KUBERNETES LIBRARY 開發實戰 WITH CLIENT-GO

SDN X CLOUD NATIVE MEETUP #6

WHO AM I

CHEN YUN CHEN (ALEX)

- ME@YUNCHEN.TW
- BLOG.YUNCHEN.TW

EXPERIENCE

- SOFTWARE ENGINEER AT LINKER NETWORKS

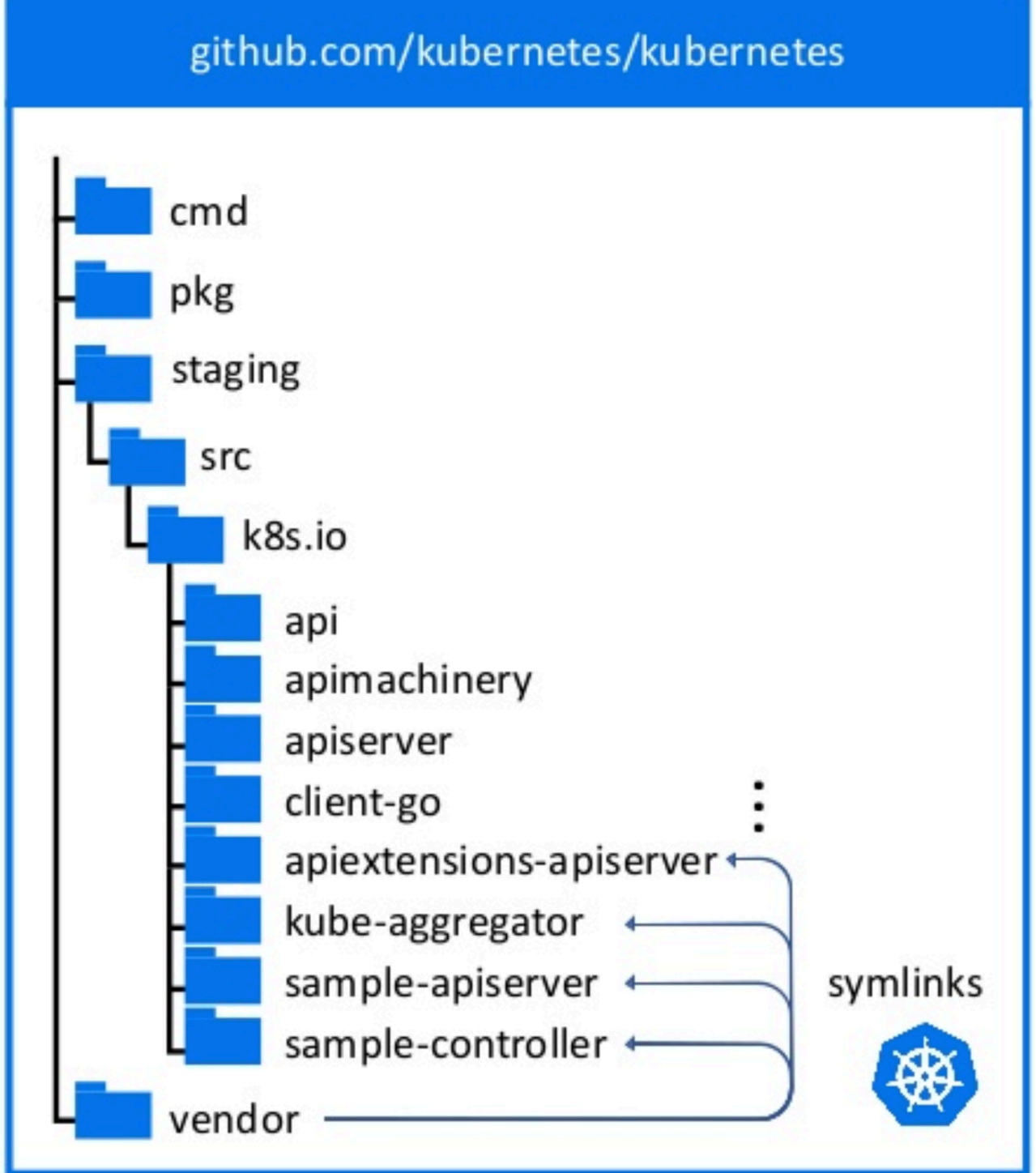
什麼是CLIENT-GO?

想像是把KUBECTL拆成各種工具包
GO語言只要拿你所需要的工具
就可以管理集群上想監控的部分

其它語言 PYTHON JAVA
HTTPS://GITHUB.COM/KUBERNETES-CLIENT

誰會需要 CLIENT-GO ?

- DEBUG
- 產品/專案本身特色即為USER提供CLUSTER操作
- 哪些專案也用CLIENT-GO:
  -  ETCD-OPERATOR
  -  PROMETHEUS-OPERATOR



API : RESOURCES OBJECT
APIMACHINERY : OPTIONS OBJECT

- K8S.IO/KUBERNETES
- K8S.IO/CLIENT-GO
- K8S.IO/APISERVER

HTTPS://GITHUB.COM/KUBERNETES

API/CORE/V1/TYPES.GO

HTTPS://GITHUB.COM/KUBERNETES/API/BLOB/MASTER/CORE/V1/TYPES.GO

```
type Volume struct {}
type PersistentVolume struct {}
type PersistentVolumeClaim struct {}
type Container struct {}
type Pod struct {}
type Service struct {}
type Node struct {}
type Namespace struct {}
type ConfigMap struct {}
...
```

