Persistent Storage for Windows workloads in Kubernetes

Anusha Ragunathan

Senior Software Engineer, Docker Inc.

Jean Rouge

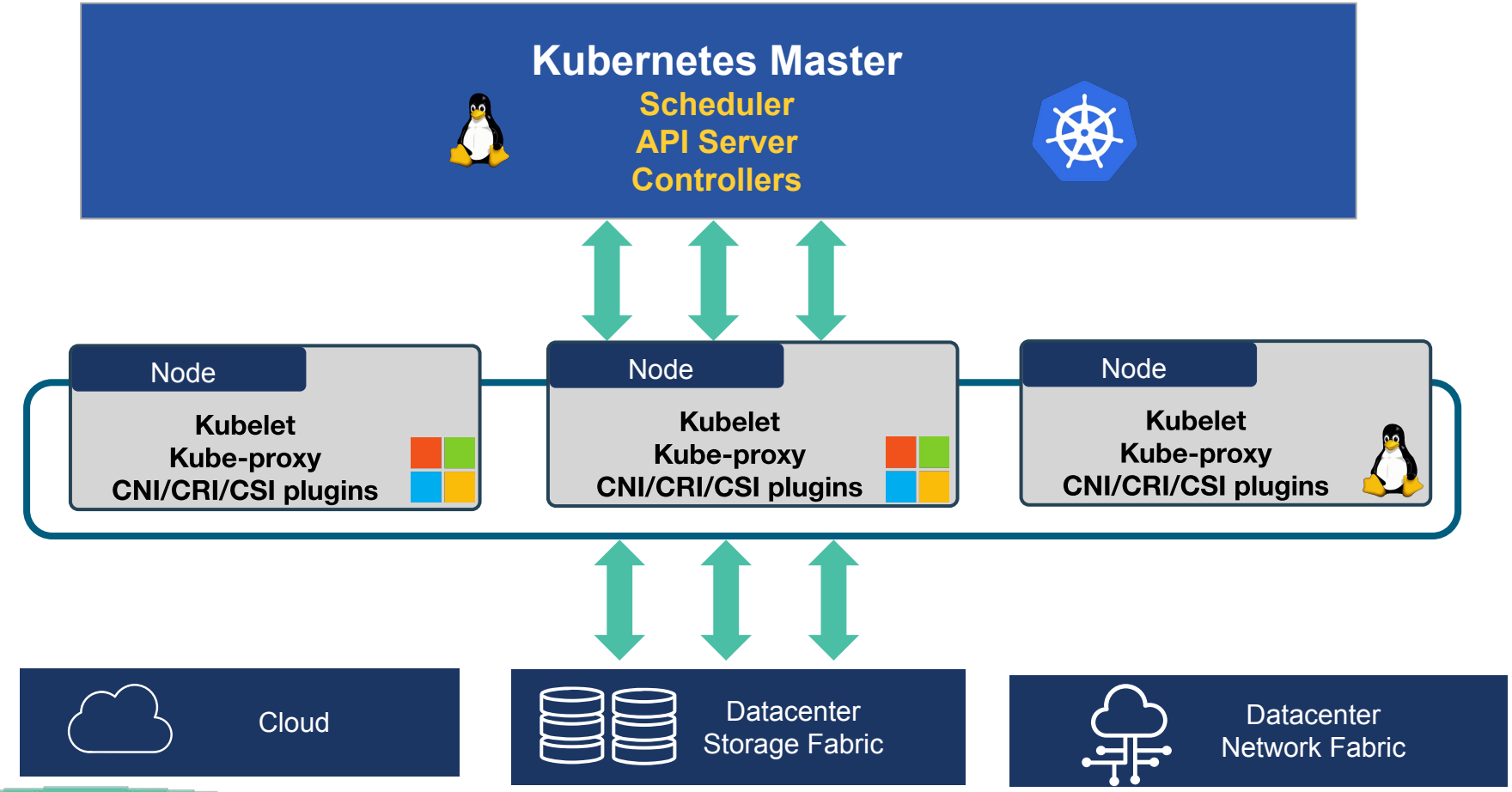
Senior Software Engineer, Docker Inc.

