

Intro to Docker



© 2018 by Innovation In Software Corporation

INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

Data Persistence

© 2018 by Innovation In Software Corporation



Agenda

- Docker Container Volumes
- Volume Types (bind:managed)
- Volume Use Cases
- Storage Drivers
- Metadata



Docker Images - UFS

Union Filesystem Limitations:

- Read/Write w/ Copy On Write slows file system down
- Data not easily
 - Shared between containers
 - Imported to containers
 - Off-loaded from Docker Host
 - Backed-up



Docker Container Volumes

© 2018 by Innovation In Software Corporation



Discussion

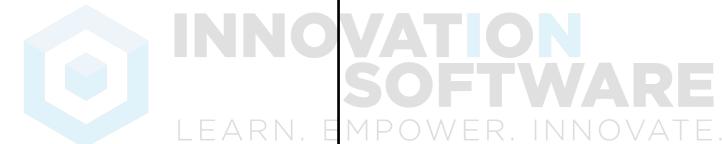
What are some examples where Container Data Persistence might be required?



Discussion

What are some examples where Container Data Persistence might be required?

- Where Data Persistence is important:
 - Database Records (e.g. MySQL, MariaDB, Cassandra)
- Shared data from host to containers
 - build tools (e.g. Jenkins jobs)
 - automation tools (e.g. Ansible playbooks)
- Share data between containers
 - Container-A creates data, Container-B uses created data
- Access to Docker host's data
 - /var/log, /etc/hosts, /var/lib/, etc



Docker Volume Types

Bind Mount Volume

Docker Managed Volume

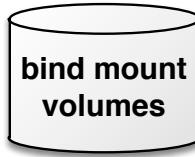
© 2018 by Innovation In Software Corporation



Volume Types

Bind Mount Volume

Docker Managed Volume



Provides Data:

- Accessibility by container and non-containers
- Persistence beyond a container's lifecycle
- Sharing between Containers
- Segmentation between Containers

Bind-mount volumes

Bind Mount Volume

Docker Managed Volume



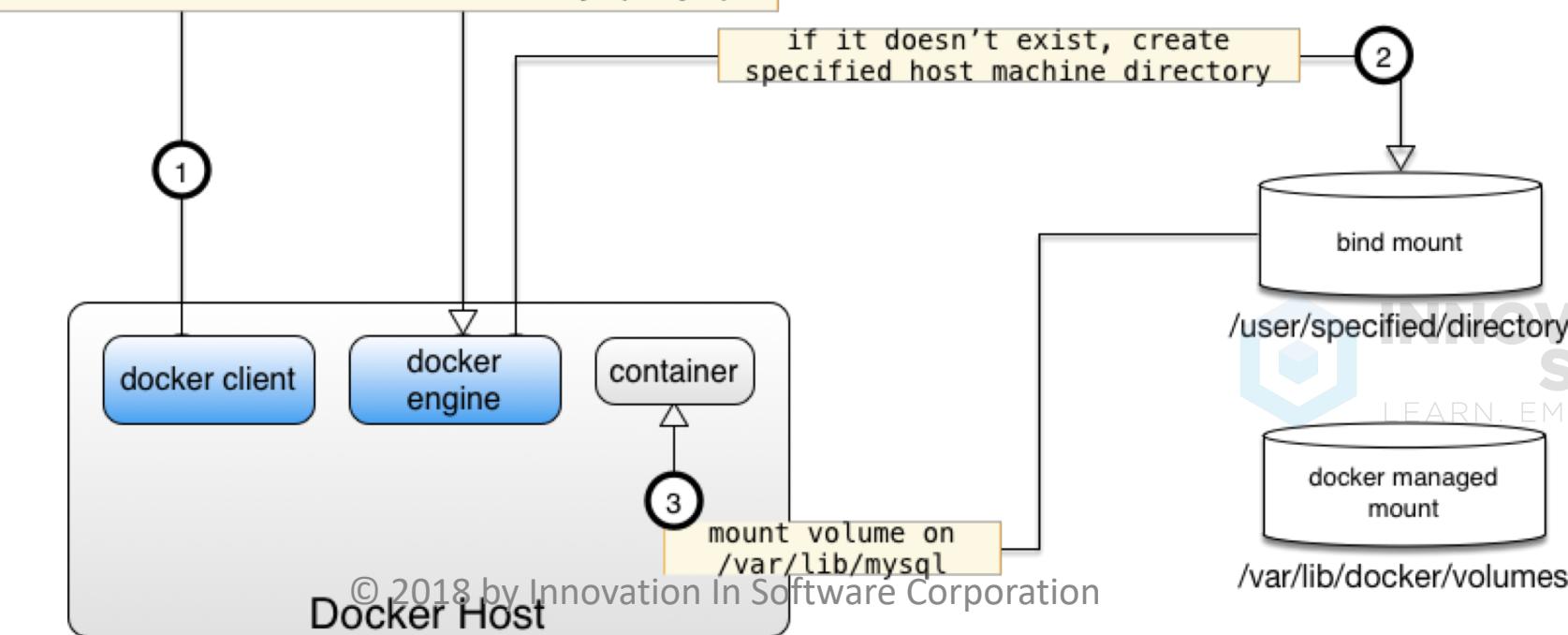
- User Defined
- Provides Data:
 - Accessible by containers and non-containers
 - Creates the directories, if required, in the volume path
 - Absolute paths must be used
 - Can not be automatically cleaned up by Docker-Engine
 - Managed by User

Bind-mount volumes

Bind-Mount Volume:

- user defined
- resides outside the Docker-managed space
- can be shared between many containers

```
docker run -itd -v /src/dbase:/var/lib/mysql mysql
```



Create a bind-mount volume

Create a bind-mount volume:

- Creates the /data, if it doesn't exist, mounts on container

```
$ docker run -itd -v /data:/data busybox top
```

Mounts are found in `docker inspect` output:

```
$ docker inspect $(docker ps -lq)
"Mounts": [
    {
        "Source": "/data",
        "Destination": "/data",
        "Mode": "",
        "RW": true
    }
]
```



INNOVATION
SOFTWARE
N. EMPOWER. INNOVATE.

Create multiple bind-mount volumes

Create a bind-mount volume:

- Creates the directory, if it doesn't exist, mounts on container

```
$ docker run -itd --name share \
-v /peanut:/peanut -v /butter:/butter \
-v /jelly:/jelly busybox top
```

Mounts are found in `docker inspect` output:

```
$ docker inspect $(docker ps -lq)
"Mounts": [
    {
        "Source": "/peanut",
        "Source": "/butter",
        "Source": "/jelly",
    }
```



INNOVATION
SOFTWARE
N. EMPOWER. INNOVATE.

Create a bind-mount volume: new verbose method

Create a bind-mount volume using the new --mount option:

```
Docker run --itd --mount 'type=bind,src=/data,dst=/data' busybox top
```

The --mount option was originally only for swarm services but has been enabled for stand alone containers. The order of the arguments does not matter and it is easier to understand with the variable names. This is the suggested method for new docker containers by the official docker docs.



**INNOVATION
SOFTWARE**
LEARN. EMPOWER. INNOVATE.

Docker managed volumes

Bind Mount Volume

Docker Managed Volume



- Docker Defined Mount Point
- Provides Data:
 - Accessible by container and non-containers
 - Creates the volume in /var/lib/docker/volumes/
 - Can be automatically cleaned up by Docker-Engine
 - Can be queried for orphaned volumes
 - Managed by Docker



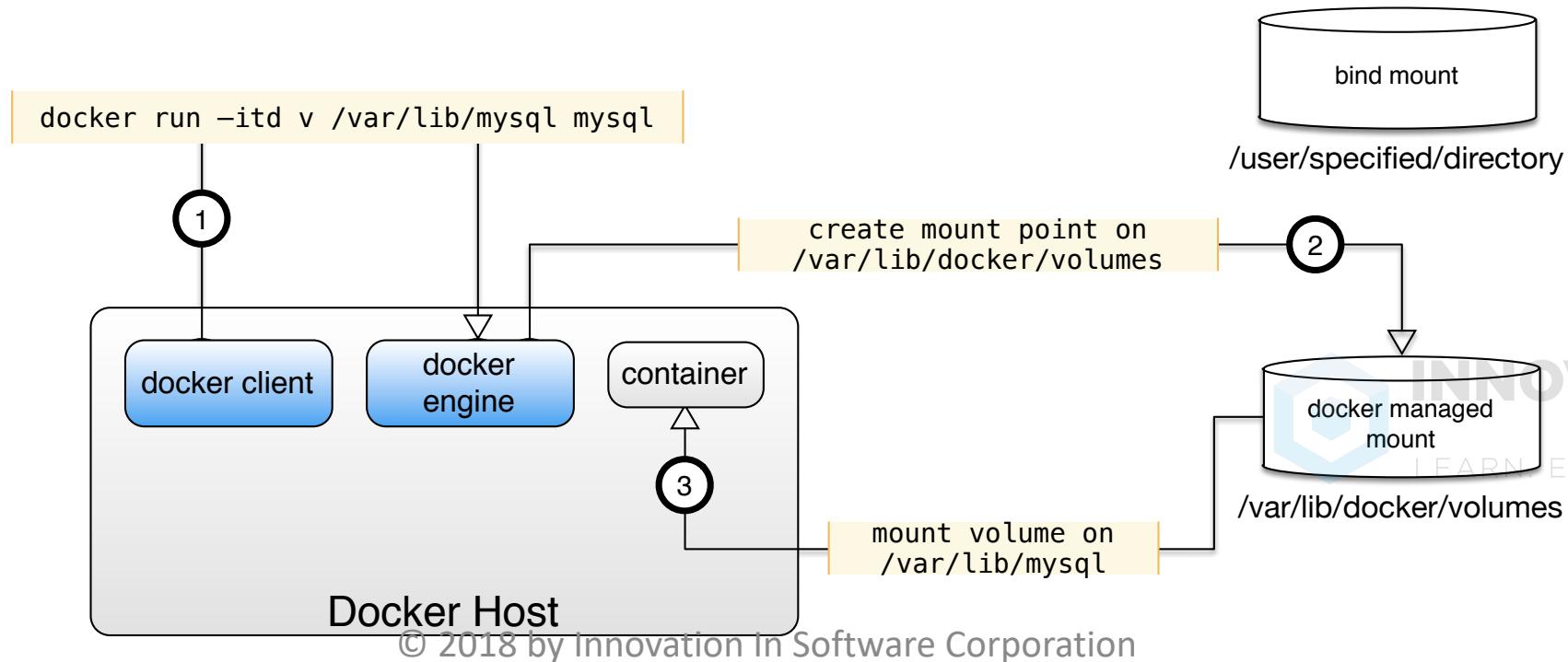
```
$ docker run -itd -v data:/data busybox top
```

```
$ docker run -itd -v /data busybox top
```

