Narrative – Docker abCI

Strategy to pursue

- build using mvn
- build wls-jaxRS-maven image from wls-mydomain image
- launch wls-jaxRS-maven docker container
- run soapui test

Slide 2 – Application Migration

- We all know the drill
  - Dev
  - Test
  - QA
  - Prod
- There's usually a deployment step in between environments
  - And it's usually involved
  - Easy to make mistakes
  - Misconfigure
- It's difficult to automate integration testing
  - Need dedicated servers
  - Expensive
  - Neglected
- When we test, we're not testing how it's going to be run
  - We spend a lot of time mocking
    - Then we have to test the mocks
    - Mocks don't tend to be kept up over time
      - Especially in a corporate environment
  - Sometimes it's the interaction with the container that matters
    - Tomcat
    - WebLogic
    - WebSphere
    - Jboss/Wildfly

Slide 3 – Docker to the rescue?

- Docker potentially solves the migration problem
  - At least you know you're running the same bits everywhere
  - There's still a configuration issue, though

        - Typically the configuration changes through the environments
            - Passwords
            - Databases
            - Message connectors
            - This session isn't about solving that problem – though
    - There's the issue of testing
        - How do we test during our build?
            - I want my component running in it's container
            - I want isolation
            - I don't want dedicated infrastructure I have to maintain

Slide 4 – Maven & SoapUI

- Maven
    - Maven is the heart of the JEE build system
    - It would have been easy to rely on the shell scripting provided by Hudson or Jenkins
    - Likewise it would have been easy to rely on what's essentially shell scripting by Ant
    - But these are hacks
    - Maven has an extensible plug-in system
    - Besides, Maven was designed to do more than build: it builds, packages, unit tests, integration tests, installs and deploys
    - Maven should take care of everything and then you can plug Maven into your abCI server
        - abCI server monitors SCM for changes
        - Checks out code
        - Builds & tests using Maven
        - Drives the build dashboard and notifications
        - Don't make your abCI server get involved with the actual build or testing
            - Separation of concerns
        - Don't use Ant from inside Maven
            - I see the ant-maven-plugin as being used to support legacy build systems
            - Not to be used for new projects
- SoapUI
    - JEE developers have been using SoapUI for years to test SOAP web services
    - SoapUI can do so much more!
        - REST web service testing
        - MDB testing using HermesJMS

        - Load testing
    - You can also make test suites
    - Better still – you can run these SoapUI test suites from within Maven!

Slide 5 – layering images

- You don't create monolithic images – each image adds a specific thing
- taylodl/centos
    - Base image
    - glibc, libs and shared libs
        - libcontainer – which we'll find useful in a moment
    - yum
    - tools useful for developing and debugging images
    - Goal was not to make the smallest image possible, ala busybox
        - I'm not running hundreds of images on a server
        - I'm running dozens
- taylodl/jdk
    - Layers on top of taylodl/centos
    - Complete JDK environment appropriately configured
        - JAVA_HOME, $JAVA_HOME/bin added to system path
- taylodl/wls
    - Layered on top of taylodl/jdk
    - WebLogic 12.1.2 (12c)
    - Does not include any domains
- taylodl/wls-mydomain
    - Layered on top of taylodl/wls
    - This is an actual domain – something you can run
        - By separating the domain from WebLogic we can apply the same domain to a different version of WebLogic
- taylodl/wls-jaxRS-maven
    - This is our project – this is what we're going to build
    - And we're layering it on top of taylodl/wls-mydomain

Slide 6 – taylodl/maven

- Docker is about containerization and what better way to demonstrate containerization than to run Maven inside a container?
- No installation required on your part
- Layered on top of taylodl/jdk
    - Which of course is layered on top of taylodl/centos
- Nothing to install! Have Docker will travel!

