Containers in Linux on z Systems: Docker

Utz Bacher <utz.bacher@de.ibm.com>
STSM Linux and Docker on z Systems

IBM

A Message Brought To You By Our Lawyers

Trademarks of International Business Machines Corporation in the United States, other countries, or both can be found on the World Wide Web at http://www.ibm.com/legal/copytrade.shtml.

The following are trademarks or registered trademarks of other companies.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.
Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
Windows Server and the Windows logo are trademarks of the Microsoft group of countries.
ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.
UNIX is a registered trademark of The Open Group in the United States and other countries.
Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.
Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

* Other product and service names might be trademarks of IBM or other companies.

• The information contained in this publication is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this publication, it is provided AS IS without warranty of any kind, express or implied. In addition, this information is based on IBM's current product plans and strategy, which are subject to change by IBM without notice. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this publication or any other materials. Nothing contained in this publication is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.
• References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in this presentation may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.

Agenda

Containers and Docker

Docker Ecosystem

Docker on z

Containers and Docker

Docker Ecosystem

Docker on z

What are Containers?

- Virtual environment within Linux OS instance
  - So applications share OS kernel
  - Only application is started, not entire Linux environment
- Efficiency: no virtualization overhead
  - No full system or para-virtualization, but isolation only by the kernel
- Own file system tree via chroot environment
- Container separation of OS objects via „name spaces“
  - Process IDs, network devices, mount points, users, and more

Docker: “Build, Ship, and Run Any App, Anywhere”

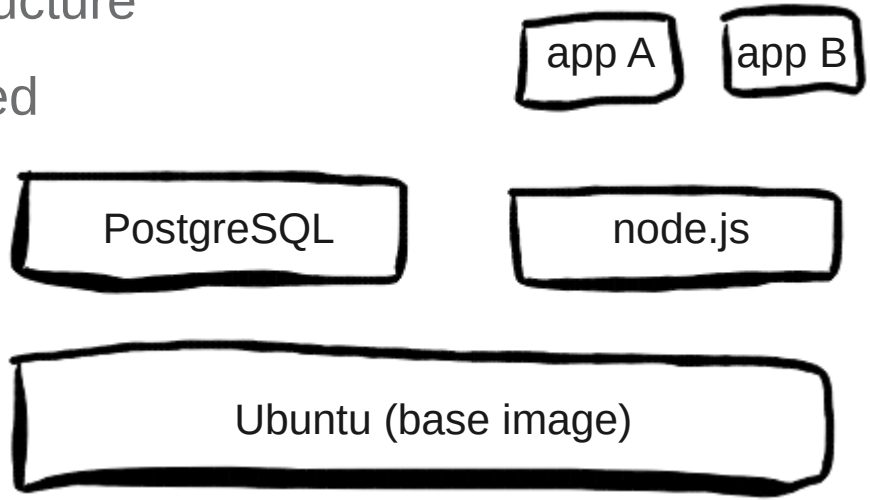
- One implementation of a container solution
- Powerful tool to build, modify, deploy, run, manage containers
  - Extreme focus on efficiency, fast response times
  - Stores incremental differences and caching whenever possible
- Registries serve as central places for images
  - Efficient distribution, versioning
- Terminology
  - image: a self contained set of files, base for a container
  - container: runnable instance, based on an image
- Maintained by Docker, Inc.

Typical Container Attributes

- Self contained sets of files – escape dependency hell, reduce test matrix
- Serve a single task
- Can build on top of each other
- Can be deployed simple and quickly
- Can easily be customized, re-packaged and versioned
- Can use synergies in the kernel, if images eventually base on the same libraries (same file in underlying images)
  - without having to use KSM (Kernel Samepage Merging)

