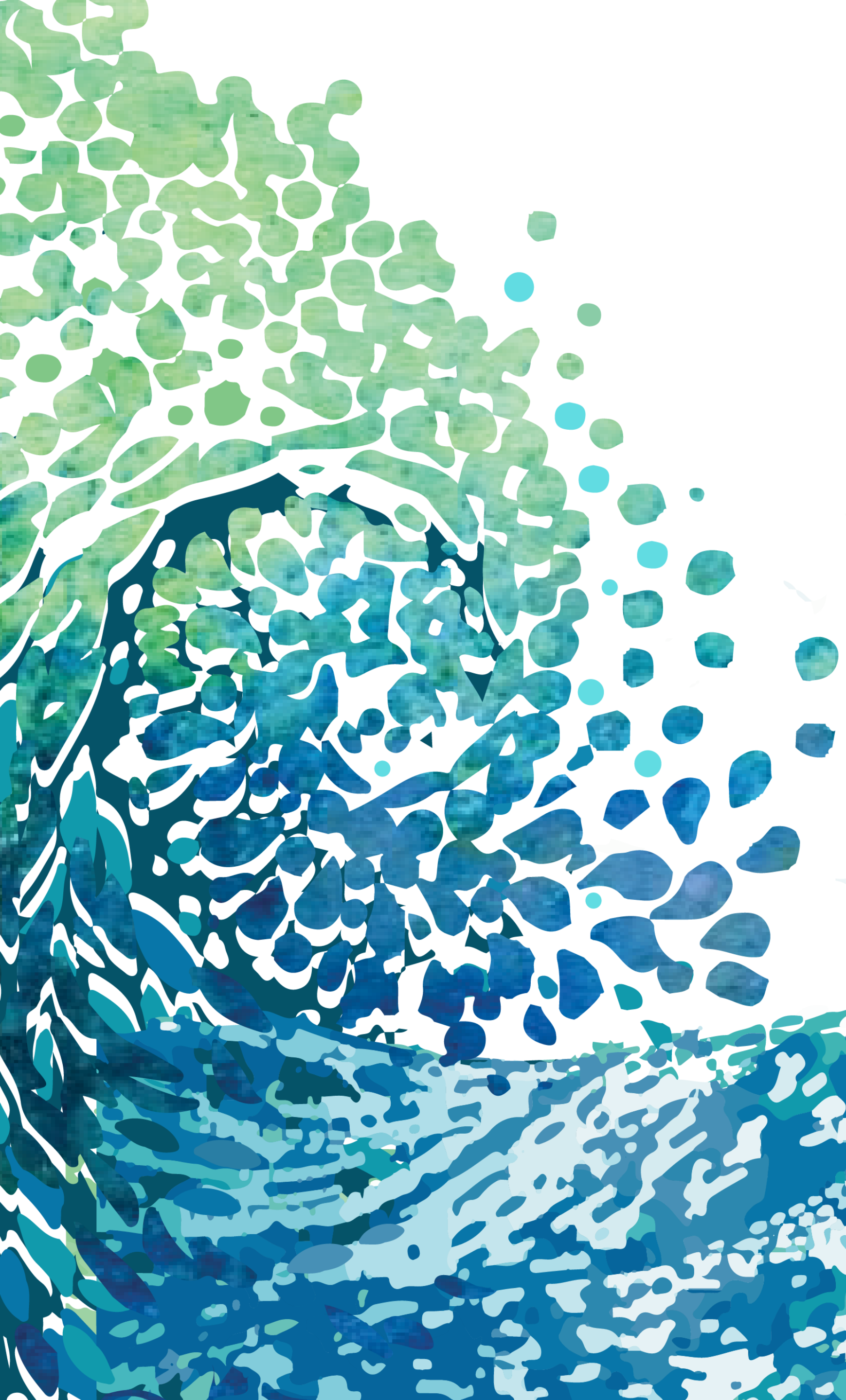


KubeCon | CloudNativeCon

Europe 2019

KubeCon | CloudNativeCon

Europe 2019

Reenforce Kubernetes image isolation in multi-tenant service

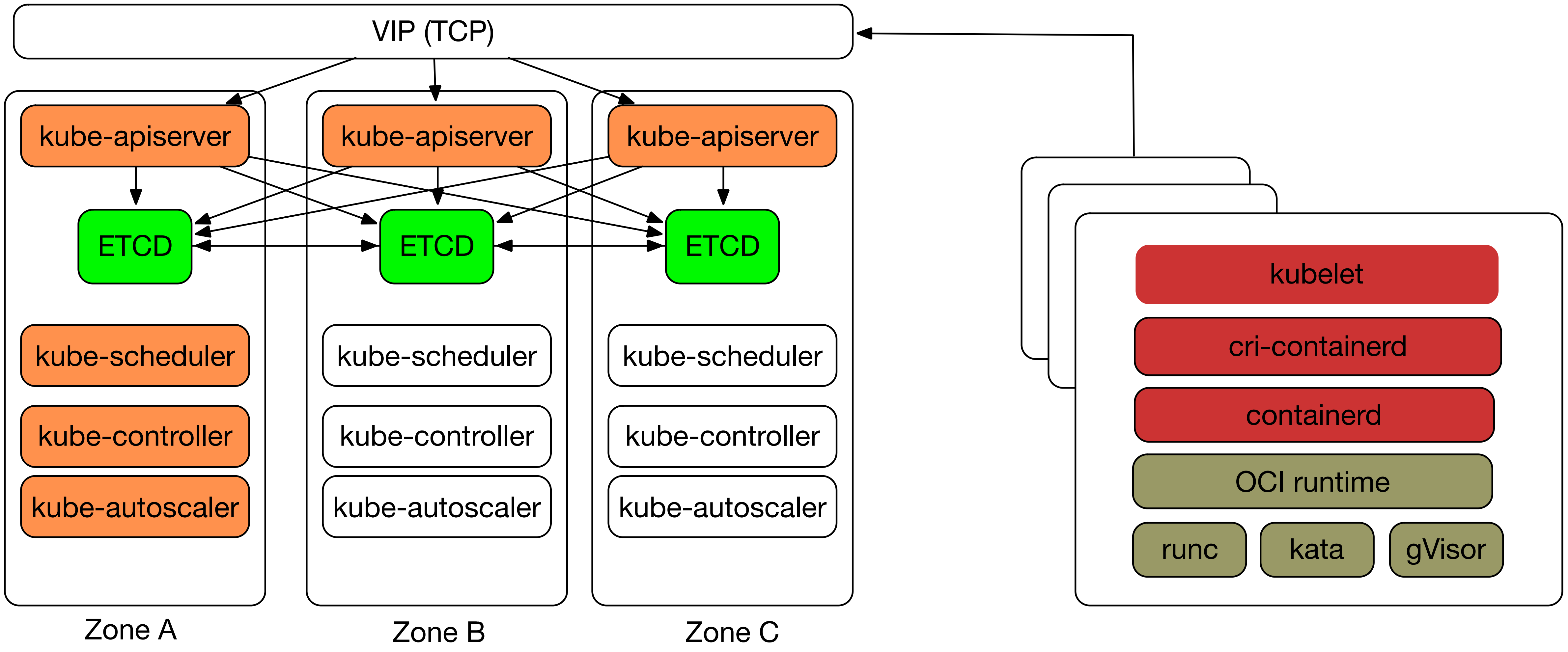
Eric Lin (Alibaba Cloud)
Elastic Container Instance product team
Kubernetes & Containerd contributor