Managed Volumes

Managed Volume:

- defined by docker-engine
- resides inside the Docker-managed space
- can be shared between many containers



Docker Volume– Use Cases



© 2018 by Innovation In Software Corporation



Docker Volume Use Cases

- Share Mount Points
- Container Troubleshooting
- Build Software on Mount Points
- Load Data from Images
- Data Protection



Scenario 1: Sharing Mount Points

Sharing Container mount points:

1. Copies the source container's mount-point endpoints
2. Can be viewed in the volume's metadata

Why use this?

- Sequential builds of content in the mount-point
- Sequential access to the content of the mount-point
- Data injection from a container of the mount-point
- Provides an easy method to migrate database
- Troubleshooting



Share Docker Volumes

Create volume mounts from a container:

- Creates the directory, if it doesn't exist, mounts on container

```
$ docker run -itd --volumes-from share \
busybox top
```

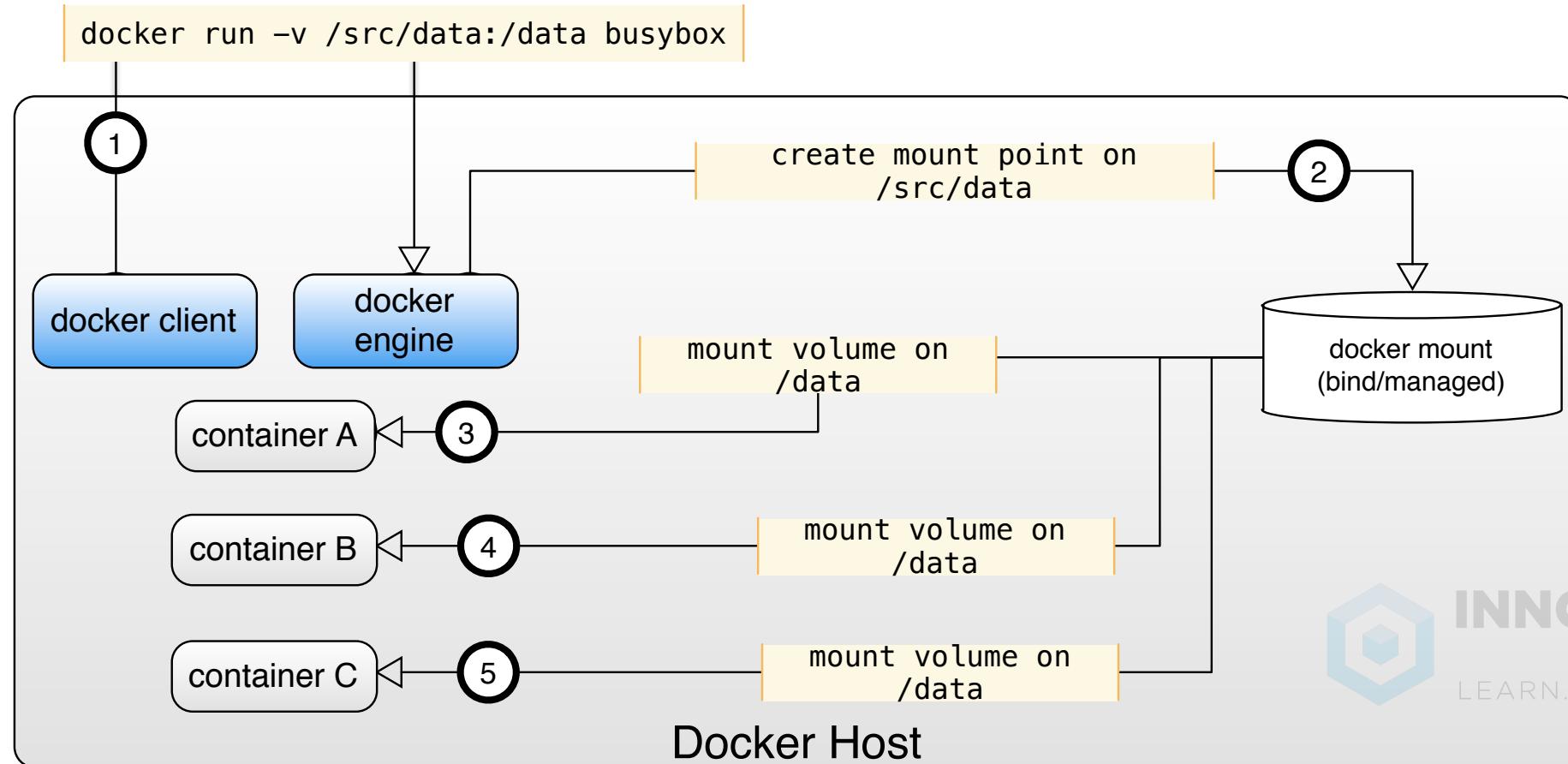
Mounts is in docker inspect output:

```
$ docker inspect $(docker ps -lq)
"Mounts": [
    {
        "Source": "/peanut",
        "Source": "/butter",
        "Source": "/jelly",
    }
]
```



INNOVATION
SOFTWARE
N. EMPOWER. INNOVATE.

Scenario 1: Sharing Mount Points



Scenario 2: Container Troubleshooting

Troubleshooting containers using Volumes:

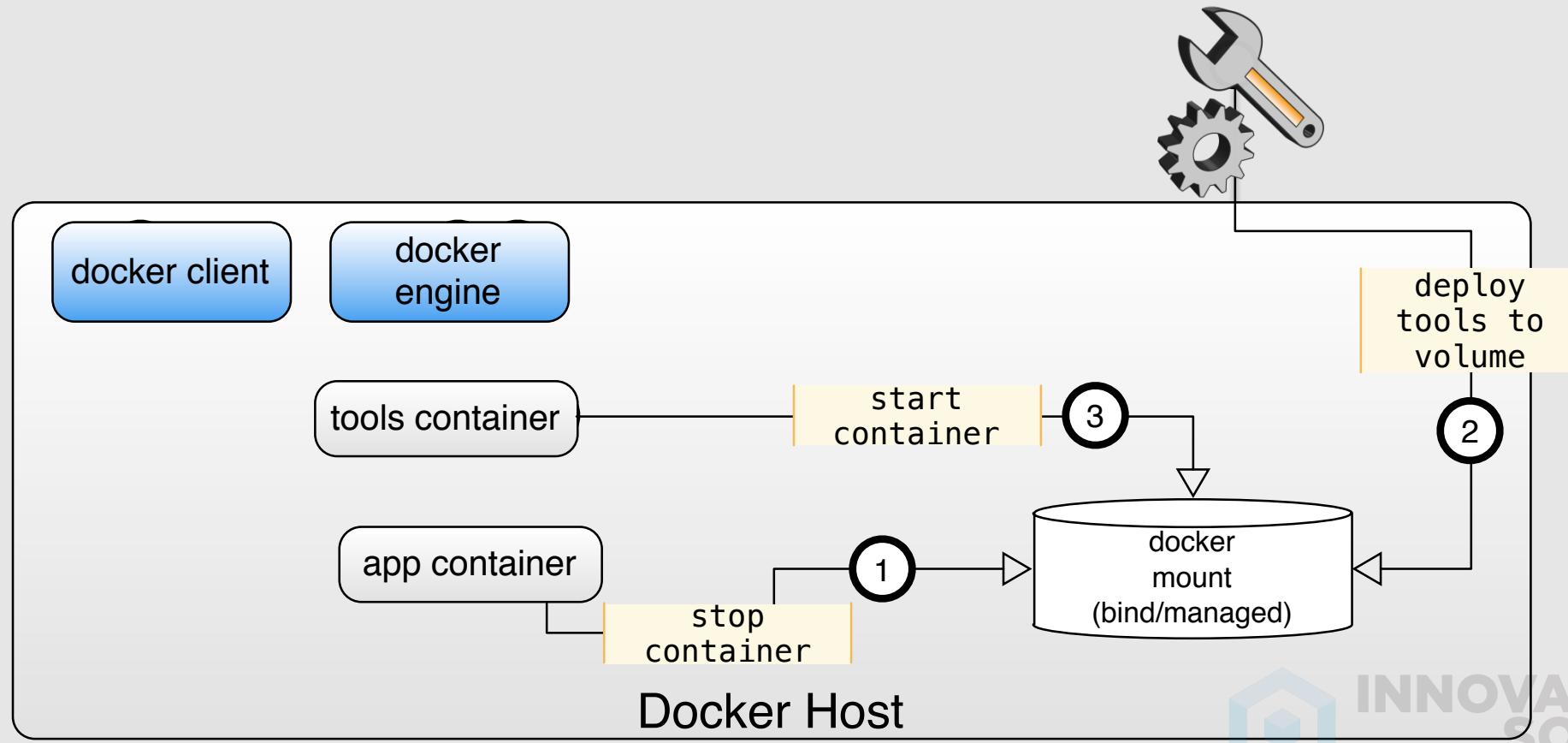
1. Stop the troublesome container
2. Copy tools into the volume
3. Run the original container or a second container

Why use this?

