SUSE

11 DECEMBER 2020

Writing Policies for Kubewarden

This is a subheading if you wish to add further detail to the slide. Ignore if you don't.

Agenda

1. Who am I?
2. What is Kubewarden?
3. Policies
   - Writing our first policy.
   - Deploying it to a cluster.
4. Next Steps

Who am I?

This isn’t me

This is me

Robert Sirchia

I'm a Senior Technical Evangelist at SUSE. Part of the SUSE & Rancher community. I specialize in cloud-native development and cloud operations.

I am all about learning and sharing this knowledge with others.

Follow me on Twitter: @robertsirc

What is Kubewarden?

Kubewarden

Kubewarden is a policy engine for Kubernetes.

Its mission is to simplify the adoption of policy-as-code.

Policy Developers

Write policies in your favorite language* not one specific to Kubewarden.

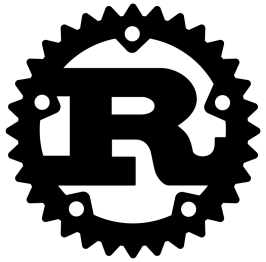
Kubernetes Operators

Policies can be distributed using container registries use your existing infrastructure and processes.

Highlights of Kubewarden

Open-Source Hub of existing policies to download and use.

Support of multiple languages such as Rust, Rego, and Go



Once a Wasm is built you can run it anywhere.

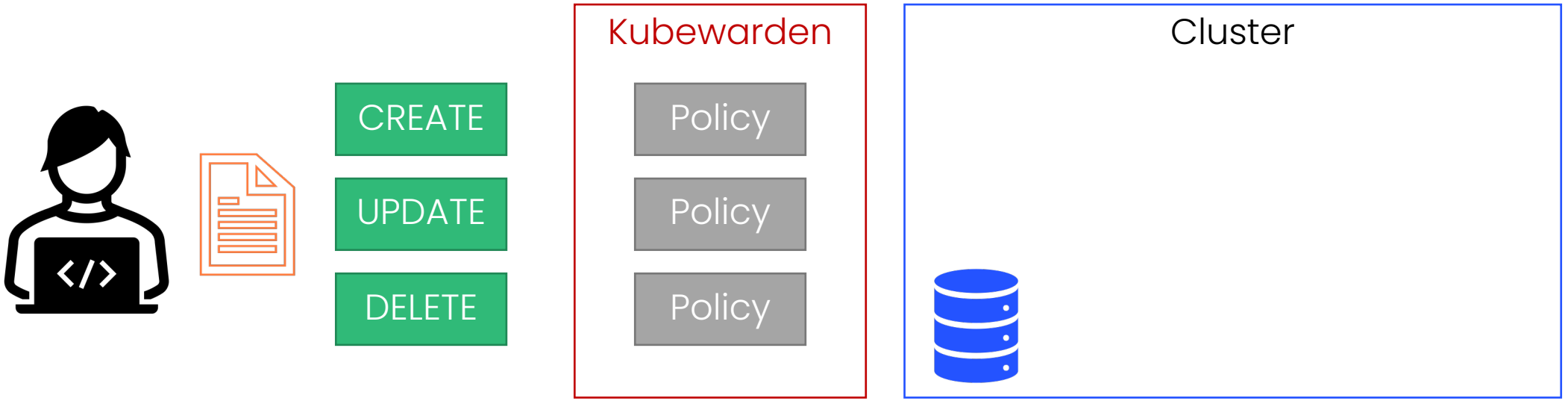
Policies

What is a policy?

