Persistent Storage with Kubernetes in Production

Which solution and why?

Kubecon + CloudNativeCon, Dec 8, 2017

Cheryl Hung, Product Manager

Cheryl

Product manager, StorageOS

@oicheryl

Slides at oicheryl.com

Objectives

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

Objectives

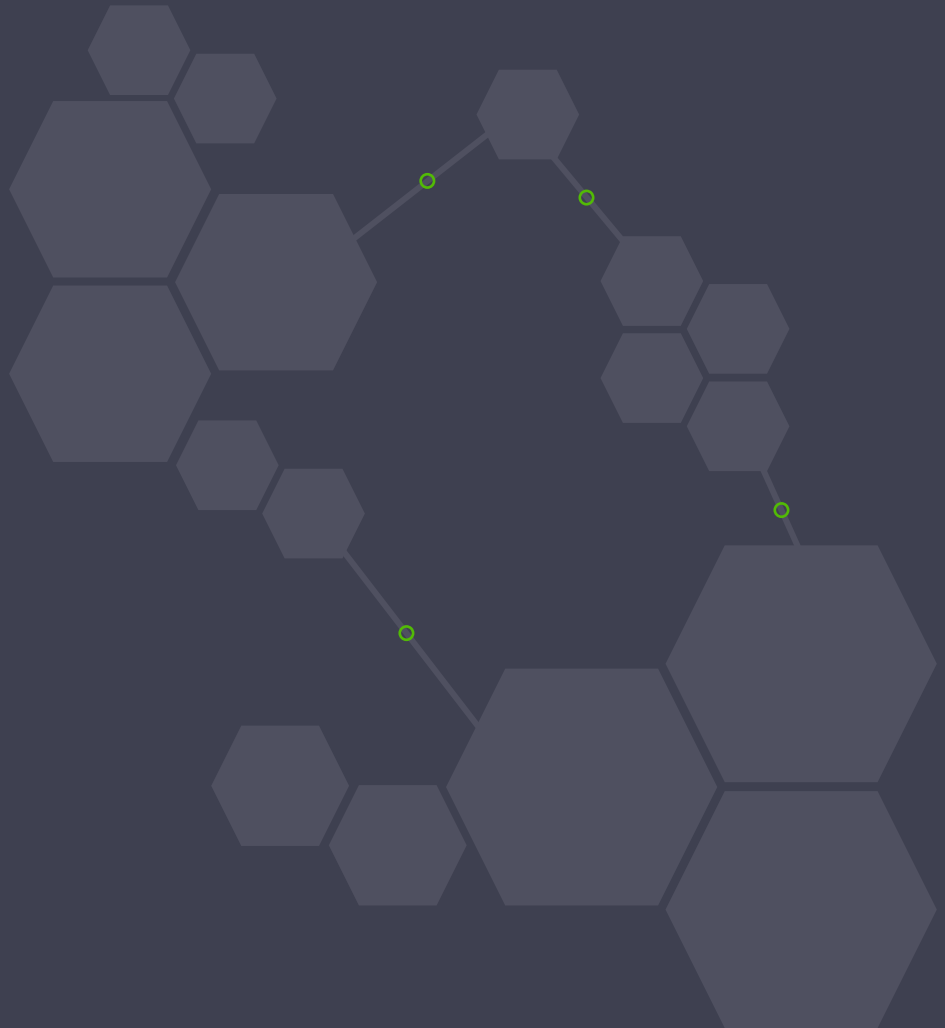
- Why is state so tricky?
- How should I compare storage?
- What storage should I use with Kubernetes?

Anti-objective:

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

STORAGEOS

Why is state so tricky?



Why do I need storage?

Why do I need storage?