- Reduces the size of the app container's base image
- Allows the tools container access to the app's data
- Troubleshooting is made easier as additional tools don't need to be installed in the app's base image



Scenario 2: Container Troubleshooting



INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

Scenario 3: Build Software on Mount Points

Building on volume mount-points:

1. First Build Container installs the application
2. Second Build Container configures the application
3. Third Container runs the application's software

Why use this?

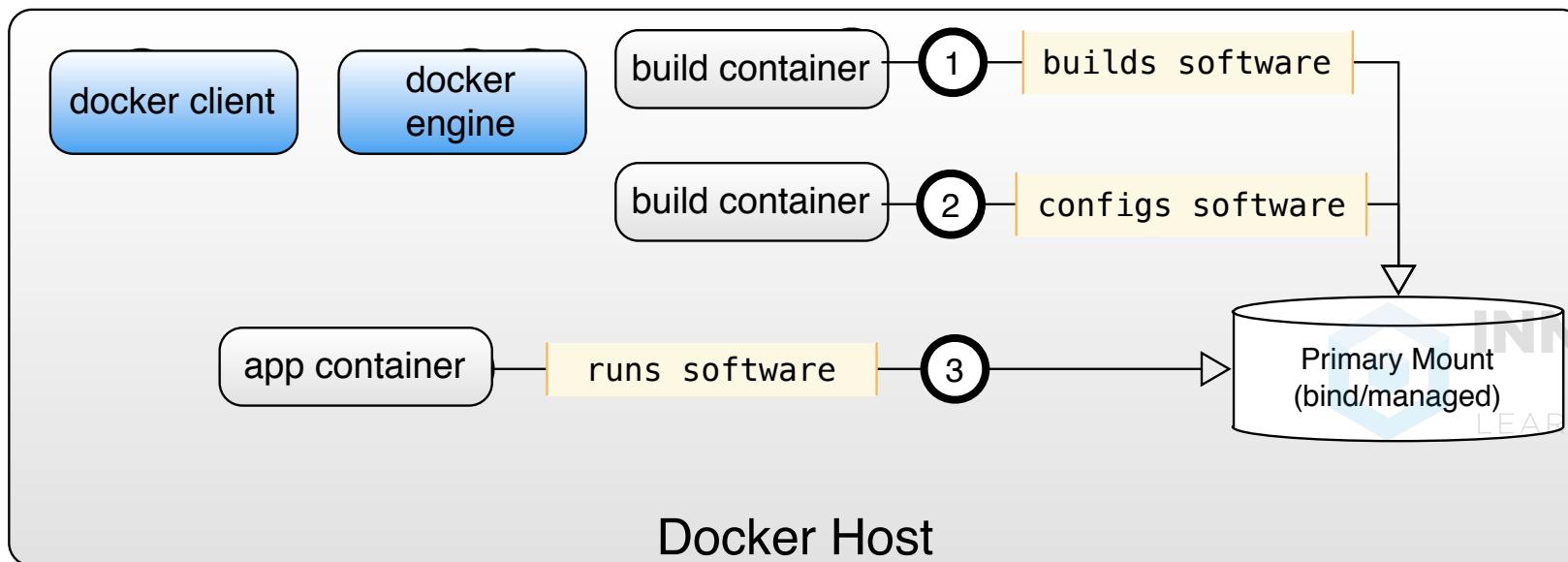
- Reduces the size of the app container's base image
- Provides a common software build
- Provides multiple configurations on the same base install



Scenario 3: Build Software on Mount Points

Using Volumes to build software:

- Multiple containers can interact with volumes
- Multiple applications can access the same data
- Containers can be aware of {{variables}} from previous build containers



Scenario 4: Load Data from Images

Load Data from Data Packed Containers:

- 1.** Images have a consistent start point
- 2.** Copy tools into the volume
- 3.** Run the original container or a second container

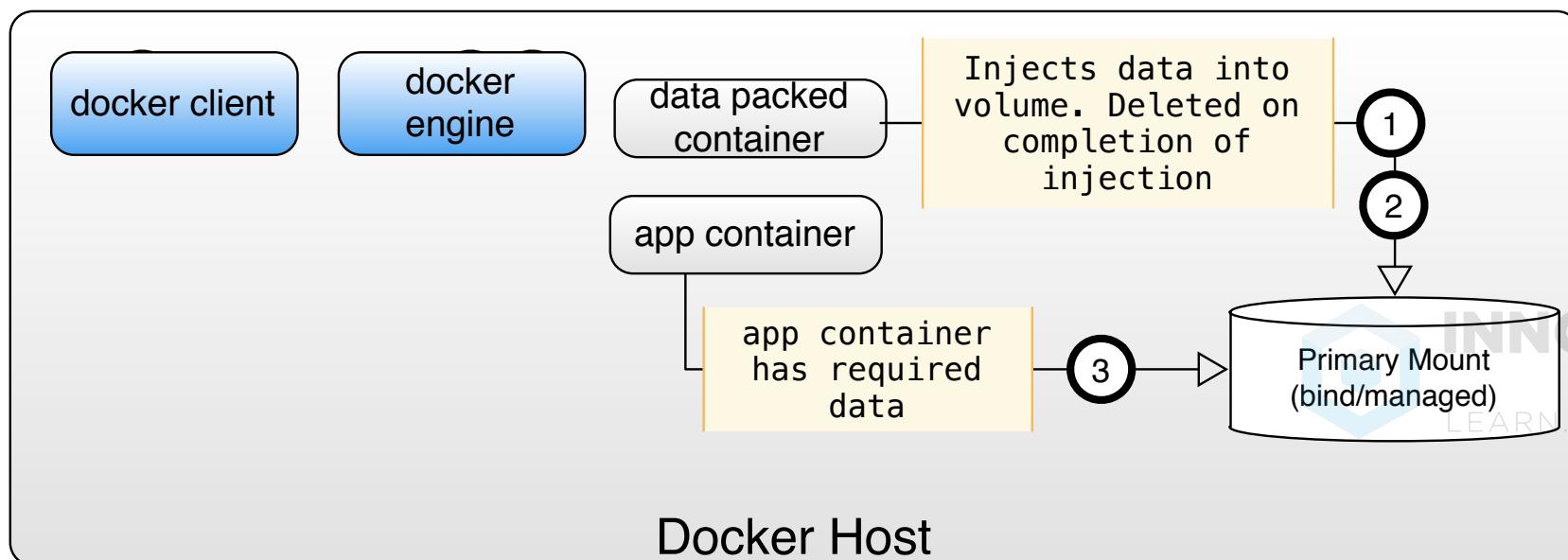
Why use this?

- Reduces the size of the app container's base image**
- Reusable consistent data from image**
- Reduce overall image storage consumption**
- Offload data writes to shared storage**



Data packed container:

- Data Packed Container has data on images to inject
- Data Packed Container injects data into volume
- App Container is deployed and has injected data in volume
- Data Packed Container is automatically deleted



Scenario 5: Data Protection

Data Protection of Volumes:

1. Volumes can be backed up by regular host utilities
2. Volumes can be added to scheduled back ups for host
3. Volumes can be restored

Why use this?

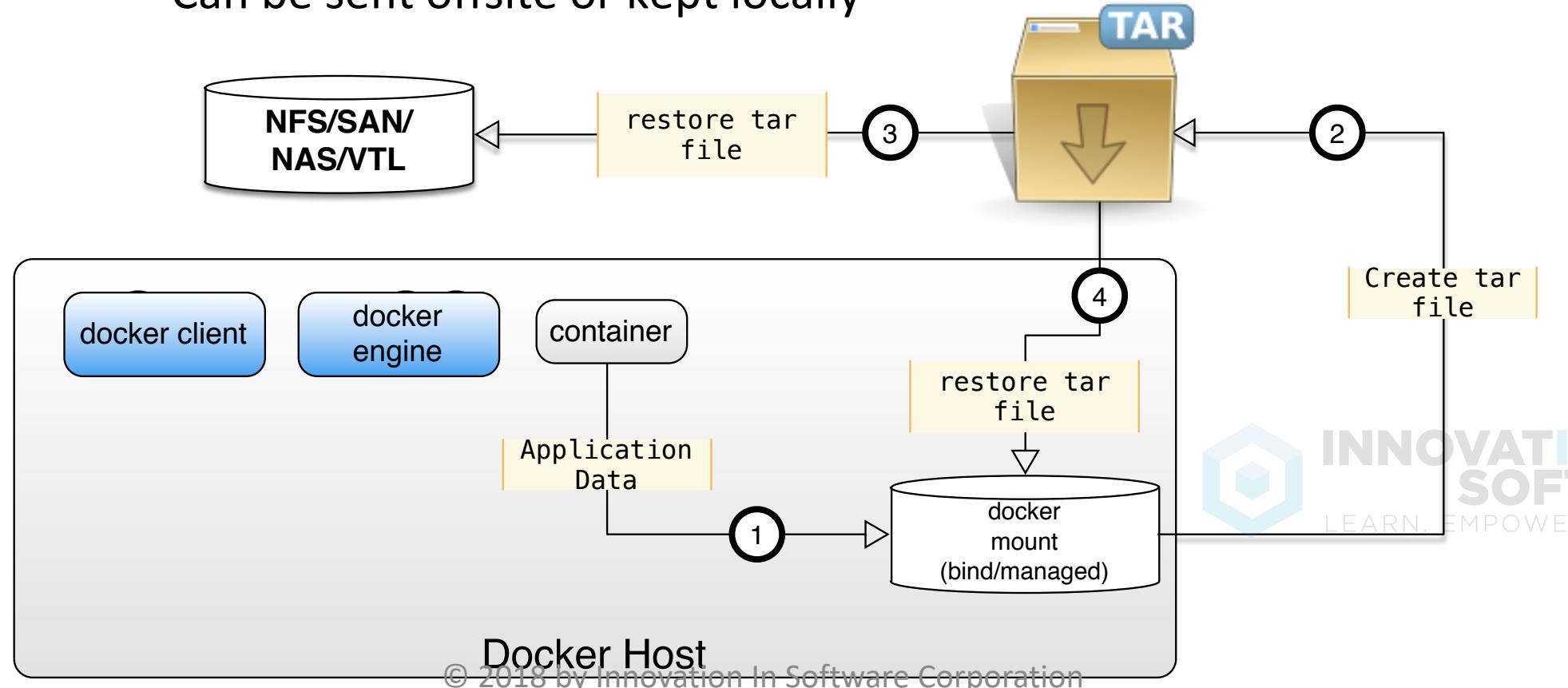
- App Data has:
 - a defined Recovery-Time-Objective
 - a defined Recovery-Point-Objective
 - regulatory requirements (e.g. SEC, BASEL)
 - importance to business functions



