BCNET
Shared IT Services for Higher Education & Research

# Conference 2017

## Docker Banner

Spend less time on maintenance, and more time on projects

# An intro by Dave

1. Warren Marusiak - Development
2. James Pohl - IT Infrastructure
3. Dave Kubert - IT Security

# The problem we tried to solve

1. How to develop a new feature in Banner, and deploy it to production in a low risk, repeatable manner?
2. How to perform a Banner upgrade in a low risk, repeatable manner?
3. Solving the first problem leads to a solution to the second problem

# Directly modify production

1. Risky
2. Apply a non-trivial modification to BANINST1.BWSKALOG, or WTAILOR.TWBKWBIS
3. BANINST1.BWSKALOG
    a. ~10000 lines
    b. ~90 functions, and procedures
    c. Referenced in 36 distinct packages
4. WTAILOR.TWBKWBIS
    a. ~8000 lines
    b. ~100 functions, and procedures
    c. Referenced in 344 distinct packages

# Directly modify production

1. What if feature takes more than a single day to implement?
    a. Unfinished code sitting in production
2. What if 5 developers implement 5 features?
    a. Even worse
3. What if 5 features interact in unforeseen ways?
    a. Hard to debug

# Use a development environment

1. Create a development environment
2. Implement the feature
3. Test the feature
4. Promote the feature to production, from development
5. Decommission the development environment

# Features of a development environment

1. A duplicate of production
    a. Database
    b. Software
    c. Hosts
    d. Hardware
2. Replaceable
    a. Create banner_dev_01
    b. Use banner_dev_01
    c. Deploy to production from banner_dev_01
    d. Decommission banner_dev_01

# How to get a development environment?

1. Clone of production database
    a. BANPROD becomes BANDEV01, BANDEV02, etc.
2. Dev instances of Banner services
    a. Forms, self service, BEP, EIS, RabbitMQ, etc.
3. Duplicate file system resources
    a. Forms .fmb/fmx, c program, cobol programs
4. Use Docker, git, and F5 to drive most of this