Typical Container Layering

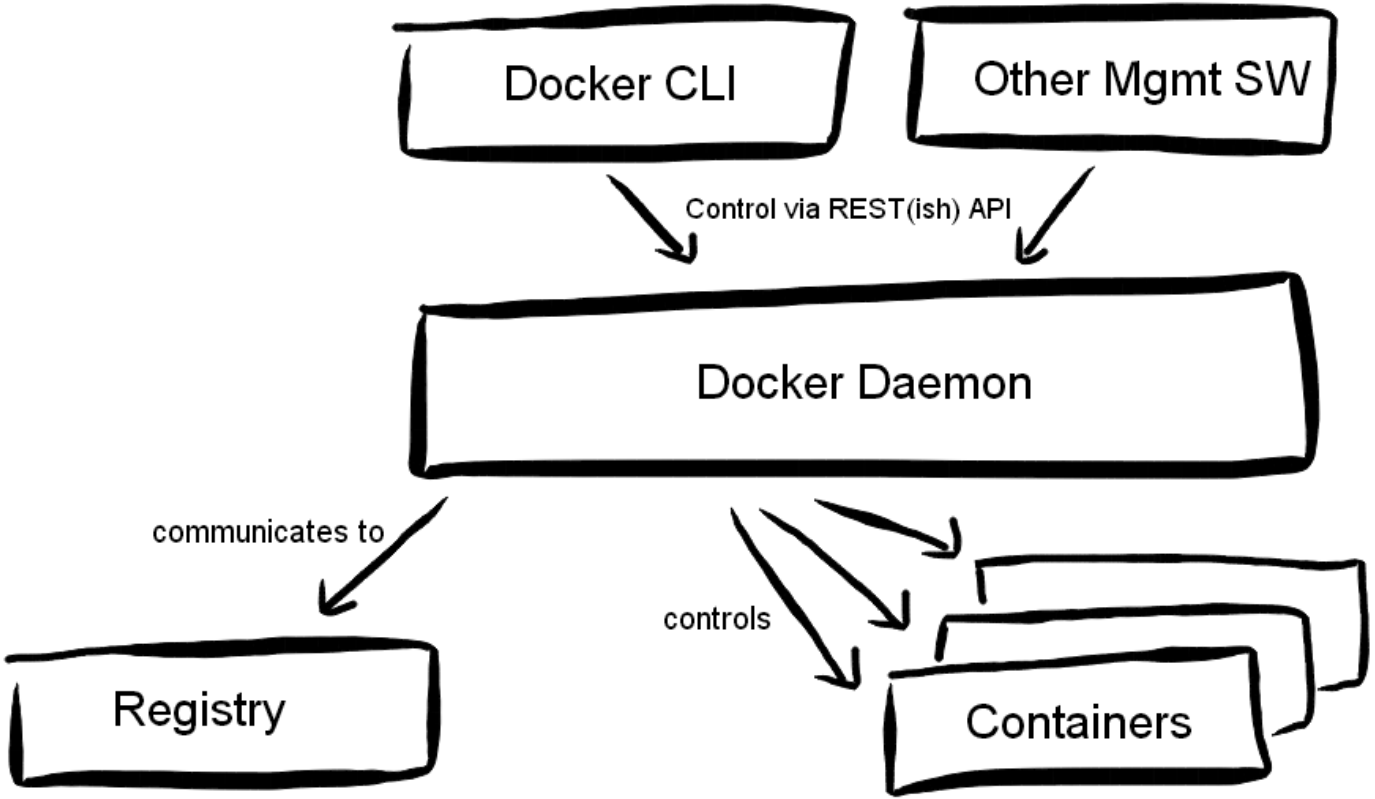
- Images can build on top of on another
  - Allows to build on common infrastructure
- Only differences are stored and pushed
  - Memory efficiency and density
- Change in underlying layer requires rebuilding all depending images
  - Will generate a new image (with new ID) for app A
  - Having both versions of app a allows for simple migration and rollback



Docker ties Dev and Ops Together

- Development and Operations use a pervasive tool chain
  - Ops uses a deployment that has been developed in and tested for
  - No more “but it worked in my environment” by a developer
  - Development can quickly get an environment which matches the real deployment
- Docker provides a universal toolchain for Dev and Ops
  - Extreme focus on usability, speed, and efficiency for increased productivity
- Decomposition of solution into microservices helps scalability, extensibility, flexibility
  - Simple updates and rollback of pieces

