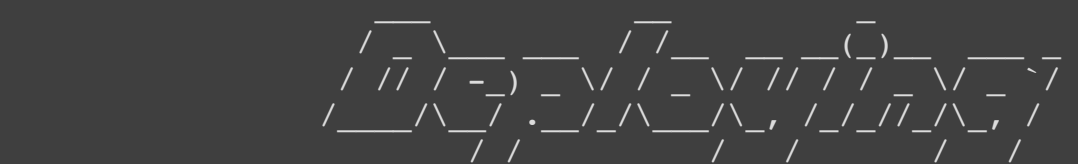

to Production in 2016

NodeConf Oslo
June 5th 2016

Luke Bond
@lukeb0nd

WHO AM I?

- I'm a backend developer, DevOps-curious
- Mostly I do Node.js and Docker
- Built an OS project called "Paz" - _*http://paz.sh*_

I work for YLD.io, a London-based software engineering consultancy
that specialises in Node.js, Docker and React.

Mostly we help enterprise companies move towards continuous delivery
and embrace DevOps practices.

AMA.

WHAT'S THIS TALK ABOUT?

This talk in a nutshell:

> Use your Linux init system to run your Node.js apps
> It's easy, powerful and the tooling is great!

- I'll talk about PM2 as a reference case
- Then I'll show you hands-on how to achieve the same things with systemd
- This talk will be mostly demo
- I will move quite quickly (sorry) but the final result will be available to copy-paste!

All unit files in this talk can be found here, along with the slides:

https://github.com/lukebond/nodeconf-oslo-20160604

PM2

- I'll be referring a lot to PM2 because everyone knows it
- Most of you probably use it in production; or something like forever, mon or nodemon
- _*Please note*_ that I have nothing against PM2 or any of these tools!

Why is PM2 so popular? Because it makes the following very easy:

1. Process management
2. Log management
3. Magic/seamless sharing of ports

PM2 has great UX too. It's a powerful tool.

LEARN TO LINUX

Why learn Linux instead of sticking to PM2 or similar?

- You can learn to do all these things yourself, using basic Linux tooling
- It's easy and it's fun
- Broaden your skill-set!
- Impress your friends!
- Learn that you don't need a process monitor
- Deploy applications that any Linux sysadmin outside the Node.js world will understand
- systemd is now more or less the standard init system

LINUX INIT SYSTEMS

- Linux has something called an "init system" that runs as PID1
- It's the ancestor of all processes on Linux; the ultimate process monitor!
- Each service gets an init script for start|stop|restart etc.
  - e.g. Databases, web servers, etc.
- Basically what PM2 does, but OS-wide
- Linux has been doing this for years
- Most modern distros use systemd as the init system

SAMPLE APP

- I've built a contrived sample app in Node.js that talks to Redis:

> https://github.com/lukebond/demo-api-redis

- It's basically HTTP Hello World with a Redis counter
- We'll set it all up with systemd
- You will need:
  - A version of Linux with systemd *
  - Node installed
  - Redis installed

* These distros: https://en.wikipedia.org/wiki/Systemd#Adoption_*and*_reception

MY FIRST UNIT FILE

- We tell systemd about out services by writing unit files
- Let's write our first unit file for our Node.js sample app

```
$ cat /etc/systemd/system/demo-api-redis@.service
[Unit]
Description=HTTP Hello World
After=network.target

[Service]
User=luke
Environment=REDIS_HOST=localhost
WorkingDirectory=/home/luke/Development/demo-api-redis
ExecStart=/usr/bin/node index.js

[Install]
WantedBy=multi-user.target
```

- Create this file and copy it into the above directory (grab it from my GitHub repo!)
- Signal systemd to reload the config
- Enable and start the service *

```
$ systemctl daemon-reload
$ systemctl enable demo-api-redis@1
$ systemctl start demo-api-redis@1
```

- Of course it fails because Redis isn't running!
- Let's explore dependencies with systemd...

* Learn more about `systemctl` here:
  https://www.digitalocean.com/community/tutorials/how-to-use-systemctl-to-manage-systemd-serv

SYSTEMD DEPENDENCIES - Wants=

- Use `Wants=` in `[Unit]` section of unit files to declare dependencies
- Starting this unit will trigger wanted units to be started also

```
$ cat /etc/systemd/system/demo-api-redis@.service
[Unit]
Description=HTTP Hello World
After=network.target
Wants=redis.service

[Service]
User=luke
Environment=REDIS_HOST=localhost
WorkingDirectory=/home/luke/Development/demo-api-redis
ExecStart=/usr/bin/node index.js

[Install]
WantedBy=multi-user.target

$ systemctl daemon-reload
$ systemctl restart demo-api-redis@1
```

- Note that now Redis gets started too!

HANDLING CRASHES, RESTARTS, ETC. (1/3)

- Let's kill the node process and see what happens:

```
$ kill -9 $(pgrep "node index.js")
$ systemctl status demo-api-redis@1 | grep Active
Active: failed (Result: signal) since Thu 2016-06-02 11:50:32 BST; 47s ago
```

- The process hasn't been automatically restarted after the "crash"
- Add the following to the `[Service]` section of the unit file to fix this:

```
Restart=always
RestartSec=500ms
StartLimitInterval=0
```

- This example will restart the service indefinitely with 500ms delay
- There is great flexibility in how this can be configured!
- The above should be fine though

```
$ kill -9 $(pgrep "node index.js")
$ systemctl status demo-api-redis@1 | grep Active
Active: active (running) since Thu 2016-06-02 12:12:05 BST; 22s ago
```

- It has been restarted!
- What about reboots? systemd will start units on boot that are *enabled*

```
$ systemctl status demo-api-redis@1 | grep Loaded
```

- I'm going to risk a reboot; cross your fingers for me!

