T11
Virtualization Testing
Thursday, May 3rd, 2018
11:15 AM

Integration and Functional Testing Using Dockerized Dependencies

Presented by:

Irene Dhong

Bazaarvoice

Brought to you by:

350 Corporate Way, Suite 400, Orange Park, FL 32073
888-268-8770 ⋅ 904-278-0524 - info@techwell.com - http://www.stareast.techwell.com/

Irene Dhong

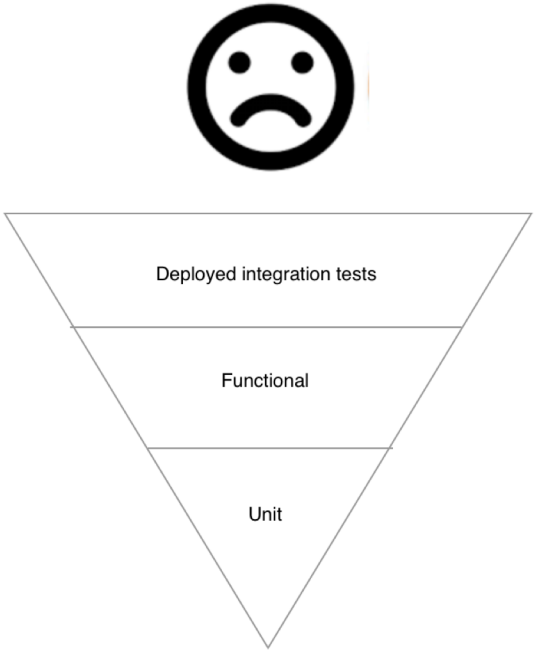
Bazaarvoice

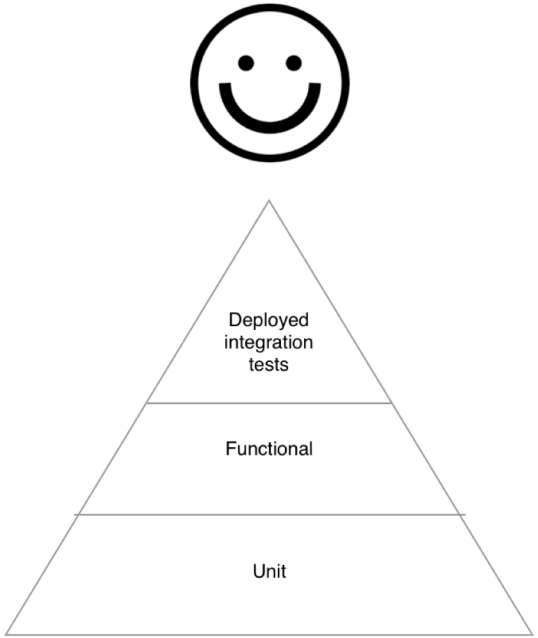
Irene Dhong has been a QA Engineer for over 6 years. She has experience as a build engineer and a system administrator. She's passionate about building frameworks and tools that maintain a high level of quality in lean, rapid development environments. She enjoys bringing quality to new challenges like machine learning, natural language processing, and complex microservice ecosystems. Irene loves to encourage community among women in technology and runs a women's leadership guest speaker series at Bazaarvoice.

Integration and Functional Testing Using Dockerized Dependencies

Irene Dhong

Ice Cream Cone





The Struggle

- High quality low defects
- Expensive
  - Long running tests
  - Extremely hard to add a new test
  - Debugging difficult or near impossible

How did we get here?

- Amount of mocking out needed
- Interactions hard to simulate
- Manual Custom ops code
- Dozens of dependencies
- Fast pace of changing code

My wish list

- Be able to write and run tests quickly
- Have all dependencies mocked out
- Try to catch quality issues before it even reaches ci
- Test on a very low level
- Write hermetic tests
- Have access to service/application/logs while developing
- Reuse code

Docker

- ***Docker*** *is a computer program that performs operating-system-level virtualization also known as containerization.* *wikipedia
- *Docker is a platform for developers and sysadmins to* ***develop, deploy, and run*** *applications with containers. The use of Linux containers to deploy applications is called containerization. Containers are not new, but their use for easily deploying applications is.* *Docker.com