Agenda

- Persistent Storage in Kubernetes
  - **Concepts, Provisioning and Interface types**
- Current state of state in Windows
  - **In-tree, External Provisioner, FlexVol**
- Future: CSI plugins

Persistent storage in Kubernetes

Kubernetes Architecture



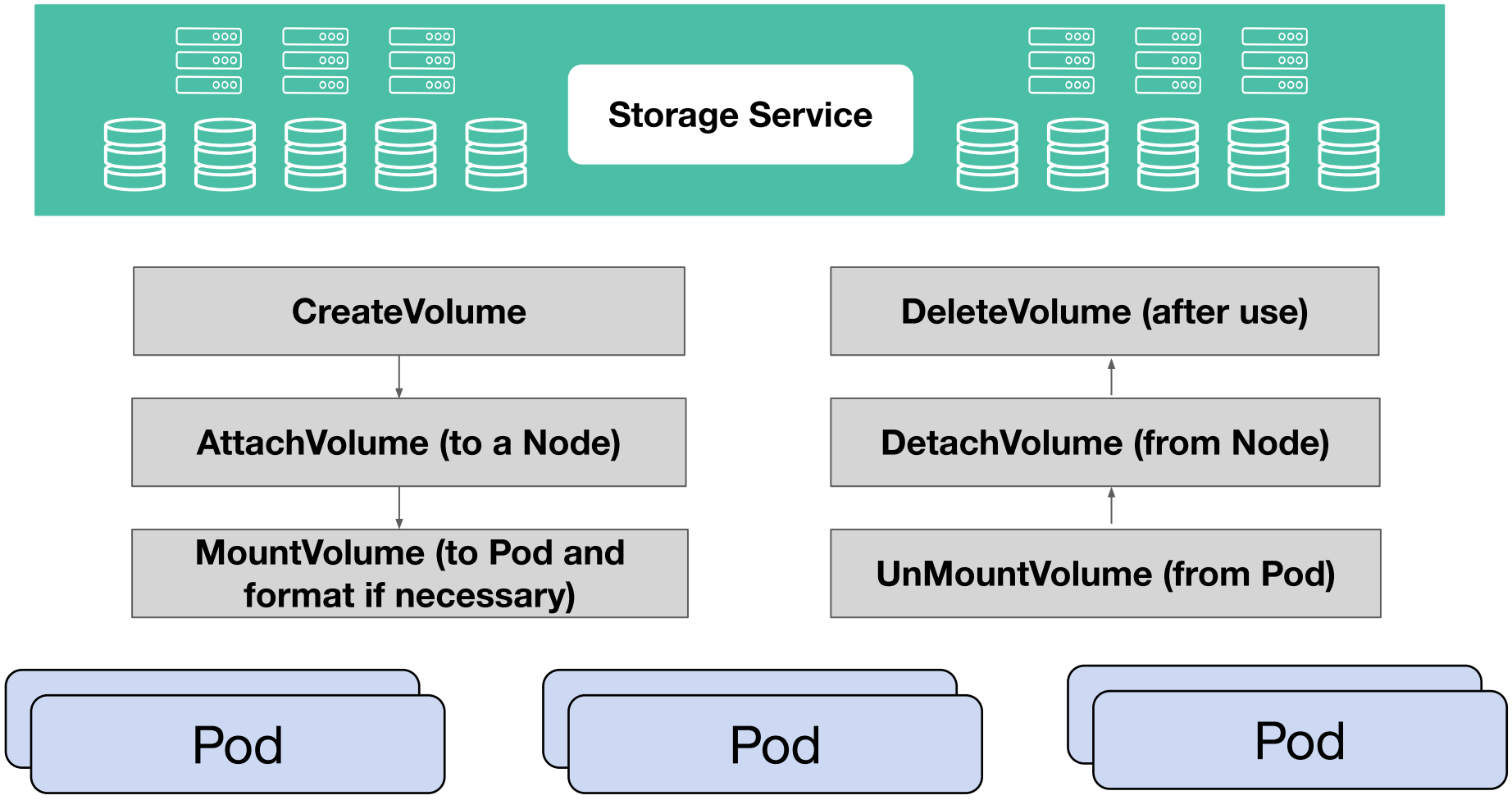
Components of Kubernetes storage

- **Cluster-wide** controller loops in master nodes
  - PV controller
  - Attach/Detach controller
  - Node controller
- OS-specific logic in **worker** nodes
  - Kubelet
  - Container runtimes
  - Flexvolume and CSI node plugins

