



# Persistent Storage with Kubernetes in Production

## *Which solution and why?*

CNCF Paris, 16 January 2018

Cheryl Hung, Product Manager



# Cheryl

Product manager, StorageOS

@oicheryl

Cloud Native London



- 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?





# Why is state so tricky?



# Why do I need storage?

@oicheryl





## First challenge: No pet storage

@oicheryl





## Second challenge: Data needs to follow

@oicheryl



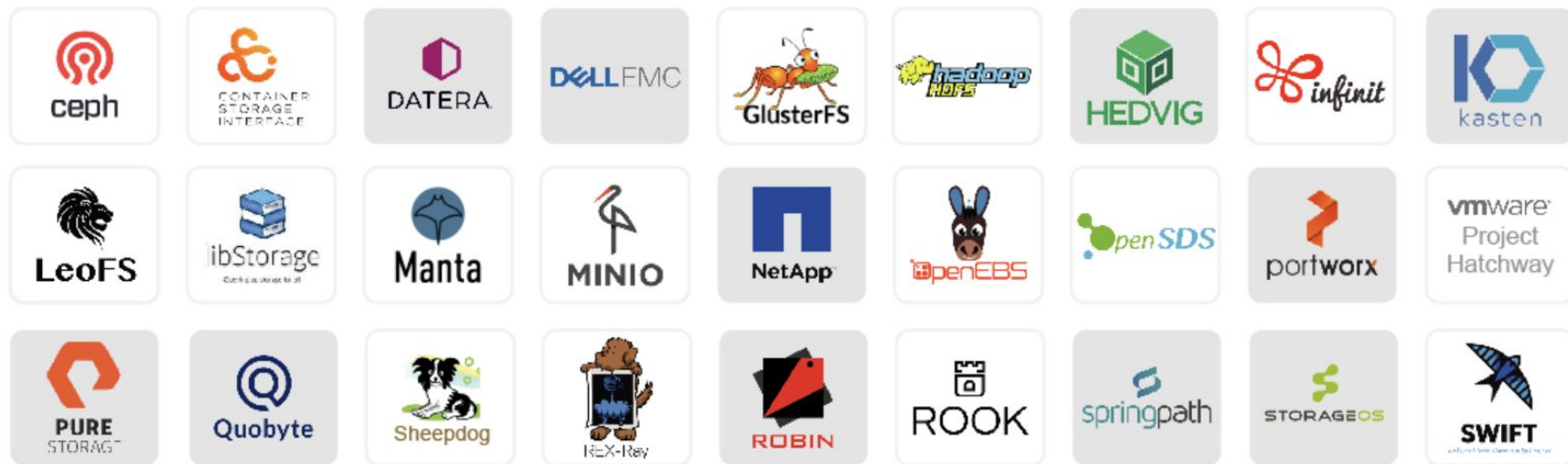
## Third challenge: Humans are fallible




@oicheryl



# How should I compare storage?

## Cloud-Native Storage



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

