🔥 DevOps Deployment Patterns 🔥

👩 SODA Social London, September 2016 👨

Luke Bond

👋 `T: @lukeb0nd | E: luke.n.bond@gmail.com | G: @lukebond`

What You Will Learn From This Talk 📚

- Common application architectures & suitable deployment patterns therefore
- Process monitors (spoiler: use Linux)
- Scaling and service discovery considerations
- A little bit about containers along the way

The aspect of "deployment" I'm focused on is the process; what runs it, how it it looked after, etc.

Application Deployment Patterns 🚀

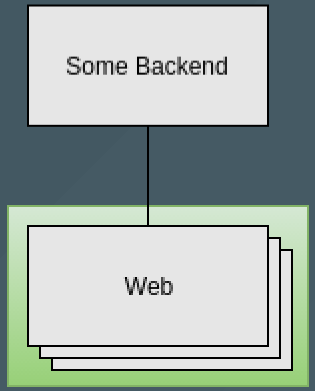
For simplicity, let's consider the following common application types:

- Simple web-app
- Simple web-app + API + websockets + cache + database
- Microservice-based distributed system with a message broker

Let's explore deployment patterns for these types of apps.

TL;DR: don't overcomplicate; you probably don't need Kubernetes.

Simple Web-App



Simple Web-App

- Easily horizontally scalable
- No communication between components or across hosts
  - Hence no service discovery needed
- For single-threaded runtimes like Node.js, run process for each core
- Scale boxes horizontally; every box is the same
- Load balance (e.g. *nginx*, *HAProxy*, *balance*)

Recommendation: use **systemd** (or your preferred process monitor if you must!).

`<Aside>`

Process Monitors

There are various popular ones in use today:

- mon, nodemon, monit, forever
- PM2
- unicorn, gunicorn
- daemontools
- runit
- supervisord

These range from simple process monitors to almost full init systems.

Use systemd as your Process Monitor

- IMHO the init system is the ultimate process monitor
  - But maybe you don't have root access to your Linux hosts
  - I wonder if this is why process monitoring tools have become popular
  - `<philosphising>`
    - A half-way house on the journey to DevOps?
  - `</philosphising>`

Use systemd as your Process Monitor

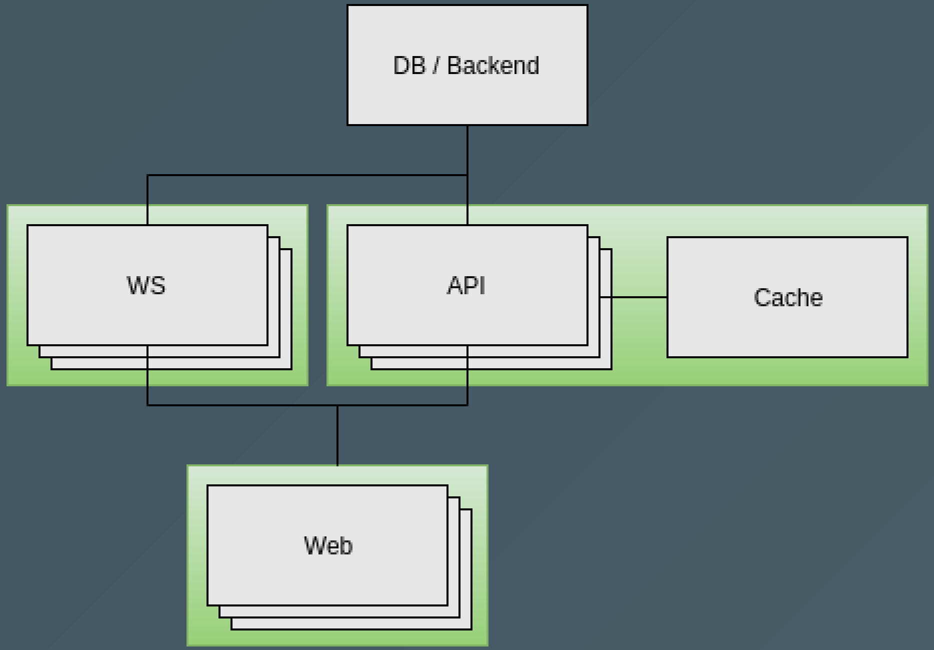
- The features of process monitors can be easily recreated with *systemd*, plus a few other tools
- See my talk here where I did this in 20 minutes:
  - Video: https://opbeat.com/events/nodeconf-oslo-2016/#deploying-and-running-node-js-to-production-in-2016
  - Slides: https://github.com/lukebond/nodeconf-oslo-20160604/blob/master/nodeconf-oslo-20160604.pdf
- Read/watch this for the *how-to*; I won't go into it here

You: I'm happy with my process monitor; why switch to systemd?

