SUSECON digital 21

18-20 MAY 2021

Rust in Userspace: Systems programming in a cloud native world

Nick Gerace (@nickgeracehacks)

Software Engineer at SUSE

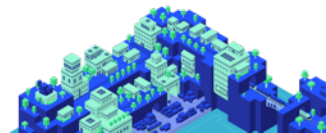
Let’s get you writing some cloud-native Rust code.

The goal of this session to get you up out your chair, go to your window, lean out, and yell “I’M A RUSTACEAN!”

We know these things!

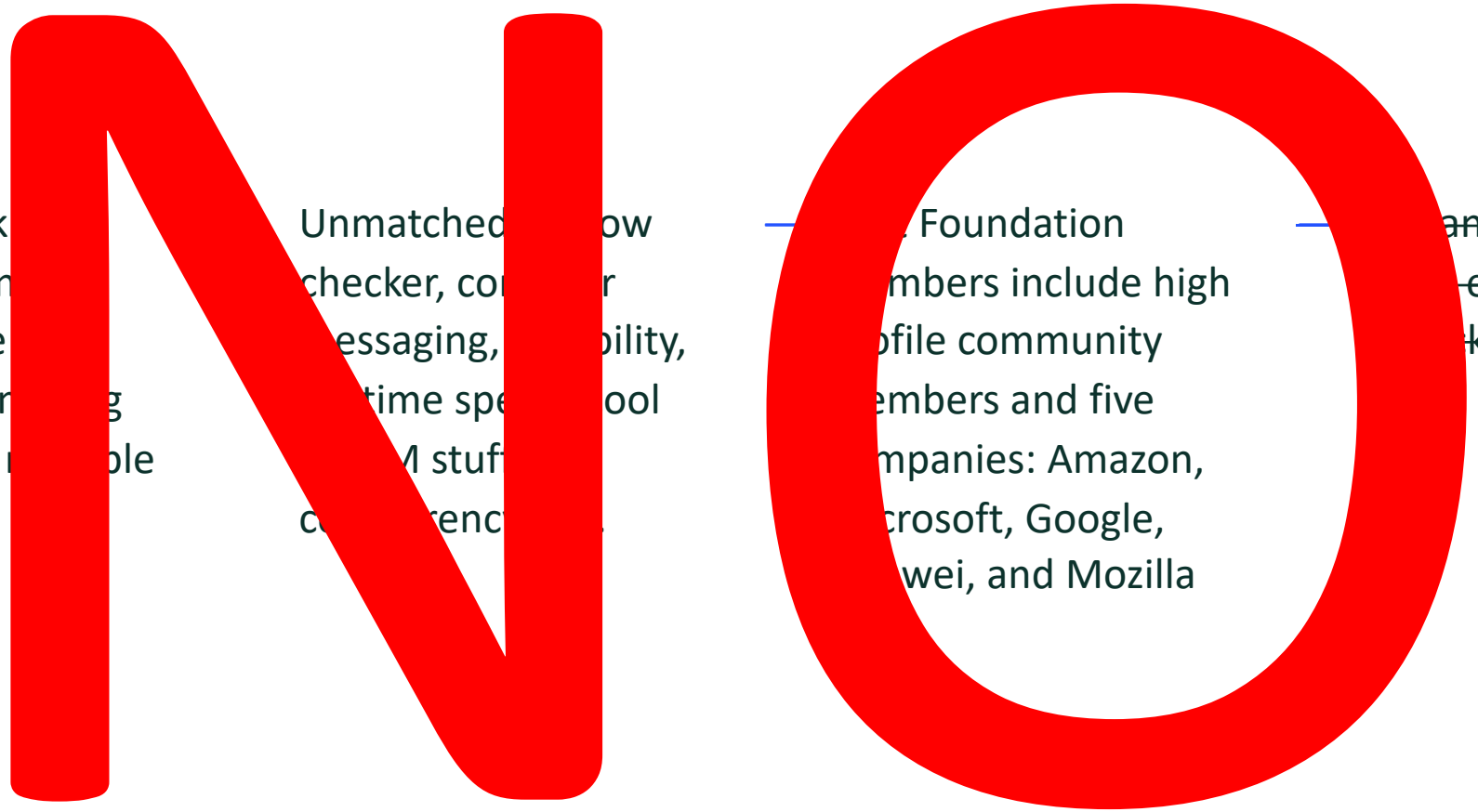
I can’t take it anymore!

