Building an Open Native FreeBSD CI system

A work-in-progress report

#about

dch@FreeBSD.org | SkunkWerks.at EuroBSDcon 2024
Dublin | Vienna, Austria

Agenda

- scratching the itch
- the general idea
- how could we foster an Open CI?
- the agent-server protocol
- the per-job configuration
- lua bits
- some C internals
- ** distribute and process multiple concurrent tasks

Scratching The Itch

- FreeBSD has great primitives
  - zfs
  - lua in base
  - pf.conf
  - jails & bhyve
- CI tools have *BSD as after-thoughts
- need extensive scripting to leverage them

The General Idea

" A FreeBSD-native CI for the discerning UNIX user. C for the fiddly bits, more lua than C.

- use all the toys
- from embedded h/w (serial console)
- up to big servers & typical CI
- use what you need, extensible to other OS
- `pkg install && we are ready`

Fostering Open CI

- first, get the interfaces right
- clear & documented interfaces
- pluggable code / modular
- re-implement agent, transport, UI as desired
- forum & realtime chat
- but more than that, community

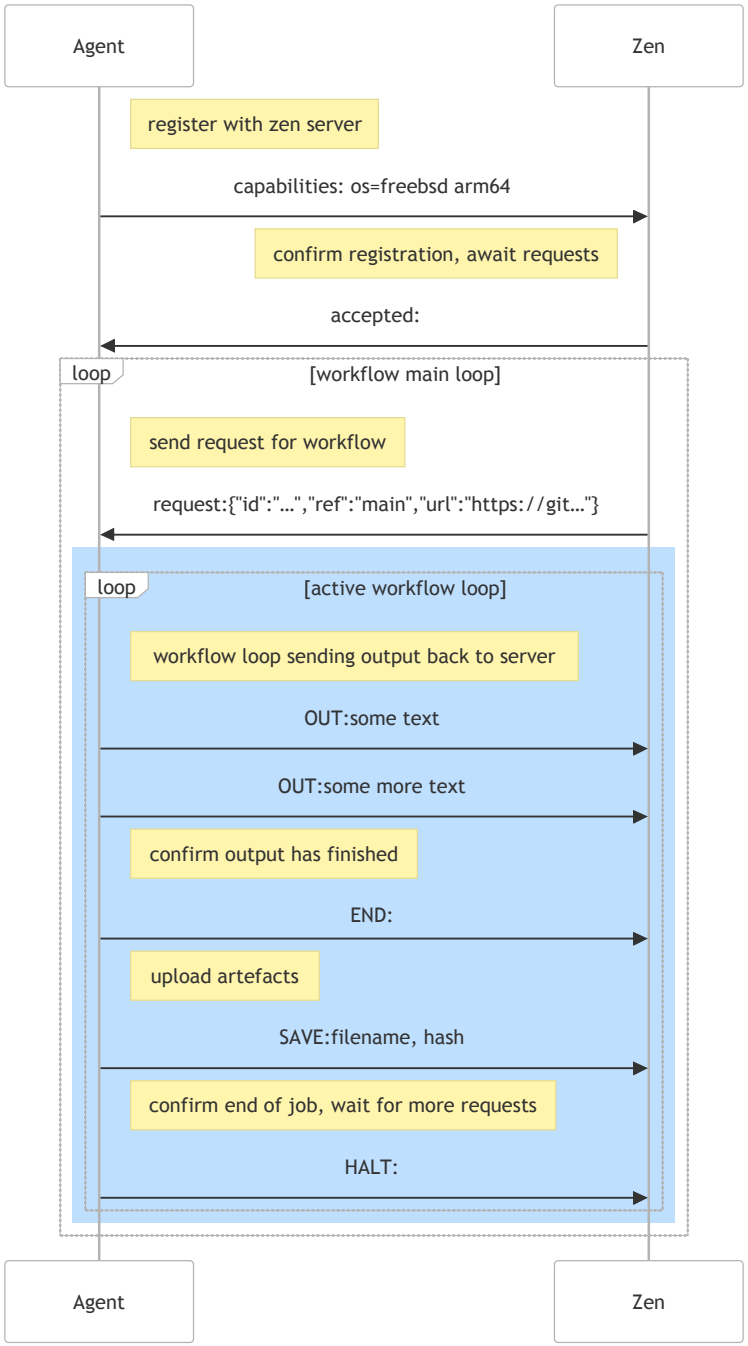
Overview - Agents

- subscribes to wss events from server
- register tags and queue capabilities
- agents execute jobs on request
- need git creds for cloning repos
- fetch pre-encrypted secrets
- streaming logs via wss
- uploads artefacts
- run sandboxed lua scripts at each stage

Overview - Servers

- handles incoming webhooks & direct submissions
- converts UCL config into a DAG of tasks
- ... that can be run across multiple agents
- ... either in series [] , or parallelised {}
- ... just a JSON blob of arrays and objects
- zero knowledge of keys, secrets
- does not touch source code
- stores logs, jobs & artefacts

Protocol - Agent <> Server



Protocol - JSON

```json
{
  "id": "fd34272e569e71a49b6f600b56b756f88c8ba2f0",
  "cache": "9c12faf497b70c2c633f8d8153c10fcfba...",
  "org": "example",
  "ref": "0a994c48004787513bcb32531e14cce516ac27bc",
  "url": "https://forge/example/zen.git"
}
```