- The tooling is better (especially `journalctl`)
- You have more control; you can choose your components
- Run your Node.js apps the same way you run non-Node.js apps
- Any Linux sysadmin will understand it
- No need to reinvent the wheel
- Learn more about the OS on which your apps run
- What runs your process monitor on reboot? Your init system!

</Aside>

More Complex App w/ API + WS + DB



More Complex App w/ API + WS + DB

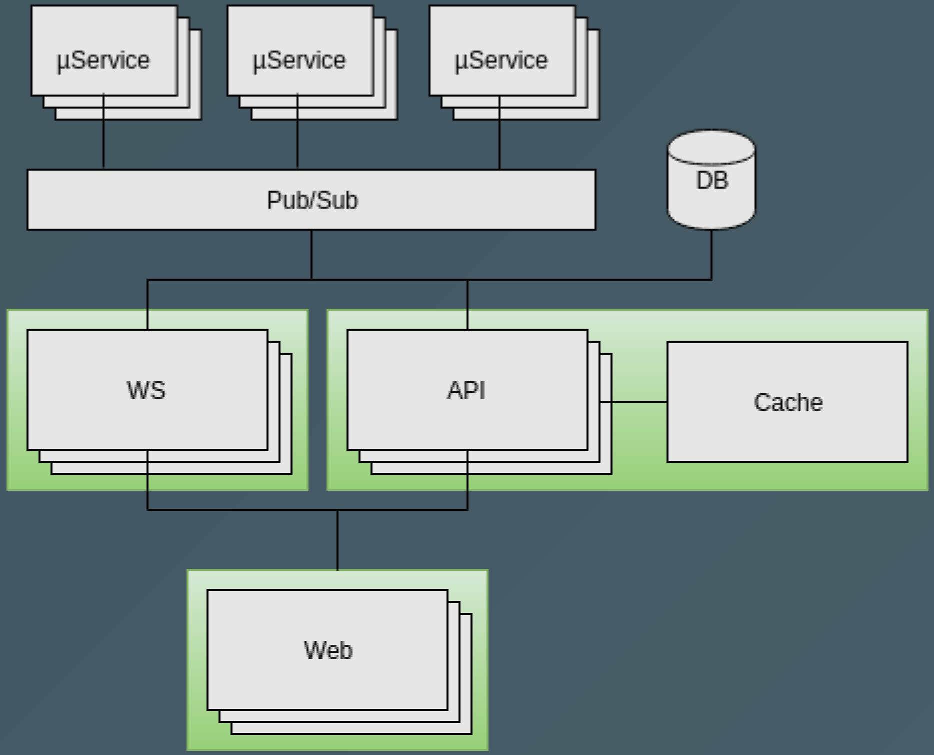
- Easily horizontally scalable
- Communication between components required
- Scale out WS & API by process and across hosts
  - Communication between hosts then required
  - Host affinity becomes important; e.g. Redis on same host as APIs
- How to do service discovery without too much complexity?

More Complex App w/ API + WS + DB

Recommendations: you need service discovery but keep it simple

- Use **AWS CloudFormation** and/or **Elastic Beanstalk**
- Use **Google App Engine**
- Use **Fleet** and a dynamically configured **nginx** or **HAProxy**
  - Easy to transition from using **systemd** to **CoreOS Fleet** 👌
- Use the new **Docker Swarm** in Docker 1.12

Distributed Microservices System



Distributed Microservices System

- You don't want to be thinking about any plumbing
- You don't want to be specifying exactly what runs where
- You want a declarative platform with service discovery
- You want a platform that will reschedule on app or host failures

Distributed Microservices System

Recommendation:

- Use **Kubernetes**
- Use hosted Kubernetes such as **Tectonic** or **Google App Engine**
- Use a PaaS such as **Deis**
- **Docker Swarm** someday- not yet proven at scale IMO

Conclusion & General Guidelines

- **Keep it simple** or suffer the operational pain
  - No unnecessary complexity in stack or deployment
- Being **12-factor** saves you operational pain & brings portability
- **Containers** help if your application is sufficiently complex
  - Devs can test and debug whole system on their laptop
  - Homogeneous deployments even for heterogenous stacks
- Test, test, test; smoke test releases, ensure you can rollback easily
- **systemd** over **PM2** 💪

Thanks!

👋 `T: @lukeb0nd | E: luke@yld.io | G: @lukebond`