Dockerized World

- Elements of our system are run in containers
- Links used to expose elements to each other
- Use official images for infrastructure

Supports different kinds of testing

- Unit-like
- Local Integration
  - Can be run locally or in ci
- Deployed integration tests in production or production-like environment

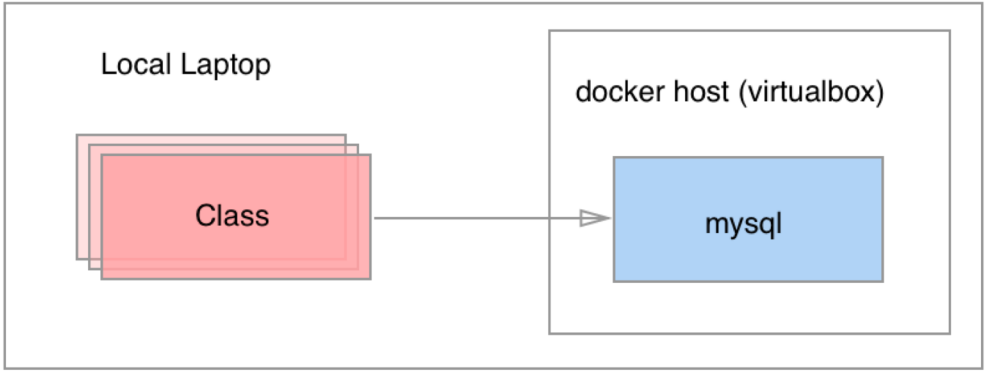
Unit like testing environment

@BeforeSuite

- Start dep

@Test

- Test function
- Verify data



A Hermetic functional test environment

@BeforeSuite

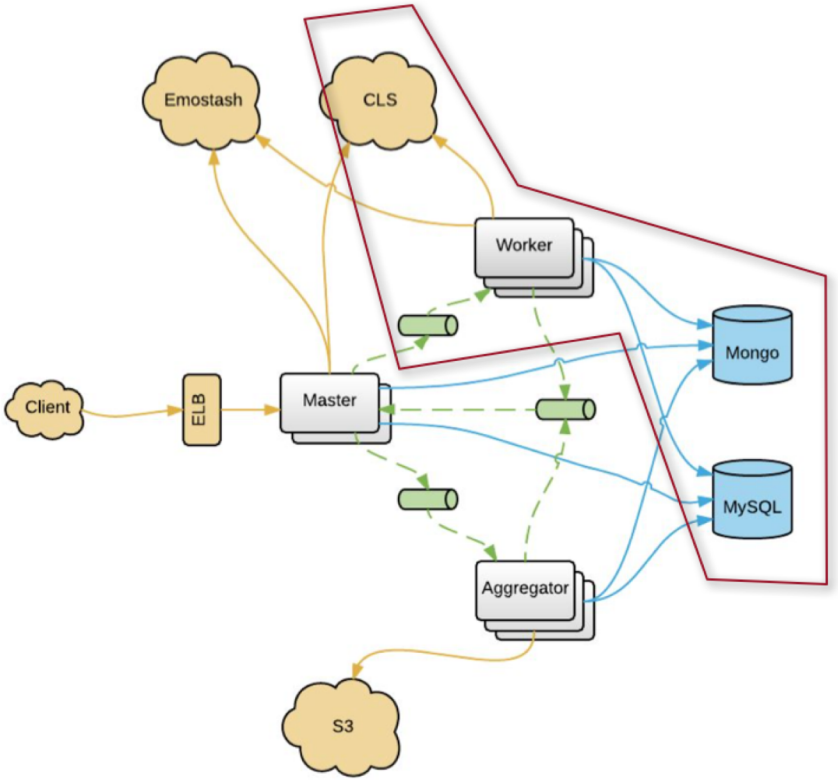
- Start dep and local app

@Test

- Setup, Start operation, Verify data

@AfterTest

- Clean up data



bazaarvoice

Local Integration Test Environment

@BeforeSuite

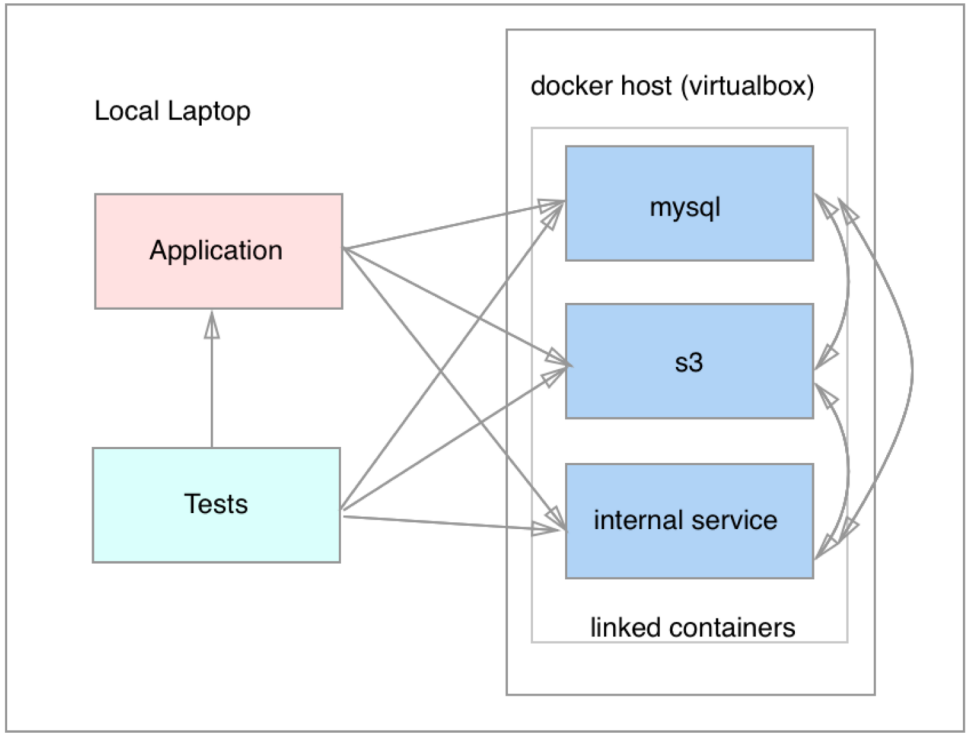
- Start dep and local app

@Test

- Setup data, Trigger app, Verify data

@AfterTest

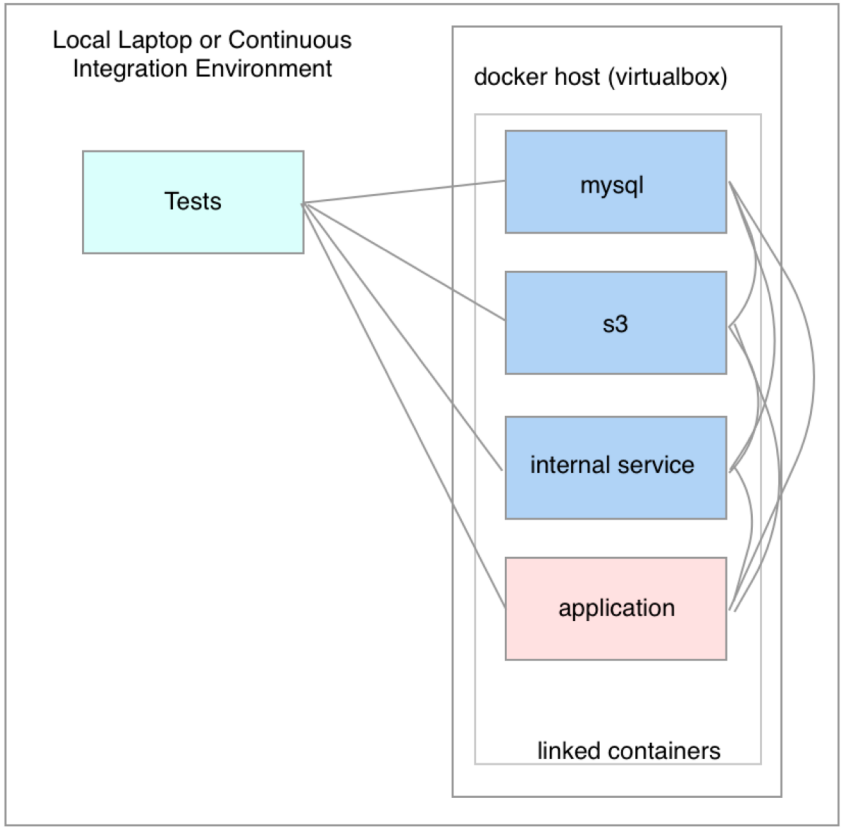
- Clean up



bazaarvoice

Local Integration Test Env or CI

- Start dockerized dependencies
- Start dockerized application
- Run tests



bazaarvoice

Integration tests run in deployed env

- Reuse test framework using config.yaml
- Omit startup, setup, and cleanup steps

```yaml
local:
  app_url: localhost
  app_port: 7003
  s3_host: fakes3
  s3_port: 4569
  s3_bucket: app
  s3_credentials_bucket: app-keys
  s3_credentials_file: private/keys.json
  mysql_url: mysql
  mysql_port: 3306
  cls_url: cls
  cls_port: 9080