Docker Structure



Dockerfile Example

- Use Dockerfiles for controlled builds of images:

```
# use this base image. Downloaded if not present.
FROM rhel:7.1

MAINTAINER Whatever my name is <some@address.com>

# run commands:
RUN yum install -y httpd

# copy files into the image
ADD index.html /var/www/html/

# publish a port of the container
EXPOSE 80

# how the container is started
ENTRYPOINT ["/usr/sbin/apachectl","-DFOREGROUND"]
```

- Note: each step will be built using an intermediate container, resulting in an intermediate image on the way to the final image

Microservice Architectures

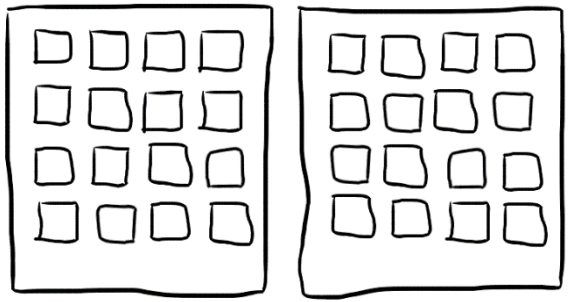
From monolithic architectures …

- Architecture aligned and optimized to server boundaries
  - Installed on these servers
- Layered applications with coarse granularity
  - Static, long living
- Often went virtualized, but no progress in solution structure
- Use components which fit to solution

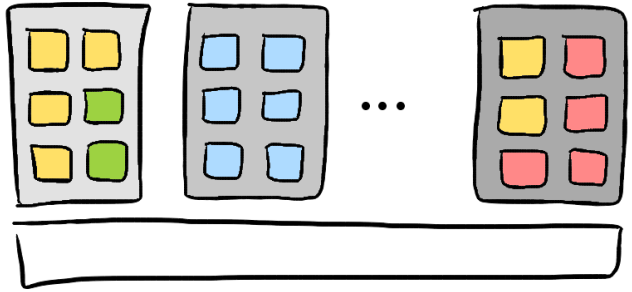
... to microservices

- Solution broken into small units
  - Delivered in containers
- Independent services, loosely coupled
  - Short life time, fast changes
- Resilient services
  - Scaling components is simple
- Using best tool for each subtask

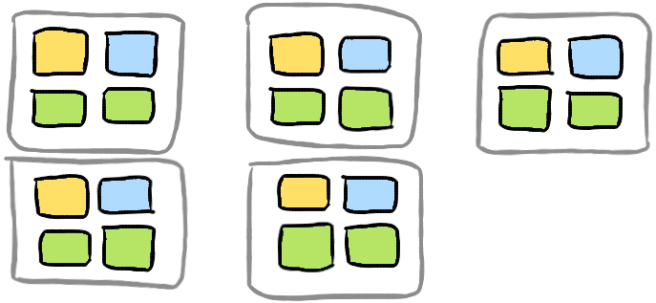
Microservice Deployment Types



Scale up for maximum efficiency

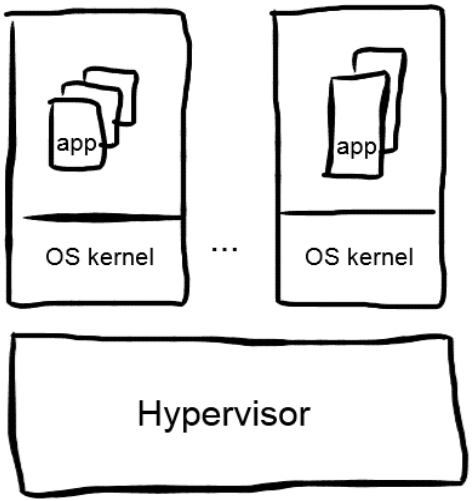


Isolation, QoS and scaling
for tiers and tenants



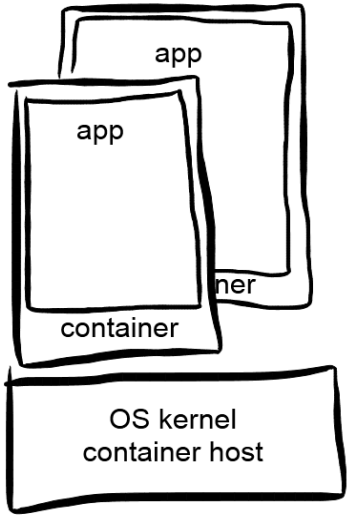
Grouping microservices end-to-end allows for
simple scaling and optimized local communication