- hashes for user-supplied (untrusted) data
- fs paths are opaque, type & length safe
- ideal for zfs snapshots

Protocol - extensible validation

```
type = "object"
required [
    "id",  "cache",  "org",  "ref",  "url"
]
properties {
    id    { type: "string" }
    cache { type: "string" }
    org   { type: "string" }
    ref   { type: "string" }
    url   { type: "string" }
    cmd   { type: "string" }
}
```

- validated by libUCL
- JSON schema (v4 draft)

Agent - ZFS - sources

- bare git repo, or fetched tarball
- `/zen/:org-sha256/:url-sha256/.git`
- zfs snapshotted after each commit fetch
- peek inside bare repo to find the ucl config
- evaluate the config to know what to do

UCL configs (1/2)

```
image = "freebsd/latest";
tags = "OS=freebsd ARCH=arm64 CI=true FLAG_IS_ON"
language = "elixir";
# branches to include/exclude from CI
include [
    "main",
    "(merge|pulls)/.+",
]
exclude [ "wip/.+", ]
 env { lang = "en_US.UTF-8";
       lc_all = "en_US.UTF-8";
}
...
```

UCL configs (2/2)

```
pipeline [
    { setup { tasks = "mix do deps.get, deps.compile";  } }
    { build { env { mix_env = "prod"; }
              tasks = "mix release --env=prod";
              artefacts [ "_build/.+.zip", ] } }
    { deploy { secrets { foo =  "e7d5e36e8d470c3e51...", }]
...
version = 0.1;
```

- pipeline tasks can be [] or {}
- secrets are symmetric encrypted blobs
- server never has key, only agent has it

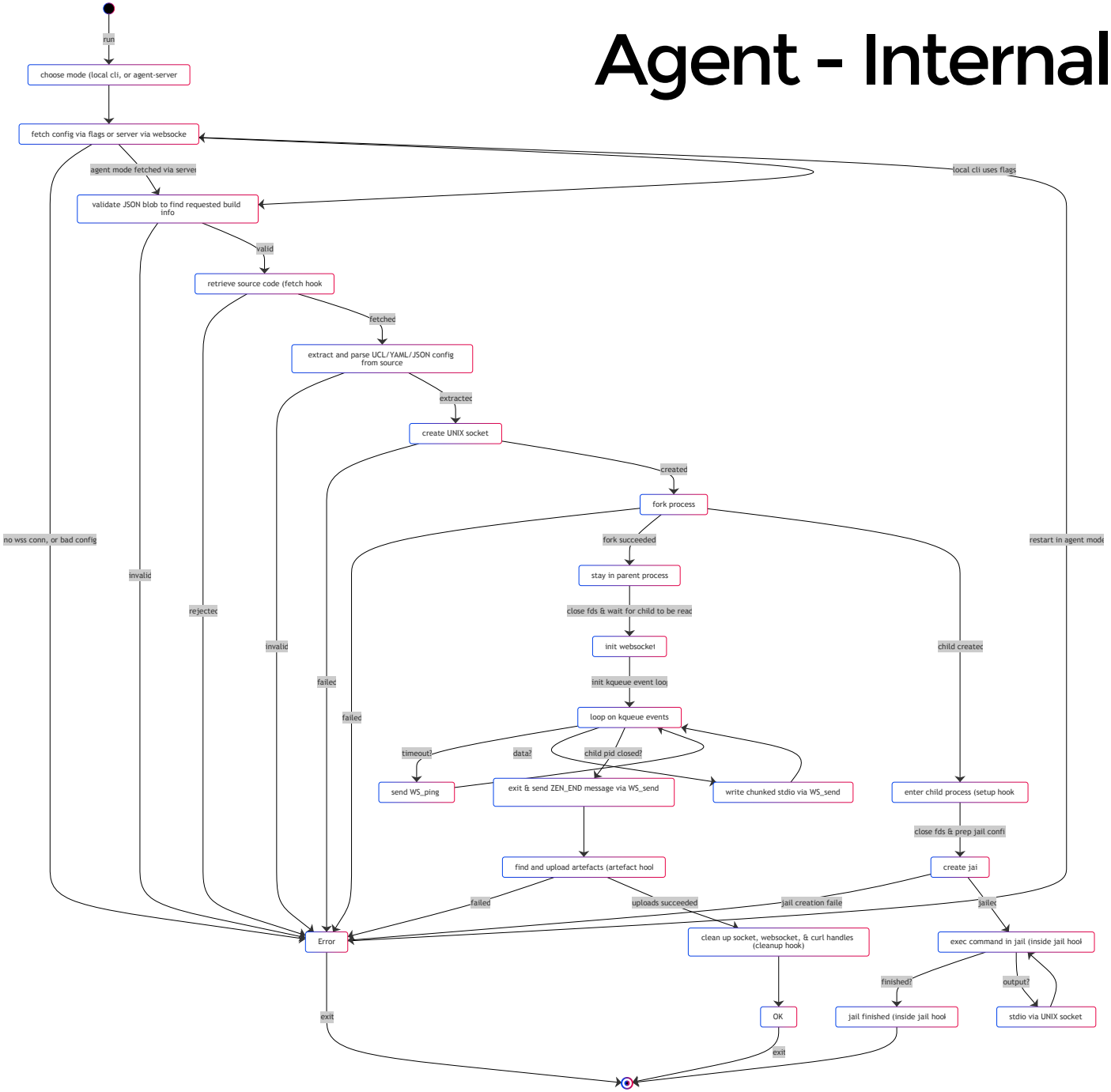
Agent - ZFS - jails

- existing template jail image as zfs dataset
- cloned & mounted for jailing
- mount the sources zfs snapshot as a r/w clone
- make a jail
- exec required commands inside jail
- upload artefacts
- dispose of jail and destroy clones

