

# Dino apps deserve love too!

Michael Irwin - April 25, 2018 - GOTO Chicago

You're using Docker? Does that mean...



“Legacy apps deserve love too”

- Steve Singh, Docker CEO

DINO  
~~LEGACY~~ APPS

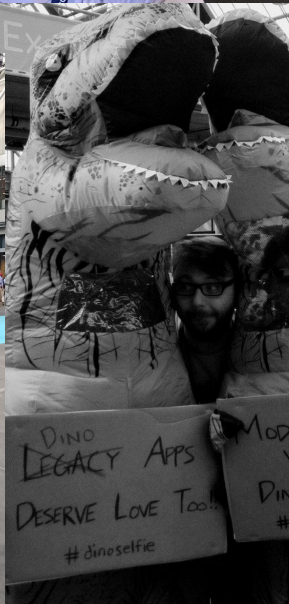
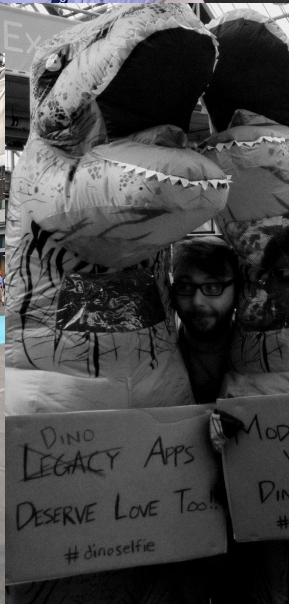
DESERVE LOVE TOO!!

#dinoselfie

MODERNIZE  
YOUR  
DINO APPS

#dinoselfie





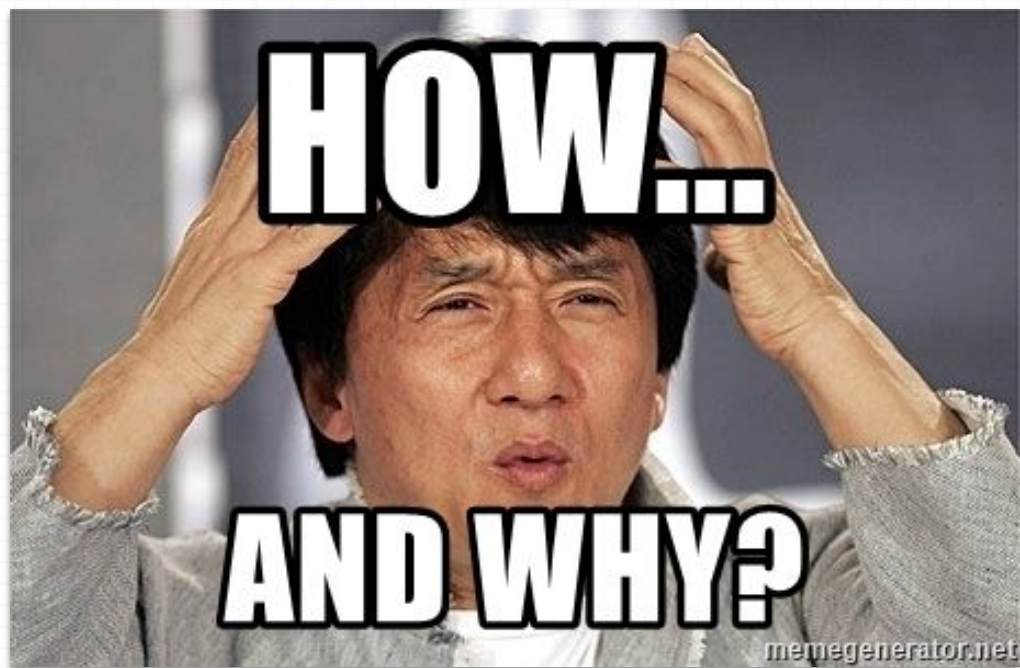
# “Dino apps” = ???