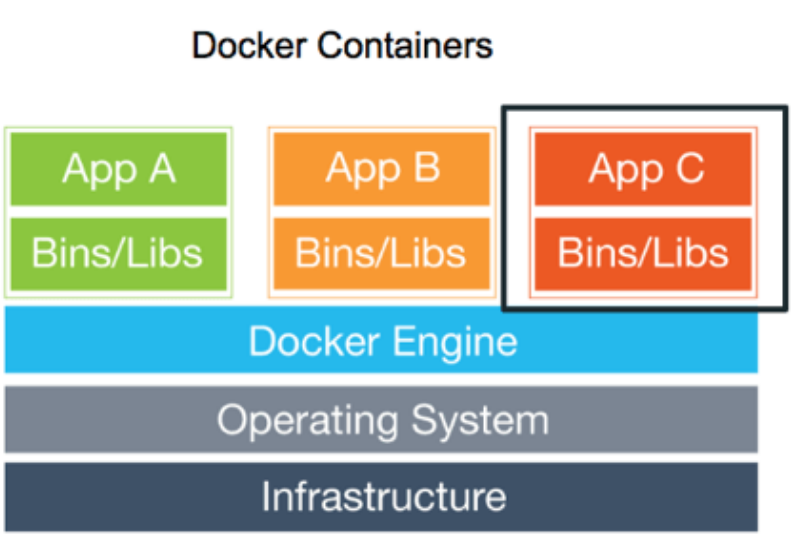
# 5 developers working on 5 features

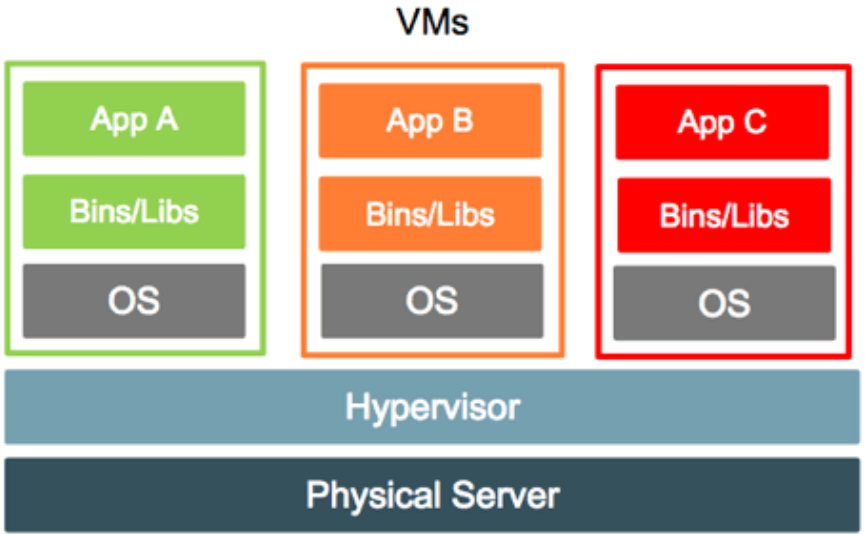
1. Each feature has a dedicated development environment
2. Allows isolated development, and testing
3. Multiple developers can work on a single feature in a single development environment

# Containerization

1. Containerization is a way of wrapping up a piece of software, and all of its dependencies, in a self contained, runnable piece of software that isolates the things inside it, from everything else
2. Isolation allows incompatible pieces of software to run side by side, on the same host
3. We use Docker for containerization

# Containers vs vms



# Docker

1. Docker overhead is negligible
2. Startup, and shutdown quickly
3. Images are small
    a. A tomcat 8.5 image without an OS is around 140 MB
    b. A tomcat 8.5 image with an OS is around 370 MB
    c. A solr image without an OS is around 270 MB
    d. A solr image with an OS is around 520 MB
4. No internal storage
5. Utilize storage on the host

# How to Docker?

1. There are 2 principal components to think about when working with Docker
2. Images
    a. Images are like a class in OOP
    b. Created by compiling source code into a binary
    c. Can be instantiated many times, with varying parameters
3. Containers
    a. Containers are like objects in OOP
    b. A running instance of an image

# Dockerfile

```
1  FROM alpine:3.4
2
3  MAINTAINER Warren Marusiak <marusiak@unbc.ca>
4
5  RUN apk add --update \
6          git \
7          python3 \
8  && rm -rf /var/cache/apk/*
9
10 RUN mkdir repo
11
12 WORKDIR /repo
13
14 COPY entrypoint.py /
15
16 ENTRYPOINT ["/usr/bin/python3", "/entrypoint.py"]
17
18 CMD ["--version"]
19
```

# Entrypoint

```
#!/opt/rh/python33/root/usr/bin/python
import os
import sys
import argparse
import threading
import shutil
import fileinput
import subprocess
import signal
import time

parser = argparse.ArgumentParser('docker entrypoint parameter parser')

parser.add_argument('--rabbitmqhost', dest='rabbitmqhost', action='store', default='rabbitmq.unbc.ca', help='rabbitmq host')
parser.add_argument('--rabbitmqport', dest='rabbitmqport', action='store', default='443', help='rabbitmq port')
parser.add_argument('--rabbitmqadminuser', dest='rabbitmqadminuser', action='store', default='admin', help='rabbitmq admin username')
parser.add_argument('--rabbitmqadminpass', dest='rabbitmqadminpass', action='store', help='rabbitmq admin password')
parser.add_argument('--dbhost', dest='dbhost', action='store', default='pg-uni-oradbe-01.unbc.ca', help='database hostname')
parser.add_argument('--dbport', dest='dbport', action='store', default='1521', help='db port')
parser.add_argument('--dbsid', dest='dbsid', action='store', default='BPREPROD', help='db sid')
parser.add_argument('--baninst1pass', dest='baninst1pass', action='store', help='baninst1 password')
parser.add_argument('--eventspass', dest='eventspass', action='store', help='events password')
parser.add_argument('--cdcadminpass', dest='cdcadminpass', action='store', help='cdcadmin password')

options = parser.parse_args()
```