deployedEnv:
  app_url: app.prod.us-east-1.nexus.bazaarvoice.com
  s3_host: s3
  s3_bucket: app
  s3_credentials_bucket: app-keys
  s3_credentials_file: private/keys.json
  mysql_url: 127.0.0.1
  mysql_port: 6666
  cls_url: cls-bazaar.prod.us-east-1.nexus.bazaarvoice.com
```

Issue: Hard to add data

- Simulate real life situation
- Reuse existing data loading capabilities
- Explore transitional states
- Insight to data
- Large datasets
- Clean databases
- Hermitic data

Issue: Huge amounts of mocking out

- One time cost
- Maintained through dockerfile
- Stubbed dockerized dependency only tests interactions that tests require
- As dependency changes, maintenance is little as can use POJO's

Issue: Manual custom ops code

- Database table creation in dockerfile
- Devops/dev updates table files in one file
- Same image used for dev, devops, and testing

```
FROM mysql:5.7.15

ENV MYSQL_DATABASE ucdm_export
COPY ucdm_export.sql /docker-entrypoint-initdb.d/1.sql

EXPOSE 3306
EXPOSE 3406
CMD ["mysqld"]
```

Issue: Disaster Recovery/Perf Test

- Redeploy application pointing to dockerized dependency
- Hybridization

Issue: Debugging in Deployed Environment

- Advantages over everything running locally
- Advantages over deployed environment

Challenge: No official docker image

- S3, sqs
- FakeS3
- Redis

Challenge: Impossible to dockerize

- Stub it and dockerize it!
- Quick flask server
- Don’t hardcode, make stub interactive
