STORAGEOS

Persistent Storage with Kubernetes in Production
Which solution and why?

Kubecon + CloudNativeCon, Dec 8, 2017

Cheryl Hung, Product Manager

Cheryl

Product manager, StorageOS

CNCF Ambassador

Slides at oicheryl.com

STORAGEOS

- Why is state so tricky?
- How should I compare storage?
- What storage should I use with Kubernetes?

- Why is state so tricky?
- How should I compare storage?
- What storage should I use with Kubernetes?

Anti-objective:

- Should I use a database/message queue/key-value store... for my app?

STORAGEOS

Why is state so tricky?

THERE IS **NO SUCH THING** AS A
"**STATELESS**" ARCHITECTURE
IT'S JUST **SOMEONE ELSE'S PROBLEM**

STORAGEOS

First challenge: No pet storage

Second challenge: Data needs to follow

How should I compare storage?

From the CNCF Landscape
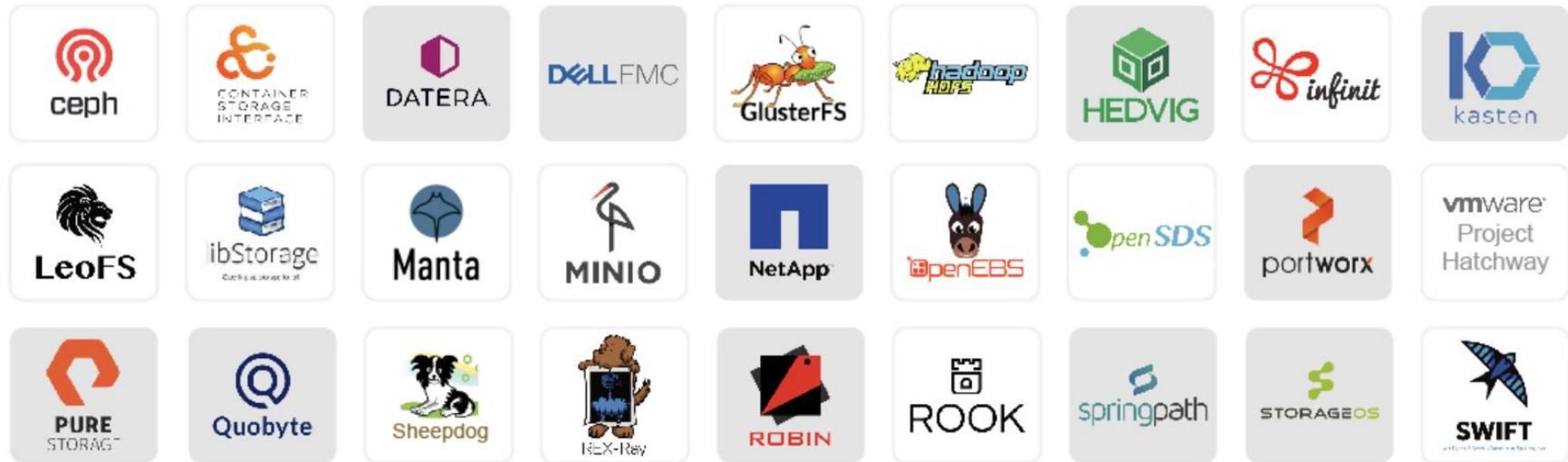
Cloud Storage

	Azure	Google	AWS
Object Storage	Azure Blob Storage	Google Cloud Storage	Amazon Simple Storage Service (S3)
Virtual Machine / Block Storage	Azure Page Blobs / Premium Storage	Persistent Disk	Amazon Elastic Block Storage (EBS)
File Storage	Azure File Storage	🚫	Amazon Elastic File System (EFS)
Long Term Cold Storage	Azure Cool Storage	Google Coldline Storage	Amazon Glacier
Hybrid / Gateway Storage	Azure Storsimple	🚫	AWS Storage Gateway

STORAGEOS

Eight Principles of Cloud Native Storage

Horizontally scalable

No single point of failure

Resilient and self healing

Minimal operator overhead

Decoupled from the underlying platform

STORAGEOS

1 Application centric

Storage should be presented to and consumed by applications, not by operating systems or hypervisors

STORAGEOS

1 Application centric

Storage should be presented to and consumed by applications, not by operating systems or hypervisors

2 Platform agnostic

The storage platform should be able to run anywhere. Upgrades and scaling is non-disruptive.

STORAGEOS

1 Application centric

Storage should be presented to and consumed by applications, not by operating systems or hypervisors

2 Platform agnostic

The storage platform should be able to run anywhere. Upgrades and scaling is non-disruptive.

3 Declarative/ composable

Storage resources should be declared and composed just like all other resources required by applications and services.