# 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



## 1 Application centric

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





## 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.



## 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.



## 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.

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**



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**

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**

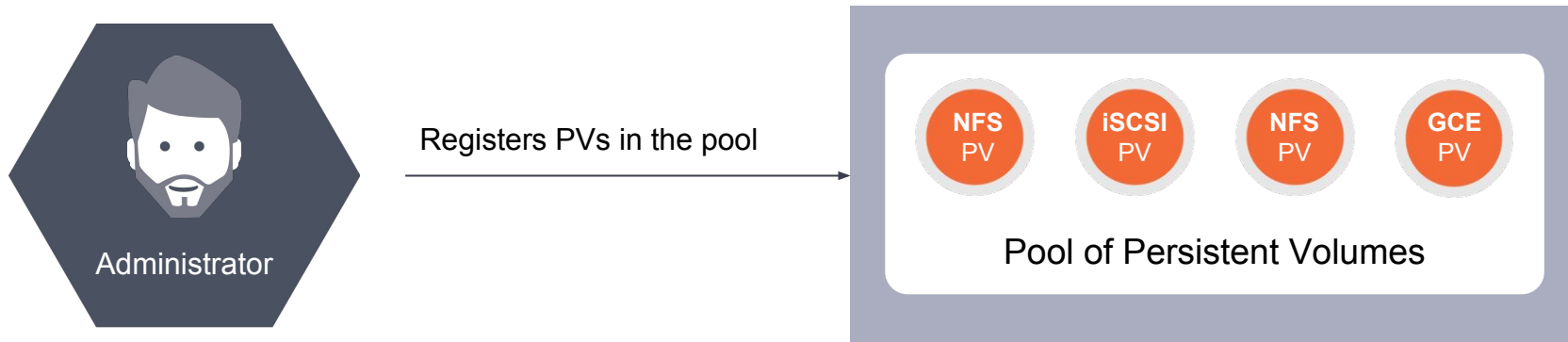
**8 Consistently  
available**

# What storage should I use with Kubernetes?



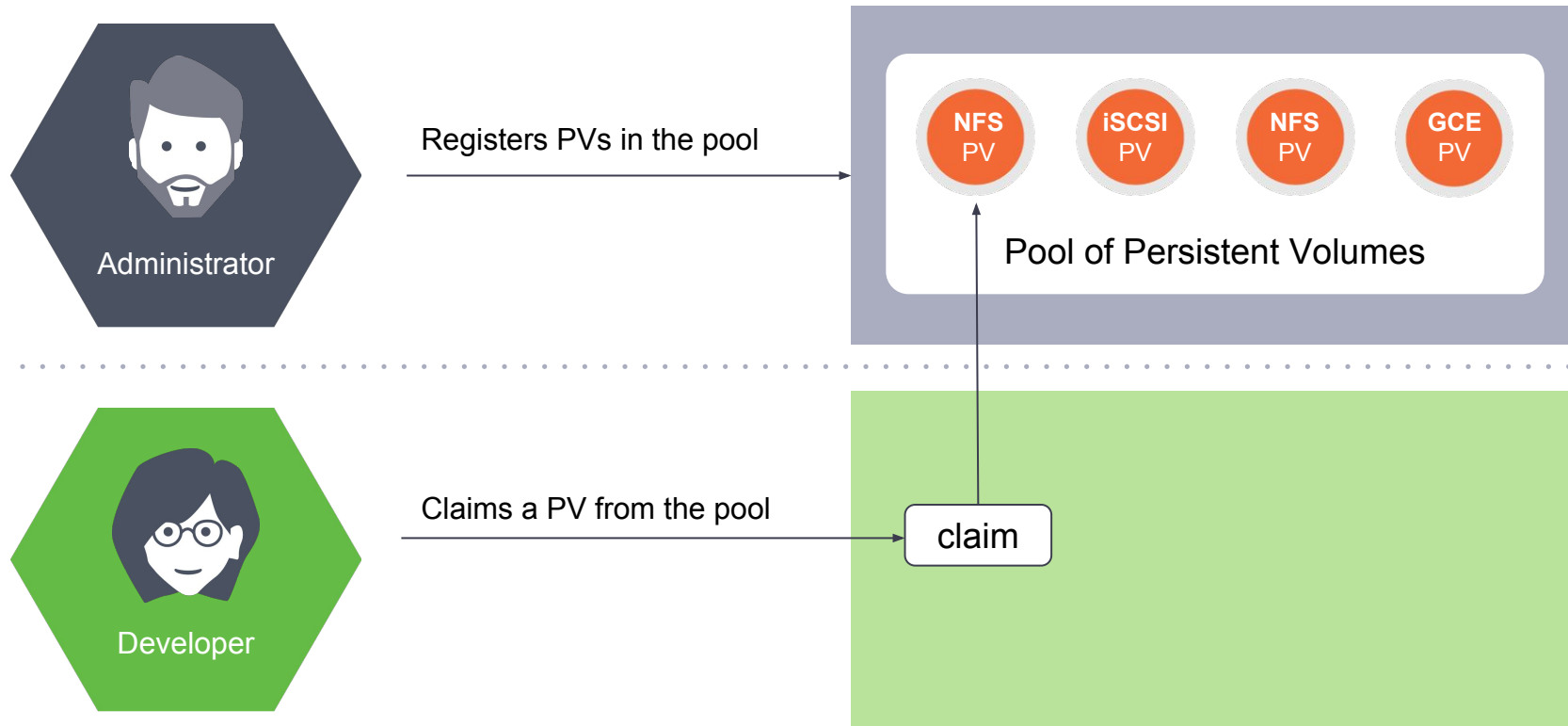
# Kubernetes Storage Model: Persistent Volumes and Claims