Storage Concepts

Pods	Persistent Volume Claim (PVC)	Persistent Volume (PV)
● A set of running containers representing a workload	● A storage claim made by a user ● Just like how Pods consume *Compute* resources, PVC consume *Volume* resources. ● Just like how Pods can request specific levels CPU & memory, PVCs can request specific sizes and access modes. ● Pods reference PVC	● Storage resources in a cluster ● Lifecycle independent of a Pod

Life of a PV


Storage Service
CreateVolume
AttachVolume (to a Node)
MountVolume (to Pod and format if necessary)
DeleteVolume (after use)
DetachVolume (from Node)
UnMountVolume (from Pod)
Pod
Pod
Pod

PV Provisioning

- Provisioning is the creation/allocation of Persistent Volumes

- Static and Dynamic Provisioning
  - Static: pre creation of PV
  - Dynamic: automatic creation of PV based on size, permissions requested.

- Dynamic Provisioning through Storage Classes
  - Provides a way for Admins to describe “classes” of storage available. Eg, different performance SLAs, value-add features such as replication, backup, etc
  - Backed by a provisioner

State of state in Windows

(In-tree, FlexVol, External)

Persistent storage interfaces

- In-tree volume plugins
- Flexvolume plugins
- External provisioners
- CSI plugins

Persistent Storage Interfaces

Plugin Type	Pros	Cons	Future
InTree	No extra installation necessary	Release cycles tied to k8s releases	Superseded by CSI
External Provisioner	Customizable code for provisioning maintained independently	Limited customization for attach and mount ops	Superseded by CSI
FlexVolume	Customizable code attach/mount maintained independently	Host based exec model reduces portability. Plugin lifecycle is non-native to k8s.	Deprecated for Linux Will be used for Windows
CSI	Highly customizable code maintained independently and based on standard	Requires installation and configuration	The future of storage plugins

Considerations for Windows

- Disk device and Volume enumeration interfaces
  - Object based (as opposed to file-based on Linux)
- File System support
  - NTFS for block devices
  - SMB for shared storage
- Powershell cmdlets/scripts for many actions