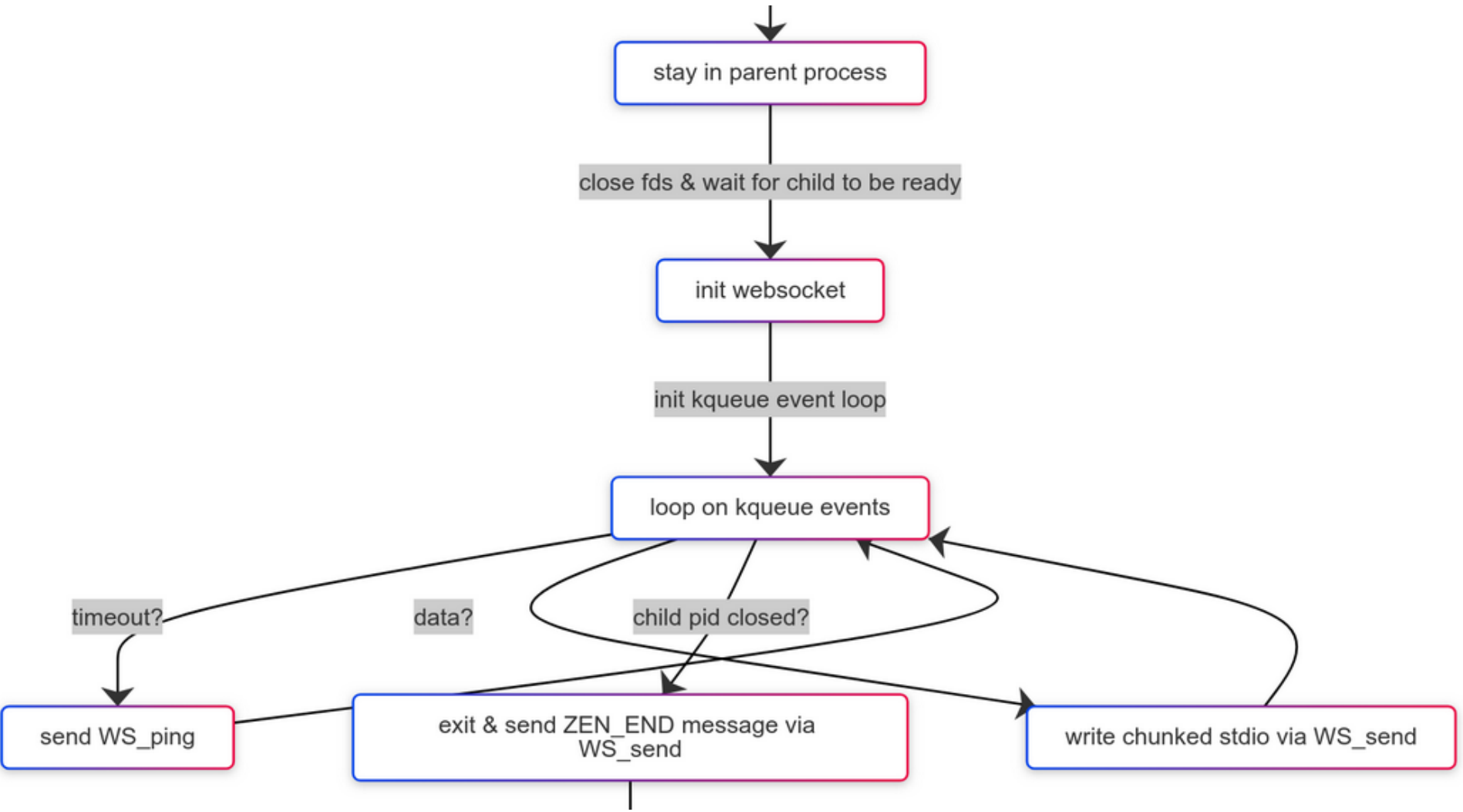
Agent - Dirty C Bits

- open websocket (wss, libcurl)
  - register capabilities
  - wait for jobs
- init lua state & load plugins (lua)
- fetch sources & load git-specific conf (libUCL)
- add unix domain socket, then kqueue & fork!
  - parent grabs stdio to stream it over wss
  - child jails itself and runs whatever
- upload artefacts
- tedious cleanup