@oicheryl



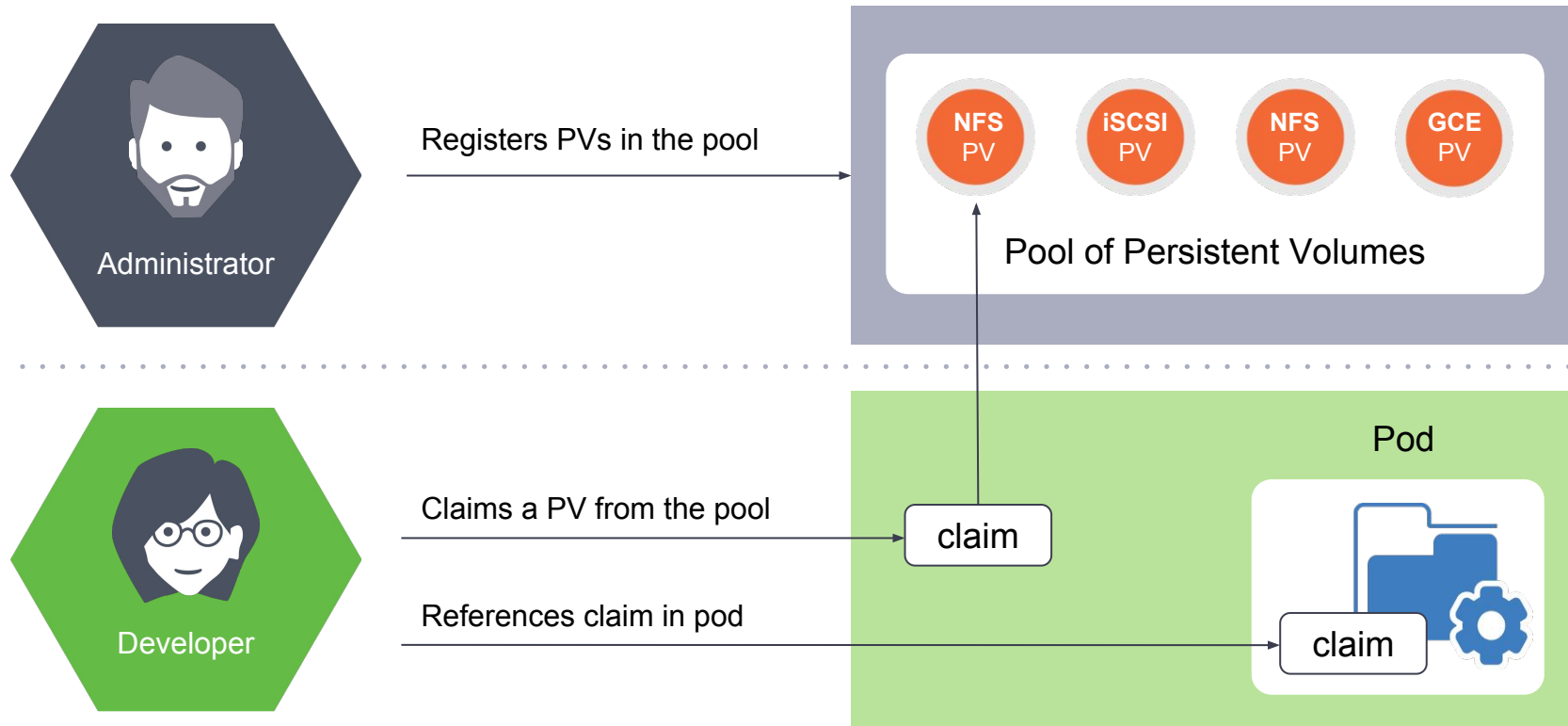
# Kubernetes Storage Model: Persistent Volumes and Claims

@oicheryl



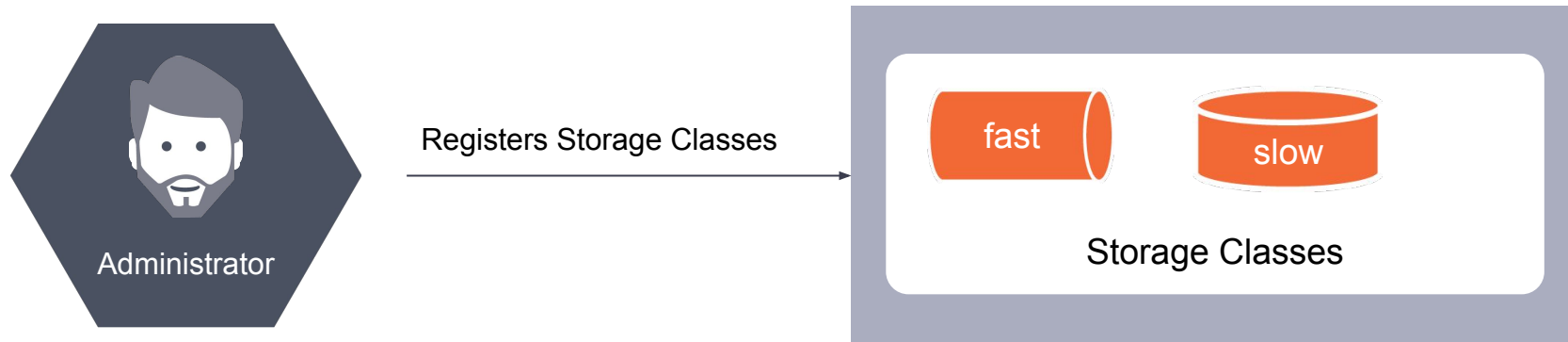
# Kubernetes Storage Model: Persistent Volumes and Claims