Agenda

- Views of Isolation in k8s
- Image isolation in k8s
- Image isolation in containerd
- Future works
- Q/A

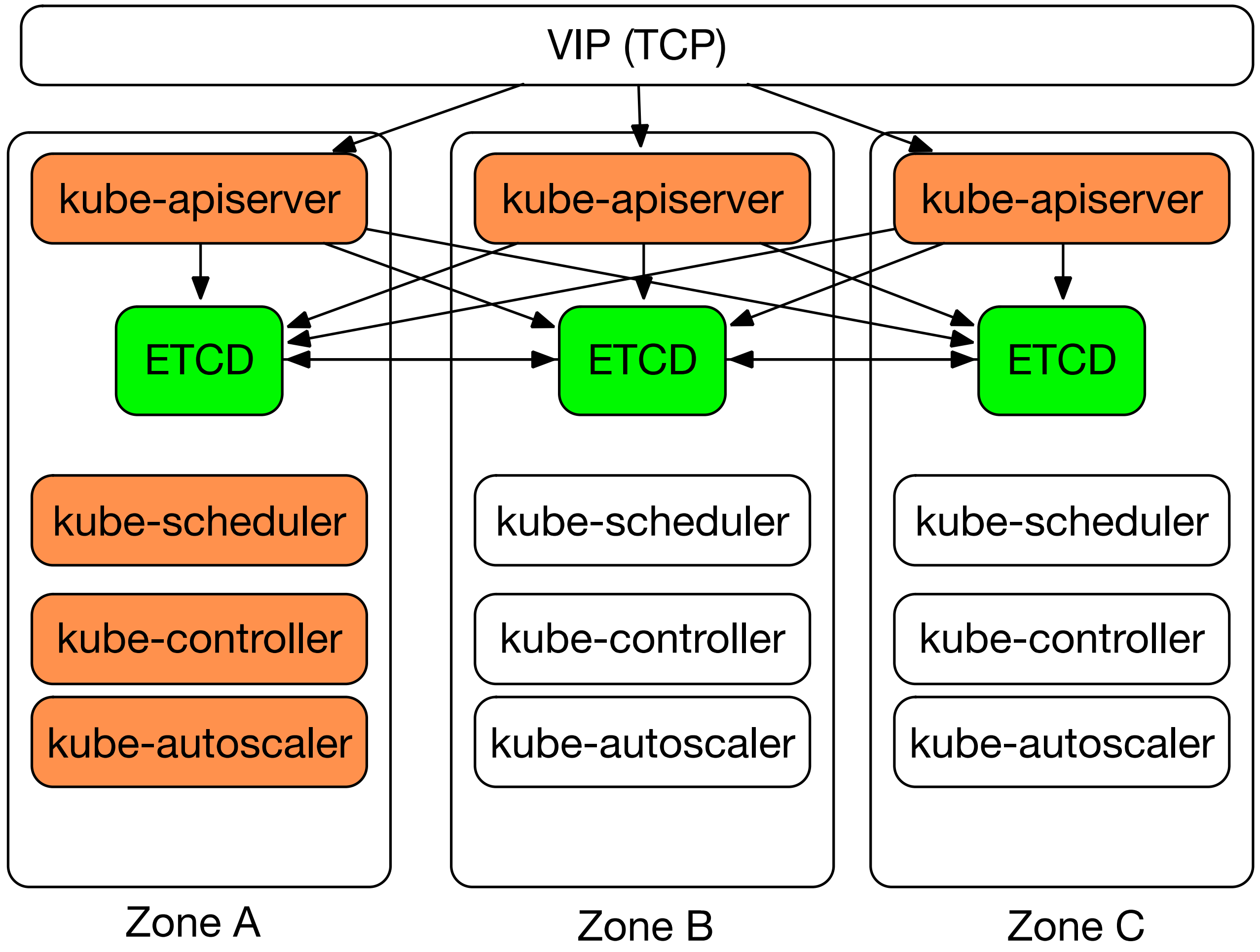
Views of isolation in k8s

KubeCon CloudNativeCon Europe 2019



API View Isolation

- Namespace
  - Resources partition
- Authorization
  - Request Allowed/Denied
- Admission Controller
  - Request filter



ContainerRuntime View Isolation

- Static Data Isolation
  - Image
  - Container
  - Snapshot/Rootfs
- Runtime Isolation
  - runc
  - runv/kata
  - gVisor

