



Container Based Virtualization Applied

**33rd Space Symposium
Colorado Springs, CO
April 3-6 2017**

**Richard Monteleone
Sr. Systems Engineer**

Virtualized Ground System (VGS) → “The Big Picture”

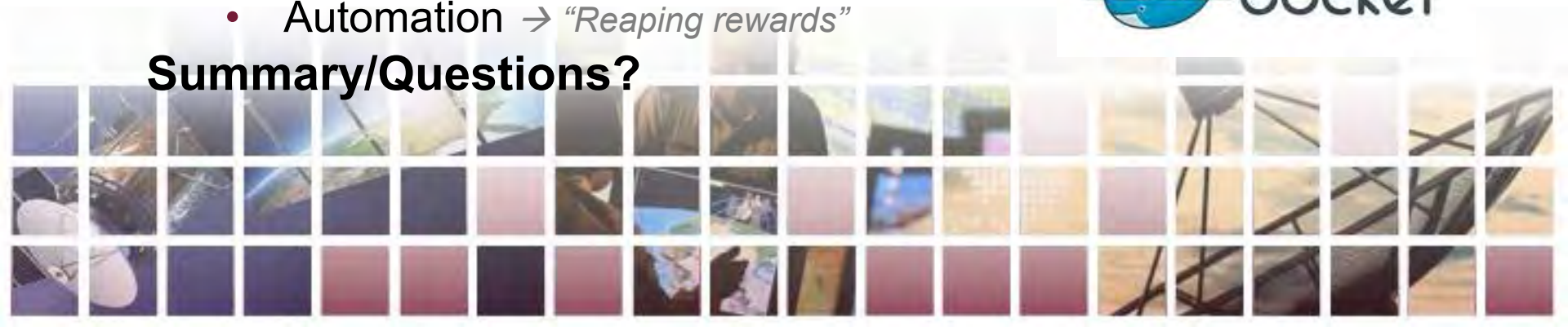
- vFEP (Virtual Front-End Processor) → “Quick look inside”
- H/W Virtualization with Virtual Machines (VM’s)
- OS Virtualization with Containers → “Let’s compare”

Container Technology Applied

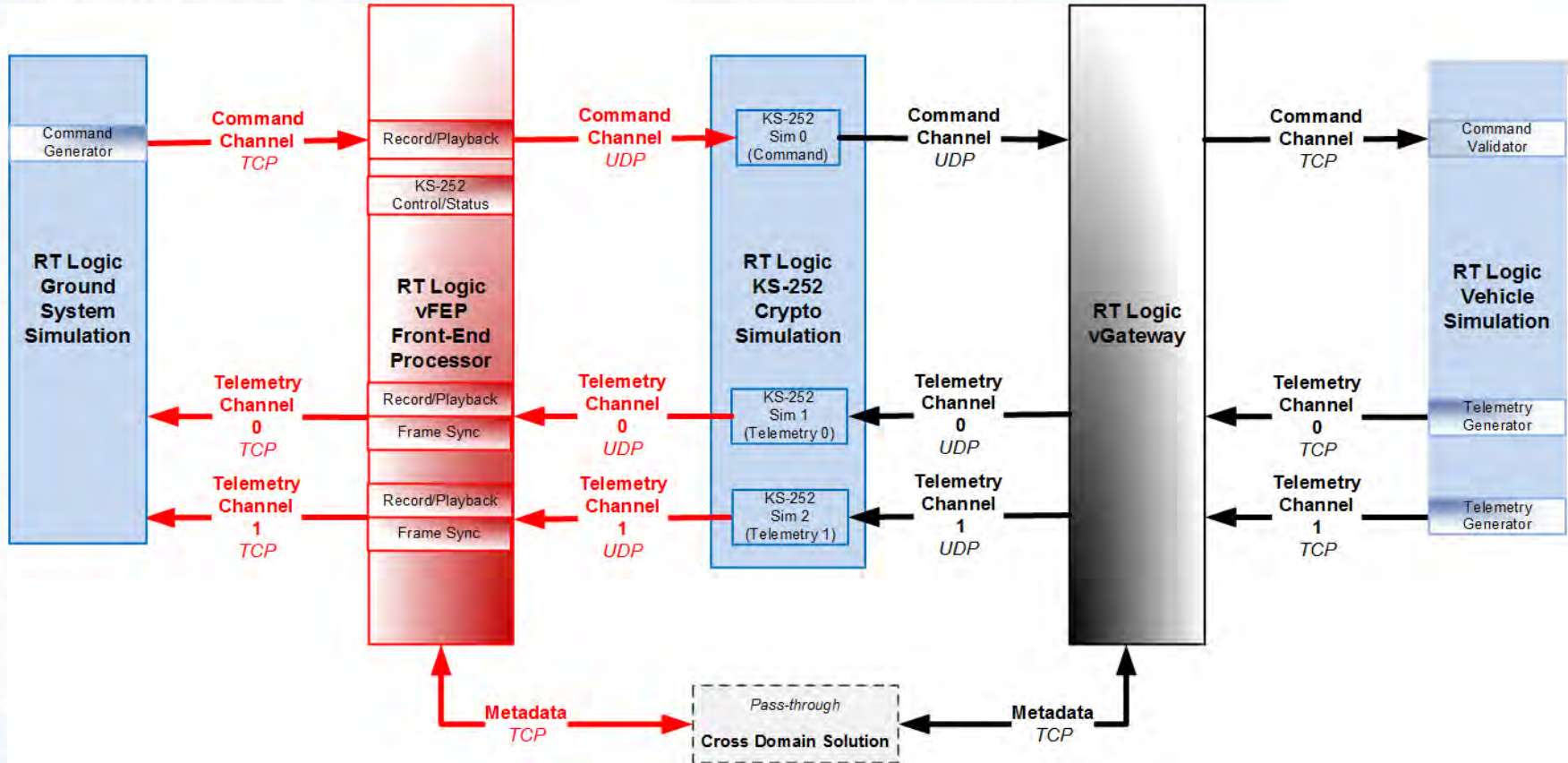
- Making the transition → “Approach taken”
- Building/deploying/running Docker containers
 - Container isolation and monitoring
- Automation → “Reaping rewards”



