

# How to Containerize WebSphere Application Server Traditional, and Why You Might Want To

David Currie @dcurrie  
*david\_currie@uk.ibm.com*

## InterConnect 2017



# Please note

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

# Agenda

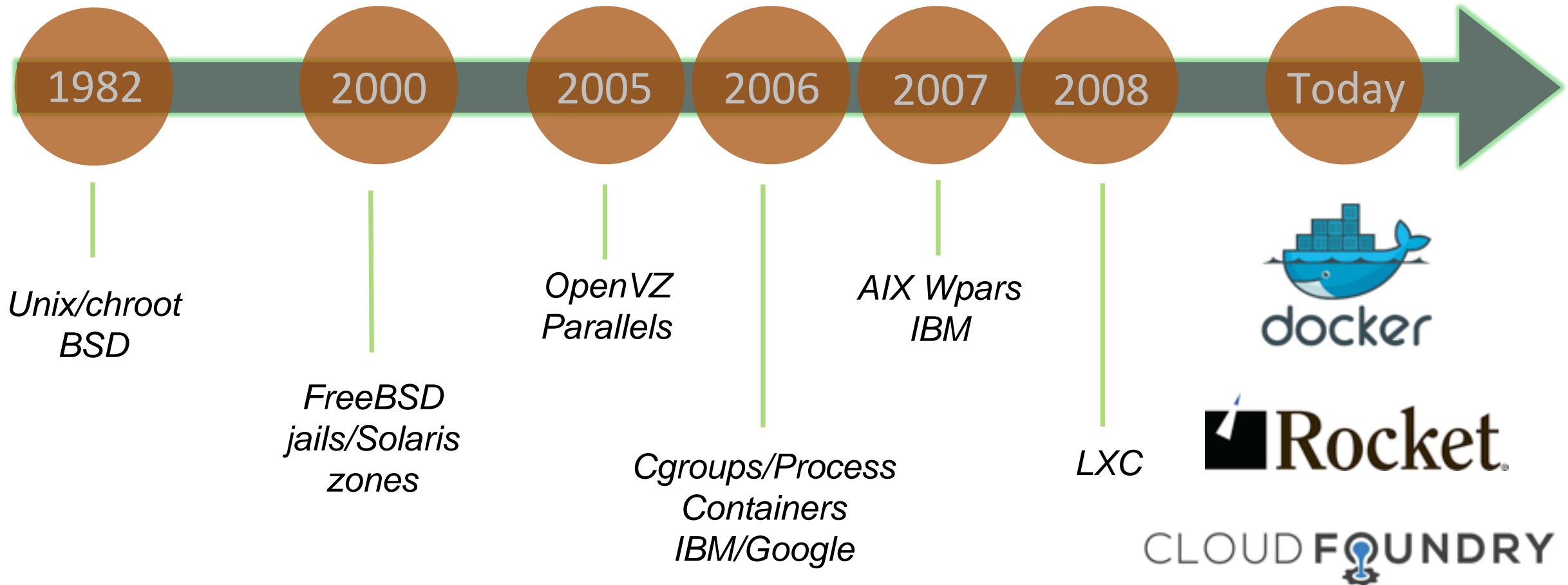
Overview of Containers and Docker

Use Cases for WAS traditional and Docker

