KubeCon | CloudNativeCon

North America 2019

KubeCon | CloudNativeCon

North America 2019

Superpowers for Windows Containers

Jean Rouge

Deep Debroy

Speakers

Jean Rouge,
Senior Software Engineer,
Docker Inc.
SIG-Windows contributor

Deep Debroy,
Engineering Manager,
Docker Inc.
SIG-Windows tech lead

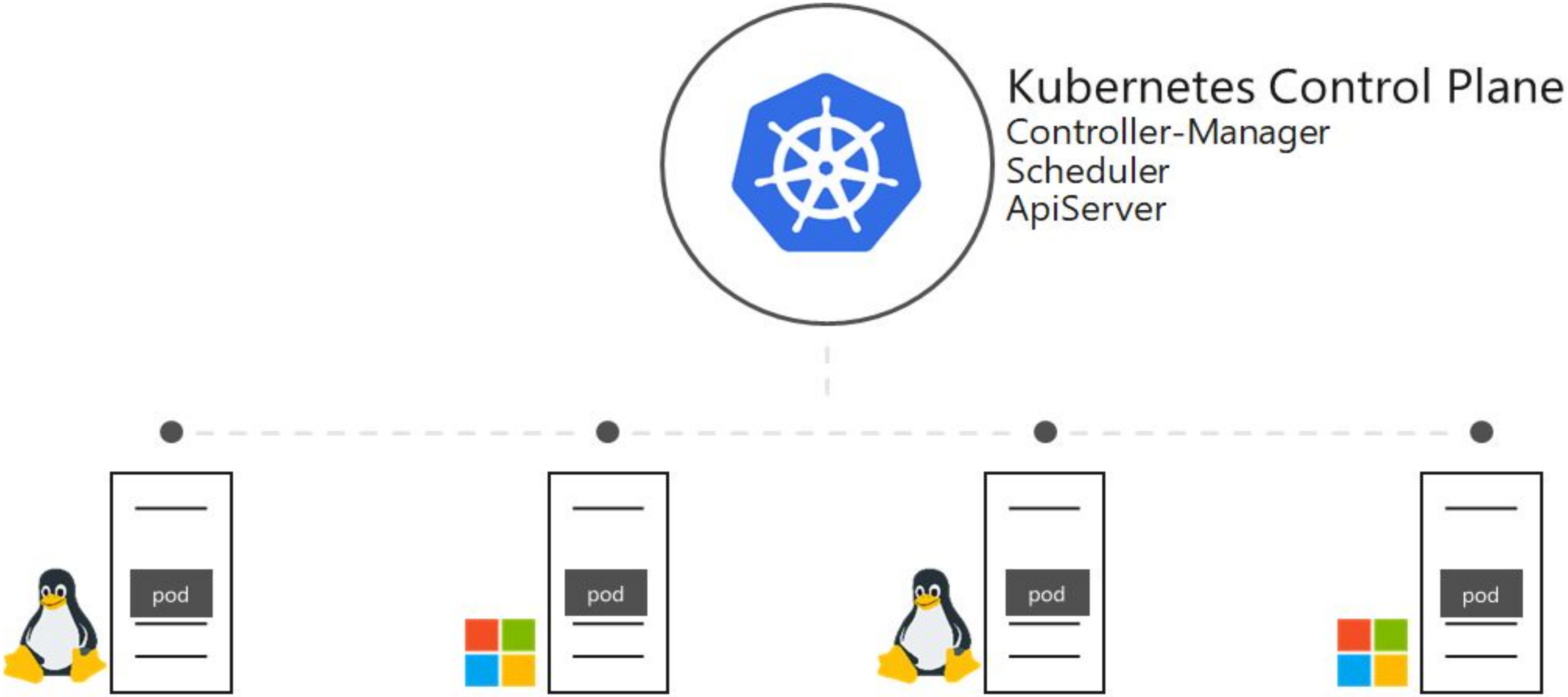
Agenda

- ❑ Support for containers and Kubernetes on Windows
- ❑ Windows containers and privileged operations
- ❑ Privileged Proxies for Windows containers
- ❑ Design of a Privileged Proxy for storage
- ❑ Use cases of Privileged Proxies
- ❑ Conclusion

