**Commit容器镜像**

1. **新增Commit API**
2. **定义API结构体**

编辑pkg/api/types.go和pkg/api/v1/types.go,添加API的结构体定义

|  |
| --- |
| type CommitList struct {  TypeMeta `json:",inline"`  ListMeta `json:"metadata,omitempty"`  Items []Commit `json:"items" description:"list of imagecommits"`  }  type Commit struct {  TypeMeta `json:",inline"`  ObjectMeta `json:"metadata,omitempty"`  Spec CommitSpec `json:"spec,omitempty"`  Status CommitStatus `json:"status,omitempty"`  }  type CommitSpec struct {  Images []Image `json:"images"`  RestartPolicy RestartPolicy `json:"restartPolicy,omitempty"`  PodName string `json:"podName,omitempty"`  NodeName string `json:"nodeName,omitempty"`  ImagePullSecrets []LocalObjectReference `json:"imagePullSecrets,omitempty"`  }  type Image struct {  ContainerName string `json:"containerName"`  From string `json:"from,omitempty"`  Image string `json:"image"`  ImagePushPolicy PushPolicy `json:"imagePushPolicy,omitempty"`  }  type ContainerCommitStatus struct {  ContainerName string `json:"containerName"`  ImageID string `json:"imageID"`  Phase CommitPhase `json:"phase"`  RestartCount int `json:"restartCount,omitempty"`  }  type CommitStatus struct {  ContainerCommitStatuses []ContainerCommitStatus `json:"containerCommitStatuses,omitempty"`  Message string `json:"message,omitempty"`  } |

**2.注册API**

编辑pkg/api/register.go和pkg/api/v1/register.go, 注册API

|  |
| --- |
| func init() {  Scheme.AddKnownTypes("",  //add-by-lmq 2015-10-19 commitImage start  &Commit{},  &CommitList{},  //add-by-lmq 2015-10-19 commitImage end  )  // Legacy names are supported  }  //add-by-lmq 2015-10-19 commitImage start  func (\*Commit) IsAnAPIObject() {}  func (\*CommitList) IsAnAPIObject() {}  //add-by-lmq 2015-10-19 commitImage end |

**3.生成api之间的编码解码函数**

运行/hack/update-generated-conversion.go和/hack/update-generated-deep-copies.go, 生成API多个版本间的转换函数即可

**4.实现API Server写Etcd时的各个函数**

**1）pkg/registry/commit/etcd/etcd.go**

|  |
| --- |
| package etcd  import (  "github.com/GoogleCloudPlatform/kubernetes/pkg/api"  "github.com/GoogleCloudPlatform/kubernetes/pkg/fields"  "github.com/GoogleCloudPlatform/kubernetes/pkg/labels"  "github.com/GoogleCloudPlatform/kubernetes/pkg/registry/cachesize"  "github.com/GoogleCloudPlatform/kubernetes/pkg/registry/commit"  "github.com/GoogleCloudPlatform/kubernetes/pkg/registry/generic"  etcdgeneric "github.com/GoogleCloudPlatform/kubernetes/pkg/registry/generic/etcd"  "github.com/GoogleCloudPlatform/kubernetes/pkg/runtime"  "github.com/GoogleCloudPlatform/kubernetes/pkg/storage"  )  // rest implements a RESTStorage for replication controllers against etcd  type REST struct {  \*etcdgeneric.Etcd  }  // NewREST returns a RESTStorage object that will work against replication controllers.  func NewREST(s storage.Interface, storageDecorator generic.StorageDecorator, useWatchCache \*bool) \*REST {  prefix := "/commits"  newListFunc := func() runtime.Object { return &api.CommitList{} }  storageInterface := storageDecorator(  s, cachesize.GetWatchCacheSizeByResource(cachesize.Commits), &api.Commit{}, prefix, commit.Strategy, newListFunc, useWatchCache)  store := &etcdgeneric.Etcd{  NewFunc: func() runtime.Object { return &api.Commit{} },  // NewListFunc returns an object capable of storing results of an etcd list.  NewListFunc: newListFunc,  // Produces a path that etcd understands, to the root of the resource  // by combining the namespace in the context with the given prefix  KeyRootFunc: func(ctx api.Context) string {  return etcdgeneric.NamespaceKeyRootFunc(ctx, prefix)  },  // Produces a path that etcd understands, to the resource by combining  KeyFunc: func(ctx api.Context, name string) (string, error) {  // the namespace in the context with the given prefix  return etcdgeneric.NamespaceKeyFunc(ctx, prefix, name)  },  // Retrieve the name field of a replication controller  ObjectNameFunc: func(obj runtime.Object) (string, error) {  return obj.(\*api.Commit).Name, nil  },  // Used to match objects based on labels/fields for list and watch  PredicateFunc: func(label labels.Selector, field fields.Selector) generic.Matcher {  return commit.MatchCommit(label, field)  },  ResourceName: "commits",  // Used to validate imagecommit creation  CreateStrategy: commit.Strategy,  // Used to validate imagecommit updates  UpdateStrategy: commit.Strategy,  Storage: storageInterface,  }  return &REST{store}  } |