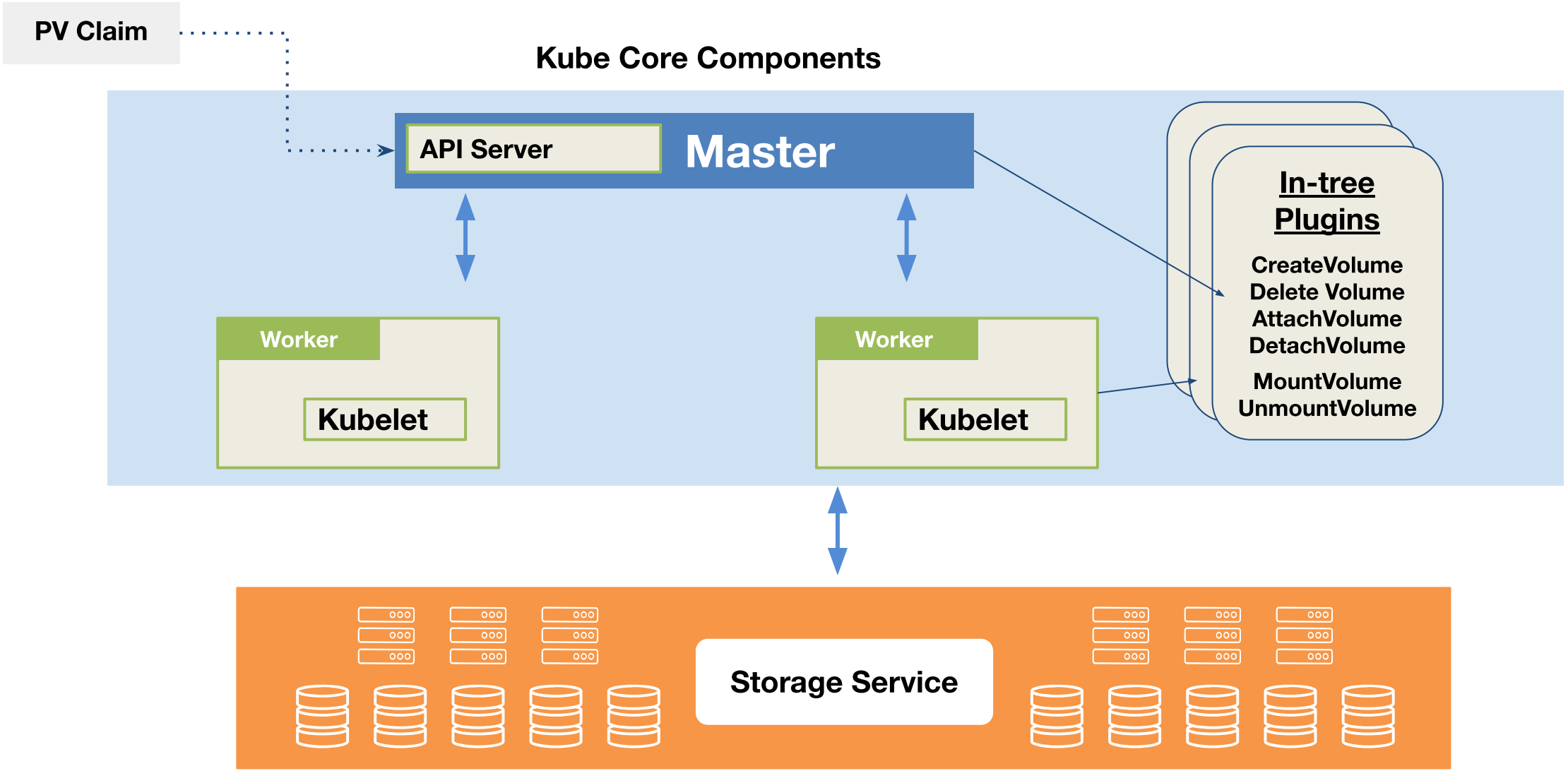
In-tree storage support for Windows

- Windows kubelet.exe supports:
  - Partition disk and format volumes with NTFS
  - Link volumes to container's file system
  - Link SMB shares to container's file system