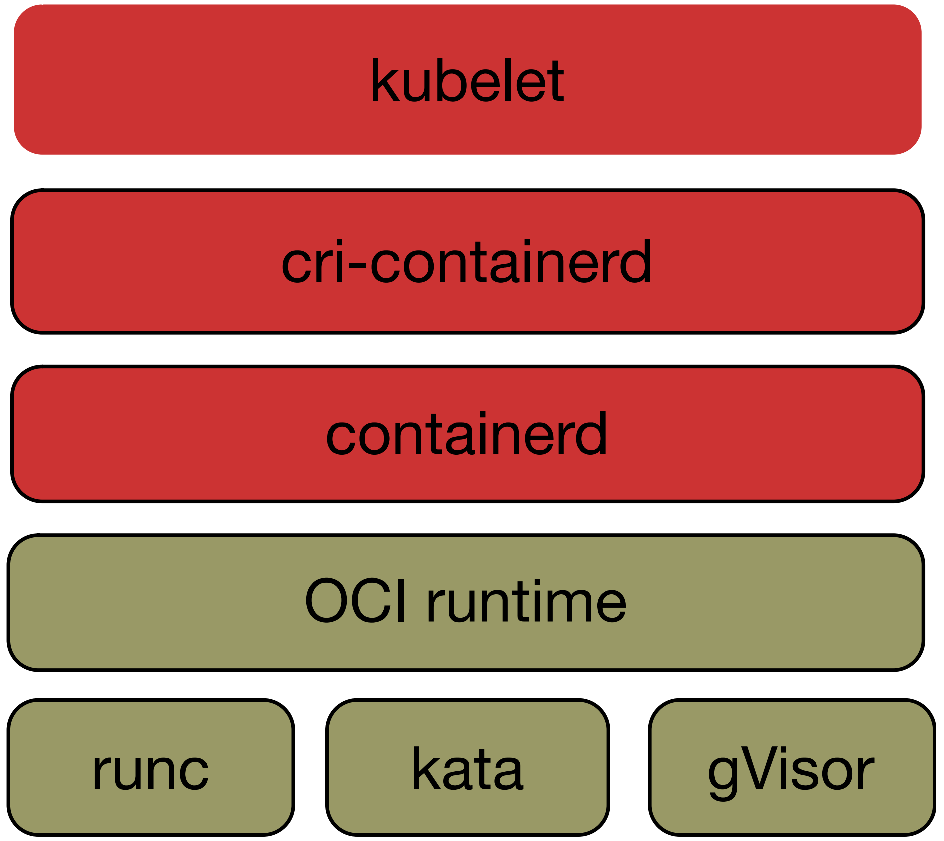
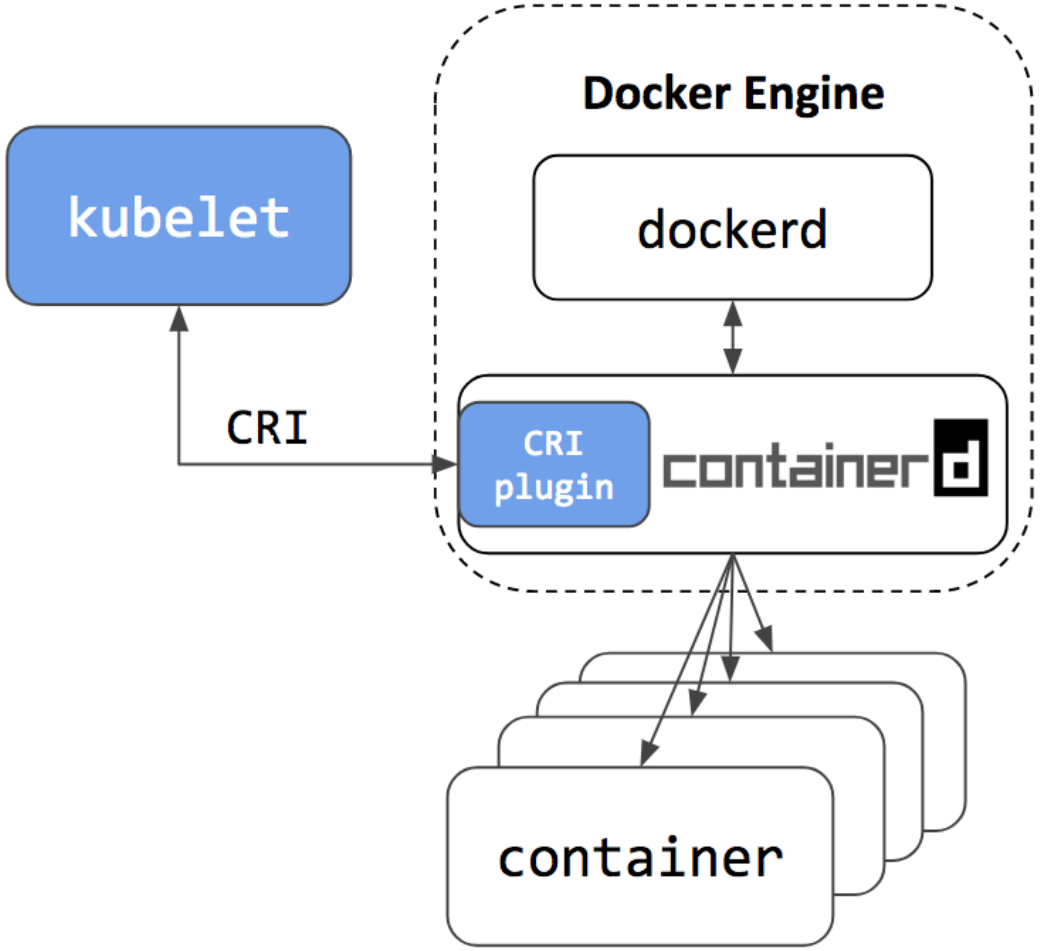


Image isolation in k8s

- Weak Image Isolation
  - Not under control of the API View
  - Shared across the cluster
  - Little protection for pulled private images (AlwaysPull admission)

