Hidden Generics in Kubernetes' API

- Eirik Albrigtsen
- **clux** / **@sszynrae**
- **kube-rs**
- slides at **http://clux.github.io/kubecon2020**

Hidden Generics in Kubernetes' API

- Finding invariants in Go codebase
- Use Rust Generics to model the API
- Rust Controllers

Kubernetes Invariants

- apimachinery/meta/v1/types.go
- client-go/kubernetes/typed
- kubernetes.io/docs/concepts

types.go: TypeMeta

types.go#L36-56

```
type TypeMeta struct {
    // +optional
    Kind string `json:"kind,omitempty"
protobuf:"bytes,1,opt,name=kind"`
    // +optional
    APIVersion string `json:"apiVersion,omitempty"
protobuf:"bytes,2,opt,name=apiVersion"`
}
```

types.go: ObjectMeta

types.go#L108-L282

```go
type ObjectMeta struct {
    Name string
    Namespace string

    Labels map[string]string
    Annotations map[string]string
    OwnerReferences []OwnerReference
    Finalizers []string
    ClusterName string
    ManagedFields []ManagedFieldsEntry
}
```

types.go: List

types.go#L913-L923

```go
type List struct {
    TypeMeta `json:",inline"`
    ListMeta `json:"metadata,omitempty"`
    Items []runtime.RawExtension `json:"items"`
}
```

TYPES.GO: LISTOPTIONS

types.go#L328-L412

```
type ListOptions struct {
    TypeMeta
    LabelSelector string
    FieldSelector string
    Watch bool
    AllowWatchBookmarks bool
    ResourceVersion string
    ResourceVersionMatch ResourceVersionMatch
    TimeoutSeconds *int64
    Limit int64
    Continue string
}
```

TYPES.GO: APIRESOURCE

types.go#L998-L1032

```go
type APIResource struct {
    Name string
    SingularName string
    Namespaced bool
    Group string
    Version string
    Kind string
    Verbs Verbs
    ShortNames []string
    Categories []string
    StorageVersionHash string
}
```

Types.go

- 339 lines of code
- 928 lines of comments

CLIENT-GO: DEPLOYMENT

deployment.go#L41-L55

```
type DeploymentInterface interface {
    Create(ctx context.Context, deployment *v1.Deployment, opts
metav1.CreateOptions) (*v1.Deployment, error)
    Update(ctx context.Context, deployment *v1.Deployment, opts
metav1.UpdateOptions) (*v1.Deployment, error)
    UpdateStatus(ctx context.Context, deployment *v1.Deployment, opts
metav1.UpdateOptions) (*v1.Deployment, error)
    Delete(ctx context.Context, name string, opts
metav1.DeleteOptions) error
    DeleteCollection(ctx context.Context, opts metav1.DeleteOptions,
listOpts metav1.ListOptions) error
    Get(ctx context.Context, name string, opts metav1.GetOptions)
(*v1.Deployment, error)
    List(ctx context.Context, opts metav1.ListOptions)
(*v1.DeploymentList, error)
    Watch(ctx context.Context, opts metav1.ListOptions)
(watch.Interface, error)
    Patch(ctx context.Context, name string, pt types.PatchType, data
[]byte, opts metav1.PatchOptions, subresources ...string) (result
*v1.Deployment, err error)
    GetScale(ctx context.Context, deploymentName string, options
metav1.GetOptions) (*autoscalingv1.Scale, error)
    UpdateScale(ctx context.Context, deploymentName string, scale
*autoscalingv1.Scale, opts metav1.UpdateOptions)
(*autoscalingv1.Scale, error)
```

CLIENT-GO: Pod

pod.go#L39-L54

```
type PodInterface interface {
    Create(ctx context.Context, pod *v1.Pod, opts 
metav1.CreateOptions) (*v1.Pod, error)
    Update(ctx context.Context, pod *v1.Pod, opts 
metav1.UpdateOptions) (*v1.Pod, error)
    UpdateStatus(ctx context.Context, pod *v1.Pod, opts 
metav1.UpdateOptions) (*v1.Pod, error)
    Delete(ctx context.Context, name string, opts 
metav1.DeleteOptions) error
    DeleteCollection(ctx context.Context, opts metav1.DeleteOptions, 
listOpts metav1.ListOptions) error
    Get(ctx context.Context, name string, opts metav1.GetOptions) 
(*v1.Pod, error)
    List(ctx context.Context, opts metav1.ListOptions) (*v1.PodList, 
error)
    Watch(ctx context.Context, opts metav1.ListOptions) 
(watch.Interface, error)
    Patch(ctx context.Context, name string, pt types.PatchType, data 
[]byte, opts metav1.PatchOptions, subresources ...string) (result 
*v1.Pod, err error)
    GetEphemeralContainers(ctx context.Context, podName string, 
options metav1.GetOptions) (*v1.EphemeralContainers, error)
    UpdateEphemeralContainers(ctx context.Context, podName string,
ephemeralContainers *v1.EphemeralContainers, opts
metav1.UpdateOptions) (*v1.EphemeralContainers, error)
```