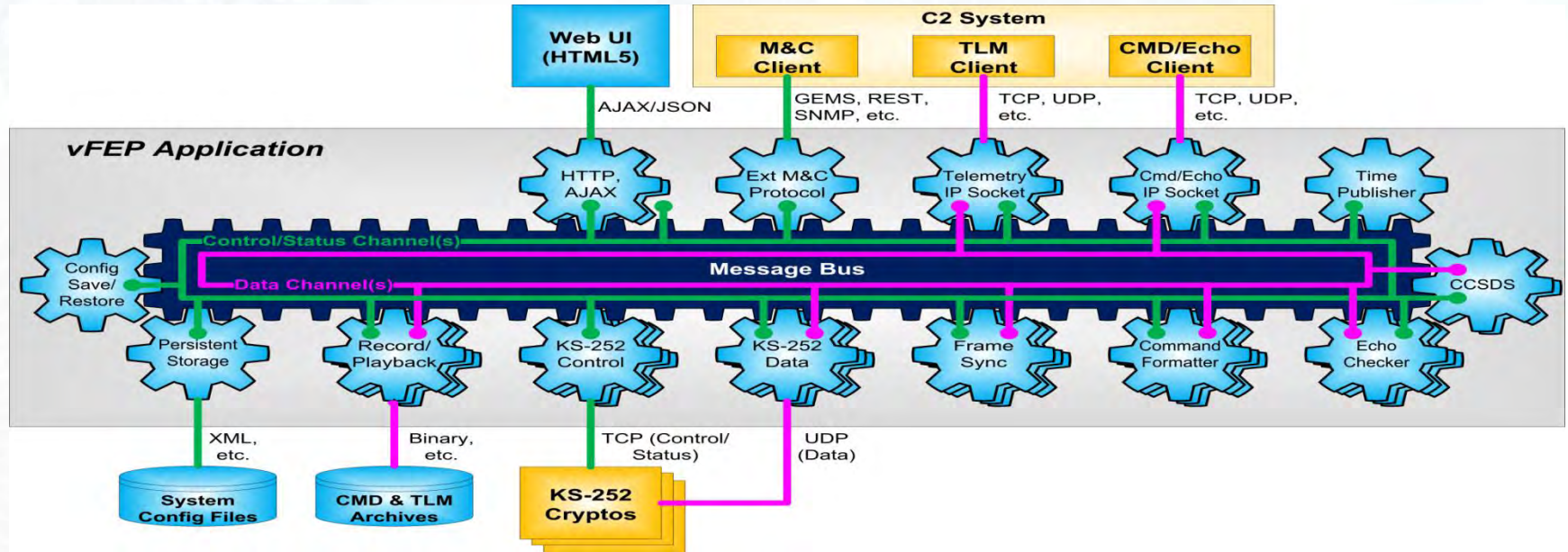
Summary/Questions?