Volume backup using TAR

Data Protection using TAR:

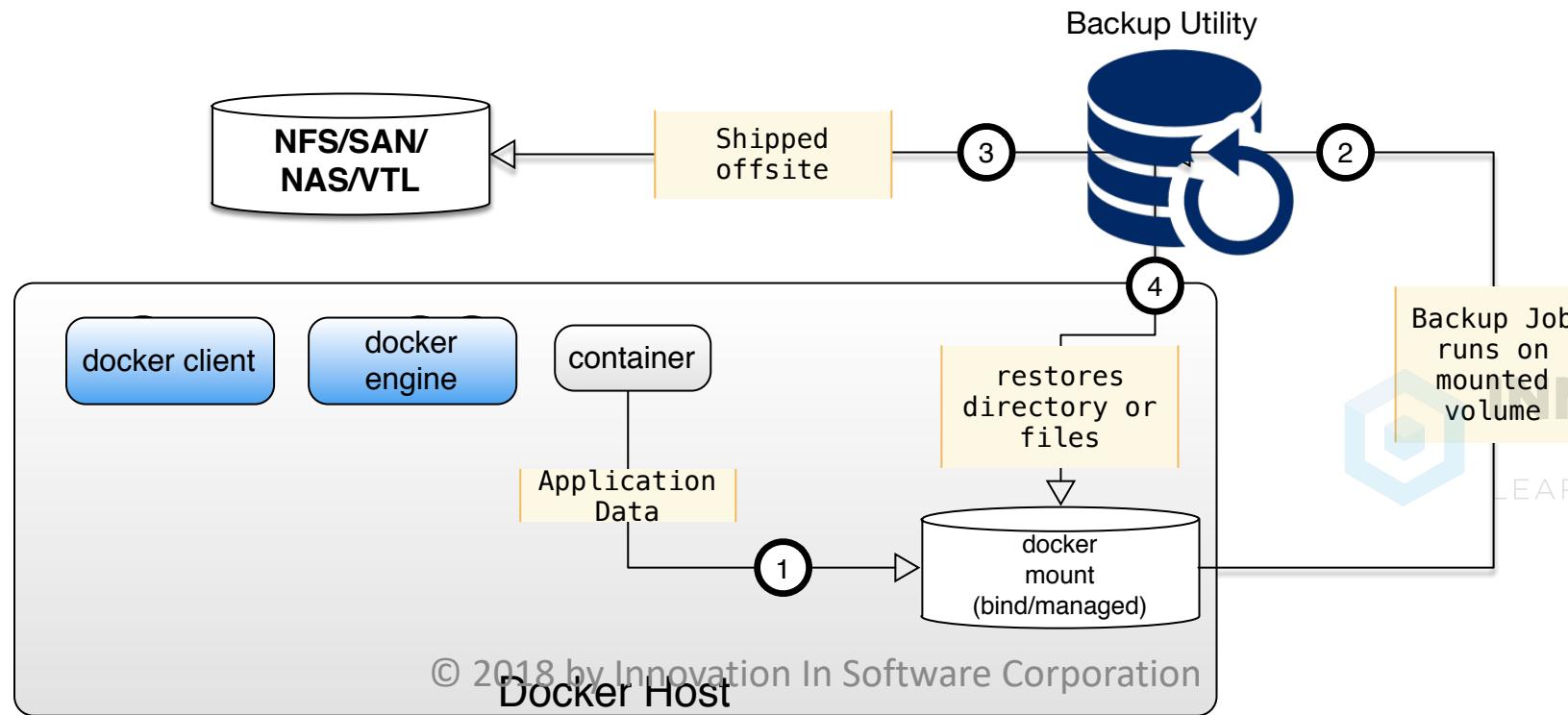
- Creates TAR of docker host's bind-mount volume (dir)
- Can be sent offsite or kept locally



Volume protection using backup utility

Data Protection using Backup Utility:

- mount-point added to backup schedule
- Backups stored locally and shipped offsite
- Backup Utility provides recoveries



Orphaned Volumes

Cleaning up volumes

- Attempts to clean up a volume

```
$ docker rm -v <container>
```

- If volume is in use, the volume is not deleted

Identify for dangling volumes:

```
$ docker volume ls -f dangling=true
```

Orphaned Volume:

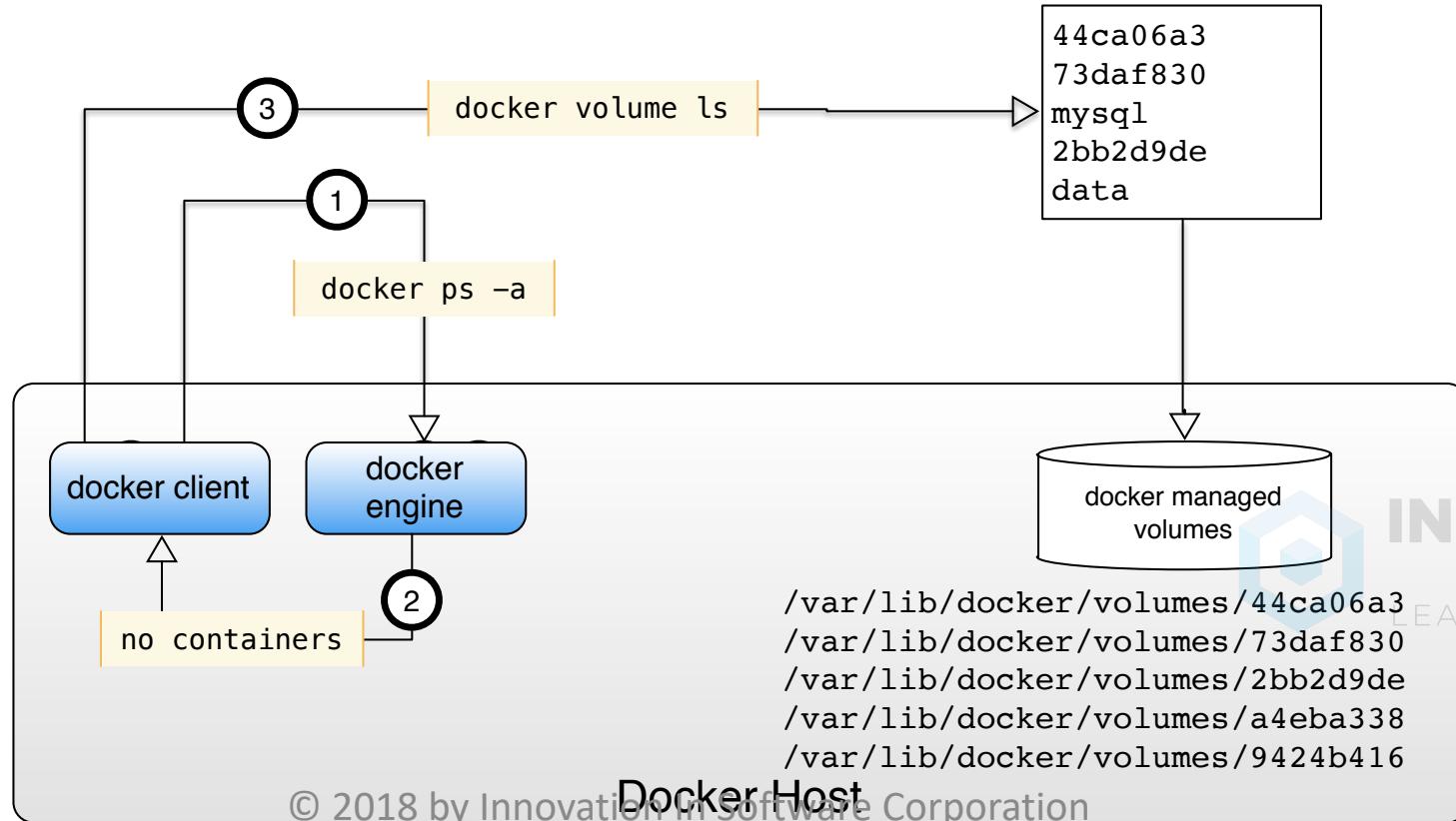
- Consumes storage space
- Data is no longer being used by a container
- Data may still be needed