STORAGEOS

1 Application centric

Storage should be presented to and consumed by applications, not by operating systems or hypervisors

2 Platform agnostic

The storage platform should be able to run anywhere. Upgrades and scaling is non-disruptive.

3 Declarative/ composable

Storage resources should be declared and composed just like all other resources required by applications and services.

4 API driven

Storage resources and services should be easy to be provisioned, consumed, moved and managed via an API.

STORAGEOS

Storage services should integrate and inline security features such as encryption and RBAC.

5 Natively secure

Storage services should integrate and inline security features such as encryption and RBAC.

The platform should be able to move application data between locations, dynamically resize and snapshot volumes.

5 Natively secure

6 Agile

STORAGEOS

@oicheryl

Storage services should integrate and inline security features such as encryption and RBAC.

The platform should be able to move application data between locations, dynamically resize and snapshot volumes.

The storage platform should offer deterministic performance in complex distributed environments.

5 Natively secure

6 Agile

7 Performant

STORAGEOS

Storage services should integrate and inline security features such as encryption and RBAC.

The platform should be able to move application data between locations, dynamically resize and snapshot volumes.

The storage platform should offer deterministic performance in complex distributed environments.

The storage platform should ensure high availability, durability, consistency with a predictable, proven data model.
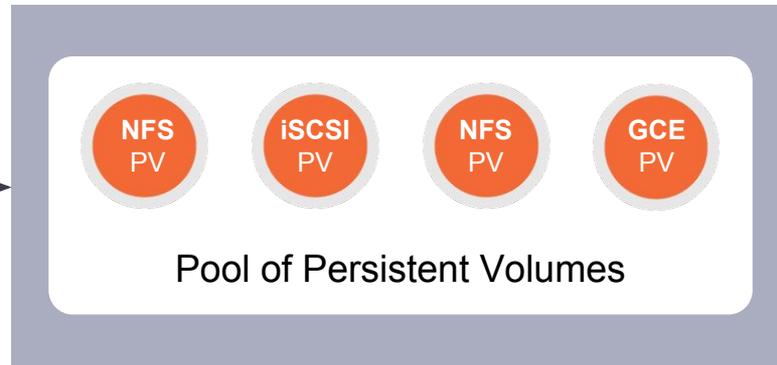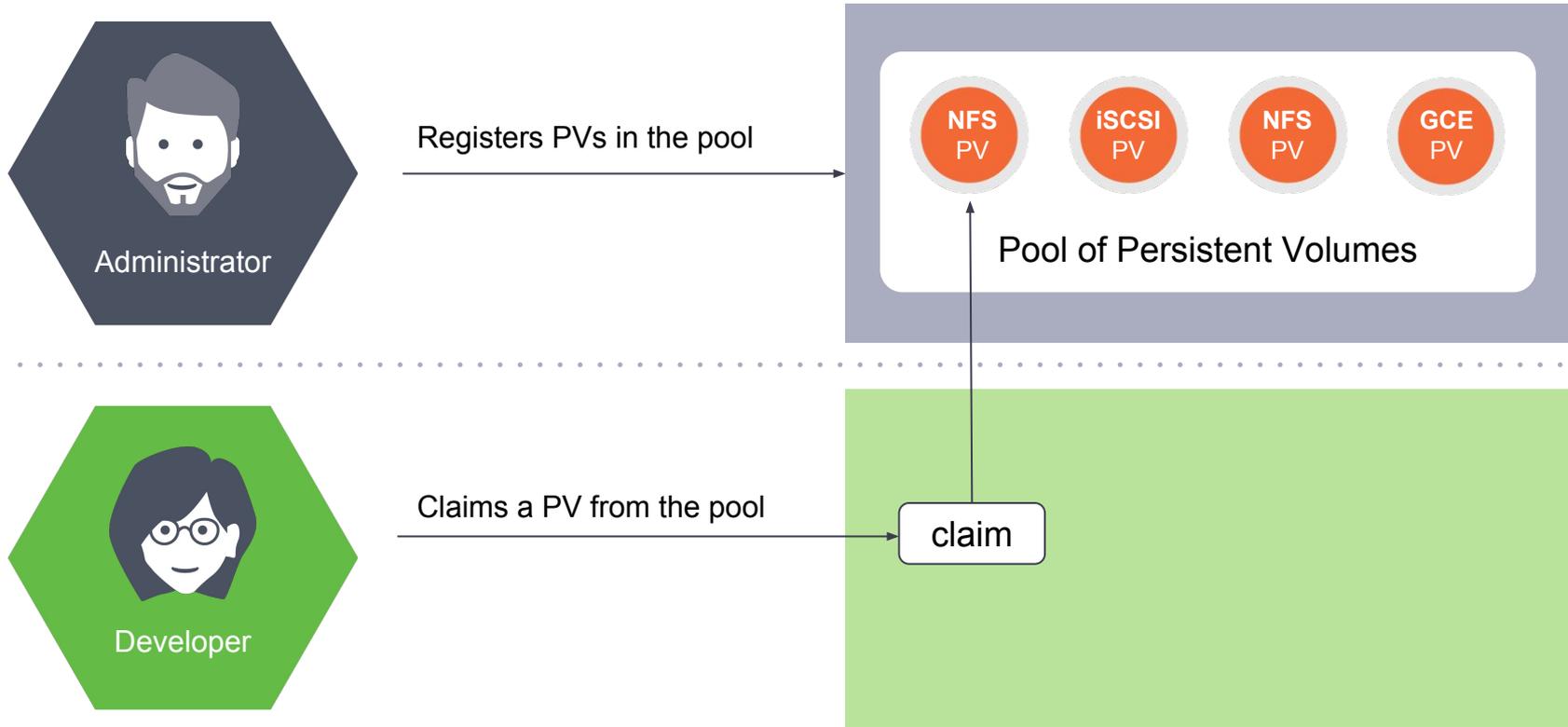
5 Natively secure

