What we'll discuss today

Programmatically Interacting with the API

- Kubernetes Objects in Go
- ClientSets
- API Machinery
- Out-of-Cluster vs In-Cluster Authn
- Informers and Caching

Source:
Insert source data here
Insert source data here
Insert source data here

Kinds are represented as Golang structs; usually in a package whose path corresponds to the GVK

pkg/apis/*group*/*version*/types.go

Deployment for example is found in:
k8s.io/kubernetes/apps/v1/types.go

```
// Deployment provides declarative updates for Pods and ReplicaSets.
type Deployment struct {
        metav1.TypeMeta
        // +optional
        metav1.ObjectMeta

        // Specification of the desired behavior of the Deployment.
        // +optional
        Spec DeploymentSpec

        // Most recently observed status of the Deployment.
        // +optional
        Status DeploymentStatus
}
```

Client Libraries

- client-go
- kubernetes-client
  - Python
  - Java
  - dotnet
  - JavaScript
  - Haskell

client-go

What's Included:

- The **kubernetes** package contains the clientset to access Kubernetes API.
- The **discovery** package is used to discover APIs supported by a Kubernetes API server.
- The **dynamic** package contains a dynamic client that can perform generic operations on arbitrary Kubernetes API objects.
- The **plugin/pkg/client/auth** packages contain optional authentication plugins for obtaining credentials from external sources.
- The **transport** package is used to set up auth and start a connection.
- The **tools/cache** package is useful for writing controllers.

01 Compatibility Matrix

	Kubernetes 1.15	Kubernetes 1.16	Kubernetes 1.17
kubernetes-1.15.0	√	+-	+-
kubernetes-1.16.0	+-	√	+-
kubernetes-1.17.0 / v0.17.0	+-	+-	√
HEAD	+-	+-	+-