- taylodl/maven has everything you need in it's local Maven repository
- Though not being done here, using Docker's volume management tools is a great way to manage Maven repositories
  - Potentially less cumbersome than Nexus

Slide 7 – Docker in Docker

- taylodl/maven is a Docker image that will be run in a Docker container
- We're going to be using Maven to build and run a Docker container
- That means we need to be able to use Docker within Docker!
- I'll show you in a minute how to do it, but to pull it off the libcontainer library needs to be installed in your image
  - taylodl/centos has you covered
- Demo – running Docker in Docker
  - First show how we can't run Docker
    - docker run -i -t –net=host -v /home/developer/docker/maven-projects:/maven-projects taylodl/maven
      - docker run images
      - bash: docker: command not found
  - Now show how we can run Docker in Docker
    - docker run -i -t –net=host -v /home/developer/docker/maven-projects:/maven-projects -v /usr/bin/docker:/usr/bin/docker -v /var/run/docker.sock:/var/run/docker.sock
      - This may not work in anything other than Centos 6.5
      - We'll have a workaround soon, but first a word from our sponsors...
      - If it doesn't work don't sweat it...we'll get to it next
    - If it works then let's run wls-mydomain
      - docker run -p 7001:7001 taylodl/wls-mydomain &
      - Access http://localhost:7001/console from browser

Slide 8 – Docker ports

- The docker-maven-plugin we're going to be using requires HTTP access to Docker
- Typically you haven't set that up – your Docker client has been accessing the daemon using /var/run/docker.sock
- We need to stop the docker daemon and restart it to allow HTTP access
  - sudo service docker stop
  - sudo docker -H tcp://127.0.0.1:2375 -H unix:///var/run/docker.sock -d &
    - Docker 1.3 also has port 2376 registered with IANA for https
    - This would be the prefered method

    - docker images (just to show Docker is running properly)
- If we failed to demo running Docker in Docker during slide 7 then now is the time to demo it
    - docker run -i -t –net=host -e DOCKER_HOST=tcp://127.0.0.1:2375 -v /home/developer/docker/maven-projects:/maven-projects -v /usr/bin/docker:/usr/bin/docker taylodl/maven
        - Notice we had to specify a DOCKER_HOST environment variable
            - This is needed for the Docker client
            - The docker-maven-plugin defaults to this entry if the environment variable is missing
        - Notice there's no point mapping the volume /var/run/docker.sock
        - Let's run wls-mydomain
            - docker run -p 7001:7001 taylodl/wls-mydomain &
        - Access http://localhost:7001/console from browser

slide 9 – NetBeans project

- Open up NetBeans locally
- Look at HelloWorld.java
    - Look at service
- Look at Person.java
    - Notice it's a simple POJO
    - Technically a bean, but super simple
- It's the application/json annotation on HelloWorld's getText that determines Person.java is being returned as json
    - But we're not here to talk about creating JEE projects
    - We just have a simple project to use

slide 10 – which came first?

- The easiest way to create a SoapUI project is to create and deploy our project and then run SoapUI against that
- Of course that means we can't start off with SoapUI testing
    - We have to add it in later
    - It's all part of iterative development
    - Create our project without unit testing, run the image and then run SoapUI against the image
    - Then we can add in the SoapUI testing

Slide 11 – docker-maven-plugin

- First step is to get our project “containerized” so we have a Docker image we can run

- Once we have a running Docker image we can creeate a SoapUI project which we can use for testing
- Demo
  - cd ~/docker/maven-projects/jaxRS-maven
  - emacs pom-docker.xml
    - Checkout the properties near the top of the file
      - wls.mydomain
      - wls.mydomain.autodeploy
      - wls.mydomain.port
        - This is the port we want to forward through on the container
        - The actual port taylodl/wls-mydomain exposes is always 7001
    - The properties were created for my convenience
    - Let's scroll down to the docker-maven-plugin
      - Plugin identifiers:
        - groupId: org.jolokia
        - artifactId: docker-maven-plugin
        - version: 0.10.5
      - dockerHost: http://127.0.0.1:2375
        - This is supposed to be the default, but I prefer making things explicit
      - Scroll down a little further to look at the executions
        - One run during pre-integration-test
          - goals: stop, start
        - The other run during post-integration-test
          - goals: stop
      - Now let's inspect the image
        - the image name is taylodl/wls-jaxRS-maven
        - build section – used to build the image
          - from: taylodl/wls-mydomain
            - layering it on top of taylodl/wls-mydomain
          - exportDir: the directory within the container to which we're to export the artifact(s)
          - assemblyDescriptorRef: artifact
            - artifact-with-dependencies
            - artifact – good for WebLogic autodeploy
            - project – good for open-directory deployment
            - rootWar – good for Tomcat
          - port: the port to be exposed from the container
            - always going to be 7001 for WebLogic

