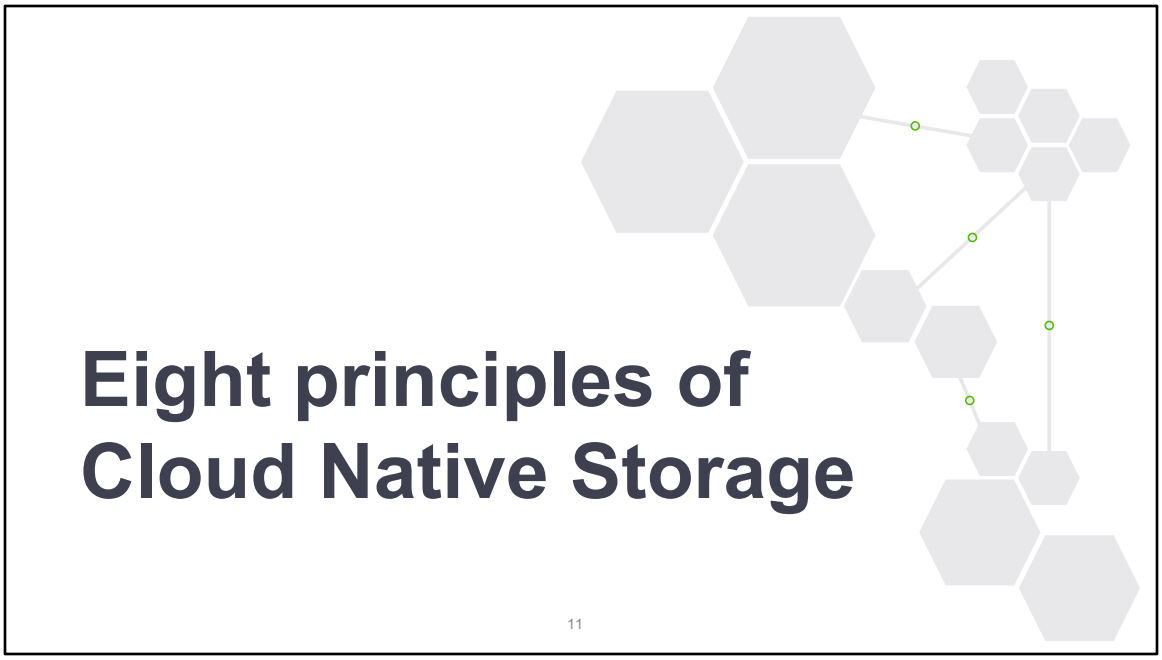


Docker volumes allow you to share data between host and containers through local volumes.

Here you run docker volume create which creates a volume called mydata, mount that into the container at /data, and write a file. When the container exits, it's persisted under /var/lib/docker/volumes.

On the plus side, you can't get faster than writing to your local host. The downside is that because the data tied to that host, it doesn't follow you around and if that host goes down your data is inaccessible. Plus there's no locking so you have to be careful with consistency, plus it's subject to the noisy neighbour problem.

To extend storage Docker has volume plugins, but first I want to



lay out the eight principles of cloud native storage, then we'll look at some of the most popular ones out there and evaluate them based on these eight principles.

First off,

What is Cloud Native?

Horizontally scalable
Built to handle failures
Resilient and survivable
Minimal operator overhead
Decoupled from the underlying platform

What do I mean by cloud native?

Horizontally scalable
Built to handle failures, so no single point of failure
Resilient and survivable, in other words it should be self healing
Minimal operator overhead - it should be API-driven and automatable
Decoupled from the underlying platform and hardware.

How does that apply to storage?

Eight principles of Cloud Native Storage

1. Platform agnostic

The storage platform should be able to run anywhere and not have proprietary dependencies that lock an application to a particular platform or a cloud provider. Additionally, it should be capable of being scaled out in a distributed topology just as easily as it could be scaled up based on application requirements. Upgrades and scaling of the storage platform should be implemented as a non-disruptive operation to the running applications.