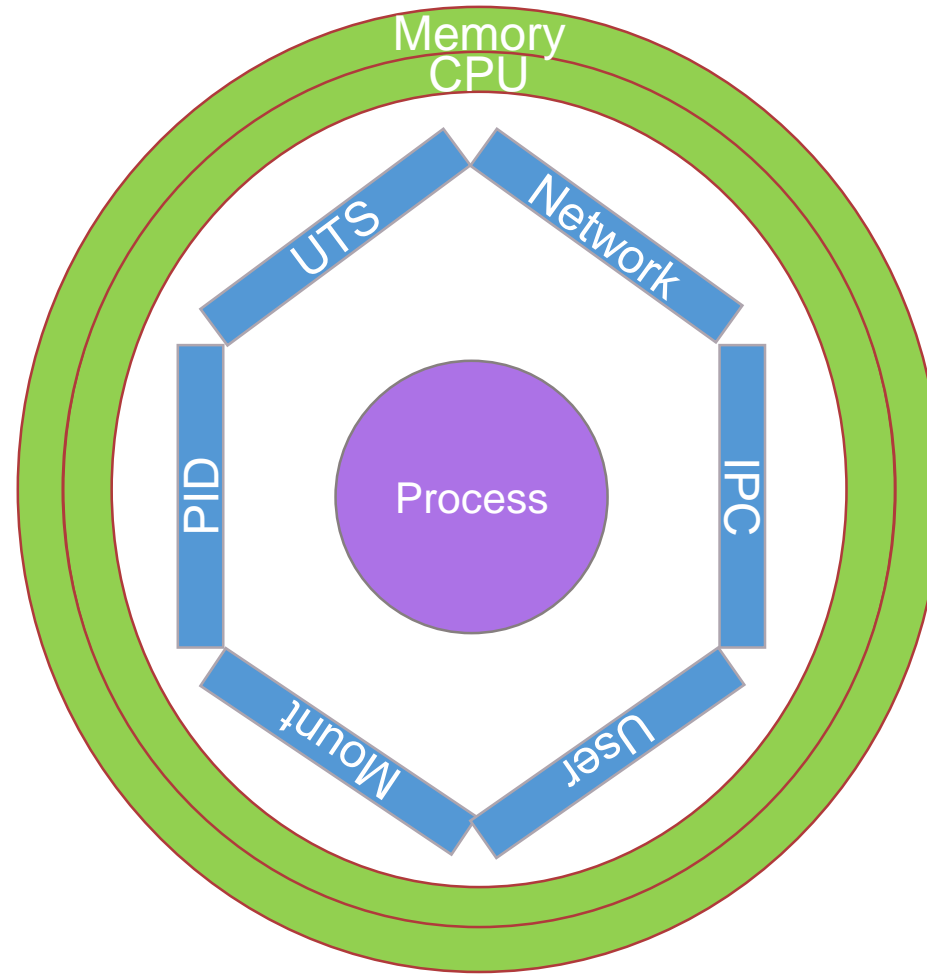
Getting started with WebSphere Application  
Server traditional and Docker

# Overview of Containers and Docker

# Container History



# Containers



= Namespaces for isolation



= Control groups for resource constraint

# Building a Docker image

Dockerfile

```
FROM websphere-liberty  
COPY app.war
```

```
docker build --tag app
```