APIMACHINERY/PKG/APIS/META/V1/TYPES.GO

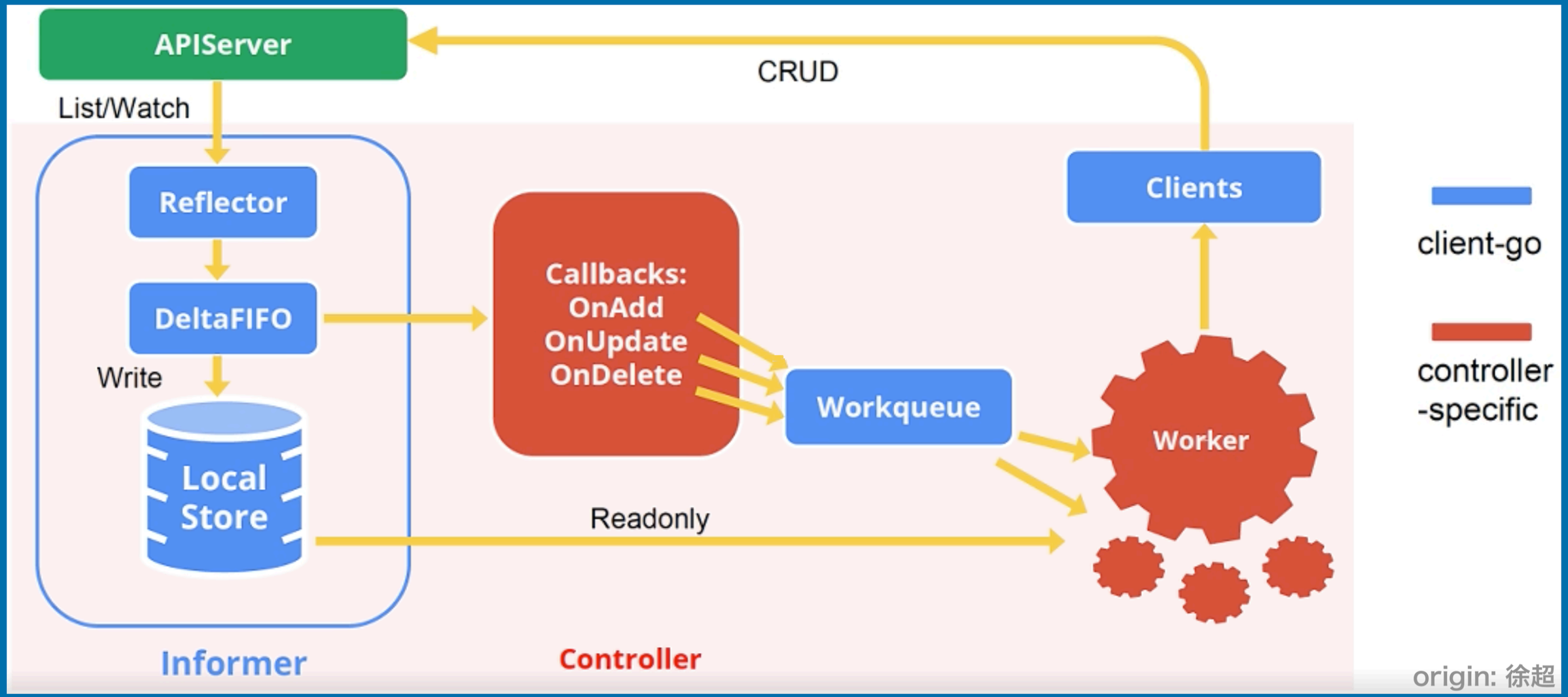
HTTPS://GITHUB.COM/KUBERNETES/APIMACHINERY/BLOB/MASTER/PKG/APIS/META/V1/TYPES.GO

```
type ListOptions struct {}
type GetOptions struct {}
type DeleteOptions struct {}
type ExportOptions struct {}
...
```

APIMACHINERY/PKG/APIS/META/V1/META.GO

HTTPS://GITHUB.COM/KUBERNETES/APIMACHINERY/BLOB/MASTER/PKG/APIS/META/V1/META.GO

```
type Object interface {
    GetNamespace() string
    SetNamespace(namespace string)
    GetName() string
    SetName(name string)
    GetGenerateName() string
    SetGenerateName(name string)
    GetUID() types.UID
    SetUID(uid types.UID)
    GetLabels() map[string]string
    SetLabels(labels map[string]string)
    ...
}
```


APIServer
CRUD
List/Watch
Reflector
DeltaFIFO
Write
Local Store
Informer
Callbacks:
OnAdd
OnUpdate
OnDelete
Workqueue
Clients
Worker
Readonly
Controller
client-go
controller
-specific
origin: 徐超