- Topped Stack Overflow’s annual survey as the most loved programming language for multiple years now
- Unmatched borrow checker, compiler messaging, reliability, runtime speed, cool WASM stuff, concurrency, etc.
- Rust Foundation members include high profile community members and five companies: Amazon, Microsoft, Google, Huawei, and Mozilla
- ~~Has an overly eager and vocal engineer behind it, Nick Gerace~~

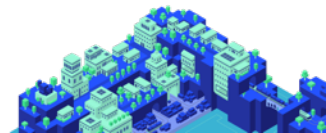


We know these things!

I can’t take it anymore!



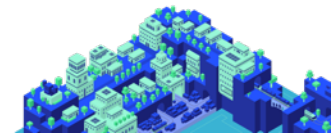
- Topped Stack Overflow’s an survey as the loved progra g language for ble years now
- Unmatched ow checker, co r essaging, bility, time spe ool M stuff c renc
- Foundation mbers include high ofile community embers and five mpanies: Amazon, crosoft, Google, wei, and Mozilla
- ~~an overly eager and engineer behind k Gerace~~



Today’s Focus: Userspace and Cloud-Native

Going off the beaten path

- Many organizations focus on Rust in embedded and OS use cases (C/C++ areas)
- Rust in a cloud-native world: containers, Kubernetes, immutable operating systems, openSUSE MicroOS, Rancher, etc.
- My experiences writing CLI applications, containerized services, and Kubernetes controllers Rust
- Where and how to get started: static binaries, recommended libraries, and general tips

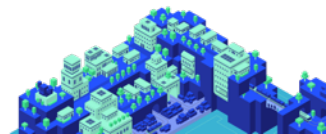


Today's Focus: Userspace and Cloud-Native

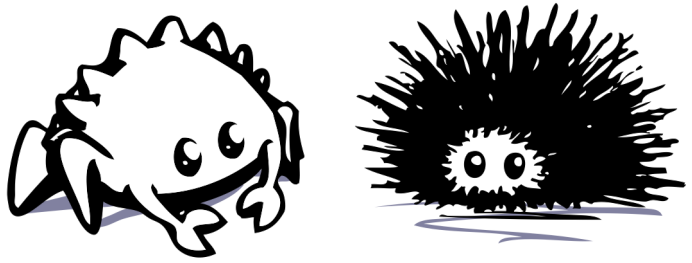
Going off the beaten path

YES

- Many c zations focus on n embedded OS cases (C/C+
- Rust in a c native world: cor rs, Kubernete operating openSUSE oOS, Rancher, e
- My experie vriting CLI applicati ontainerized s and Kubernetes controllers Rust
- Where and how to get started: static binaries, recommended libraries, d general tips



Agenda



1. Why Rust in Userspace?
2. Exploring Rust in a Cloud-Native World
3. My Experiences Writing a Kubernetes Operator and a Containerized Service in Rust
4. How and Where to Get Started with Cloud Native Rust

Why Rust in Userspace?

cargo build –release

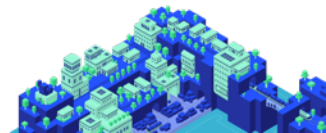
Building[===>] 1/4: suse-con-1 v2021.0.0

8

Let’s Agree on Some Definitions

From our dear friends at Wikipedia

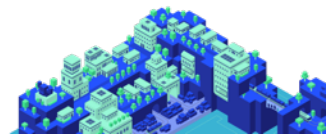
- **Userspace**
  - “A modern computer operating system usually segregates virtual memory into kernel space and user space.”
  - “Kernel space is strictly reserved for running a privileged operating system kernel, kernel extensions, and most device drivers.”
  - “User space is the memory area where application software and some drivers execute.”
- **Cloud-Native**
  - “[Build] and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds”
  - “...containers, microservices, serverless functions and immutable infrastructure, deployed via declarative code are common elements of this architectural style”



Humble Beginnings and Systems Programming

Rust's popularity in embedded and OS use cases

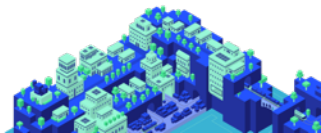
- Graydon Hoare's personal project from 2006 to **stable** in 2015
  - Borrow-checker: automatic memory management without a garbage collector
  - Mozilla Research, Servo engine for Firefox, safe/fast concurrency for a web browser
- Popularity in for OS and embedded use cases
  - **~70%** of Microsoft Security Response Center issues are **memory-safety related**
  - Ferrous Systems, Oxide Computer, System76, Redox OS



The Elephant in the Room: Why not just use Go?

