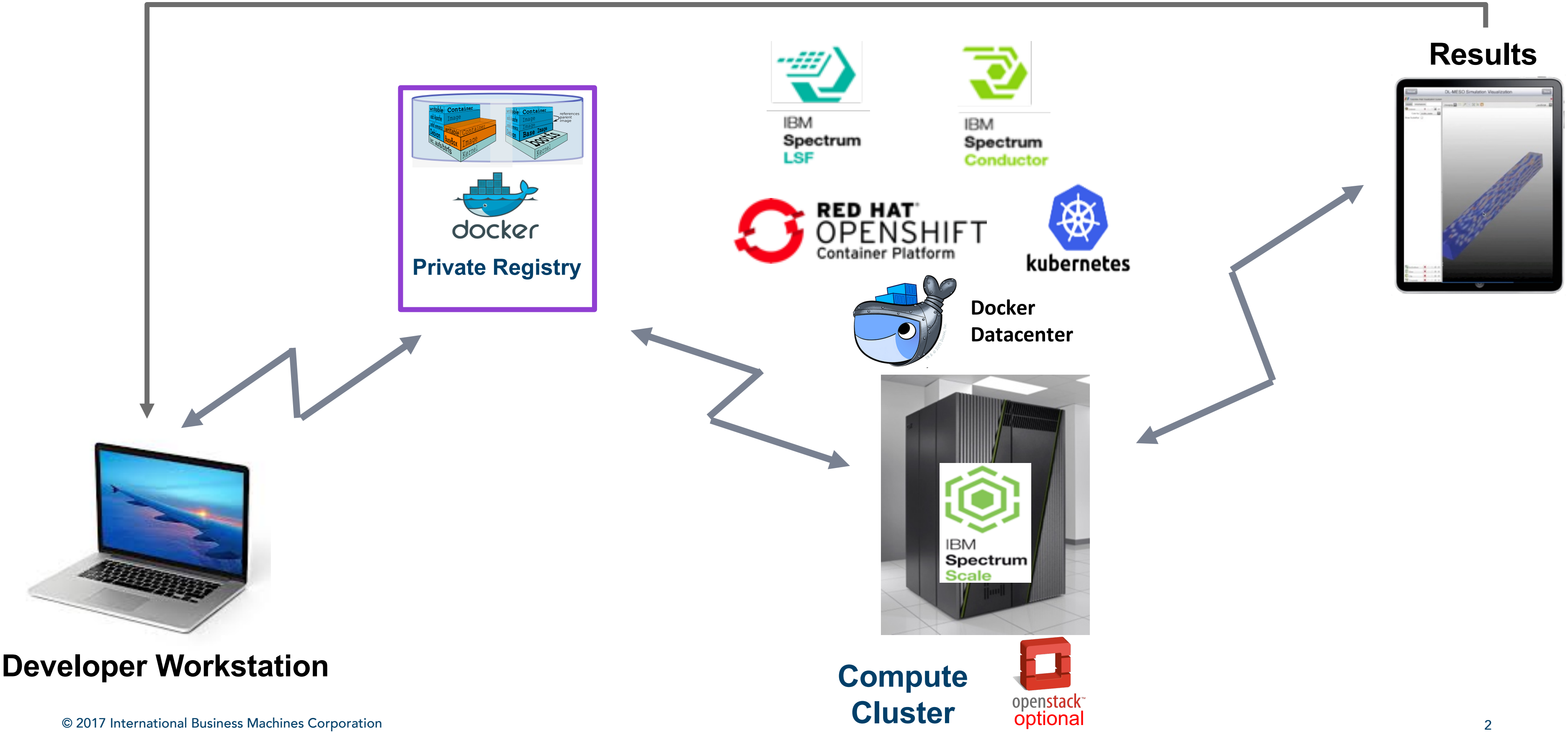
Spectrum Scale and Containers

Sandeep Gopisetty, Heiko Ludwig, Mohamed Mohamed, Robert Engel
Cloud Systems Research – Almaden