Introduction to Windows Containers

Windows Server 2016	Initial support for containers on Windows
Windows Server 2019	Enhancements for containers on Windows in orchestrators
1.14 + Windows Server 2019	GA support for Windows workloads on Kubernetes clusters using HCSv1/Docker

Windows Containers in Kubernetes



Kubernetes Control Plane
Controller-Manager
Scheduler
ApiServer

Worker Nodes
Kublet
Kub-proxy
CRI/CNI/Storage Plugins

Privileged Operations with Windows Containers

- Windows does not support container capabilities and privileges
- Containers cannot perform “privileged” operations on Windows:
    - Access and manage host registry
    - Manage host networking configuration
    - Access and manage storage drives on host
- Limited ability to act as Kubernetes DaemonSets
    - No native support for containerized CSI/CNI plugins

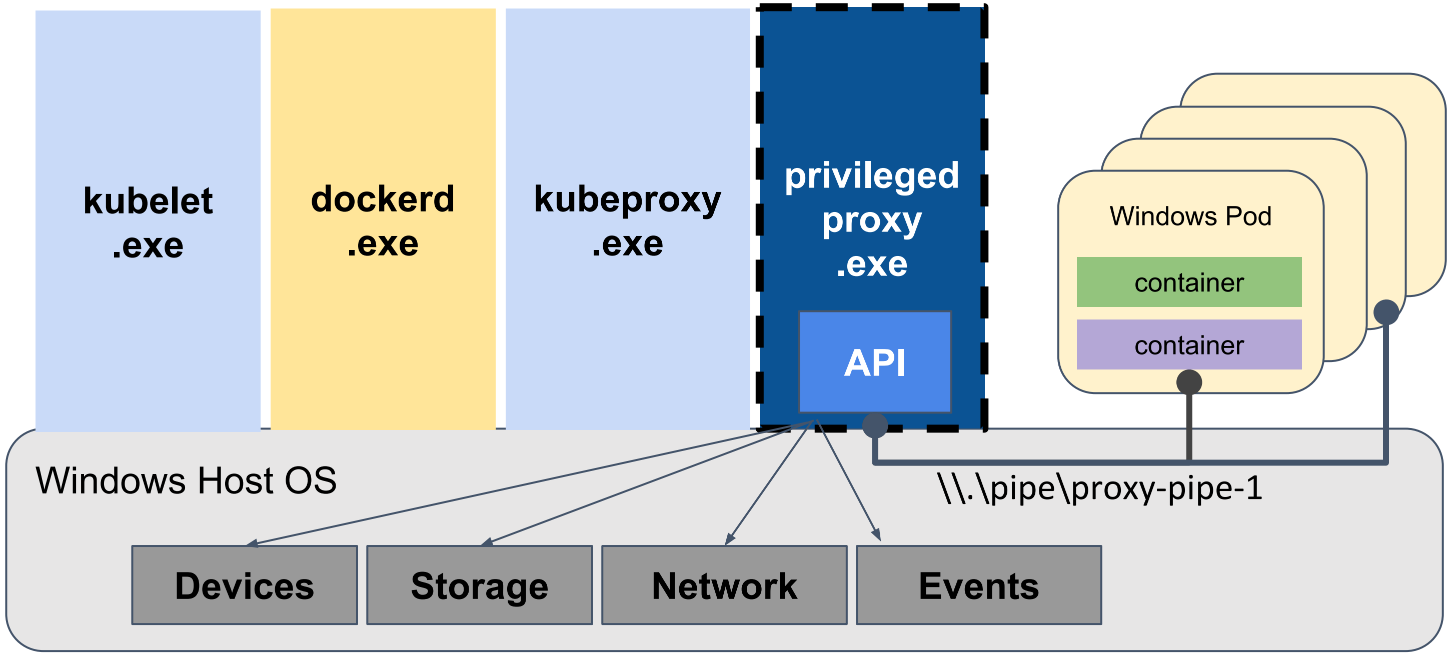
Workarounds for Privileged Operations

- Remote access into the host OS shell from a container
  - Containers need to know host address
  - Challenging to constrain access from Kubernetes
- Runtime class to launch a process from container image
  - Requires runtime enhancements on Windows
- Privileged proxy binary running on host
  - Focus of this presentation

