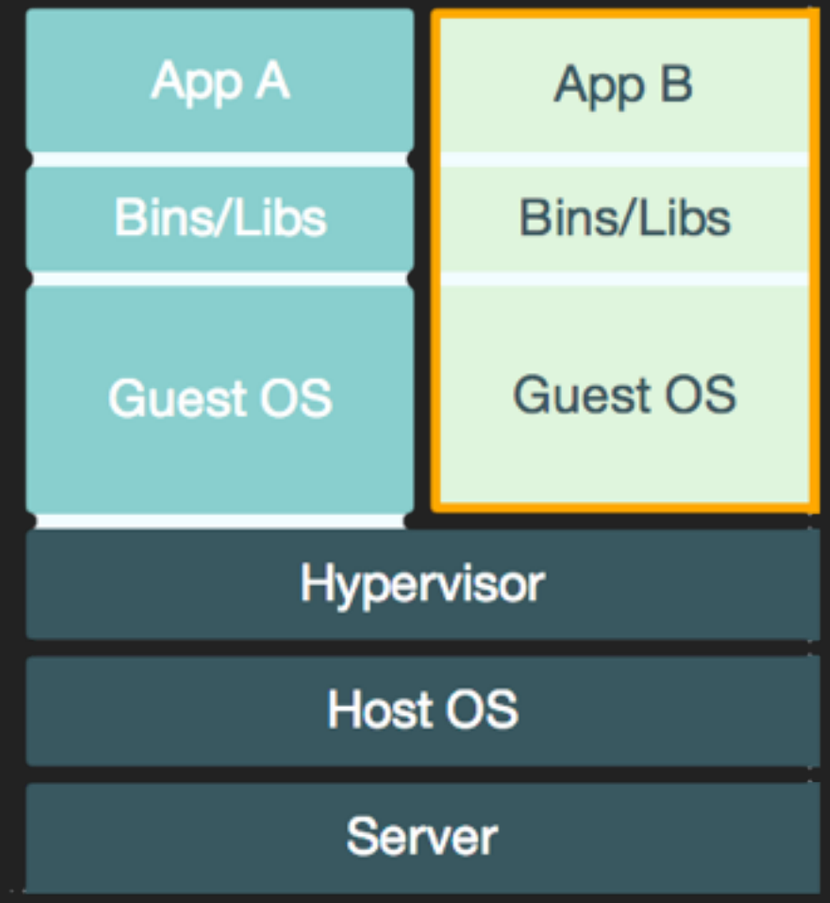
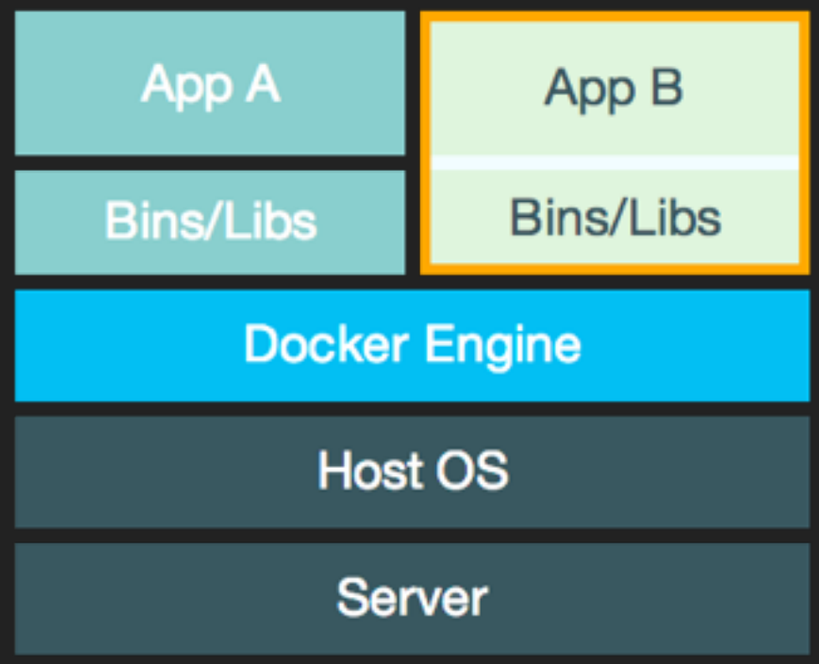
JIRI SEDLACEK

DEVELOPMENT AND DEPLOYMENT SIMPLIFICATION WITH CONTAINERS

WHAT IS DOCKER

- One possible way how to package and ship applications with all dependencies
- Lightweight virtualization - no OS emulation





DOCKER ATTRIBUTES

- faster spin-up times compared to VMs, low memory footprint
- incremental images
- private/public image registry
- one-command deployments
- the workflow is completely different (and not always simpler)

BACKGROUND OF OUR CASE

- No rocket science
  - we developed RESTful webservices deployed to Tomcat + some asynchronous data processing on background + little bit of data analytics
- Webapp server + OLTP + OLAP + document db + JMS

WHY DOCKER?