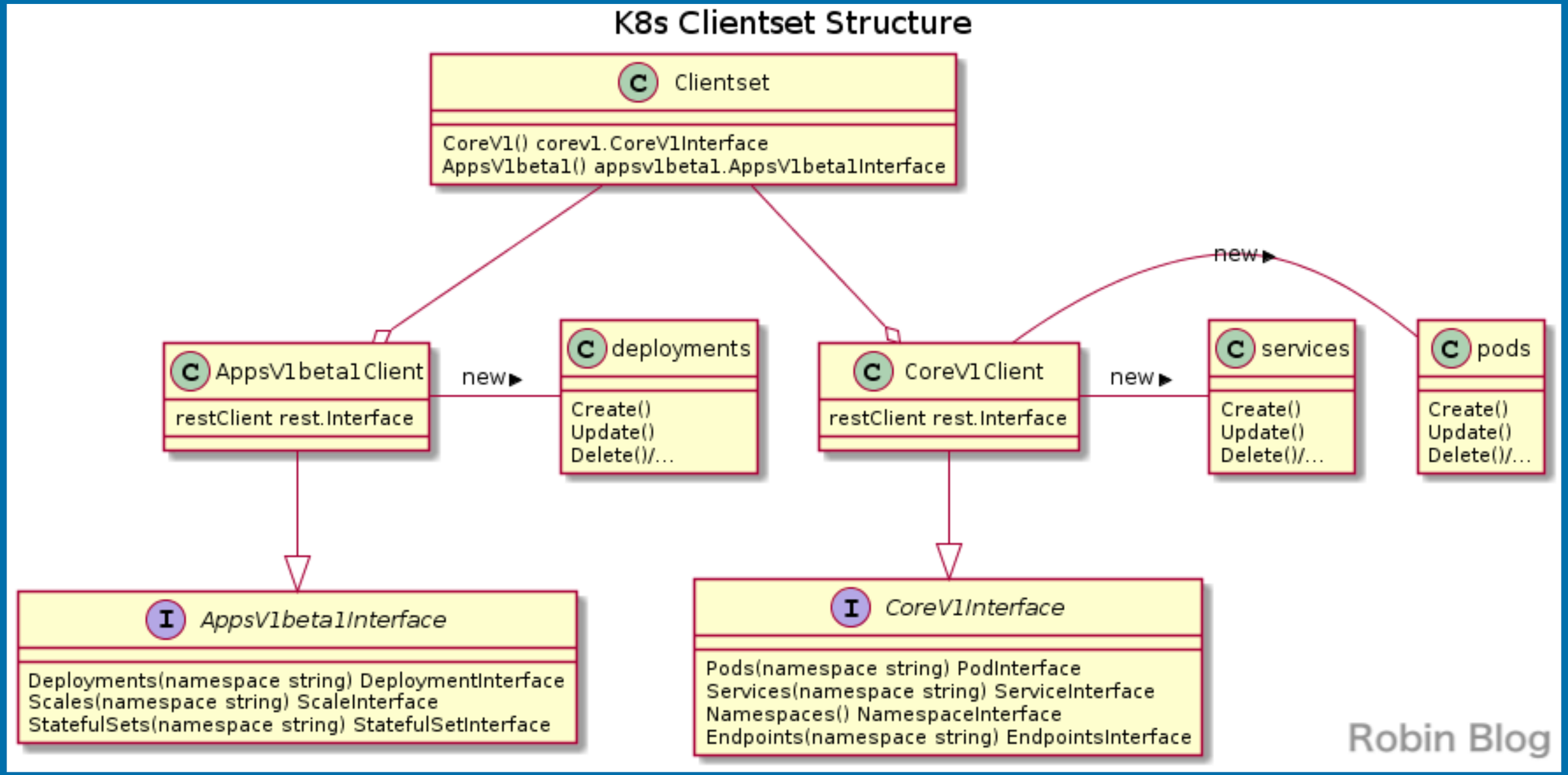
CLIENTS COMPONENT TYPE

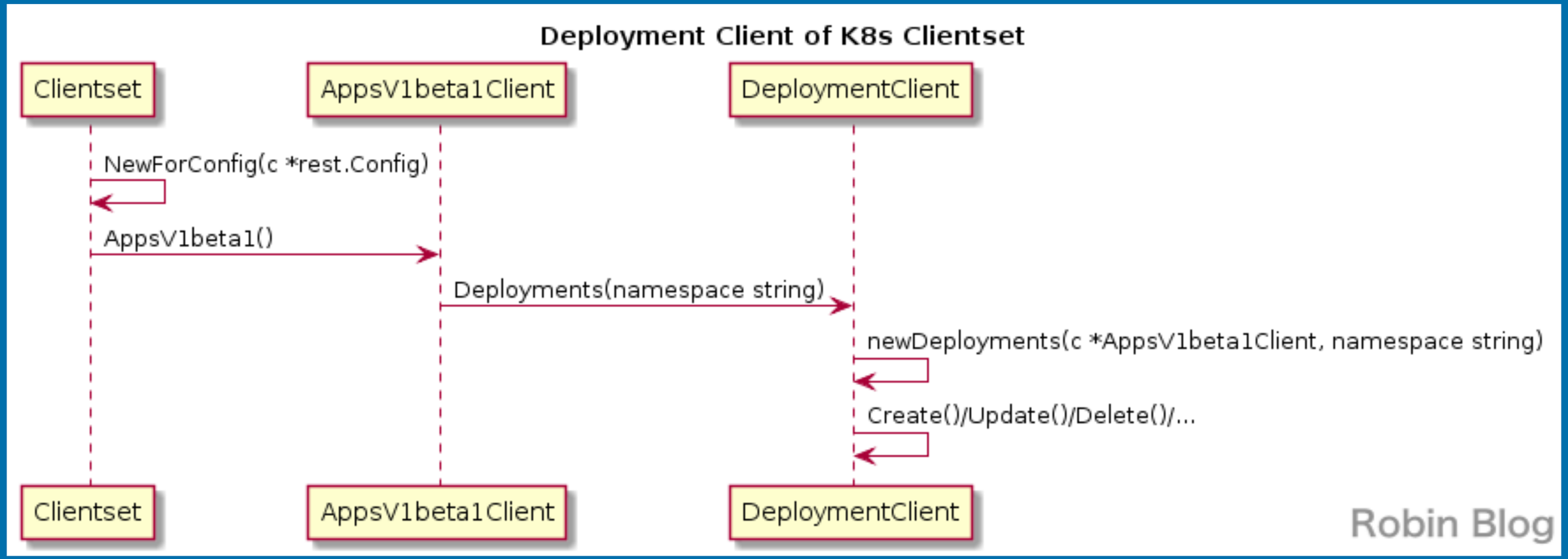
- CLIENTSET `return struct{}`

CLIENT-GO/KUBERNETES

CLIENTSET

```
type Clientset struct {
    *discovery.DiscoveryClient
    admissionregistrationV1alpha1 *admissionregistrationv1alpha1.AdmissionregistrationV1alpha1Client
    appsV1                        *appsv1.AppsV1Client
    authenticationV1              *authenticationv1.AuthenticationV1Client
    autoscalingV2beta1            *autoscalingv2beta1.AutoscalingV2beta1Client
    batchV1                       *batchv1.BatchV1Client
    batchV1beta1                  *batchv1beta1.BatchV1beta1Client
    certificatesV1beta1           *certificatesv1beta1.CertificatesV1beta1Client
    coreV1                        *corev1.CoreV1Client
    eventsV1beta1                 *eventsv1beta1.EventsV1beta1Client
    extensionsV1beta1             *extensionsv1beta1.ExtensionsV1beta1Client
    networkingV1                  *networkingv1.NetworkingV1Client
    policyV1beta1                 *policyv1beta1.PolicyV1beta1Client
    schedulingV1alpha1            *schedulingv1alpha1.SchedulingV1alpha1Client
    settingsV1alpha1              *settingsv1alpha1.SettingsV1alpha1Client
    storageV1                     *storagev1.StorageV1Client
    ...
}
```


K8s Clientset Structure
Clientset
CoreV1() corev1.CoreV1Interface
AppsV1beta1() appsv1beta1.AppsV1beta1Interface
AppsV1beta1Client
restClient rest.Interface
new
deployments
Create()
Update()
Delete()/...
CoreV1Client
restClient rest.Interface
new
services
Create()
Update()
Delete()/...
new
pods
Create()
Update()
Delete()/...
AppsV1beta1Interface
Deployments(namespace string) DeploymentInterface
Scales(namespace string) ScaleInterface
StatefulSets(namespace string) StatefulSetInterface
CoreV1Interface
Pods(namespace string) PodInterface
Services(namespace string) ServiceInterface
Namespaces() NamespaceInterface
Endpoints(namespace string) EndpointsInterface
Robin Blog


Deployment Client of K8s Clientset
Clientset
AppsV1beta1Client
DeploymentClient
NewForConfig(c *rest.Config)
AppsV1beta1()
Deployments(namespace string)
newDeployments(c *AppsV1beta1Client, namespace string)
Create()/Update()/Delete()/...
Clientset
AppsV1beta1Client
DeploymentClient
Robin Blog

CLIENTS 從K8S集群外來管理

```
import (
    "os"
    "path/filepath"

    "k8s.io/client-go/kubernetes"
    "k8s.io/client-go/tools/clientcmd"
)

...

// Windows: c:\Users\$UserName
os.Getenv("USERPROFILE")

// Linux: /Users/UserName/
os.Getenv("HOME")

// 直接使用kubectl的config(/.kube/config)來產生clientsets
kubeconfig := filepath.Join(os.Getenv("HOME"), ".kube", "config")
k8s, err := clientcmd.BuildConfigFromFlags("", kubeconfig)
clientset, err := kubernetes.NewForConfig(k8s)
```

CLIENTS 從K8S集群內來管理

```
import (
    "k8s.io/client-go/rest"
    "k8s.io/client-go/kubernetes"
)

...