You should, really!

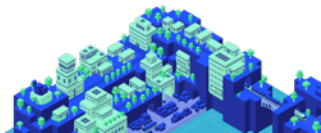
- Not my favorite question, but I understand why you want to know
  - You can love and use multiple languages <3
- Faster delivery and faster onboarding
  - &str versus String
- More features with less verbosity (goroutines)
- More mature ecosystem (barring modules in 2019)
- Not performance-bound, but still lightning fast!
- Existing ecosystem of packages, modules and libraries (helm, client-go, etc.)
- Kubernetes and Docker are written in Go
  - Ecosystem keystones



Why Rust in Userspace?

Again, you can love and use multiple languages <3

- What we know
  - Best of both worlds: performance without a garbage collector and behind the safety of a borrowing system
  - Strong focus on concurrency by using lifetimes to avoid collisions
    - All the options: OS threads, multi-processing, green threads, etc.
- Result and Option types (begone, “nil pointer exception”!)
  - No concept of ”null” or “nil” combined with advanced error handling
- Cross-platform **cargo** over **make**
- **crates.io** and **docs.rs (cargo publish)**
  - Automatic documentation and package management
- *Ability* to use “no_std” and bare metal (no “runtime” required)
- Minimal standard library
- Compile-time reliability
- Functional programming features (including closures!)
- Friendly compiler messages, and a best-in-class community
  - *Yes, these matter, and arguably more than most ”technical” advantages*



Exploring Rust in a Cloud-Native World

cargo build –release

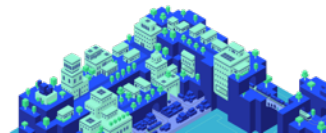
Building[===>] 2/4: suse-con-2 v2021.0.0

13

Exploring Cloud-Native Rust Projects

It’s only the beginning

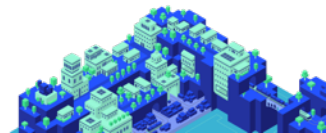
- **AWS Firecracker:** microVMs for multi-tenant, minimal-overhead execution of container and function workloads
- **AWS Bottlerocket OS:** Linux-based operating system meant for hosting containers
- **TiKV:** distributed, and transactional key-value database
- **Sonic:** schema-less search backend (alternative to Elasticsearch)
- **Krustlet:** Kubelet for running WASM



SUSE and Cloud-Native Rust in the Future?

Nick's galaxy brainstorm (or brainfart)

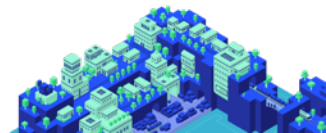
- **openSUSE MicroOS:** immutability and reproducible builds driven by a language as fast as C/C++ without the memory safety implications (isolated execution, transactional updates, functional similarities, etc.)
- **openSUSE Build System:** lightning fast and safe concurrent builds; an opportunity to take full advantage of hardware
- **Rancher v2.x**: Rust services and controllers for runtime consistency and reduced technical debt, Rust CLI applications to interface with Rancher for edge deployment, privileged applications for security
- **Harvester:** potential use case for microVMs, bare metal access, advanced provisioning and tracing... the sky is the limit here



The Technology Behind Them

We have technology!

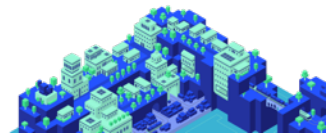
- **WebAssembly:** near-native code execution speed in the web browser
- **anyhow and eyre:** advanced error handling at your fingertips
- **serde, reqwest, actix, warp:** serialization and servers
- **tokio and rayon:** asynchronous and concurrent programming made easy
- **clux/kube-rs:** client for Kubernetes in the style of a more generic client-go
- **clux/controller-rs:** Kubernetes controller/operator leveraging kube-rs



The Technology Behind Them

We have technology!

