Containerised Development of a Scientific Data Management System

Ben Leighton, Andrew Freebairn, Ashley Sommer, Jonathan Yu, Simon Cox
Some context

I'm part of a team of developers in Land and Water, we work across project to build environmental infrastructures to better help us manage the environment.

This talk is a bit about how we use containers to make everyone's life a bit better, minimising grind and maximising work on interesting things.

If you haven’t heard of containers before saying they are a bit like lightweight virtual machines is succinct, moderately accurate but unfortunately doesn’t communicate just how fast, simple, composable, reproducible, portable, and descriptive they actually are! There are lots of container technologies but I use and talk about Docker.
Containers are fun, but why?
We fight against entropy!

“Fake Quotes But Real News!”

“It worked on my machine”!

“It worked in dev and in test!”

“A bug in prod is going to be hard to debug because it is so different than dev”

“Great weekend. Unfortunately I’ve no idea how anything worked last week!”

“I’d like to reuse just a bit of that inseparable complex system!”

“I couldn’t spin up a database because I’m not a DBA!”

“I’m fairly sure I’ve not made any assumptions!”

“Oh your new here, well you’ll probably need a while to understand our mess-dev process”

“Our randomly hacking together infrastructure works but we don’t know why!”

“We depend on Heroku or AWS or Google and you’ll need to too and pay them”

“It worked last year!”

“I am shocked that this wiki page of important instructions was never tested, never worked and is out of date!”

“Ops concerns are not my job!”

“Ops concerns are not my job!”
We fight against entropy!

"It worked on my machine!"

"Our randomly hacking together infrastructure works but we don’t know why!"

"Great weekend. Unfortunately I’ve no idea how anything worked last week!"

"I’m fairly sure I’ve not made any assumptions!"

"Oh I am shocked that this wiki page of important instructions was never tested, never worked and is out of date!"

"Oh your new here, well you’ll probably need a while to understand our mess dev process"

The system isn’t well described

Production environments are radically different from dev and test

"It worked in dev and in test!"

"Ops concerns are not my job!"

Communication and collaboration is hard and there is insufficient automation and unnecessary duplication

"A bug in prod is going to be hard to debug because it is so different than dev"

"It worked last year!"

Reproducibility is difficult

Portability is low and there is risk of lock in, wider acceptance, and costs

"We depend on Heroku or AWS or Google and you’ll need to too and pay them"

It is hard to separate and quickly and simply re-compose components

"I couldn’t spin up a database because I’m not a DBA!"

"I’d like to reuse just a bit of that inseparable complex system!"

Production environments are radically different from dev and test

"It worked in dev and in test!"

"Ops concerns are not my job!"

Communication and collaboration is hard and there is insufficient automation and unnecessary duplication

"A bug in prod is going to be hard to debug because it is so different than dev"

"It worked last year!"

Reproducibility is difficult

Portability is low and there is risk of lock in, wider acceptance, and costs

"We depend on Heroku or AWS or Google and you’ll need to too and pay them"

It is hard to separate and quickly and simply re-compose components

"I couldn’t spin up a database because I’m not a DBA!"

"I’d like to reuse just a bit of that inseparable complex system!"
CKAN open source metadata / data management
CKAN is* pretty annoying to install

* Things might be improved in the recent 2.8 release as they include a containerized approach that might actually work (I’ve not tested it yet)

CKAN was so hard that for a while there was a company - Datacats - with a business model for using docker to simplify CKAN installations

CKAN uses:
- Redis
- Postgres
- Solr
- Python Flask

CKAN is highly extensible with an large plugins ecosystem

Command line tools are used to administer aspects of CKAN
Container dev workflows
Developers get to work in ways and with tools they like

Containers encourage breaking up application and sourcing components from elsewhere and so there is reuse

Developers think sooner about dependencies, system architecture, portability, "documentation", and more holistically

Deployments are simply and portable documented as "Infrastructure as Code" i.e. they are executable and provable - Simpler than Ansible, Puppet etc

Any continuous Integration System can deploy, less is hardcoded in CD

Manual Deployments are very similar to CD deployments

Test / Prod

Containers encourage breaking up application and sourcing components from elsewhere and so there is reuse
Developers environments are very different from one another and different from container / prod

Developer Environments are well described they aren’t “Infrastructure as Code”

Container engineering happens soon but not necessarily during dev so maybe devs aren’t thinking holistically enough early enough

Container Engineering is replicating dependency installs and system linkages that “have” to had happened in dev somehow already

Continuous Integration / Deployment Server / Person

Test / Prod
Can we do better?