First challenge: No pet storage

Second challenge: Data needs to follow

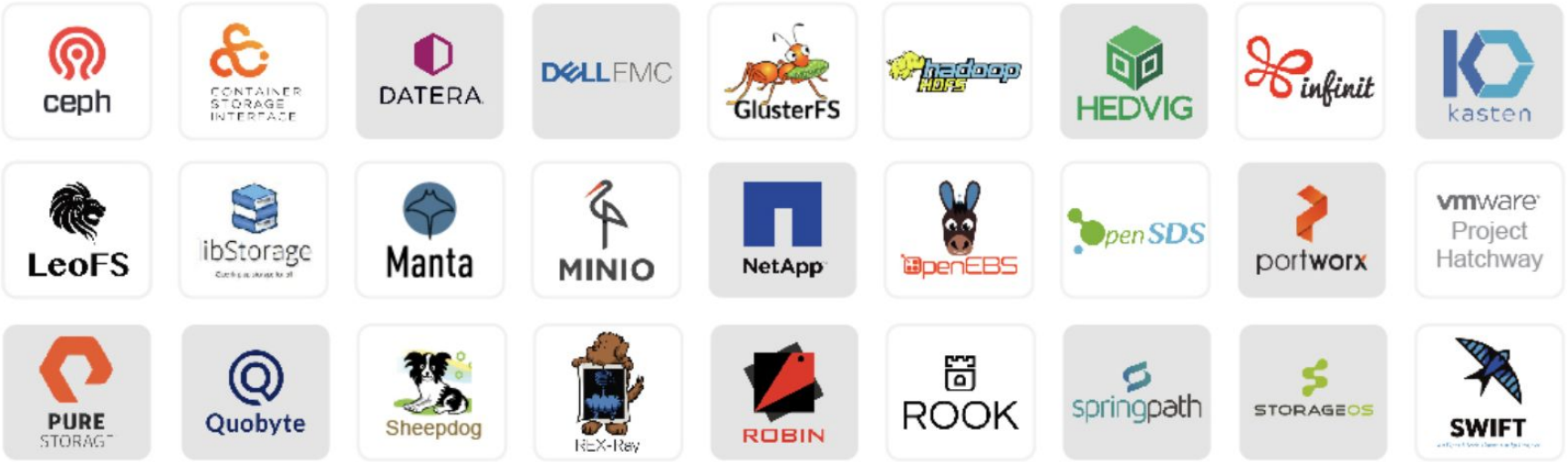
Third challenge: Humans are fallible

STORAGEOS

How should I compare storage?

From the CNCF Landscape

Cloud-Native Storage



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

Eight Principles of Cloud Native Storage

What is Cloud Native?

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

Eight principles of Cloud Native Storage

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.

Eight principles of Cloud Native Storage

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.

Eight principles of Cloud Native Storage

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.

Eight principles of Cloud Native Storage

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

5 Natively secure

Eight principles of Cloud Native Storage

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

5 Natively secure

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

6 Agile

Eight principles of Cloud Native Storage

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

5 Natively secure

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

6 Agile

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

7 Performant

Eight principles of Cloud Native Storage

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

5 Natively secure

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

6 Agile

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

7 Performant

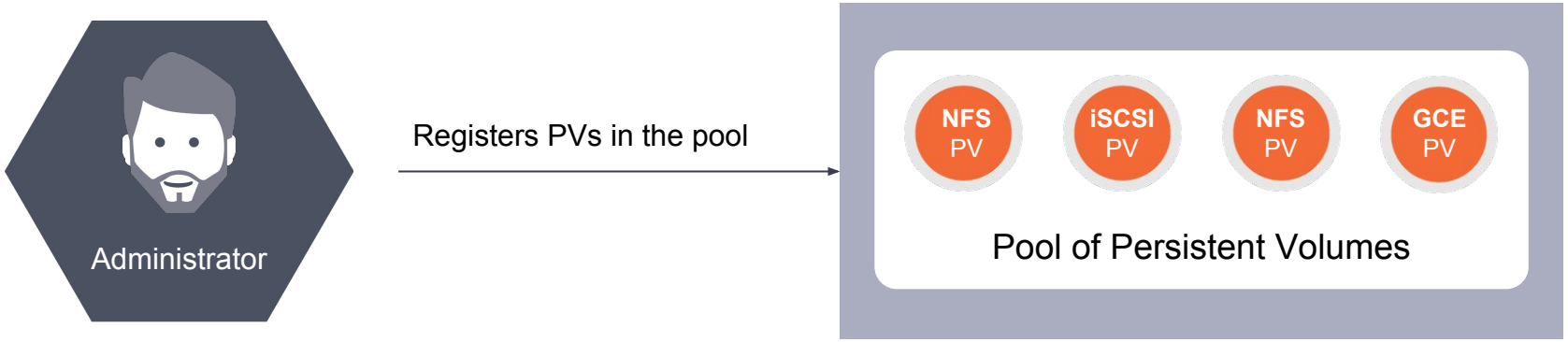
The storage platform should ensure high availability, durability, consistency with a predictable, proven data model.