@oicheryl



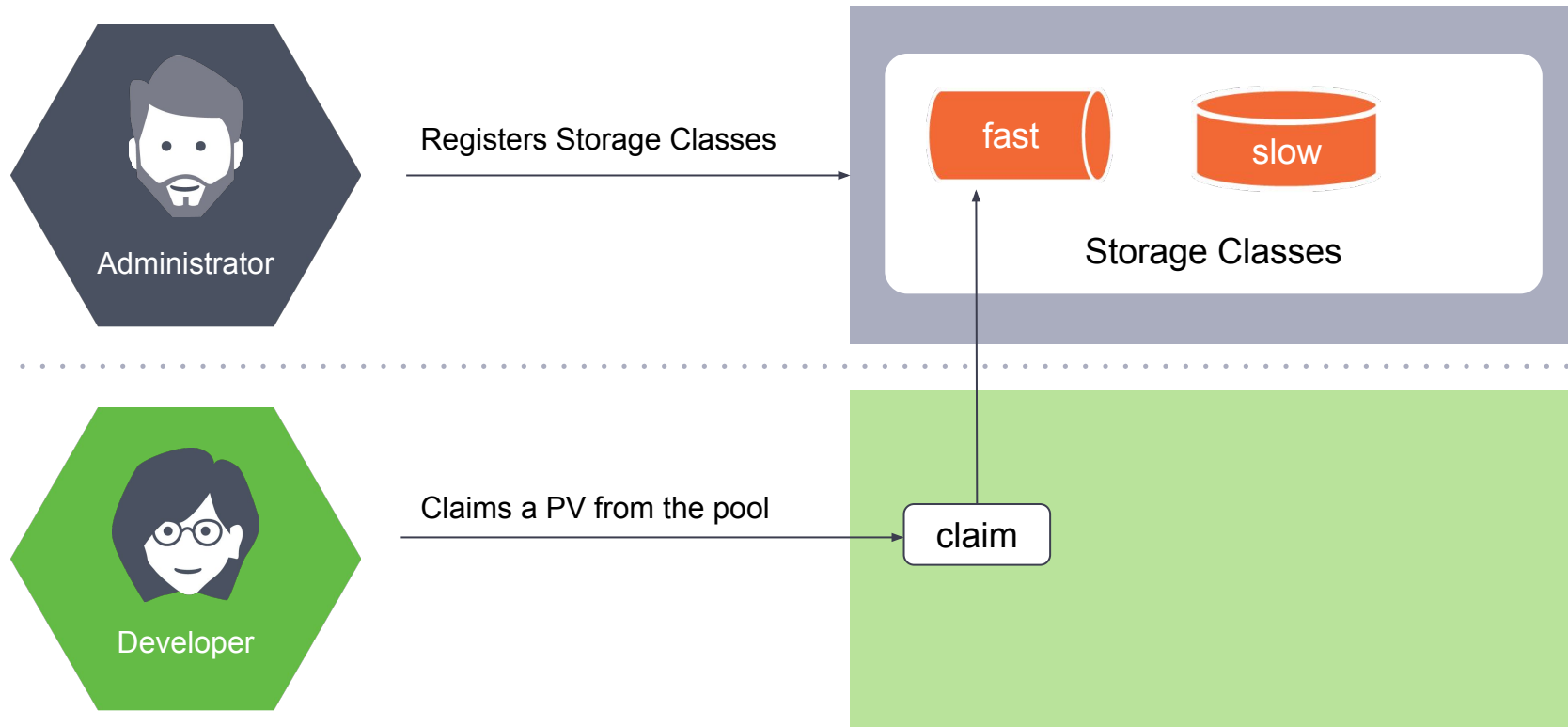
# Dynamic provisioning with Storage Classes

