Automating the Build Pipeline for Docker Container

Nikolai Reed, Jürgen Walter, and Samuel Kounev

University of Würzburg

November 9-10, 2017
SSP 2017 Karlsruhe, Germany
The lack of reproducibility of scientific experiments is a huge problem in research [5]

Scientific papers often lack a detailed description on how to apply the research software [5, 2]

Many software projects in public repositories cannot be built or installed within a hour [2]
Problem: Traditional deployment of software can be cumbersome due to compilation, required adaptations, dependency resolution, and lack of developer knowledge [2, 3, 4]

Problem: Classical virtualization using virtual machines leads to heavy weight containers [1]

Novel container technologies like Docker
- provide a reproducible environment to run software
- run on every machine the same
- allow for a fast deployments
- provide smaller containers compared to VM images
In a Nutshell

- This talk motivates and explains the integration of Docker into an automated build and delivery pipeline
- The SSP community provides more containerized services
- The SSP community solves the reproducibility problem through containerization
Introduction to Docker
Creation of Docker Containers

- Docker containers include dependencies required to run the software
- Required software packages are integrated into the Docker image during its build process
- Required information is contained in the so called *DockerFile* [3]
Creation of Docker Containers

- **DockerFile example for the apache webserver**

```bash
# A basic apache server. To use either add or bind mount content under /var/www
FROM ubuntu:12.04

RUN apt-get update &&
    apt-get install -y apache2 &&
    apt-get clean &&
    rm -rf /var/lib/apt/lists/*

ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2

EXPOSE 80

CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]
```
Creation of Docker Containers

- **DockerFile** for our performance model extraction software

  # Pull base image
  FROM openjdk:8u111-jre

  # Expose port of the Docker container
  EXPOSE 8080

  # Define working directory
  WORKDIR /opt

  # Add Software and server
  ADD pmxConsole.jar /opt/data/
  ADD target/pmxserver-0.0.1-SNAPSHOT.jar /opt/

  # Create directories during build process
  RUN \
      mkdir /opt/input && \
      mkdir /opt/zip && \
      mkdir /opt/download && \
      mkdir /opt/output && \
      mkdir /opt/uploaded

  # Start command to run wenn container is started
  ENTRYPOINT ["java", "-jar", "/opt/pmxserver-0.0.1-SNAPSHOT.jar"]
Docker Commands

- Creation of Docker container
  - `docker build -t descartesresearch/pmx-dml-server .`

- Running a Docker container
  - `docker run -d -p 8080:8080 descartesresearch/pmx-dml-server`

- Pulling a Docker container
  - `docker pull descartesresearch/pmx-dml-server`
The combination of Docker and CI allows to join their benefits

Benefits of an automated Docker build Pipeline
- fast time to production
- low overhead for developers and operators
- fast distribution and deployments
Automated Pipeline Process

- Push to Git (Code Repository)
- Pull from Git
- Notify (Failure/success)
- Pull from Jenkins
  - Fetch changes
  - Compile code
  - Test program
  - Build Docker (NEW)
- Push to Docker (Docker Repository)

NEW compared to non-dockerized CI
- We already had a running Jenkins CI-server setup available that we could expand upon.

<table>
<thead>
<tr>
<th>Component Type</th>
<th>Component Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version Control Server</td>
<td>GitLab (Git repository hosting service)</td>
</tr>
<tr>
<td>Docker Engine</td>
<td>standard (running on a Linux 64-bit system with Jenkins)</td>
</tr>
<tr>
<td>CI Server</td>
<td>Jenkins (an open source automation CI-server)</td>
</tr>
<tr>
<td>Docker build software</td>
<td>GitLab plugin (interaction with GitLab)</td>
</tr>
<tr>
<td></td>
<td>CloudBees plugin (build and publish of docker images)</td>
</tr>
</tbody>
</table>
Alternatives for Automated Pipeline Setup

- **Plugin**
  - Travis: native Docker integration
  - Team Foundation Server (Windows): Docker plugin available
- **Command line scripts**
  - Writing your own scripts based on the Docker file structure
- **External service**
  - GitLab already offers a CI service that is capable of building Docker images
  - DockerHub can be configured to build images from repositories of BitBucket and GitHub
We apply the presented build pipeline for

- The performance model extraction tool **Performance Model eXtractor (PMX)** [6]

- The **Pet Supply Store**, a micro-service reference test application for model extraction, cloud management, energy efficiency, power prediction, multi-tier auto-scaling

Further ideas

- **Simulation as a Service**
- **Performance evaluation as a Service**
- …
Conclusion

- Docker can be applied to solve the reproducibility problem.
- The combination of Docker and CI allows to join their benefits.
- Their combination enables fast time to production with low overhead for operators and developers.
Thank You!

Nikolai Reed, Jürgen Walter, and Samuel Kounev

University of Würzburg

November 9-10, 2017
SSP 2017 Karlsruhe, Germany 2017
References


