Packaging Applications as Containers to Avoid Dependency and Maintenance Nightmares

Matt Blasinski
Software Developer
DoIT Productivity & Collaborative Solutions
Who I Am

- Graduated from UW-Madison in 2005
- Working at DoIT since 2001, full time since 2005
- Developer for WiscMail/Office 365
- Focus primarily on our deployment/management tools
- Using Docker for 1.5 years now
- My cat doesn’t find Docker nearly as fascinating as I do
Why is the speaker wearing a pirate hat?

- Maybe he kidnapped our actual speaker?
- Did he lose a bet?
- Good luck charm against bad hair days when presenting?
- Does it keep the rain off?
- Bringing it back as a fashion trend?
- Rum?
Overview

We’ll talk about an application I made to automatically create change records:

- Identify problems
- Use Docker to address those problems
- Rinse and repeat
Automated/Continuous Deployment

Changes should be:

- Small
- Pushed out quickly and frequently
- Communicated and documented
  - DoIT IT Service Management (change records)
  - Our needs (emails)
Issues with Filing Change Records

- Requires waiting before change can be completed
- Disruption from focus on other work
- Takes time from other work
- Often skipped
Solution

File change records automatically!
Automated Change Records

Iteration #1

Standalone Code and Libraries
Iteration #1

Perl module/script calls Change Management tool’s REST API

Process:

1. Authenticates
2. Get CR template
3. Update values
4. Save CR
5. Make relationships
6. Logout
Dependencies Challenges

- Not all libraries/modules are installed by default
  - e.g., JSON
- Different applications may require different versions of libraries
- Languages may not be available by default
Possible Dependency Solutions

- Can/should we use our deployment tools to install dependencies?
- Can/should we have sys admins install dependencies for us?
Issue #1

We need something that is complete and consistent, so that it runs the same, no matter where we run it.
Docker

Docker:

An open platform for developers and sysadmins to build, ship, and run distributed applications, whether on laptops, data center VMs, or the cloud.
Docker Containers:

Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries – anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment.
Sidebar Example

Email Sending Container
Emailer - Problems

- Same few lines of code to send an email are repeated frequently
- Need library/method available everywhere
Emailer - Solution

- Made this a container instead of writing code over and over or keeping libraries maintained
- Dependencies, libraries, and scripts all packaged together
- Runs like a single command/script
FROM perl

MAINTAINER mailteam-tech@office365.wisc.edu

RUN cpanm Email::Sender && cpanm Email::Stuffer && rm -rf .cpanm

COPY command/lib/Emailer.pm /emailer/Emailer.pm

COPY bin/emailer-app.pl /emailer/emailer-app.pl

ENTRYPOINT /emailer/emailer-app.pl
```
docker run -it --rm \
-e "email_to=matt.blasinski@wisc.edu" \
-e "email_from=Matt Blasinski" \
-e "email_from_address=mablasinski@doit.wisc.edu" \
-e "email_subject=This is a test message" \
-e "email_body=Here is the content.\nHere's more content." \
--name emailer docker.doit.wisc.edu/ecc/emailer:latest
```
Automated Change Records

Iteration #2

Container
Iteration #2 - Advantages

- Moved previously written libraries and scripts into a container
- Can now package other libraries with it
  - Bye-bye homemade JSON parser!
Iteration #2 - Drawbacks

- Has a lot more config than emailer
  - Can’t build config into container (especially secrets)
  - Needs to be passed at runtime
    - Command line environment variables
    - Environment files
    - Volume mounts
  - User needs to worry about the config
Iteration #2 - More Drawbacks

- Container won’t work on machines that don’t have Docker installed
- Just replaced some dependencies (libraries) with different ones (Docker)
- Passing parameters is not elegant
Issue #2

Docker can help package dependencies together and set up the environment, but it might not be the best user interface to call services.
Automated Change Records

Iteration #3

Microservices
Iteration #3 - Microservices

- Small, single-purpose web services
- Use common, lightweight protocols - HTTP(S)
- Can be thrown away and rebuilt in a small period of time
Iteration #3 - Microservices

- Web Service Wrapper
  - Dancer (Web Services Framework)
    - Loads config and libraries, does auth checks, etc
  - Twiggy (Web Server)
    - Handles web requests
- Docker container loads the config at startup
- Expose code that ran in docker/command line as a web service
Iteration #3 - Microservices

All the user needs to do is supply the parameters for creating change records
Additions to original Dockerfile….

EXPOSE 6443

COPY command/lib/CherwellUtil.pm /wiscit/CherwellUtil.pm

COPY lib/WiscitWrapper.pm /wiscit/WiscitWrapper.pm

COPY bin/wiscit-entrypoint.sh /wiscit/wiscit-entrypoint.sh

COPY bin/wiscit-app.pl /wiscit/wiscit-app.pl

COPY etc/base_config.yml /wiscit/base_config.yml
#!/usr/bin/env perl

use lib './';
use warnings;
use strict;
use Dancer;
use WebServiceWrapper;

my $appdir = '/wiscit';
Dancer::Config::setting('appdir', $appdir);
Dancer::Config::setting('confdir', $appdir);
Dancer::Config::load();
dance;
#!/bin/bash -e

# construct the entire config file from the base, allowed_hosts, and auth_basic files

cat /wiscit/base_config.yml /secrets/allowed_hosts.yml /secrets/auth_basic.yml > /wiscit/config.yml

twiggy --server Twiggy::TLS --tls-key /etc/ssl/private/wiscit.key --tls-cert /etc/ssl/certs/wiscit.cert --listen :6443 /wiscit/wiscit-app.pl
Call:

curl 'https://localhost:6443?target=mbv-dev&desc=my%2C%20change'
Issue #3

We’re using a lot of containers, so we need to make a lot of updates whenever we make changes.
Automated Change Records

Iteration #4

Customized Parent
Iteration #4 - Customized Parent

- Base all similar images off a single, customized parent image that you maintain
- Build layers
  - Copy specific libraries/scripts into children as needed
- Lowest common denominator
- Run security scans on the parent image
Key Takeaways
Bite Off Small Pieces

- Start with something you know works
- Iterations are important
- Updating containers is easy
Microservices

- Expose containers via web services
- Split them if they grow too large
- Quick and easy to modernize/redo as needed
K.I.S.S.

- Use layers to avoid duplication
- As components get smaller, complexity shifts to interactions between them
- Use orchestration and configuration management tools
Thanks for listening!

Matt Blasinski

matt.blasinski@wisc.edu
Questions?

“Questions are the easy part....answers raise a doubt.”
-Jimmy Buffett

Matt Blasinski
matt.blasinski@wisc.edu