@oicheryl



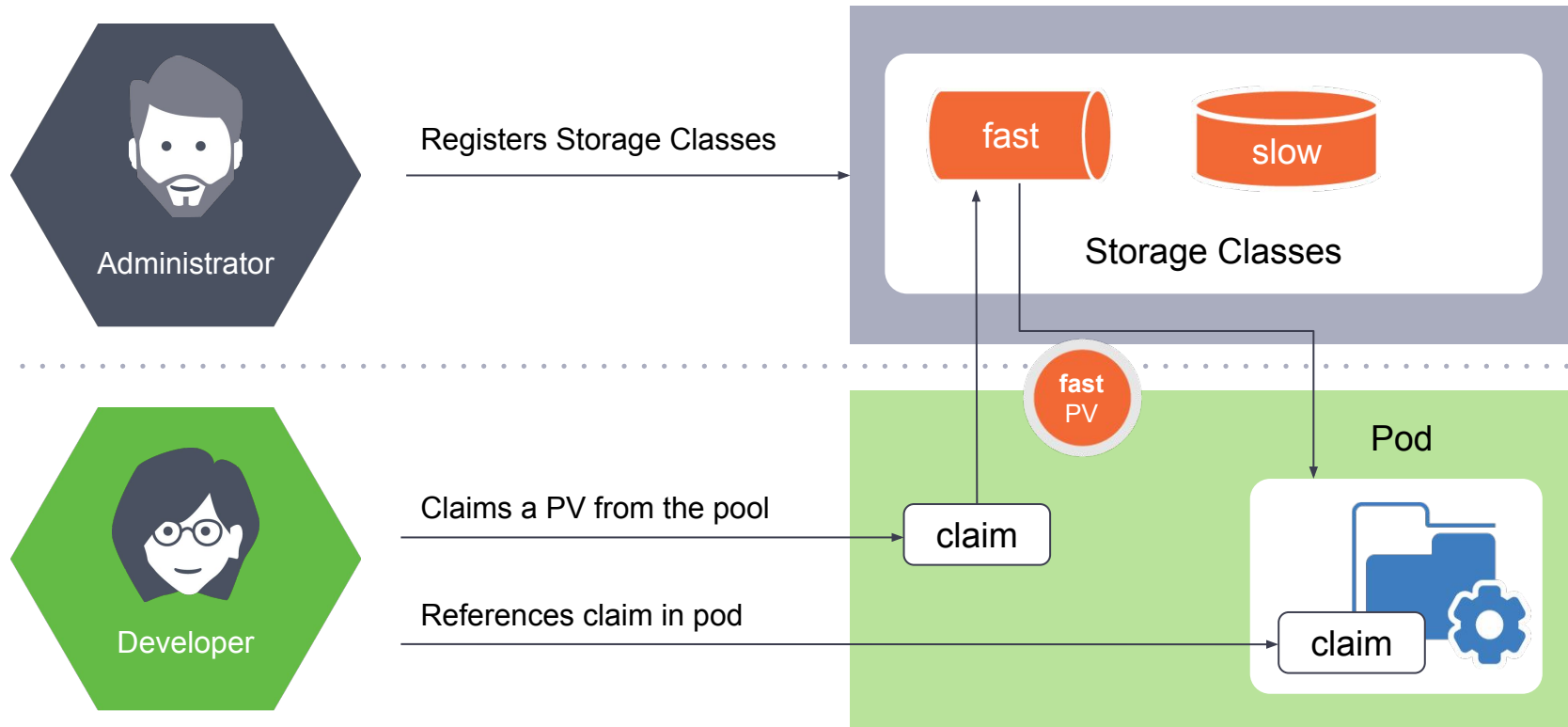
# Dynamic provisioning with Storage Classes

@oicheryl



# Dynamic provisioning with Storage Classes

@oicheryl





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

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



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**?

