## Contents

# 1 P2PRC tracking home server

## 1.1 Abstract

This is a library on top of P2PRC to create easier abstractions to setup servers on spare laptops at home and run applications which can exhibit a monolithic behaviour (Ex: Self hosted jitsi, Invidious). The motivation is to ensure users can share self hosted infrastructure with friends and can design a really simple replicatable instances of servers.

## 1.2 Problems being addressed

1. Remembering ports mapped and stateful restart. Rather than the OS remember information of the processes. The P2P network should also track it and propagate it to other nodes in the network. This is to ensure when a process fails there is a formally representable reason which can appropriate corrective actions.

2. Arbitrary instructions used to start a process and should be remembered to reproduce a process if a node is down.

## 1.3 What are attempting to do

The section below describes in detail the ideal strategy of addressing the problems stated above.

### 1.3.1 Building Abstraction

We are proposing to build a library of the Haskell bindings which do the following:

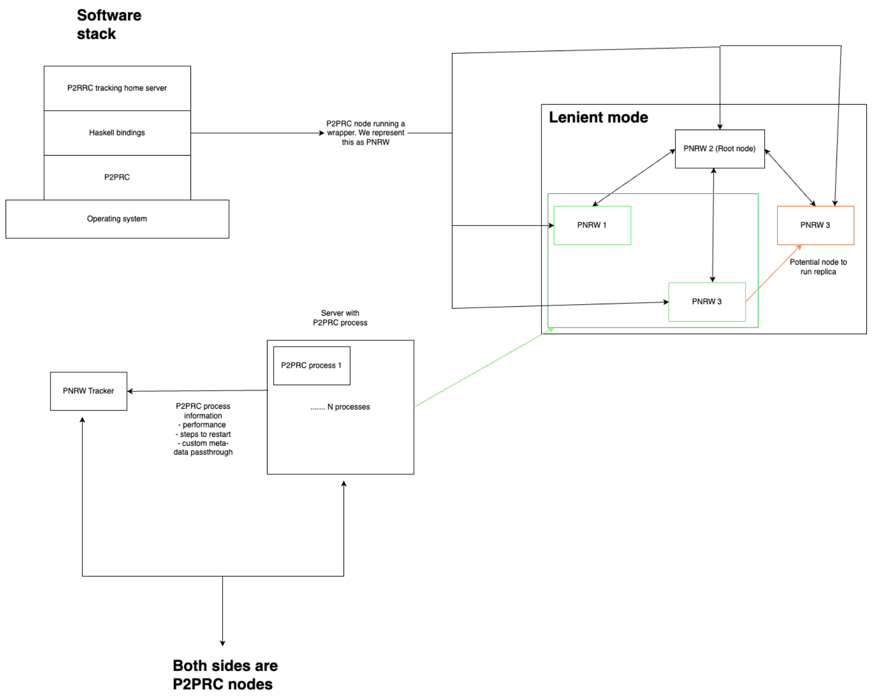
- Creating a P2PRC process and allowing users to modify it's behaviour in functional manner with for instance immutable datatypes.
- Allowing users to run this abstraction in a reproducible manner or constraining it to run on precise sets of machines.
- Should be able to represent the deployment strategy used (ex: Nix, Ansible etc...). This keeps the deployment agnostic and not vendor specific.
- The intended idea of this is to only work with monolithic applications.

### 1.3.2 Building it for only home server users

- The use case is to ensure it's easier to setup a set of servers and shuffle P2PRC processes around when needed. We intend to do this by ensuring a process is represented as type.
- Targets initially with the assumption that there is only 1 reliable root node to relay traffic to the make the setup easier. Future plans will create a DNS layer which can make root nodes and P2PRC processes reproducible.

### 1.3.3 Lenient rules

- This library is built for home users, therefore the library will be setup for P2PRC using the unsafe mode which will ensure that all public keys will be added to the SSH auth list of all nodes available in the network. While this is not secure it comes with pre-assumption that all nodes joining the network are home servers and are needed to access each other with limited rules (set by user permission from the OS side).
- We assume that all P2PRC processes run on a bare-metal machine without any virtualisation. This is because most users run on machines which they own and we offload to the users responsibility to choose a tool that can support reproducible builds like Nix.


Software stack
P2RRC tracking home server
Haskell bindings
P2PRC
Operating system
P2PRC node running a wrapper. We represent this as PNRW
Lenient mode
PNRW 2 (Root node)
PNRW 1
PNRW 3
PNRW 3
Potential node to run replica
Server with P2PRC process
P2PRC process 1
....... N processes
PNRW Tracker
P2PRC process information
- performance
- steps to restart
- custom meta-data passthrough
Both sides are P2PRC nodes

### 1.4 Plan

Based on the Haskell bindings to build an extended library (There are still changes needed in the Haskell bindings before the add-on can be built).

#### 1.4.1 Handle more cases than machine name

The current haskell library seems to only support 1 type Machine name which is parseable from the config file. We would need to handle conditions such as handling types such as Baremetal and UnsafeMode. On this proposed library they would both be set to true.

```
instance FromJSON P2prcConfig where
parseJSON (Object o) = do

  machineName <- o .: "MachineName"

  pure
    $ MkP2prConfig
      { machineName=machineName
    }

parseJSON _ = mzero
```

### 1.4.2 Representation of a process

To design a data structure that can represent current set of properties:

1. IP address (string)
2. Port no (int)
3. Task name (string)
4. Task id (int)
5. Encoded deployment script (multi-line string)
6. Command to run deployment script (multi line string)
7. Command to kill deployment script (multi line string)
8. Status (boolean)
9. Domain name (string)

### 1.4.3 Representation of a node:

Follows the implemented Haskell data structure from the bindings:

```
instance FromJSON ServerInfo where
parseJSON = withObject "ServerInfo" $
  \ o -> do

    name        <- o .: "Name"
    ip4str      <- o .: "IPV4"
    ip6str      <- o .: "IPV6"
    latency     <- o .: "Latency"
    download    <- o .: "Download"
    upload      <- o .: "Upload"
    serverPort  <- o .: "ServerPort"
    bmSshPort   <- o .: "BareMetalSSHPort"
    nat         <- o .: "NAT"
    mEscImpl    <- o .: "EscapeImplementation"
    custInfo    <- o .: "CustomInformation"
```

### 1.4.4 Function expected to be built:

The following refers to the functions we would be building for our abstraction:

1. SpinProcess(<P2PRC process type>,<Node Type>)
2. KillProcess(<P2PRC process type>)
3. ProcessInformation(<process id> or <domain name>) returns <P2PRC process type>
4. ListProcess() return <P2PRC process type>

## 1.5 Terms

P2PRC process: A p2prc process refers to potentially an instance (i.e web-server or any task). This process stores information such as:

- Memory usage.
- Instructions to re-produce that task etc...

Root node: Refers to a node running P2PRC with a public IPV4 address to relay traffic through for nodes behind NAT and can potentially even act a proxy for nodes(Used for domain name mapping) in the P2P network.