Eight principles of Cloud Native Storage

1. Platform agnostic
2. API driven

Storage resources and services should be easy to be provisioned, consumed, moved and managed via an API, and provide integration with application runtime and orchestrator platforms.

Eight principles of Cloud Native Storage

1. Platform agnostic
2. API driven
3. Declarative and composable

Storage resources should be declared and composed just like all other resources required by applications and services, allowing storage resources and services to be deployed and provisioned as part of application instantiation through orchestrators.

@oicheryl

Eight principles of Cloud Native Storage

1. Platform agnostic
2. API driven
3. Declarative and composable
4. Application centric

STORAGEOS

Storage should be presented to and consumed by applications and not by operating systems or hypervisors. It is no longer desirable to present storage to operating systems instances, and then, later, have to map applications to operating system instances to link to storage (whether on-premises or in a cloud provider, or on VMs or bare metal) . Storage needs to be able to follow an application as it scales, grows, and moves between platforms and clouds.

@oicheryl

Eight principles of Cloud Native Storage

1. Platform agnostic
2. API driven
3. Declarative and composable
4. Application centric
5. Agile

STORAGEOS

The platform should be able to dynamically react to changes in the environment and be able to move application data between locations, dynamically resize volumes for growth, take point in time copies of data for data retention or to facilitate rapid recovery of data, and integrate naturally into dynamic, rapidly changing application environments.

Eight principles of Cloud Native Storage

1. Platform agnostic
2. API driven
3. Declarative and composable
4. Application centric
5. Agile
6. Natively secure

Storage services should integrate and inline security features such as encryption and RBAC and not depend on secondary products to secure application data.

Eight principles of Cloud Native Storage

1. Platform agnostic
2. API driven
3. Declarative and composable
4. Application centric
5. Agile
6. Natively secure
7. Performant

The storage platform should be able to offer deterministic performance in complex distributed environments and scale efficiently using a minimum of compute resources.

Eight principles of Cloud Native Storage

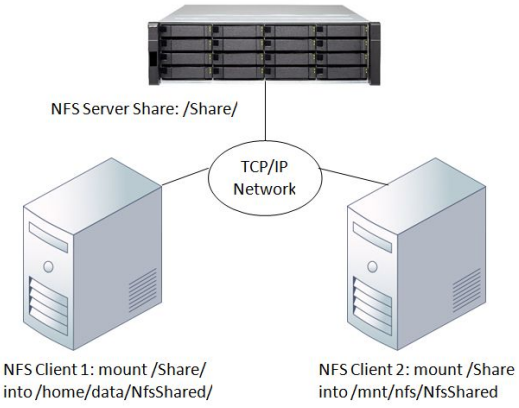
1. Platform agnostic
2. API driven
3. Declarative and composable
4. Application centric
5. Agile
6. Natively secure
7. Performant
8. Consistently available

The storage platform should manage data distribution with a predictable, proven data model to ensure high availability, durability, consistency of application data. During failure conditions, data recovery processes should be application independent and not affect normal operations.

So that's the eight principles of cloud native storage. Now we'll take a look at few different paradigms, a popular example of each and score them against those 8 principles.

Warning in advance, this is really high level but I hope I can put these into the context of running them with Docker.

Centralised file system: NFS



Your classic NAS or network attached storage is NFS. You take one NFS server and export the local filesystem over the network to a number of clients. Who here uses NFS?