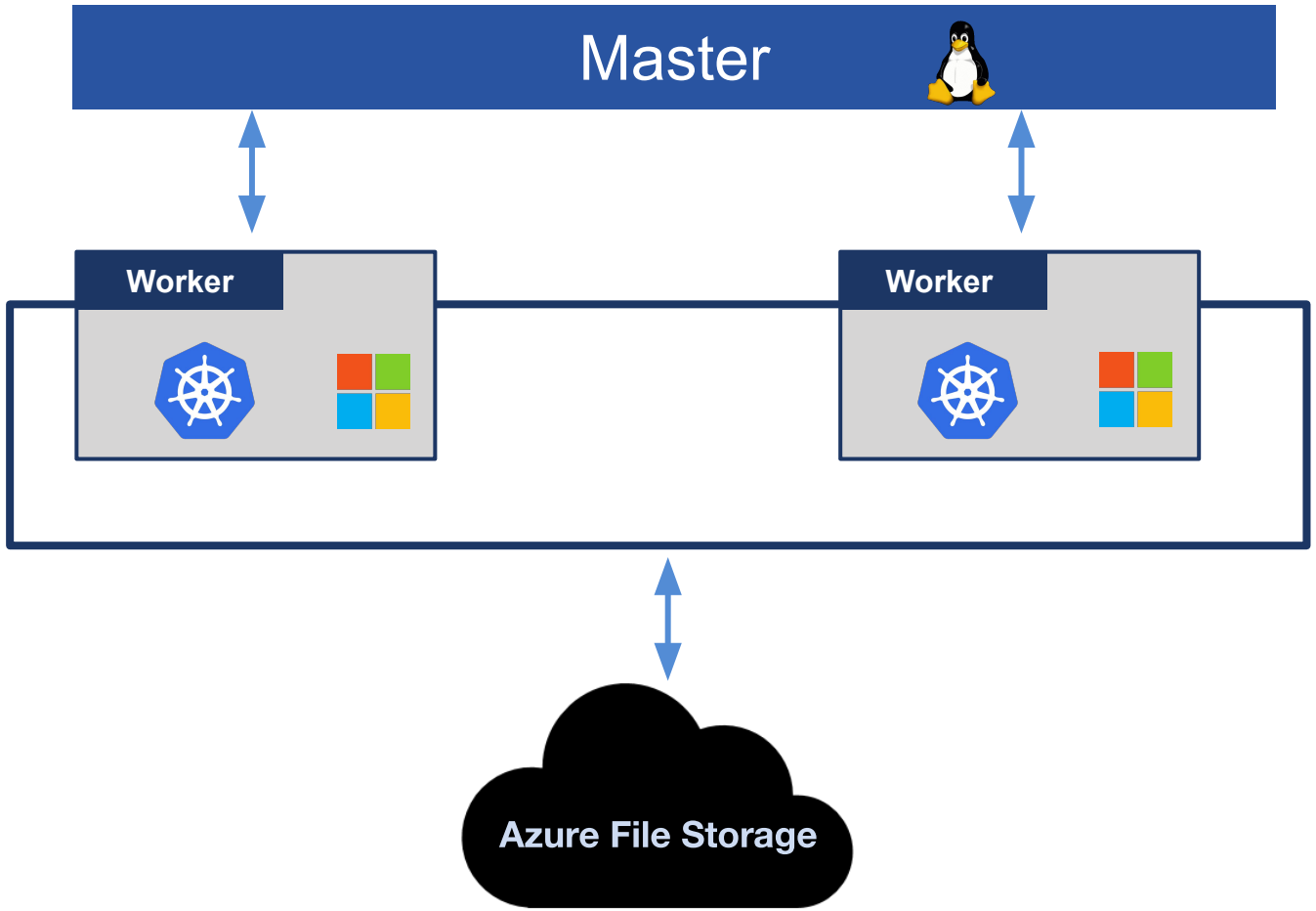
In-tree plugins in Windows

- File based cloud volumes:
  - Azure File through SMB
- Block based cloud volumes:
  - Azure Disk
  - GCE PD
  - EBS [WIP]
- iSCSI support (WIP. Alpha targeting 1.16)

