QueensJS
June 5th, 2019

Tackling dependency management and dependency migration

… with poise and grace

Introduction

Speaker: Aori Nevo, PhD
Twitter, github, linkedIn: @aorinevo

- Senior Software Engineering Manager @ Pager
- Manage backend and platform teams
- Served in the Army for 4 years
- BA, MA, MS, and PhD in Mathematics
- ...

Goals 

- Communicate the importance of dependency management and dependency migration (DMx2)
- Provide overview of the process
- Discuss some strategies
- Empower developers

Why you should listen to what I have to say?

TL;DR I've done my fair share of migrations… successfully:

- Node 4 -> Node 8
- Node 8 to Node 10 (twice, at two different jobs)
- Webpack 1 to webpack 2
- Webpack 2 to webpack 4
- Hapi 16 to Hapi 18
- Etc ...

Definitions

Dependency management is the process by which a system's dependencies are kept up-to-date. Up-to-date means some range of acceptable versions.

Dependency migration is the process by which a system's dependency is migrated from one version to another.

DMx2 benefits

- Keep up with latest security and vulnerability patches and fixes
- Improved performance
- Reduction in production bugs
- Reduced cost associated with maintaining out-of-date deps - this his HUGE! (think deps that are no longer maintained, deprecated, etc…)
- … and more!


RENOVATE

Renovate

“Save time and reduce risk by automating dependency updates in software projects. Fully customizable with a setting to suit every workflow.”

- https://renovatebot.com/

Renovate does not dictate which versions of a dependency are needed, that’s set by the system’s architect and passed as a configuration to Renovate. Nor does Renovate provide one central view to see that state of your dependencies across applications.

The process

1. Understanding the dependency state of your system in real time

- Create (or buy) a **reporting** tool that can, in real time, output the exact state of dependencies in your system.

	dep1 (lodash)	dep2 (axios)	dep3 (react)	dep4 (express)	dep5 (hapi)
app1 (bag)	1.0.0	N/A	N/A	2.0.0	0.14.0
app2 (credit)	N/A	3.0.0	N/A	2.0.0	10.0.0
app3 (chat)	2.0.0	2.0.0	3.0.0	2.0.0	9.0.0

- Convert the data into a meaningful **metric**
  - Percentage of dependencies out of date (PDOD) = (# of reds)/(# of reds + # of greens)
- **Define** a healthy state for your system based on metrics
  - The system is healthy if PDOD < 20%

Trainings

2.

- Document migration steps and **strategies**
- Create DMx2 training materials
  - Have a deck prepared
  - The deck can serve as a **reference** for when developers work on DMx2 tasks and sub-tasks
- Set clear expectations on next steps
  - Have a list repos with targeted migration completion date (or release)
  - Keep that list up-to-date

3.

Distribute DMx2 tasks across the team

- **Distribute** the work through whatever ticket management system you use (i.e. JIRA, VersionOne) - make sure those tasks are assigned and hold assignees accountable. Be smart here:
    - If the task is small enough, just include it as part of another feature or enhancement you're working on.
    - If large in scope, create that ticket.
- Set clear **target** dates/iterations.
- **Track** progress at least once a week.
    - Repeat the process wherever there is degradation in the health of the system..
    - Take a breather when the system is in a healthy state and direct bandwidth elsewhere for the time being.

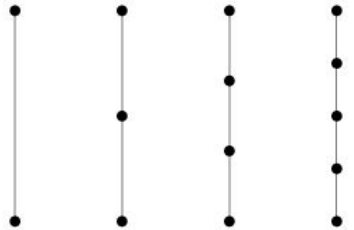
Enforcement

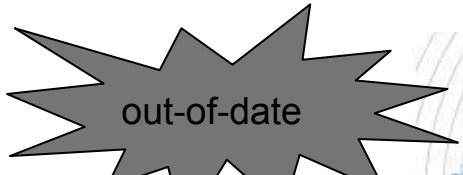
4.

