CONTINUOUS DELIVERY WITH DOCKER CONTAINERS AND JAVA EE

Markus Eisele
markus@jboss.org

@myfear
blog.eisele.net
THE WORLD WE LIVE IN TODAY

Customers and consumers
• Ubiquitous access to data and services
• Impatient, want everything NOW
• Increased QoS expectations

Businesses
• New opportunities and markets
• Threat of being disrupted, intense competition
• Small time frames to get products and services out
TRADITIONAL SOFTWARE DELIVERY ENVIRONMENT
TYPICAL ASSUMPTIONS AND EXPECTATIONS

- Software should never break.
- Ops teams are not required in application design discussions.
- Production environments are provisioned/through a mostly manual process.
- Developers should not have any access to the production environment.
- You have to give a lot of lead time for getting an application environment.
- An application is deployed to production after all development is complete.
- Deployments are a headache—software is deployed using a mostly manual process.
- We cannot keep deploying code to production on a regular basis.
Developers should focus on writing code.
Continuous Delivery with Docker and Java EE

Quality engineers should focus on testing.

**DEV**

1. Request a VM
2. Request a middleware platform
3. Set up the environment for the application
4. Configure build scripts
5. Write code for application
6. Integration test
7. Deploy to Share Dev
8. Unit test

**TEST**

1. Request a VM
2. Request a middleware platform
3. Set up the environment for the application
4. Build and deploy application
5. Integration testing
   - Regression testing
   - Functional testing
   - Non-functional testing
   - Performance testing
   - Stress testing
   - Manual testing
6. Sign off to deploy application

**OPS**

1. Request a VM
2. Request a middleware platform
3. Set up the environment for the application
4. Build and deploy application
5. Get application fix
6. Root cause analysis
7. Production issue
8. Monitor the application
9. Smoke test
Ops engineers should focus on providing reliable and stable environments.
BUT THIS IS SOOOOOOOOOOOOOOOOOO YESTERDAY.
CONTINUOUS DELIVERY

• Continuous Integration
• Fail fast and recover
• Self service
• 100% Automation
• Push to Prod
• Proactive Monitoring and Metrics
• Requires fast and Consistent Build and Deploy
TRADITIONAL SILOS
BREAKING THEM DOWN (THE MONOLITHIC WAY)
BREAKING THEM DOWN (THE MICROSERVICE WAY)
TRADITIONAL ARCHITECTURE
SCALING == SCALING THE COMPLETE STACK

Red Hat JBoss EAP
TOMORROWS APPROACH (MICROSERVICES)
PACKAGE ONCE RUN EVERYWHERE (CONTAINER)

Physical | Virtual | Private | Public
SCALING SERVICES ON DEMAND

Physical | Virtual | Private | Public
PYRAMID OF MODERN APPLICATION DEVELOPMENT
WHAT’S IN IT FOR JAVA EE DEVELOPERS?

• Developers can focus on the app
• Standardized Development Environment
• Spin-up and Tear-Down of different instances in seconds.
• Easy, centralized configuration
• “Build, Ship and Run: Any App, Anywhere”
• Faster and more predictable delivery
CONTAINERS, WHAT’S SO GREAT?

• Put everything in a box
• It’s cheap (compared to special shipments)
• I don’t have to care if they are delivered on a ship, a truck or a train
• I can put a look on it and make sure that my integrity is kept
• If a container breaks replace it
CONTAINERS, WHAT ARE THE DRAWBACKS?

• One size doesn’t fit all.
• I hope no one drops my container.
• No one is going to care if I put a sign on it saying fragile.
• I cannot change the routing while in process.
• I’m not in control.
CONTAINER DEPLOYMENT

• Many organizations today are not ready to directly deploy the same containers in all environments
• “Build, Ship and Run Any App, Anywhere” (Docker Slogan)
• Requires adopting DevOps principles and Microservices architecture
• Today typical Java EE application still requires small variations between environments.
• E.g. Memory usage, clustering, endpoints, security patches etc
• The key is to minimize the variations.
CAN CONTAINERS BE ABUSED?

• Containers are immutable
• Release vs Patch
• If it break, replace it. Do not try to fix them.
• Complex environment gets complex to maintain even with containers
• Use a orchestration tool like OpenShift
HOW TO DO IT?
JAVA EE DEVELOPMENT WORKFLOW
SOLUTION ONE: DEVELOPMENT ENVIRONMENT
SOLUTION TWO: CI ENVIRONMENT
SOLUTION THREE: CLOUD DEV ENVIRONMENT
SOLUTION FOUR: CLOUD DEVOPS ENVIRONMENT
<table>
<thead>
<tr>
<th></th>
<th>Developer Experience</th>
<th>Ops User Experience</th>
<th>Cluster Management</th>
<th>Container &amp; Registry</th>
<th>Container Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>fabric8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kubernetes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>atomic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>docker</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DEVELOPER UI
MORE RESOURCES AND READINGS

• Fabric8 Guide

• Running WildFly on Fabric8 / OpenShift
  http://blog.eisele.net/2015/07/running-wildfly-on-openshift-3-with-kubernetes-fabric8-on-windows.html

• Handy Resources for WildFly on Docker
  http://blog.eisele.net/2015/01/java-ee-docker-wildfly-and-microservices-on-docker.html

Follow:
- @myfear
- @jbossdeveloper
- @fabric8io
http://developers.redhat.com/promotions/distributed-javaee-architecture/