Automation Made Simple with Rust

Morgan (pcwizz)

Rust and Tell, September 2019

Introduction

- Statefull services
  - PostgreSQL
  - Redis
  - RabbitMQ
- Multi-environment
  - Public Cloud
  - On prem
  - VMs or containers
- Lifecycle
  - Provisioning
  - Updating
  - Scaling
  - Backup

Challenges

Deceptive similarity

The states any one service can be in is generic among all other services, however the actions required to transition are generally unique to each service.

User abstraction

Automation is supposed to enable developers to use a service without remembering how to maintain it.

Scale

Testing 2 services on a laptop should work just as well as 10000 on a public cloud.

Goals

Maintainability

We want a sensibly DRY codebase.

Appropriate abstraction

We want to present a high level status.

Reliability

Out automation system shouldn't require much manual operation.

Rust's answers

Generics

Common code can be shared safely.

Tuple enums

We can express state in layers including relevant information as required.

Safety

If our code compiles then we can be pretty confident it will just keep running.

Example #1: Utilising traits and generics

```rust
pub trait Updatable {
    fn update_available(&self) -> bool;

    fn update(&self) -> Status;
}
```

Example #1: Utilising traits and generics

```
pub trait Persistant {}

pub trait LiveSnapshotable {}

impl<T> Updatable for T where T: LiveSnapshotable + Persistant {
    fn update_available(&self) -> bool { unimplemented!() }

    fn update(&self) -> Status { unimplemented!{} }
}
```

Example #2: Abstracting state with enums

```rust
pub enum Status {
    New,
    Deployed(DeployedStatus),
    Destroyed
}

pub enum DeployedStatus {
    Running(RunningStatus),
    Failing
}
```

Example #2: Abstracting state with enums

```
pub enum RunningStatus {
    HighLoad(Box<dyn RunningInformation>),
    GettingFull(Box<dyn RunningInformation>),
    Normal(Box<dyn RunningInformation>)
}

pub trait RunningInformation {
    fn query_latency_devation(&self) -> i16;
    fn disk_usage(&self) -> Option<Vec<u8>>;
    fn cpu_load(&self) -> Vec<u8>;
}
```

Example #2: Abstracting state with enums

```
match self.status() {
    New => Some(self.deploy()),
    Deployed(deployed) => match deployed {
            Running(running) => match running {
                Normal if update_available => Some(self.update()),
                Normal => Some(Deployed(Running(Normal))),
                _ => Some(self.fix())
            },
            Failing => Some(self.fix())
    },
    Destroyed => None
}
```

Encouraging Rust for Devops

Go dominates this field because:

- Mature libraries
- Simplicity
- Convenience

Rust pain points

- Finding mature libraries with nice APIs
- Clear documentation with examples
- Getting used to the borrow checker