6 Agile

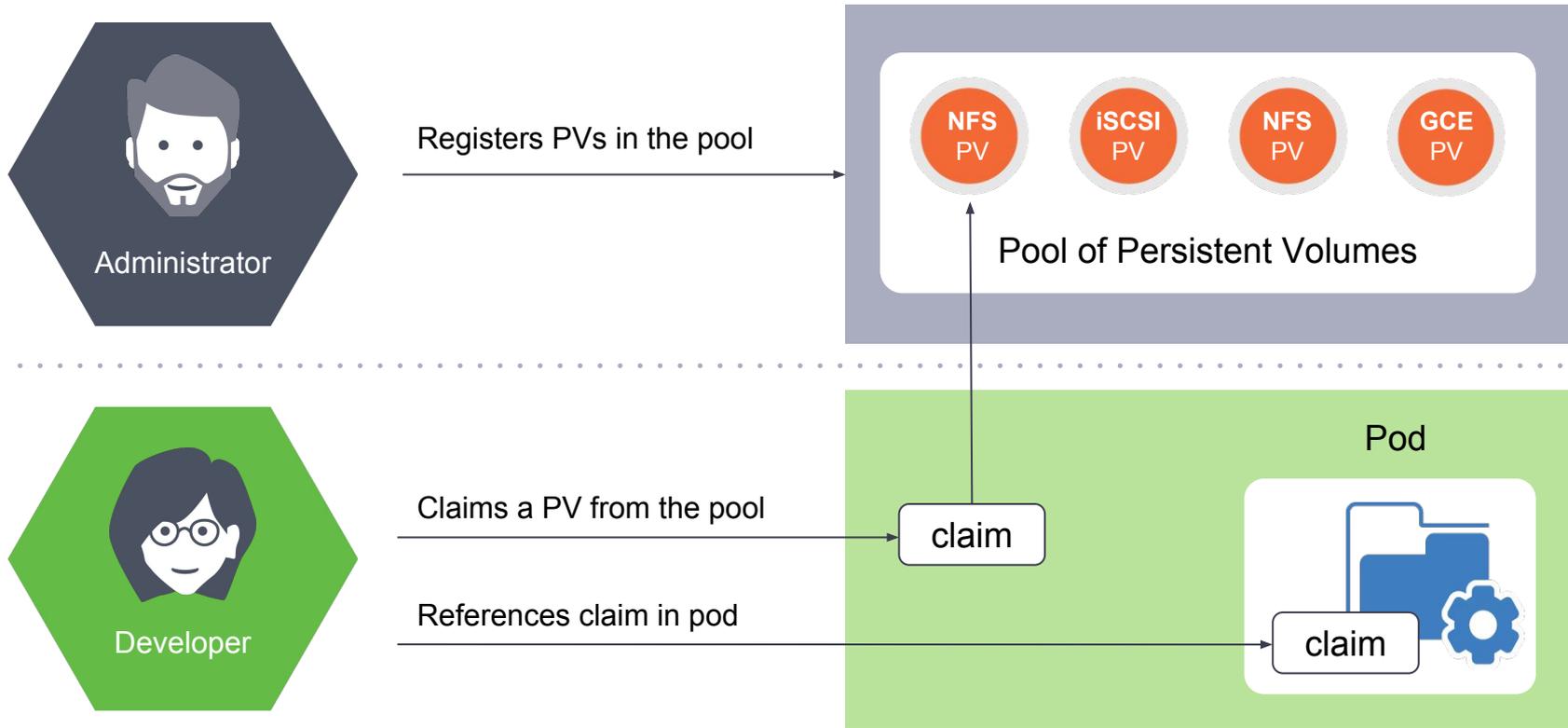**7 Performant**

8 Consistently available

STORAGEOS

STORAGEOS

What storage should I use with Kubernetes?

