"This Rust Project could have been a Bash Script"

Revelations of writing simple Rust

Oliver Calder | `snapd`

Hastily made via `marp`

Some background about me

Some background about me

- Likes Rust

Some background about me

- Likes Rust
- Likes to self-host things

Some background about me

- Likes Rust
- Likes to self-host things
- Is sometimes lazy

Next steps obvious (?)

Next steps obvious (?)

Write a reverse proxy manager

Next steps obvious (?)

Write a reverse proxy manager

in Rust

Introducing: `nrpctl` *

`nrpctl` key features:

`nrpctl` key features:

- Add new proxy: `nrpctl add cloud.calder.dev 5000`

`nrpctl` key features:

- Add new proxy: `nrpctl add cloud.calder.dev 5000`
- Remove proxy: `nrpctl remove cloud.calder.dev` (wow)

`nrpctl` key features:

- Add new proxy: `nrpctl add cloud.calder.dev 5000`
- Remove proxy: `nrpctl remove cloud.calder.dev` (wow)
- Configuration options: ports, SSL, gzip, IPv6, …

BUT

I hear what you're thinking

BUT

I hear what you're thinking

"Doesn't Caddy do all of this in a declarative[1] way?"

[1] better

Yes.

Yes.*

*I didn't know Caddy existed until someone told me about it yesterday

Some background about me

- Likes Rust
- Likes to self-host things
- Is sometimes lazy

Some background about me

- Likes Rust
- Likes to self-host things
- Is sometimes lazy
- *Doesn't know Caddy*

anyway...

Reverse proxies are all pretty much the same:

Reverse proxies are all pretty much the same:

- Listen for this domain

Reverse proxies are all pretty much the same:

- Listen for this domain
- Forward to this port

Reverse proxies are all pretty much the same:

- Listen for this domain
- Forward to this port
- Maybe set up SSL certs

Reverse proxies are all pretty much the same:

- Listen for this domain
- Forward to this port
- Maybe set up SSL certs
- Maybe set some special content options

Conclusion:

Conclusion:

This could just be a bash script which dumps a template to a file, substitutes a few variables, ...

so I created a new Rust project and imported `clap` and discovered

The joys of Clap-Driven Development (CDD)

