Lightweight PaaS for Jenkins CI Environments with Docker

Josef Fuchshuber
QAware GmbH
http://www.qaware.de

June 25, 2014
About me

Job:
– Senior Software Architect (JEE)
– Manager of software development tools

Company: QAware GmbH, Munich
– Custom software company
– Employees: 53
– Customers:
  • Telecommunication industry
  • Automotive industry
Agenda

• Introduction to Docker

• Dynamic Build Slaves with Docker

• Lightweight PaaS: Continuous Deployment with Jenkins and Docker
“Containerization is the new virtualization”
Definition of QAware’s „TI architecture“

- JAR/WAR/EAR
- (Cron) jobs
- ...

- OS
- VM (Java, .NET, …)
- Server (db, web, …)
- Libraries
- ...

- Server
- Memory
- Network equipment
- ...

Application packages

System software

Hardware

Remote protocols
The Challenge
The Docker solution
Quick Facts: 🐳 docker

• Docker Inc. (formerly dotCloud Inc.)
• Introduced in March 2013
• Version 1.0 released on June 4 2014
• Huge Github community (13,000 stars and counting)
• Open source: Apache v2 licence
Why should we use Docker?

- It makes it easy to create lightweight, portable, and self-sufficient containers.
- Configure once, run anywhere
- Solving dependency hell
- Huge win for automation and deployments
- Containers are perfect for:
  - Continuous integration & test applications
  - Build services
  - Run services
  - Build your own Platform-as-a-Service (PaaS)
Some Docker vocabulary

- Image
- Registry
- Container
- Layer
- Dockerfile
- Linking
Insights: Under the hood

**Union file systems (UnionFS)**
- Union file systems operate by creating layers.
- They combine layers into a single image.
- Supports: AUFS, btrfs, vfs and DeviceMapper

**Namespaces & control groups**
- Provide isolated workspace for containers.
- Controlling resource (CPU, memory, block I/O, network, etc.)

**Container formats**
- Wraps these all together:
  - Libcontainer
  - LXC
  - …
VMs vs. Containers

VMs:
- Hypervisor 1
- Host OS
- Hardware
- Type 1 / Type 2 Virtualization

Containers:
- Hypervisor 2
- Host OS
- Hardware
- Docker
- Containerization

Containers are lightweight compared to virtual machines. They share the same host OS, reducing the overhead and making them more efficient for development and deployment.
A self-sufficient runtime for Linux containers.

Commands:
- `attach` Attach to a running container
- `build` Build an image from a Dockerfile
- `commit` Create a new image from a container’s changes
- `cp` Copy files/folders from the container’s filesystem to the host path
- `diff` Inspect changes on a container’s filesystem
- `events` Get real-time events from the server
- `export` Stream the contents of a container as a tar archive
- `history` Show the history of an image
- `images` List images
- `import` Create a new filesystem image from the contents of a tarball
- `info` Display system-wide information
- `inspect` Return low-level information on a container
- `kill` Kill a running container
- `load` Load an image from a tar archive
- `login` Register or login to the docker registry server
- `logs` Fetch the logs of a container
- `port` Lookup the public-facing port which is NAT-ed to PRIVATE_PORT
- `pause` Pause all processes within a container
- `ps` List containers
- `pull` Pull an image or a repository from the docker registry server
- `push` Push an image or a repository to the docker registry server
- `restart` Restart a running container
- `rm` Remove one or more containers
- `rmi` Remove one or more images
- `run` Run a command in a new container
- `save` Save an image to a tar archive
- `search` Search for an image in the docker index
- `start` Start a stopped container
- `stop` Stop a running container
- `tag` Tag an image into a repository
- `top` Lookup the running processes of a container
- `unpause` Unpause a paused container
- `version` Show the docker version information
- `wait` Block until a container stops, then print its exit code
Agenda

• Introduction to Docker

• Dynamic Build Slaves with Docker

• Lightweight PaaS: Continuous Deployment with Jenkins and Docker
Example of a Jenkins infrastructure

- Jenkins (Master)
- SVN/Git
- Repositories (Maven, RPM, ...)
- SonarQube
- Jenkins Slave (Ubuntu Linux)
- Jenkins Slave (CentOS Linux)
- Jenkins Slave (Windows Server 2012)
- Jenkins Slave (Mac OS X Server)
What Docker changes