8 Consistently available

STORAGEOS

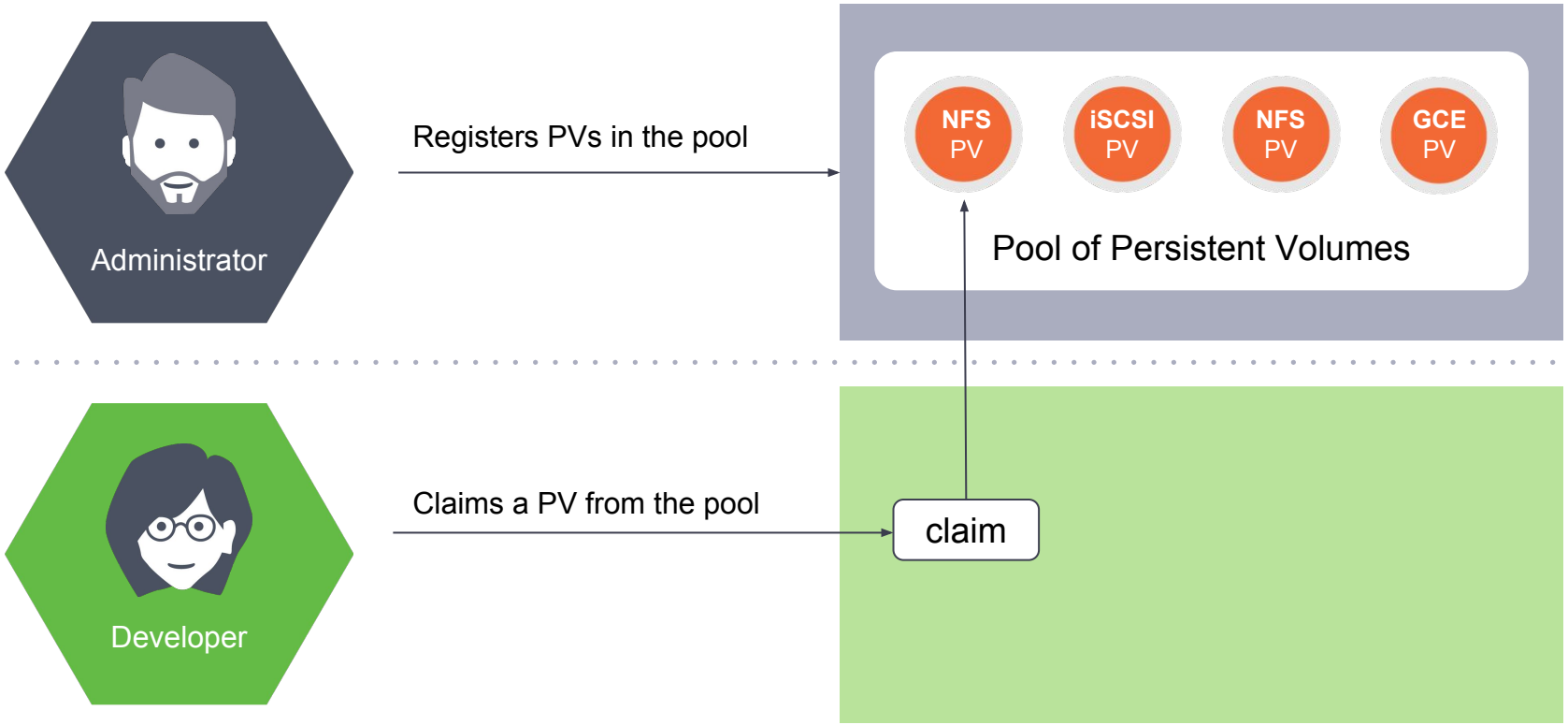
What storage should I use with Kubernetes?

Kubernetes Storage Model: Persistent Volumes and Claims



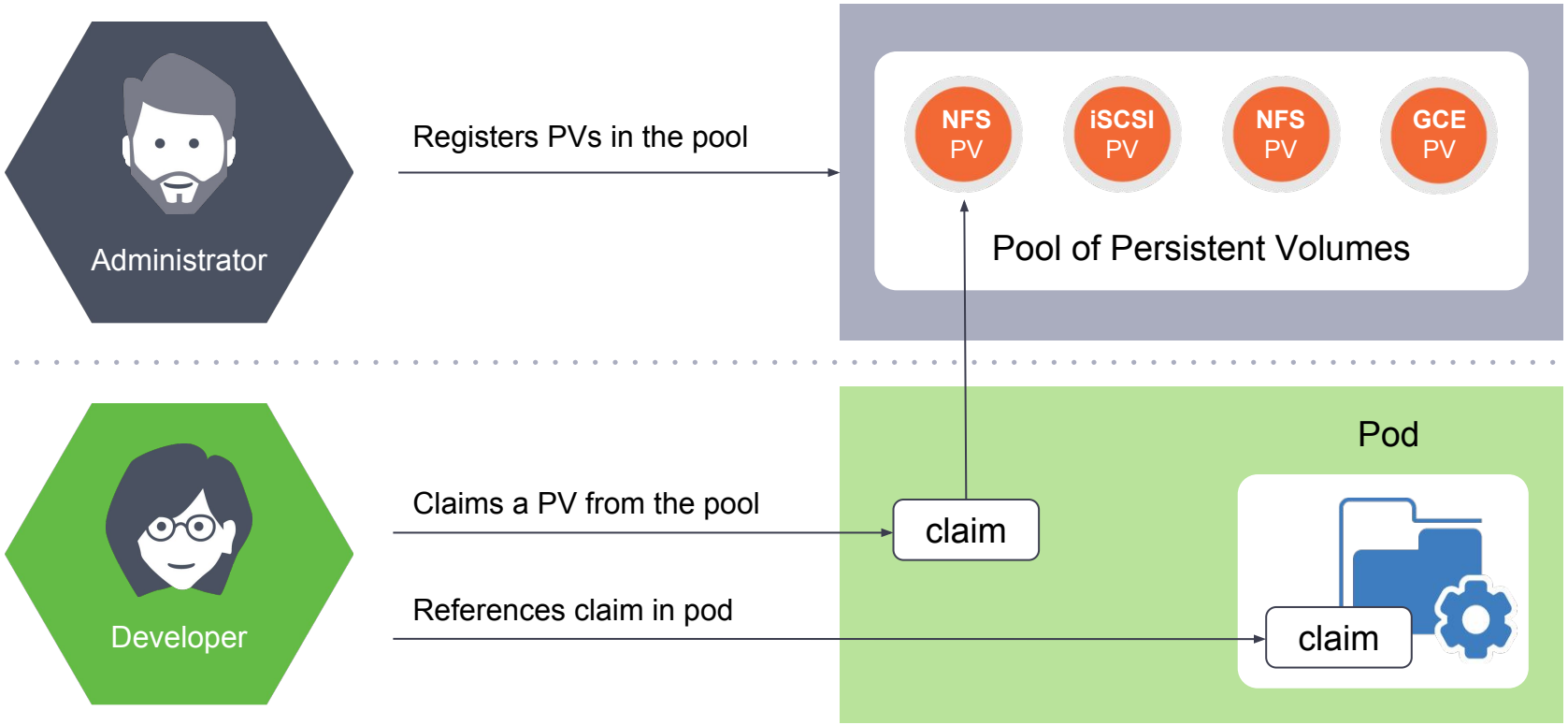
Kubernetes Storage Model: Persistent Volumes and Claims