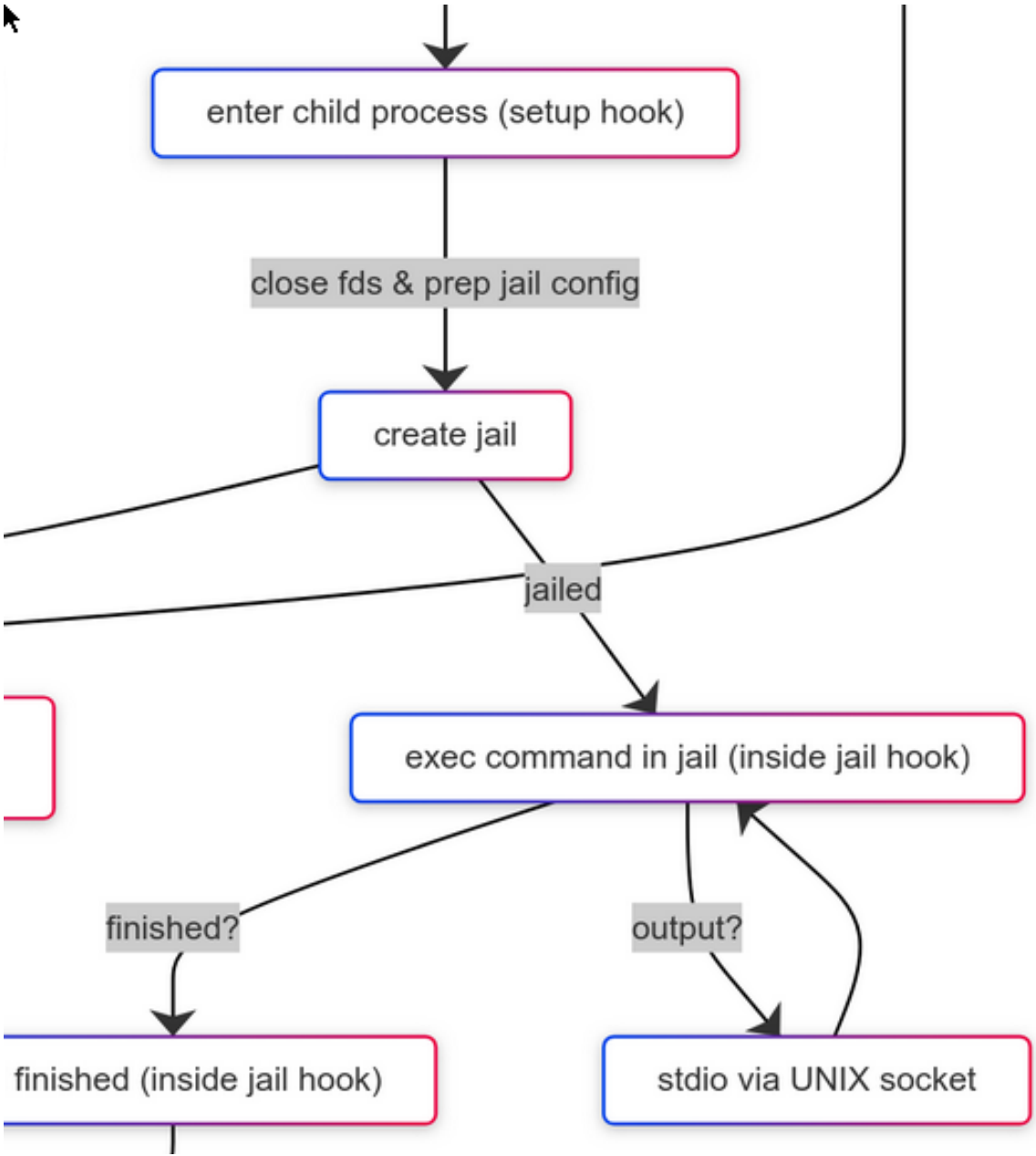
Agent - Internal Flow



Agent - kqueue loop



Agent - jail loop



Implementations

- web server, written in Phoenix / Elixir
- use `websocat` for testing
- 2 clients for spec testing
  - Elixir version (cross platform runtime)
  - C version (native FreeBSD with jail support)
  - keeps us honest

Lua Overview

- lightweight MIT licensed language
- quick to learn the basics
- designed for embedding & C integration
  - no `dir()` function
  - stack based
  - thread & fork safe
- 24k LoC in ANSI C
- 240KiB compiled
- damn fast

Lua in FreeBSD

- boot loader
- flua scripting (FreeBSD-Lua)
  - hashing
  - jails
  - posix
  - ucl config & schema
  - HTTP/1.1 via fetch()
- pkg tool
- zfs channel programs
- nuageinit ...

Adding Lua to base: `libfetch`

- tiny wrapper around `fetch(3)`
- register C functions with lua
- export new functions into a namespace

```
 1 static const struct
 2 luaL_Reg fetchlib[] = {
 3         {"get_url", lfetch_get_url},
 4         {"parse_url", lfetch_parse_url},
 5         {NULL, NULL}
 6 };
 7
 8 int
 9 luaopen_fetch(lua_State *L)
10 {
11         luaL_newlib(L, fetchlib);
12         return (1);
13 }
```

using `fetch` from Lua

```
$ /usr/libexec/flua -l f=fetch

> f.parse_url("http:\\")
nil     Failed to parse URL

> r = f.parse_url("http://user:pass@localhost:1999/"
    .. "rain?beret=raspberry")

> for k,v in pairs(r) do; print('\t',k,v); end
                user        user
                doc         /rain?beret=raspberry
                scheme      http
            ...

> f.get_url("http://w3.org/", "/tmp/w")
true
```

lua plugins

- implement as much as possible in lua
- within jail, or in wrapper
- because no `dir()` we load plugins via C
- but do init in lua
- allows future sandboxing of plugin init
- everything takes & returns a lua state (stack)
- this state is available in both forks
- plugins do zfs manipulation via delegation
- try to use existing pf rulesets