- **WebAssembly:** near-native code execution speed in the web browser
- **anyhow and eyre:** advanced error handling at your fingertips
- **serde, reqwest, actix, warp:** serialization and servers
- **tokio and rayon:** asynchronous and concurrent programming made easy
- **clux/kube-rs:** client for Kubernetes in the style of a more generic client-go
- **clux/controller-rs:** Kubernetes controller/operator leveraging kube-rs
- Thank you to Eirik Albrigtsen (@sszynrae)!



My Experiences Writing a Kubernetes Operator and a Containerized Service in Rust

cargo build –release

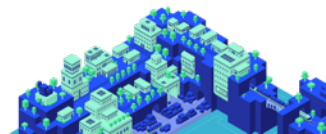
Building[===>] 3/4: suse-con-3 v2021.0.0

18

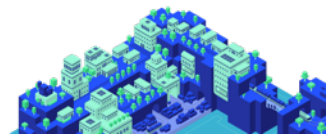
The containerized service: kimager

I should probably update this thing

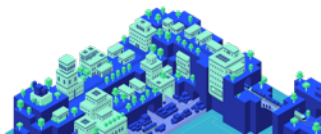
- Needed to log the existence of images on a Kubernetes cluster
  - ”first time I’ve seen that image”
  - “the last container using that image has been deleted”
- Persistent service that is deployed via a Helm chart
  - Only prerequisite: **helm**
- Using hashing and bi-directional maps to store metadata rather than large objects
  - Everything is entirely in-memory
  - Will be replaced by an undirected graph (someday...)
- ~3MB compressed Docker image without stripping debug symbols
  - Static binary using **musl** instead of **glibc**



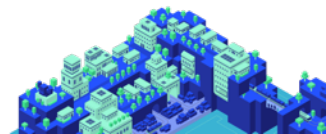
```rust
#[tokio::main]
async fn main() -> Result<()> {
    if env::var("RUST_LOG").is_err() {
        env::set_var("RUST_LOG", "info");
    }
    env_logger::builder().format_module_path(false).init();
    debug!("Starting watcher...");
    kimager::watch(Client::try_default().await?).await?;
    debug!("Watcher has stopped.");
    Ok(())
}
```

```rust
pub async fn watch(client: Client) -> Result<()> {
    let pods: Api<Pod> = Api::all(client.clone());
    let wp = ListParams::default().timeout(0);
    let mut event_driver = EventDriver::new();
    loop {
        debug!("Creating stream with Pods API abstraction...");
        let mut stream = pods.watch(&wp, "0").await?.boxed();
        debug!("Watching events...");
        while let Some(status) = stream.try_next().await? {
            match status {
                WatchEvent::<Pod>::Added(pod) => {
                    event_driver.new_event(pod, EventType::Added).await
                }
                WatchEvent::<Pod>::Deleted(pod) => {
                    event_driver.new_event(pod, EventType::Deleted).await
                }
                WatchEvent::<Pod>::Error(report) => error!("{}", report),
                _ => {}
            }
        }
        warn!("Restarting watcher...");
    }
}
```

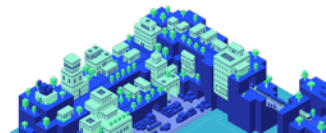
```
FROM clux/muslrust:stable AS build
WORKDIR /build/
COPY Cargo.toml Cargo.toml
COPY Cargo.lock Cargo.lock
COPY src/ src/
RUN cargo build --release && strip /build/target/x86_64-unknown-linux-musl/release/kimager

FROM scratch
WORKDIR /bin/
COPY --from=build /build/target/x86_64-unknown-linux-musl/release/kimager .
ENTRYPOINT ["/bin/kimager"]
```

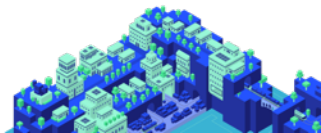
The Kubernetes controller: krunvm-operator

I should probably release this thing

- Unfinished SUSE Hack Week 2021 project
  - *(In my defense, I was on 75% vacation that week while helping with the Hack Week Rust Bootcamp -- shoutout to Ferrous Systems!)*
- Launching microVMs by creating a custom resource, which is then reconciled by the krunvm-operator
  - **krunvm** and **libkrun:** creating and managing isolated, lightweight, microVMs on Linux (KVM) and macOS (Hypervisor.framework)
- Functional and deployable via a Helm chart
  - Runs in a privileged container to use **libkrun** on the host insecurely
  - I never claimed that this project made sense