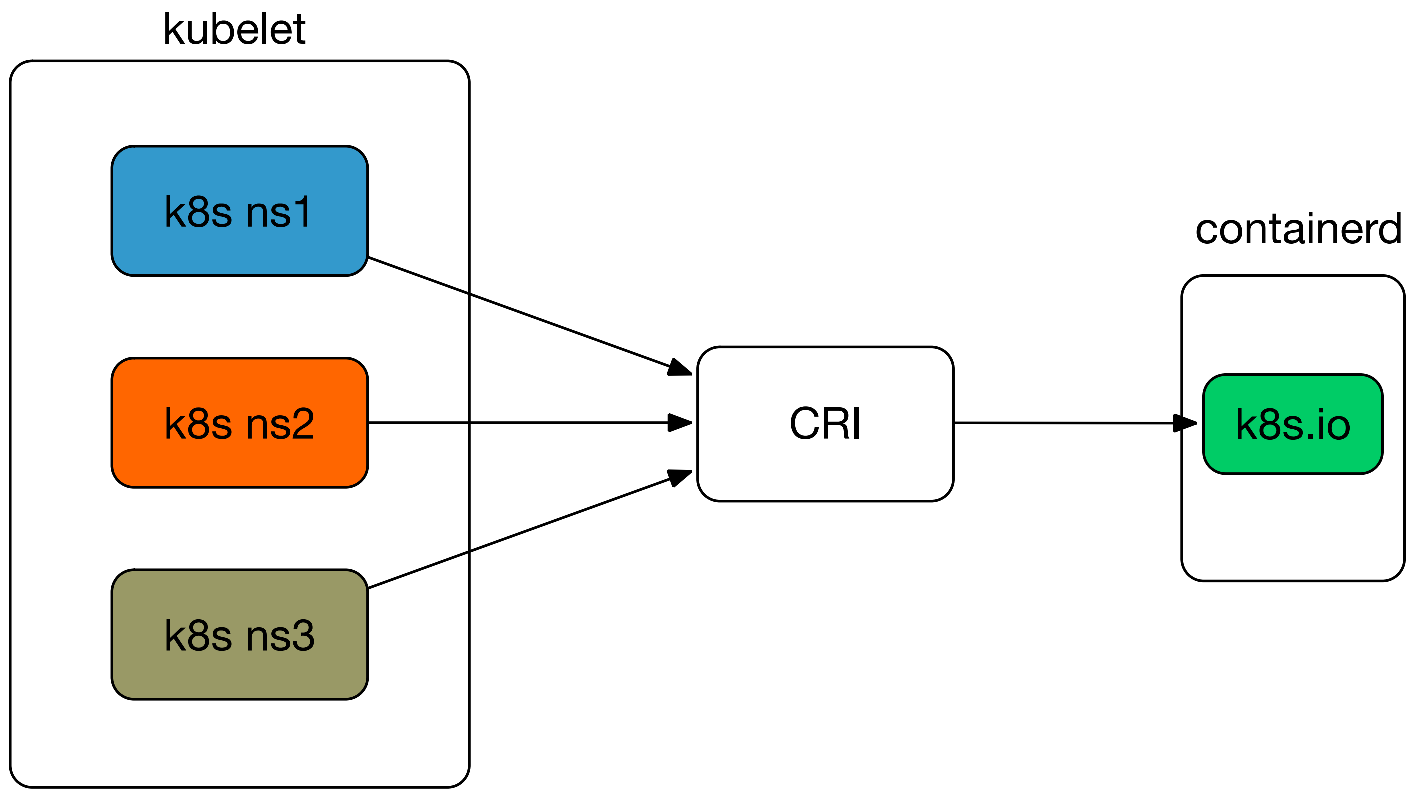
Image management in k8s

- K8s doesn't manage images itself
  - Container Runtime Interface API
    - PullImage()/LoadImage()
    - RemoveImage()
    - ListImages()/StatusImage()



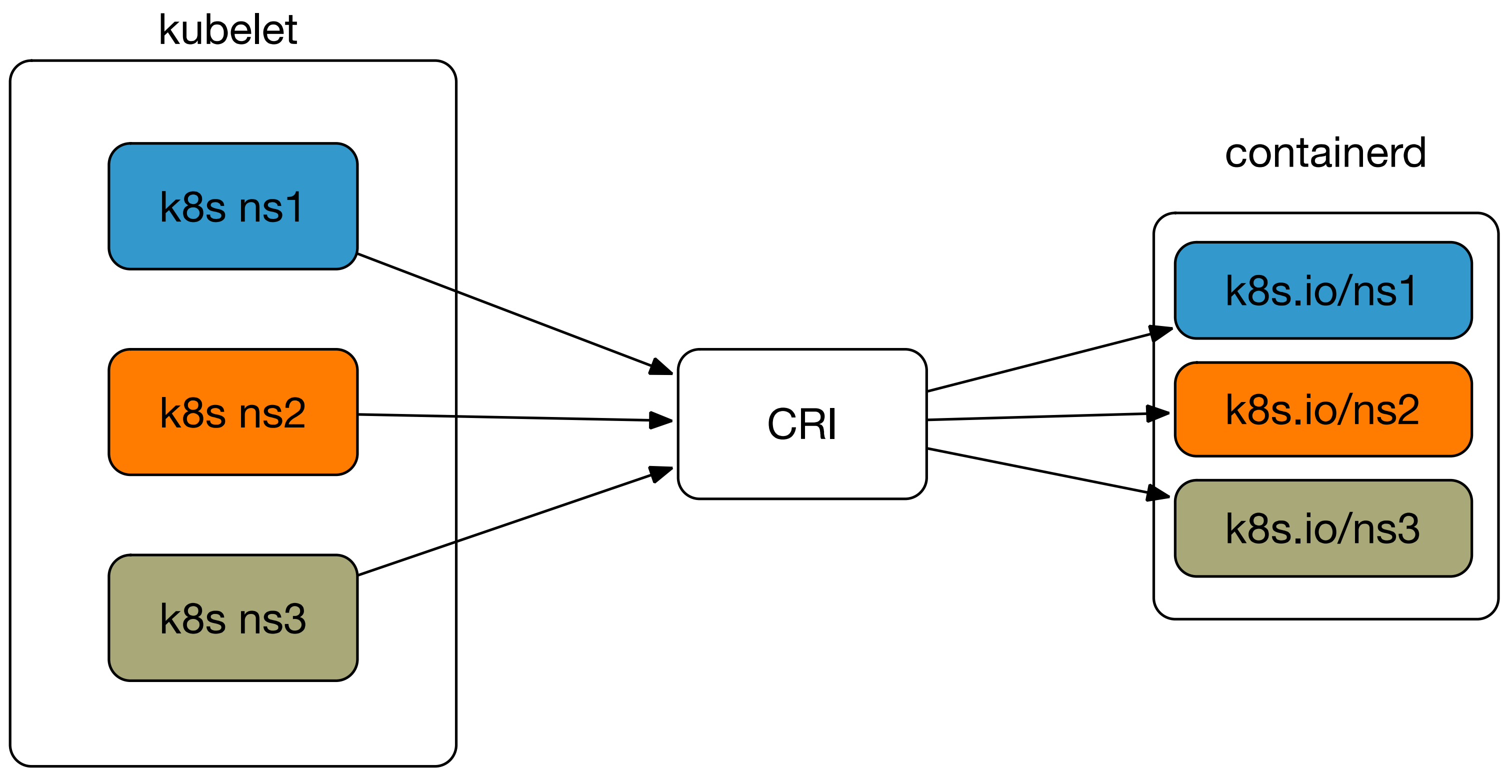
Containerd Multi-Tenancy

- Containerd offers a fully namespaced API
- Many container engine built on containerd
  - Docker
  - Pouch
  - CRI-Containerd
  - …