Any app not currently under active development





Photo credit: <https://observationdeck.kinja.com/want-to-join-a-dinosaur-dig-this-summer-you-can-1584367030>

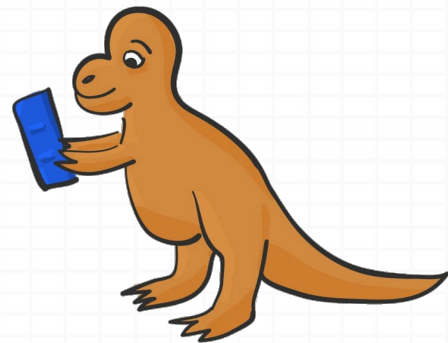




# The Migration Journey

1. Build the app
2. Put it in a pipeline
3. Build the runtime
4. Test the app

... all without changing a single line of code in the app itself!



# Pre-Migration Notes

- App needs to allow external configuration
  - Environment variables, external config files, etc.
  - Obviously, secrets should NEVER be baked into your images
- Each step's completion should be considered a success
  - You don't have to complete all of them to gain benefits

# Our Dino Application - CheapChomp

- Simple webapp that displays specials at campus dining centers
- Admins can log in and schedule upcoming specials
  - Global admins can modify all
  - Local admins can modify only specific dining centers

CheapChomp

HelpLogin

Upcoming Specials

Owens Food Court	
Mangia!	
Mar 12	Extra meat spaghetti
Olives	
Mar 12	Gyros from Chloe

Copyright © 2007 Virginia Polytechnic Institute and State University. All rights reserved.

Bug/Feature Request



# Step One...

1. Build the app
2. Put it in a pipeline
3. Build the runtime
4. Test the app

# Build the app (applies really only to compiled apps)

- **Goal** - be able to reliably reproduce builds for the app itself
- Things to think about...
  - What build tools are needed?
  - What versions for those tools are known to work?
  - Are dependencies pinned to specific versions?
- Some tips...
  - Use docker-compose to make it easy to launch
  - Consider mounting artifact repos to enable faster builds
  - Consider mounting output back onto host

Let's build this app!



## Step Two...

1. Build the app
2. Put it in a pipeline
3. Build the runtime
4. Test the app

# Put in a pipeline

- **Goal** - enable automated builds to produce images upon code changes
- Things to think about...
  - Use “Pipeline as Code” whenever possible
  - Use whatever is available to you. Don’t run your own build server if you don’t have to
  - Don’t make your build scripts so complicated you can’t run it on your own machine
  - If your build system allows it, save the output from Step 1 as an artifact

Let's show a pipeline!



# Pipeline Feedback in GitLab

The screenshot displays the GitLab web interface for a project named 'cheapchomp'. On the left, a sidebar contains navigation links: Overview, Repository, Issues (0), Merge Requests (0), CI / CD, Pipelines (selected), Jobs, Schedules, Environments, and Clusters. The main content area shows the breadcrumb 'mikesir > cheapchomp > Pipelines > #11885'. A green status box indicates 'passed'. Below this, it states 'Pipeline #11885 triggered 4 days ago by mikesir'. A section titled 'Put app into pipeline' shows a summary: '1 job from master in 12 seconds (queued for 2 seconds)'. A commit hash 'd99ad92a' is displayed with a link icon and a file icon. At the bottom, a 'Pipeline' tab is active, showing a 'Jobs' count of 1. Under the 'Build' section, a job named 'compile' is shown with a green checkmark and a refresh icon.

C cheapchomp

- Overview
- Repository
- Issues 0
- Merge Requests 0
- CI / CD
- Pipelines**
- Jobs
- Schedules
- Environments
- Clusters

mikesir > cheapchomp > Pipelines > #11885

passed Pipeline #11885 triggered 4 days ago by mikesir

### Put app into pipeline

1 job from master in 12 seconds (queued for 2 seconds)

d99ad92a ...