        - run section – used to run the image
            - ports:port – the IP port forwarding
            - wait
                - url: URL to hit to determine the container is up and running properly
                - time: time in milliseconds to wait for the container to come up
    - docker run -i -t –net=host -e DOCKER_HOST=tcp://127.0.0.1:2375 -v /home/developer/docker/maven-projects:/maven-projects -v /usr/bin/local/docker:/usr/bin/local taylodl/maven
    - cd /maven-projects/jaxRS-maven
    - cp pom-docker.xml pom.xml
    - mvn clean integration-test
    - docker ps
        - notice taylodl/wls-jaxRS-maven is running
        - that's because we ran up to integration-test, post-integration-test which would have stopped the container was not run!
    - curl http://localhost:7001/jaxRS-maven-1.0-SNAPSHOT/resources/helloworld/don/taylor
        - we're running against the Docker container running the image we just created using Maven!
    - exit
    - ifconfig
        - get the IP address of the virtual machine so we can hit it with SoapUI

Slide 12 – Soap UI

- At this point we've built our taylodl/wls-jaxRS-maven and have confirmed it's running
- Now we need to use SoapUI to create our test project
- Launch SoapUI on the Mac
- Create new REST project
    - http://<ip address of virtual machine>:7001/jaxRS-maven-1.0-SNAPSHOT/resources/helloworld/{firstName}/{lastName}
        - Note the whole /{firstName}/{lastName} syntax bit!
        - That'll save you a whole lot of time!
            - Already defines and correctly configures all your parameters
    - Select JSON and click green 'Go' icon
        - You should have your result
- Right-click hourglass icon (http://...) and select 'Generate TestSuite'
    - Note we can generate load tests too, if we'd like
    - Click Ok and then name the suite whatever you want

- You may wonder how this is going to work when we run the soap-ui-maven plugin
  - The IP address we used may change
  - That's OK, we can fix that in the plugin's configuration!
- Now right-click project and select Save As...
  - Save in the maven-project directory as jaxRS-maven-soapui-project.xm

Slide 13 – SoapUI & Maven

- Now that we have a SoapUI TestSuite created we need to run it during integration test
- Demo
  - cd ~/docker/maven-projects/jaxRS-maven
  - emacs pom-soapui.xml
    - scroll down to the soapui-maven-plugin
      - Plugin identifiers:
        - groupId: com.smartbear.soapui
        - artifactId: soapui-maven-plugin
        - version: 4.6.1
    - Scroll down a little further to look at the executions
      - Run during integration-test
        - goals: test
    - Now let's look at the plugin configuration:
      - testSuite: name of testSuite to run
      - testCase: name of testCase to run
      - host: allows us to override the host
        - remember when we were worried about the IP address the test suite was using? This is how we override it!
  - docker run -i -t –net=host -e DOCKER_HOST=tcp://127.0.0.1:2375 -v /home/developer/docker/maven-projects:/maven-projects -v /usr/bin/local/docker:/usr/bin/local taylodl/maven
  - cd /maven-projects/jaxRS-maven
  - cp pom-soapui.xml pom.xml
  - mvn clean install
  - The build may fail
    - docker-maven-plugin:stop doesn't always give the container enough time to stop
    - Author is aware of this (this is a GitHub project), fix is pending
    - The best option for now is to not stop the container
    - We never noticed before because we never executed the post-integration-

        test phase, we only executed up to the integration-test phase

- Still, we see the SoapUI testing passed
    - And that's what we wanted
- docker images
    - and we see taylodl/jaxRS-maven has been created