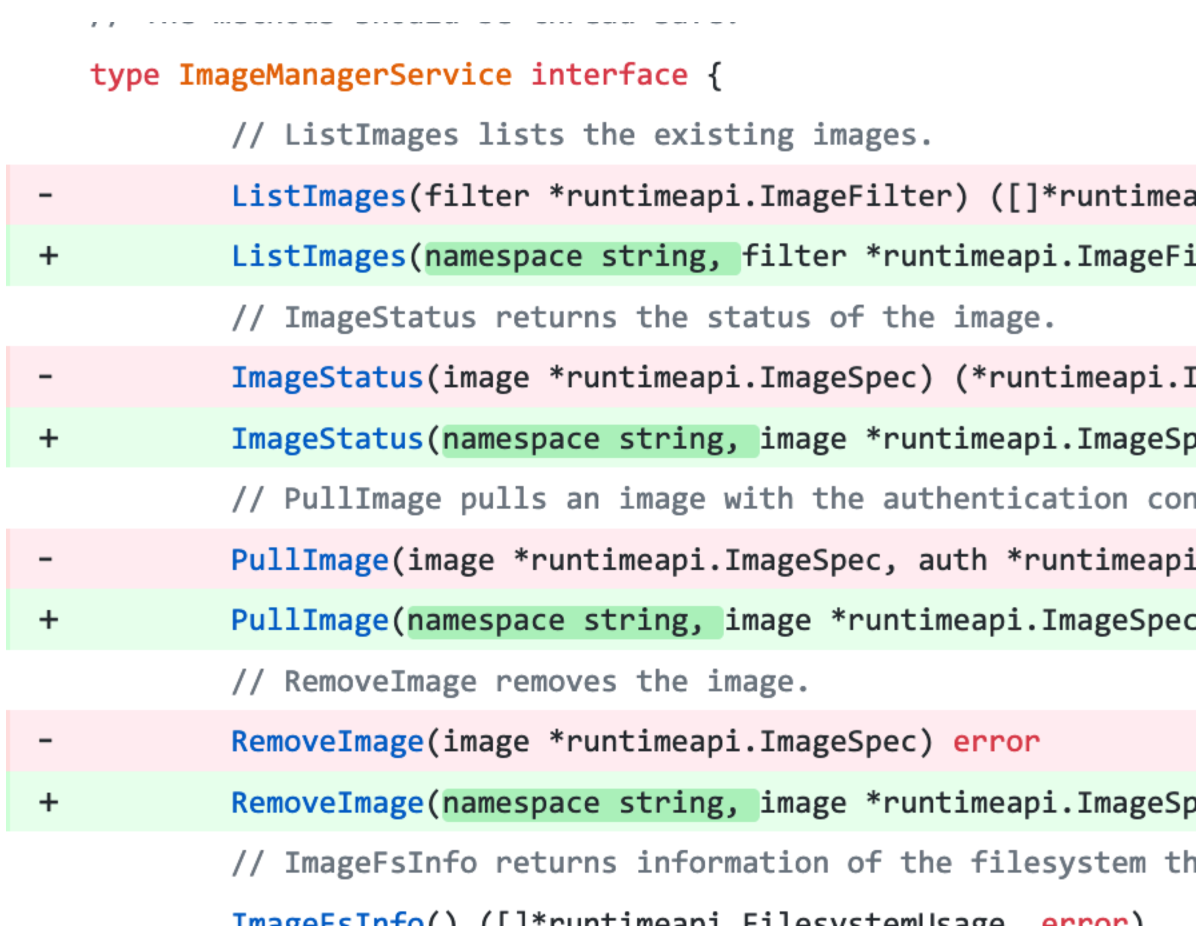
Reenforce it!

- Utilize containerd namespaces
  - [WIP] pass namespace info to CRI #73517

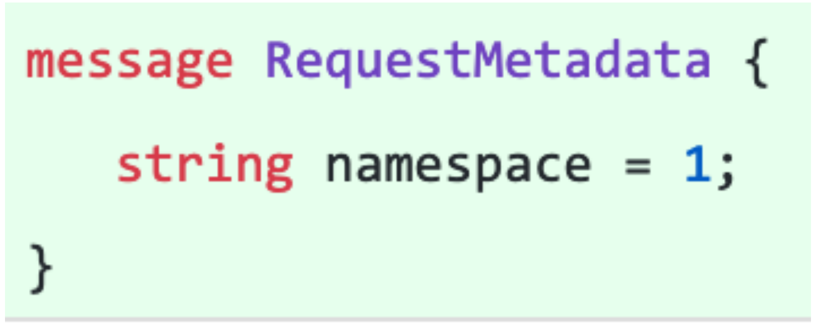


Reenforce it!