Plugins - implementation

- mostly same as other lua code

```
lua_State *
plugins_init(lua_State *L, const char *dir)
{

  char *init_lua;
  if (asprintf(&init_lua, "%s/init.lua", dir) == -1) {
    log_fatal("plugins: "
        .. "failed to allocate memory for init.lua path");
    lua_close(L);
    exit(EXIT_FAILURE);
  }
```

Plugins - implementation

- load & exec init.lua file from plugins dir

```
...
  if (luaL_dofile(L, init_lua) != LUA_OK) {
    free(init_lua);
    log_error("plugins: error loading init.lua: %s",
        lua_tostring(L, -1));
    lua_close(L);
    exit(EXIT_FAILURE);
  }

  free(init_lua);
  return L;
}
```

- plugins are loaded from the host filesystem
- but can execute *inside* the jail

Plugins - registration & notification

- `init()` returns a table of "events"
- we use this as an event bus
- works in both forks but not across them

```lua
-- given an event name, a module name, and a function,
-- insert it into the subscriptions table
local function subscribe(event, module, func)
  -- check if the event exists already in the subscriptions table
  if not subscriptions[event] then
    subscriptions[event] = {}
  end
  subscriptions[event][module] = func
  print("      subscribed to " .. event)
end
```

Plugins - hello_world

```lua
hello = {}

local function hello.world()
    log.debug("lua: hello world")
end

local function hello.registered()
    log.debug("lua: hello registered")
end

local function hello.init()
    zen.subscribe({ "world", "registered", })
end

return hello
```

Plugins calling plugins - Oh My!

- event bus in action!

```lua
fs = {}

function fs.hello()
    log.debug("lua: hello from fs")
end

function fs.mount() ... end

function fs.unmount() ...  end

function fs.init()
    zen.subscribe({
        "hello",
        "mount",
        "unmount",
    })
end

return fs
```

UI & UX

- not quite at `pkg install && ready to go` yet
- the CLI tool & the web tool are in flux
- the UNIX side is more interesting for us here
- but the web side is very pretty

UI - cli websocket view

- for testing, run `websocat` in 1 terminal
- and agent in another
- fake a job with a JSON blob

```
$ websocat --text -s [::1]:4000
> register:ZEN
    ZEN_AGENT=4c4c4544-0036-4410-8032-b6c
    ZEN_ARCH=amd64
    ZEN_HOSTNAME=akai.skunkwerks.at
    ZEN_OSNAME=FreeBSD
    ZEN_OSRELEASE=15.0-CURRENT
< request:{"id":"ec648131-4888-4bf2-...",
  "cache":"abc123",
  "ref":"main",
  "url":"https://github.com/example/simpl
  "org":"demo"}
> OUT: hello world
...
> SAVE: b76db083efa308456766...,world.txt
> HALT:
```