# Docker images

1. docker build -t unbc/foo:r1.1 .
2. Looks in current directory for a file called Dockerfile
3. Compiles the Dockerfile into an image and tags it unbc/foo:r1.1
4. Image unbc/foo:r1.1 can be run on any docker host

```
[root@fuson rabbitmq]# docker images
REPOSITORY                      TAG                 IMAGE ID            CREATED             SIZE
unbc/foo                        r1.1                9a2a64740edc        1 seconds ago       262.4 MB
unbc/odi                        latest              cb4fe4169da1        4 days ago          944.5 MB
unbc/ssomanager                 latest              19e1367ecd00        5 days ago          1.718 GB
unbc/weblogic1036adminserver    2017R1              eebf86fc2338        7 days ago          1.639 GB
unbc/ethos                      latest              1ecff41119c9        7 days ago          1.226 GB
```

# Docker containers

1. Docker containers are created by running a docker image on a docker host
2. From the official tomcat docker hub page
    a. docker run -it --rm -p 8888:8080 tomcat:8.0
    b. docker run -it --rm -p 8889:8080 tomcat:8.0
3. We have 2 instances of tomcat:8.0 running on the same host, one watching traffic on 8888, the other watching traffic on 8889

# Docker run

```
docker run --name prodformsreports \
--restart=always \
--log-opt max-size=500m --log-opt max-file=2 \
-p 8888:8888 -p 7001:7001 \
-v /data/banner/forms/ellucian/prod/bin/:/u01/app/oracle/forms/baseline/ \
-v /data/banner/forms/unbc/prod/bin/:/u01/app/oracle/forms/custom/ \
-v /data/banner/web_resources/prod/public_http/:/u01/app/oracle/ssb/public_http/ \
-v /data/banner/directories/images:/u01/app/oracle/forms/images/ \
-d unbc/formsreports \
-w <weblogic_pass> -h tnsserver.local -n 3838 -u www_user -p <www_user_pass> -s BANPROD -c
printserver.local -m mailserver.local -f -r
```

# Docker compose

```
[root@pg-uni-docker-02 xwiki]# cd /u105/data/docker/docker-compose/banner/
[root@pg-uni-docker-02 banner]# ls
banprod  base  bdev03  bdev04  bdev05  bdev06  bdev07  bdev08  bdev09  bpre
[root@pg-uni-docker-02 banner]# cd bdev03/
[root@pg-uni-docker-02 bdev03]# ls
authenticate  bannersso  bannerwl  bannerXE  banportals  bep  formsreports
[root@pg-uni-docker-02 bdev03]#
```

1. Compose helps manage the complexity of run commands
2. Compose file per image per environment
3. Contains all parameters necessary to run the image in the environment

# Docker compose

```
[root@pg-uni-docker-02 base]# cat formsreports
version: "2"
services:
 formsreports:
  image: unbc/formsreports:2017R2
  ports:
  - "100${DEV_ENV}:8888"
  dns_search: unbc.ca
  dns:
  - 142.207.144.4
  - 142.207.144.5
  logging:
   options:
    max-size: "500m"
    max-file: "4"
  restart: always
  volumes:
  - /u105/data/banner/forms/ellucian/$DATABASE_ENV/bin/:/u01/app/oracle
  - /u105/data/banner/forms/unbc/$DATABASE_ENV/bin/:/u01/app/oracle/for
  - /u105/data/banner/forms/unbc/$DATABASE_ENV/patch/:/u01/app/oracle/f
  - /u105/data/banner/web_resources/$DATABASE_ENV/public_http/:/u01/app
  - /u105/data/banner/directories/images:/u01/app/oracle/forms/images/
  command:
  - -frl
  - --wlpass
  - "${SECRET_weblogicpassword}"
  - --ldaphost
```

# Docker Banner at UNBC

1. Hosts, storage, and source control
2. Images, containers, and compose
3. Load balancer
4. Password management

# Hosts, storage, and source control

1. Collection of physical, and virtual servers running Docker daemon
2. The more, the better
3. Commodity hardware
4. Minimal install
    a. OS, essential packages, and Docker daemon
5. Any image, any environment, any host
6. Data in Docker storage tracked via git