Dynamic provisioning with Storage Classes

@oicheryl

Administrator

Registers Storage Classes

fast

slow

Storage Classes

fast PV

Pod

Developer

Claims a PV from the pool

claim

References claim in pod

claim

© StorageOS Ltd.

STORAGEOS

Meet Jane
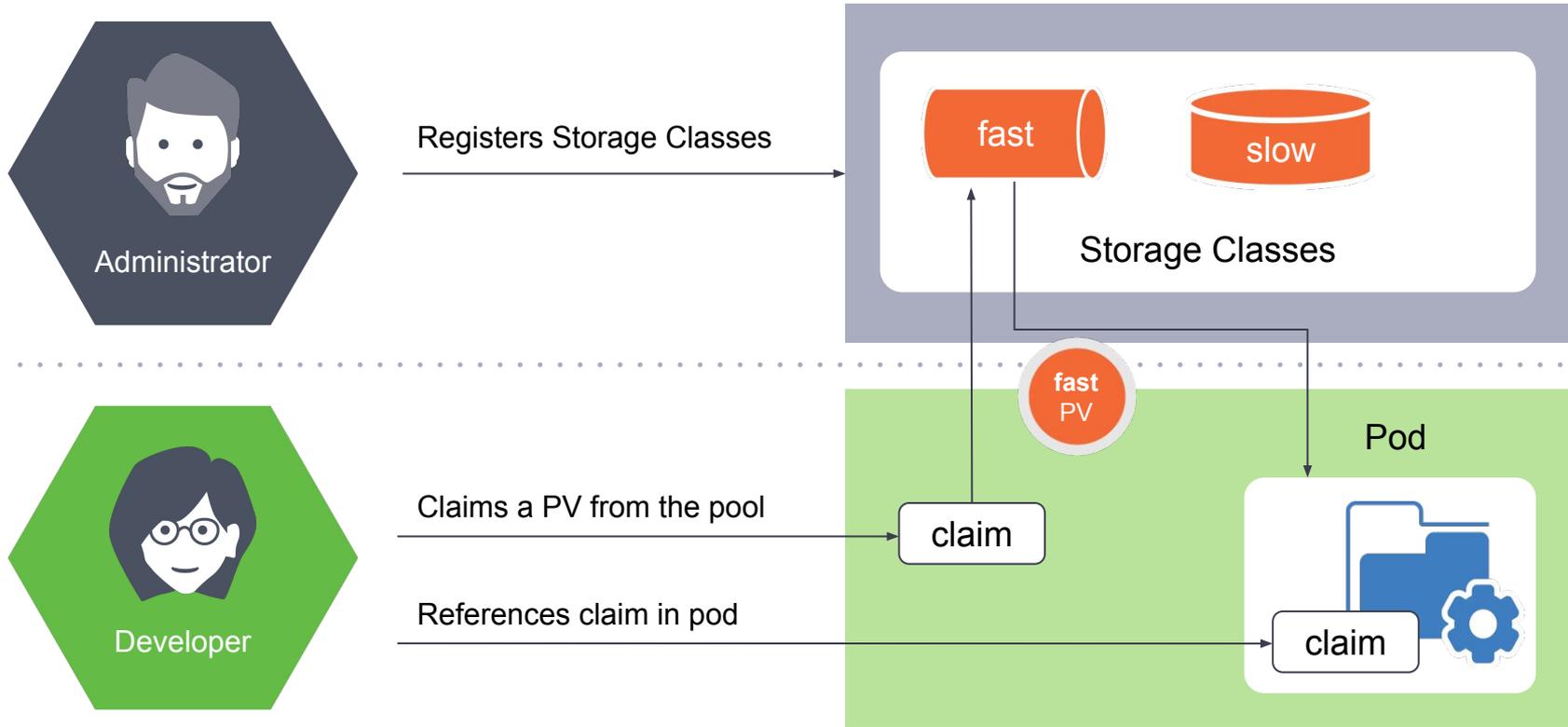
- A DevOps engineer at a media company
- Migrating client Wordpress websites into Kubernetes
- Wants to follow the cloud native principles