Dean Hildebrand, Amit Warke
Cloud Storage Software Research – Almaden

Spectrum Scale Container Developer Workflow



Results

docker

Private Registry

IBM Spectrum LSF

IBM Spectrum Conductor

RED HAT OPENSHIFT Container Platform

kubernetes

Docker Datacenter

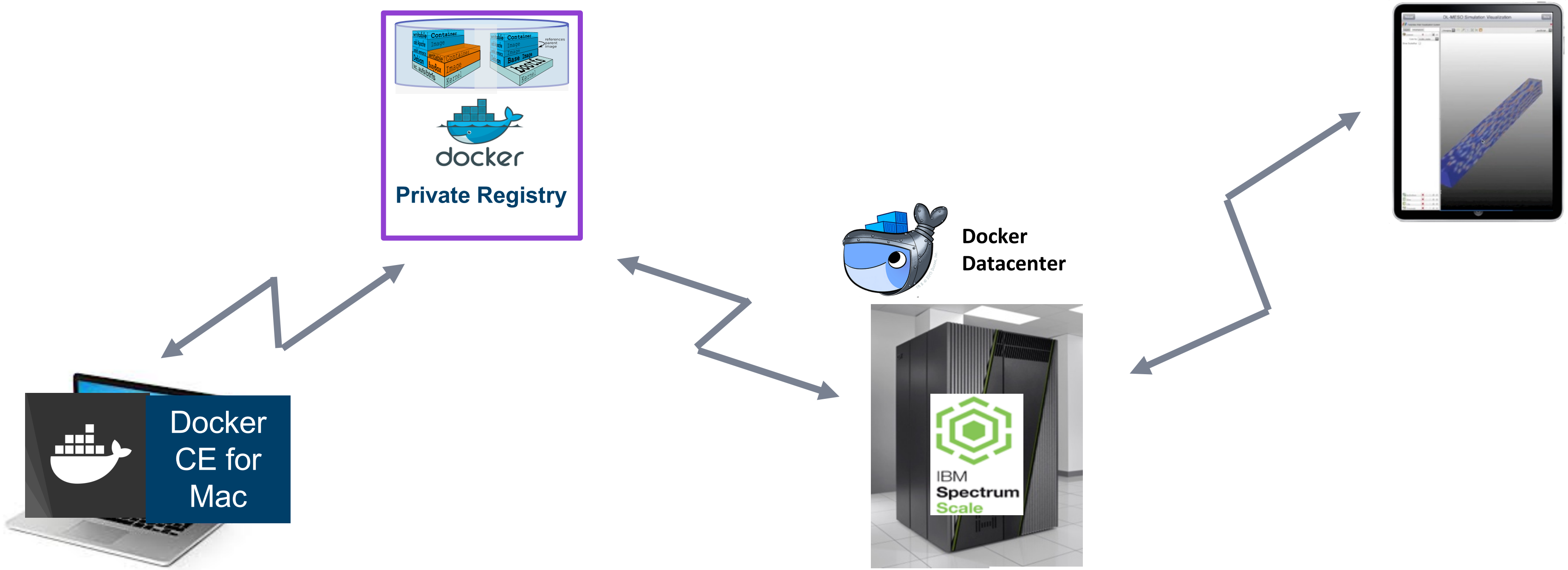
IBM Spectrum Scale

Developer Workstation

Compute Cluster

openstack optional

Spectrum Scale Container Developer Demo



Results

Private Registry

Docker Datacenter

Docker CE for Mac

Developer Workstation

Compute Cluster

VirtualBox

Containers

Self Contained (Portable)

Data Access Isolation (Multi-Tenancy)