- Add the namespace dimension to CRI
  - [WIP] pass namespace info to CRI #73517

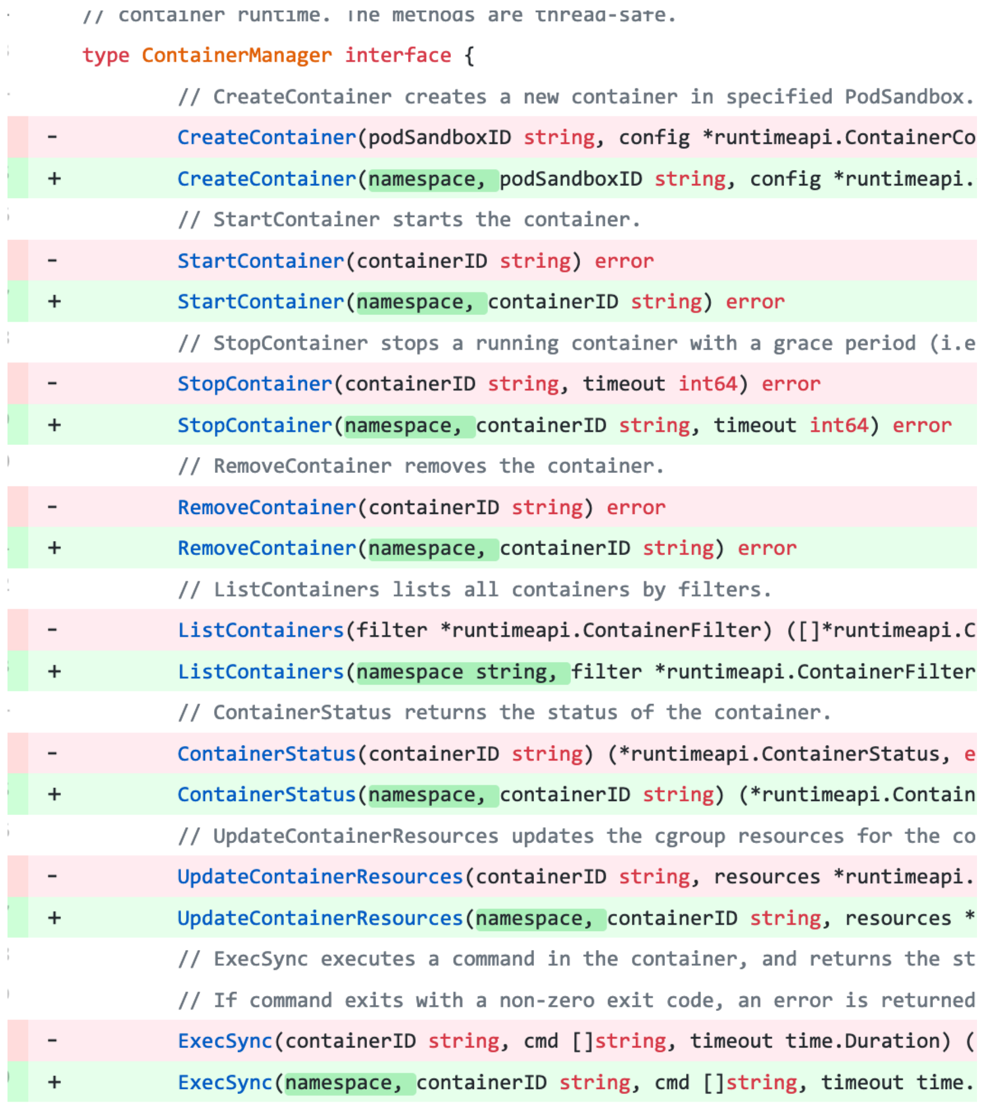


```
type ImageManagerService interface {
        // ListImages lists the existing images.
-       ListImages(filter *runtimeapi.ImageFilter) ([]*runtimea
+       ListImages(namespace string, filter *runtimeapi.ImageFi
        // ImageStatus returns the status of the image.
-       ImageStatus(image *runtimeapi.ImageSpec) (*runtimeapi.I
+       ImageStatus(namespace string, image *runtimeapi.ImageSp
        // PullImage pulls an image with the authentication con
-       PullImage(image *runtimeapi.ImageSpec, auth *runtimeapi
+       PullImage(namespace string, image *runtimeapi.ImageSpec
        // RemoveImage removes the image.
-       RemoveImage(image *runtimeapi.ImageSpec) error
+       RemoveImage(namespace string, image *runtimeapi.ImageSp
        // ImageFsInfo returns information of the filesystem th
        ImageFsInfo() ([]*runtimeapi.FilesystemUsage, error)
```

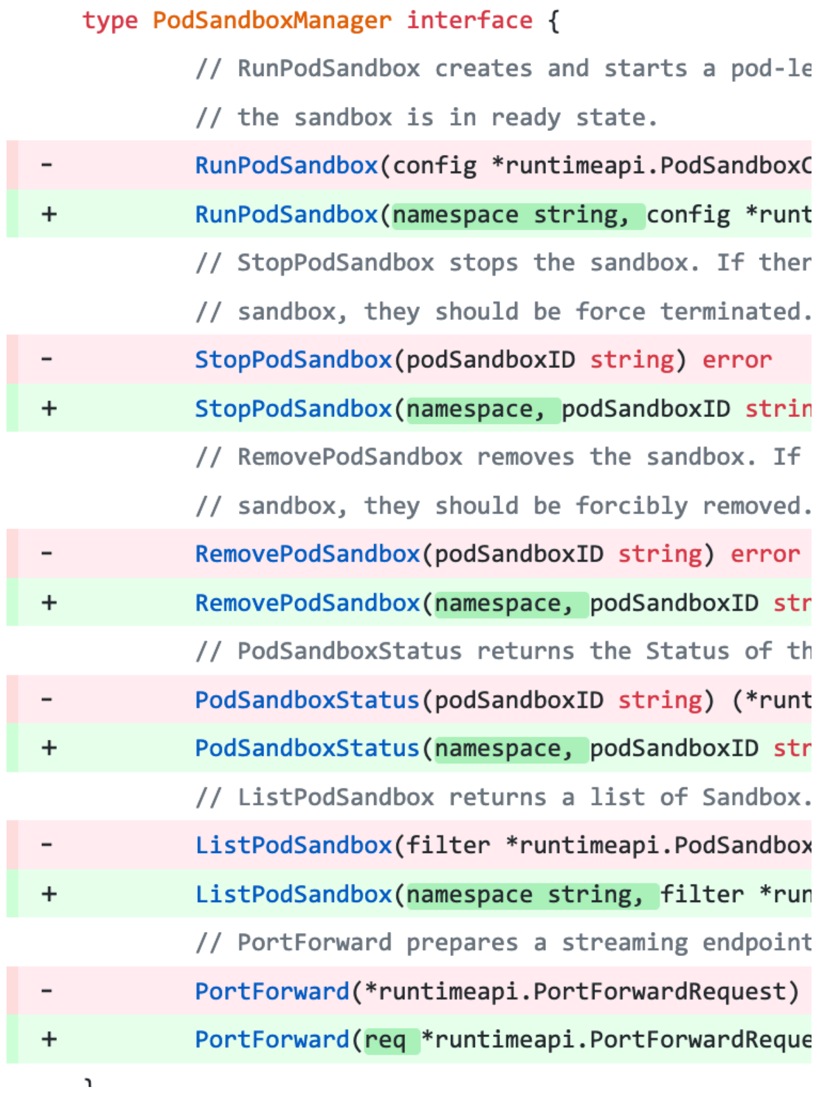


```
message RequestMetadata {
    string namespace = 1;
}
```