STORAGEOS

Proliferating plugins

Volume Plugin	Internal Provisioner	Config Example
AWSElasticBlockStore	✓	AWS
AzureFile	✓	Azure File
AzureDisk	✓	Azure Disk
CephFS	-	-
Cinder	✓	OpenStack Cinder
FC	-	-
FlexVolume	-	-
Flocker	✓	-
GCEPersistentDisk	✓	GCE
Glusterfs	✓	Glusterfs
iSCSI	-	-
PhotonPersistentDisk	✓	-
Quobyte	✓	Quobyte
NFS	-	-
RBD	✓	Ceph RBD
VsphereVolume	✓	vSphere
PortworxVolume	✓	Portworx Volume
ScaleIO	✓	ScaleIO
StorageOS	✓	StorageOS

1. What is my **use case**?

2. What are my **performance requirements**?

3. How should developers **access** storage?
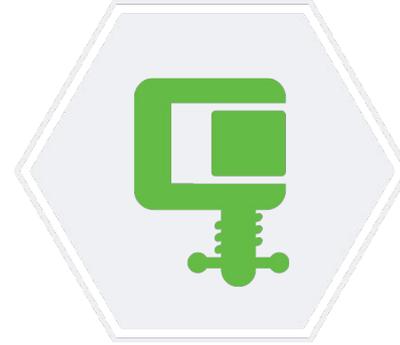
4. Where is the storage **deployed and managed**?

STORAGEOS

App Binaries   **App data**   **Config**   **Backup**

App Binaries
Ephemeral

App data
Latency,
availability,
performant
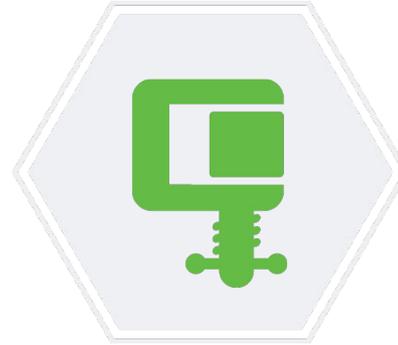
Config
Shared

Backup
Cost efficient,
cloud
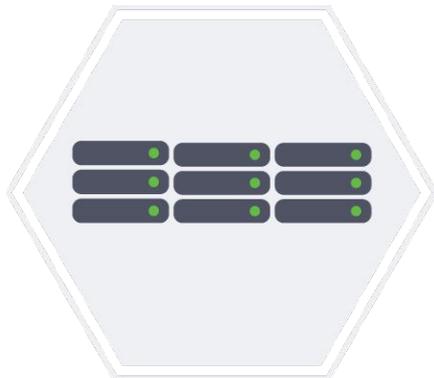
Block
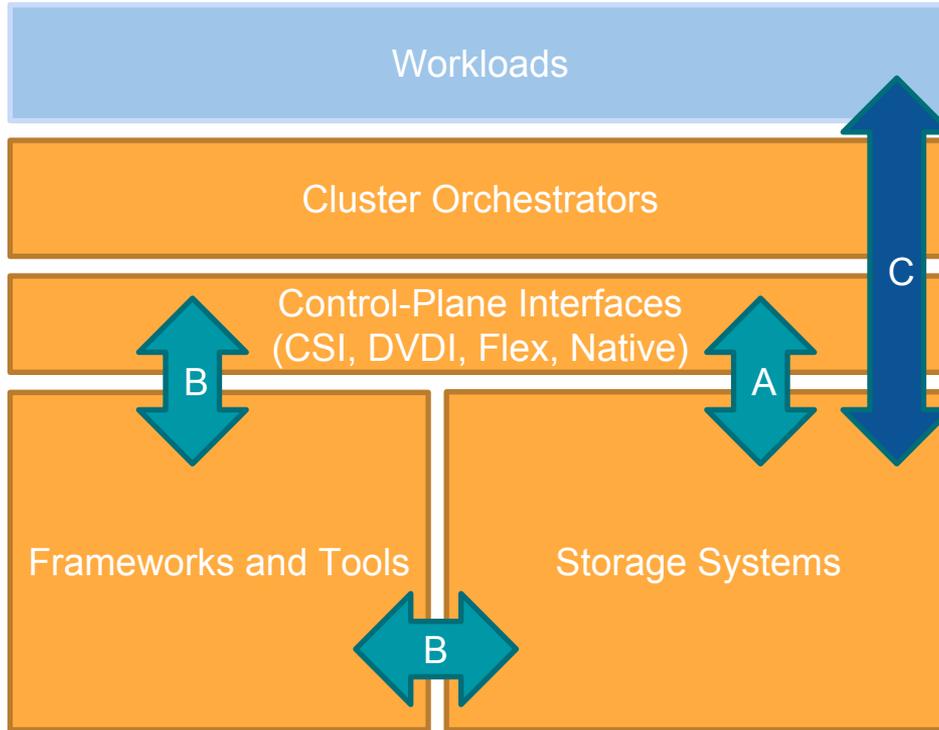Fixed-size 'blocks' in a rigid arrangement – ideal for enterprise databases

File
'Files' in hierarchically nested 'folders' – ideal for active documents

Object
'Objects' in scalable 'buckets' – ideal for unstructured big data and archiving

●CO supports one or more **Interfaces** to interact with the Storage System

●Storage System can **(A)** support control-plane interface API directly and interact directly with the CO or can **(B)** interact with the CO via an **API framework layer** or other **Tools**.