# 1. What is my use case?

@oicheryl



**App binaries**



**App data**



**Config**



**Backup**

## 2. What are my performance requirements?

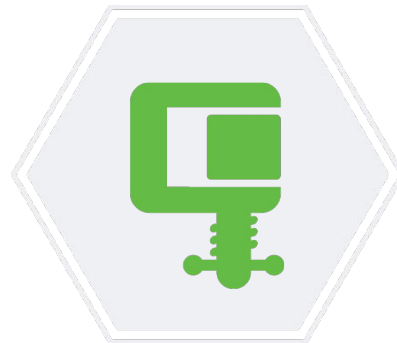
@oicheryl



**App binaries**  
Ephemeral



**App data**  
Latency,  
availability,  
performant



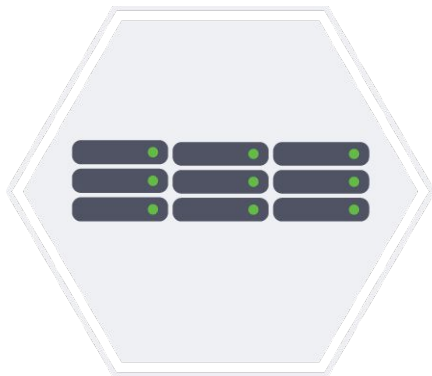
**Config**  
Shared



**Backup**  
Cost efficient,  
cloud

### 3. How should developers access storage?

@oicheryl



#### **Block**

Fixed-size 'blocks' in a rigid arrangement – ideal for enterprise databases



#### **File**

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

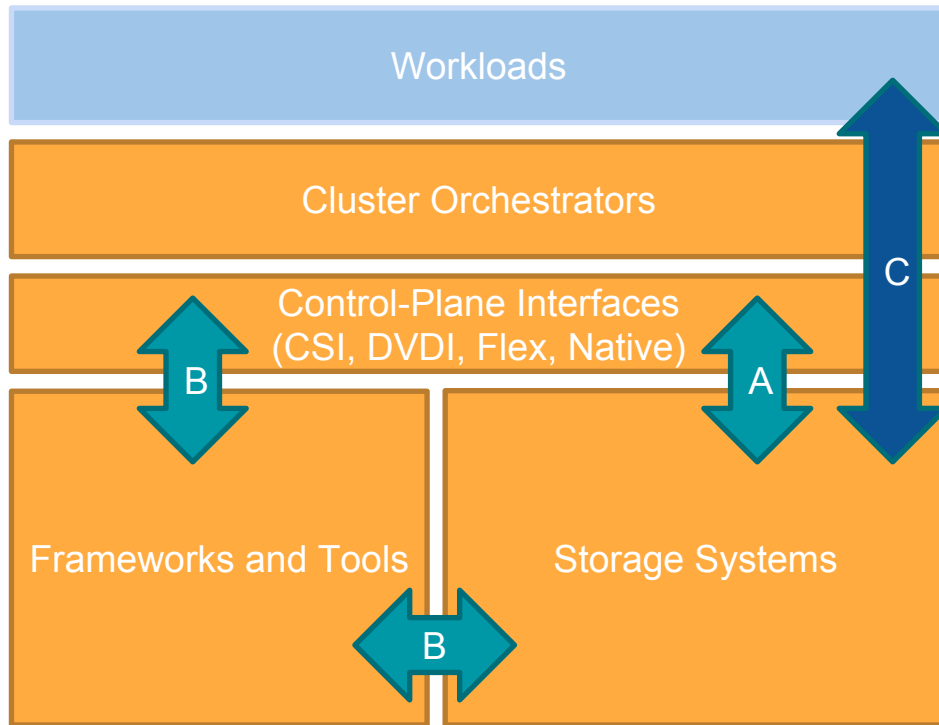


#### **Object**

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

## 4. Where is the storage deployed and managed?

@oicheryl



- 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



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

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 (not NFS)



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

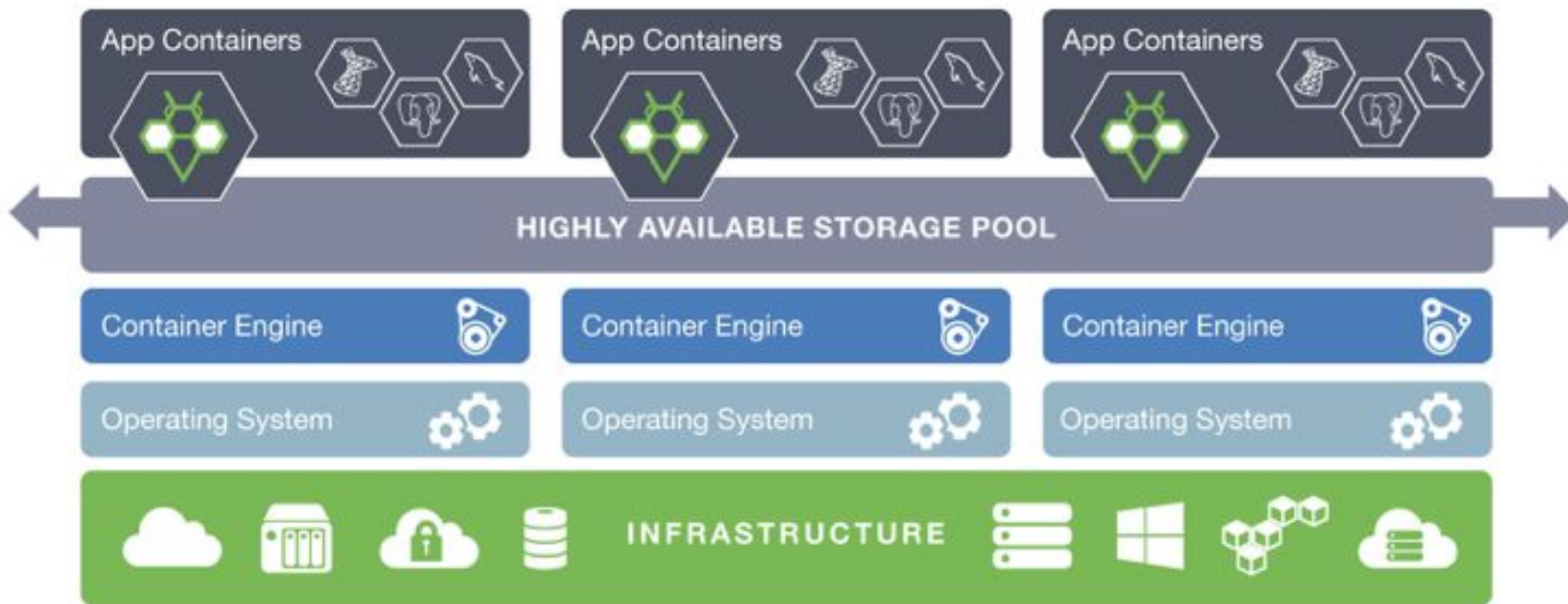
**Cloud:** Managed object store, long term cold storage