server 1

Docker  
Engine

Image: app

app.war

server

java

ubuntu

websphere-  
liberty

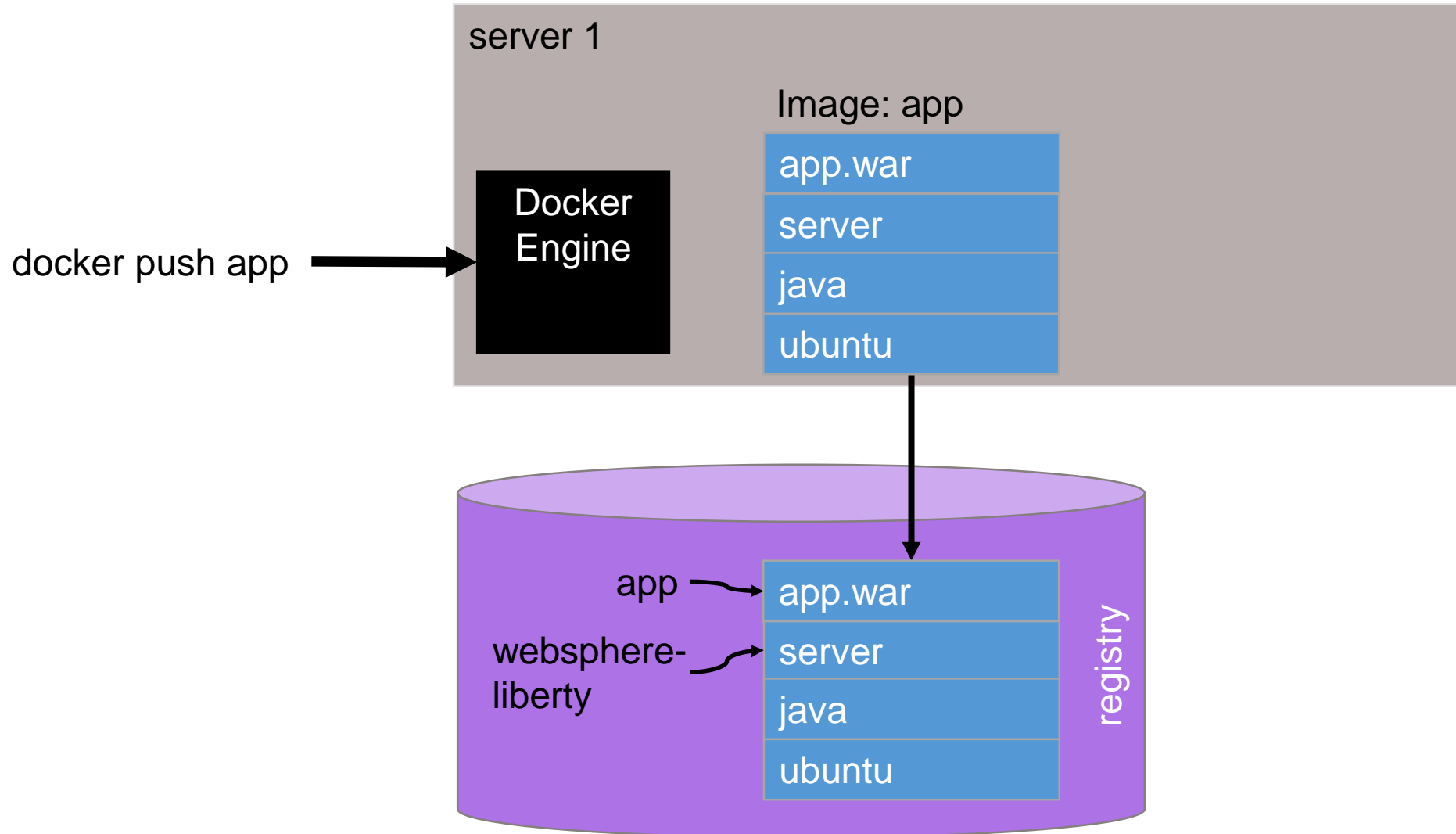
server

java

ubuntu

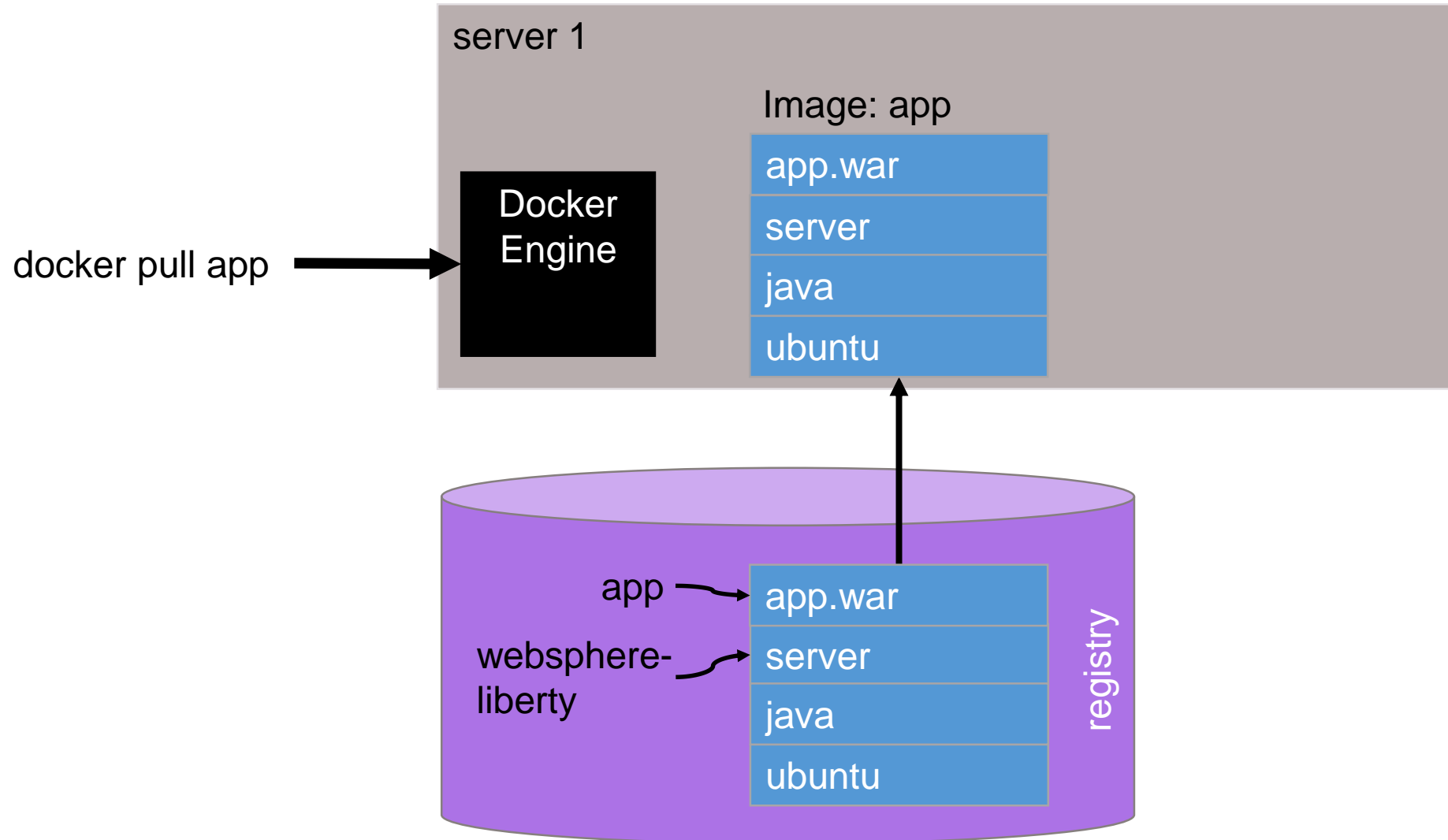
registry

# Pushing to a Registry

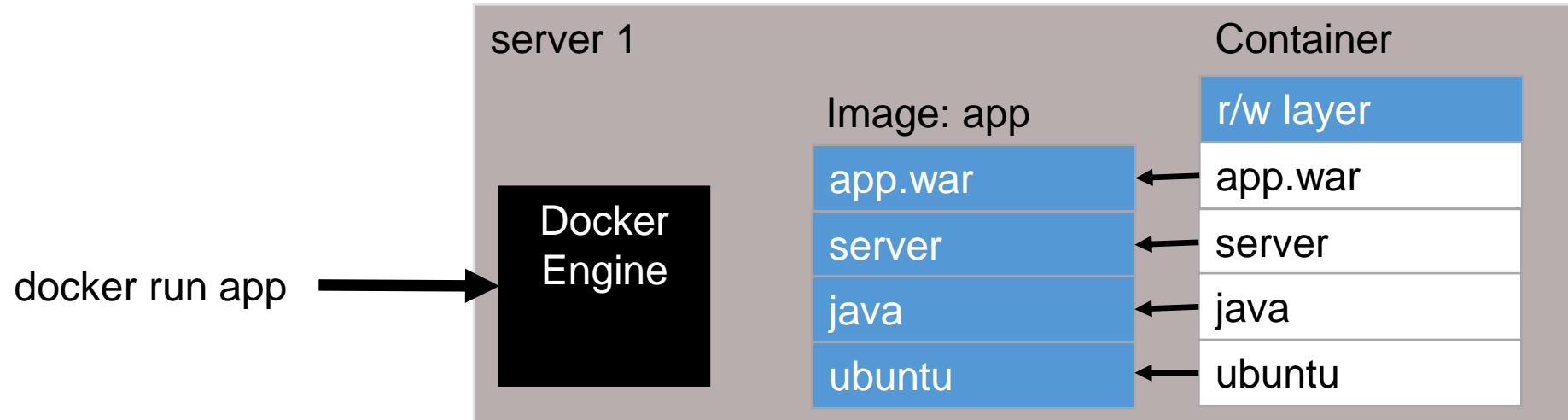




# Pulling from a Registry



# Running a Container



# Getting started with WAS traditional and Docker

# Docker Quick Start

Linux – run natively e.g. on Ubuntu

```
# From apt.dockerproject.org/repo  
sudo apt-get install docker-engine
```

Docker for Windows / Mac

‘Native’ applications running Docker engine  
in Hyper-V / xhyve VM

Docker Machine

Create VM with Docker installed

```
docker-machine -d virtualbox dev  
docker-machine -d openstack ... test  
docker-machine -d softlayer ... prod
```

Set environment variables to configure  
docker CLI to use machine

```
eval $(docker-machine env dev)
```