●Storage system must support the ability to provision and consume (C) volumes through a standard interface to be considered **Interoperable**

●Workloads interact (C) with storage systems over various data-plane methods

STORAGEOS

Jane's storage requirements

- Database location, credentials
- Postgres database for application data
- User uploaded media
- Database and website backups

STORAGE**OS**

1. **Use case?** Configuration
2. **Performance requirements?** Shared across instances
3. **Access?** Kubernetes provides Secrets for sensitive data such as passwords, and ConfigMap for arbitrary config. Both can be accessed by the application through environment variables
4. **Deployed and managed?** Tight integration with Kubernetes

1. **Use case?** Shared media
2. **Performance requirements?** Large blobs of data, shared across pods
3. **Access?** Shared filesystem
4. **Deployed and managed**?

Cloud: Managed NFS, or object store if the app can support it

On prem: Distributed FS (but please not NFS!)

STORAGEOS

1. **Use case?** Backup and archival
2. **Performance requirements?** Durability, cost, snapshots
3. **Access?** Object store
4. **Deployed and managed?**

Cloud: Managed object store, long term cold storage

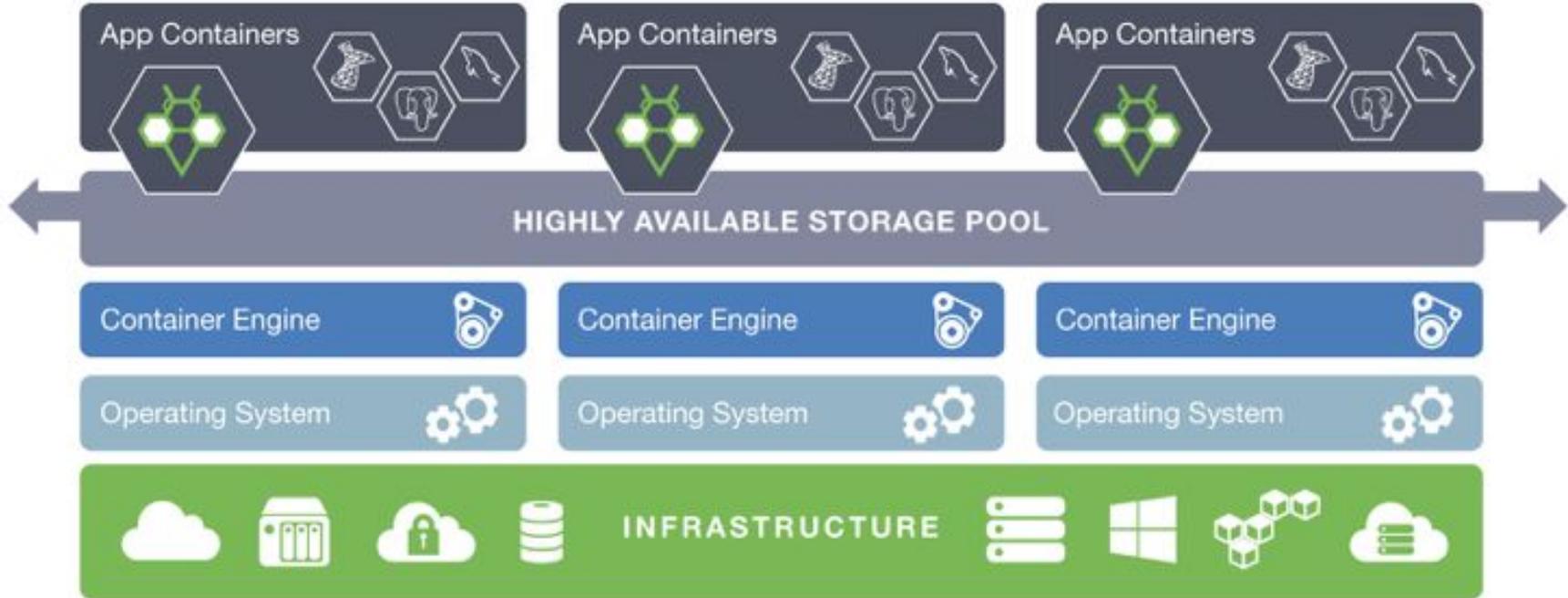**On prem**: Object store, NAS

STORAGEOS

1. **Use case?** Transactional database
2. **Performance requirements?** High availability, low latency, deterministic performance
3. **Access?** Database connector
4. **Deployed and managed?**