LOGS (2/3)

- systemd has a powerful tool for working with logs for all services: `journalctl`
- To scroll through logs for a unit or service:

```
$ journalctl -u demo-api-redis@1
```

- To follow said logs:

```
$ journalctl -u demo-api-redis@1 -f
```

- You can ask for logs since the last boot:

```
$ journalctl -u demo-api-redis@1 --boot
```

- You can ask for logs since a certain time:

```
$ journalctl -u demo-api-redis@1 --since 08:00
$ journalctl -u demo-api-redis@1 --since today
$ journalctl -u demo-api-redis@1 --since yesterday
$ journalctl -u demo-api-redis@1 --since 2016-06-02 15:36:00
```

- You can filter by log level (console.log, console.error, etc.):

```
$ journalctl -u demo-api-redis@1 -p err
```

- There is so much more you can do; it's _*super*_ powerful. Great docs here:
  https://www.digitalocean.com/community/tutorials/how-to-use-journalctl-to-view-and-manipulat

MULTIPLE INSTANCES

- First, let's modify the unit file to set different ports for them

```
# /etc/systemd/system/demo-api-redis@.service
[Unit]
Description=HTTP Hello World
After=network.target
Requires=redis.service

[Service]
Environment=REDIS_HOST=localhost
Environment=LISTEN_PORT=900%i
WorkingDirectory=/home/luke/Development/demo-api-redis
ExecStart=/usr/bin/node index.js
Restart=always
RestartSec=500ms
StartLimitInterval=0

[Install]
WantedBy=multi-user.target
```

- And now reload the unit and start and enable the other instances:

```
$ systemctl daemon-reload
$ systemctl enable demo-api-redis@{2,3}
$ systemctl start demo-api-redis@{2,3}
$ netstat -tlpn | grep 900
tcp6     0     0 :::9001      :::*       LISTEN      2654/node
tcp6     0     0 :::9002      :::*       LISTEN      2656/node
tcp6     0     0 :::9003      :::*       LISTEN      2704/node
```

- Cool. There is one final feature: a local load balancing proxy...

SIMPLE LOAD BALANCING WITH balance (3/3)

- `balance` is a simple, light-weight load balancer

  https://www.inlab.de/balance.html

- We can set it up with a one-liner:

```
$ balance -f 9000 127.0.0.1:900{1,2,3}
$ curl localhost:9000
"Hello, world 192.168.1.39! 20 hits."
```

- But let's do this the systemd way, with the following unit file

SIMPLE LOAD BALANCING WITH balance

```
# /etc/systemd/system/balance.service
[Unit]
Description=Balance - Simple TCP Load Balancer
After=syslog.target network.target nss-lookup.target

[Service]
ExecStart=/usr/bin/balance -f 9000 127.0.0.1:9001 127.0.0.1:9002 127.0.0.1:9003

[Install]
WantedBy=multi-user.target
```

- As usual, signal systemd to reload then enable and start the service

```
$ systemctl daemon-reload
$ systemctl enable balance
$ systemctl start balance
```

- Does it work?

```
$ curl localhost:9000
"Hello, world 172.20.10.2! 29 hits."
```

WHERE TO FROM HERE?

- This is just the basics of systemd
  - Despite approaching feature-parity with PM2
- It should be easy to build something dynamic on top of this
  - As opposed to hardcoded ports in `balance.service`
- SSL termation, hooking up to external load balancers, etc. I'll leave to you
- Containers!
  - Normally I'd do all this with containers
    - Using *rkt* or *runc* because Docker & systemd sometimes don't play nicely together
  - I left it out today to reduce the number of new things introduced
  - Talk to me about containers, Node.js & systemd if you're interested!

CONCLUSION

- Learn to use systemd for your Linux production machines
- Use my unit files as a starting point
- systemd has a learning curve but it isn't difficult
- The tools are mature and powerful
- You'll realise that you don't need a process monitor
  - What starts your process monitor, after all?

LINKS

- Repository with slides, unit files etc. here:
  https://github.com/lukebond/nodeconf-oslo-20160604

FURTHER READING

- systemd distros:
  https://en.wikipedia.org/wiki/Systemd#Adoption_*and*_reception
- Good article on using `systemctl`:
  https://www.digitalocean.com/community/tutorials/how-to-use-systemctl-to-manage-systemd-serv
- Good article on using `journalctl`:
  https://www.digitalocean.com/community/tutorials/how-to-use-journalctl-to-view-and-manipulat
- The creator of systemd talking about security features:
  https://www.youtube.com/watch?v=hiW8eIdcRgo&list=PLlh6TqkU8kg_3FpXLlHMnoVqKZysIzXlK&index=6
- Videos from systemd conf 2015:
  https://www.youtube.com/channel/UCvq_RgZp3kljp9X8Io9Z1DA
- systemd man pages:
  https://www.freedesktop.org/software/systemd/man/systemd.unit.html
  https://www.freedesktop.org/software/systemd/man/systemd.service.html
- Slides presented with **mdp**:
  https://github.com/visit1985/mdp

THANKS!

Thanks for listening! Go and read the repo and play.

Any questions, contact me:

@lukeb0nd

luke@yld.io

Or come and say hi today!