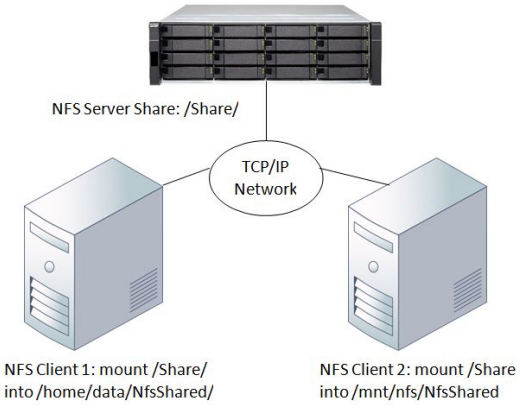
It doesn't really follow any of the eight principles; it's hard to scale horizontally, it's not integrated into Docker or Kubernetes natively, and it's a single point of failure so there's no availability guarantees, although there are commercial options for failover.

First designed in 1984 - definitely not cloud native. I've scored NFS 0.

@oicheryl

Centralised file system: NFS

0



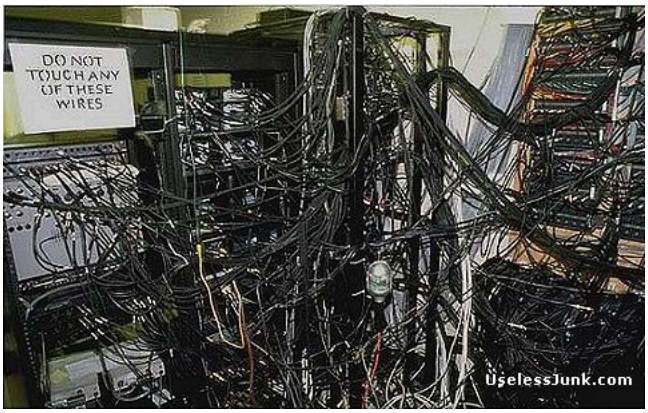
STORAGEOS

Your classic NAS or network attached storage is NFS. You take one NFS server and export the local filesystem over the network to a number of clients. Who here uses NFS?

It doesn't really follow any of the eight principles; it's hard to scale horizontally, it's not integrated into Docker or Kubernetes natively, and it's a single point of failure so there's no availability guarantees, although there are commercial options for failover.

First designed in 1984 - definitely not cloud native. I've scored NFS 0.

Storage array: Dell EMC



Next up is the classic hardware storage array like Dell EMC. This image is totally unfair of course but it does show the complexity.

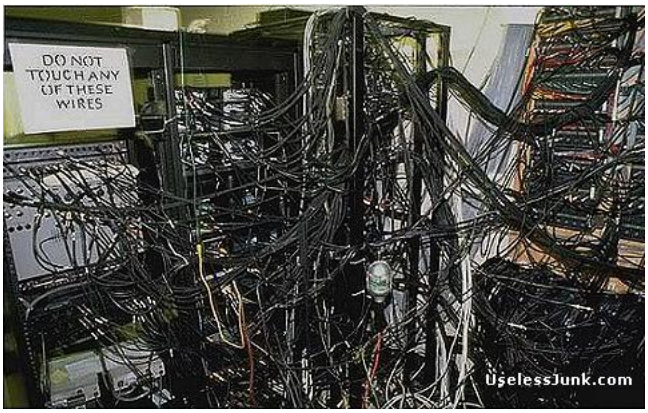
It's even less platform agnostic, since you have to buy the hardware from a specific vendor, and typically is easier to scale up than horizontally. It's also absurdly expensive, has long lead times, inefficient (no thin provisioning) and definitely not application centric.

But it's optimised for deterministic performance, which is why many enterprises who use databases still use storage arrays. Anybody using storage arrays from Dell EMC, HPE, NetApp, Hitachi and so on?

Also not very cloud native - I've given it 2.