Pipeline Jobs 1

Build

compile

# Step Three...

1. Build the app
2. Put it in a pipeline
3. Build the runtime
4. Test the app

# Build the Runtime

- **Goal** - be able to produce a Docker image that contains everything needed to run the app, excluding external services
- Things to think about...
  - What server is being used? What about it's runtime?
  - What libraries/modules/extensions are added?
  - Make note of things that would vary between apps (config, etc.) to identify what's universal across apps and what changes
- Once you can build the runtime, update your build pipeline!

Let's build the runtime!



## Step Four...

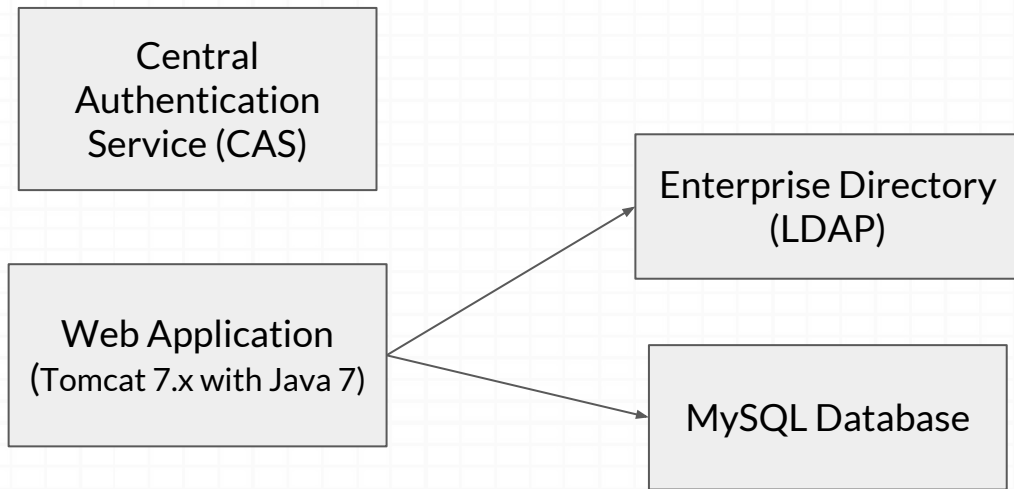
1. Build the app
2. Build the runtime
3. Put it in a pipeline
4. Test the app

# Test the app

- **Goal** - be able to spin up an isolated environment to run end-to-end automated tests of the app
- Things to think about...
  - Use the right tool/language for the job - these tests don't have to be written in the same language/framework as the app
  - What external services does your app depend on? Do you have images for them?
  - What mock data do you need to test each scenario?

# CheapChomp Architecture

- Dependencies on external services run by other units
  - ED contains people and group memberships (used for authorization)