Reenforce it!



```diff
     // container runtime. The methods are thread-safe.
     type ContainerManager interface {
             // CreateContainer creates a new container in specified PodSandbox.
-            CreateContainer(podSandboxID string, config *runtimeapi.ContainerCo
+            CreateContainer(namespace, podSandboxID string, config *runtimeapi.
             // StartContainer starts the container.
-            StartContainer(containerID string) error
+            StartContainer(namespace, containerID string) error
             // StopContainer stops a running container with a grace period (i.e
-            StopContainer(containerID string, timeout int64) error
+            StopContainer(namespace, containerID string, timeout int64) error
             // RemoveContainer removes the container.
-            RemoveContainer(containerID string) error
+            RemoveContainer(namespace, containerID string) error
             // ListContainers lists all containers by filters.
-            ListContainers(filter *runtimeapi.ContainerFilter) ([]*runtimeapi.C
+            ListContainers(namespace string, filter *runtimeapi.ContainerFilter
             // ContainerStatus returns the status of the container.
-            ContainerStatus(containerID string) (*runtimeapi.ContainerStatus, e
+            ContainerStatus(namespace, containerID string) (*runtimeapi.Contain
             // UpdateContainerResources updates the cgroup resources for the co
-            UpdateContainerResources(containerID string, resources *runtimeapi.
+            UpdateContainerResources(namespace, containerID string, resources *
             // ExecSync executes a command in the container, and returns the st
             // If command exits with a non-zero exit code, an error is returned
-            ExecSync(containerID string, cmd []string, timeout time.Duration) (
+            ExecSync(namespace, containerID string, cmd []string, timeout time.
```



```diff
     type PodSandboxManager interface {
             // RunPodSandbox creates and starts a pod-le
             // the sandbox is in ready state.
-            RunPodSandbox(config *runtimeapi.PodSandboxC
+            RunPodSandbox(namespace string, config *runt
             // StopPodSandbox stops the sandbox. If ther
             // sandbox, they should be force terminated.
-            StopPodSandbox(podSandboxID string) error
+            StopPodSandbox(namespace, podSandboxID strin
             // RemovePodSandbox removes the sandbox. If
             // sandbox, they should be forcibly removed.
-            RemovePodSandbox(podSandboxID string) error
+            RemovePodSandbox(namespace, podSandboxID str
             // PodSandboxStatus returns the Status of th
-            PodSandboxStatus(podSandboxID string) (*runt
+            PodSandboxStatus(namespace, podSandboxID str
             // ListPodSandbox returns a list of Sandbox.
-            ListPodSandbox(filter *runtimeapi.PodSandbox
+            ListPodSandbox(namespace string, filter *run
             // PortForward prepares a streaming endpoint
-            PortForward(*runtimeapi.PortForwardRequest)
+            PortForward(req *runtimeapi.PortForwardReque
```

Issues

- Image/Container garbage collect in k8s
  - Iterate all namespaces when GC-ing
- Complexity increased
  - Iterate resources with namespace
  - Namespace lifecycle
- Still WIP in upstream

Problem solved?

- Image management in containerd
  - Images are somehow shared across namespaces
- Need to dive in containerd
  - Show you a demo

Image in containerd (Demo)

1. Pull test image `ctr -n ns1 images pull docker.io/library/python:latest`

2. Get content sha256 from /var/lib/containerd/

   io.containerd.content.v1.content/blobs/sha256/

3. git clone github.com/linxiulei/fake_registry.git

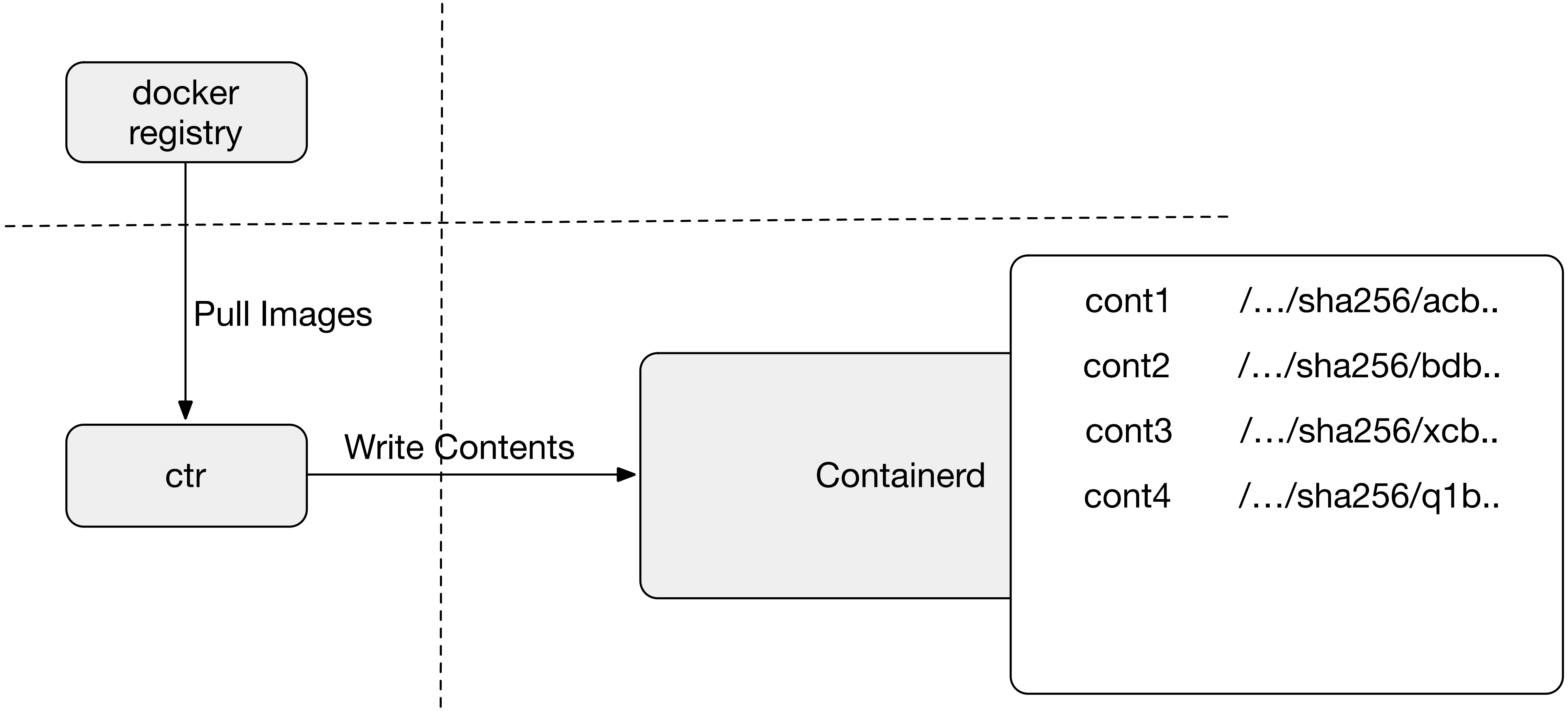
4. Modify the config and `go run server.go`