**On prem:** Object store, NAS

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

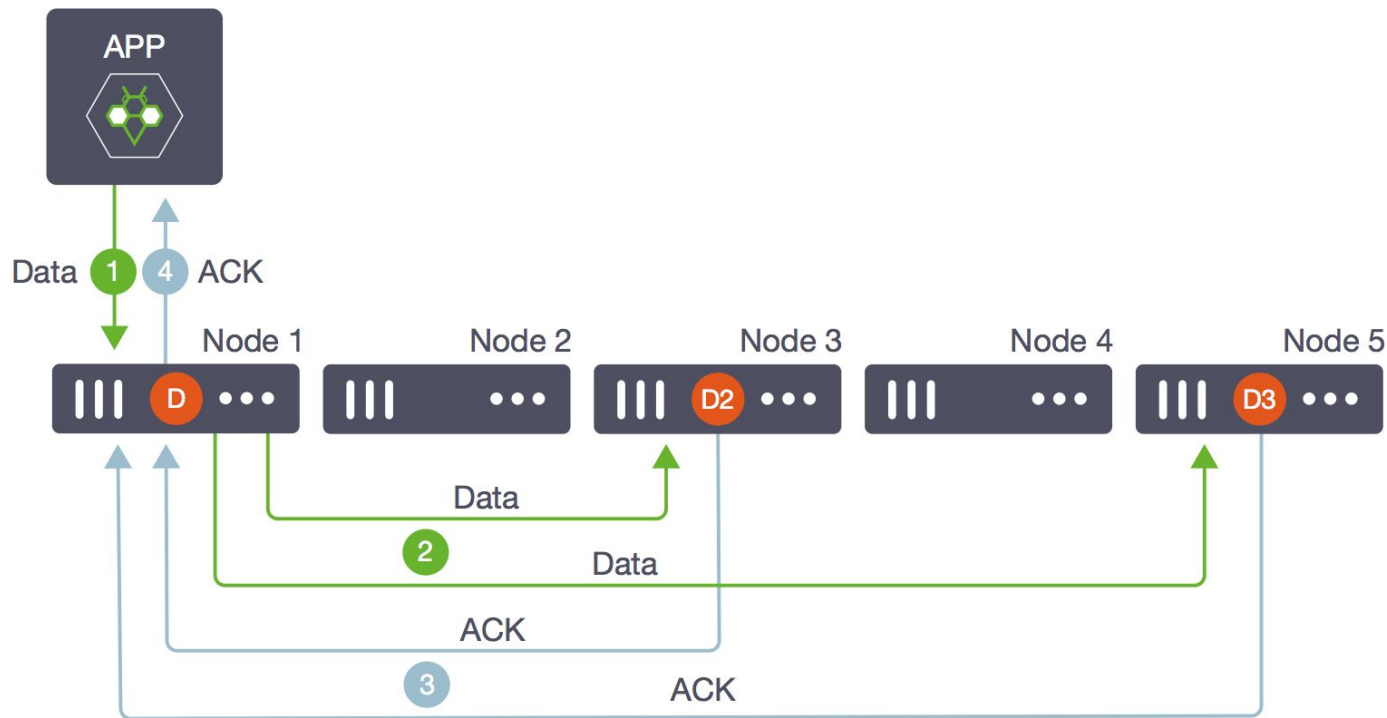




**KEEP  
CALM  
IT IS  
DEMO  
TIME**

# High availability with StorageOS

@oicheryl



# 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?**





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.



## Browser-based demo

- [my.storageos.com/main/tutorials](https://my.storageos.com/main/tutorials)

## Quickstart

- [storageos.com/kubernetes](https://storageos.com/kubernetes)





# Thanks

## Questions?



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



# What is StorageOS?

@oicheryl

Platform  
agnostic

Horizontally  
scalable

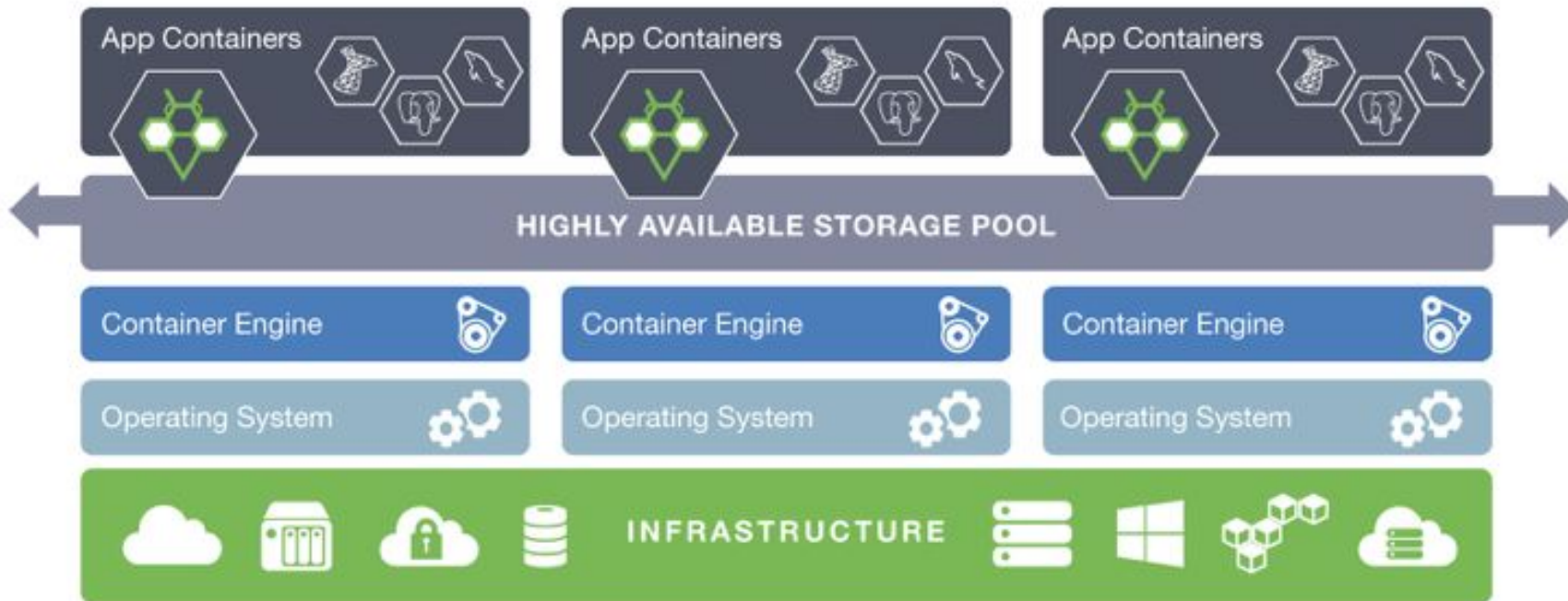
Database (ie.  
block)