- Risk analysis

```
FROM python:2.7.10

COPY . /usr/local/cms2stub/

WORKDIR /usr/local/cms2stub/

RUN pip install -r requirements.txt

EXPOSE 5000

CMD ["python", "cms2_stub.py"]
```

Challenge: Security

- Keys
- Mimic

Challenge: Development changes code

- Update shared pojos, classes, and tests
- Build latest image with latest table scripts or changes
- PR build which runs all local tests

Challenge: Multitude of technologies

- Mvn, gradle, sbt
- Python, Java, Javascript, Scala

Challenge: Multitude of technologies

- Use common scripts

```
#!/bin/bash

# Run infrastructure
echo "Bring up mysql container"
docker run -d --name mysql \
           -p 3306:3306 \
           12345.dkr.ecr.us-east-1.amazonaws.com/data-services/mumford:mysql-5.6

echo "Bring up fakes3 container"
docker run -d --name mumford-submission.fakes3 \
              -p 4569:4569 \
              12345.dkr.ecr.us-east-1.amazonaws.com/data-services/common:fakes3-0.2.4
```

Testing specific cons

- Investment of time
- Mocked/Stubbed components
- Isolated from the bigger picture/world
- Too close to the code

Pros generally

- Offline development
- Containers are isolated so don’t have to pollute machine with various packages and system services
- Dev use same infrastructure for unit like dependency testing
- Devops use same image for deployment with different config
- Reuse of code in all aspects

Testing specific pros