Virtualization vs. Containers



Infrastructure oriented:

- coming from servers, now virtualized
- virtual server resource management
- several applications per server
- isolation
- persistence

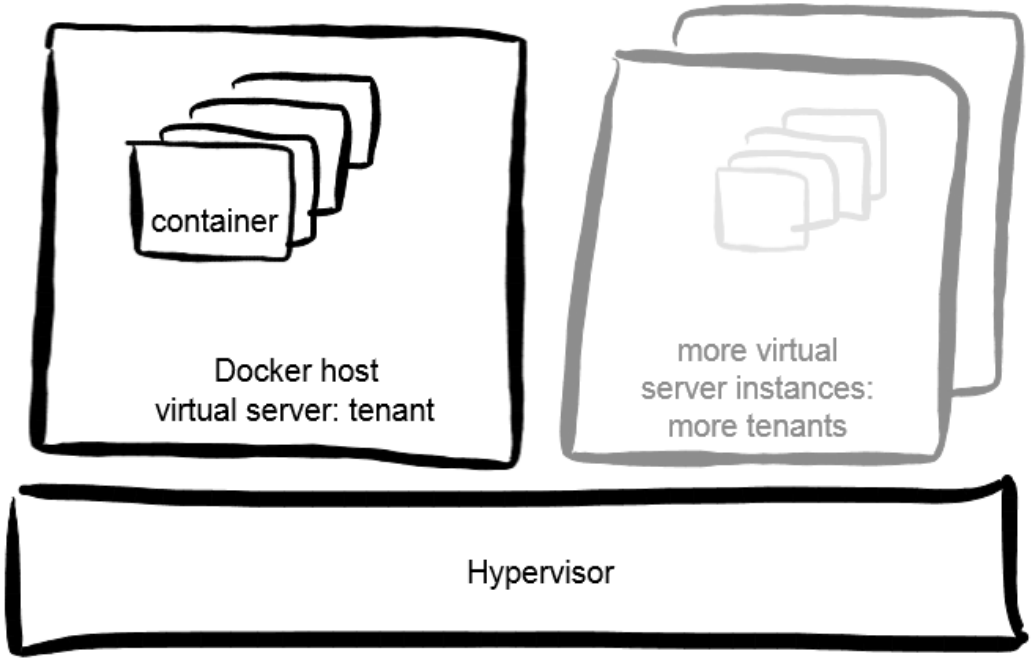


Service oriented:

- application-centric
- application management
- solution decomposed
- DevOps
- dynamic

Virtualization and Containers

- Virtual machine separation between tenants
  - Virtualization management for infrastructure
  - Isolation
- Many containers within tenants
  - Container efficiency
  - Docker management and ecosystem