Storage array: Dell EMC

2



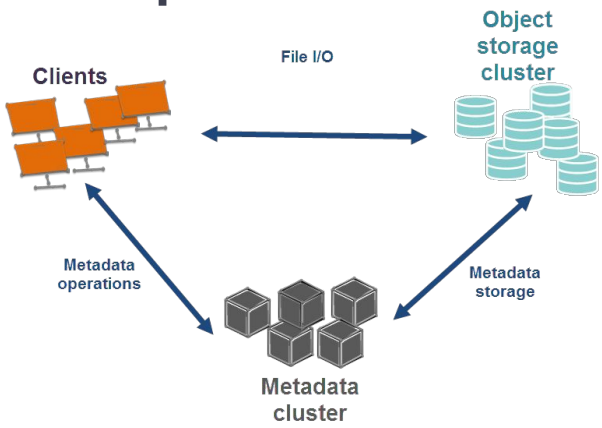
Next up is the classic hardware storage array like Dell EMC. This image is totally unfair of course but it does show the complexity.

It's even less platform agnostic, since you have to buy the hardware from a specific vendor, and typically is easier to scale up than horizontally. It's also absurdly expensive, has long lead times, inefficient (no thin provisioning) and definitely not application centric.

But it's optimised for deterministic performance, which is why many enterprises who use databases still use storage arrays.

Also not very cloud native - I've given it 2.

Distributed: Ceph



Jumping ahead, let's talk about distributed storage like Ceph, which is a distributed object store maintained by Red Hat. Distributed architectures typically trade off performance against consistency, so you get better performance if you don't need strong consistency.

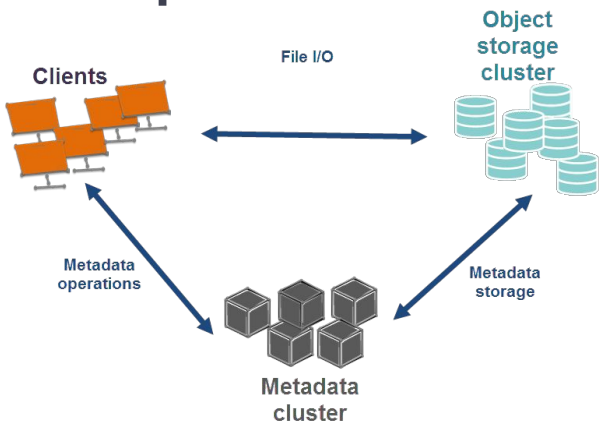
So how cloud native is Ceph? Distributed architectures are usually designed with scaling in mind, and although it's not natively integrated with Docker or K8s you can look into a project called Rook for the latter.

The big downsides of Ceph is that it's complicated to set up, and failures are expensive. Because of the way the data is distributed across all nodes in a cluster means that any failures need rebuilding from the whole cluster. The distributed architecture also means that one write fans out between 13 and 40 times, which limits the performance you can get from your cluster.

4/8 By the way, if you want to look at the numbers I'm giving, these slides are on oicheryl.com.

Distributed: Ceph

4



Jumping ahead, let's talk about distributed storage like Ceph, which is a distributed object store maintained by Red Hat. Distributed architectures typically trade off performance against consistency, so you get better performance if you don't need strong consistency.

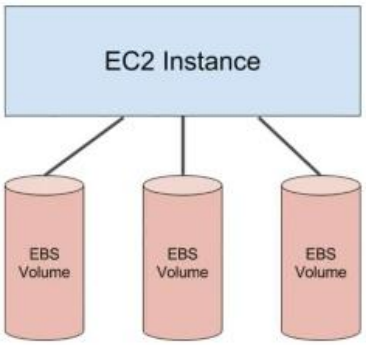
So how cloud native is Ceph? Distributed architectures are usually designed with scaling in mind, and although it's not natively integrated with Docker or K8s you can look into a project called Rook for the latter.

The big downsides of Ceph is that it's complicated to set up, and failures are expensive. Because of the way the data is distributed across all nodes in a cluster means that any failures need rebuilding from the whole cluster. The distributed architecture also means that one write fans out between 13 and 40 times, which limits the performance you can get from your cluster.

4/8 By the way, if you want to look at the numbers I'm giving, these slides are on oicheryl.com.