Virtualized Ground System (VGS)

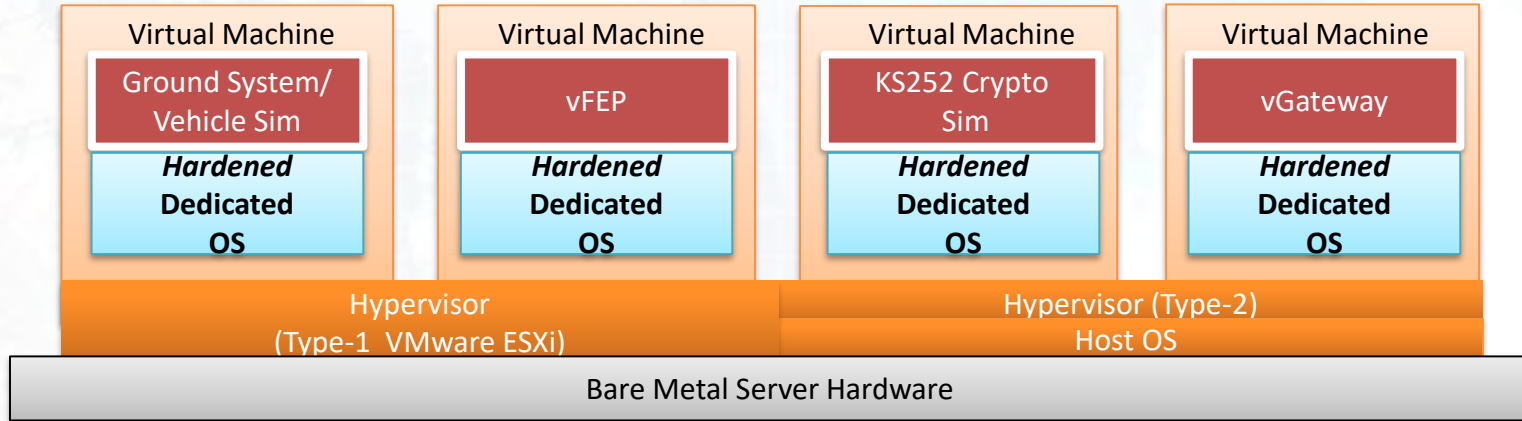


vFEP “Quick look inside”



- 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)

Hardware Virtualization – Virtual Machines



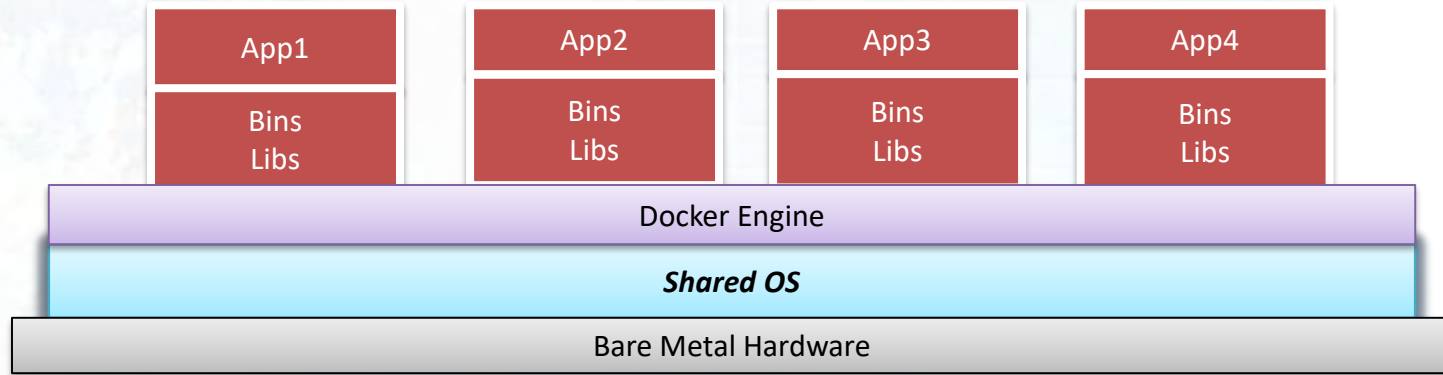
Applications installed and configured on individual VMs

- **Hardened Dedicated OS**
- Application ISO images *mounted* and installed
- Command and telemetry channels *interactively* user configured

Things are **really good now** ... could they be **even better?**

- Hardware sharing, Snapshots, vMotion, VM templates, **Isolation**, OVA's, Secure, Stable, Scalable

Operating System Virtualization - Containers



Containers: How do they differ from VM's?