- Jenkins (Master)
- SVN/Git
- Repositories (Maven, RPM, ...)
- SonarQube
- Docker Host
  - Jenkins Slave (Ubuntu Linux)
  - Jenkins Slave (CentOS Linux)
  - Jenkins Slave (Windows Server 2012)
  - Jenkins Slave (Mac OS X Server)
- Docker Repository
The idea: The Docker build workflow

1. Developers look for base images in public/private Docker repositories
2. Optional: Customize it and push it back to the Docker repository
3. Jenkins administrator defines images as Jenkins slave
4. Define Jenkins job(s) for this Docker slave.
5. Run Jenkins job:
   1. Jenkins starts defined image as Docker container
   2. Jenkins runs the job inside the container
   3. Jenkins stops the container
Quick Facts: Docker Plugin

• Links:
  – [https://github.com/jenkinsci/docker-plugin](https://github.com/jenkinsci/docker-plugin)

• Features:
  – Dynamic provisioning of a Jenkins slave on a Docker host
  – Run a single build job
  – Tear-down the slave
  – Commit the container
Requirements for the container

• Connectable: SSH server
• Accessible: User (e.g. „jenkins“)
• Runnable: Java JDK

- Documentation: [https://wiki.jenkins-ci.org/display/JENKINS/Docker+Plugin](https://wiki.jenkins-ci.org/display/JENKINS/Docker+Plugin)
- Ready-made jenkins slave: „evarga/jenkins-slave“
## DEMO

<table>
<thead>
<tr>
<th>Configuration</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cloud</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Docker</strong></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>docker-host-1</td>
</tr>
<tr>
<td>Docker URL</td>
<td><a href="http://localhost:4243">http://localhost:4243</a></td>
</tr>
<tr>
<td>Connection Timeout</td>
<td>5</td>
</tr>
<tr>
<td>Read Timeout</td>
<td>15</td>
</tr>
<tr>
<td><strong>Images</strong></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>jenkins-1</td>
</tr>
<tr>
<td>Labels</td>
<td>dyn-node-kiga</td>
</tr>
<tr>
<td>Credentials</td>
<td>jenkins (Docker Build Node Login)</td>
</tr>
<tr>
<td>Remote Filing System Root</td>
<td>/home/jenkins</td>
</tr>
<tr>
<td>Instance Cap</td>
<td></td>
</tr>
<tr>
<td>DNS</td>
<td>0.0.0.0</td>
</tr>
<tr>
<td>Hostname</td>
<td></td>
</tr>
</tbody>
</table>
Why use dynamic slaves with Docker

• Fast startup
• Every job runs in its own clear container
• Job-parallelization is no problem
• Lazy resource binding and no long-running processes
• DevOps: Separation of concerns
  – Developer: Worries about the container's inside
  – Ops: Worries about the Jenkins configuration
Agenda

• Introduction to Docker

• Dynamic Build Slaves with Docker

• Lightweight PaaS: Continuous Deployment with Jenkins and Docker
Why to use an own lightweight PaaS solution in your CI/CD process?

• You can‘t use a commercial PaaS.

• You have a complex TI architecture.

• Integrate early and often! We need an scalable automated build and deployment process.
Lightweight PaaS components

Packaged application

Jenkins

Lightweight Docker PaaS Solution

Docker base images

Docker daemon
Docker makes „Infrastructure as Code“ easy

- Application Provisioning
- Server Provisioning
- Bootstrapping
- Bare Metal Provisioning
- Hardware

Diagram shows the relationships and dependencies between different image layers, such as base-image, base-app-service-image, base-app-db-image, java-image, mysql-image, app-server-image, app-service-image, and app-db-image.
The Jenkins build-pipeline for Docker deployments

- Build repo
- Artifact repo
- Docker repo

- Compile, test & package
- Create app packages
- Provision docker images
- Run integration-test
- Deploy & run test env.

Docker file(s)
Quick Facts: Docker build publish Plugin

• Links:
  – https://wiki.jenkins-ci.org/display/JENKINS/Docker+build+publish+Plugin
  – https://github.com/jenkinsci/docker-build-publish-plugin

• Features:
  – Build your project from a Dockerfile
  – Build and tag the image
  – Publish to private or public Docker repository
Quick Facts: Docker build step plugin

• Links:

• Features:
  – Execute Docker commands into your job as a build step
  – Export build variables
    • `DOCKER_CONTAINER_IDS` – The IDs of created/started containers
    • `DOCKER_IP_$HOSTNAME` – The IP of running container with hostname `$HOSTNAME`
Thank You To Our Sponsors

Platinum

Gold

Silver

Corporate

Community