Data Management

High Performance Data Access

Fast and lightweight

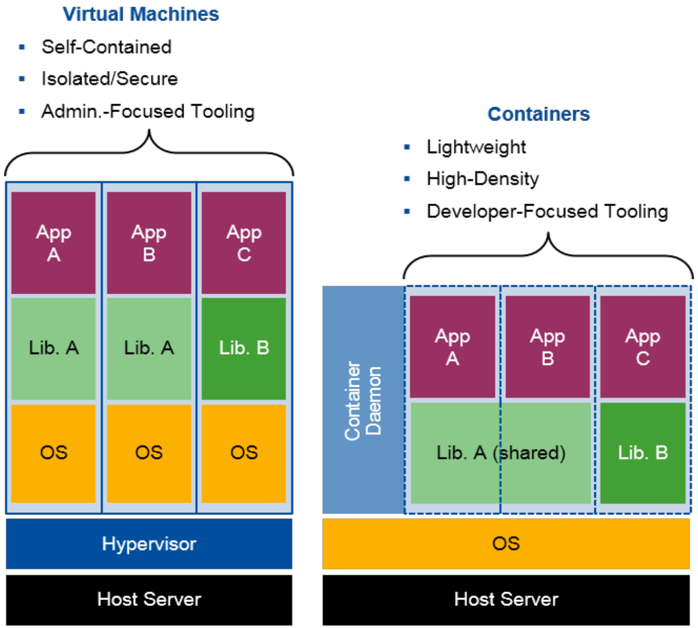
Resource Utilization

Open-Source

Global Repositories

Containers, Containers, Containers

- HPC and Scientific Computing
  - Portable and reproducible science
  - One-Click Laptop to Supercomputer
- Apps in Clouds
  - Scheduling and Auto-Scaling
  - Improved resource utilization
  - Isolation and Multi-Tenancy
- Development, DevOps and continuous integration
  - Re-use of applications and services
  - Simplify and accelerate application deployment



Virtual Machines

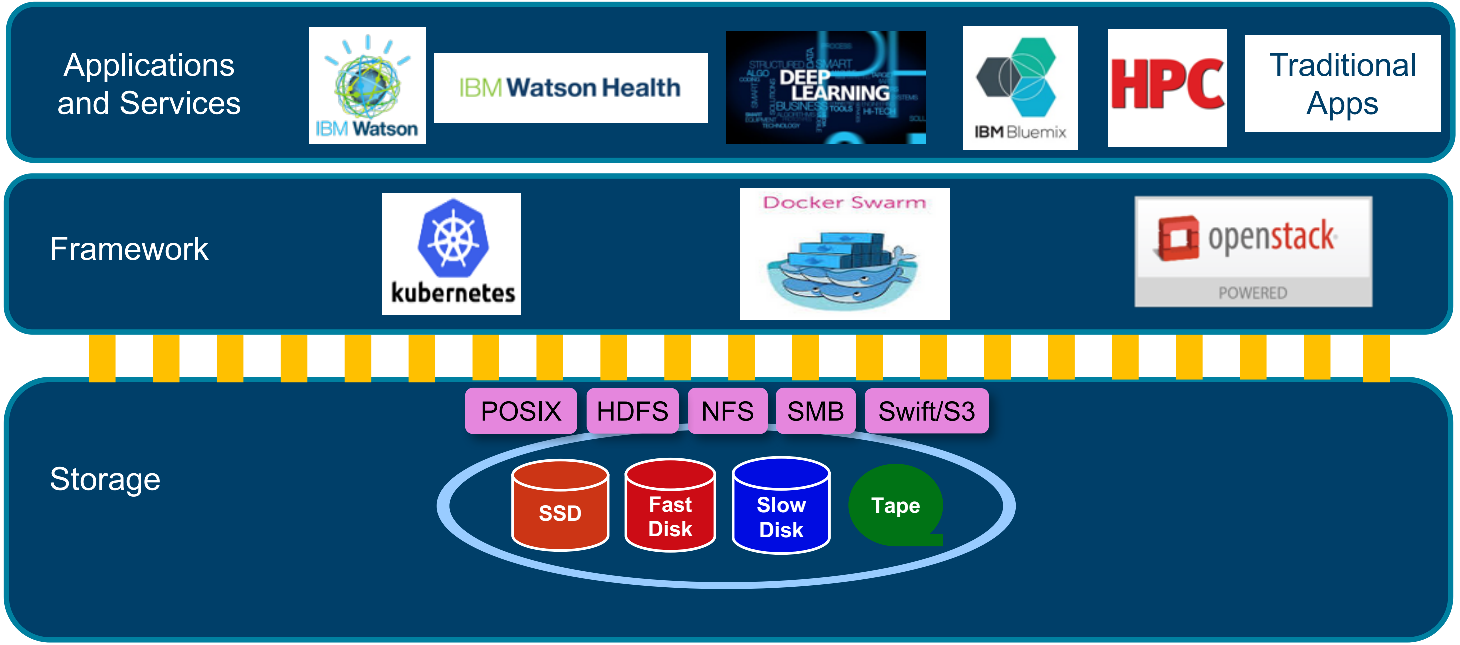
- Self-Contained
- Isolated/Secure
- Admin.-Focused Tooling