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

Docker/K8s  
integration

High  
availability





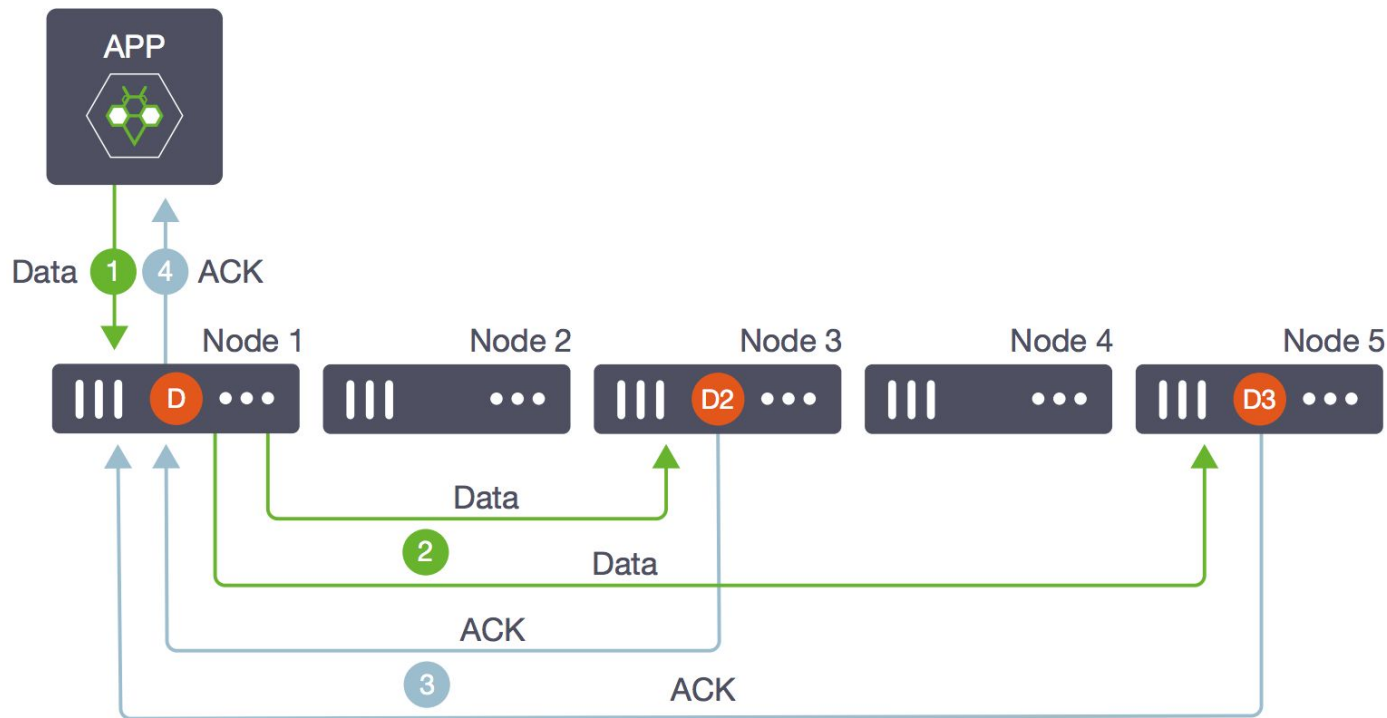
StorageOS is conceptually pretty simple; it's a virtualization layer on top of any commodity or cloud storage. It's deployed as one [container](#) per node, similar to a DaemonSet.

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.

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

@oicheryl





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.

## More reading

Download the technical architecture overview at [storageos.com/storageos-platform-architecture-overview](https://storageos.com/storageos-platform-architecture-overview).

Try out in your browser, with zero downloads or configuration: [my.storageos.com/main/tutorials](https://my.storageos.com/main/tutorials)

Full documentation at [docs.storageos.com](https://docs.storageos.com).