@oicheryl

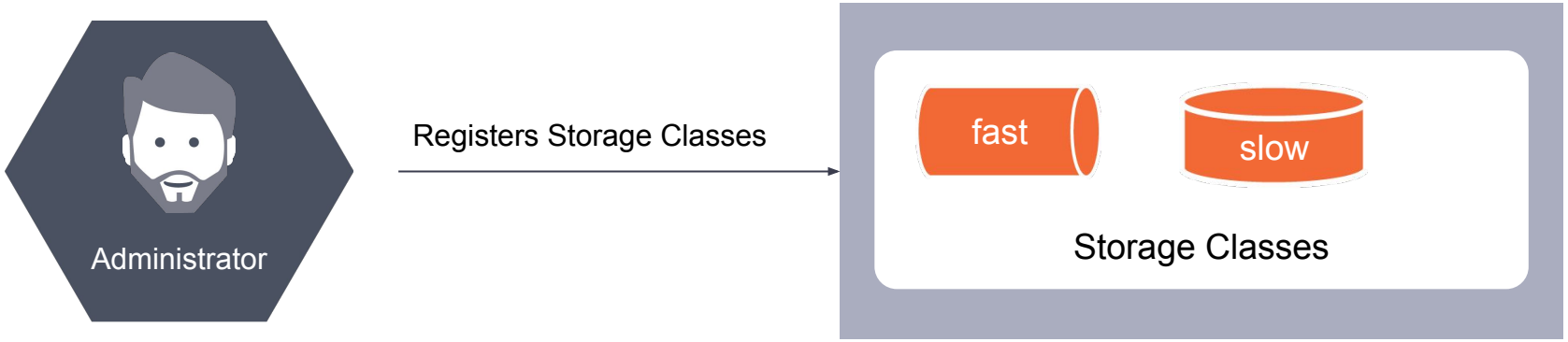


Kubernetes Storage Model: Persistent Volumes and Claims