Cloud: Cloud volumes (watch out for attach/detach times, compliance) or managed db (limited offerings)

On prem: Software defined storage

STORAGEOS

To Recap…

1 Application centric

5 Natively secure

2 Platform agnostic

6 Agile

3 Declarative/ composable

7 Performant

4 API driven

8 Consistently available

1. **Use case?**
2. **Performance requirements?**
3. **Access?**
4. **Deployed and managed?**

STORAGEOS

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

STORAGEOS

Browser-based demo

- my.storageos.com/main/tutorials

Quickstart

- storageos.com/kubernetes

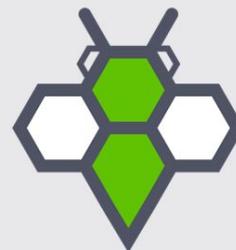We're hiring! London and NYC roles

- storageos.com/careers

A software-defined, scale-out storage platform for running enterprise containerized applications in production
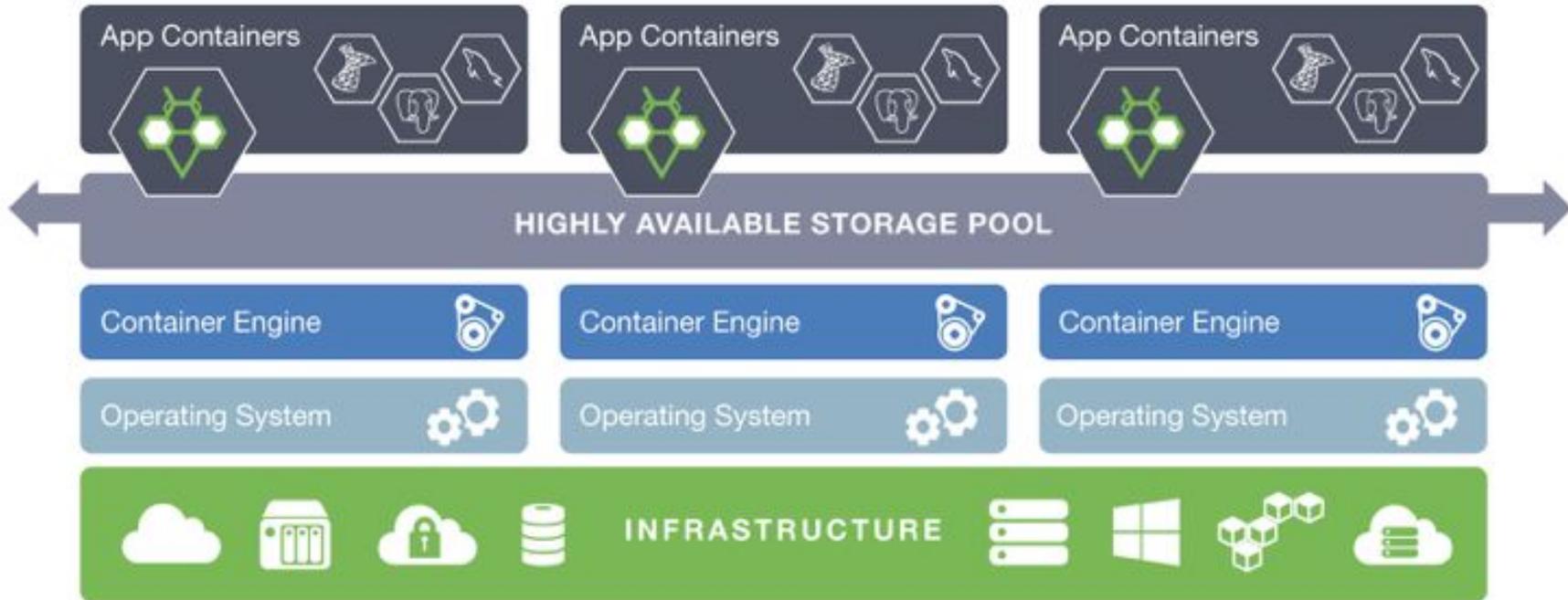
Platform agnostic

Horizontally scalable

Database (ie. block)

A software-defined, scale-out storage platform for running enterprise containerized applications in production