Source:
https://github.com/kubernetes/client-go/

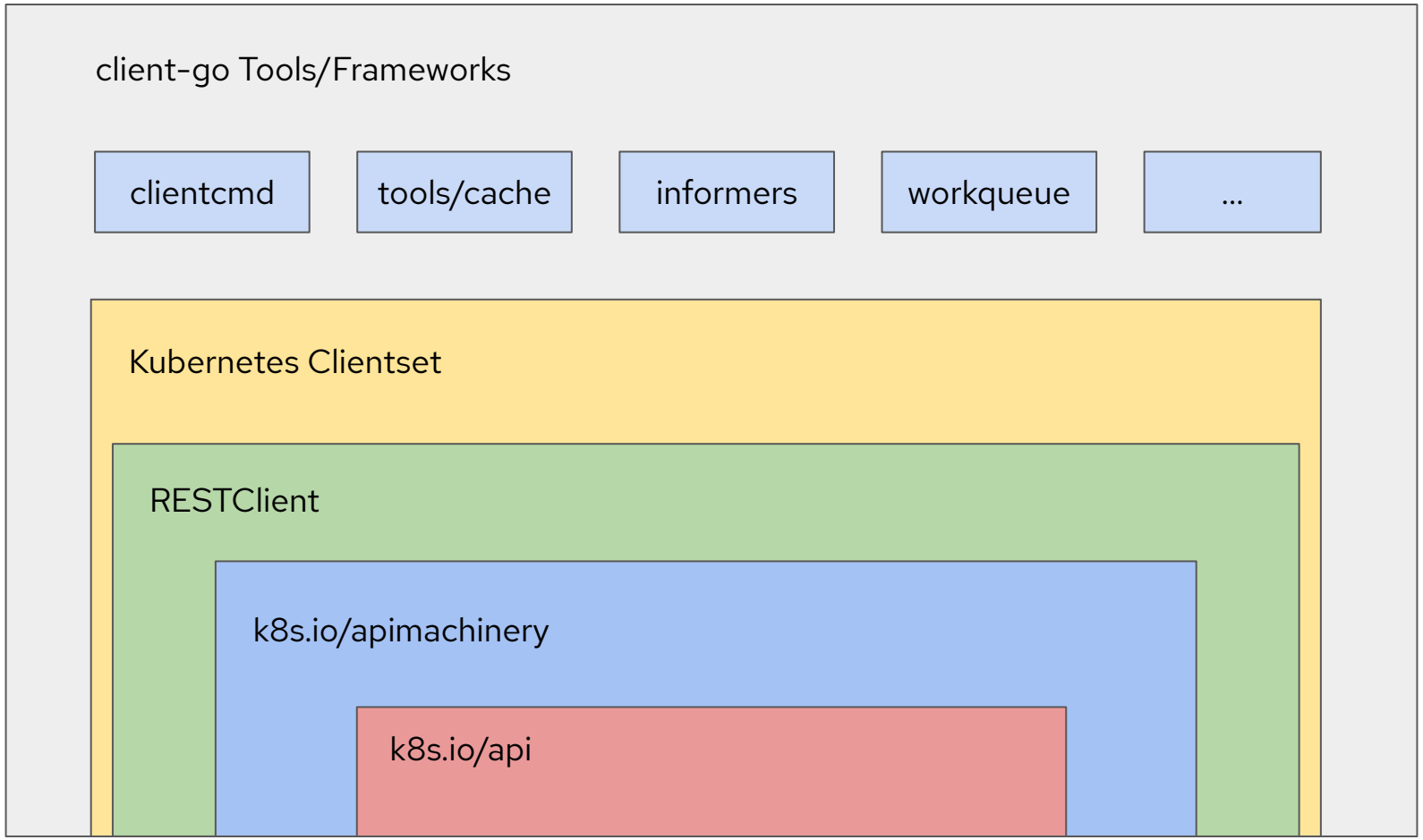
Contents of client-go

Clients

- Clientset
- Dynamic Client
- RESTclient

Utilities for Writing Controllers

- Workqueue
- Informers
- Shared Informers


client-go Tools/Frameworks
clientcmd
tools/cache
informers
workqueue
...
Kubernetes Clientset
RESTClient
k8s.io/apimachinery
k8s.io/api

In-Cluster vs Out-of-Cluster Authn

```
import (
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
    "k8s.io/client-go/tools/clientcmd"
    "k8s.io/client-go/kubernetes"
)

kubeconfig = flag.String("kubeconfig", "~/.kube/config", "kubeconfig file")
flag.Parse()
config, err := clientcmd.BuildConfigFromFlags("", *kubeconfig)
clientset, err := kubernetes.NewForConfig(config)

pod, err := clientset.CoreV1().Pods("book").Get("example", metav1.GetOptions{})
```

- **clientcmd** to read and parse the kubeconfig
- **kubernetes** package for client sets for Kubernetes resources
- /var/run/secrets/kubernetes.io/serviceaccount
- Utilize builder pattern to get example Pod from book Namespace