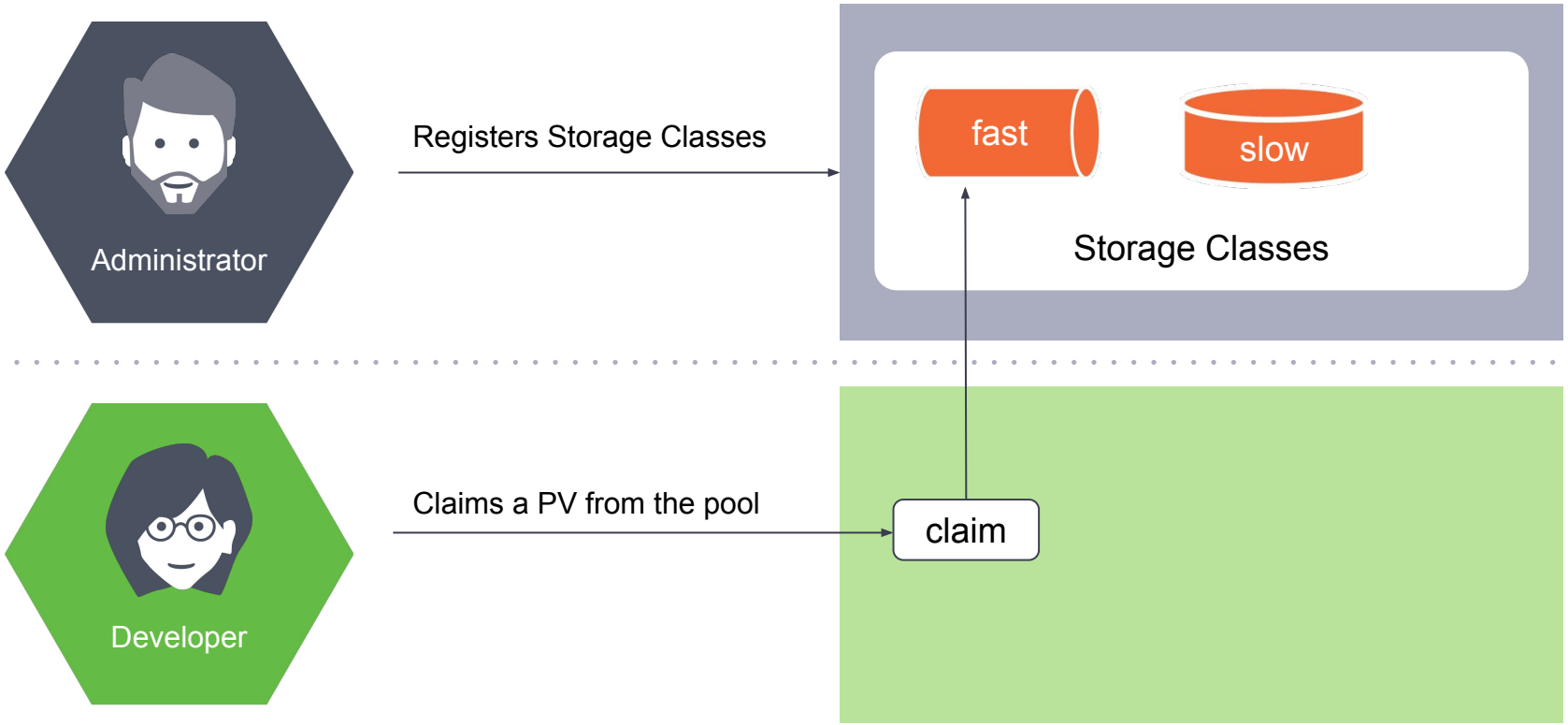
@oicheryl



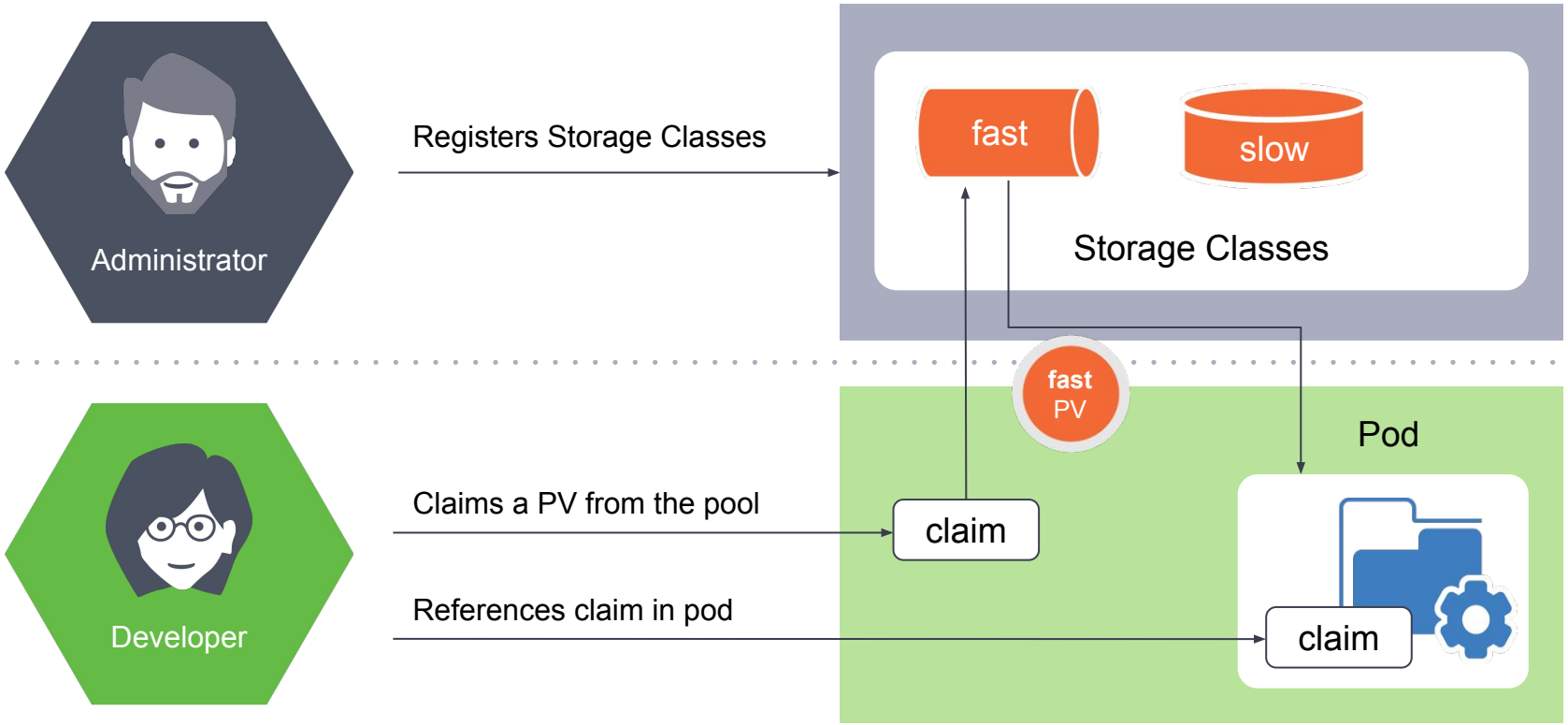
Dynamic provisioning with Storage Classes