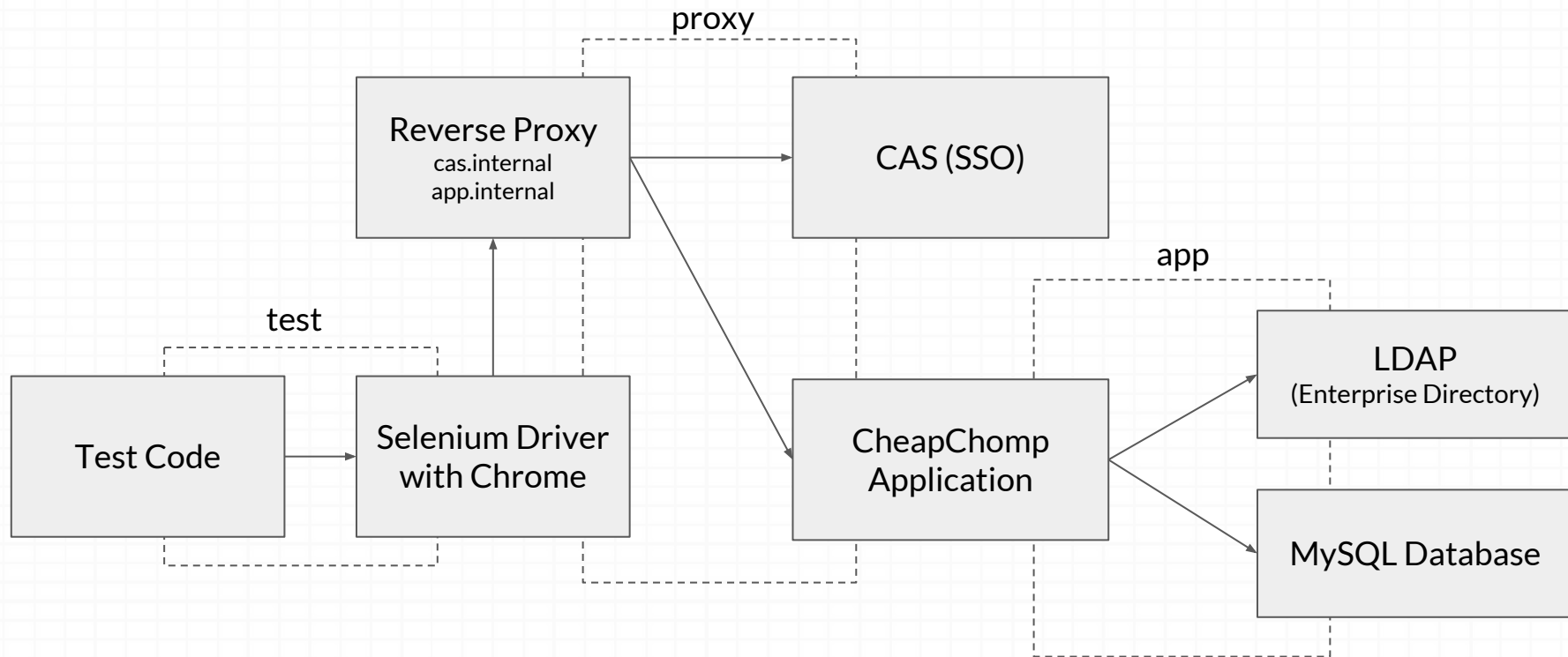
# Testing Pre-Docker

- Tied to whatever capabilities service providers give us
  - We can only use data that exists in the directory
- How do we authenticate in an automated way?
  - If using a real SSO service, don't want to include credentials in source

# Solutions for Testing Pre-Docker

- Don't test (obviously scary!)
- Test manually
  - Works, but time and people intensive
  - If on faster release cycles, repetition => fatigue => missed regressions
- Add abstraction layers
  - Instead of using people from the directory, use “local” people
  - Adds complexity to the codebase
  - Dummy data = no risk of exposure/data loss

# Testing with Docker





# To test our application...

- We want to start our entire application (good use for Docker Compose)
- Using the `--exit-code-from` flag, we get...
  - A stack that tears down when the test container completes
  - The exit code for docker-compose is the exit code of the test container

```
docker-compose -f docker-compose-tests.yml -p cheapchomptests up \  
  --force-recreate --build --exit-code-from bdd
```

Let's test our app!

# Recapping our Journey


1. Build the app
2. Put it in a pipeline
3. Build the runtime
4. Test the app

... all without changing a single line of code in the app itself!






# Because there's this...

 docker trusted registry

Search





mikesir

Repositories > mikesir/cheapchomp > Images

 mikesir/cheapchomp

No description

INFOIMAGESMANIFESTSWEBHOOKSPOLICIESSETTINGS

<input type="checkbox"/>	TAG	OS/ARCH	ID	SIZE (COMPRESSED)	LAST PUSHED	VULNERABILITIES	
<input type="checkbox"/>	tomcat-7-jre7	 amd64	 7f5ce52aa7	214.25 MB	🕒 13 minutes ago by  mikesir	 <b>26 critical 57 major 7 minor</b>	<a href="#">View details</a>