k8s, err := rest.InClusterConfig()
clientset, err := kubernetes.NewForConfig(k8s)
```

IF 從集群內：CLUSTERROLE

需事先定義要哪些資源的權限

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: resources-editor
rules:
- apiGroups: ["extensions", "apps"]
  resources: ["deployments", "replicasets", "pods", "services", "endpoints", "jobs", "nodes"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

IF 從集群內：CLUSTERROLEBINDING

將定義好的權限賦予至SERVICEACCOUNT

```yaml
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: editor-resources
  namespace: default
subjects:
- kind: ServiceAccount
  name: default
  namespace: default
roleRef:
  kind: ClusterRole
  name: resources-editor
  apiGroup: rbac.authorization.k8s.io
```

DISCOVERY

```
import (
    "fmt"
)

...

version, _ := clientset.Discovery().ServerVersion()
fmt.Println(version)
// v1.11.0

api, _ := clientset.Discovery().ServerGroups()
for _, api := range apiList.Groups {
    fmt.Printf("%s : %s \n", api.Name, api.Versions[0].Version)
}
// scheduling.k8s.io : v1beta1
// storage.k8s.io : v1
// networking.k8s.io : v1

resources, _ := clientset.Discovery().ServerResources()
for _, r := range resourceList {
    fmt.Printf("%s : %s \n", r.GroupVersion, r.APIResources[0].Name)
}
// apps/v1 : controllerrevisions
// batch/v1 : jobs
// networking.k8s.io/v1 : networkpolicies
```

GET(NAME. *V1.GETOPTIONS)

KUBECTL GET POD PODNAME

```
import (
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
)

...

clientset.CoreV1().Pods("namespace").Get("name", metav1.GetOptions{})
```

```
// GetOptions is the standard query options to the standard REST get call.
type GetOptions struct {
    // When specified:
    // - if unset, then the result is returned from remote storage based on quorum-read flag;
    // - if it's 0, then we simply return what we currently have in cache, no guarantee;
    // - if set to non zero, then the result is at least as fresh as given rv.
    ResourceVersion string `json:"resourceVersion,omitempty" protobuf:"bytes,1,opt,name=resourceVersion"`
}
```

LIST(*V1.LISTOPTIONS)

KUBECTL GET PODS

```
import (
    corev1 "k8s.io/api/core/v1"
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
)

...

pods := []*corev1.Pod{}
podsList, err := clientset.CoreV1().Pods("namespace").List(metav1.ListOptions{})
//podsList.Items: []Pod{}
for _, p := range podsList.Items {
    pods = append(pods, &p)
}
```

```
// ListOptions is the query options to a standard REST list call.
type ListOptions struct {
    // When specified for list:
    // - if unset, then the result is returned from remote storage based on quorum-read flag;
    // - if it's 0, then we simply return what we currently have in cache, no guarantee;
    // - if set to non zero, then the result is at least as fresh as given rv.
    // +optional
    ResourceVersion string `json:"resourceVersion,omitempty" protobuf:"bytes,4,opt,name=resourceVersion"`
}
```