- **Shared OS** for containers
 - + More resource efficient (only use what they need when they need it)
 - + Extremely lightweight, fast to start
 - + Capable of running directly on Bare Metal H/W, less H/W required
 - + No Hypervisor required - OS Kernel/Container compatibility required
 - Failures/cycling of the Docker Engine-OS-H/W are more impactful

Methodology leveraged to produce/evaluate “good” containers

<https://12factor.net/> “The 12 Factor Application”

- I. Codebase (*single purpose*)
- 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 (*easily replaceable*)
- X. Dev/prod parity (*similarity*)
- XI. Logs
- XII. Admin processes

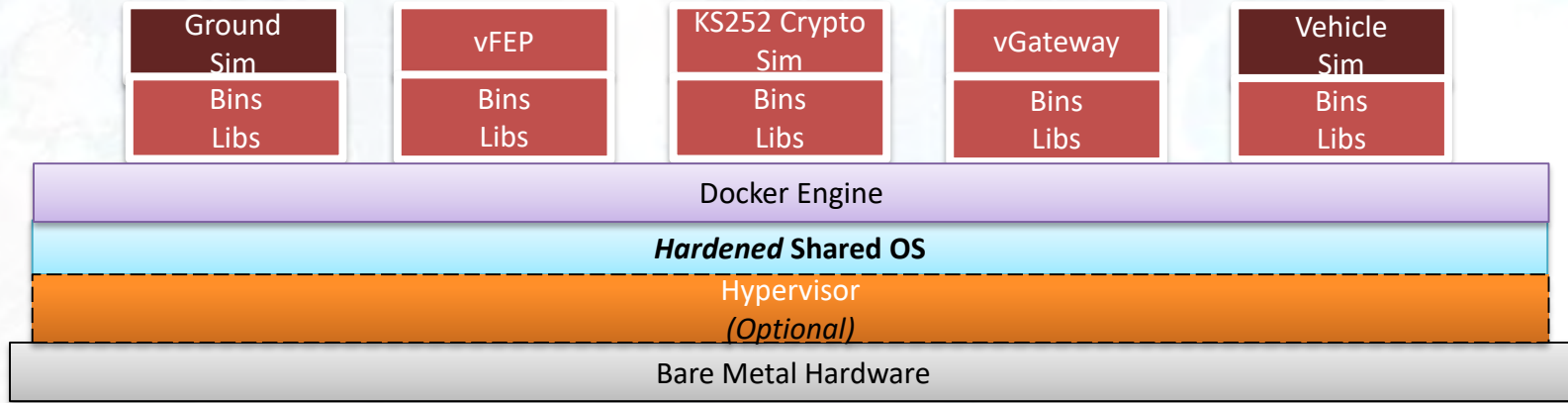


Transitioning into Containers cont'd

Changes to how we install/configure/deploy/run applications

- GS Veh Simulator (*single purpose*)
 - ✓ Multiple applications (Ground System and Vehicle Simulation)
- Don't store data within a container (*backing services and disposable*)
 - ✓ vFEP Recording/Playback of command and telemetry data
 - ✓ Storing configuration files
- Application lifetime
 - ✓ Lifetime management no longer controlled internally
- Interactive application configuration and deployment
 - ✓ Eliminate ISO mounts for application installation
 - ✓ Need to automate image building and the deployment of containers

VGS deployment with Containers



Split GS Veh Simulator into two containers

Running the Docker Engine on a **single Hardened Shared OS**

Configured Docker version 1.12 and 1.13 environments

- Optionally running the Docker Engine/Host OS in a VM
 - Leveraging both H/W and OS Virtualization Technologies
 - Increased capabilities/flexibility

Docker files create Docker images

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
 - Defined a (*single*) ENTRYPOINT to execute each container
- Removal of internal service lifetime configuration
 - Now managed with the container lifetime

Building images/running containers

Initially images are built and containers are run manually

- Built images from instructions in Dockerfile(s)
 - `docker build -t="vgs/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 vgs/vfep:1.0.1`

