Applying Container Technology to the Virtualized Ground System

GSAW 2017 “Looking Beyond the Horizon”
Richard Monteleone

© 2017 by RT LOGIC Inc. Published by The Aerospace Corporation with permission

All brands and trademarks mentioned in this presentation which are possibly registered or protected by third parties are solely subject to the trademark and ownership rights of the registered owner(s)
RT Logic Virtualized Ground System (VGS) “The Big Picture”

• vFEP (Virtual Front-End Processor) Application
• Using hardware (H/W) virtualization with virtual machines (VM’s)

Comparing H/W Virtualization (VM’s) to Containers

Applying a container technology “It’s Go Time!”

• Making the transition into containers
• Building/deploying/running Docker containers
• Automation
• Container isolation and monitoring
  – How isolated are containers?
  – Monitoring the Docker Engine and containers

Summary

Questions?
VFEP Taking a look inside

- Publish/Subscribe message bus architecture (loosely-coupled components, independently versioned)
- Highly configurable, extensible, scalable, secure and efficient
  - Auto-created user interface and auto-generated documentation
- Extensive API Support (GEMS, REST XML/JSON, SNMP)
Quick look at VGS VM’s and how we use them

- Applications installed and configured on individual VMs
  - Dedicated OS
  - Application ISO images mounted and installed
  - Command and telemetry channels *interactively* user configured
  - Firewall (iptables) and service configuration (lifetime management)
- Things are **really good now** but could they be **even better**?
  - Hardware sharing, Snapshots, vMotion, VM templates, Application isolation, OVA’s, Secure, Stable, Scalable
Quick look at containers. How do they differ from VM’s?

- Shared OS across containers
  - Containers are more resource efficient (only use what they need when they need it)
    - More containers running on less Bare Metal Hardware
  - Extremely fast to start
  - Extremely lightweight
  - Docker Engine OS (kernel) and container compatibility required
  - Failures/cycling of the Docker Engine-OS-H/W can be more impactful
- Capable of running directly on Bare Metal Hardware
Containers (SaaS) and the 12 Factor App

- Methodology leveraged to produce “good” containers
  https://12factor.net/

I. Codebase (single purpose, one code base/application versioned independently)

II. Dependencies (be explicit)

III. Configuration and code separation

IV. Backing services (think resources)

V. Build/release/run (separation)

VI. Processes (stateless, non-sharing)

VII. Port binding

VIII. Concurrency

IX. Disposability (lightweight, fast startup, graceful shutdown)

X. Dev/prod parity (keep as similar as possible)

XI. Logs

XII. Admin processes
Transitioning into Containers cont’d

VGS changes related to how we install/configure/run applications

– GSVeh Simulator
  ✓ Multiple applications (Ground System and Vehicle Simulation)
– Don’t store data within a container
  ✓ vFEP Recording/Playback of command and telemetry data
  ✓ Storing configuration and log files
– Application lifetime
  ✓ Lifetime management no longer controlled internally
– Interactive application configuration and deployment
  ✓ Eliminate ISO mounts for application installation
  ✓ Need to automate the building of images and the deployment of containers
VGS deployment with Docker containers

– Split the GSVeh Simulator into two containers
– Same configuration as before
  • One command channel and two telemetry channels
– Configured Docker version 1.12 and 1.13 environments
  • Optionally running our Docker Engines in VM’s
Creating Docker images, what needed to be done?

- Images are used to create immutable container instances
- Dockerfiles contain the instructions needed to build each image
  - Build images FROM a (lightweight) initial image
  - Extensive use of LABELs to support image/container traceability
  - COPY/RUN used to install and configure each application
  - Explicit EXPOSE for container to container communication
  - Defined VOLUMEs as storage for record/playback of command and telemetry data, configuration files, and logs.
  - Defined a (single) ENTRYPOINT to execute each container
- Removal of internal service lifetime configuration
  - Now managed with the container lifetime
Initially images are built and containers are run manually

- Built images from instructions in Dockerfile(s)
  - `docker build -t="gsaw/vfep:1.0.1" .`
- Create the VGS network
  - `docker network create --driver bridge vgs_network`
- Run a container from an image as a daemon on the docker host
  - `docker run -d -p 30010:30001 --net=vgs_network --name vfepA gsaw/vfep:1.0.1`

Equivalent using Docker Compose

- Define a single `docker-compose.yml` service definition file
- Single command: `docker-compose up`
  - Builds images “if necessary”, creates a container network, deploys and runs all containers

Virtual Ground System Operational !!!
Build/Deploy/Test Automation

(A) Nightly Product Builds

(B) Nightly Docker/Compose Builds

(C) Virtualized Ground System Operational!!!
Running “bad” potentially disruptive containers in the VGS

- CPU Stress container example run:
  - `docker run -it --rm --network=vsc_network --name=cpu_stress gsaw/cpu_stress:1.0.0`

- Memory Test (limit memory_test container to 50meg)
  - Verify the VGS maintains the normal operational state
  - Undisturbed by “bad” containers sharing same Docker Engine/OS
  - Can I see what’s really going on in this container environment?
Monitoring Docker with cAdvisor

View/monitor the Docker Engine and images/containers
  - Insight into resource limitations/utilization and performance
  - `docker run ... --publish=8080:8080 --detach=true
                            --name=cadvisor  google/cadvisor:latest`

---

![Monitoring Dashboard](image-url)
CPU Monitoring with cAdvisor

Processes

<table>
<thead>
<tr>
<th>User</th>
<th>PID</th>
<th>PPID</th>
<th>Start Time</th>
<th>CPU %</th>
<th>MEM %</th>
<th>RSS</th>
<th>Virtual Size</th>
<th>Status</th>
<th>Running Time</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>16.169</td>
<td>16.142</td>
<td>15:18</td>
<td>1.40</td>
<td>0.00</td>
<td>1.21 MB</td>
<td>11.09 MiB</td>
<td>5s+ 0:00:00</td>
<td>cpu_stress.sh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16.218</td>
<td>16.169</td>
<td>15:18</td>
<td>0.00</td>
<td>0.00</td>
<td>308.00 KiB</td>
<td>4.03 MiB</td>
<td>3s 0:00:00</td>
<td>sleep</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CPU

Total Usage

Usage per Core

Usage Breakdown
Container Benefits

• Application scalability
• Lightweight
  – Very fast startup, smaller size, easy to distribute
• Cost reductions
  – Increase of workloads running on less H/W
  – Less OS’s to license/manage/patch/update
• Containers are properly isolated from one another
• Perfect mechanism to support end-user/customer extensibility
• Facilitates troubleshooting/debugging
• More opportunities for automation in dev/test environments

Container Security

• Smaller footprints (fewer OS’s) means a smaller attack surface
• Vulnerabilities are inevitable
  – Visible image/container metadata – be careful
  – Image manipulation/injection concerns
Container History and Maturity

- Containers date back prior to 2009 - Linux Containers (LXC)
  - https://content.pivotal.io/infographics/moments-in-container-history
- Transitioning from Docker 1.12 to 1.13 was seamless
- Windows containers
- Competition coming from rkt on CoreOS
  - https://coreos.com/rkt

Container Standards

- https://www.opencontainers.org/
Thanks for Attending
GSAW 2017

Richard Monteleone
Sr. Systems Engineer
Satellite Ground Systems
12515 Academy Ridge View
Colorado Springs, CO 80921
rmonteleone@rtlogic.com