# Docker Hub images

WAS 8.5.5 developer licensed images

WAS 9.0 try/buy licensed images

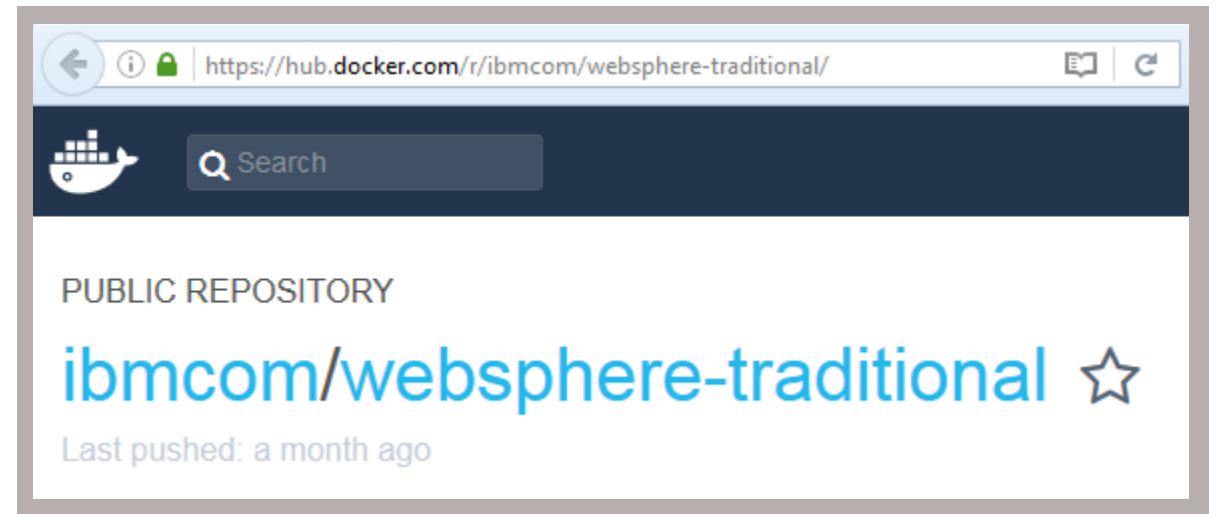
Start server and monitor PID file:

```
$ docker run -p 9043:9043 -p 9443:9443 -d \
  --name=ws ibmcom/websphere-traditional
```

Optionally add -e UPDATE\_HOSTNAME=true

Retrieve password for admin console:

```
$ docker exec ws cat /tmp/PASSWORD
```



# Profiles

Images with the 'profile' tag (and 'latest') have a profile pre-created

Images with the 'install' tag defer profile creation to container startup allowing profile, node, cell and hostname to be overridden

```
$ docker run --name test -h test  
-p 9043:9043 -p 9443:9443 -d  
-e HOST_NAME=test  
-e PROFILE_NAME=AppSrv02  
-e CELL_NAME=DefaultCell02  
-e NODE_NAME=DefaultNode02  
ibmcom/websphere-traditional:install
```

A profile from the host or a Docker volume can also be mounted at container startup

Combined with the 'swinging profiles' capability in WAS this allows you to move a profile from an image at one fixpack level to another

# Building your own Base or Developer Image

1. Obtain Installation Manager and WAS binaries from Fix Central and developerWorks or Passport Advantage
2. Host binaries on an HTTP/FTP server
3. Use Dockerfile.prereq to build prereq image
4. Run prereq image to output a TAR file containing the product install

5. Use Dockerfile.install to build install image from TAR file
6. Optionally use Dockerfile.profile to add profile to image

Final image size is around 1.7 GB

# Configuration and Data

For development, use admin console, remote tools support or wsadmin for application configuration and deployment

For production, wsadmin script deployment of application and build in to image

Use *-conntype NONE* so that server does not have to be running

Expectation is that WAS traditional containers are long-lived (may be started/stopped multiple times)

May still be desirable to persist certain files/directories outside of the container e.g. transaction logs, message stores or logs



# Building ND Images

Build an install image as for base/developer but using ND binaries

Create a Deployment Manager image with a dmgr profile

Create a managed node image

- Runs a node agent and application server

- Federates to the deployment manager on start-up

Application server (and application) may be configured in to image at build time (e.g. used as template for cluster member) or created at runtime via deployment manager

Some configuration (e.g. SIBus cluster members) must be configured via deployment manager

# Creating an ND Topology

Create a multi-host overlay network (or use host-level networking)