Containers

- Lightweight
- High-Density
- Developer-Focused Tooling

Next Gen, Micro-Service, and Traditional Applications

New Gen Application, Framework, and Storage EcoSystem



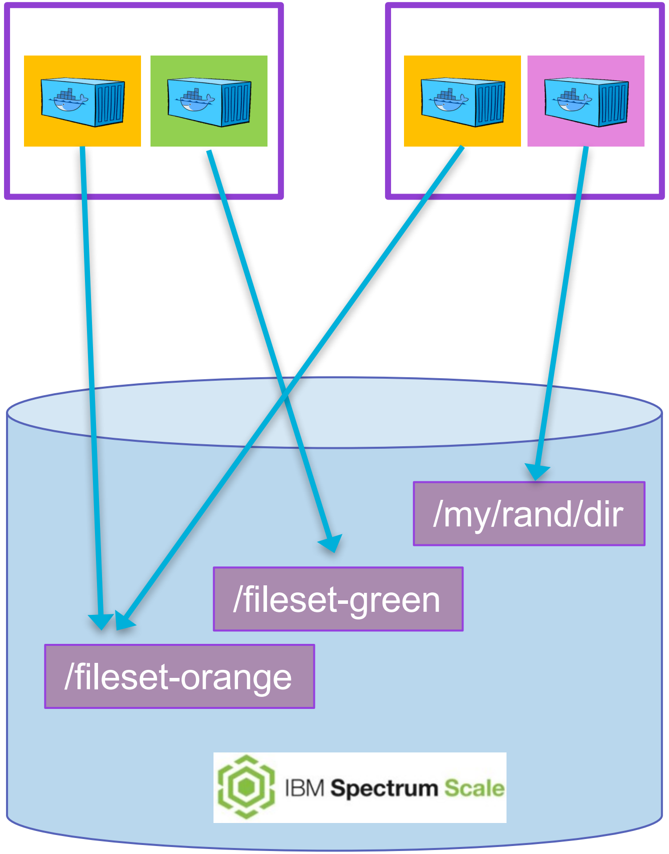
A unified storage experience across runtimes

1. Self Contained (Portable)

- Containers can package OS, libraries, app. or any other req. software
- Key requirement is the same OS kernel
  - Linux can run on Linux, Windows can run on Windows
- Configurable configurability (args passed to container)
  - Nothing
  - Additional arguments
  - External data volumes
  - Application to run
- Container lifetime can be tied to application
  - Allows container scheduler to perform H/A
- Note the difference between an Image, and a Dockerfile
  - Image – An ordered collection of changes organized in static layers
    - Guaranteed to be the same no matter where it is copied and executed
  - Dockerfile – A file containing all the commands that you would run to create an image
    - Not guaranteed to create the same image from one execution to another
      - E.g., the results of 'yum install gcc' can very based upon configuration and time

2. Data Access Isolation (Multi-Tenancy)

- Spectrum Scale commands not accessible
- Changes to image
  - Private to that image
  - Can be saved or discarded by admin
- Changes to external volumes
  - Can only access its volumes (and no other)
  - Volumes can be any file path
  - Userids can be the same in container as in FS
    - Linux user namespaces can also do mapping
  - Root can access any file 'in volume'
  - ACLs work as per usual
    - POSIX ACLs can be set from inside container
  - SELinux can be label volumes and only allow access from specific containers