Privileged Proxy for Windows Containers

- Regular binary on host performs privileged operations on behalf of containers
  - Potentially configured as a Windows service
  - Surfaces named pipes and APIs
- Supported operations may be scoped to OS subsystems
- Operations can be validated against policies

Privileged Proxy Architecture



```yaml
apiVersion: v1
kind: Pod
metadata:
 name: test-pod1
spec:
 nodeSelector:
   beta.kubernetes.io/os: windows
 containers:
 - name: container1
   image: org/image:tag
   volumeMounts:
   - name: proxy-pipe
     mountPath: \\.\pipe\proxy-pipe-1
 volumes:
   - name: proxy-pipe
     hostPath:
       path: \\.\pipe\proxy-pipe-1
       type: “”
```

Privileged Proxy: Considerations

- Proxy binary will need to be deployed/maintained on host
    - Use host bring-up/preparation scripts

- Restrict access to named pipes surfaced by privileged proxy
    - Use Pod Security Policy and Service Accounts
    - Use custom webhook/OPA policies

Privileged Proxy: Access Control with PSP

Deny hostpath mounts by default

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: deny-hostpath
spec:
  ...
  # Skip HostPath as allowed volume type
  volumes:
    - 'configMap'
    - 'emptyDir'
    - 'projected'
    - 'secret'
    - 'downwardAPI'
    - 'persistentVolumeClaim'
 ...
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: restricted-role
rules:
- apiGroups:
     - extensions
  resources:
     - podsecuritypolicies
  verbs:
     - use
  resourceNames:
     - deny-hostpath
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: restricted-binding
roleRef:
  kind: ClusterRole
  name: restricted-role
  apiGroup: rbac.authorization.k8s.io
subjects:
# Authorize all service accounts/users in all namespaces
- kind: Group
  apiGroup: rbac.authorization.k8s.io
  name: system:serviceaccounts
- kind: Group
  apiGroup: rbac.authorization.k8s.io
  name: system:authenticated
```

Privileged Proxy: Access Control with PSP

Allow hostpath mounts in privileged ns

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: allow-hostpath
spec:
  ...
  # Add HostPath as allowed volume type
  volumes:
    - 'configMap'
    - 'emptyDir'
    - 'projected'
    - 'secret'
    - 'downwardAPI'
    - 'persistentVolumeClaim'
    - 'hostPath'
 ...
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: privileged-role
rules:
- apiGroups:
     - extensions
  resources:
     - podsecuritypolicies
  verbs:
     - use
  resourceNames:
     - deny-hostpath
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: privileged-binding
roleRef:
  kind: ClusterRole
  name: privileged-role
  apiGroup: rbac.authorization.k8s.io
subjects:
# Authorize service accounts in a privileged namespace
- kind: Group
  apiGroup: rbac.authorization.k8s.io
  name: system:serviceaccounts:privileged-namespace
```

Privileged Proxy: Benefits

- Plugin/Distro vendors can distribute product and environment specific binaries in Windows container images
    - While depending on community maintained proxies