In the context of Kubewarden

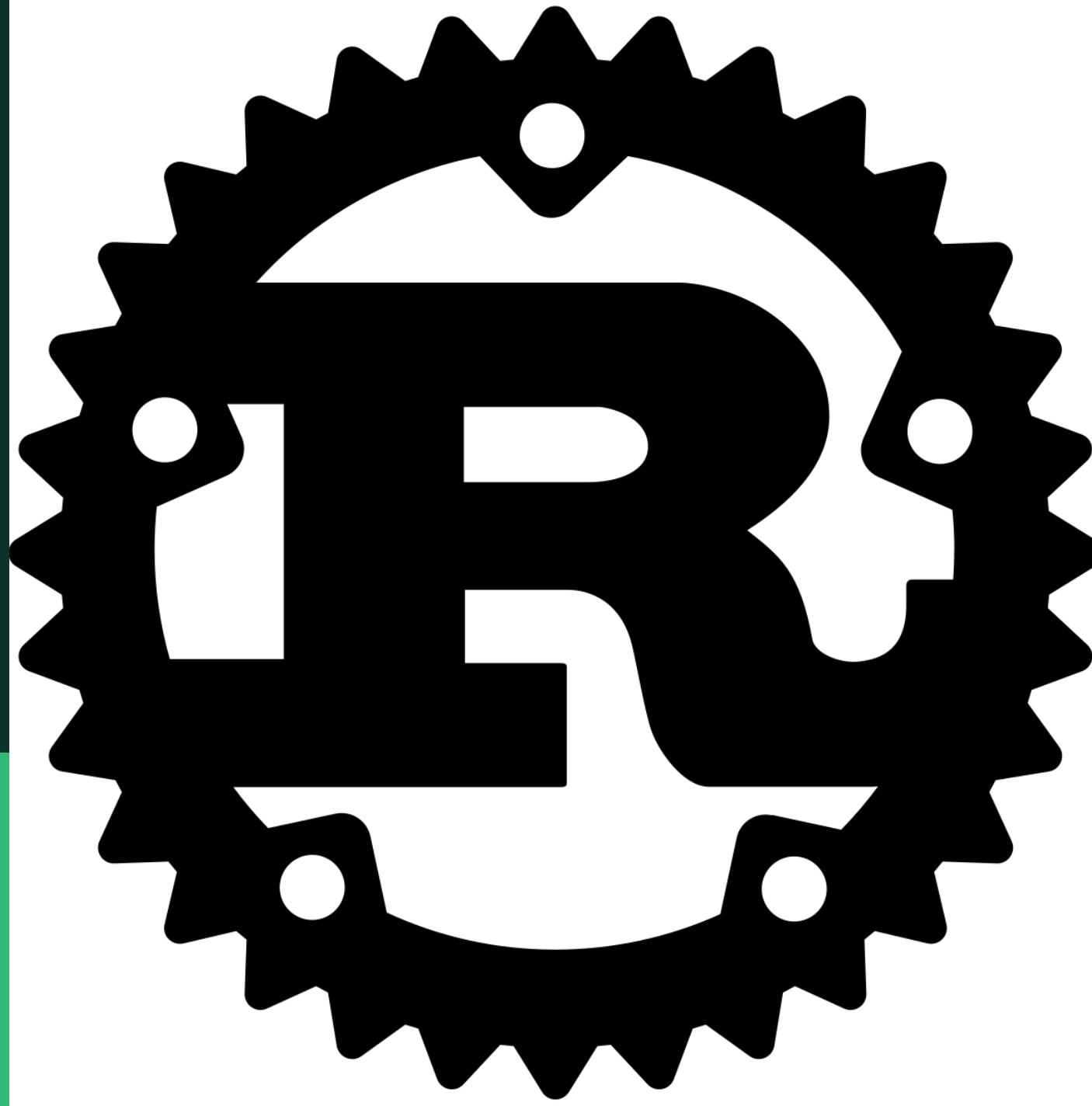
- These are small compiled binaries that do a specific task
- Delivered as WebAssembly binaries.
- All run in Kubewarden's policy-server.

How policies work?



Writing our first Policy

For this we will be using Rust.



What are we building?

Policy that limit's the CPU of a container.

Setup and Configuration

- VSCode
  - Rust Extension
- Install
- Verify

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

```
rustup -V
rustup 1.24.3 (ce5817a94 2021-05-31)
info: This is the version for the rustup toolchain manager, not the rustc compiler.
info: The currently active `rustc` version is `rustc 1.57.0 (f1edd0429 2021-11-29)`
```

Cluster Installation

- Rancher Desktop (or another cluster)
- Helm
- Kwctl
- Install Kubewarden


```
helm repo add kubewarden https://charts.kubewarden.io
kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.5.3/cert-manager.yaml
kubectl wait --for=condition=Available deployment --timeout=2m -n cert-manager --all
helm install --wait -n kubewarden --create-namespace kubewarden-crds kubewarden/kubewarden-crds
helm install --wait -n kubewarden kubewarden-controller kubewarden/kubewarden-controller
```

Creating a Rust Policy

- Install cargo-generate
- Generate our project from a Rust template

```
cargo install cargo-generate
```

```
cargo generate --git https://github.com/kubewarden/policy-rust-template \
               --branch main \
               --name pod-sizer
```

Updating the Settings for the Policy

```rust
#[test]
fn accept_settings_with_cpu_limits_set() -> Result<(), ()> {
    let cpu_limits = String::from("0.1");
    let settings = Settings { cpu_limits };

    assert!(settings.validate().is_ok());
    Ok(())
}

#[test]
fn reject_settings_with_no_cpu_limits_set() -> Result<(), ()> {
    let cpu_limits = String::new();
    let settings = Settings { cpu_limits };

    assert!(settings.validate().is_err());
    Ok(())
}
```

```rust
#[derive(Serialize, Deserialize, Default, Debug)]
#[serde(default)]
pub(crate) struct Settings {
    pub cpu_limits: String,
}
```

```rust
impl kubewarden::settings::Validatable for Settings {
    fn validate(&self) -> Result<(), String> {
        info!(LOG_DRAIN, "starting settings validation");
        if self.cpu_limits.is_empty() {
            Err(String::from("No CPU limits is set."))
        } else {
            Ok(())
        }
    }
}
```