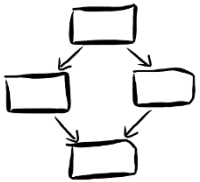
Containers and Docker

Docker Ecosystem

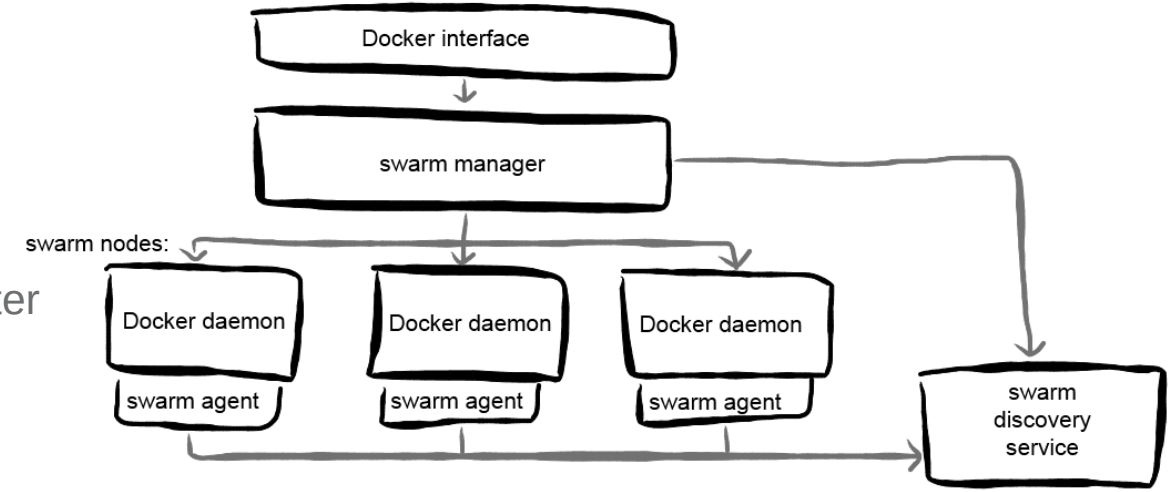
Docker on z

Docker Ecosystem: Basic Docker Tools

- *machine*: provision Docker onto hosts (local VMs, Clouds)
- *compose*: create multi-container applications, manage and scale them through single commands



- *swarm*:
  use Docker on an entire cluster

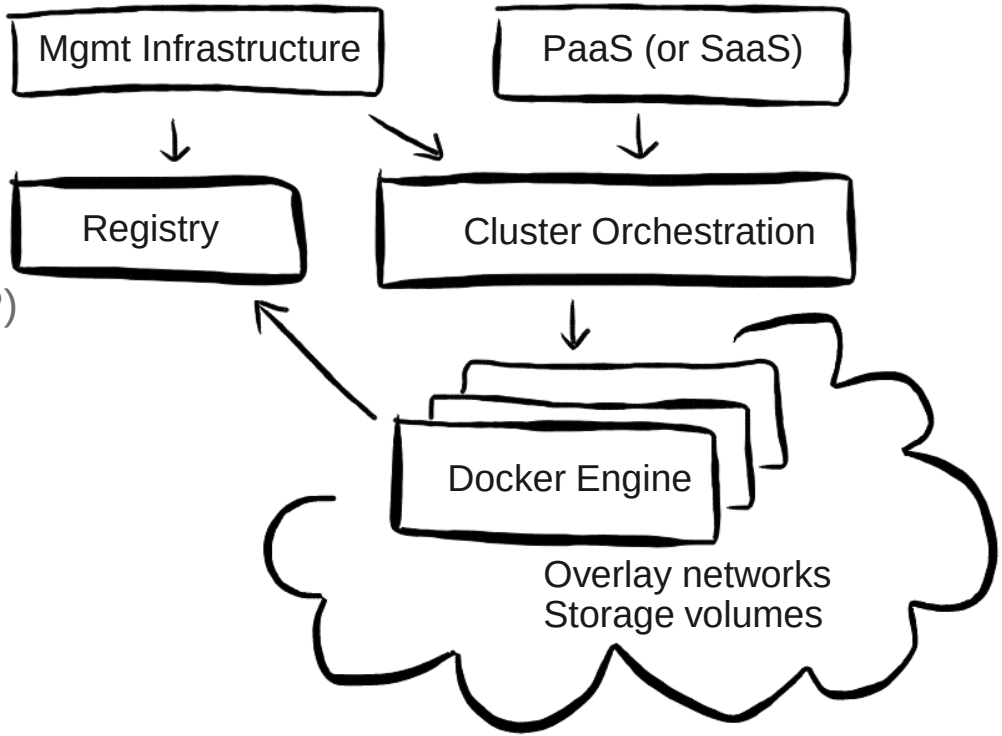


Docker Ecosystem: Registry

- Docker Hub: Public Registry with User and Organization Management
  - Private areas available
  - Contains ~100 official images of companies (Ubuntu, MongoDB, …)
  - Automated builds possible
- On-premise Private Registry („*distribution*“): Open Source
  - Simple user management (No web UI)
- *Docker Trusted Registry* (DTR): Commercial Docker Offering
  - User and organization management
  - AD/LDAP authentication
- SUSE *Portus*: Open Source Authorization Service and Frontend for Private Registry
  - Users and organization management
  - LDAP authentication