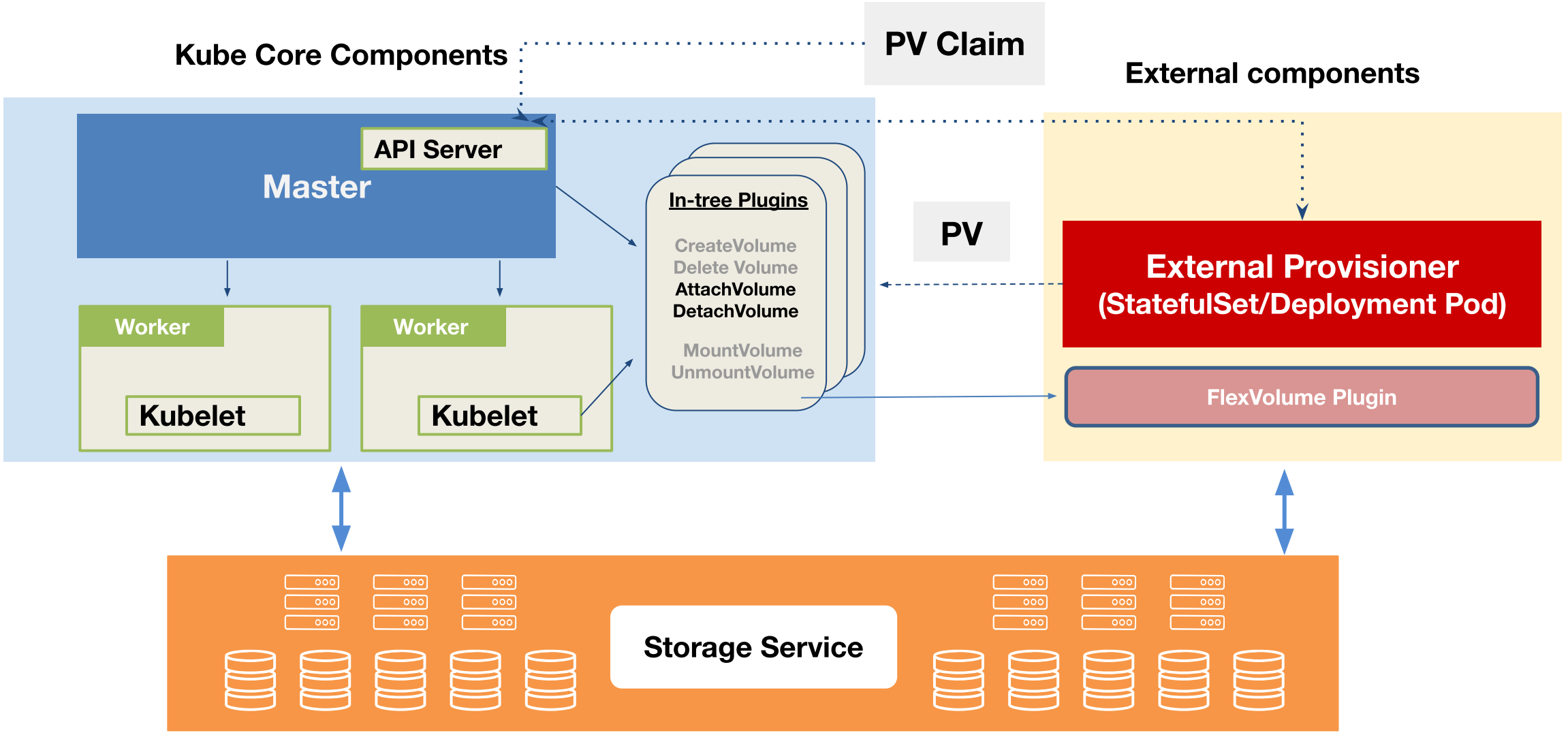
In tree Plugin Architecture



Demo 1: In-tree storage with Azure File



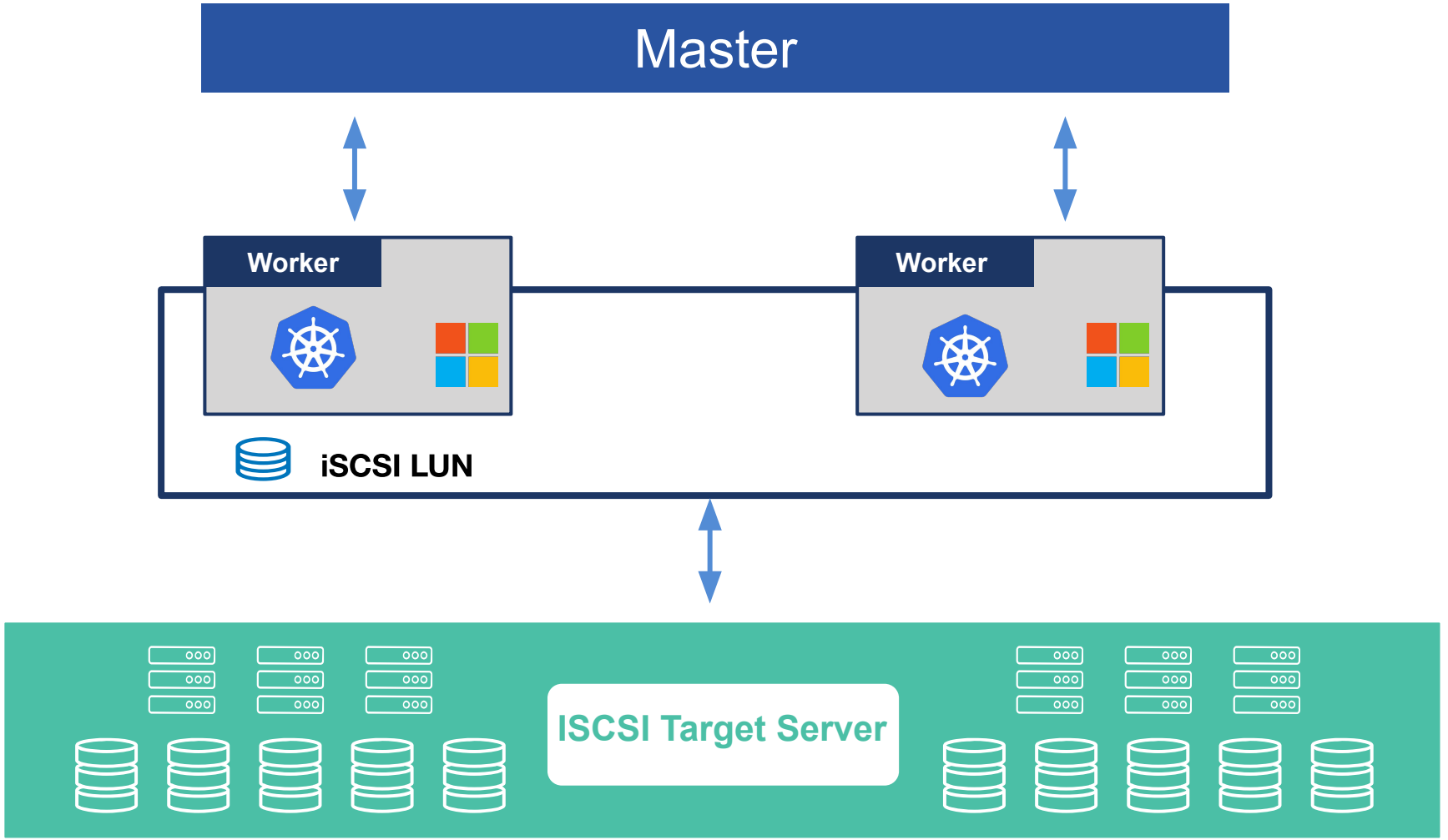
External Provisioner Architecture



Flexvolume plugins for Windows

- Attach/mount pre-provisioned volumes over:
  - SMB (shared file system)
  - iSCSI (dedicated block)
- Dynamic provisioning with external provisioners:
  - Set PV source to supported Flexvolume plugins