Dynamic provisioning with Storage Classes



Dynamic provisioning with Storage Classes



Meet Jane

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

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

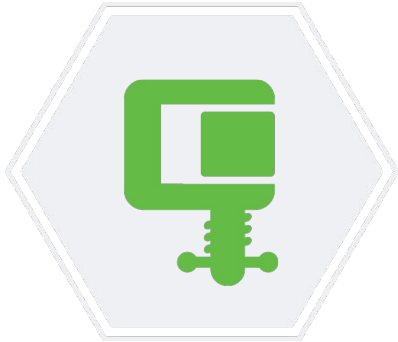
Key information

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?

App Binaries
Ephemeral

App data
Dedicated, performant, highly available



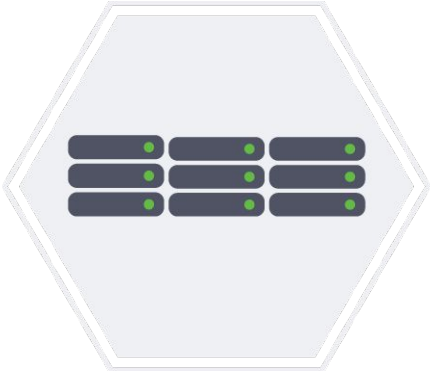
Config
Shared persistent

Backup
Cost efficient, cloud backup

2. What are my performance requirements?

- High performance
- Low latency
- Throughput
- High availability / replication
- Shared to multiple instances

3. How should developers access storage?

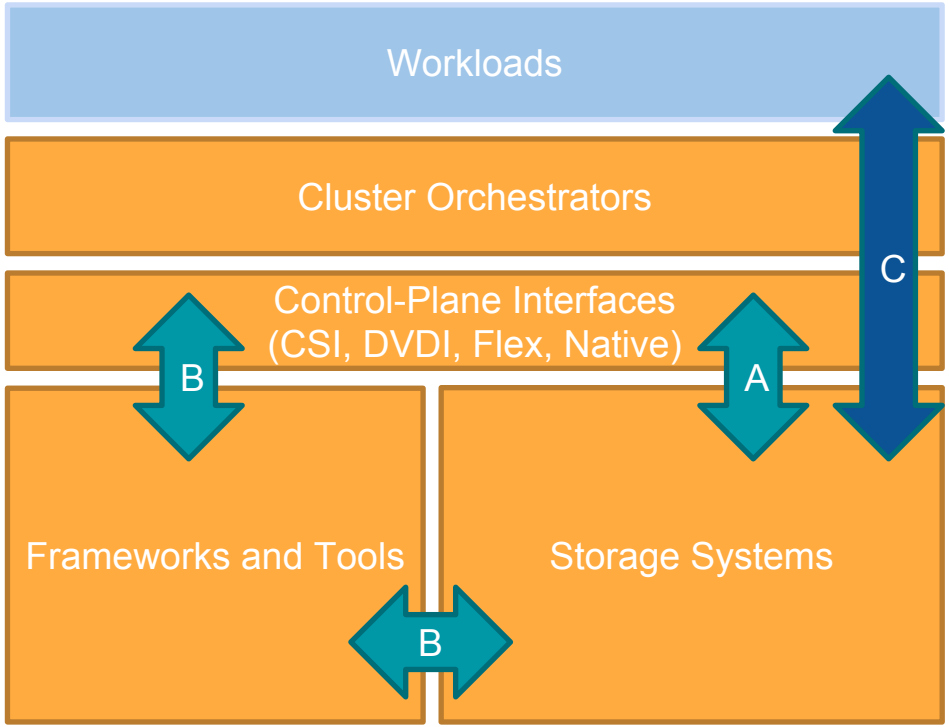


Block Storage
Fixed-size ‘blocks’ in a rigid arrangement – ideal for enterprise databases

File Storage
‘Files’ in hierarchically nested ‘folders’ – ideal for active documents

Object Storage
‘Objects’ in scalable ‘buckets’ – ideal for unstructured big data, analytics and archiving

4. Where is the storage deployed and managed?



- 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

Jane’s storage requirements

- Configuration data
- Postgres database for application data
- Shared media store for uploads
- Backup and archival