Relevant bit of the Dev Environments the “running components” are in containers just like in prod

IDES and debuggers are adjacent and run on the main developer machine much like traditional dev

Developers get to “mostly” work in ways and with tools they like

Dev Environments are consistent across developers and actively maintained during feature dev, bug fixes etc. Not as a separate step

Less work as low duplication between dependencies / system structure in dev vs prod

Concerns in Prod e.g. Backup is thought about as part of dev

Automation is portable and applicable in any CD, dev deployments are similar to CD

Prod and Dev should be as similar as possible
CKAN dev what did we do?

Our CKAN container composition includes backup and restore even in dev

If you use PyCharm we inject in remote debugging in slight variant on vanilla containers - currently managed in branches likely to move to a switch based approach

You can use your favorite IDE
Don’t use Docker Bind mounts
Do use sshfs sidecar containers for maximum reliance on container / Docker file

Dev is still in containers and a minor variant on prod, it is easy for new developers to deploy, consistent, single point of truth

Continuous Deployment is mostly a call out to scripts and containers present in dev - you can mimic the CD system in dev
Docker + Jenkins + CD - what did we do?

Jenkins instance itself is Dockerized

Continuous Deployment automatically builds branch commits using dockerized process to test server

By heavily Depending on Dockerized dev / build it was relatively easy to create a variant test to build to test backup and restore by restoring latest backup files

Slack notifications help integrate in Real Time with Team Activity and help quality stay high and insure against reversion to “works on my machine”
Applying these patterns for Kubernetes
Applying these patterns for Kubernetes

Derived from MAGDA an open source product led by Data 61, fast, powerful, well engineered - To be clear the negative quotes at the start of the presentation don’t really apply here. MAGDA is well maintained and high quality

Knowledge Network for data discovery

Complex stack of microservices

Build and Deployment processes are semi automated

Dev environments require extensive setup and are evolving

Local Dev is a Hybrid of K8S Cluster and Local Components

Build and Running

- These instructions assume you are using a Bash shell. You can easily get a Bash shell on Windows by installing the `git` client.

**Prerequisites**

- **Node.js** - To build and run the TypeScript/JavaScript components, as well as many of the build scripts. Version 9+ works fine as of March 2018.
- **Java 8 JDK** - To run the JVM components, and to build the small amount of Java code.
- **son** - To build the Scala components.
- **yarn** - `npm` replacement that makes node deep in a monorepo much easier.
- **lerna** - To manage our multiple project repo. Once you have node, `yarn` installed, installing lerna is as simple as `yarn global add lerna`.
- **pancake** - To manage design components. Once you have Node.js installed, installing pancake is as simple as `yarn global add pancake`.

To push the images and run them on Kubernetes, you’ll need to install:

- **ONU tar** - (Mac only) MacOS ships with `brew install`. However, you will need `tar` for docker images operations. On MacOS, you can install `tar` via `Homebrew`: `brew install tar`.
- **kubectl** - For the (kubectl), tool used to control your Kubernetes cluster. You will also need this to deploy to our test and production environment on Google Cloud.
- **Homebrew** - To manage Kubernetes deployments and config.
- **Docker** - MAGDA uses `docker` command line tool to build docker images.

You’ll also need a Kubernetes cluster - to develop locally this means installing either `minikube` or `docker` (MacOS only at this stage). Potentially you could also do this with native Kubernetes, or with a cloud cluster, but we haven’t tried it.
Applying these patterns for Kubernetes

A K8S dev and deploy container

- Captures dev environment requirements
- Potentially as an image snapshot for fast dev setup
- Can target a shared cluster instead of minikube
- Wildcarded domains for per deployment / container urls
- Parameterizes Helm to deploy to a namespace
Thanks

Links
https://research.csiro.au/ei/
https://github.com/CSIRO-enviro-informatics

Land and Water
Ben, Andrew, Ashley, Jonathan, Jevy
D61
Hendra
Thanks to Magda Team - Alex Gilleran et. al.
e   ben.leighton@csiro.au