(revelation #1)

```rust
#[derive(Parser)]
#[command(version, about, long_about = None)]
pub struct Cli {
    /// Specify the path to the nrpctl configuration TOML file
    #[arg(
        short,
        long,
        value_name = "FILE",
        default_value_t = String::from("/etc/nrpctl/config.toml")
    )]
    pub config: String,

    #[command(subcommand)]
    pub command: Command,
}
```

```rust
#[derive(Subcommand)]
pub enum Command {
    /// Initialize a new nrpctl configuration file (args ignored if running in a snap)
    Init {
        /// The directory in which to write nginx configurations
        #[arg(
            short,
            long,
            value_name = "DIR",
            default_value_t = String::from("/etc/nginx/sites-enabled")
        )]
        sites_enabled_dir: String,
    },

    /// Display information about active reverse proxies
    Status,

    /// Render nginx configurations for reverse proxies
    Render {
        /// Render only the configuration for the given domain
        listen_domain: Option<String>,
    },

    // ... snip ...
}
```

```rust
#[derive(Subcommand)]
pub enum Command {
    // ... snip ...

    /// Add a new reverse proxy
    Add {
        /// The source domain for which to listen (e.g. `cloud.mydomain.com`)
        listen_domain: String,

        /// The port on which to listen for requests
        #[arg(short, long, default_value_t = 80)]
        listen_port: u16,

        /// The destination domain to which to forward requests
        #[arg(long, default_value_t = String::from("localhost"))]
        dest_domain: String,

        /// The destination port to which to forward matching requests
        dest_port: u16,

        /// Set the client max body size for incoming requests
        #[arg(long)]
        client_max_body_size: Option<String>,

        /// Enable gzip for responses from this domain
        #[arg(long)]
        gzip: bool,

        /// Listen on IPv6 as well as IPv4
        #[arg(long)]
        ipv6: bool,

        /// Set up SSL encryption (HTTPS) using certbot, with or without redirection from port 80.
        /// If `listen-port` is 80 and SSL is set to redirect, changes `listen-port` to 443.
        #[arg(long)]
        ssl: Option<SSLSelection>,
    },

    // ... snip ...
}
```

```rust
#[derive(Subcommand)]
pub enum Command {
    // ... snip ...

    /// Get a configuration value for a reverse proxy
    Get {
        /// The source domain for this reverse proxy
        listen_domain: String,

        /// The configuration to get (gets all if unspecified)
        key: Option<ProxySettingKey>,
    },

    /// Set a configuration value for a reverse proxy
    Set {
        /// The source domain for this reverse proxy
        listen_domain: String,

        /// The configuration key to set
        key: ProxySettingKey,

        /// The value to set for the configuration key
        value: String,
    },

    /// Unset a configuration value for a reverse proxy
    Unset {
        /// The source domain for this reverse proxy
        listen_domain: String,

        /// The configuration key to unset
        key: ProxySettingKeyOptional,
    },
}
```

```rust
/// ProxySetting defines the keys and corresponding values which can be retrieved or set for a
/// given proxy.
pub enum ProxySetting {
    ListenPort(u16),
    DestDomain(String),
    DestPort(u16),
    ClientMaxBodySize(Option<String>),
    Disabled(Option<bool>),
    Gzip(Option<bool>),
    Ipv6(Option<bool>),
    Ssl(Option<SSLSelection>),
}

/// ProxySettingKey defines the keys which can be retrieved or set for a given proxy.
#[derive(clap::ValueEnum, strum::EnumIter, Clone, Copy, Serialize)]
#[serde(rename_all = "kebab-case")]
pub enum ProxySettingKey {
    /// The port on which to listen for requests
    ListenPort,
    /// The destination domain to which to forward requests
    DestDomain,
    /// The destination port to which to forward requests
    DestPort,
    /// The maximum acceptable request body size (e.g. "512m")
    ClientMaxBodySize,
    /// Whether the reverse proxy is disabled
    Disabled,
    /// Whether to gzip response content
    Gzip,
    /// Whether to listen on IPv6 as well as IPv4
    Ipv6,
    /// Whether to set up an SSL certificate, with or without redirection from port 80.
    /// If `listen-port` is 80 and SSL is set to redirect, changes `listen-port` to 443.
    /// [possible values: false, no-redirect, redirect]
    Ssl,
}
```

```rust
/// Set the given key to the given value and return the new setting.
pub fn set_key(&mut self, key: ProxySettingKey, value: String) -> Result<ProxySetting> {
    let setting = key.parse(value)?;
    match &setting {
        ProxySetting::ListenPort(port) => self.listen_port = *port,
        ProxySetting::DestDomain(domain) => self.dest_domain = domain.clone(),
        ProxySetting::DestPort(port) => self.dest_port = *port,
        ProxySetting::ClientMaxBodySize(size) => self.client_max_body_size = size.clone(),
        ProxySetting::Disabled(val) => self.disabled = *val,
        ProxySetting::Gzip(val) => self.gzip = *val,
        ProxySetting::Ipv6(val) => self.ipv6 = *val,
        ProxySetting::Ssl(val) => self.ssl = *val,
    };
    Ok(setting)
}
```

```rust
impl ProxySettingKey {
    pub fn parse(&self, value: String) -> Result<ProxySetting> {
        Ok(match self {
            ProxySettingKey::ListenPort => {
                let port: u16 = value
                    .parse()
                    .with_context(|| format!("Failed to parse value as listen port: {value}"))?;
                ProxySetting::ListenPort(port)
            }

            // ... snip ...

            ProxySettingKey::Ipv6 => {
                let ipv6: bool = value
                    .parse()
                    .with_context(|| format!("Failed to parse value as boolean: {value}"))?;
                ProxySetting::Ipv6(Some(ipv6))
            }
            ProxySettingKey::Ssl => {
                let ssl: SSLSelection = value
                    .parse()
                    .with_context(|| format!("Failed to parse value as SSLSelection: {value}"))?;
                ProxySetting::Ssl(Some(ssl))
            }
        })
    }
}
```

Problem:

Problem:

`nrpctl` changes system state, how to handle errors?

Solution:

Solution:

Functional programming tricks for transactions with rollback

(revelation #2)

```rust
pub struct Transaction {
    stack: Vec<Box<dyn FnOnce() -> Result<String>>>,
}

impl Transaction {
    /// Create a new transaction.
    pub fn new() -> Transaction {
        Transaction { stack: Vec::new() }
    }

    /// Immediately call the given function. If it succeeds, it should return a callback which will
    /// roll back any changes which the function caused; that callback is added to the transaction.
    /// If it errors, then call the callback for every previous function which succeeded, and
    /// return the original error, chained with any errors which resulted from any failed
    /// rollbacks.
    pub fn do_or_rollback(
        &mut self,
        f: impl FnOnce() -> Result<Box<dyn FnOnce() -> Result<String>>>,
    ) -> Result<()> {
        match f() {
            Ok(callback) => {
                self.stack.push(callback);
                Ok(())
            }
            Err(e) => Err(self.rollback(e)),
        }
    }

    fn rollback(&mut self, cause: Error) -> Error {
        self.stack.drain(..).fold(cause, |err, f| match f() {
            Ok(desc) => {
                println!("{}", desc);
                err
            }
            Err(restore_err) => restore_err.context(err),
        })
    }
}
```

```rust
fn add(
    // ... snip ...
) -> Result<()> {
    // ... snip ...

    let mut config = Config::read(config_path)?;

    // snip config.add(...) ...

    let mut transaction = Transaction::new();

    transaction.do_or_rollback(|| config.write_nginx_site(&listen_domain))?;

    transaction.do_or_rollback(nginx::reload)?;

    let ssl_selection = match ssl {
        Some(sel) => sel,
        None => SSLSelection::False,
    };

    transaction.do_or_rollback(|| certbot::handle_ssl_selection(&listen_domain, ssl_selection))?;

    transaction.do_or_rollback(|| backup_and_write_config(config_path, &config))?;

    println!("Successfully added {listen_domain}");
    Ok(())
}
```

If it compiles, it usually runs

(revelation #3 -- hopefully)

Live Demo

On Prod

(Very Safe)