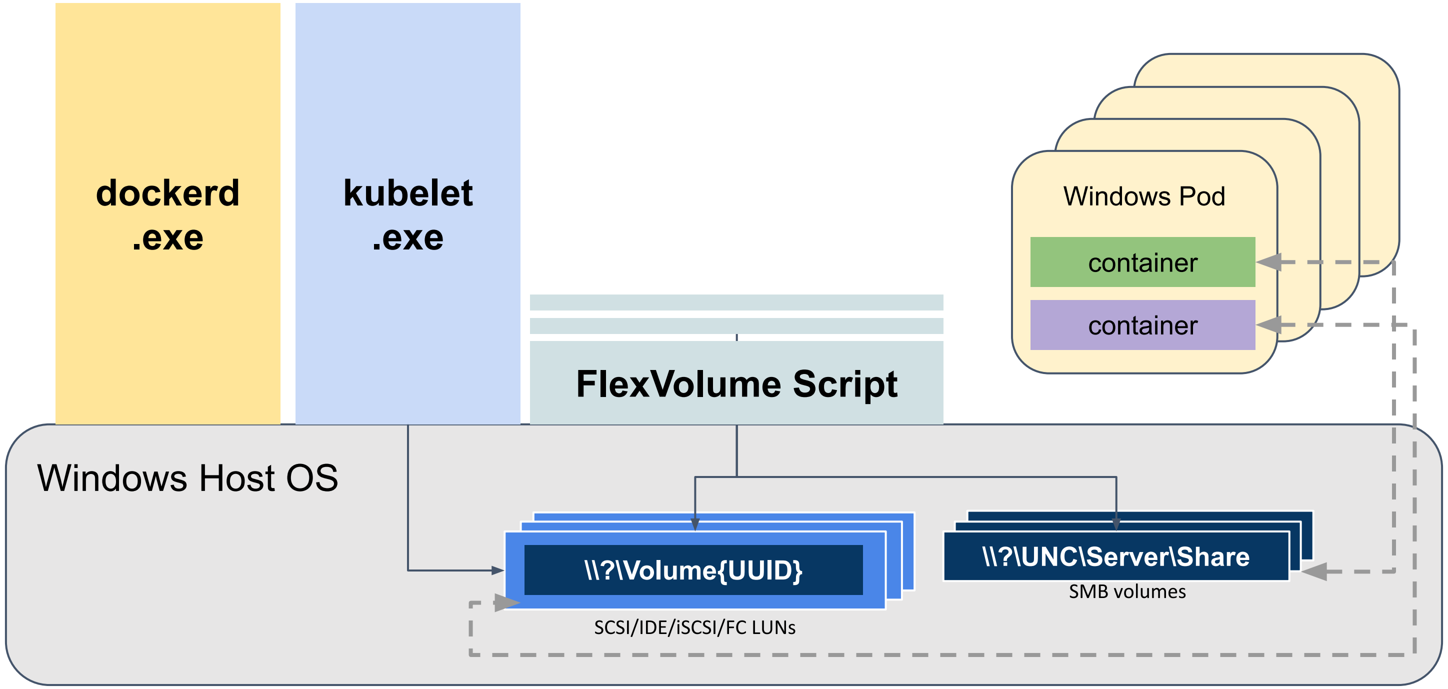
- Operators can deploy, configure and maintain the life-cycle of containerized plugins for Windows using native Kubernetes constructs like Daemonsets

Privileged Proxy Use-Case: Storage

Legacy storage plugin models that support Windows nodes:

- In-tree plugins compiled into Kubelet.exe
- FlexVolume plugin scripts that Kubelet invokes on host
- Direct access to host drives and volumes from Kubelet/scripts running on host

Persistent Storage for Windows Today



Privileged Proxy Use-Case: Storage

Container Storage Interface (CSI) Node plugins

- Implement the modern CSI spec [GA in 1.13]
- Typically distributed as containerized binaries for Linux
- Deployed in privileged pods on Linux nodes
- Need a similar mechanism for Windows nodes