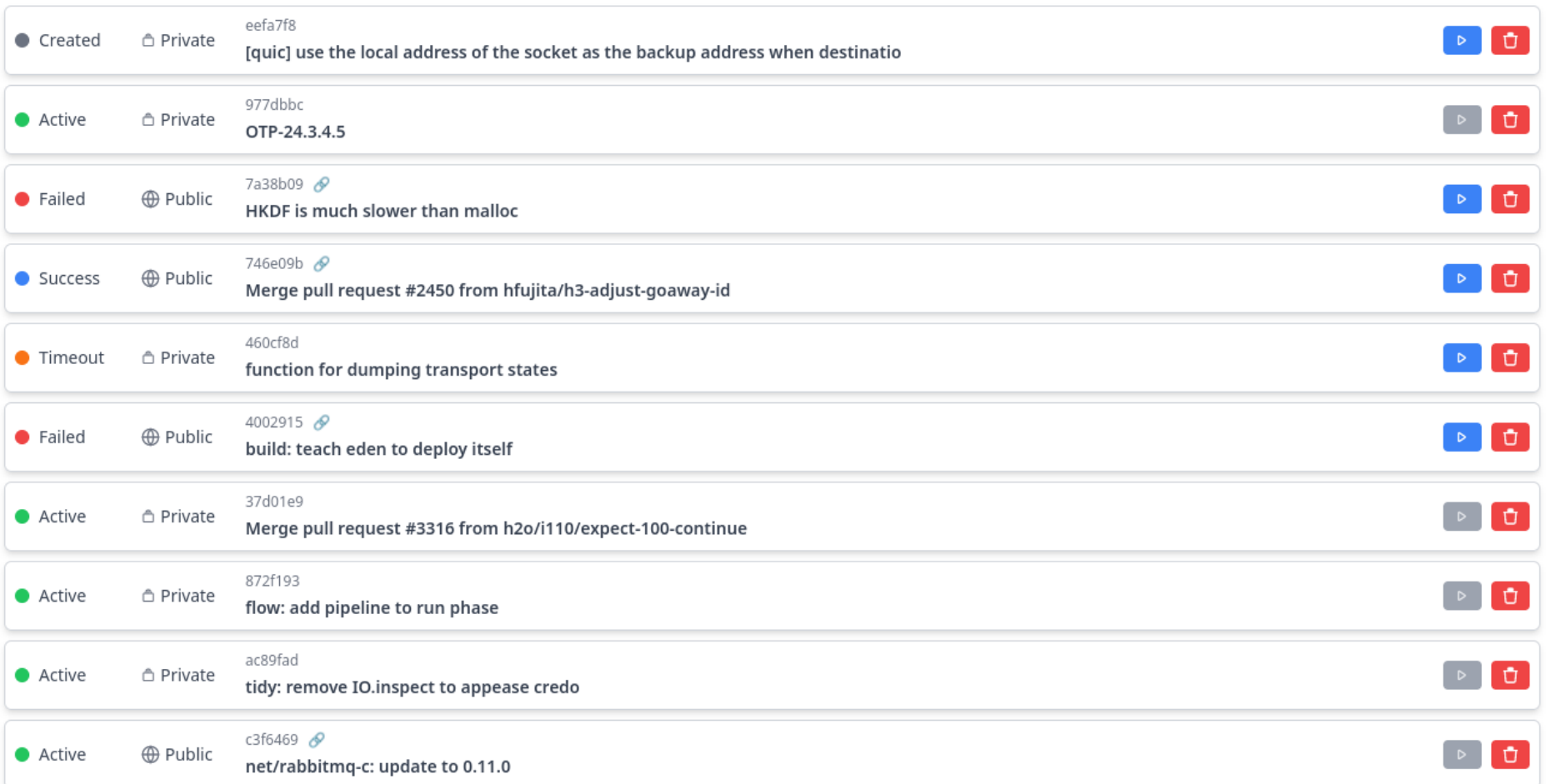
UI - cli agent

```
doas env ZEN_TOKEN=0badcafe  ./bin/zen  -u ws://localhost:4000/agent/demo  -d /tmp/zen  -p plugins  -v
10:33:17 DEBUG src/options.c:20: config:
10:33:17 DEBUG src/options.c:21:   agent:   0decaf
10:33:17 DEBUG src/options.c:22:   ws url:  ws://localhost:4000/agent/demo
10:33:17 DEBUG src/options.c:23:   dir:     /tmp/zen
10:33:17 DEBUG src/options.c:24:   tmp:     /tmp/zen.0decaf.84PE3J
10:33:17 DEBUG src/options.c:25:   cmd:     /sbin/ping -c 3 localhost
10:33:17 DEBUG src/options.c:26:   plugins: plugins
10:33:17 DEBUG src/options.c:27:   tags:    ZEN ZEN_AGENT=4c4c4544-0036-4410-8032-b6c04f334732 ZEN_ARCH=amd64
ZEN_HOSTNAME=akai.skunkwerks.at ZEN_OSNAME=FreeBSD ZEN_OSRELEASE=15.0-CURRENT
10:33:17 TRACE src/zen.c:66: zen: initialised lua std library
10:33:17 TRACE src/zen.c:70: zen: loaded log module
10:33:17 TRACE src/zen.c:82: zen: populated env from tags
10:33:17 DEBUG src/log.c:198: lua: init started in state started
10:33:17 TRACE src/log.c:190: lua: dump zen:
{
    config = {
        plugins_list = {},
        plugins_path = "plugins",
        url = "ws://localhost:4000/agent/demo",
        verbose = 1
    },
    env = {
        ZEN = "1",
        ZEN_AGENT = "4c4c4544-0036-4410-8032-b6c04f334732",
        ZEN_ARCH = "amd64",
        ZEN_HOSTNAME = "akai.skunkwerks.at",
        ZEN_OSNAME = "FreeBSD",
        ZEN_OSRELEASE = "15.0-CURRENT"
    },
    state = "function: 0x207560"
}
10:33:17 TRACE src/log.c:190: lua: env:
10:33:17 TRACE src/log.c:190:   ZEN:    1
10:33:17 TRACE src/log.c:190:   ZEN_AGENT:      4c4c4544-0036-4410-8032-b6c04f334732
10:33:17 TRACE src/log.c:190:   ZEN_ARCH:       amd64
10:33:17 TRACE src/log.c:190:   ZEN_HOSTNAME:   akai.skunkwerks.at
10:33:17 TRACE src/log.c:190:   ZEN_OSNAME:     FreeBSD
10:33:17 TRACE src/log.c:190:   ZEN_OSRELEASE:  15.0-CURRENT
10:33:17 TRACE src/log.c:190: lua: subscriptions:
10:33:17 INFO  src/log.c:206: lua: publishing event hello
```