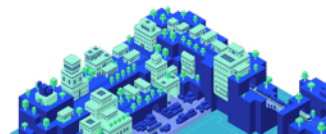
```rust
let client = ctx.get_ref().client.clone();
ctx.get_ref().state.write().await.last_event = Utc::now();
let name = Resource::name(&foo);
let ns = Resource::namespace(&foo).expect("foo is namespaced");
let foos: Api<Foo> = Api::namespaced(client, &ns);

let new_status = Patch::Apply(json!({
    "apiVersion": "clux.dev/v1",
    "kind": "Foo",
    "status": FooStatus {
        is_bad: foo.spec.info.contains("bad"),
        //last_updated: Some(Utc::now()),
    }
}));
let ps = PatchParams::apply("cntrlr").force();
let _o = foos
    .patch_status(&name, &ps, &new_status)
    .await
    .map_err(Error::KubeError)?;
```

Reflections on the Service and the Controller

Skunkworks, but with crabs

- Biased, but a very good experience
  - Using tokio and anyhow affirmed my stance on the minimal standard library
  - Using kube-rs felt even better than client-go in some ways
  - Result and Option types give me heightened confidence at runtime
  - Lighting fast and small binaries (it’s difficult to go faster and smaller without sacrificing security!)
- Missing pieces
  - The Go modules for Rancher, Helm, Tekton, etc. were missed
  - The compiler is not known for its speed, and building static binaries in a container exacerbates that



How and Where to Get Started with Cloud Native Rust

cargo build –release

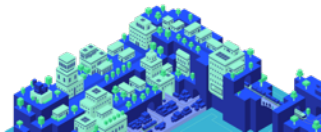
Building[===>] 4/4: suse-con-4 v2021.0.0

26

How to Get Started: Setting Up

The tools of the trade

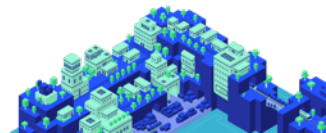
1. Install rustup.rs and rust-analyzer in order to access multiple toolchains, and the best IDE-like experience with your favorite editor
2. Read the “The Rust Programming Language Book” (at least, the first 12 chapter including the first big project)
3. Try to install a few Rust crates (libraries and applications) to get a feel for cargo
    1. $ **cargo install ripgrep && rg –help**
4. Try out some small ideas and proofs of concept with the playground: https://play.rust-lang.org/
5. Create a small CLI application using popular libraries from “Awesome Rust” and the ”areweXyet” sites
6. Catch up on popular Rust bloggers and videographers (too many to list!)
    - Jon Gjengset, Jane Lusby, Amos (fasterthanli.me), Read Rust from Wesley Moore, etc.
    - Streamers: https://github.com/jamesmunns/awesome-rust-streaming



How to Get Started: Writing Cloud-Native Rust

Let’s try Kubernetes!