Database location and credentials

1. **Use case?** Config
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

User media files

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

Cloud: Object store, if the app can support it, or managed NFS

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

Database and website backup

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

Cloud: Managed object store or long term cold storage

On prem: object store, NAS

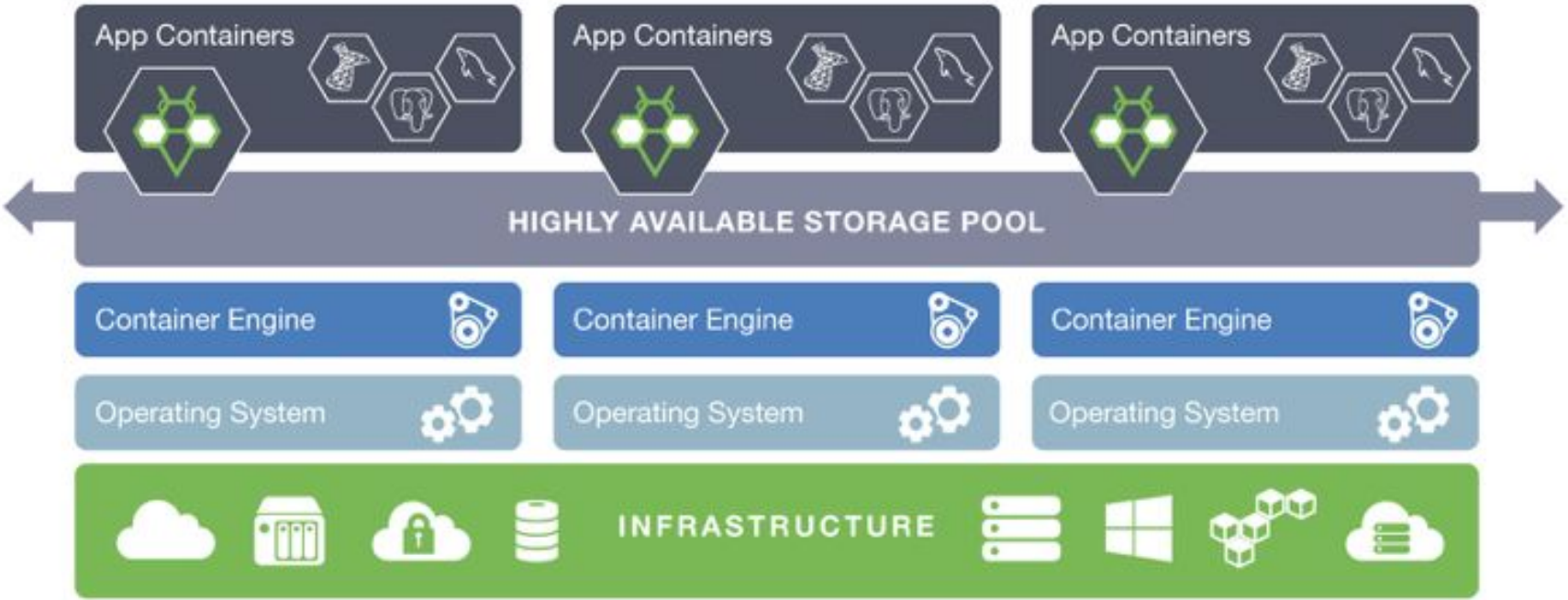
Postgres for application data

1. **Use case?** Transactional database
2. **Performance requirements?** HA, 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 (maximise configurability)

Software-defined storage



KEEP
CALM
IT IS
DEMO
TIME

To Recap…

Evaluating storage

1 Application centric

2 Platform agnostic

3 Declarative/ composable

4 API driven

5 Natively secure

6 Agile

7 Performant

8 Consistently available

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

CSI launched as alpha in Kubernetes 1.9

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 resources

Browser-based demo

- demo.storageos.cloud

Kubernetes quickstart

- storageos.com/kubernetes

We’re hiring! London and NYC roles

- storageos.com/careers

STORAGEOS

Thanks

Slides at oicheryl.com

What is StorageOS?

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

What is StorageOS?

Platform agnostic

Horizontally scalable

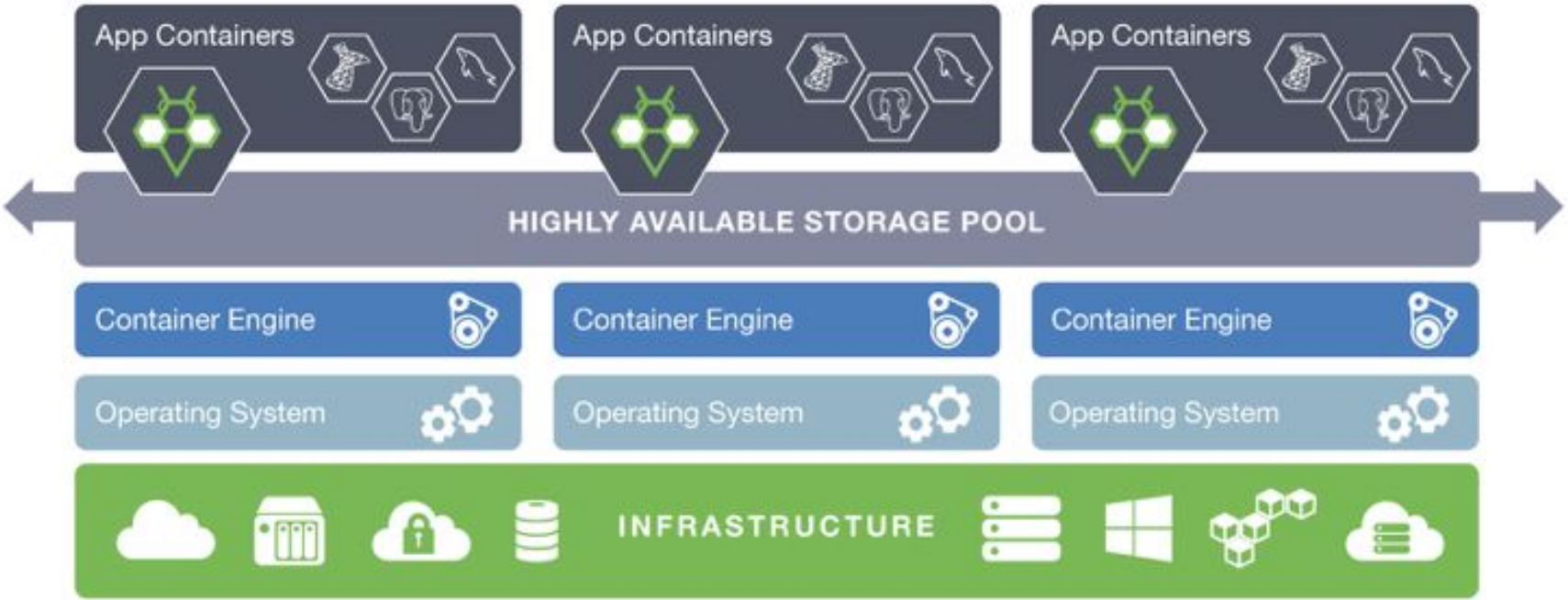
Database (ie. block)