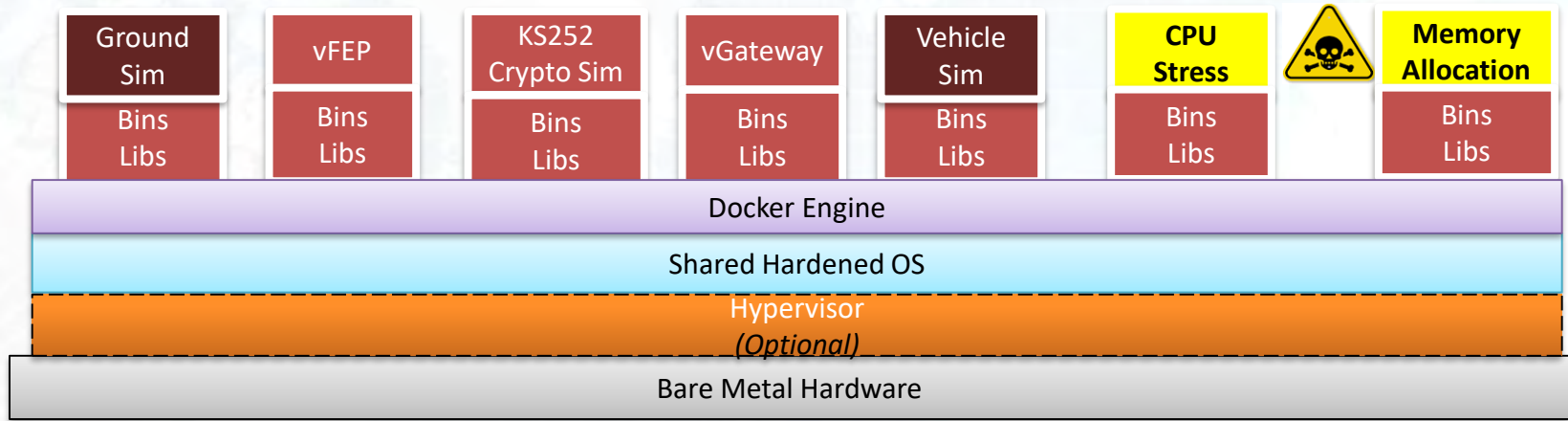
Equivalent using Docker Compose

- 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 !!!



Container Isolation



Running “*bad*” disruptive containers in the VGS

- CPU Stress container
- Memory Allocation container

Verify the VGS maintains a normal operational state

- ✓ Undisturbed by “*bad*” containers sharing the same Docker Engine/OS

What’s really going on in the container environment?

Monitoring the environment (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`

vfepA
(/docker/93acde01a75b706d795cdf0129651141539b36c56dad4d696!
Isolation



Build/Deploy/Test Automation with Containers



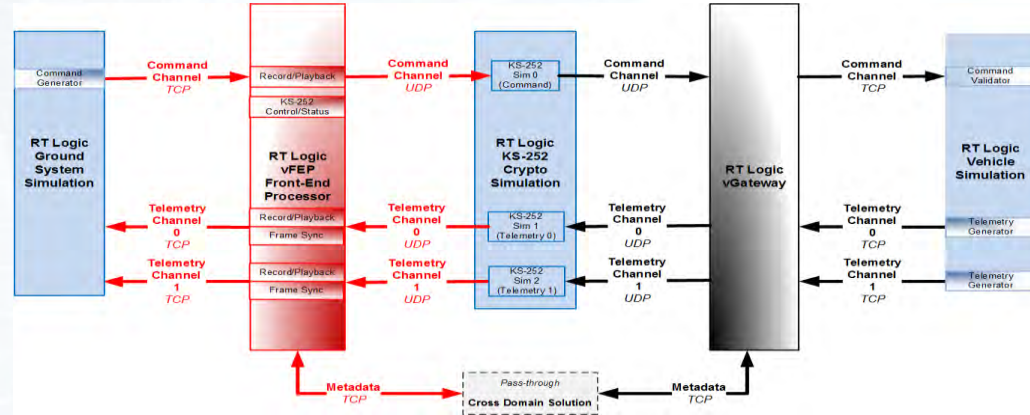
(A)

On-Demand & Nightly
Product Builds



(B)

On-Demand & Nightly
Docker Builds



(C)

Ground System Operational!!!

Container Benefits

- Application scalability
- Lightweight
 - Very fast startup, smaller in size, easily updated/distributed
- Cost reductions
 - More 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>
- Windows containers a reality
- Docker transition from versions 1.12 to 1.13 was seamless
- Competition coming from rkt on CoreOS
 - <https://coreos.com/rkt>



Container standards

- Open Container Initiative
 - <https://www.opencontainers.org>
- Open industry standards
 - Container Formats
 - Runtime



Questions?

Thank You