Dangling Volumes

Dangling Volume:

- Orphaned volume
- Data is no longer being used



Docker Storage Drivers

© 2018 by Innovation In Software Corporation



Docker Storage Drivers

Driver	Common on	Name	Disabled on
OverlayFS OverlayFS2	ext4 / xfs	overlay	btrfs aufs overlay overlay2 zfs eCryptfs
AUFS	ext4 / xfs	aufs	btrfs aufs eCryptfs
Device Mapper	direct-lvm	devicemapper	N/A
Btrfs	btrfs only	btrfs	N/A
VFS*	debugging only	vfs	N/A
ZFS	zfs only	zfs	N/A



More info: <https://docs.docker.com/engine/userguide/storagedriver/selectadriver/>

© 2018 by Innovation In Software Corporation

Current Storage Driver

- Locate with `docker info` command
- Listed under “Storage Driver”

```
# docker info
Containers: 16
Images: 38
Server Version: 1.11.2
Storage Driver: overlay
... SNIP ...
```



AUFS

AUFS

Device Mapper

BTRFS

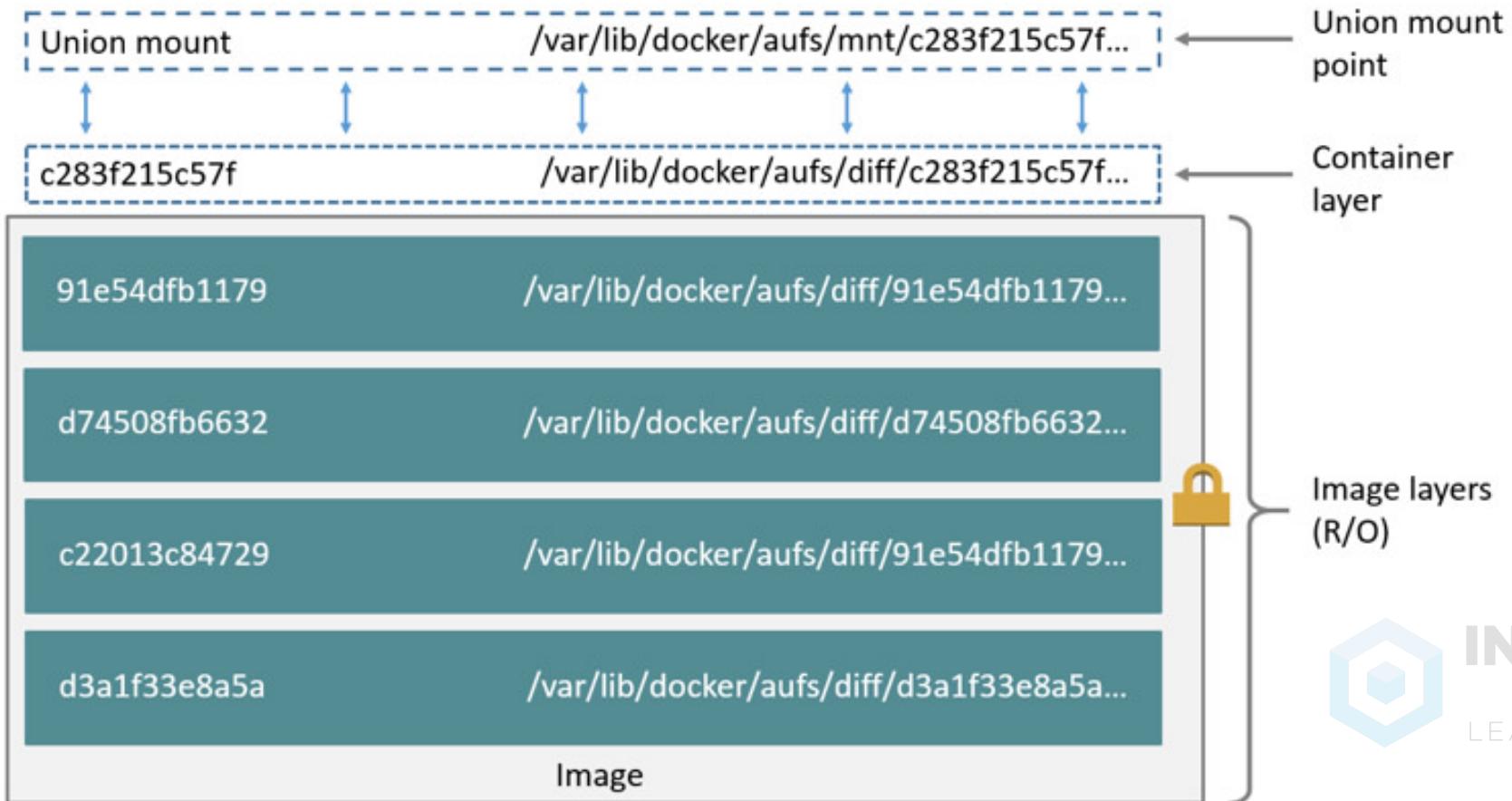
Advanced Multi Layered Unification FileSystem

- Original storage driver
- Implements union mounts
- Requires compatible kernel (not included in mainline Linux)
- File level technology (entire file must be copied)
- Default for:
 - Debian
 - Ubuntu



AUFS Union Mount

AUFS Device Mapper BTRFS



Device Mapper

AUFS

Device Mapper

BTRFS

Stores every image and container on its own thin-provisioned copy-on-write virtual device

- Writes with allocate-on-demand strategy
- Uses loop-lvm as the default
 - Should not be used for production – direct-lvm is preferred
 - direct-lvm requires metadata and data devices created with LVM
- Block oriented technology
- Default for:
 - Red Hat OS's (Fedora, RHEL, CentOS)

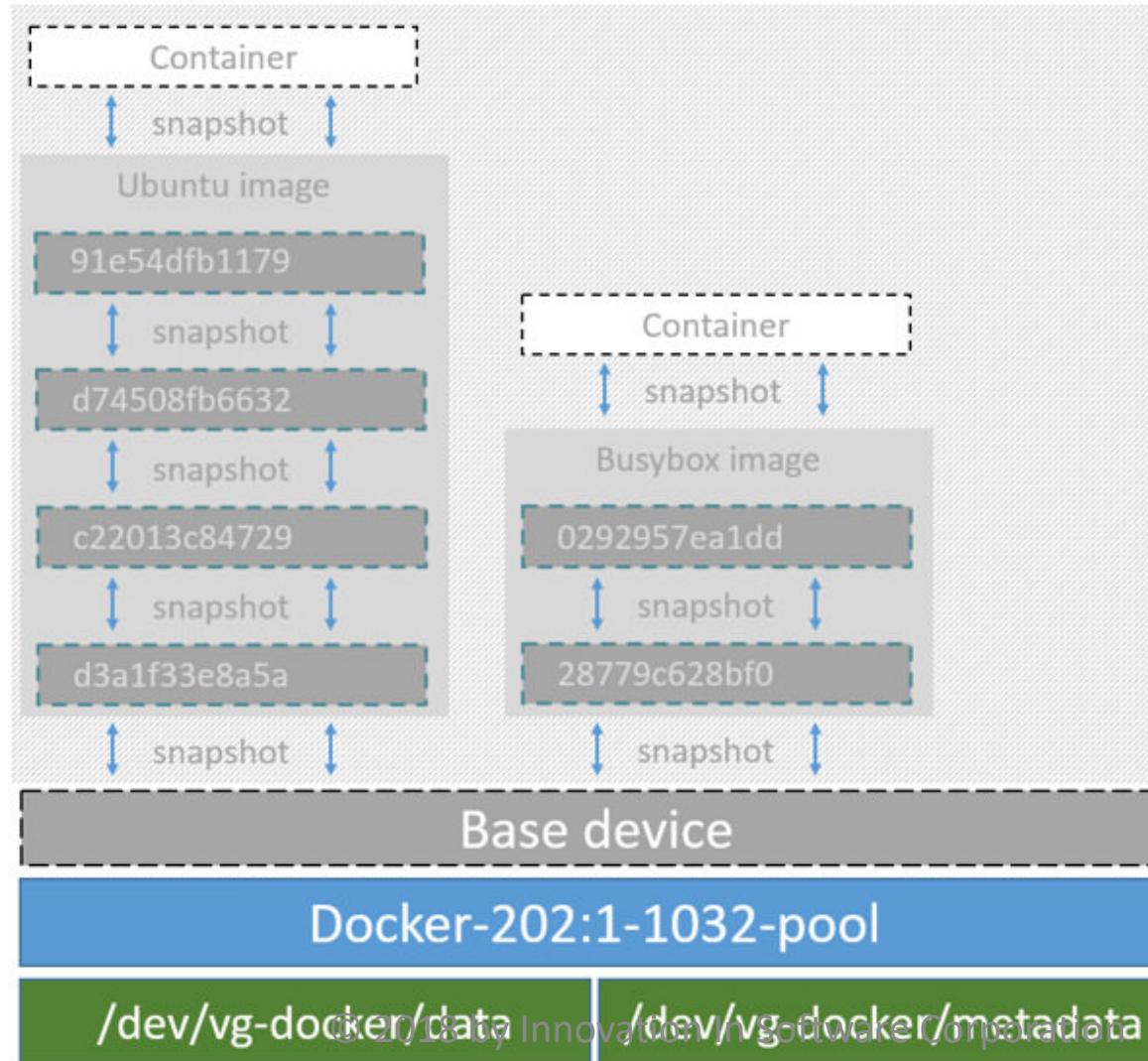


Device Mapper Architecture

AUFS

Device Mapper

BTRFS



BTRFS (B-tree file system)

