JETSTACK by Venafi

Talking to Kubernetes with Rust

James Laverack

Staff Solutions Engineer

KubeCon | CloudNativeCon

Europe 2023

- Rust!
- How the Kubernetes API really works
- How to make Rust talk Kubernetes
- What this means for you and your project

What is Rust?

The (boring) facts about Rust

- Compiled
- Statically-typed
- “Borrow checker” instead of garbage collection or manual memory management
- Fast execution speed
- Powerful type system
- LLVM-backed (Can compile to WASM, x86_64, ARM, etc.)

The parts that matter (to me!):

- Welcoming community, just like over here in the cloud native world ✨
- Very well thought out, mature ecosystem of supporting projects (Tokio for async, Cargo for dependencies, etc.)
- Some of the most helpful error messages and ergonomics from a compiler I've ever seen.

How does anything talk to Kubernetes?

YAML fans cover your eyes… 🙈

The Kubernetes API is actually JSON 😱

```
$ kubectl get po --v=6

I0404 14:40:46.784889   3367 loader.go:373] Config loaded
from file:  /Users/james/.kube/config

I0404 14:40:46.804114   3367 round_trippers.go:553] GET
https://127.0.0.1:59011/api/v1/namespaces/default/pods?limit
=500 200 OK in 16 milliseconds

NAME   READY    STATUS    RESTARTS    AGE

debian    1/1  Running    0          14m
```

```
$ kubectl get po --v=8

[…]

I0404 14:42:15.687384   3469 request.go:1171] Response Body:
{"kind":"Table","apiVersion":"meta.k8s.io/v1","metadata":{"resourceVersion":"1968211"},"columnDefinitions":[{
"name":"Name","type":"string","format":"name","description":"Name must be unique within a namespace. Is
required when creating resources, although some resources may allow a client to request the generation of an
appropriate name automatically. Name is primarily intended for creation idempotence and configuration
definition. Cannot be updated. More info:
http://kubernetes.io/docs/user-guide/identifiers#names","priority":0},{"name":"Ready","type":"string","format
":"","description":"The aggregate readiness state of this pod for accepting
traffic.","priority":0},{"name":"Status","type":"string","format":"","description":"The aggregate status of
the containers in this pod.","priority":0},{"name":"Restarts","type":"string","format":"","description":"The
number of times the containers in this pod have been restarted and when the last container in this pod has
restarted.","priority":0},{"name":"Age","type":"s [truncated 3582 chars]
```

All we really need are sockets

```
$ yq '.clusters[0].cluster.certificate-authority-data' < ~/.kube/config |
base64 -d > ca.pem

$ yq '.users[0].user.client-certificate-data' < ~/.kube/config | base64 -d >
client-cert.pem

$ yq '.users[0].user.client-key-data' < ~/.kube/config | base64 -d >
client-key.pem

$ curl -H "Accept: application/json"
'https://127.0.0.1:59011/api/v1/namespaces/default/pods?limit=500' --cacert
ca.pem --cert client-cert.pem --key client-key.pem
```

Rust has perfectly good dependencies for HTTP requests, TLS, file reading, YAML, JSON parsing…

But we can do much better

The kube crate*

*In Rust, dependencies are called “crates”

> A Rust client for Kubernetes in the style of a more generic client-go, a runtime abstraction inspired by controller-runtime, and a derive macro for CRDs inspired by kubebuilder. Hosted by CNCF as a Sandbox Project

https://github.com/kube-rs/kube/blob/59141648bba7bf38998a714b6dfee807f5a5368c/README.md

```rust
use k8s_openapi::api::core::v1::Pod;
use kube::{
    api::{Api, ListParams},
    Client,
};

#[tokio::main]
async fn main() -> anyhow::Result<()> {
    let client = Client::try_default().await?;
    let pods: Api<Pod> = Api::default_namespaced(client);
    let pod_list = pods.list(&ListParams::default()).await?;
    let names = pod_list.into_iter()
             .map(|pod| pod.metadata.name.unwrap_or("".into()))
             .collect::<Vec<String>>();
    println!("{names:?}");
    Ok(())
}
```

```rust
use k8s_openapi::api::core::v1::Pod;
use kube::{
    api::{Api, ListParams},
    Client,
};

#[tokio::main]
async fn main() -> anyhow::Result<()> {
    let client = Client::try_default().await?;
    let pods: Api<Pod> = Api::default_namespaced(client);
    let pod_list = pods.list(&ListParams::default()).await?;
    let names = pod_list.into_iter()
            .map(|pod| pod.metadata.name.unwrap_or("".into()))
            .collect::<Vec<String>>();
    println!("{names:?}");
    Ok(())
}
```