Demo 2: External Provisioning + iSCSI

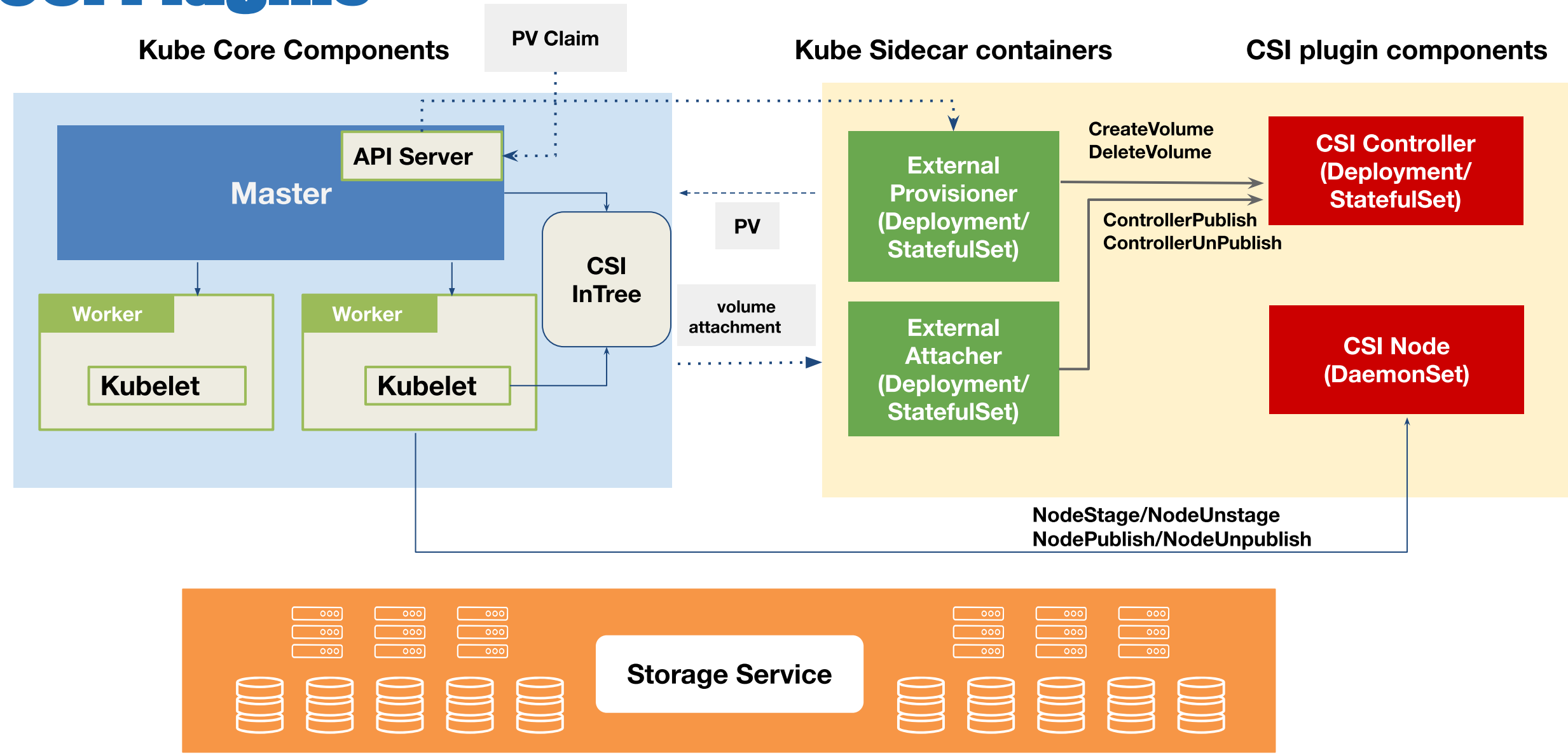


Demo 2: External Provisioning + iSCSI

https://github.com/wk8/k8s-win-iscsi-demo

Future: CSI

CSI Plugins


Kube Core Components
PV Claim
Kube Sidecar containers
CSI plugin components
API Server
Master
Worker
Kubelet
Worker
Kubelet
CSI
InTree
PV
volume
attachment
External
Provisioner
(Deployment/
StatefulSet)
External
Attacher
(Deployment/
StatefulSet)
CreateVolume
DeleteVolume
ControllerPublish
ControllerUnPublish
CSI Controller
(Deployment/
StatefulSet)
CSI Node
(DaemonSet)
NodeStage/NodeUnstage
NodePublish/NodeUnpublish
Storage Service

CSI Operations

Operation	Invocation
Provision/De-provision	PV claim => CSI External Provisioner => CSI Plugin => PV with CSI volume source
Attach/Detach	CSI in-tree => VolumeAttachment => CSI External Attacher => CSI Plugin
Mount/Unmount	Kubelet CSI client => CSI Plugin
Backup [Alpha]	Volume Snapshot object => CSI External Snapshotter => CSI Plugin
Restore [Alpha]	PV claim => CSI External Provisioner => CSI Plugin

CSI plugin required capabilities

- ❑ register as a plugin with the API & the kubelet
- ❑ make syscalls to the OS
- ❑ mount volumes accessible to other host processes

How all CSI plugins do it

- register as a plugin with the API & the kubelet

  => use the node driver registrar sidecar - unix socket

- make syscalls to the OS

  => run in privileged mode

- mount volumes accessible to other host processes

  => use mount propagation

CSI plugins for Windows

- CSI support for Windows is a **Work in Progress**.
- No privileged support for containers in Windows yet (ETA 2020)
- Design options for deploying plugin:
  - Host process
  - Container with privileged host-based proxy

Opt 1: CSI node plugins as host process

- Plugin process runs directly on host
- Handles plugin registration with kubelet
  - Cannot use node driver registrar sidecar
  - Other side cars can be scheduled on Linux masters.

Opt 1: CSI node plugins as host process

But...

- ❏ packaging/distribution?
- ❏ independent distribution, life-cycle and log management

Opt 2: CSI node plugins as containers

- ❑ Plugin process packaged and deployed as containers
- ❑ Use node driver registrar for plugin registration
- ❑ Distribution, life-cycle and log management through Kubernetes
- ❑ Requires a privileged proxy (or containerd-shim based process) in host for operations

Opt 2.1: use a proxy host process

raw binary exposing a versioned API on a Windows named pipe and making syscalls on behalf of CSI plugin pods

But...

- ❏ API versioning?
- ❏ restrict use to authorized pods

Opt 2.2: host-process shim container

package a binary in a container, and modify containerd to run it as a raw binary, with all the semantics of a “normal” container

But...

- ❏ hacky!!
- ❏ might result in poor UX (“fake” container)

Help us choose!

https://github.com/kubernetes/community/tree/master/sig-windows

Thanks to Deep Debroy (@ddebroy)

& Patrick Lang (@PatrickLang)

& Michael Michael (@michmike)

Summary

- In-tree and Flexvolume plugins ready to be used.
- External provisioners coming soon.
- CSI plugins for Windows node plugins in future.


Thank You
Q&A