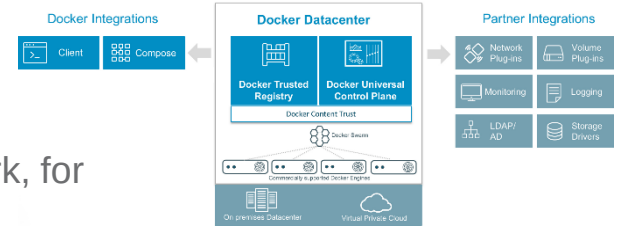
Docker Ecosystem: How It Plays Together

- PaaS
  - Cloud Foundry
  - OpenShift
  - BlueMix
  - Mesos frameworks (e.g. Marathon)
- Management
  - Docker *Universal Control Plane* (UCP)
  - IBM *UrbanCode Deploy* (UCD)
  - or part of PaaS
- Orchestration
  - Docker *swarm & compose*
  - Apache *Mesos*
  - Google *Kubernetes*
  - Hashicorp *Nomad*

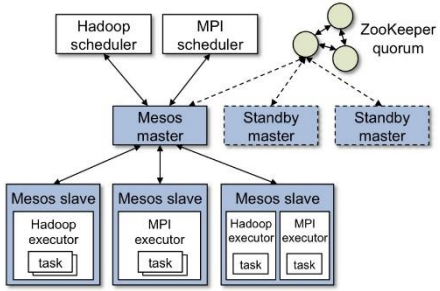


Docker Ecosystem: Cluster Orchestration

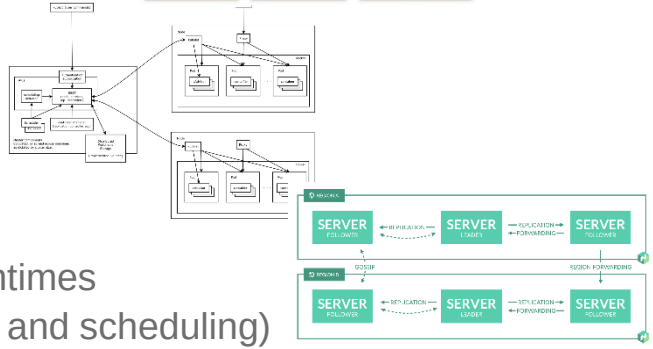
- Docker *swarm* and *compose*
  - Simple cluster framework fitted to run Docker containers
  - Composite applications with compose
  - Docker acquired makers of Mesos Aurora scheduling framework, for integration of Aurora parts into swarm



- Apache *Mesos*
  - Large scale cluster project
  - Marathon framework schedules containers
  - Mesos intends to run containers natively (without additional framework)
  - IBM intends to add value with Platform Computing scheduler (EGO)



- Google *Kubernetes*
  - Large scale cluster manager/scheduler by Google
  - Base for CNCF (Cloud Native Compute Foundation) orchestration
  - Grouping and co-location of containers as pods, forming a service



- Hashicorp *Nomad*
  - Cluster manager and scheduler for VMs, Containers, language runtimes
  - Simple, efficient, scalable, limited scope (just cluster management and scheduling)

Containers and Docker

Docker Ecosystem

Docker on z

Docker on z Systems

- Docker is written in Go
  - golang on x86, gccgo on s390x (at first)
  - Docker has accepted patches for full gcc support
    - Includes CI (being established for z Systems)
  - golang port available, switch to golang over time
- Availability in distributions
  - Support available today: Fedora 23, SUSE SLES 12 SP1+containers module
  - Ubuntu 16.04 by April 2016
  - Working with Red Hat on inclusion (tech preview from IBM on devWorks)
- Docker ecosystem working at large: registry, compose, swarm, cAdvisor, machine (prototype), kubernetes (prototype), ...
  - Community base images available on Docker Hub
- IBM UrbanCode Deploy supports Linux on z Systems