**2) pkg/registry/commit/etcd/rest.go**

|  |
| --- |
| package commit  import (  "fmt"  "github.com/GoogleCloudPlatform/kubernetes/pkg/api"  "github.com/GoogleCloudPlatform/kubernetes/pkg/api/validation"  "github.com/GoogleCloudPlatform/kubernetes/pkg/fields"  "github.com/GoogleCloudPlatform/kubernetes/pkg/labels"  "github.com/GoogleCloudPlatform/kubernetes/pkg/registry/generic"  "github.com/GoogleCloudPlatform/kubernetes/pkg/runtime"  "github.com/GoogleCloudPlatform/kubernetes/pkg/util/fielderrors"  )  // cjStrategy implements verification logic for Commit.  type cjStrategy struct {  runtime.ObjectTyper  api.NameGenerator  }  // Strategy is the default logic that applies when creating and updating Commit objects.  var Strategy = cjStrategy{api.Scheme, api.SimpleNameGenerator}  // NamespaceScoped returns true because all Commits need to be within a namespace.  func (cjStrategy) NamespaceScoped() bool {  return true  }  // PrepareForCreate clears the status of a Commit before creation.  func (cjStrategy) PrepareForCreate(obj runtime.Object) {  commit := obj.(\*api.Commit)  commit.Status = api.CommitStatus{}  }  // PrepareForUpdate clears fields that are not allowed to be set by end users on update.  func (cjStrategy) PrepareForUpdate(obj, old runtime.Object) {  // TODO: once Commit has a status sub-resource we can enable this.  // newCommit := obj.(\*api.Commit)  // oldCommit := old.(\*api.Commit)  // newCommit.Status = oldCommit.Status  }  // Validate validates a new commit.  func (cjStrategy) Validate(ctx api.Context, obj runtime.Object) fielderrors.ValidationErrorList {  commit := obj.(\*api.Commit)  return validation.ValidateCommit(commit)  }  // AllowCreateOnUpdate is false for Commit; this means a POST is  // needed to create one.  func (cjStrategy) AllowCreateOnUpdate() bool {  return false  }  // ValidateUpdate is the default update validation for an end user.  func (cjStrategy) ValidateUpdate(ctx api.Context, obj, old runtime.Object) fielderrors.ValidationErrorList {  validationErrorList := validation.ValidateCommit(obj.(\*api.Commit))  updateErrorList := validation.ValidateCommitUpdate(old.(\*api.Commit), obj.(\*api.Commit))  return append(validationErrorList, updateErrorList...)  }  func (cjStrategy) AllowUnconditionalUpdate() bool {  return true  }  // ControllerToSelectableFields returns a label set that represents the object.  func CommitToSelectableFields(Commit \*api.Commit) fields.Set {  return fields.Set{  "metadata.name": Commit.Name,  "spec.nodeName": Commit.Spec.NodeName,  "spec.podName": Commit.Spec.PodName,  }  }  // MatchController is the filter used by the generic etcd backend to route  // watch events from etcd to clients of the apiserver only interested in specific  // labels/fields.  func MatchCommit(label labels.Selector, field fields.Selector) generic.Matcher {  return &generic.SelectionPredicate{  Label: label,  Field: field,  GetAttrs: func(obj runtime.Object) (labels.Set, fields.Set, error) {  rc, ok := obj.(\*api.Commit)  if !ok {  return nil, nil, fmt.Errorf("Given object is not a commit job.")  }  return labels.Set(rc.ObjectMeta.Labels), CommitToSelectableFields(rc), nil  },  }  } |

**5.注册路由**

编辑pkg/master/master.go，注册路由

**5.添加client**

仿照其他资源，在pkg/client下添加该资源的client

1. **实现原理**

Kubelet会list&watch commit资源，并调用docker的API把容器commit为image，并把image push到镜像仓库。如果commit json里的字段不合法，比如容器不存在等，commit会失败。