CLIENT-GO: HEADER

deployment.go#L17

```
// Code generated by client-gen. DO NOT EDIT.

package v1
```

CLIENT-GO

- tons of generated code per object
- **specialized client api**
- **specialized informers**
- more than 100K lines of code

Bryan Liles: **client-go is not for mortals**

kubernetes.io: api endpoints

api-concepts#standard-api-terminology

Cluster-scoped resources

```
GET /apis/GROUP/VERSION/RESOURCETYPE
GET /apis/GROUP/VERSION/RESOURCETYPE/NAME
```

Namespace-scoped resources

```
GET /apis/GROUP/VERSION/RESOURCETYPE
GET /apis/GROUP/VERSION/namespaces/NAMESPACE/RESOURCETYPE
GET /apis/GROUP/VERSION/namespaces/NAMESPACE/RESOURCETYPE/NAME
```

Broken: empty api group

```
GET /api/v1/pods

       !=

GET /apis/core/v1/pods
```

kubernetes.io: watch events

api-concepts#efficient-detection-of-changes

```
{ "type": "ADDED", "object": { \
    "kind": "Pod",  "apiVersion": "v1", \
    "metadata": {"resourceVersion": "10596", ...}, ...} }
{ "type": "MODIFIED", "object": { \
    "kind": "Pod", "apiVersion": "v1", \
    "metadata": {"resourceVersion": "11020", ...}, ...} }
```

kubernetes.io: watch events - source

- apimachinery:watch/watch.go#L40-L70
- apimachinery:meta/watch.go#L31-L40

```go
const (
    Added    EventType = "ADDED"
    Modified EventType = "MODIFIED"
    Deleted  EventType = "DELETED"
    Bookmark EventType = "BOOKMARK"
    Error    EventType = "ERROR"
)

type WatchEvent struct {
    Type string `json:"type"`
    Object runtime.RawExtension `json:"object"`
}
```

Rust Modelling

- **clux/kube-rs**
- Arnav Singh / @Arnavion - **k8s-openapi**

k8s-openapi: Resource Trait

```rust
pub trait Resource {
    const API_VERSION: &'static str;
    const GROUP: &'static str;
    const KIND: &'static str;
    const VERSION: &'static str;
}
```

k8s-openapi: Metadata Trait

```rust
pub trait Metadata: Resource {
    fn metadata(&self) -> &ObjectMeta;
}
```

kube-rs: Resource struct

```rust
pub struct Resource {
    pub api_version: String,
    pub group: String,
    pub kind: String,
    pub version: String,
    pub namespace: Option<String>
}
```

kube-rs: Resource namespaced ctor

```rust
use k8s_openapi::Resource as ResourceTrait;

impl Resource {
    pub fn namespaced<K: ResourceTrait>(ns: &str) -> Self {
        Self {
            api_version: K::API_VERSION.to_string(),
            kind: K::KIND.to_string(),
            group: K::GROUP.to_string(),
            version: K::VERSION.to_string(),
            namespace: Some(ns.to_string()),
        }
    }
}
```

kube-rs: Url mapper

```rust
impl Resource {
    fn make_url(&self) -> String {
      format!("/{group}/{api_version}/{namespaces}{resource}",
        group = if self.group.is_empty() {"api"} else {"apis"},
        api_version = self.api_version,
        resource = to_plural(&self.kind.to_ascii_lowercase()),
        namespaces = self.namespace.as_ref()
          .map(|n| format!("namespaces/{}/", n))
          .unwrap_or_default())
    }
}
```

kube-rs: Dynamic API

```rust
impl Resource {
    pub fn create(&self, pp: &PostParams, data: Vec<u8>)
        -> Result<Request<Vec<u8>>>
    {
        let base_url = self.make_url() + "?";
        let mut qp = Serializer::new(base_url);
        if pp.dry_run {
            qp.append_pair("dryRun", "All");
        }
        let urlstr = qp.finish();
        let req = http::Request::post(urlstr);
        req.body(data).map_err(Error::HttpError)
    }
}
```

kube-rs: Typed API

```
pub struct Api<K> {
    resource: Resource,
    client: Client,
    phantom: PhantomData<K>,
}

let api: Api<Pod> = Api::namespaced(client, ns);
```

kube-rs: Typed API methods

```rust
impl<K> Api<K>
where K: Clone + Deserialize + Metadata,
{
    pub async fn create(&self, pp: &PostParams, data: &K)
        -> Result<K>
    where K: Serialize,
    {
        let bytes = serde_json::to_vec(&data)?;
        let req = self.resource.create(&pp, bytes)?;
        self.client.request::<K>(req).await
    }
}
```

Code Generation

- first class integration via cargo build
- **procedural macros**
- #[derive(CustomTrait)]
- #[custom_trait_attr]