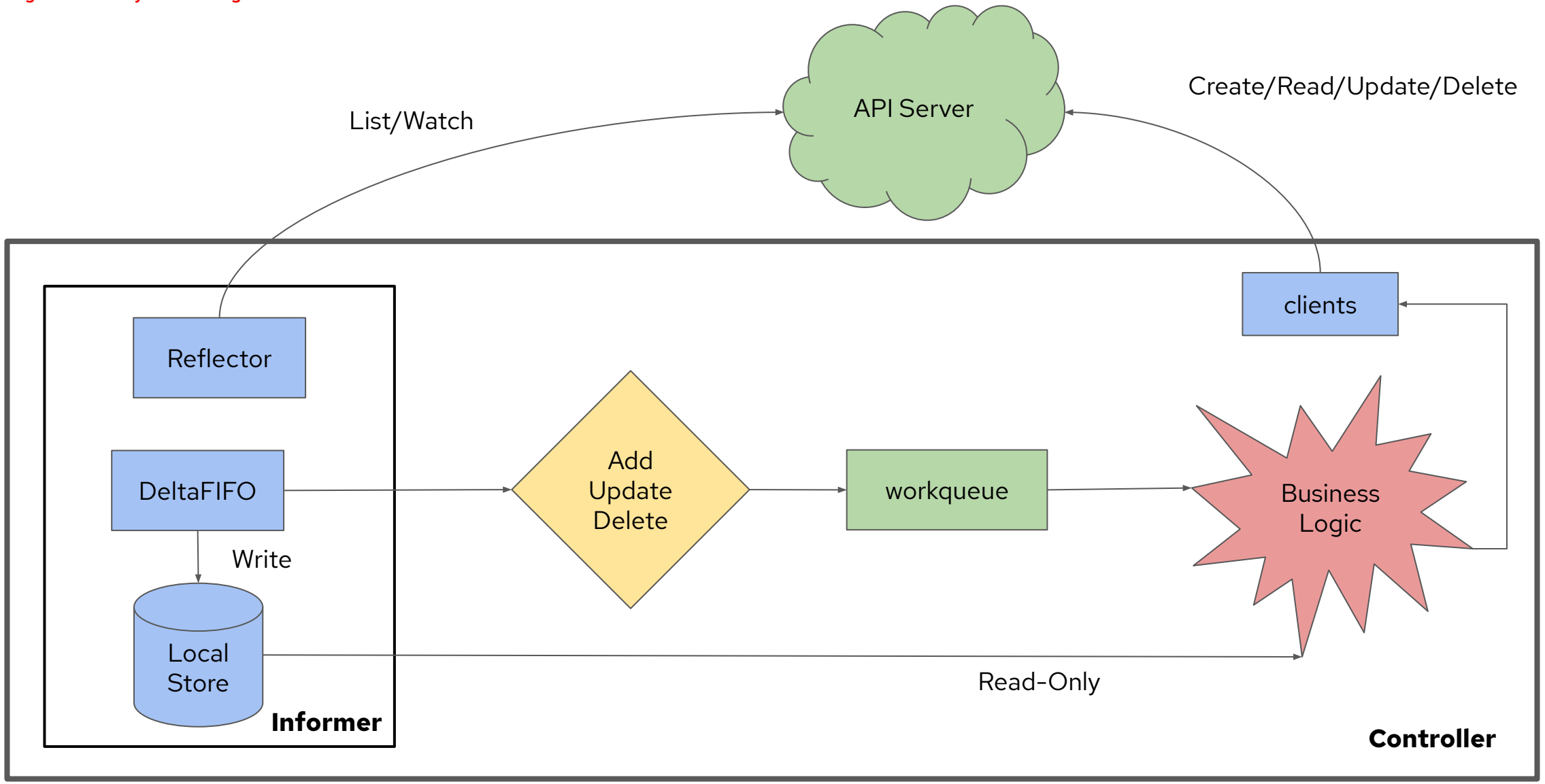
Source:

In-Cluster vs Out-of-Cluster Authn

```
config, err := rest.InClusterConfig()
if err != nil {
    // fallback to kubeconfig
    kubeconfig := filepath.Join("~", ".kube", "config")
    if envvar := os.Getenv("KUBECONFIG"); len(envvar) >0 {
        kubeconfig = envvar
    }
    config, err = clientcmd.BuildConfigFromFlags("", kubeconfig)
    if err != nil {
        fmt.Printf("The kubeconfig cannot be loaded: %v\n" , err
        os.Exit(1)
   }
}
```