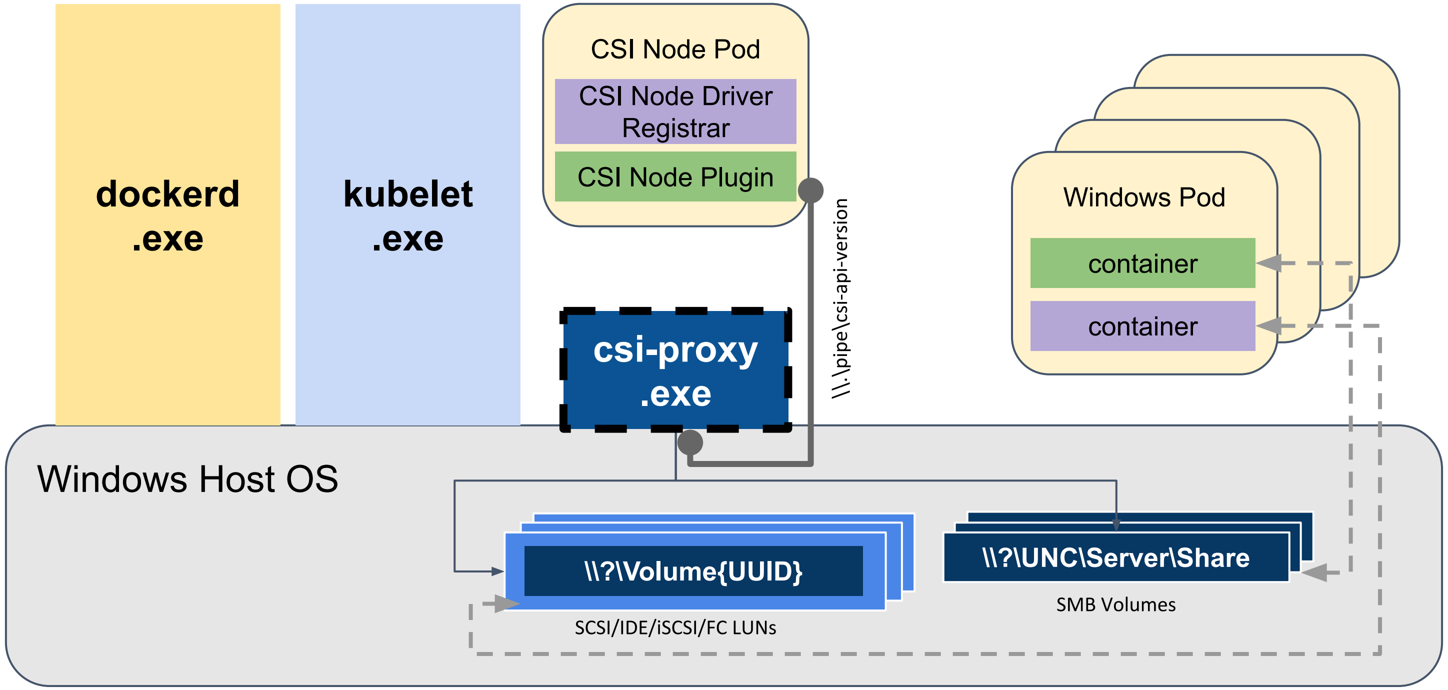
Privileged Storage Operations

Container Storage Interface (CSI) Node plugins need to:

- Scan physical disk objects based on SCSI IDs
- Partition a disk and create and format a partition
- Mount SMB shares
- Interact with iSCSI targets

CSI Proxy surfaces an API through named pipes to enable these

Privileged Proxy Use-Case: Storage



Proxy API Versioning

The proxy needs to be easy to evolve & maintain:

- Add new capabilities
- Modify existing capabilities
- Preserve backward-compatibility across release cycles

Proxy API Versioning

Same notion of API groups and versions as k8s itself uses:

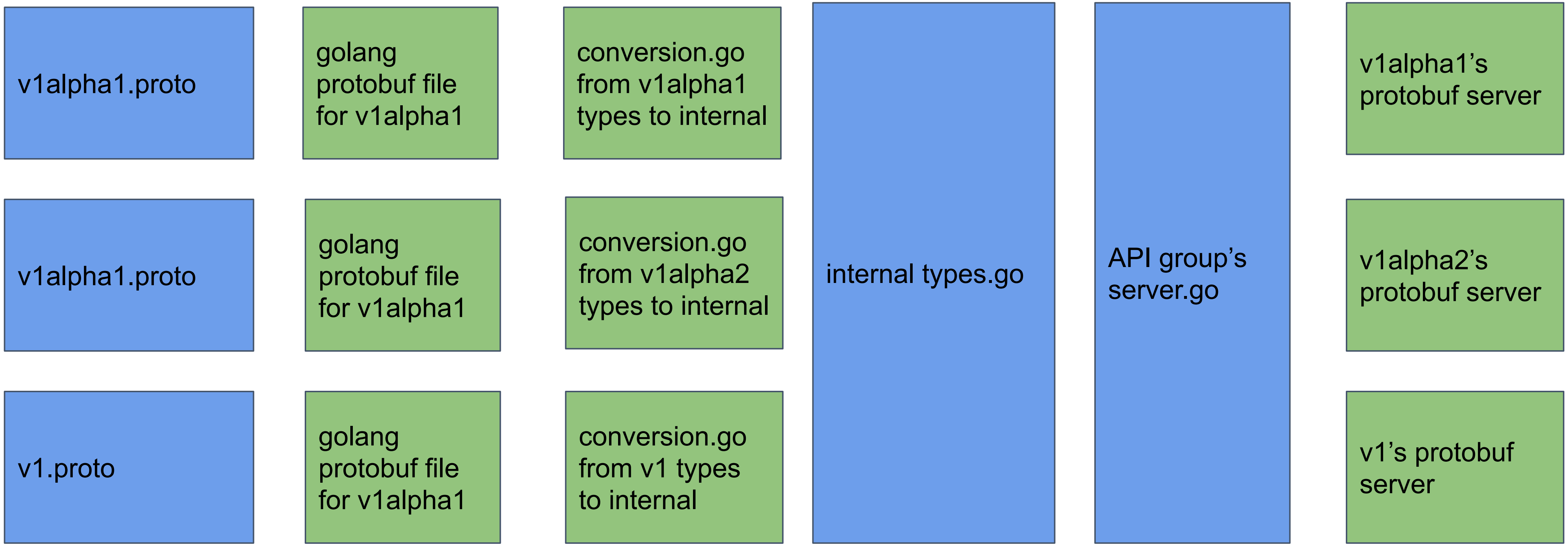
- Capabilities grouped by API groups
  - Disk, Volume, FileSystem, SMB, iSCSI
- Each API group has one or several versions
- Versions maintained then deprecated according to a release schedule

Proxy API Versioning

Internally:

- Each API group has a single internal representation for all versioned objects, and a single server that handles all versions for that group
- Auto-generated code handles conversion from versioned types to internal representations, creating named pipes...

Proxy API Versioning



v1alpha1.proto

golang protobuf file for v1alpha1

conversion.go from v1alpha1 types to internal

v1alpha1.proto

golang protobuf file for v1alpha1

conversion.go from v1alpha2 types to internal

v1.proto

golang protobuf file for v1alpha1

conversion.go from v1 types to internal

internal types.go

API group's server.go

v1alpha1's protobuf server

v1alpha2's protobuf server

v1's protobuf server

Green files are auto-generated

Proxy versions: deployment

- Cluster administrators need to make sure the right proxy version is present on the nodes where they need them.
- Each version of CSI proxy maintains up to 12 months or 3 releases (whichever is longer) for each API group.
- Possible to run several versions of CSI proxy on the same host.

CSI-Proxy Demo

Other Privileged Proxy Use Cases

- Container Network Interface (CNI) plugins
  - With community maintained proxy for HNS API calls

- DaemonSet for node monitoring and diagnostics
  - With community maintained proxy for collecting host Event Logs, ETW traces and other log sinks.

Future Directions

- Configurable set of proxies loaded by Kubelet
  - Eases life-cycle management of proxy binaries

- Native support for privileged containers on Windows

Q&A

Thank you!

Q&A