# Hosts, storage, and source control

1. Hosts mount storage containing forms, plsql, c, cobol, etc
2. Tracked via git locally
3. Pushed to, and pulled from bitbucket which is a Docker image itself

# Hosts, storage, and source control

1. Everything is tracked via git
2. Git is the key to our ability to promote code from development to production
3. Changes are made in dev, test, committed, and pushed to bitbucket
4. Changes in bitbucket can be pulled to another location
5. Using git to move changes around guarantees nothing gets missed
6. Allows us to do diffs between environments

# Images, containers, and compose

1. Many images
    a. Student API
    b. Banner 9 Student Registration
    c. Banner Portal
    d. RabbitMQ
    e. EIS
    f. BEP
    g. Forms
    h. SSO
    i. t2202a

# Images, containers, and compose

1. Many environments
    a. BANPROD
    b. BPREPROD
    c. BQAWEEK
    d. BDEV03 - BDEV09
2. Haven’t scaled past BDEV09 due to inability to automate load balancer work
    a. This is future work

# Images, containers, and compose

1. Compose file for each image/environment pair
2. Templated for easier management
3. Tracked via git, mounted on every host

# Load balancer

1. We use an F5
2. Each service is represented by a virtual server
3. Virtual server balances to a pool containing multiple nodes
4. Each node is a single container running on a host
5. Typically, we have each service balancing across 2 nodes on 2 different hosts
6. Scale dynamically by starting additional containers on other hosts, and adding them to the pool

# Password management

1. Passwords are stored in secret server
2. Script queries secret server by secret id
3. Compose files have environment specific secret ids

# Handling changes in load

1. Many users start to hit a system, and overwhelm the available resources?
2. Use compose to start new containers for overwhelmed system component on additional docker hosts
3. Add new containers to load balancer virtual server pool
4. When peak usage goes down, stop new containers, and remove them from load balancer virtual server pool
5. Don’t have to plan for peak usage as we can adapt on the fly

# Handling patching - a work in progress

1. To patch a service, we update the Docker image, and tag the new version
2. Update the compose file with the new tag
3. Start new containers on available docker hosts
4. Update load balancer virtual server pool with new nodes
5. Stop load balancer from adding new connections to old nodes
6. When old nodes have no connections we can safely stop old containers
7. This is future work for us, not quite there yet

# Value Proposition

1. First Banner upgrade was a total disaster that nearly failed 4 times
2. Expensive, lots of overtime
3. Weak ability to rollback after a failure
4. Now Banner upgrades are reliable, and routine
5. Minor overtime on upgrade weekend for rollout, and testing
    a. Not really necessary, done as a convenience
6. Easy to rollback from failure, simply redeploy pre-update versions of everything

# Storage

1. Netapp Storage.
    a. We heavily use snapmanager for Oracle
    b. Daily snapshots/flexclones for development
    c. Weekly snapshots/flexclones for development
    d. Dedicated snapshots/flexclones for particular projects.
    e. Provides proof of backups working.

# Storage

We use NFS mounts for:

1. Common Docker compose files
2. Common files
    a. Moving away from this to dedicated volumes per container project
    b. To Netapp Docker Volume Plugin and individual volume mounts per container
        i. More secure because of less exposure to all hosts
        ii. Automatic mount of resources through docker compose

# Host OS

1. CentOS 7+ and some Oracle Linux
    a. weblogic docker containers use Oracle Linux for support & licensing
    b. Moving to CentOS as it is supported by Docker Datacenter
    c. Likely to be more consistent support as they are Dev for RHEL
2. Security updates are automatic other than Docker & IPTables

# Host OS

1. We use SELinux
    a. NFS doesn't allow for full implementation
2. Have enabled User Namespace to address container separation
    a. Will have to maintain uid mappings per Container Service
    b. Ensures root in container is not UID 0 on the host
    c. uid 0 mapped to uid 50000 within container

# Oracle

1. Expensive, but necessary for Ellucian
2. We are setting up for Per core licensing
3. OracleVM server and dedicated cores
    a. After looking at usage we found that we had more compute than needed.
    b. Allows for less expensive license versus a campus agreement.