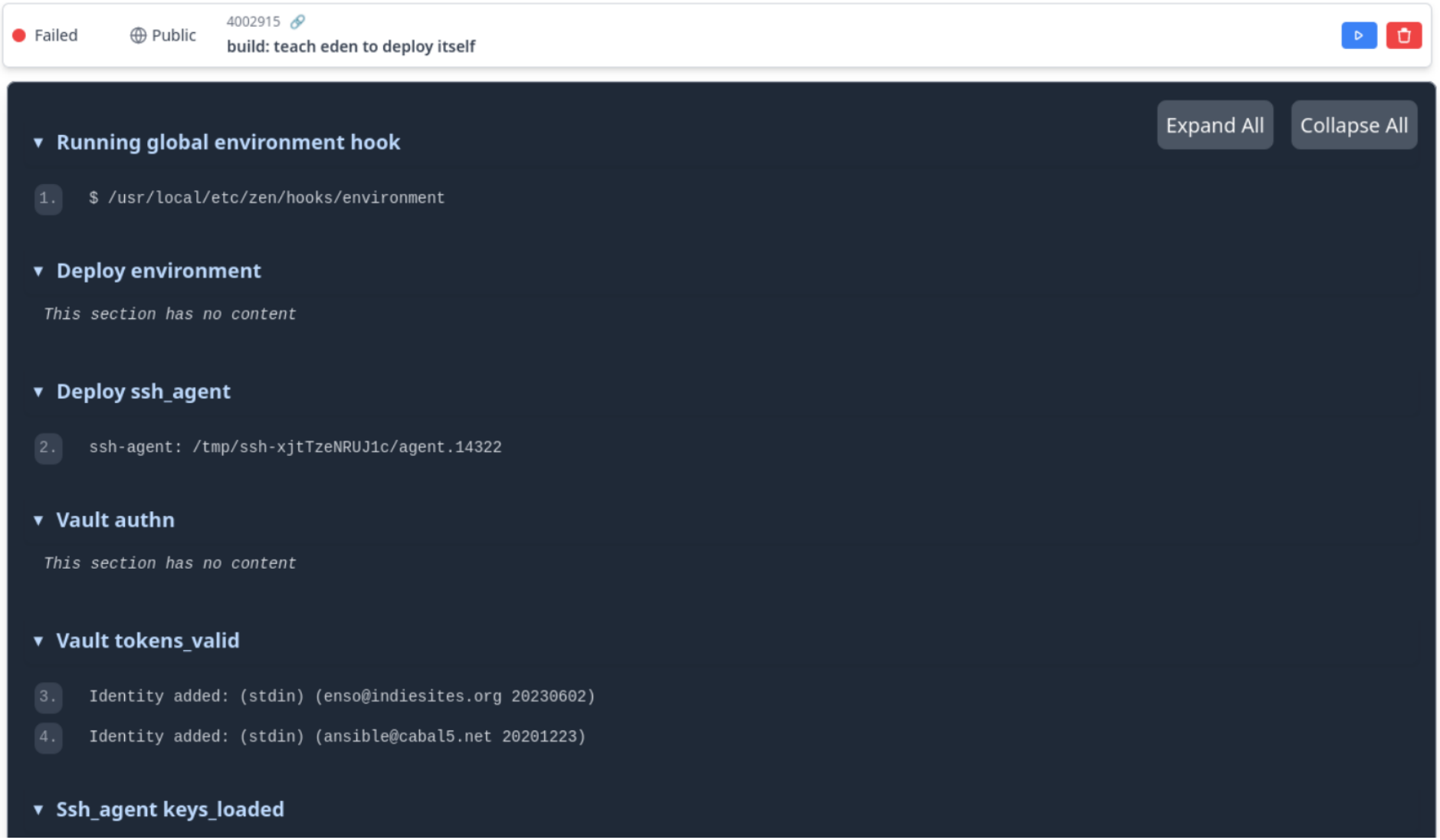
UI - websocket messages

```
23:01:41 TRACE zen.c:617: zen: send message: OUT: PING localhost (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.027 ms

* WS-ENC: sending [TEXT payload=0/104]
* WS-ENC: buffered [TEXT payload=104/104]
* WS: flushed 110 bytes
23:01:42 TRACE zen.c:607: zen: send liveness ping
* WS-ENC: sending [PING payload=0/4]
* WS-ENC: buffered [PING payload=4/4]
* WS: flushed 10 bytes
23:01:42 TRACE zen.c:617: zen: send message: OUT: 64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.095 ms
...
* WS-ENC: sending [TEXT payload=0/63]
* WS-ENC: buffered [TEXT payload=63/63]
* WS: flushed 69 bytes
23:01:46 TRACE zen.c:617: zen: send message: OUT: --- localhost ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.027/0.083/0.114/0.032 ms

* WS-ENC: sending [TEXT payload=0/158]
* WS-ENC: buffered [TEXT payload=158/158]
* WS: flushed 166 bytes
* WS-ENC: sending [TEXT payload=0/4]
* WS-ENC: buffered [TEXT payload=4/4]
* WS: flushed 10 bytes
* WS-ENC: sending [TEXT payload=0/25]
* WS-ENC: buffered [TEXT payload=25/25]
* WS: flushed 31 bytes
* WS-ENC: sending [TEXT payload=0/25]
* WS-ENC: buffered [TEXT payload=25/25]
* WS: flushed 31 bytes
* WS-ENC: sending [TEXT payload=0/25]
* WS-ENC: buffered [TEXT payload=25/25]
* WS: flushed 31 bytes
* WS-ENC: sending [TEXT payload=0/4]
* WS-ENC: buffered [TEXT payload=4/4]
* WS: flushed 10 bytes
* WS-ENC: sending [TEXT payload=0/11]
* WS-ENC: buffered [TEXT payload=11/11]
* WS: flushed 17 bytes
23:01:46 TRACE zen.c:654: zen: parent cleaning up
* WS-ENC: sending [CLOSE payload=0/0]
* WS-ENC: buffered [CLOSE payload=0/0]
* WS: flushed 6 bytes
23:01:46 INFO  zen.c:661: zen: parent cleaned up
```