5. Pull fake image `ctr -n ns2 images pull localhost:8084/library/test:latest`

Image in containerd

- Image
  - content0 (config.json)
    - metadata of content1 (compressed/uncompressed size, digest)
    - metadata of content2
    - …
  - content1 (layer0.tar.gz)
  - content2 (layer1.tar.gz)
  - ….

How images were pulled



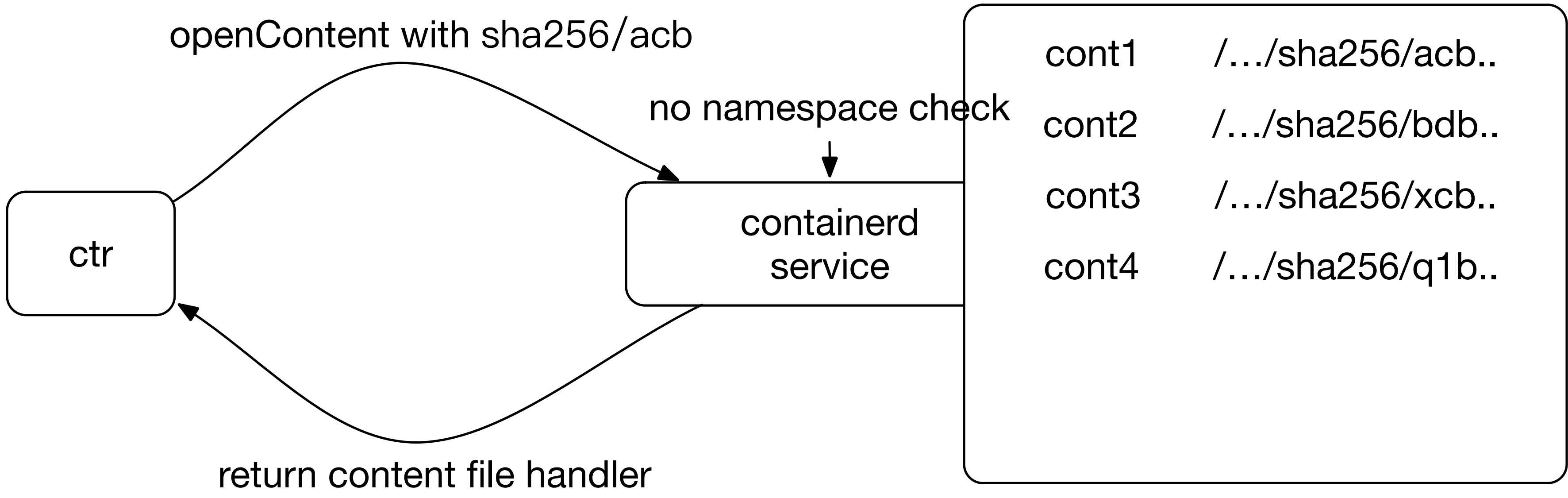
How images were pulled

Retrieve the manifest

```
1. curl https://registry-1.docker.io/v2/library/busybox/manifests/latest \
       -H "Authorization: Bearer $token”

< Content-Type: application/vnd.docker.distribution.manifest.list.v2+json
< Docker-Content-Digest:
sha256:f7891ea6bcd0ce73aa5aa5080f1163c96e74538d80c63baa3d18c33016be87f5

2. curl https://registry-1.docker.io/v2/library/busybox/manifests/
sha256:f7891ea6bcd0ce73aa5aa5080f1163c96e74538d80c63baa3d18c33016be87f5 -H "Authorization:
Bearer $token"

{ "schemaVersion": 2, "mediaType": "application/vnd.docker.distribution.manifest.list.v2+json", "manifests":
[ { "mediaType": "application/vnd.docker.distribution.manifest.v2+json", "size": 1370, "digest":
"sha256:af0c785e711e34f8d0ba5a346e9a7900f6557d9cd96a0e7d0ea6e51adba6e797", "platform":
{ "architecture": "amd64", "os": "linux"
```

Back to problem

- image management in containerd
  - contents of the image are shared across namespaces by default for better performance
  - There is no namespace boundary when using contentWriter API (for better performance)



Reenforce it!

- Able to specify the sharing policy across namespaces [shared/ isolated]
  - **metadata: define content sharing policy #2889 [MERGED]**

```
# config.toml
[plugins.bolt]
    content_sharing_policy = "isolated"
```

Wrapup

- [CRI] Fully utilize containerd namespaced api
  - **[WIP] pass namespace info to CRI #73517**

- Able to specify the sharing policy across namespaces [shared/isolated]
  - **metadata: define content sharing policy #2889 [MERGED]**

Future works

- Share public images across namespaces
  - nested namespaces
- Image/Container garbage collect in k8s
  - iterate all namespaces when GC-ing
  - namespace lifecycle sync
- Scheduler to aware image-namespace locality
- Convincing the community with image isolation

Q&A

Thank you

linxiulei@gmail.com