WATCH(*V1.LISTOPTIONS)

KUBECTL GET PODS --WATCH

```
import (
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
)

...

clientset.CoreV1().Pods("namespace").Watch(metav1.ListOptions{ResourceVersion: "0"})
```

```
// ListOptions is the query options to a standard REST list call.
type ListOptions struct {
    // When specified for list:
    // - if unset, then the result is returned from remote storage based on quorum-read flag;
    // - if it's 0, then we simply return what we currently have in cache, no guarantee;
    // - if set to non zero, then the result is at least as fresh as given rv.
    // +optional
    ResourceVersion string `json:"resourceVersion,omitempty" protobuf:"bytes,4,opt,name=resourceVersion"`
}
```

UPDATE(*V1.RESOURCE)

UPDATESTATUS(*V1.RESOURCE)

```
import (
    "fmt"

    corev1 "k8s.io/api/core/v1"
    "k8s.io/apimachinery/pkg/api/errors"
)

...

pod := corev1.Pod{}

for {
    _, err := clientset.CoreV1().Pods("namespace").Update(&pod)
    if errors.IsConflict(err) {
        fmt.Println("encountered conflict, retrying")
    } else if err != nil {
        panic(err)
    } else {
        break
    }
}
```

PATCH(NAME. TYPES.PATCHTYPE. []BYTE)

```
import (
    corev1 "k8s.io/api/core/v1"
)

...

pod := corev1.Pod{}
patchBytes, err := json.Marshal(pod)
clientset.CoreV1().Pods("namespace").Patch("name", types.JSONPatchType, patchBytes)


// Similarly to above, these are constants to support HTTP PATCH utilized by
// both the client and server that didn't make sense for a whole package to be
// dedicated to.
type PatchType string

const (
    JSONPatchType           PatchType = "application/json-patch+json"
    MergePatchType          PatchType = "application/merge-patch+json"
    StrategicMergePatchType PatchType = "application/strategic-merge-patch+json"
)
```

DELETE(NAME, *V1.DELETEOPTIONS)

KUBECTL DELETE POD PODNAME

```
import (
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
)

...

clientset.CoreV1().Pods("namespace").Delete("name", metav1.DeleteOptions{})
```

```
// DeleteOptions may be provided when deleting an API object.
type DeleteOptions struct {
    // Must be fulfilled before a deletion is carried out. If not possible, a 409 Conflict status will be
    // returned.
    // +optional
    Preconditions *Preconditions `json:"preconditions,omitempty" protobuf:"bytes,2,opt,name=preconditions"`

    // Should the dependent objects be orphaned. If true/false, the "orphan"
    // finalizer will be added to/removed from the object's finalizers list.
    // Either this field or PropagationPolicy may be set, but not both.
    // +optional
    OrphanDependents *bool `json:"orphanDependents,omitempty" protobuf:"varint,3,opt,name=orphanDependents"`

    PropagationPolicy *DeletionPropagation `json:"propagationPolicy,omitempty" protobuf:"varint,4,opt,name=propagationPolicy"`
}
```

*V1.DELETEOPTIONS.PROPAGATIONPOLICY

```
// Orphans the dependents.
DeletePropagationOrphan DeletionPropagation = "Orphan"
// Deletes the object from the key-value store, the garbage collector will
// delete the dependents in the background.
DeletePropagationBackground DeletionPropagation = "Background"
// The object exists in the key-value store until the garbage collector
// deletes all the dependents whose ownerReference.blockOwnerDeletion=true
// from the key-value store.  API sever will put the "foregroundDeletion"
// finalizer on the object, and sets its deletionTimestamp.  This policy is
// cascading, i.e., the dependents will be deleted with Foreground.
DeletePropagationForeground DeletionPropagation = "Foreground"
```


LEVEL 1
CREATE, UPDATE & DELETE
DEPLOYMENT
(MORE: RESTFUL API SERVER)

(MORE:)GORILLA/MUX

```
import (
    "net/http"
    "log"
    "github.com/gorilla/mux"
)

...

func CreateDeploymentHandler(w http.ResponseWriter, r *http.Request) {
    //Do something here
    w.Write([]byte("Success!\n"))
}

func UpdateDeploymentHandler(w http.ResponseWriter, r *http.Request) {
    //Do something here
    w.Write([]byte("Success!\n"))
}

func DeleteDeploymentHandler(w http.ResponseWriter, r *http.Request) {
    //Do something here
    w.Write([]byte("Success!\n"))
}

func main() {
    r := mux.NewRouter()
    r.HandleFunc("/deployment/create", CreateDeploymentHandler).Methods("POST")
    r.HandleFunc("/deployment/update", UpdateDeploymentHandler).Methods("PUT")
    r.HandleFunc("/deployment/delete", DeleteDeploymentHandler).Methods("Delete")

    log.Fatal(http.ListenAndServe(":8000", r))
}
```

TESTING: FAKE CLIENTSET

```
import (
    corev1 "k8s.io/api/core/v1"
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
    "k8s.io/client-go/kubernetes/fake"
)

...

namespace := "default"
pod := corev1.Pod{
  ObjectMeta: metav1.ObjectMeta{
    Name: "K8S-Pod-1",
  },
}
fakeClientset := fake.NewSimpleClientset()
_, err := fakeClientset.CoreV1().Pods(namespace).Create(&pod)
result, err := suite.kubectl.GetPod("K8S-Pod-1")
```