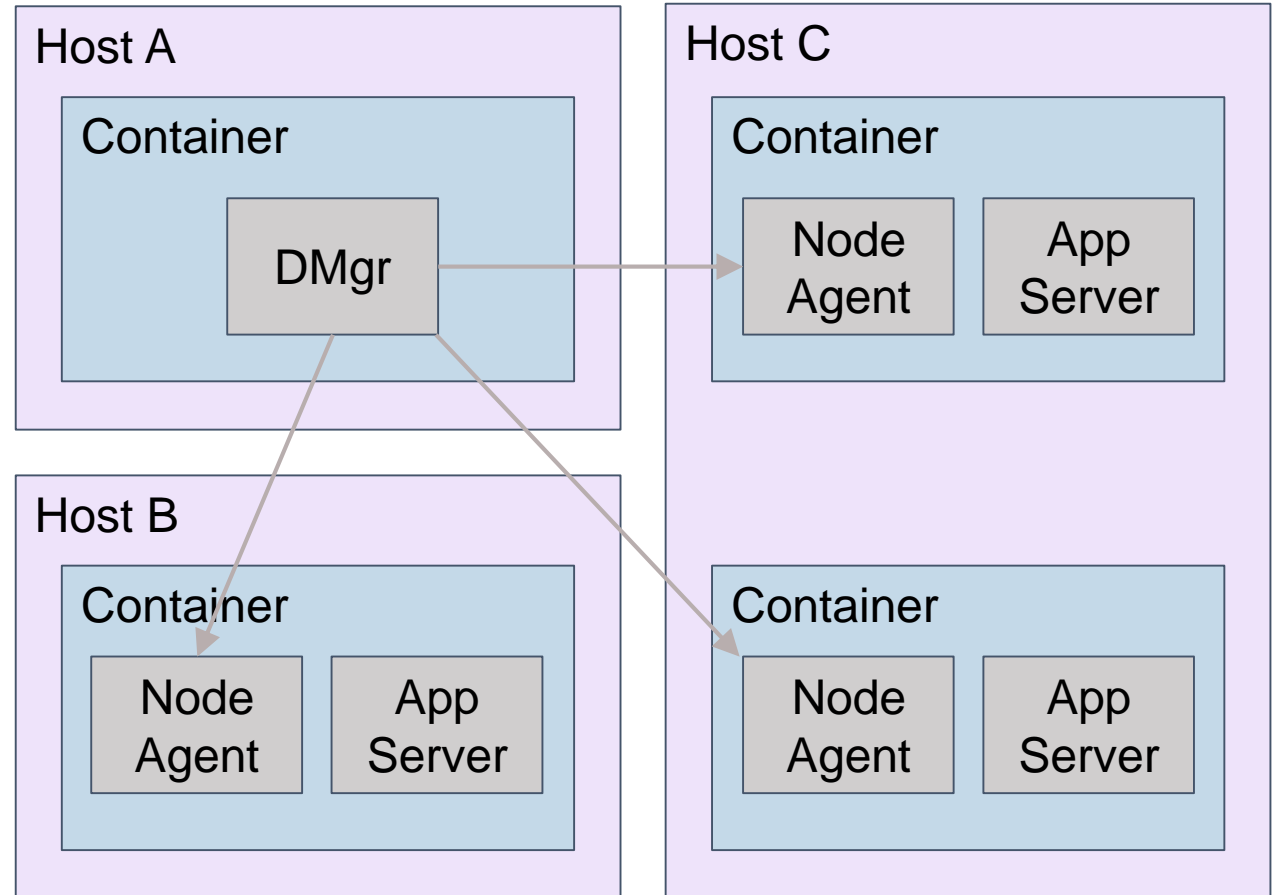
```
$ docker network create cell
```

Run deployment manager

```
$ docker run --name dmgr -h dmgr  
--net=cell -p 9060:9060 -d dmgr
```

Run application server image that federates to dmgr

```
$ docker run --name server1 -h server1  
--net=cell -p 9080:9080 -d appserver
```



