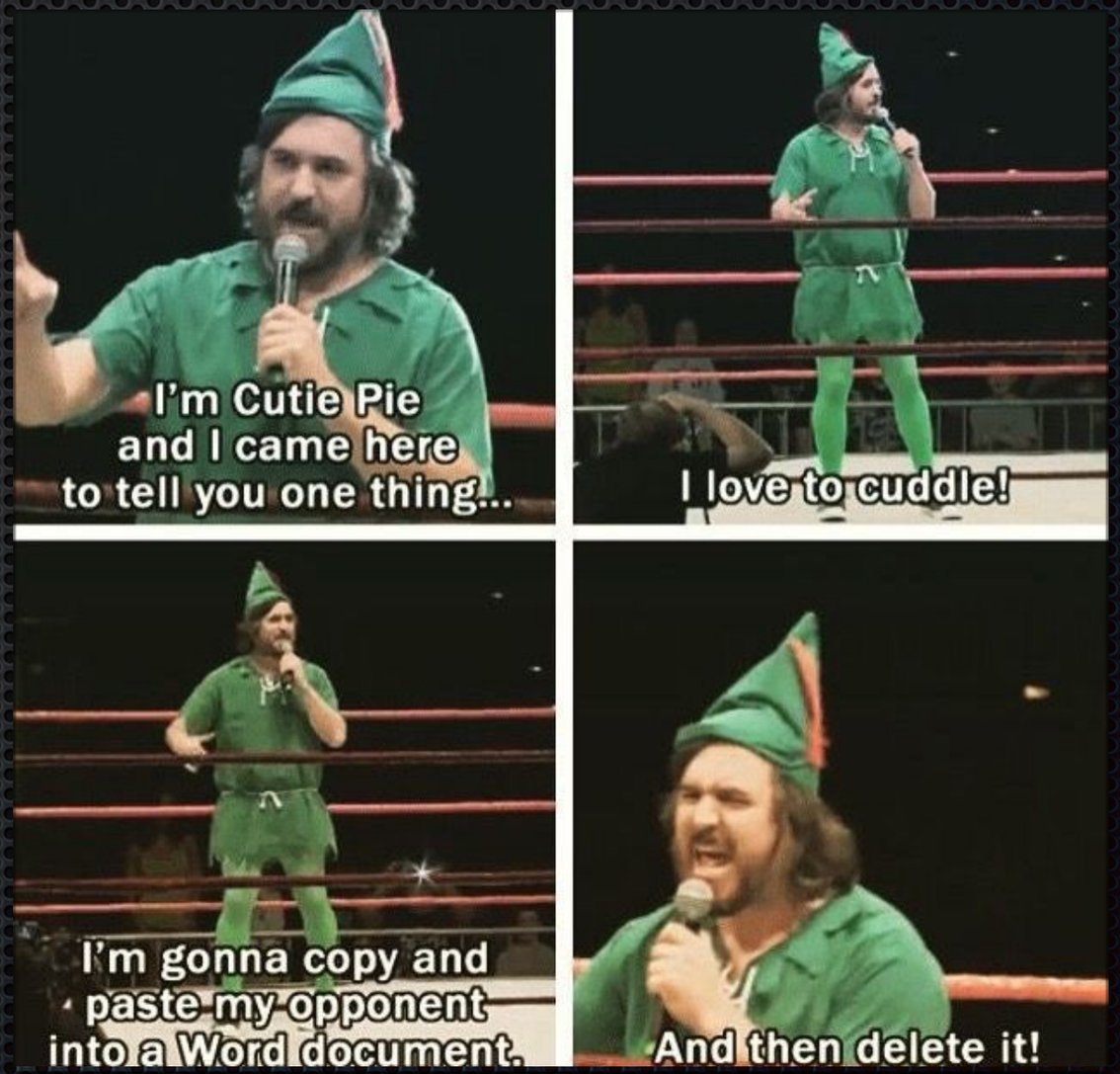
Dockerized Development Environments

Practical examples of efficiency gains.

Before we start, it's import to consider:

Contextual Appropriateness

- Containerization is not suited for contexts where your application manages a machine and it's state
- There is very much still a role for Vagrant in this space



Evaluating Docker as a tool:

OK, so what is Appropriate?

- There has been a lot of praise for the advantages Docker brings to DEPLOYING code in a production context. Let's instead evaluate this tool exclusively from the Development context.

- Qualifying a 'Good Development Environment' is highly subjective. Some practical examples of what Docker does well, are evident if we take a look at a demo project that incorporates these features.

Advantage #1:

Managing Dependencies

- Dockerfiles are the building blocks of any containerized solution that leverages the Docker platform. They are the Recipe required to create your Application's run time.

- This design has beneficial side-affects:
  - Enforced standardization and consistency.
  - All runtime components are now versioned, and present in a container only if required.

Advantage #2:

Compartmentalization

- Docker advocates a “one process” per container philosophy.
- There are inherent advantages to adopting this approach:
  - Every individual support component can now also be versioned.
  - Troubleshooting can be quite simple when we can manage the logs and stdout of all processes in our environment separately and simply.

Advantage #3:

Container Eco-Systems

- Containers are isolated, but they aren't meant to run in isolation. Docker brings the ability to build complex inter-container networking.
- You can model complex production environments locally:
  - Design and Architect your application with minimal resources
  - Enhanced ability for developers to recreate production scenarios to troubleshoot or optimize performance.

A Practical Example:

Where does the code go?

- As a developer, I want to use my IDE on the host machine to manage the code inside the container.

- The Trick: We need to explicitly manage file and user permissions so we match the interior of the container, to what's on our host machine. There are two solutions to this: match at build time, or match at run time.

In retrospect ...

Play to Docker's Strengths

- If you're going to experiment, assess if your application is a good fit for containerization. Webapps are a great fit, system configuration tools are generally not good candidates.

- Docker does Dependency Management, Compartmentalization and System Modeling (via Container Eco-Systems) really well. If you're able to build your development environment on these 3 pillars, you'll find opportunities to gain efficiency.

- If your workflow has different core tenants, you may be looking at the wrong solution.