Docker and Registry Will Do „Multi-Arch“

- Docker and registry will become multi-architecture aware
  - Contribution mainly driven by IBM
  - Today, docker images are blobs and happen to be mostly x86‘ish
  - Changes for future releases of docker and registry:
    - Architecture awareness of Docker
    - Pushing content of different architectures to an existing *name:tag* will add that architecture to the definition (manifest)
- i.e. „docker pull ubuntu:16.04“ will do the right thing
  - wherever you are

Docker on z Systems – Summary and Outlook

- Docker and base ecosystem available with full functionality
- Content (images) being added for most popular Open Source projects
- Docker today enables mixed architecture development and deployment
- Multi-arch support and availability of Ubuntu further simplify portability
- Second level virtualization provides perfect tenant isolation with low overhead while providing Docker agility and efficiency
- Docker performance inherits platform performance characteristics
  - Allows both scale-up and scale-out in a box: structure solutions along client requirements, not environment-imposed restrictions

THANK YOU

Open Container Initiative

- Docker is de-facto container format standard
  - CoreOS launched competitive and open approach (rocket container runtime, appc container format)
- Open Container Initiative to define industry standard container format and runtime
  - Housed under the Linux Foundation, sponsored by many IT companies
    - Including CoreOS, Docker, Google, IBM, the Linux Foundation, Mesosphere, Microsoft, Red Hat, SUSE, VMWare, and many more
- Docker donated their container format and runtime („runc“)
- OCI principles for container specification:
  - Not bound to specific higher level stack (e.g. orchestration)
  - Not bound to particular client, vendor, or project
  - Portable across OS, hardware, CPU architectures, public clouds

Docker Ecosystem: OpenStack

- Management integration and standardization (keystone etc.)
  - But giving up on Docker CLI flexibility
- OpenStack components
  - *Nova*: Docker virt driver
    - Runs Docker images on hosts, images stored in glance
  - *Heat*: Docker plugin
    - Use Docker containers in Heat templates
  - *Magnum*: control orchestration via Docker and Kubernetes
    - Goal to fully leverage Docker efficiency
    - Multi-tenancy for Docker and Kubernetes
- Side note, different direction: *Kolla* deploys OpenStack environment in containers

Docker And Cross-Platform Portability

- Docker user experience (CLI, REST API) is identical across platforms
- Containers in binary form are not portable, so source code or s390x binary must be available
- Microservice architectures often have clean structure and simple individual components
- Containers are often created through Dockerfiles (build descriptions) containing:
  - Specification of base image
    - If same distribution is available on s390x, usually simple
    - Currently, closest thing to Ubuntu on x86 is Debian on z
    - If base image is not available, need some workarounds to get there (e.g. „golang“)
  - Additional steps to modify image. Very often platform independent

Docker and Performance

- Docker containers mainly use namespaces for isolation and cgroups for resource control
  - Starts workload and gets out of the way – application runs directly on kernel
- Workload performance under Docker is defined through platform performance
  - Docker has no direct implication on workload runtime behavior
  - If you can run hundreds of applications in a Linux, you can run it under Docker with about the same performance
  - SDN and SDS mechanisms chosen define networking and storage limitations
    - Typically low overhead
- Scaling characteristics of z Systems allows for both scale-up and scale-out
  - Hundreds to low thousands of containers in a large Linux system
  - Hundreds of tenants on smaller scale Linux systems
  - Design environment according to your solution requirements, not according to your systems constraints!

Docker on z: Getting Started

- Base images
  - Create based on your host distro (e.g. with a script from blog below)
  - Use a public z Systems image from Docker hub (no warranty for content!): https://registry.hub.docker.com/search?q=s390x
- A lot of Open Source applications being made available (build description, Dockerfiles), linked from
  - https://www.ibm.com/developerworks/community/groups/community/lozopensource/
  - https://github.com/linux-on-ibm-z/ (e.g. for cAdvisor)
  - https://registry.hub.docker.com/search?q=s390x
- Tutorial with z in mind at http://containerz.blogspot.com/
  - first steps, ecosystem, advanced topics