# Use Cases for WAS traditional and Docker

| Use Case                        | Traditional                                                                                                 | Docker                                                                                                                         |
|---------------------------------|-------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| 1. Application Isolation        | Requires separate OS instances for each application, leading to resource fragmentation and higher overhead. | Applications run in isolated containers, sharing the host OS kernel, which reduces overhead and improves resource utilization. |
| 2. Rapid Deployment and Scaling | Deployment is slow and manual, often requiring physical server provisioning and configuration.              | Containers can be deployed and scaled rapidly using automation tools like Kubernetes or Docker Swarm.                          |
| 3. Consistent Environment       | Environments (development, testing, production) often differ, leading to "it works on my machine" issues.   | Containers ensure the application runs in the same environment across all stages of the development lifecycle.                 |
| 4. Resource Efficiency          | Each application has its own OS, consuming significant memory and CPU resources.                            | Containers share the host OS, allowing for higher density and more efficient use of hardware resources.                        |
| 5. Easy Updates and Rollbacks   | Updating an application often involves upgrading the entire OS or VM, which is risky and time-consuming.    | Applications can be updated or rolled back by simply replacing the container image, without affecting other containers.        |
| 6. Microservices Architecture   | Microservices are difficult to manage and scale using traditional VM-based architectures.                   | Containers are the ideal environment for microservices, allowing for fine-grained control and scaling of individual services.  |

| Use Case                   | Traditional                                                                                                        | Docker                                                                                                                                                    |
|----------------------------|--------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| 7. Security and Compliance | Applications are isolated by the OS, but updates and patches are often delayed, creating security vulnerabilities. | Containers can be updated and patched more frequently, and their isolation is enforced by the kernel, providing a more secure environment.                |
| 8. Portability             | Applications are often tied to specific hardware or OS configurations, making them difficult to move.              | Containers are portable and can run on any system that supports the container engine, facilitating cloud migration and hybrid environments.               |
| 9. Cost Reduction          | High resource consumption and slow deployment lead to higher infrastructure costs.                                 | Improved resource efficiency and rapid deployment lead to lower overall infrastructure and operational costs.                                             |
| 10. DevOps Integration     | Traditional workflows are often siloed, with development, testing, and deployment handled separately.              | Containers integrate seamlessly with DevOps practices, enabling continuous integration and deployment (CI/CD) pipelines.                                  |
| 11. Disaster Recovery      | Recovery from a disaster often involves restoring the entire OS and all applications, which is complex and slow.   | Containers can be backed up and restored more easily, and their state can be saved as images, simplifying disaster recovery.                              |
| 12. Multi-tenancy          | Sharing a single OS across multiple tenants is difficult and risky.                                                | Containers provide a natural environment for multi-tenancy, allowing multiple users or organizations to run their applications on the same host securely. |

# Developer productivity

# Containers as lightweight virtual machines

Single container  
platform for  
existing and new  
workloads

# Evolving to microservices via the Strangler Pattern





# Summary

Overview of Containers and Docker

Use Cases for WAS traditional and Docker

Getting started with WebSphere Application  
Server traditional and Docker

# Notices and disclaimers

Copyright © 2017 by International Business Machines Corporation (IBM). No part of this document may be reproduced or transmitted in any form without written permission from IBM.

## **U.S. Government Users Restricted Rights — use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.**

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed “as is” without any warranty, either express or implied. In no event shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted according to the terms and conditions of the agreements under which they are provided.

IBM products are manufactured from new parts or new and used parts. In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply.”

## **Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.**

Performance data contained herein was generally obtained in a controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and

the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer’s responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer’s business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer is in compliance with any law.

# Notices and disclaimers continued

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a particular, purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

IBM, the IBM logo, ibm.com, Aspera®, Bluemix, Blueworks Live, CICS, Clearcase, Cognos®, DOORS®, Emptoris®, Enterprise Document Management System™, FASP®, FileNet®, Global Business Services®, Global Technology Services®, IBM ExperienceOne™, IBM SmartCloud®, IBM Social Business®, Information on Demand, ILOG, Maximo®, MQIntegrator®, MQSeries®, Netcool®, OMEGAMON, OpenPower, PureAnalytics™, PureApplication®, pureCluster™, PureCoverage®, PureData®, PureExperience®, PureFlex®, pureQuery®, pureScale®, PureSystems®, QRadar®, Rational®, Rhapsody®, Smarter Commerce®, SoDA, SPSS, Sterling Commerce®, StoredIQ, Tealeaf®, Tivoli® Trusteer®, Unica®, urban{code}®, Watson, WebSphere®, Worklight®, X-Force® and System z® Z/OS, are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at: [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

# InterConnect 2017