@oicheryl

Public cloud: AWS EBS



STORAGEOS

Now if you're a hipster and you want to join all the cool kids on public cloud, then EBS is a popular option, which stands for Elastic block storage.

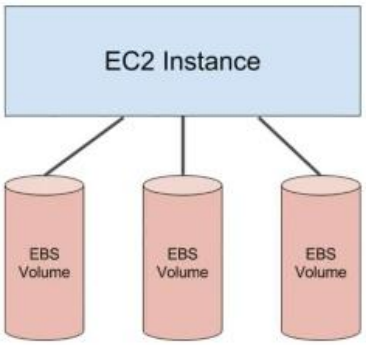
EBS is pretty nice, everything is scalable, highly consistent and high performance. On the downside, you have a maximum of 40 EBS volumes you can mount per EC2 instance, which limits how many containers you can run per EC2 instance, and when you move containers between hosts, mounting physical block devices to nodes takes at least 45 seconds.

I'll also mention S3 which is Amazon's eventual consistency object store and seemingly powers half the internet. Lots of users choose a single Availability Zone because it's cheap, but because only S3 guarantees 99.9% monthly uptime, which is 43 minutes downtime a month, outages take out half the internet too, which might be a good thing if you're as addicted to Reddit as I am. Great for better for backups and non-critical data, not so great for business data.

How are we doing on the cloud native front? Well, it's scalable and highly consistent and high performance, which is great. But you're locked into Amazon as a cloud provider, which is obviously how they like it, it gets pretty expensive which I'm sure some of you know, and there's privacy issues about moving sensitive data to the cloud.

Public cloud: AWS EBS

6



Now if you're a hipster and you want to join all the cool kids on public cloud, then EBS is a popular option, which stands for Elastic block storage.

EBS is pretty nice, everything is scalable, highly consistent and high performance. On the downside, you have a maximum of 40 EBS instances you can mount per EC2 instance, which limits how many containers you can run per EC2 instance, and mounting physical block devices to nodes takes at least 45 seconds, which is not good for being able to move containers to different hosts.

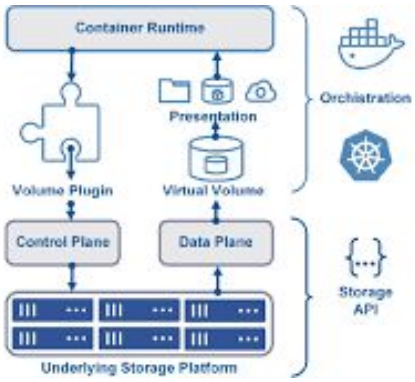
I'll also mention S3 which is Amazon's eventual consistency object store and seemingly powers half the internet. Lots of users choose a single Availability Zone because it's cheap, and because only S3 guarantees 99.9% monthly uptime, which is 43 minutes downtime a month, outages take out half the internet too, which might be a good thing if you're as addicted to Reddit as I am. Great for better for backups and non-critical data, not so great for business data.

How are we doing on the cloud native front? Well, it's scalable and highly consistent and high performance, which is great. But you're locked into Amazon as a cloud provider, which is obviously how they like it, it gets pretty expensive which I'm sure some of you know, and there's privacy issues about moving sensitive data to the cloud.

6/8 By the way, if you want to look at the numbers I’m giving, these slides are on oicheryl.com.

@oicheryl

Volume plugin: StorageOS




```
$ docker volume create
--driver storageos
--opt size=15
--opt storageos.feature.replicas=2
volume-name
```

STORAGEOS

Volume plugins are Docker's way of extending storage capabilities, and StorageOS is an example of a distributed block storage platform which is deployed with Docker.