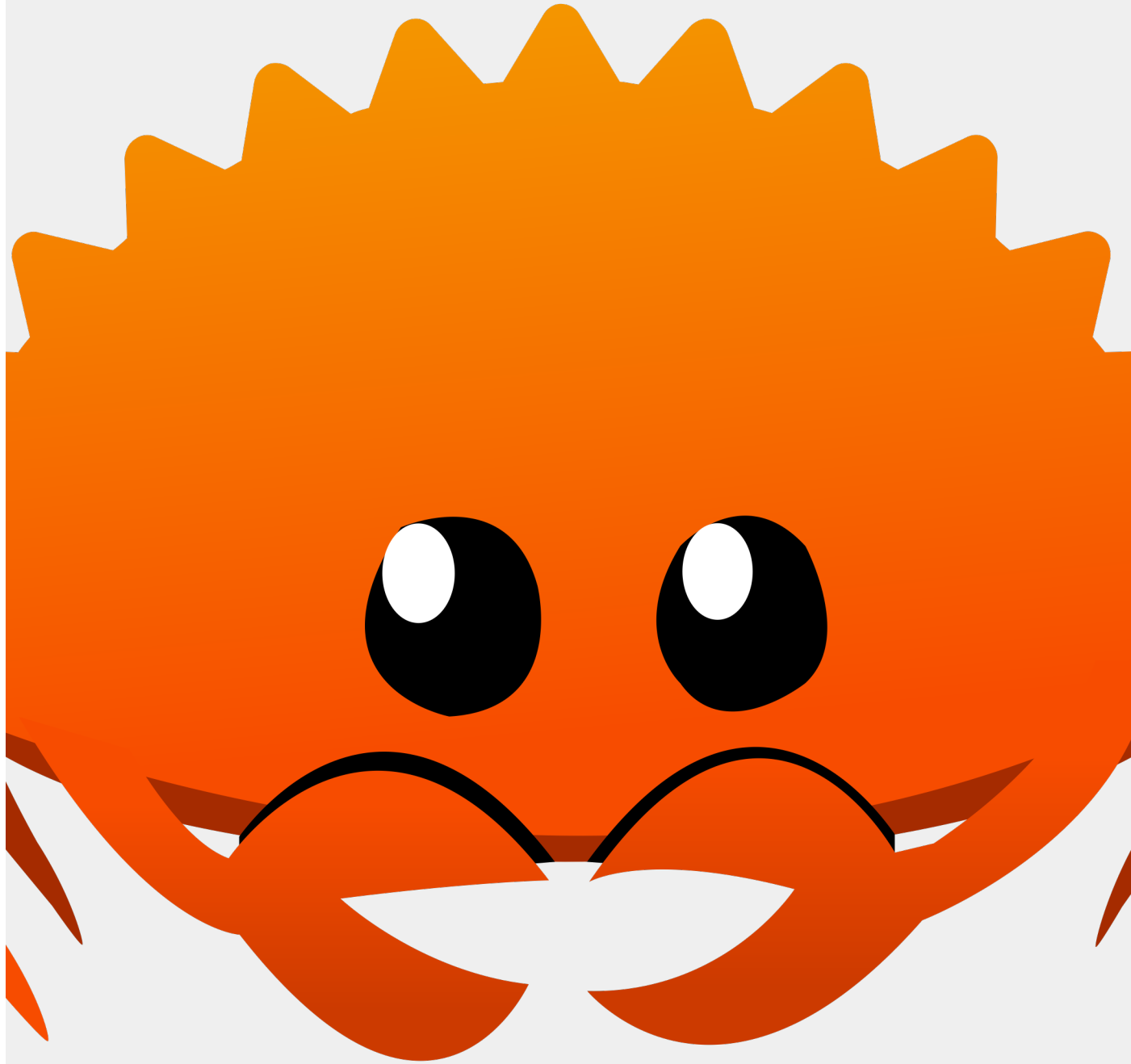
- **Creating a “Kubernetes-unaware” containerized daemonset, deployment or job**
  - Your first containerized Rust program does not need to do anything Kubernetes-specific
  - Try porting an existing, non-persistent, CLI-driven application to a Kubernetes job
  - You can use a musl builder (or Windows nanoserver image; for the “chaotic-neutral”) for maximum efficiency
- **Creating a “Kubernetes-aware” containerized daemonset, deployment or job**
  - If starting with a tokio-rs main async function, *do not be afraid to use blocking calls*
  - Even if you want to a kube-rs client, you do not have to write a service
    - Alternatively, write a Kubernetes job for non-persistent execution
  - Create a client with kube-rs and follow the examples from its repository,
    - You can focus on the examples using the Pods and Jobs APIs
- **Creating a Kubernetes controller**
  - Fork *clux/controller-rs* and add your code to the reconcile function
  - Creating a Kubernetes job or deployment when a CR is reconciled can be a good place to start
    - Perhaps, you would like to focus on the application/service rather than the controller/operator



Thank You!

Nick Gerace (@nickgeracehacks)

Software Engineer at SUSE



Credits in Order of Appearance

Images on the left, informational sources on the right

- https://www.rust-lang.org/logos/cargo.png
- https://rustacean.net/more-crabby-things/safeandunsafe.svg
- https://rustacean.net/assets/rustacean-orig-noshadow.svg
- https://web.archive.org/web/20160609195720/https://www.rust-lang.org/faq.html#project
- https://msrc-blog.microsoft.com/2019/07/22/why-rust-for-safe-systems-programming/
- https://thenewstack.io/rust-vs-go-why-theyre-better-together/