Who supervises supervisors?

Introduction to using systemd with Erlang

@hauleth 2020

Who am I?

- Hauleth - https://hauleth.dev
- Erlang/Elixir Developer @ Kobil GmbH
- EEF Observability WG Member
- Contributor to OpenTelemetry
- Author of `systemd` Erlang library (and few other libraries)

Some knowledge needed

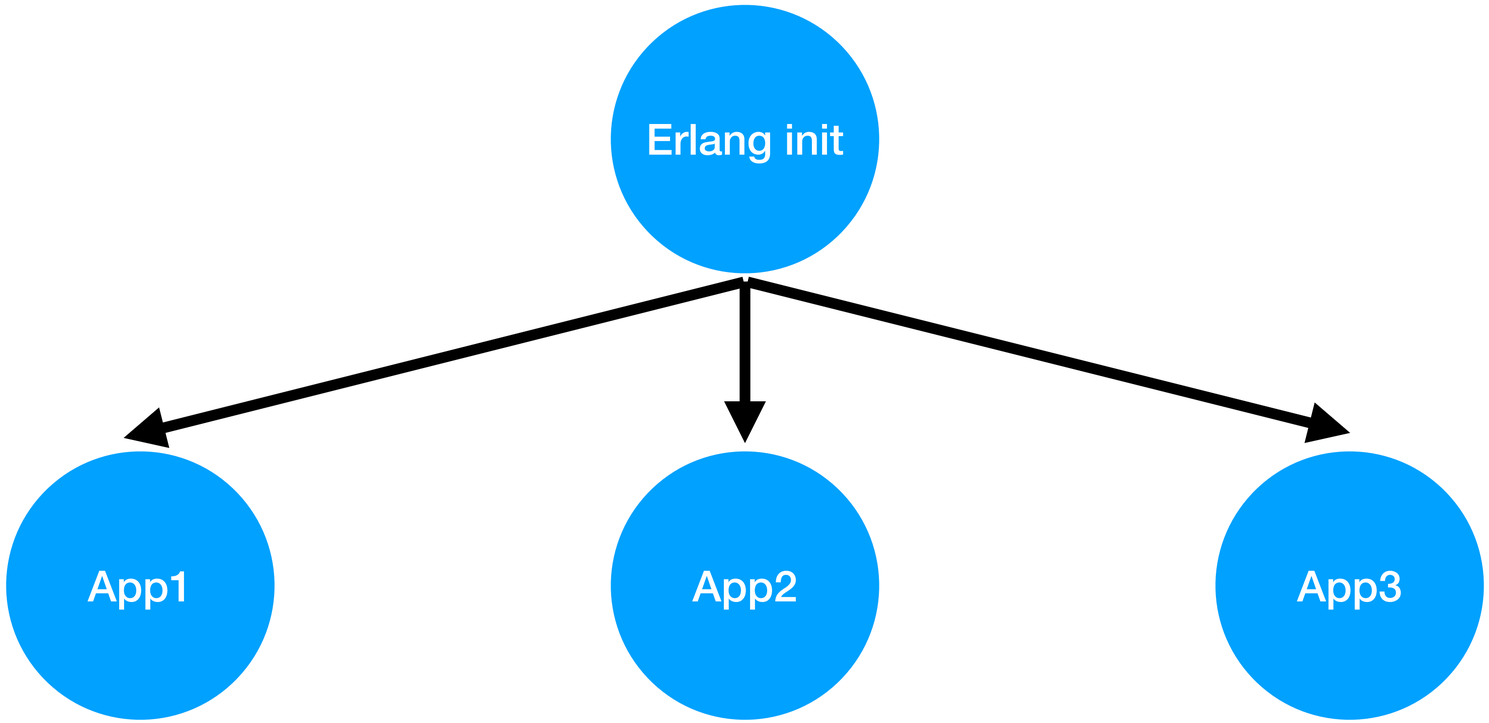
I will not explain everything, there is no time for that

systemd is ok

but it has flaws

- What is init system
- How it works on the high level
- What systemd features are interesting for us
- How we can use these features

?


Erlang init
App1
App2
App3

*nix init system

How your system starts

- First process ran by your OS
- Run with PID 1
- When it exits, everything goes down, either by shutdown or errors out
- Responsible for everything else
- Responsible for reaping zombies (in most cases)
- In systemd it is the system supervisor (not all inits does that)

Similarities between system init and Erlang init

- When it dies, (virtual) machine dies
- It is responsible for starting everything else
- It reads job configuration and then starts required jobs
- Listen on messages when to stop the (virtual) machine

Why bother?

"Top-level" process management

- Ordering process startup
- Restarting process when it died
- Startup and shutdown hooks
- Process isolation and hardening
- Resource limits
- System state monitoring

Centralised log management

journald

- All systems send logs to single place
- No need for manual management of log rotation
- Keeping metadata together with logs
- Built-in log dispatching tooling
- Supports both stdout and direct logging via socket
- Logs stored in binary format

How to use it?

Version simple

```
defp deps do
  [
    # …
    {:systemd, "~> 0.0"}
    # …
  ]
end
```

How to use it?

Version hard

```
defp deps do
  [
    # …
    {:systemd, "~> 0.0"}
    # …
  ]
end
```

```
defmodule MyApp.Application do
  use Application

  def start(_, _) do
    :logger.add_handlers(:systemd)

    # …
  end
end
```

Notifications and watchdog

- Informing administrator about state of the process
- Health checks whether the system is still alive
- Triggering restarts from within the application
- Storing opened file descriptors (like sockets) between application starts

How to use it?

```
[Service]
# …
Type=notify
ExecStart=/path/to/rel/bin start
```

```
defmodule MyApp.Application do
  use Application

  def start(_, _) do
    children = [
      # …
      :systemd.ready()
    ]

    # …

    Supervisor.start_link(
      children,
      opts
    )
  end
end
```

Lazy startup
socket activation

- Start application just when the request arrives
- Keep sockets open as soon as system starts
- Keep sockets open between application restarts
- Use sockets from the privileged scope without superuser

How to use it?

Intentionally left blank

Questions?