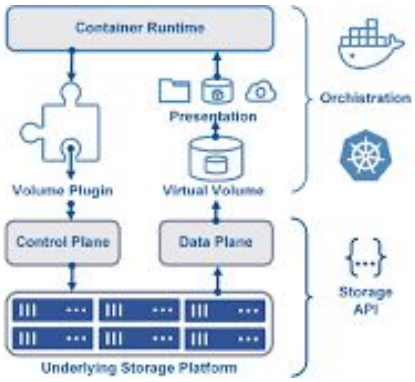
To use StorageOS you could create a docker volume with the storageos driver, set the size is 15 GB and in this case, tell it to create two replicas of the volume on other nodes. This gets you the high availability (if the node with the master volume goes down you can promote one of the replicas to a new master), plus all the volumes are accessible from any node, so if your container goes down you can spin it up anywhere without worrying about which host it's on. But it's not a distributed filesystem, so the StorageOS can schedule the master volume to the same node as the container, meaning your reads are local and fast and deteministic.

Given this was built with those principles in mind, I'm obviously giving it an 8, but there are some downsides; for instance, right now it assumes your cluster is geographically close, so cross availability zone replication would be slow.

@oicheryl

Volume plugin: StorageOS

8




```
$ docker volume create
--driver storageos
--opt size=15
--opt storageos.feature.replicas=2
volume-name
```

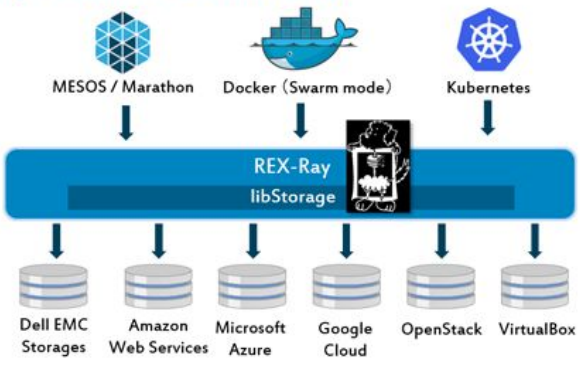
STORAGEOS

Volume plugins are Docker's way of extending storage capabilities, and StorageOS is an example of a distributed block storage platform which is deployed with Docker.

To use StorageOS you could create a docker volume with the storageos driver, set the size is 15 GB and in this case, tell it to create two replicas of the volume on other nodes. This gets you the high availability (if the node with the master volume goes down you can promote one of the replicas to a new master), plus all the volumes are accessible from any node, so if your container goes down you can spin it up anywhere without worrying about which host it's on. But because it's not a distributed filesystem, the StorageOS scheduler can always schedule the master volume to the same node as the container, meaning your reads are local and you get good throughput.

Given this was built with those principles in mind, I'm obviously giving it an 8, but there are some downsides; for instance, right now it assumes your cluster is geographically close, so cross availability zone replication would be slow.

Plugin framework: REX-Ray



I mentioned Dell EMC before as a hardware vendor, they are also the developers of REX-Ray which I want to mention because superficially it looks like another storage plugin.

REX-Ray doesn't provide storage itself; it's a framework which supports a number of different storage systems. Not really a cloud native - it's just a connector to existing storage options. So I'm not going to give it a score.

Of course, you can run lots of these things in combination. You could run StorageOS on EBS, REX-Ray on top of Ceph, or NFS from VMs. There's no one solution, and often it's a bit of trial and expensive (error). But hopefully those eight principles have given you a way to evaluate what you need against what you're currently using.

K8S Storage SIG & CNCF Storage WG: https://github.com/cncf/wg-storage

Objective is to define an industry standard “Container Storage Interface” (CSI) that will enable storage vendors (SP) to develop a plugin once and have it work across a number of container orchestration (CO) systems.

If you’re interested in learning more, standards are continuing to improve and the K8s Storage Special Interest Group and CNCF Storage Working Group are proposing a Container Storage Interface to make it easier to move between storage options.


STORAGEOS
Thanks
Slides at oicheryl.com
© 2013-2017 StorageOS Ltd. All rights reserved.