Docker/K8s integration

High availability
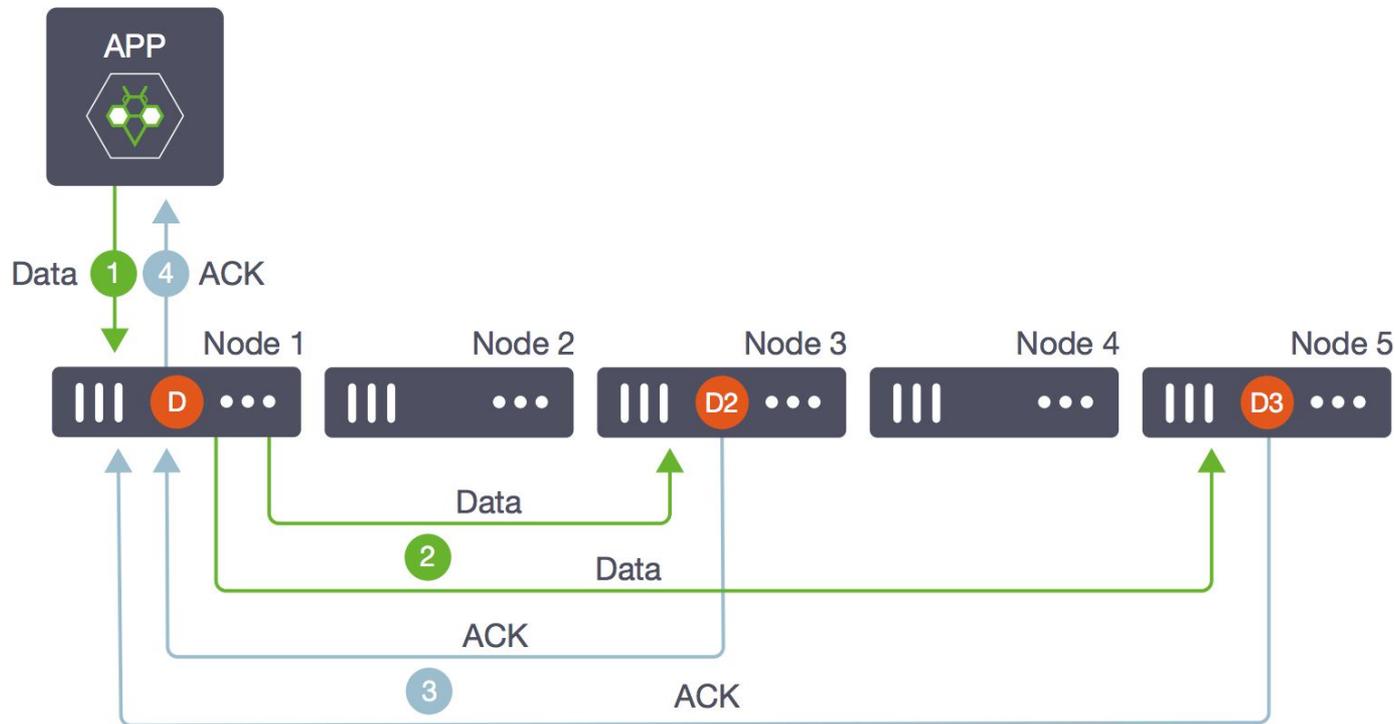
StorageOS architecture

StorageOS is conceptually pretty simple; it's a virtualization layer on top of any commodity or cloud storage. It's deployed as one [container](#) per node, similar to a DaemonSet.

1. Nodes contribute local block storage to the storage pool.
2. Virtual volumes (block storage formatted with a standard filesystem) are created using the [StorageOS volume plugin](#).
3. Any pods can mount the virtual volumes from any node. If a pod is rescheduled to a different node, StorageOS simply redirects reads and writes so the pod can continue to access the storage.

STORAGE**OS**

It's designed to scale horizontally by adding more nodes. New nodes contribute their storage into the storage pool, or, if they don't have storage themselves, can access storage on other nodes.

High availability with StorageOS

APP

Data **1** **4** ACK

Node 1 Node 2 Node 3 Node 4 Node 5

D D2 D3

Data

2

Data

ACK

3

ACK

STORAGEOS

StorageOS uses a hybrid master/replica architecture, where replicas are distributed across nodes.

Replication is very simple in StorageOS. Volume D is created with two replicas. StorageOS creates the replicas (D2, D3) and schedules them to two different nodes (N3, N5). Incoming writes to D are synchronously replicated to D2 and D3, ie. writes are not persisted until acknowledged by both replicas.

If N1 fails, one of D2 or D3 gets promoted to master, providing instant failover and no interruption of service. StorageOS creates and resyncs a new replica on N2 or N4 in the background.

STORAGEOS

More reading

Download the technical architecture overview at
storageos.com/storageos-platform-architecture-overview.

Try out in your browser, with zero downloads or configuration:
my.storageos.com/main/tutorials

Full documentation at docs.storageos.com.