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

Docker/K8s integration

High availability

StorageOS architecture



StorageOS architecture

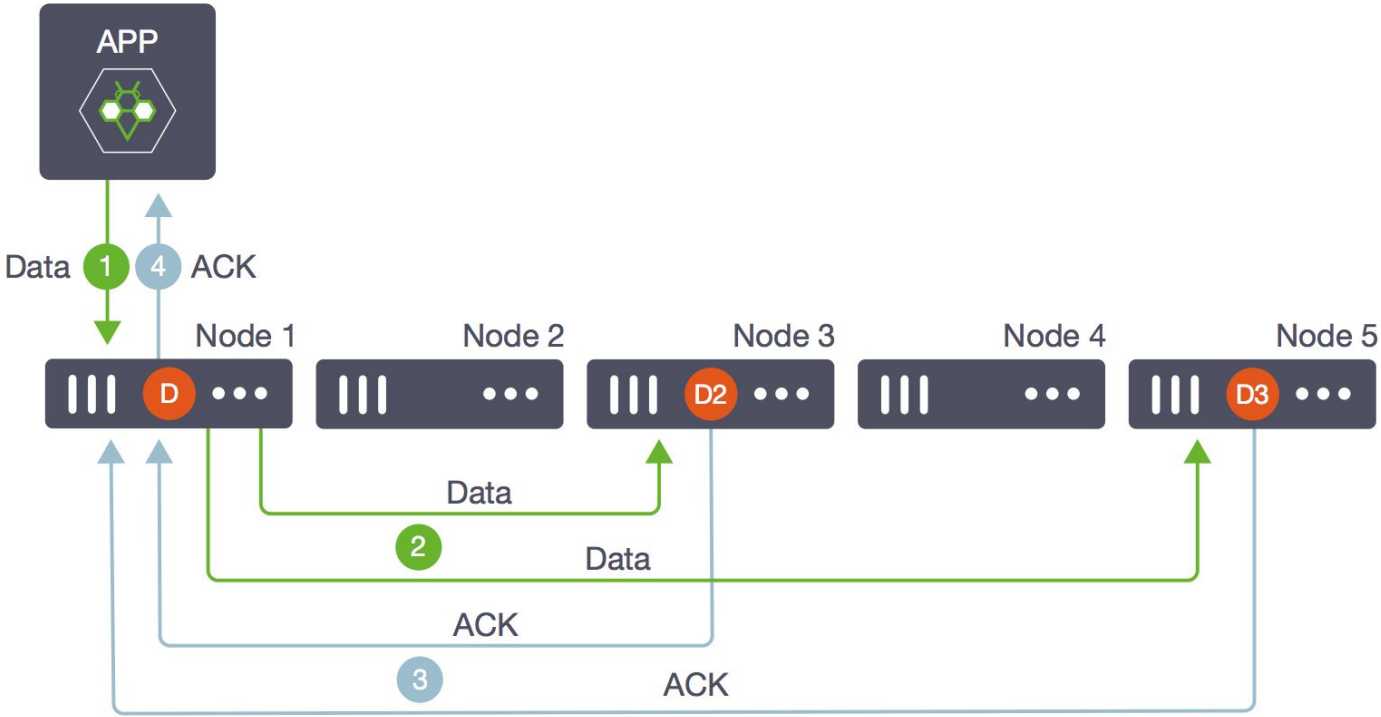
StorageOS is conceptually pretty simple; it's a virtualization layer on top of any commodity or cloud storage. It's deployed as one container per node, similar to a DaemonSet.

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 is unaware of the underlying storage.

StorageOS architecture

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



High availability with 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.

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

Read the full documentation at docs.storageos.com.