KUBERNETES.INTERFACE

```
import (
    "k8s.io/client-go/kubernetes"
)

...

func CreatePod(clientset kubernetes.Interface, namespace string, pod *corev1.Pod) (*corev1.Pod, error) {
    return clientset.CoreV1().Pods(namespace).Create(pod)
}

func GetPod(clientset kubernetes.Interface, namespace string, podName string) (*corev1.Pod, error) {
    return clientset.CoreV1().Pods(namespace).Get(podName, metav1.GetOptions{})
}

func DeletePod(clientset kubernetes.Interface, namespace string, podName string) (*corev1.Pod, error) {
    return clientset.CoreV1().Pods(namespace).Delete(podName, &metav1.GetOptions{})
}
```


LEVEL 2
CREATE, UPDATE & DELETE
DEPLOYMENT WITH TESTING

CLIENTS COMPONENT TYPE

- **CLIENTSET** `return struct{}`

  CLIENT-GO/KUBERNETES

- **DYNAMIC CLIENT** `return map[string]interface{}`

  CLIENT-GO/DYNAMIC

DYNAMIC CLIENT

```
import (
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
    schema "k8s.io/apimachinery/pkg/runtime/schema"
)

...

resource  := schema.GroupVersionResource{Group: "", Version: "v1", Resource: "pods"}
dynamicClient.Resource(resource, "namespace").List(metav1.ListOptions{})
```

CLIENTS COMPONENT TYPE

- **CLIENTSET** `return struct{}`

CLIENT-GO/KUBERNETES

- **DYNAMIC CLIENT** `return map[string]interface{}`

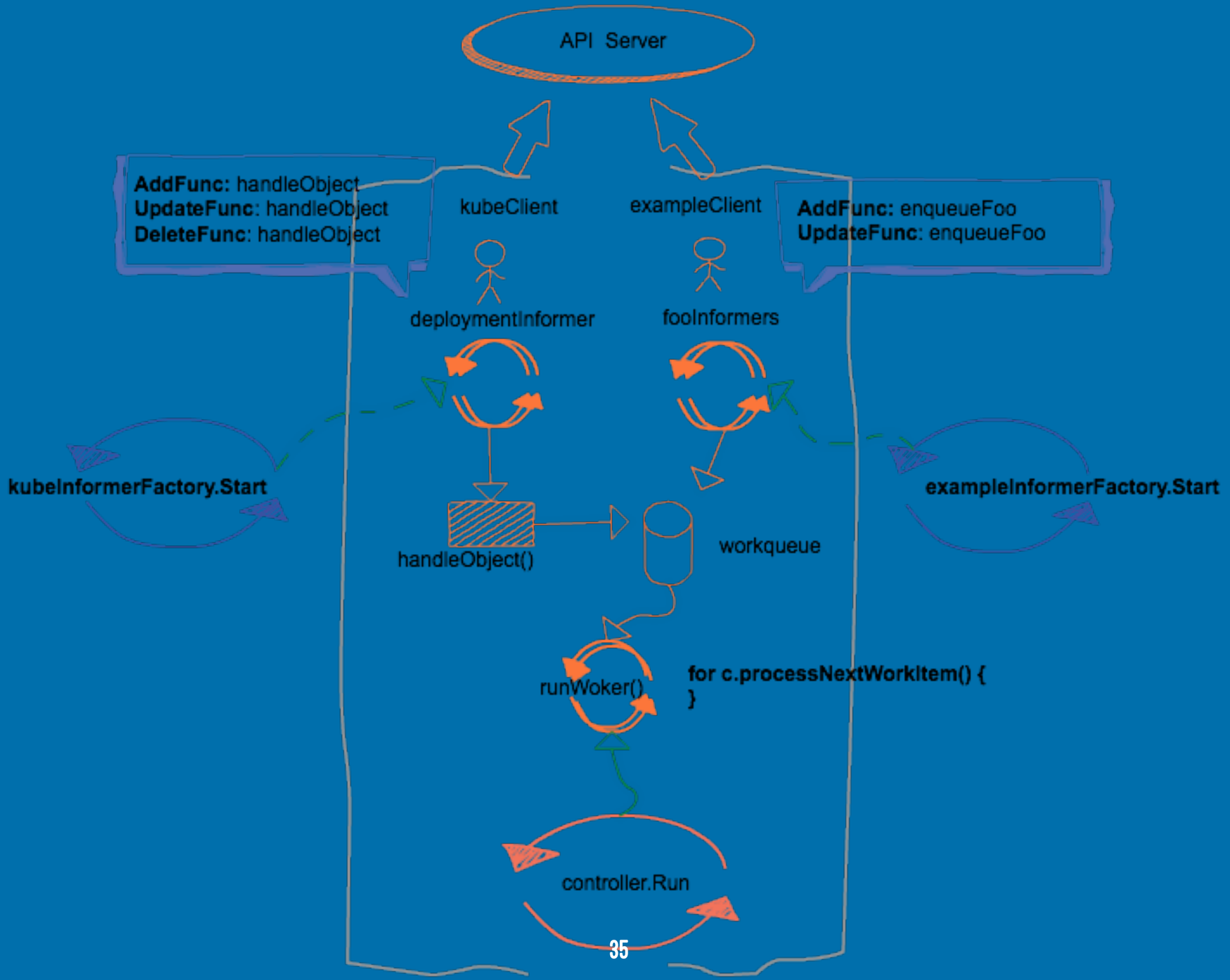
CLIENT-GO/DYNAMIC

- **REST CLIENT** `return struct or byte[]`

CLIENT-GO/REST

REST CLIENT

```
import (
    corev1 "k8s.io/api/core/v1"
)

...

var pod corev1.Pod
restClient.Get().Resource("pods").Namespace("namespace").DO().Into(&pod)
restClient.Get().Resource("pods").Namespace("namespace").DORaw() // byte[]
```