Updating the Policy

```
let pod = match serde_json::from_value::<apicore::Pod>(validation_request.request.object) {
        Ok(pod) => pod,
        Err(_) => return kubewarden::accept_request(),
};
```

```
#[derive(Debug, PartialEq)]
enum PolicyResponse {
    Accept,
    Reject(String),
}
```

```
fn validate_pod(pod: apicore::Pod, settings: settings::Settings) -> Result<PolicyResponse> {
    let pod_spec = pod.spec.ok_or_else(|| anyhow!("invalid pod spec"))?;

    let all_containers = pod_spec.containers.into_iter().all(|container| {
        container_at_or_under_limit(container, settings.cpu_limits.clone())
    });

    if all_containers {
        Ok(PolicyResponse::Accept)
    } else {
        Ok(PolicyResponse::Reject("Rejected".to_string()))
    }
}
```

Updating the Policy cont.

```
match validate_pod(pod, settings)? {
    PolicyResponse::Accept => kubewarden::accept_request(),
    PolicyResponse::Reject(message) => kubewarden::reject_request(Some(message), None),
}
```

```
fn container_at_or_under_limit(container: apicore::Container, settings_cpu_limit: String) -> bool {
    true
}
```

Policy Testing

```rust
use std::collections::BTreeMap;

use k8s_openapi::apimachinery::pkg::api::resource::Quantity as apimachinery_quantity;

#[test]
fn pods_at_limit_set() -> Result<()> {
  let cpu_limits = String::from("1.5");

  let mut _limits: BTreeMap<String, apimachinery_quantity> = BTreeMap::new();
  _limits.insert(String::from("cpu"), apimachinery_quantity { 0: String::from("1.5") });

  assert_eq!(
      validate_pod(
          apicore::Pod {
              spec: Some({
                  apicore::PodSpec {
                      containers: vec![
                          apicore::Container {
                              resources: Some({
                                  apicore::ResourceRequirements {
                                      limits: Some(_limits),
                                      ..apicore::ResourceRequirements::default()
                                  }
                              }),
                              ..apicore::Container::default()
                          }
                      ],
                      ..apicore::PodSpec::default()
                  }
              }),
              ..apicore::Pod::default()
          },
          Settings { cpu_limits }
      )?,
      PolicyResponse::Accept
  );
  Ok(())
}
```

Building the Policy

```
rustup target add wasm32-unknown-unknown
```

```
make build
```

```
target/wasm32-unknown-unknown/release/pod_sizer.wasm
```

Annotating the Policy

```
kwctl annotate target/wasm32-unknown-unknown/release/pod_sizer.wasm --metadata-path metadata.yml --
output-path annotated-pod_sizer.wasm
```

```
kwctl run --request-path test_data/pod_creation_cpu_1.json --settings-json '{ "cpu_limits": "1.0"}'
target/wasm32-unknown-unknown/release/pod_sizer.wasm
```

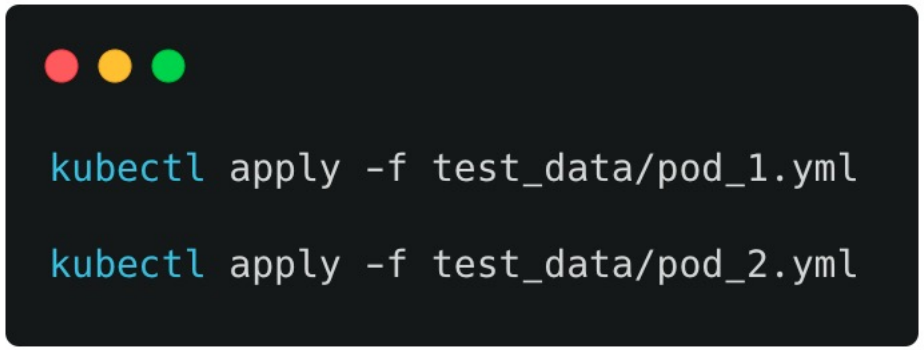
Deploying

```yaml
apiVersion: policies.kubewarden.io/v1alpha2
kind: ClusterAdmissionPolicy
metadata:
  name: pod-sizer
spec:
  module: registry://ghcr.io/robertsirc/rust-wasm-labs/pod_sizer:v0.0.1
  rules:
  - apiGroups: [""]
    apiVersions: ["v1"]
    resources: ["pods"]
    operations:
    - CREATE

  mutating: false
  settings:
    cpu_limits: "1.0"
```