ENVIRONMENT SETUP
WAS COMPLICATED

UNSATISFIED DEVELOPER

OUR REASONS TO START USING DOCKER

- difficult setup of local developer environment or using of one common dev environment
- unification of used tools (same version of MySQL and Tomcat mainly)
- quick initial data setup
- unification of workflow
- possibility to run integration tests on local environment - faster reproducing of bugs/faster development

HOW WE STARTED

- really complicated orchestration with following tools

- Gradle
- Groovy
- Java
- Docker
- Docker registry

- Custom Arquillian modules
- Flywaydb tool
- Maven repository
- Git tooling
- And custom glue code around it

HOW WE STARTED #2

- Layered images (base > Java > Tomcat, base > MariaDb, base > Java > ActiveMQ)
- Reusable containers - we were using one servlet container for more apps and for more rounds of testing
- Reusable data

CHALLENGES

- Different OS on dev machines -> Docker machine vs. Host
- Recreating vs. reusing containers
- Deployment strategy for webapps/webservices
- Different configuration for everybody/every environment
- Many usecases: starting/stopping/deploying/running all tests/running selected tests
- And all of this with one tool

DEVELOPERS WERE HAPPIER

BUT WE WANTED MORE

FROM DEV TO PRODUCTION (ALMOST)

- First approach not so fit for production, because:
  - Too tightly coupled
  - Too much magic behind that
  - We had just VM solution in containers
  - Very conservative ops were strictly against any new layer in production environment

WHAT WE THOUGHT WE NEED FOR PRODUCTION

- Iterative approach to calm down Ops
  - Multi war Tomcat –>
    Separate Tomcat for each war –>
    Separate Docker container for each war –>
    No wars, but jars in Docker (SpringBoot/DropWizard) –>
    Containers orchestration –>
    Service registry/service discovery and auto-provisioning
- **Same image in all environments, promoted to Docker registry**

BUILDING OF SERVICES

- One additional step - to create Docker image
  - Automated to build cycle
- In ideal world - master branch always deployable
  - master branch built and app image pushed to registry
  - automatic pull to test environment, if tests passed, promoted to staging
  - automatic pull to staging - if approved, one click deploy to production

DEPLOYING OF SERVICES

- No more copying of wars
- Simple using of application Docker image from repository with proper version

LOGGING OF APPS

- Logs can be collected in the same way
  - Container logs to stdout
  - Container logs to syslog
  - Container logs to logfile on Docker volume
- Logs can be forwarded by Logstash and shown in Kibana

CONFIGURATION MANAGEMENT

- Started with properties files, different one for each environment
- Simple templating system for applying configuration to each environment with defined defaults, possibility to use some Key/Value store and override settings from command line
- https://github.com/markround/tiller

DEPLOYMENT TOPOLOGY

- Everything loadbalanced behind proxy, no public access to Docker hosts
- Nginx proxy running in container as well and dynamic
  - https://github.com/jwilder/nginx-proxy
  - https://github.com/jwilder/docker-gen

WHAT WORKED FOR US

WHAT WE HAVE LEARNED

- Data storage/Databases remained on physical/virtualized hardware
- Other hosts can have unified configuration and no special packages except Docker
- Own Docker registry is a must
- Properly made layers of images help much
- It is better to prepare base image in-house
  - you have more control over security risks
  - you can make it as minimal as you need

WHAT WE HAVE LEARNED #2

- Docker image should have just one service
- It should be possible to run one image in all environments (no hardcoded non-overridable configuration inside)
- Volumes are fine, but brings dependency on particular system
- Storage drivers were not important for us, but may be for Ops
- For logging it was sufficient to use stderr and forward it by Logstash

TEXT

WHAT THE FUTURE WOULD BE

- use of Kubernetes/Mezos/Docker Swarm
  - better scaling, distributed deployment, automatic discovery
- Interesting projects
  - http://rancher.com/rancher/
  - https://github.com/spotify/helios


IDEAS TECHNOLOGY CULTURE
WIRED
WHY THE
REALLY
UK MAGAZIN
LAUNCH
OF THE YEAR
CAPS LOCK
TRUE VIDEO
INV. VIDEO
DELETE
GRAPHICS
RUN
LIST
LOAD
SAVE
NEW
DRAW
REM
RAND
RETURN
INPUT
POKE
PRINT
GOTO
GOSUB
FOR
DIM
CLEAR
COPY
CONT
CLS
BORDER
NEXT
PAUSE
LET

SHORT DEMO

Q&A