# Puppet

1. We use puppet to build to provision the Docker servers
    a. Can provision a server in 20 minutes

# VMWare

1. We have been using VM's
    a. Many VM's are relatively small 20G ram/1 vCPU
    b. Entropy becomes a problem as java uses a lot of entropy
        i. makes for slow starts
2. Moving to recycled hardware
    a. Less dependence on shared infrastructure
    b. Better entropy
    c. Better scalability and DR

# All of this replaces what we used to use

1. Dedicated hardware for each service
    a. Meant fragile environments
    b. Wary of patching/changing
    c. Development on backup or production server
    d. Rebuilds took days/ weeks with testing

# How Docker Helps Infrastructure

1. Fewer full customized server rebuilds
2. Simpler server builds
3. Less package and program conflicts
4. Smaller, cheaper, recycled, outsourced hosts  can be used
    a. increasing redundancy
    b. Reducing costs
    c. Throwaway hosts means any machine could be used
    d. Devs can use their desktops to develop

# How Docker Helps Infrastructure

1. Developers can use OS & programs of their choice
    a. Not restricted by host OS
    b. Not restricted by repositories versions
2. Developers can patch, test, and build multiple times a day
3. Developers learn and can self supp

# How Docker Helps Infrastructure

1. Better security as containers can be rebuilt reliably and quickly
2. Contained services with minimal attack vector
    a. Only the service is exposed to a load balancer/director
    b. Nothing that isn't needed is installed (like ssh/ftp/mail/cups)
    c. Everything is firewalled to the outside world

# Perspectives from management

1. We have silos (Infrastructure, Development, Support, Security)
2. The nature of Docker naturally leads to cross-group teams that are:
    a. Project oriented
    b. Ephemeral
3. Communication patterns change, become more direct
4. Traditional roles have changed, responsibility shifted, but…
5. … there is still the need for specialization
6. Managers take note: Step back but stay informed
7. Own the project and remove obstacles

# Security: Workload Separation vs Integration

1. Containers: A middle ground between VMs and bare-metal processes
2. A move from VMs to containers results in a decrease in separation
3. But a move from process to containers increases separation
4. Should you care?
    a. Increasing separation increases security
    b. Decreasing separation increases agility
    c. Tension between these is a non-trivial issue, but not necessarily a problematic one
        i. Security team becomes involved in the design process

# Security: Perspectives

1. Virtual Machines:
    a. Super secure! Complete isolation between workloads (until there isn't)
    b. Contain a complete copy of the (an) OS for every instance
    c. Configurations tend to diverge
2. Containers:
    a. Hosts can be configured (nearly) identically, with little need for alteration/evolution
    b. Containers include the minimal set of software required for the job
    c. Shared kernel, memory, host server resources, filesystems

# Security: Design considerations

1. Kernel vulnerabilities can be very problematic
2. Container breakouts
    a. Kernel namespacing
    b. Drop privileges
3. Host resource exhaustion conflicts/DoS
    a. Control groups
4. Image maintenance and patching
    a. Orchestration/management system
5. Stale images
6. Host OS hardening
    a. AppArmor, SELinux

# Security: Operational considerations

1. Image providence:
    a. Who created the image that you want to use?
    b. Are you sure it's legitimate and hasn't been tampered with?
2. Workload separation:
    a. Consider separation of workloads of different trust levels
    b. However, this breaks homogeneity of environment
    c. Increases complexity and expense
    d. Decreases agility

# Future Work

1. Automate creation of load balancer profiles so that bringing up an environment can also create the necessary load balancer configuration
2. Figure out a better way to do storage, forms being wide open on 20+ hosts isn't a great idea
3. Automatic password cycling in running containers

# Questions

1. Warren Marusiak - Development
    a. Warren.Marusiak@unbc.ca
2. James Pohl - IT Infrastructure
    a. bofh@unbc.ca
3. Dave Kubert - IT Security
    a. Dave.Kubert@unbc.ca