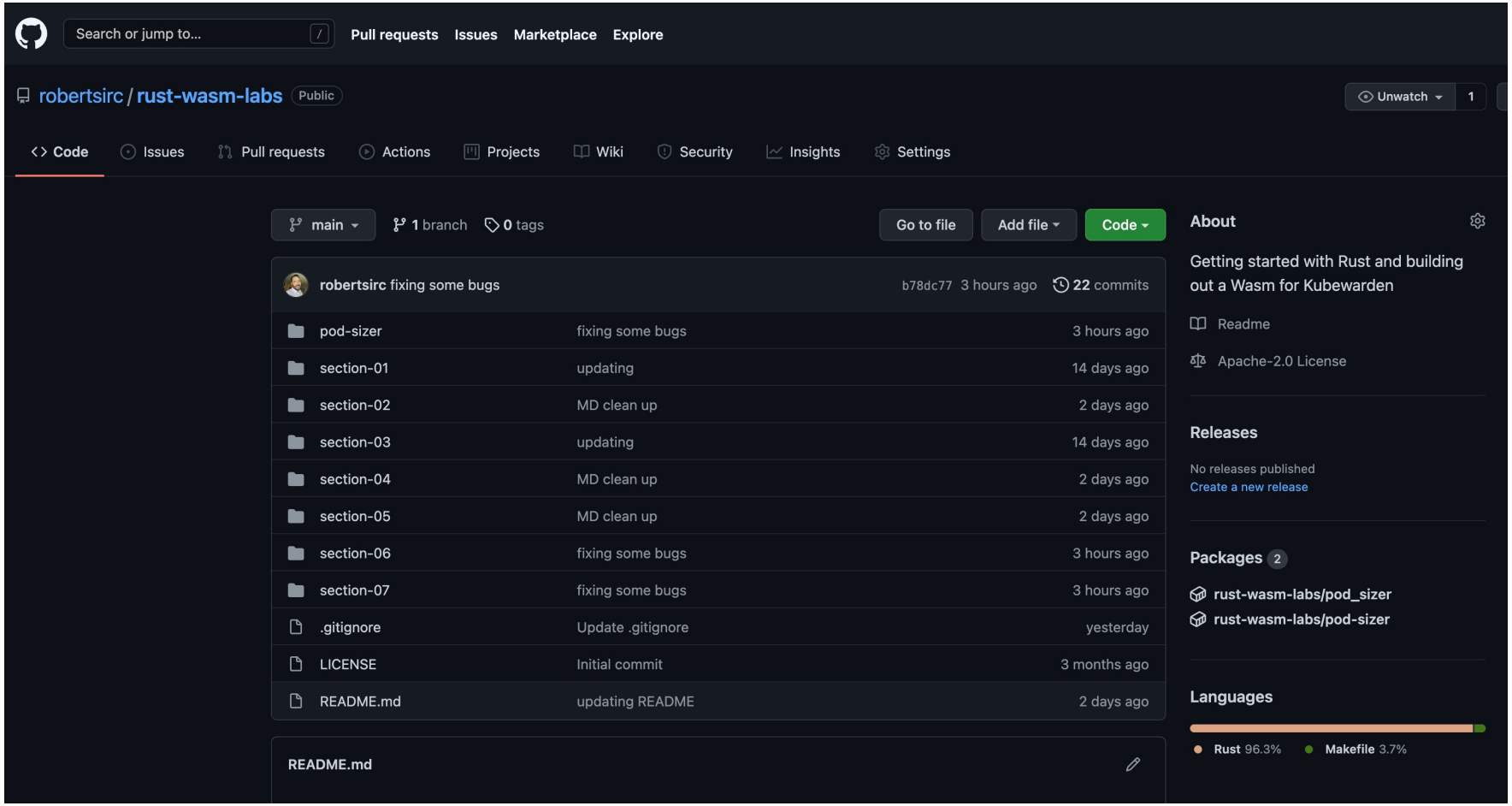
```
$ kubectl apply -f pod-sizer.yml
$ clusteradmissionpolicy.policies.kubewarden.io/pod-sizer created
```

Testing on a Cluster



Next Steps

Try this yourself!



https://github.com/robertsirc/rust-wasm-labs

Questions?

If not, you can ask after the session

Additional Resources

- Kubewarden
- Docs
- Rust
- Rancher Desktop
- Community
- Slack

Thank You

For more information, contact SUSE at:

+1 800 796 3700 (U.S./Canada)

Maxfeldstrasse 5

90409 Nuremberg

www.suse.com