Containers in Linux on z Systems: Docker

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

Containers, Docker, and Microservices
Docker Ecosystem
Docker on z
Containers, Docker, and Microservices

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 one 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 Benefits

- Facilitates portability and cross platform deployment through generic build description
  - Develop applications on x86 and build for both x86 and z platforms, seamlessly deploy to x86 and z Systems
- Package applications without worrying about dependencies on other libraries and software
  - If container app requires dependencies, creator of the container adds them to the container image
  - Entirely independent of host software level
- Simple re-use of components
  - One container image used to deploy same application many times by different people
- Supports micro-service architecture by simple deployment and management of components
  - Large application consisting of several SW components can be broken down into multiple containers to allow for reuse of parts
- Large density through lightweight container isolation mechanism in Linux kernel
  - Hundreds to thousands of virtual containers to run in one system
- Docker ties Dev and Ops together
  - Consistent environment from Dev to Ops facilitates staging and avoids environmental errors
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
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
Microservice Challenges: Latency

Internal flow between microservices

Network latencies add up in meshes of microservices.

z Systems: large complex with in-box networks reduces latencies.
Microservice Challenges: Scaling

The Scale Cube
(From Abbott & Fisher: “The Art of Scalability”)

- Replication components is mostly simple
- Splitting applications into microservices can be hard
- Data partitioning is often hard
- Scaling stateful services can be complex
  - e.g. transactional context across microservices
- z Systems can scale anywhere from horizontally to vertically, aligned
  - scale-up can simplify solutions

z Systems: sometimes bigger *is* better
Containers, Docker, and Microservices

Docker Ecosystem

Docker on z
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

- **Sonatype Nexus 3**: repository solution for various formats, including Docker
  - Merge external and internal registries
  - Free/OSS and commercial version available

Multi-arch being added (it basically works, no GUI yet)

Multi-arch works fine

Runs on x86, but serves z nodes.
No multi-arch yet

Multi-arch to be verified

Multi-arch to be verified
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)
  - IBM *Spectrum Conductor*
  - or part of PaaS

- **Orchestration**
  - Docker *swarm & compose*
  - Apache *Mesos*
  - Google *Kubernetes*
  - IBM *fr8r*
  - Hashicorp *Nomad*
Docker Ecosystem: Management

- docker-machine converts a Cloud or on-prem Linux image into a docker/swarm host

- Docker Universal Control Plane
  - Part of Docker Datacenter
  - Manages pipeline from development to operations
  - Manages swarm cluster and host resources like networks and volumes

- IBM UrbanCode Deploy
  - Part of UrbanCode product suite for DevOps
  - Manage deployment of applications on physical and virtual servers and containers
  - Supports change management and audit trails
IBM Spectrum Conductor for Containers

- Multi-tenant container infrastructure and orchestration in a shared resource environment
  - Support full life cycle of containerized application in clusters
  - API/CLI and GUI
  - Global shared access to data
  - Storage tiering
  - RBAC
  - Manage QoS per consumer
  - Service discovery support

- Community and enterprise edition available for x86
  - not yet finalized for s390x
IBM Spectrum Conductor Architecture Overview

From: Xu: EDGE_2016-SCL-2484-A Software Defined Scalable and Flexible Container Management Solution.pptx @ EDGE 2016
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
Containers, Docker, and Microservices
Docker Ecosystem

Docker on z
“Linux Your Way”: Greater Flexibility and Choice
Docker on z: Support Overview

- **SUSE:**
  - Docker engine provided as part of SLES 12. Additional components being added.
  - Fully supported

- **Canonical:**
  - Engine provided as part of universe for 16.04. Additional components being added.
  - Universe: community maintained, i.e. not officially supported

- **Red Hat:**
  - Under discussions. Client requests can only help

- **3rd Party:**
  - Formal support by Docker, Inc.: being worked on. Details tbd.
  - Roguewave: commercial support for Docker (and a lot of other Open Source packages) for z
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 impact 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
    - proof point: 10k containers (mix of heavy, medium and light workload)
  - Hundreds of tenants with smaller to medium scale Linux instances
    - proof point: several million containers on single physical footprint

- Design environment according to your solution requirements, not according to your systems constraints!
Ongoing Work: Isolated Containers

- Transparently use KVM to run Docker workload
  - Improved isolation through hardware-based virtualization technology
  - Smaller attack surface from untrusted container workload towards hosting environment

- Maintain Docker ecosystem and user experience
  - KVM not visible to user, started under the covers
  - Re-use of Docker images without any changes

- Currently engaging with Open Source community
  - First measurements show ~55ms overhead to start up a KVM per container
Ongoing Work: Autoscaling

- Increase utilization in a large scale container environment
  - While maintaining business goals

- Workload is associated to business importance

- Performance monitored across all layers
  - Workload instrumentation
  - Container infrastructure
  - Hypervisors

- Resources adjusted on these levels based on workload priorities

- Integrates with existing management and orchestration infrastructure

- Currently working on and evaluating a prototype with a large financial client
Docker on z Systems – Summary and Outlook

- Docker and base ecosystem available with full functionality
  - Based on identical source code
  - z Systems is part of Docker's „Continuous Integration pipeline“
- Commercial support available/being extended
- Docker Hub Content (images) being added for most popular Open Source projects
- Docker today enables mixed architecture development and deployment
  - Multi-arch support and 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