API Server
AddFunc: handleObject
UpdateFunc: handleObject
DeleteFunc: handleObject
kubeClient
exampleClient
AddFunc: enqueueFoo
UpdateFunc: enqueueFoo
deploymentInformer
fooInformers
kubeInformerFactory.Start
exampleInformerFactory.Start
handleObject()
workqueue
runWoker()
for c.processNextWorkItem() {
}
controller.Run

WORKQUEUE

- DELAY QUEUE

延遲某段時間才將資料放入隊列之中

(避免HOT-LOOP)

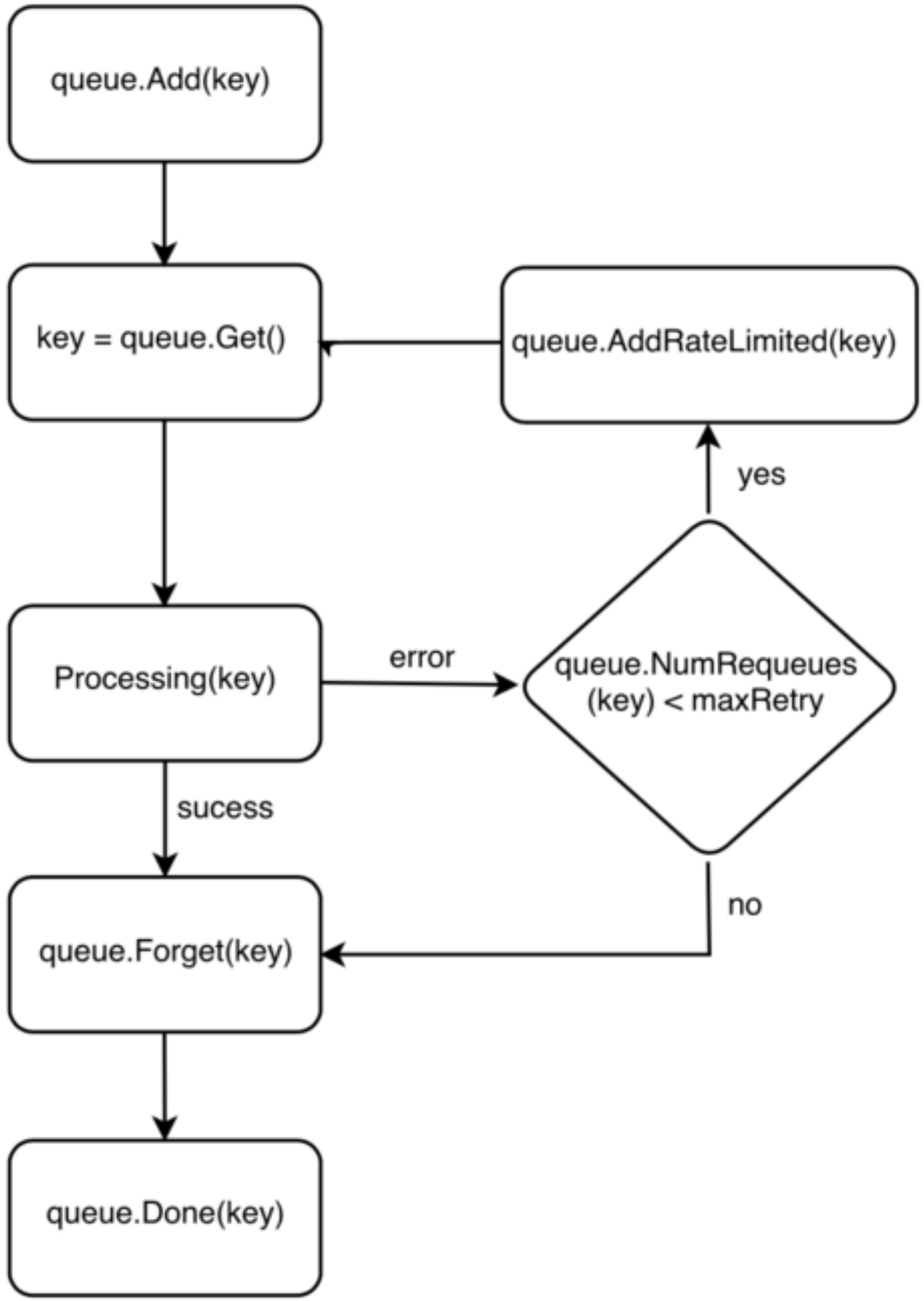
- RATE LIMITTING QUEUE

控制資料放入隊列之中的速率

(基於DELAY QUEUE的實作)

RATE LIMITTING QUEUE

```
import (
    "k8s.io/client-go/util/workqueue"
)

...

queue := workqueue.NewRateLimitingQueue(workqueue.DefaultControllerRateLimiter())
defer queue.ShutDown()


func NewItemExponentialFailureRateLimiter(baseDelay time.Duration, maxDelay time.Duration) RateLimiter {
    return &ItemExponentialFailureRateLimiter{
        failures:  map[interface{}]int{},
        baseDelay: baseDelay,
        maxDelay:  maxDelay,
    }
}

func DefaultItemBasedRateLimiter() RateLimiter {
    return NewItemExponentialFailureRateLimiter(time.Millisecond, 1000*time.Second)
}
```



INFORMER

- QUEUE.ADD(KEY)

WORKER

- QUEUE.GET()
- PROCESSING(KEY)
- QUEUE.ADDRATELIMITED(KEY)
- QUEUE.FORGET(KEY)
- QUEUE.DONE(KEY)