Serialize

```rust
#[derive(Serialize, Deserialize)]
#[serde(rename_all = "camelCase")]
pub struct FooSpec {
    name: String,
    is_bad: Option<String>,
}
```

kube-derive: CustomResource

```rust
#[derive(CustomResource, Serialize, Deserialize, Clone)]
#[kube(group = "clux.dev", version = "v1", kind = "Foo")]
#[kube(namespaced, status = "FooStatus")]
pub struct FooSpec {
    name: String,
    info: Option<String>,
}
```

Example: Using a CRD

```rust
let crds: Api<CustomResourceDefinition> = Api::all(client);
crds.create(&pp, &Foo::crd()).await;

let foos: Api<Foo> = Api::namespaced(client, &namespace);

let f = Foo::new("eirik-example", FooSpec {
    name: "i am a foo crd instance".into(),
    info: None
});
let o = foos.create(&pp, &f2).await?;
```

WatchEvent

```rust
#[derive(Deserialize, Serialize, Clone)]
#[serde(tag = "type", content = "object")]
#[serde(rename_all = "UPPERCASE")]
pub enum WatchEvent<K> {
    Added(K),
    Modified(K),
    Deleted(K),
    Bookmark(Bookmark),
    Error(ErrorResponse),
}
```

Watch

```rust
impl<K> Api<K>
where K: Clone + Deserialize + Metadata,

    pub async fn watch(&self, lp: &ListParams, rv: &str)
        -> Result<impl Stream<Item = Result<WatchEvent<K>>>>
    {
        let req = self.resource.watch(&lp, &rv)?;
        self.client.request_events::<K>(req).await
    }
}
```

Broken: Watch

- resourceVersion bookkeeping
- stale resourceVersions **#87292**
- Removed events are near useless
- 5 minute max limit **#6513**
- large data use **#90339**, **#82655**

watcher abstraction

- LIST
- stream
- track resource versions
- handle stream errors behind the scenes
- maybe RE-LIST (duplicate + dropped events)
- propagate user errors
- only propagate events

kube-runtime

- Teo K. Röijezon - **teozkr**
- Entirely Stream based solution
- watcher
- reflector with Store
- Controller

kube-runtime: watcher

```rust
enum State<K: Meta + Clone> {
    /// Empty state, awaiting a LIST
    Empty,
    /// LIST complete, can start watching
    InitListed { resource_version: String },
    /// Watching, can await/restart watch/restart
    Watching {
        resource_version: String,
        stream: BoxStream<'static, Result<WatchEvent<K>>>,
    },
}
```

```rust
watcher(api, listparams)
    -> impl Stream<Item = Result<watcher::Event<K>>>
```

kube-runtime: watcher usage

```rust
let cms: Api<ConfigMap> = Api::namespaced(client, &namespace);
let lp = ListParams::default();

let mut w = try_flatten_applied(watcher(cms, lp)).boxed();
while let Some(event) = w.try_next().await? {
    info!("Got: {:?}", event);
}
```

kube-runtime: reflector

```
pub fn reflector<K, W>(mut store: Writer<K>, stream: W)
    -> impl Stream<Item = W::Item>
where
    K: Metadata + Clone,
    W: Stream<Item = Result<watcher::Event<K>>>,
{
    stream.inspect_ok(move |event| {
        store.apply_watcher_event(event)
    })
}
```

kube-runtime: reflector usage

```rust
let cms: Api<ConfigMap> = Api::namespaced(client, &namespace);

let writer = Writer::<ConfigMap>::default();
let reader = writer.as_reader();
let rf = reflector(writer, watcher(cms, lp));

let mut w = try_flatten_applied(rf).boxed();
while let Some(event) = w.try_next().await? {
    info!("Applied {}", Meta::name(&event));
}
```

kube-runtime: Controller

```rust
#[tokio::main]
async fn main() -> Result<(), kube::Error> {
    let client = Client::try_default().await?;
    let context = Context::new(());
    let cmgs = Api::<ConfigMapGenerator>::all(client.clone());
    let cms = Api::<ConfigMap>::all(client.clone());

    Controller::new(cmgs, ListParams::default())
        .owns(cms, ListParams::default())
        .run(reconcile, error_policy, context)
        .await;
    Ok(())
}
```

kube-runtime: Controller reconciler

```
async fn reconcile(cmg: ConfigMapGenerator, ctx: Context<()>)
        -> Result<ReconcilerAction, Error>
{
    // TODO: update CM to match cmg.content
    // TODO: update CMG.status
    Ok(ReconcilerAction {
        requeue_after: Some(Duration::from_secs(300)),
    })
}
```

Building Controllers

- controller-runtime advice applies
- idempotent, error resilient reconcilers
- use server side apply
- use finalizers / ownerreferences

Examples

- **controller-rs** and **version-rs**
- Bring your own deps
- web: **actix-web**, **warp**, **rocket**
- o11y: **tracing**, **sentry**, **prometheus**

End

- Eirik Albrigtsen
- **clux** / **@sszynrae**
- **kube-rs**
- slides at **http://clux.github.io/kubecon2020**
- **babylonhealth**