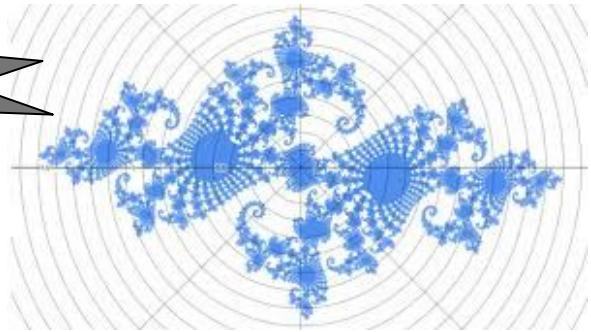
- **Automated** checks to ensure dependencies get updated. Fail builds in CI pipelines for repos where dependencies are out-of-date
- Consider enhancing your custom cli tool to do this task. That is, have the **cli** tool run on whichever CI tooling you use.

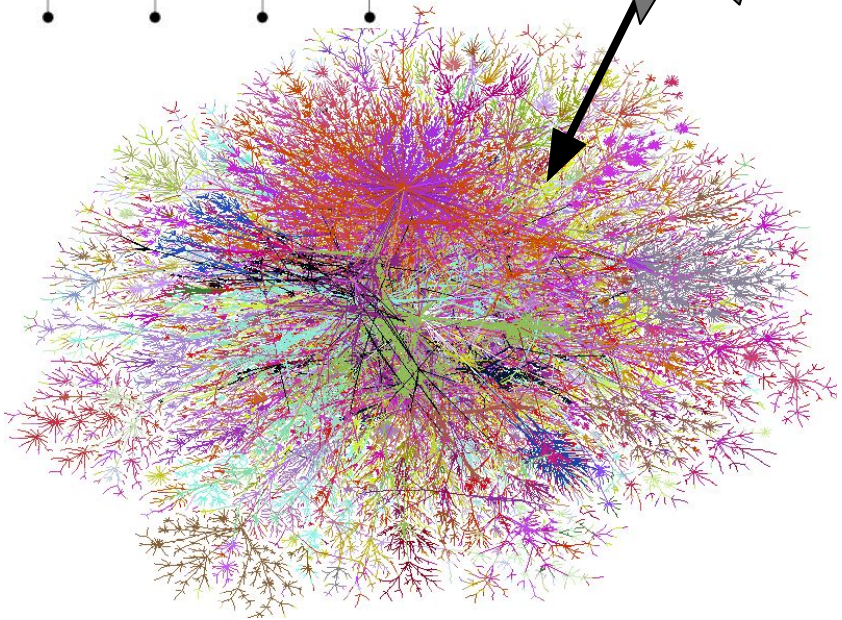
The Strategy

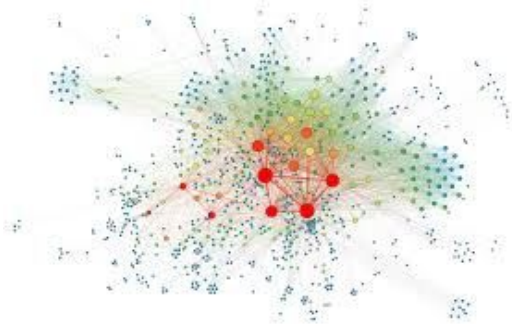
What does your system look like?