AUFS

Device Mapper

BTRFS

- Next generation filesystem and matching driver
- Supports:
 - Thin Provisioning
 - Copy-On-Write
 - Snapshotting
- Not consider as “production-ready” as devicemapper driver

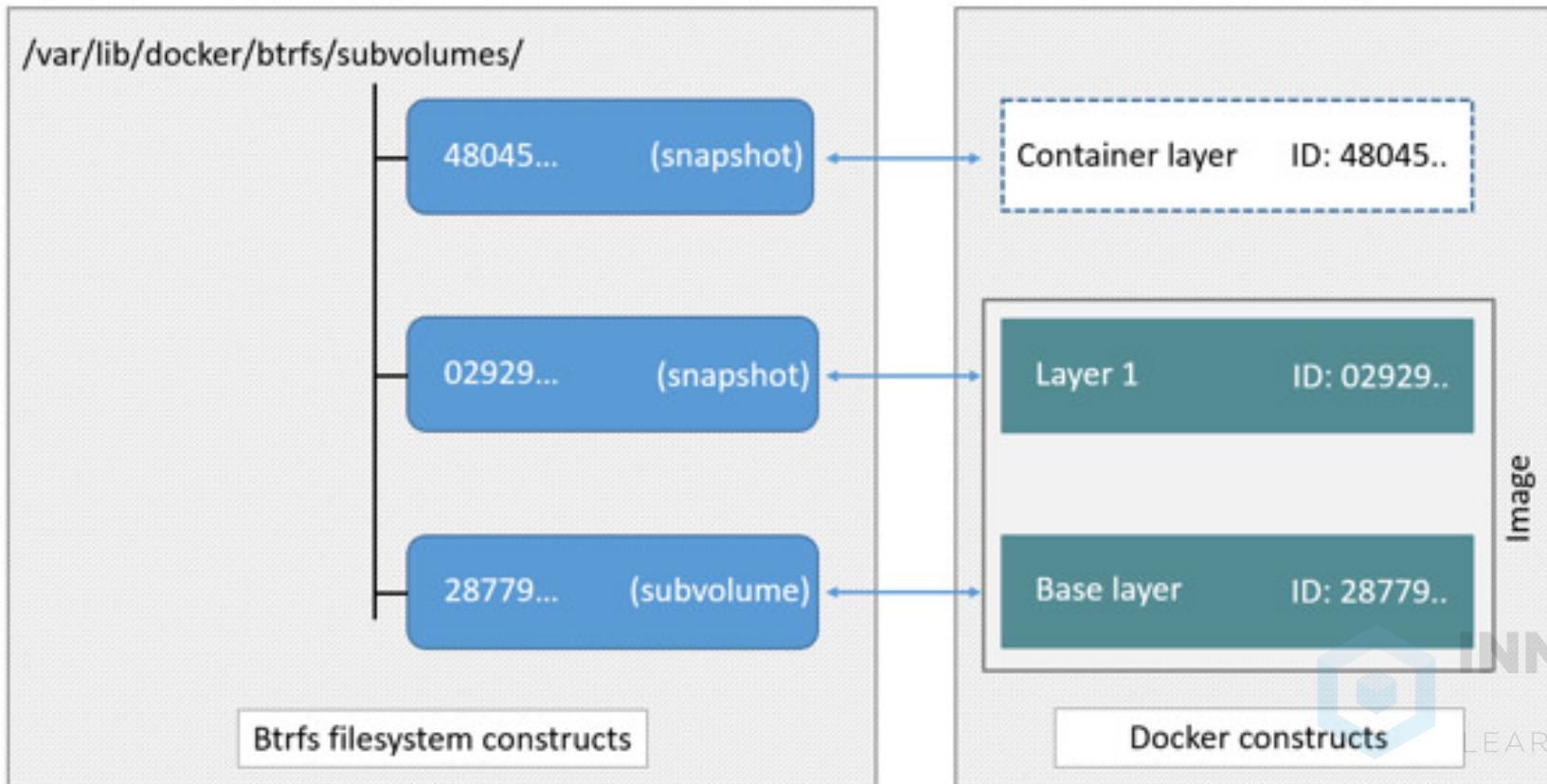


BTRFS (Volume and Layers)

AUFS

Device Mapper

BTRFS



Docker Volume Metadata

© 2018 by Innovation In Software Corporation



Volume Metadata

What is it – volume metadata describes what the volume is through the use of key value pairs

- **docker volume inspect <id>**

- Queries the Docker Engine, a json formatted output is returned

```
$ docker volume inspect <volume>
```

```
  "Name"=  
  "Driver"=  
  "Mountpoint"=  
  "Labels"=  
  "Scope"=
```





Questions



Data Persistence Labs: Tasks 1 & 2



© 2018 by Innovation In Software Corporation



Summary, Review, & Q&A

© 2018 by Innovation In Software Corporation



Summary

- Docker Container Volumes
- Volume Types (bind:managed)
- Volume Use Cases
- Storage Drivers
- Metadata



Review

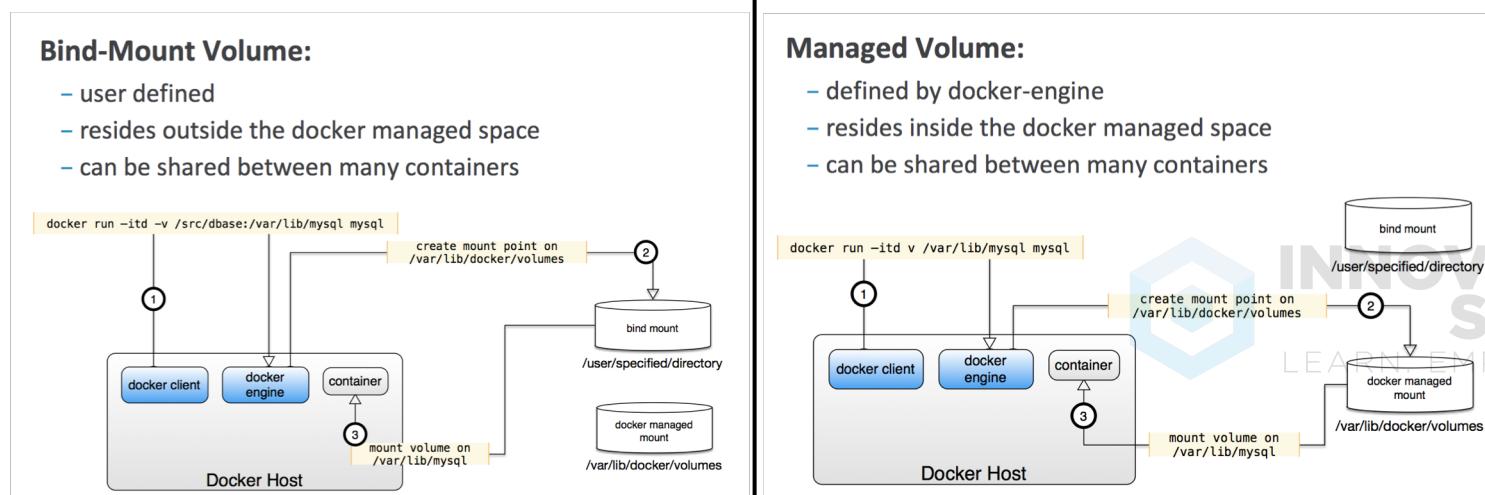
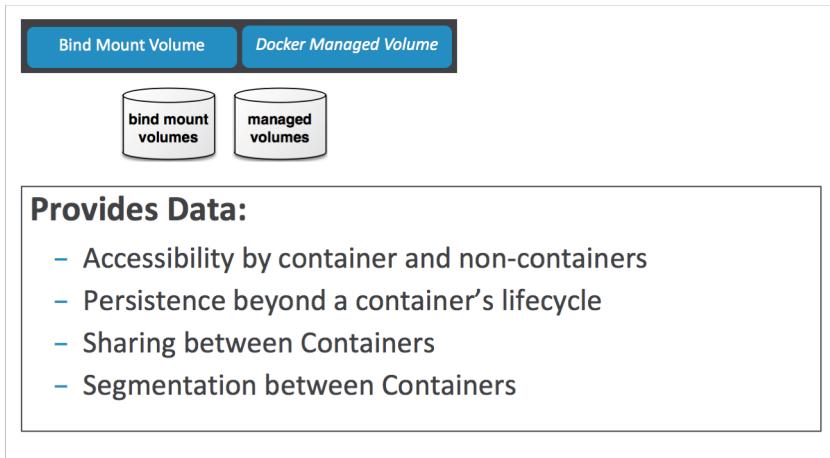
- Docker Container Volumes
- Volume Types (bind:managed)
- Volume Use Cases
- Storage Drivers

- Where Data Persistence is important:
 - Database Records (e.g. MySQL, MariaDB, Cassandra)
- Shared data from host to containers
 - builds tools (e.g. Jenkins jobs)
 - automation tools (e.g. Ansible playbooks)
- Share data between containers
 - Container-A creates data, Container-B uses created data
- Access to Docker host's data
 - /var/log, /etc/hosts, /var/lib/, etc



Review

- Docker Container Volumes
- Volume Types (bind:managed)
- Volume Use Cases
- Storage Drivers



- Docker Container Volumes
- Volume types (bind:managed)
- Volume Use Cases
- Storage Drivers

- Share Mount Points
- Container Troubleshooting
- Build Software on Mount Points
- Load Data from Images
- Data Protection



Review

Docker Container
Volumes
Volume Types
(bind:managed)
Volume Use Cases
Storage Drivers

Technology	Storage Driver Name
OverlayFS	overlay
AUFS	aufs
Device Mapper	devicemapper
Btrfs	btrfs
VFS*	vfs
ZFS	zfs



Docker Compose (Container Orchestration)