INFORMER

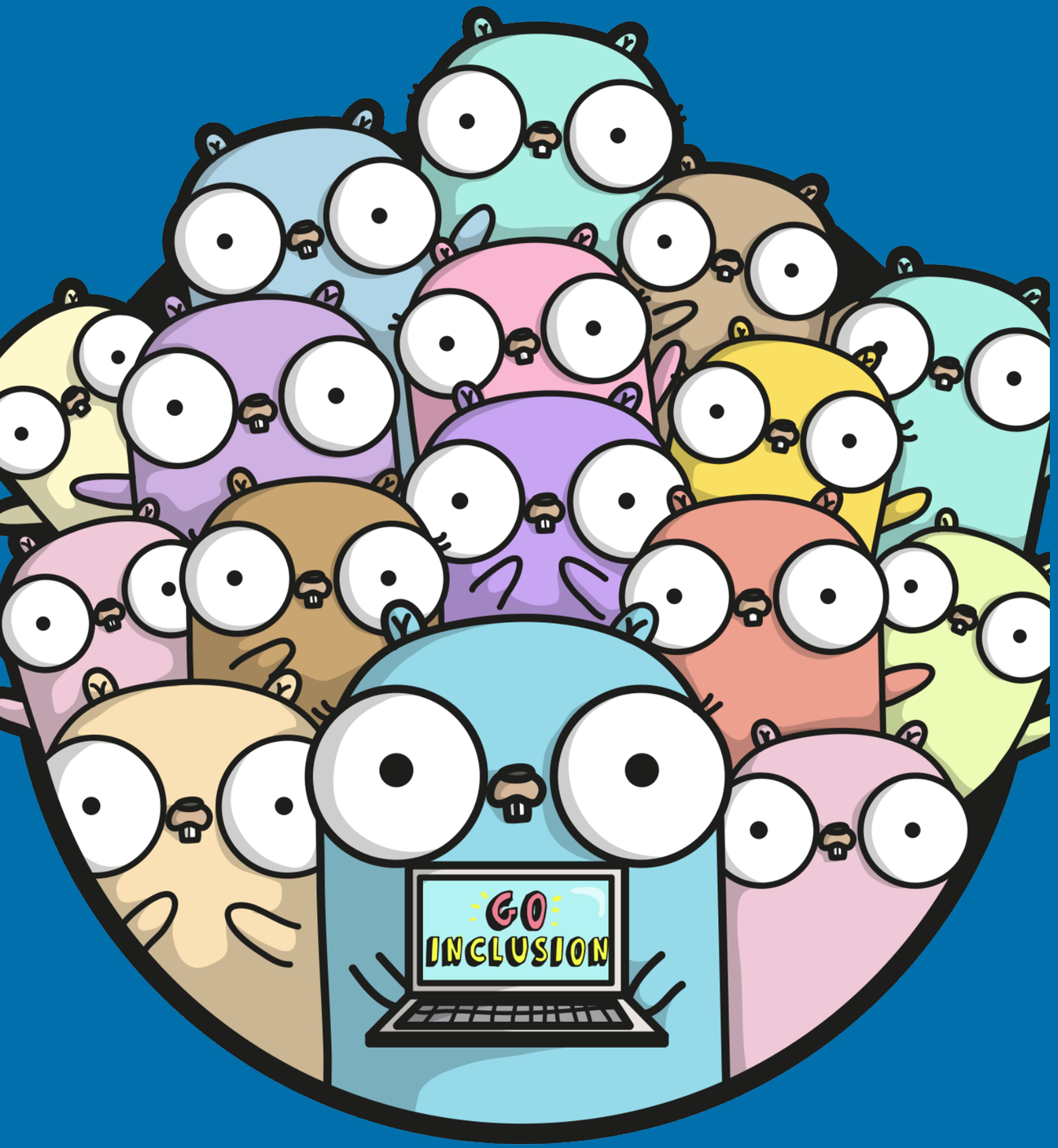
```
import (
    "k8s.io/apimachinery/pkg/fields"
    "k8s.io/client-go/tools/cache"
    corev1 "k8s.io/api/core/v1"
)

...

podListWatcher := cache.NewListWatchFromClient(clientset.CoreV1().RESTClient(), "pods", "default", fields.Everything())

indexer, informer := cache.NewIndexerInformer(podListWatcher, &corev1.Pod{}, 0, cache.ResourceEventHandlerFuncs{
    AddFunc: func(obj interface{}) {
        key, err := cache.MetaNamespaceKeyFunc(obj)
        if err == nil {
            queue.Add(key)
        }
    },
    UpdateFunc: func(old interface{}, new interface{}) {
        key, err := cache.MetaNamespaceKeyFunc(new)
        if err == nil {
            queue.Add(key)
        }
    },
    DeleteFunc: func(obj interface{}) {
        // IndexerInformer uses a delta queue, therefore for deletes we have to use this
        // key function.
        key, err := cache.DeletionHandlingMetaNamespaceKeyFunc(obj)
        if err == nil {
            queue.Add(key)
        }
    },
}, cache.Indexers{})
stop := make(chan struct{})
defer close(stop)
go informer.Run(stop)
<-stop
```

WORKER

```
import (
    "fmt"
    "k8s.io/client-go/tools/cache"
    "k8s.io/apimachinery/pkg/util/wait"
)

...

if !cache.WaitForCacheSync(stopCh, informer.HasSynced) {
    return
}
go wait.Until(runWorker, time.Second, stop)

func runWorker {
    for {
        key, quit := c.queue.Get()
        if quit {
            break
        }
    }
        defer queue.Done(key)
        obj, exists, _ := indexer.GetByKey(key)
        if exists {
            fmt.Printf("Sync/Add/Update for Pod %s\n", obj.(*v1.Pod).GetName())
        }
  }
}
```

LEVEL 3

CREATE A POD'S SERVICE DETECTOR



Q & A