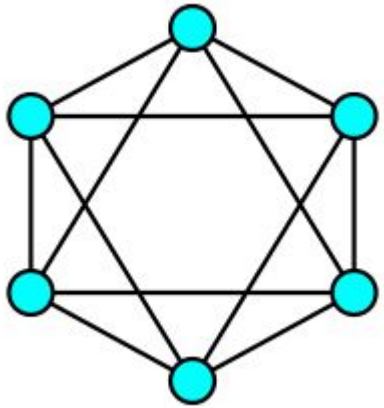


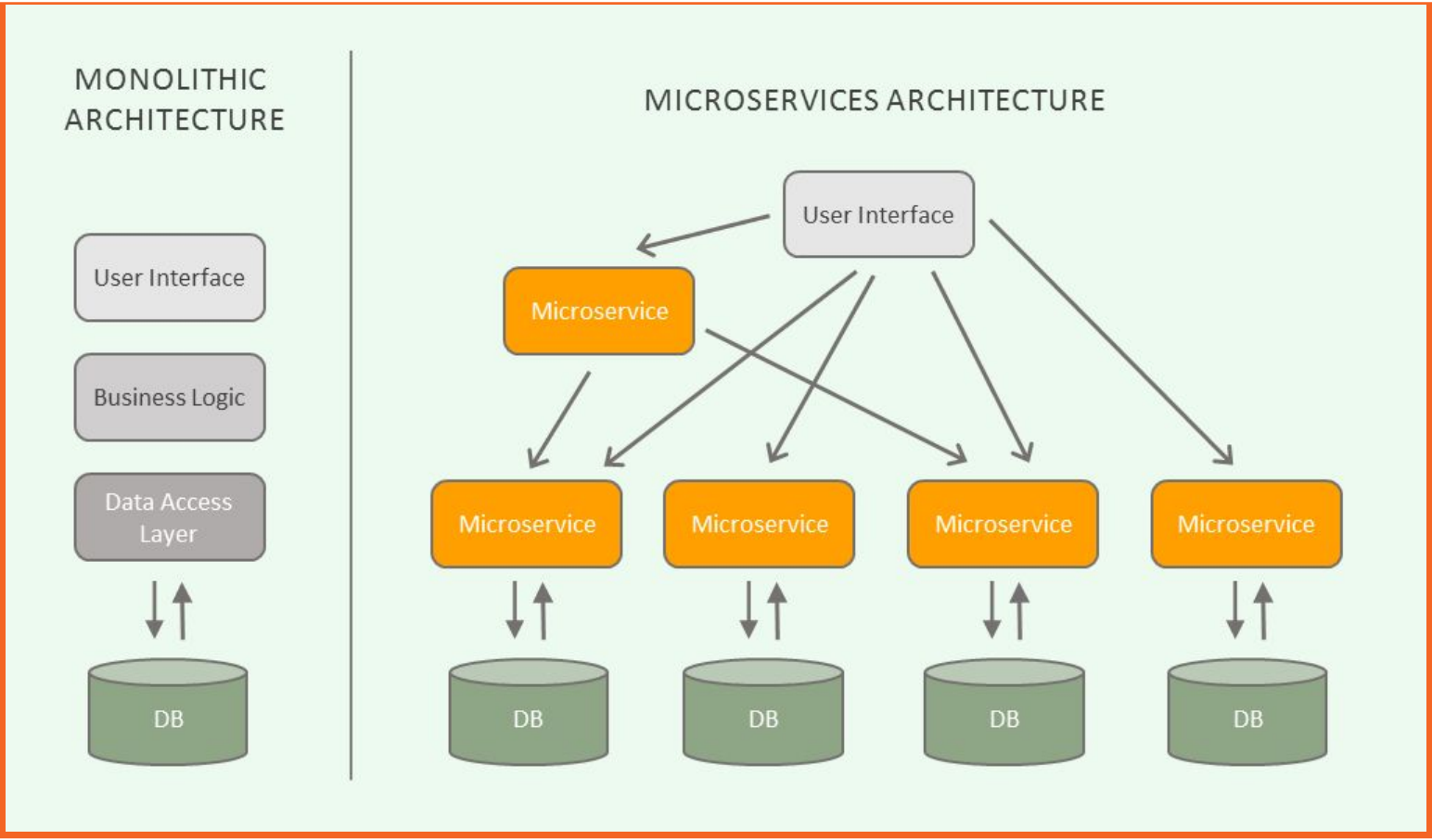






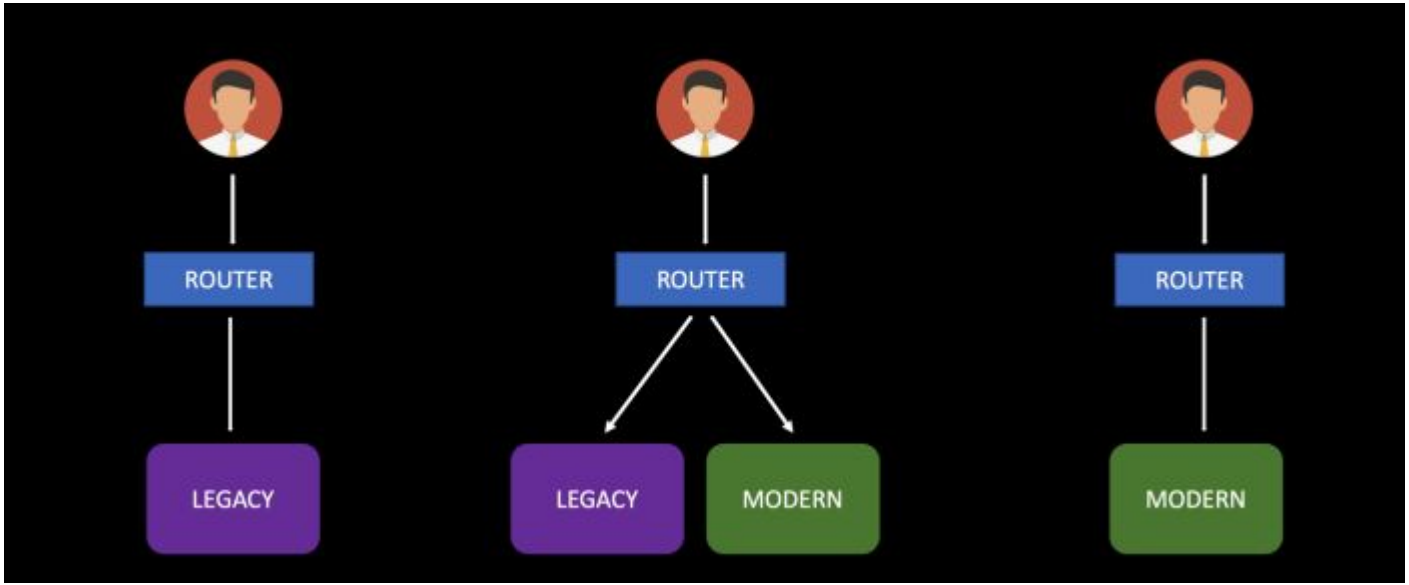






https://blog.algorithmia.com/introduction-to-microservices/

Strangler approach



1 month

∞

Yes, infinity!

Strangler approach (pros and cons)

Pros:

- Very stable
- Um, yea, that's it!

Cons:

- Very costly
- Usually takes a long time
- One time use
- Bloats infrastructure
- Maintain two "codebases" for same user experience

Iterative approach (bottom-up)

- Upgrade dependencies maintaining backwards compatibility
- Use npm and yarn aliasing to install different versions of a dependency
- Given a dependency tree for your app, start with upgrading the lowest common denominators in the tree and work your way up to the root upgrading the deps iteratively

Closing Remarks

- Don’t write off a strategy without thinking it through. Do a small POC to prove out its viability
- This is an on-going, regular part of keeping systems in a healthy state, but ultimately competes for time with other business deliverables. This will require engineers to figure out strategies for themselves to ensure that they get time to work on these tasks alongside their deliverables. Should be accounted for when sizing stories.
- If you’re chipping away at the percentage of out-of-date dependencies, that’s still a win.
- Unit Tests are must!
- Don’t try to apply the same migration strategy to every migration. Look for hints in how the dep was upgraded by the maintainers before upgrading the dep in your app.
- Can we make DMEs and DMx2 a thing?

Thank you!

Q&A