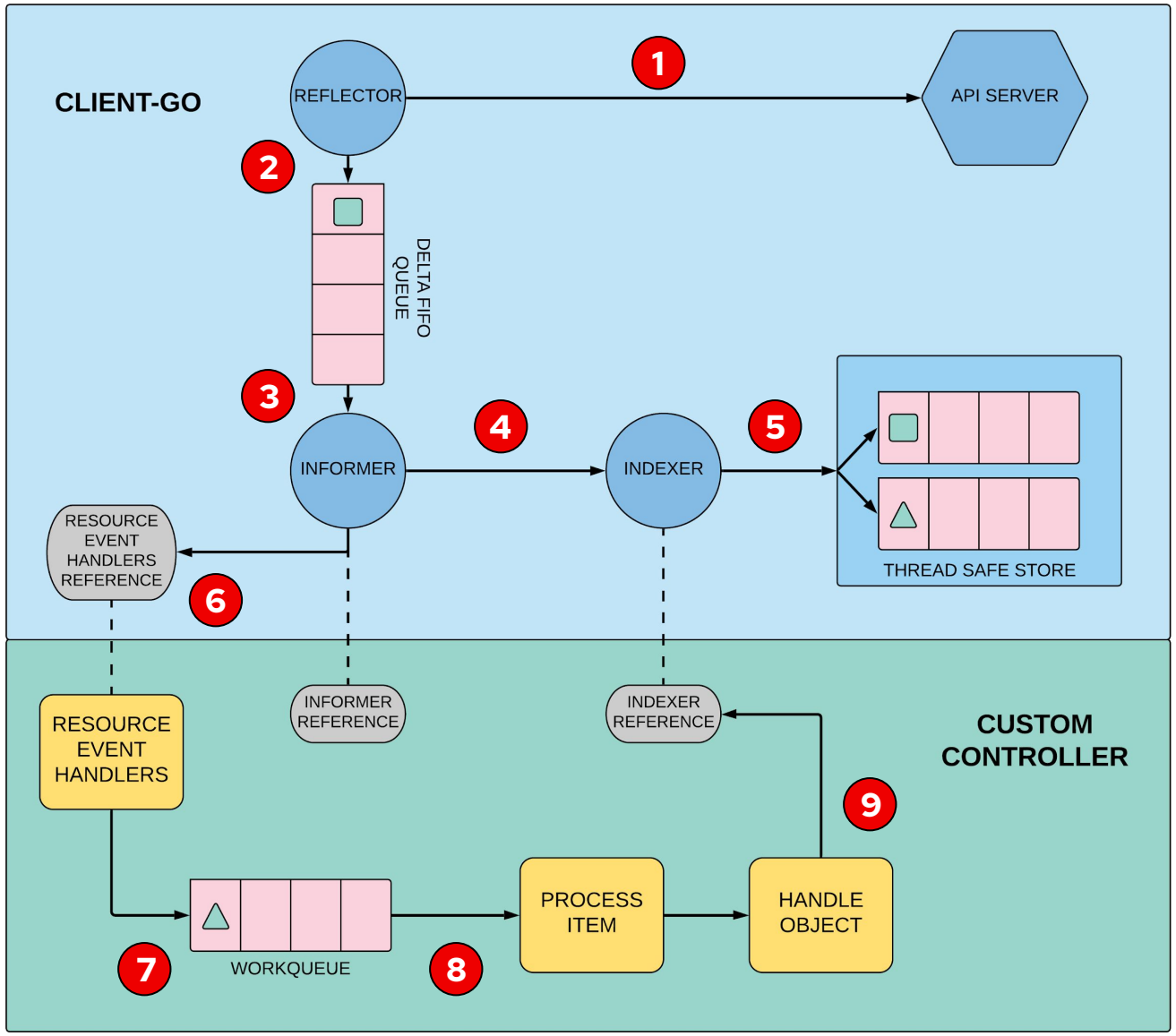
- When running a binary inside of a pod in a cluster kubelet will automatically mount a ServiceAccount into the container; /var/run/secrets/kubernetes.io/serviceaccount
- If we are not in the cluster i.e. able to get the default service account, we handle the error, in this case we fallback to looking for the kubeconfig

Source:
Insert source data here
Insert source data here
Insert source data here


API Server
List/Watch
Create/Read/Update/Delete
clients
Reflector
DeltaFIFO
Write
Local
Store
Informer
Add
Update
Delete
workqueue
Business
Logic
Read-Only
Controller

Informers and Caching

1. List and Watch
2. Add Object
3. Pop Object
4. Add Object
5. Store Object and Key
6. Dispatch Event Handler Functions
7. Enqueue Object Key
8. Get Key
9. Get Object for Key



Red Hat