Configure a client from default configuration (e.g., kubeconfig)

Wait for config to happen async

Error handling

```rust
use k8s_openapi::api::core::v1::Pod;
use kube::{
    api::{Api, ListParams},
    Client,
};

#[tokio::main]
async fn main() -> anyhow::Result<()> {
    let client = Client::try_default().await?;
    let pods: Api<Pod> = Api::default_namespaced(client);
    let pod_list = pods.list(&ListParams::default()).await?;
    let names = pod_list.into_iter()
            .map(|pod| pod.metadata.name.unwrap_or("".into()))
            .collect::<Vec<String>>();
    println!("{names:?}");
    Ok(())
}
```

Create an Api which is namespaced,
and uses the “default” namespace

Using the config we made in
the last step

Typed on a Pod

```rust
use k8s_openapi::api::core::v1::Pod;
use kube::{
    api::{Api, ListParams},
    Client,
};

#[tokio::main]
async fn main() -> anyhow::Result<()> {
    let client = Client::try_default().await?;
    let pods: Api<Pod> = Api::default_namespaced(client);
    let pod_list = pods.list(&ListParams::default()).await?;
    let names = pod_list.into_iter()
          .map(|pod| pod.metadata.name.unwrap_or("".into()))
          .collect::<Vec<String>>();
    println!("{names:?}");
    Ok(())
}
```

Use that API to perform a list

Default the listing parameters

Wait for this list to happen async

Error handling

Iterate over the list of pods

```rust
use k8s_openapi::api::core::v1::Pod;
use kube::{
    api::{Api, ListParams},
    Client,
};

#[tokio::main]
async fn main() -> anyhow::Result<()> {
    let client = Client::try_default().await?;
    let pods: Api<Pod> = Api::default_namespaced(client);
    let pod_list = pods.list(&ListParams::default()).await?;
    let names = pod_list.into_iter()
             .map(|pod| pod.metadata.name.unwrap_or("".into()))
             .collect::<Vec<String>>();
    println!("{names:?}");
    Ok(())
}
```

```rust
use k8s_openapi::api::core::v1::Pod;
use kube::{
    api::{Api, ListParams},
    Client,
};

#[tokio::main]
async fn main() -> anyhow::Result<()> {
    let client = Client::try_default().await?;
    let pods: Api<Pod> = Api::default_namespaced(client);
    let pod_list = pods.list(&ListParams::default()).await?;
    let names = pod_list.into_iter()
             .map(|pod| pod.metadata.name.unwrap_or("".into()))
             .collect::<Vec<String>>();
    println!("{names:?}");
    Ok(())
}
```

On each pod...

Grab the metadata.name

Name is an optional field on a Pod, so use the empty string if it’s not there

```rust
use k8s_openapi::api::core::v1::Pod;
use kube::{
    api::{Api, ListParams},
    Client,
};

#[tokio::main]
async fn main() -> anyhow::Result<()> {
    let client = Client::try_default().await?;
    let pods: Api<Pod> = Api::default_namespaced(client);
    let pod_list = pods.list(&ListParams::default()).await?;
    let names = pod_list.into_iter()
            .map(|pod| pod.metadata.name.unwrap_or("".into()))
            .collect::<Vec<String>>();
    println!("{names:?}");
    Ok(())
}
```

Collect the strings back into a list

Print them to STDOUT

```
$ cargo run
   Compiling k8s v0.1.0 (/Users/james/rust-k8s-https/k8s)
    Finished dev [unoptimized + debuginfo] target(s) in 0.79s
     Running `target/debug/k8s`
["debian"]
```

The kube crate can do a lot more than that!

- Create/Get/List/Patch/Update/Delete/Watch verbs
- CRD generation from Rust structs
- API objects that are typed to CRDs
- Watchers/Reflectors/Controllers
- +much more!

See their examples at https://github.com/kube-rs/kube/tree/main/examples

What does this mean for the Kubernetes ecosystem?

Language choice is one of the most impactful early decisions on any software project

Rust won’t be appropriate for every project or every team

Key Takeaway

Fear of Kubernetes compatibility isn’t a reason to avoid Rust

JETSTACK by Venafi

Thank you! 🚀

KubeCon | CloudNativeCon
Europe 2023