Previous

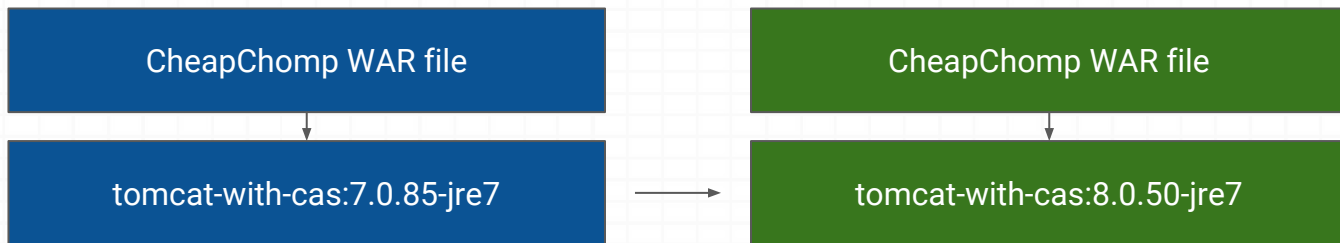
Next

Items per page 10 25 50 100



# Rebasing our App

- As an example, by simply changing the FROM command in our Dockerfile, we can move our app to Tomcat 8
- The tests we wrote in the previous step will let us know if this is safe!




# Update your app!

- **Goal** - be able to migrate our app to newer/updated runtimes
- Things to think about...
  - Everything can be changed! Runtime, build tools, app dependencies, etc.
  - But... make one change at a time
  - Make minimal (if any) changes to your E2E tests to assure their validity
- This is where the true power comes!

Show me!


# Results...

 **docker** trusted registry









Search

mikesir


Repositories > mikesir/cheapchomp > Images

 **mikesir/cheapchomp**  
No description

INFOIMAGESMANIFESTSWEBHOOKSPOLICIESSETTINGS

<input type="checkbox"/>	TAG	OS/ARCH	ID	SIZE (COMPRESSED)	LAST PUSHED	VULNERABILITIES	
<input type="checkbox"/>	tomcat-8-jre7	 amd64	 a31b0e729c	214.66 MB	14 minutes ago by  mikesir	 <b>26 critical 51 major 7 minor</b>	<a href="#">View details</a>
<input type="checkbox"/>	tomcat-7-jre7	 amd64	 7f5ce52aa7	214.25 MB	19 minutes ago by  mikesir	 <b>26 critical 57 major 7 minor</b>	<a href="#">View details</a>


# Swapping to an Alpine base...

 **docker** trusted registry

Search

mikesir

Repositories > mikesir/cheapchomp > Images

 **mikesir/cheapchomp**  
No description

INFO










IMAGES

MANIFESTS

WEBHOOKS

POLICIES

SETTINGS

<input type="checkbox"/>	TAG	OS/ARCH	ID	SIZE (COMPRESSED)	LAST PUSHED	VULNERABILITIES	
<input type="checkbox"/>	tomcat-8-jre7-alpine	 /amd64	 29d9e11705	86.99 MB	🕒 10 minutes ago by  mikesir	🚨 5 critical 14 major	<a href="#">View details</a>
<input type="checkbox"/>	tomcat-8-jre7	 /amd64	 a31b0e729c	214.66 MB	🕒 16 minutes ago by  mikesir	🚨 26 critical 51 major 7 minor	<a href="#">View details</a>
<input type="checkbox"/>	tomcat-7-jre7	 /amd64	 7f5ce52aa7	214.25 MB	🕒 20 minutes ago by  mikesir	🚨 26 critical 57 major 7 minor	<a href="#">View details</a>

# Looking at the remaining vuln's

- My app itself has a vulnerability!
- Options are to...
  - Update the library being used (if possible)
  - Raise the bar

COPY

file:2fd6ba5b8304db5fb9f0d460f0c5faf737d91e7e434c240c05e89891

5.38 MB

[show](#)