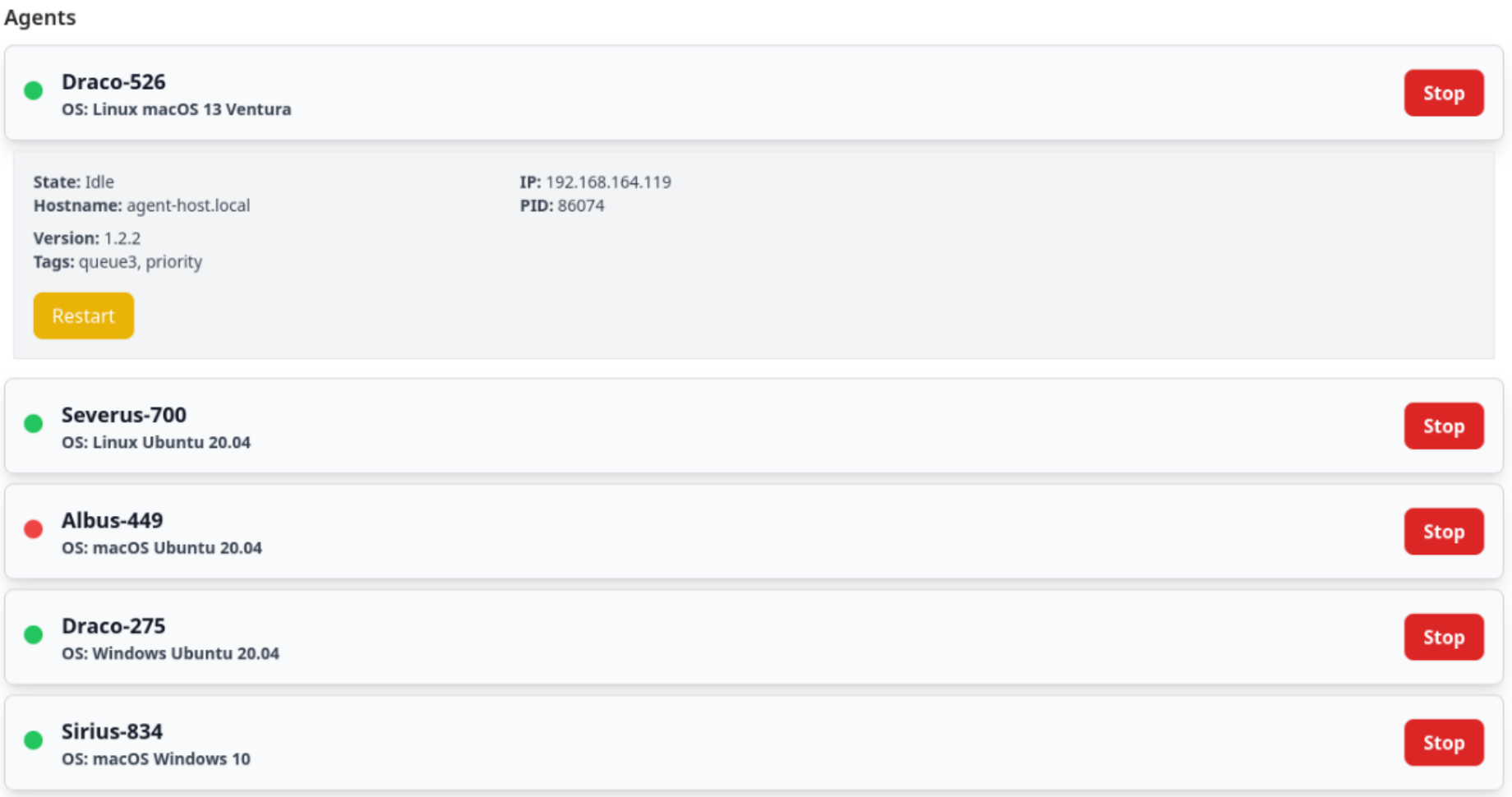
UI - jobs - active & historic



UI - detail


Failed
Public
4002915
build: teach eden to deploy itself
Expand All
Collapse All
Running global environment hook
1. $ /usr/local/etc/zen/hooks/environment
Deploy environment
This section has no content
Deploy ssh_agent
2. ssh-agent: /tmp/ssh-xjtTzeNRUJ1c/agent.14322
Vault authn
This section has no content
Vault tokens_valid
3. Identity added: (stdin) (enso@indiesites.org 20230602)
4. Identity added: (stdin) (ansible@cabal5.net 20201223)
Ssh_agent keys_loaded

UI - agents


Agents
Draco-526
OS: Linux macOS 13 Ventura
Stop
State: Idle
Hostname: agent-host.local
IP: 192.168.164.119
PID: 86074
Version: 1.2.2
Tags: queue3, priority
Restart
Severus-700
OS: Linux Ubuntu 20.04
Stop
Albus-449
OS: macOS Ubuntu 20.04
Stop
Draco-275
OS: Windows Ubuntu 20.04
Stop
Sirius-834
OS: macOS Windows 10
Stop

Thanks

- kevans@ markj@ my lua heroes
- kfv for C guidance
- FreeBSD jails WG for guidance and moral support
- @maikel for amazing looking web ui
- @jpenard for web and elixir client

- libcURL @bagder & curl community wss
- libmonocypher: Loup Vaillant & crypto advice
- log.c: @rxi for pretty logs
- libUCL: @vstakhov for config & schema
- FreeBSD, Elixir & Phoenix community

Closing

- libUCL for data format & validation
- kqueue for i/o capture of jails
- lua for easy extension
- libcURL for websocket communications
- a cross-platform dream
- open source once we are self-hosting
- both BSD and other operating systems

Help Wanted - email me

- actual use cases
- contributors and hackers
- libUCL and modern JSON schema compatibility
- more lua-accessible APIs - libpf? libzfs?