© 2018 by Innovation In Software Corporation



Provisioning Containers

Two ways to deploy containers:

Manual

- Slow
- Doesn't Scale
- Error Prone
- Not what Docker was intended to do

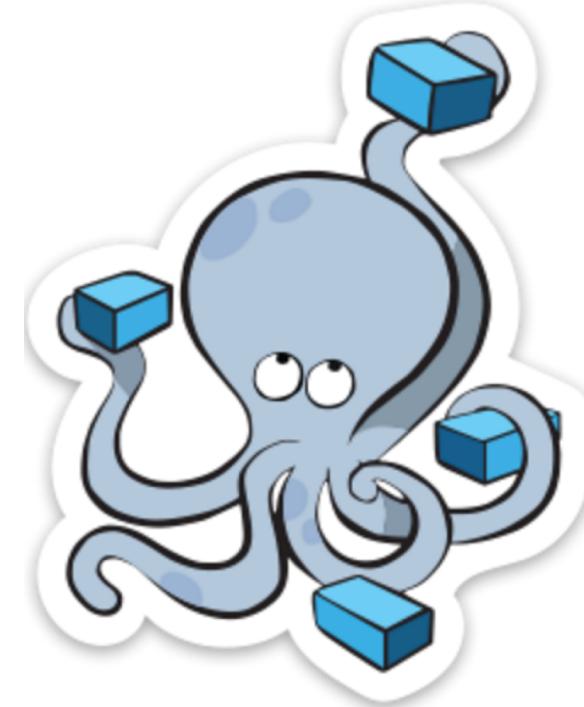
Automated

- Fast
- Scales
- Repeatable
- Aligned with Docker's design intentions



Docker Compose

- Automates the building of applications (services, networks, and volumes) defined in a single file
- Each deployment is the same as the last
 - Compose files can be versioned along with the application source code



Agenda

- Docker Compose:
 - Overview
 - Use Cases
 - Compositions
 - CLI



Docker Compose Overview



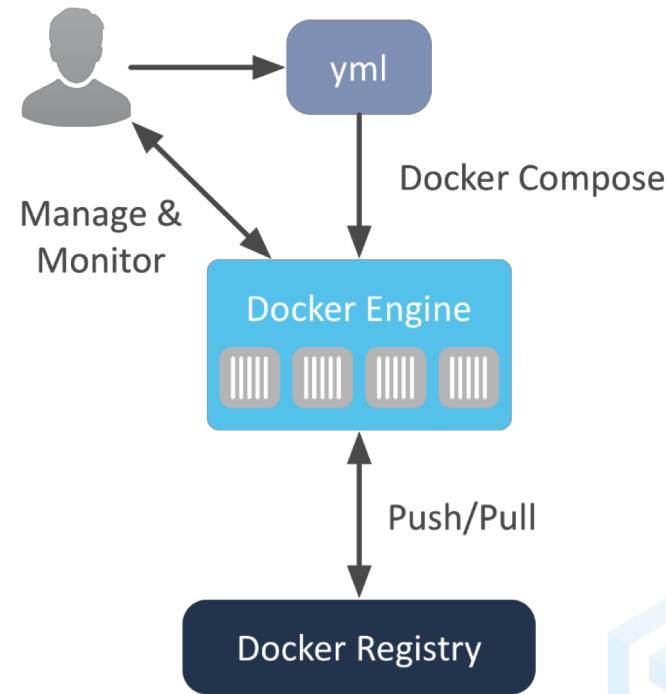
© 2018 by Innovation In Software Corporation



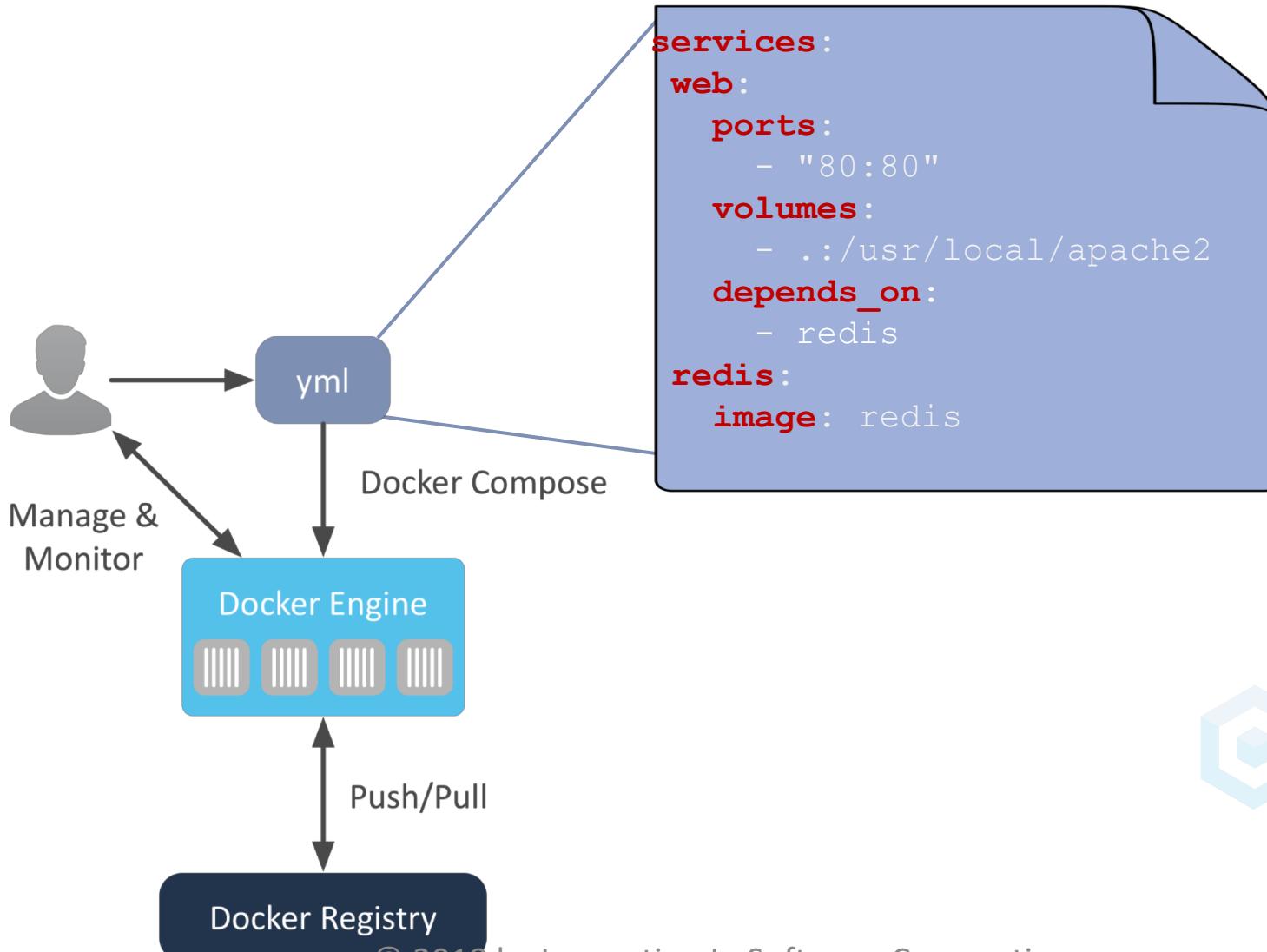
Docker Compose

Tool to create and manage multi-container applications

- Applications (services, networks, and volumes) defined in a single file:
docker-compose.yml
- Transforms applications into individual containers that are linked together
- Compose will start all containers in a single command



Docker Compose



What is a Docker Composition

*A Docker Composition is a **YAML file** where all the top level keys are the names of a service and the values are the service definition.*

As with **docker run**, options specified in the Dockerfile are respected by default and do not need to be specified again in the **docker-compose.yml**

- **Default name and location for a compose file is:**
./docker-compose.yml



Example docker-compose.yml

```
version: '2'

services:

  wordpress:
    image: wordpress
    ports:
      - 8080:80
    environment:
      WORDPRESS_DB_PASSWORD: example

  mysql:
    image: mariadb
    environment:
      MYSQL_ROOT_PASSWORD: example
```



Docker compose CLI

```
$ docker-compose --help
```

Define and run multi-container applications with Docker.

Usage:

```
docker-compose [-f <arg>...] [options] [COMMAND] [ARGS...]
```

```
docker-compose -h|--help
```

Options:

-f, --file FILE	Specify an alternate compose file (default: docker-compose.yml)
-----------------	---

-p, --project-name NAME	Specify an alternate project name (default: directory name)
-------------------------	---

--verbose	Show more output
-----------	------------------

-v, --version	Print version and exit
---------------	------------------------

-H, --host HOST	Daemon socket to connect to
-----------------	-----------------------------



INNOVATION
SOFTWARE
POWER. INNOVATE.