COMPONENTS (18)

VULNERABILITIES (1) ▾

[taglibs-standard 1.1.2](#)

**1 critical**

[ldaptive 1.0.6](#)

[ldaptive-unboundid 1.0.6](#)



# Running our app in read-only mode

- Update our docker-compose to include `read_only: true`
- For any file locations that do need to update, use tmpfs
  - Files are only stored in memory (so be careful of large files)
  - Container stops = changes gone

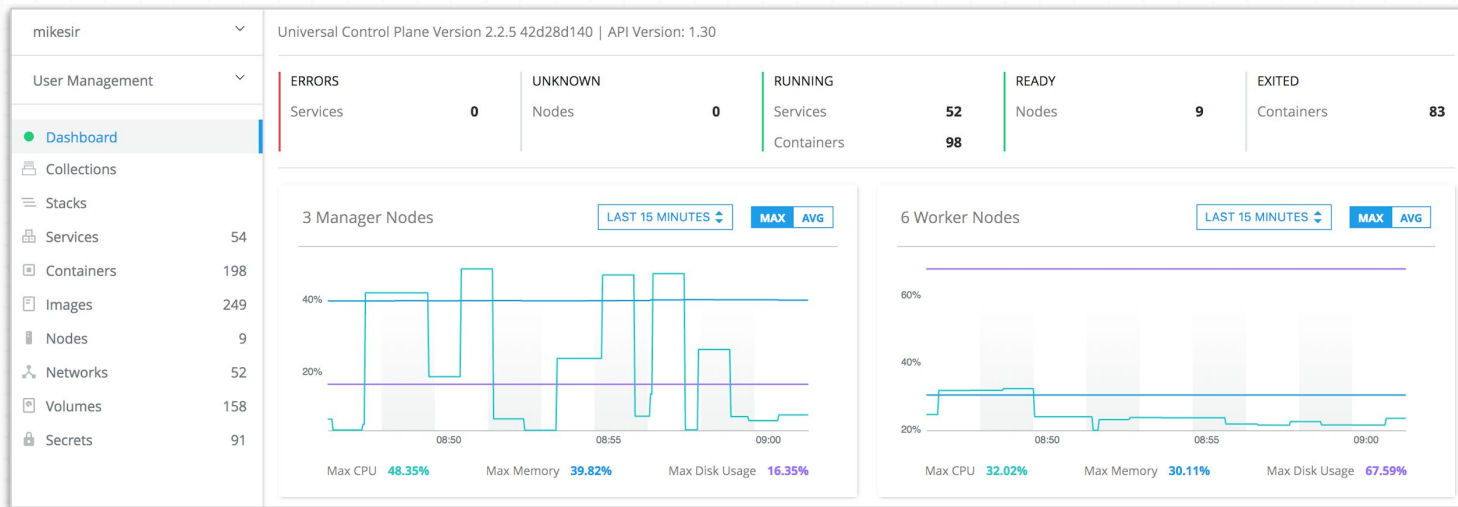
# A quick note about using read-only

- This makes the filesystem of the individual container read-only
  - You are still subject to SQL injections, improper data exposure, etc.
- This is NOT meant to be a catch-all fix
  - Each vulnerability needs to be evaluated and mitigated appropriately
- What is meant to be is a raising of the bar, another tool, another experiment for you to try
  - Other experiments include using apparmor, seccomp, etc.

Show me!

# And finally... visibility!

- With UCP, we finally have visibility of what's running where
  - Our Security Office is obviously happy about that



# Recap

- We modernized our “dino app” by...
  - Containerizing our build process
  - Putting it into a build pipeline
  - Containerizing the entire runtime
  - Add tests to exercise our app
- That allows us to now...
  - Experiment with new runtimes, updated dependencies, or new bases
  - Allow our app to run read-only
  - Have full visibility of all apps we’re running

Without changing a single line of code in the app itself!

# Thank you! Questions?

mikesir@vt.edu / @mikesir87