3. Data Management - Volumes

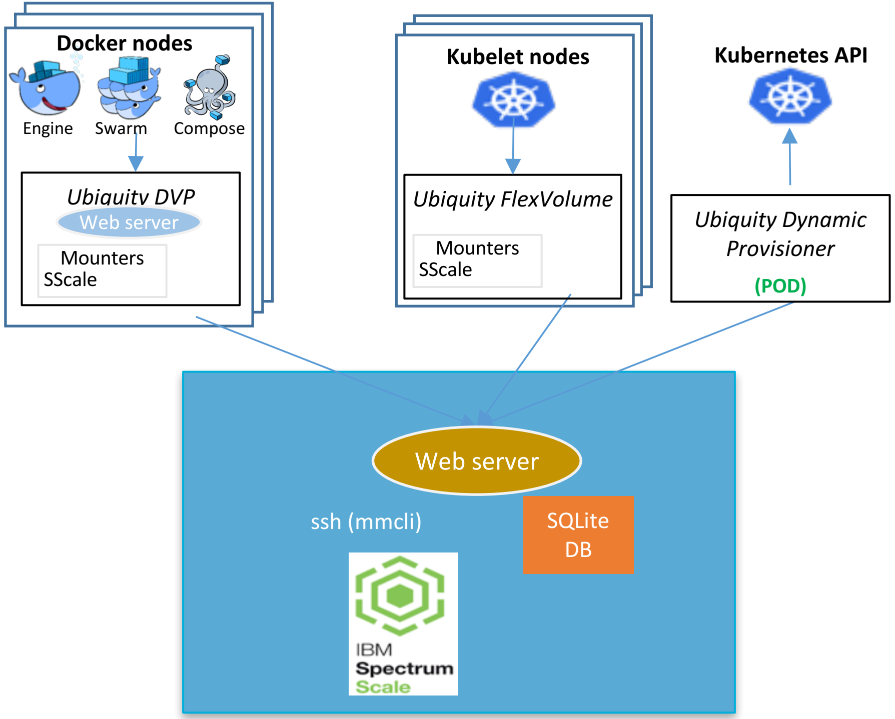
- Volume Plugins and Drivers
  - Implements storage specific parts of Docker and Kubernetes volume management commands
- Docker
  - `docker volume create/inspect/ls/prune/rm VOLUME`
  - Attach and detach GPFS and NFS volumes to hosts before/after container execution
- Kubernetes
  - Dynamic Provisioning Driver
    - Manage Persistent Volumes (PVs)
      - kubectl get pvc/pv
      - kubectl create -f claim-01.yaml
    - Creation is Asynchronous - Scans for new “Persistent Volume Claims (PVCs)” and create if required
  - FlexVol Plugin
    - Attach and detach GPFS and NFS volumes to hosts before/after container execution

3. Data Management – Changing How Users Consume Storage

- Storage allocation can be managed by Docker/K8s or CI/App admin
- Users can identify datasets by *name (no need for file path)*
  - Examples
    - Identify input read-only datasets that are shared
    - Identify storage space allocated to a user
- Docker typically assumes admins to manage volumes, but then allows users to use the created volumes (but DDC probably enhances)
- K8s has a much more sophisticated storage strategy
  - Users decide if storage is retained, recycled, or removed upon container completion
  - Users can declare the type of storage required, and it can be mapped to existing (or new) Spectrum Scale storage types
  - K8s (and OpenShift) support a range of roles

Ubiquity Storage Volume Driver

- Initially support 2 types of volumes:
  - Fileset volumes
    - Support optional quota and setting Linux userid/group permissions
    - Support both independent or dependent filesets
  - Lightweight volumes
    - Practically no limit
    - Implemented as individual subdirectories in a fileset
- Current mechanisms can set other features
- Can map existing dirs/filesets into Volumes
- Currently GPFS and CES NFS
- Can use ‘ssh’ to call
- Planned Items
  - Run Ubiquity service in a container
  - Support Spectrum Scale REST-API
  - Support additional options for Spectrum Scale features
  - Add in support for IBM Block Storage



Ubiquity Storage Volume Driver

- Initially support 2 types of volumes:

V0.1 Now Available at

Ubiquity Service	https://github.com/ibm/ubiquity
Ubiquity Docker Plugin	https://github.com/IBM/ubiquity-docker-plugin
Ubiquity K8s DP and FV	https://github.com/IBM/ubiquity-k8s

Available as an alpha release to gain experience with users and their use cases
Support on a best effort basis
Research code

- Add in support for IBM Block Storage