Docker Compose Features

- Multiple isolated environments on a single host
 - Preserve volume data between runs
 - Recreate only changed containers
 - Variable substitution
- Default project name is the basename of the project directory
 - Set a custom project name by using the `-p` command line option or the `COMPOSE_PROJECT_NAME` environment variable



Docker Compose Features

- Multiple isolated environments on a single host
- Preserve volume data between runs
 - Finds any containers from previous runs
 - Copies the volumes from the old containers to the new containers
- Recreate only changed containers
- Variable substitution



Docker Compose Features

- Multiple isolated environments on a single host
- Preserve volume data between runs
- Recreate only changed containers
- Variable substitution
- Caches the configuration used to create a container



Docker Compose Features

- Multiple isolated environments on a single host
- Preserve volume data between runs
- Recreate only changed containers
- Variable substitution
 - Supports variables in the Compose file
 - Can pass environment variables at runtime

Docker Compose Use Cases

© 2018 by Innovation In Software Corporation



Use Cases

- **Multiple development environments**
- Continuous Integration environments
- Shared Host environments
- Automated testing

Create multiple copies of a single environment

Example:

- Copy for
 - The stable branch
 - The current branch
 - Release candidate branch



Use Cases

- Multiple development environments
 - **Continuous Integration environments**
 - Shared Host environments
 - Automated testing
- On a shared build environment server

 - Can set the project name prefix to a unique build number
 - Keeps builds from interfering with each other



Use Cases

- Multiple development environments
 - Continuous Integration environments
 - **Shared Host environments**
 - Automated testing
- Allows projects that may use the same service names from interfering with each other



Use Cases

- Multiple development environments
- Continuous Integration environments
- Shared Host environments
- **Automated testing**

Run automated test scripts against a repeatable environment



Docker Compositions

© 2018 by Innovation In Software Corporation



Compose Service Definition

This is the top level key,
this is the Service
Definition

Each defined service
must specify exactly one
image

Other keys are optional
and analogous to their
docker run command-
line counter parts

```
wordpress:  
  image: wordpress:latest  
  ports:  
    - "443:443"  
environment:  
  - ADMIN_PASS=xxx  
mysql:  
  image: mysql  
  volumes:  
    - mysql-data:/var/lib/mysql  
  ports:  
    - "3306:3306"  
  links:  
    - wordpress
```



INNOVATION
SOFTWARE
LEARN. ENPOWER. INNOVATE.

<continued>

Compose Service Definition

ports: Publish a container's port(s) to the host

volumes: Bind mount a volume

environment: Set environment variables

devices: Add a host device to the container

dns: set custom DNS server

docker run --help
for more information

```
mariadb:  
  image: mariadb:latest  
  ports:  
    - "3306:3306"  
  volumes:  
    - wp-data:/var/wp-data  
  environment:  
    - DB_ADMIN_PASS=xxx  
  devices:  
    - "/dev/ttyUSB0:/dev/ttyUSB0"  
  dns:  
    - 8.8.8.8  
    - 9.9.9.9
```



INNOVATION
SOFTWARE
LEARN. ENPOWER. INNOVATE.

Linking multiple containers

Service: Wordpress

links:
Add link to another container. Provides a secure channel via which Docker containers can communicate with one another.

In this example, the link provides secure communication between the Wordpress plug-in and Wordpress application.

```
wordpress:  
  image: wordpress:latest  
  ports:  
    - "443:443"  
  environment:  
    - ADMIN_PASS=xxx  
mysql:  
  image: mysql  
  volumes:  
    - mysql-data:/var/lib/mysql  
  ports:  
    - "3306:3306"  
  links:  
    - wordpress
```



INNOVATION
SOFTWARE
LEARN. ENPOWER. INNOVATE.

Container Environment Variables

Service: Wordpress

environment:

Define environment variables set in container's runtime environment

```
wordpress:  
  image: wordpress:latest  
  ports:  
    - "443:443"  
  environment:  
    - ADMIN_PASS=xxx
```

To see what environment variables are available to a service, run:

```
# $ docker-compose run SERVICE env
```

Using the wordpress service as an example, the following slide displays the output of the command:



Show Available Environment Variables

```
# $ docker-compose run wordpress env

# HOSTNAME=f836bf313aed
# TERM=xterm
# CA_CERTIFICATES_JAVA_VERSION=20140324
# PATH=/usr/share/wordpress/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
# PWD=/
# JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre
# LANG=C.UTF-8
# JAVA_VERSION=8u66
# SHLVL=0
# HOME=/root
# JAVA_DEBIAN_VERSION=8u66-b17-1~bpo8+1
```



Environment Variable Substitution

- You are able to pass variable values from the shell environment to docker-compose
- The environment variable POSTGRES_VERSION=9.4 might be set within the shell. You could then supply the following configuration:

```
db:  
  image: "postgres:${POSTGRES_VERSION}"
```



INNOVATION
SOFTWARE
EMPOWER. INNOVATE.

Docker Compose CLI

© 2018 by Innovation In Software Corporation



Running in Detached Mode

Running **docker-compose** with **-d** runs the Docker containers in the background

```
# $ docker-compose up
# 02multiple_web_1 is up-to-date
# Attaching to web_1
# web_1 | => Configuring NGINX ...
# web_1 | => ... Starting Services
# Gracefully stopping... (press Ctrl+C
# again to force)
# Stopping 02multiple_web_1 ... done
# $ docker-compose up -d
# Starting 02multiple_web_1
# $
```



INNOVATION
SOFTWARE
RN. EMPOWER. INNOVATE.

Displaying logs for docker-compose

docker-compose logs displays log output from services

```
# $ docker-compose logs
# Attaching to web_1
# web_1 | => Configuring NGINX ...
# web_1 | => Using default NGINX
# configuration
# web_1 | => Configuring PHP-FPM ...
# web_1 | => Done!
# web_1 | => ... Starting Services
```



INNOVATION
SOFTWARE
RN. EMPOWER. INNOVATE.

Scaling docker-compose containers

docker-compose scale [SERVICE=NUM]
sets the number of containers for a service

```
# $ docker-compose scale web=3
# Creating and starting 2 ... done
# Creating and starting 3 ... done
# $ docker-compose scale web=1
# Stopping 02multiple_web_2 ... done
# Stopping 02multiple_web_3 ... done
# Removing 02multiple_web_3 ... done
# Removing 02multiple_web_2 ... Done
# $
```



INNOVATION
SOFTWARE
EMPOWER. INNOVATE.

Running a Selected Single Service

docker-compose run, runs a one-time command against a service

Docker-compose run overrides the commands specified in the docker-compose service configuration

```
# $ hostname  
# docker-workstation  
# $ docker-compose run web bash  
# [root@9ad9eec462fd /]# hostname  
# 9ad9eec462fd  
# [root@8f3f8001f8df /]# exit  
# exit  
# $ hostname  
# docker-workstation
```



INNOVATION
SOFTWARE
EMPOWER. INNOVATE.

Stopping a Selected Single Service

docker-compose stop SERVICE shuts down the service
gracefully

docker-compose kill SERVICE immediately kills the service

```
# $ docker-compose run --rm -d web
# Starting web_1
# $ docker-compose kill web
# Killing web_1 ... done
# $ docker-compose ps
# Name      Command     State        Ports
# web_1    /bin/run   Exit           137
# $
```



INNOVATION
SOFTWARE
. EMPOWER. INNOVATE.

Orchestration Lab

© 2018 by Innovation In Software Corporation



Summary, Review & QA

© 2018 by Innovation In Software Corporation



Summary

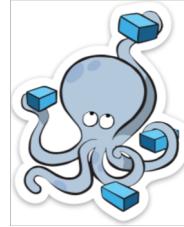
Docker Compose:

- Overview
- Use Cases
- Compositions
- CLI



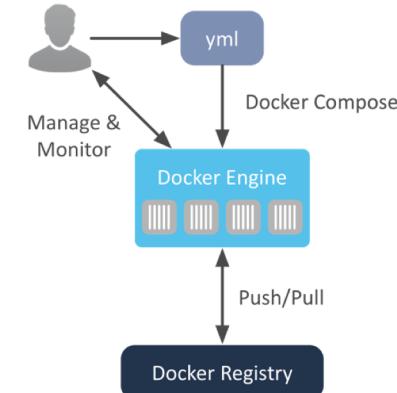
Review

- Compose Overview
- Compose Use Cases
- Compositions
- Compose CLI



Tool to create and manage multi-container applications

- Applications (services, networks, and volumes) defined in a single file: **docker-compose.yml**
- Transforms applications into individual containers that are linked together
- Compose will start all containers in a single command



- Multiple isolated environments on a single host
- Preserve volume data between runs
- Recreate only changed containers
- Variable substitution

Review

- Compose Overview
 - Compose Use Cases
 - Compositions
 - Compose CLI
- Multiple development environments
 - Continuous Integration environments
 - Shared Host environments
 - Automated testing



Review

- Compose Overview
- Compose Use Cases
- Compositions
- Compose CLI

A Docker Composition is a **YML file** where all the top level keys are the names of a service and the values are the service definition.

```
wordpress:  
  image: wordpress:latest  
  ports:  
    - "443:443"  
  environment:  
    - ADMIN_PASS=xxx  
wordpress-plug:  
  build: wp-plug/  
  volumes:  
    - wp-plug-data:/var/wp-plug-data  
  ports:  
    - "8080:5601"  
  links:  
    - wordpress
```

Review

- Compose Overview
- Compose Use Cases
- Compositions
- Compose CLI
 - Running **docker-compose** with **-d** runs the Docker containers in the background.
 - docker-compose logs** displays log output from services.
 - docker-compose scale [SERVICE=NUM]** sets the number of containers for a service.
 - docker-compose run**, runs a one-time command against a service.
**overrides the commands specified in the docker-compose service configuration
 - docker-compose stop SERVICE** shuts down the service gracefully
 - docker-compose kill SERVICE** immediately kills the service





Questions



Container Clusters (Docker Swarm)



© 2018 by Innovation In Software Corporation



Agenda

- Docker Swarm Overview
- Swarm Scheduler Strategies
- Discovery Backend
- Deploy and Configure Swarm
- Deploy a Container with Swarm



Docker Swarm Overview



© 2018 by Innovation In Software Corporation



Docker Swarm Features

- Cluster management integrated in Docker Engine
- Decentralized Design
- Declarative Service Model
- Scaling
- Desired State Reconciliation
- Multi-Host Networking
- Service Discovery
- Load Balancing
- Secure by Default
- Rolling Updates



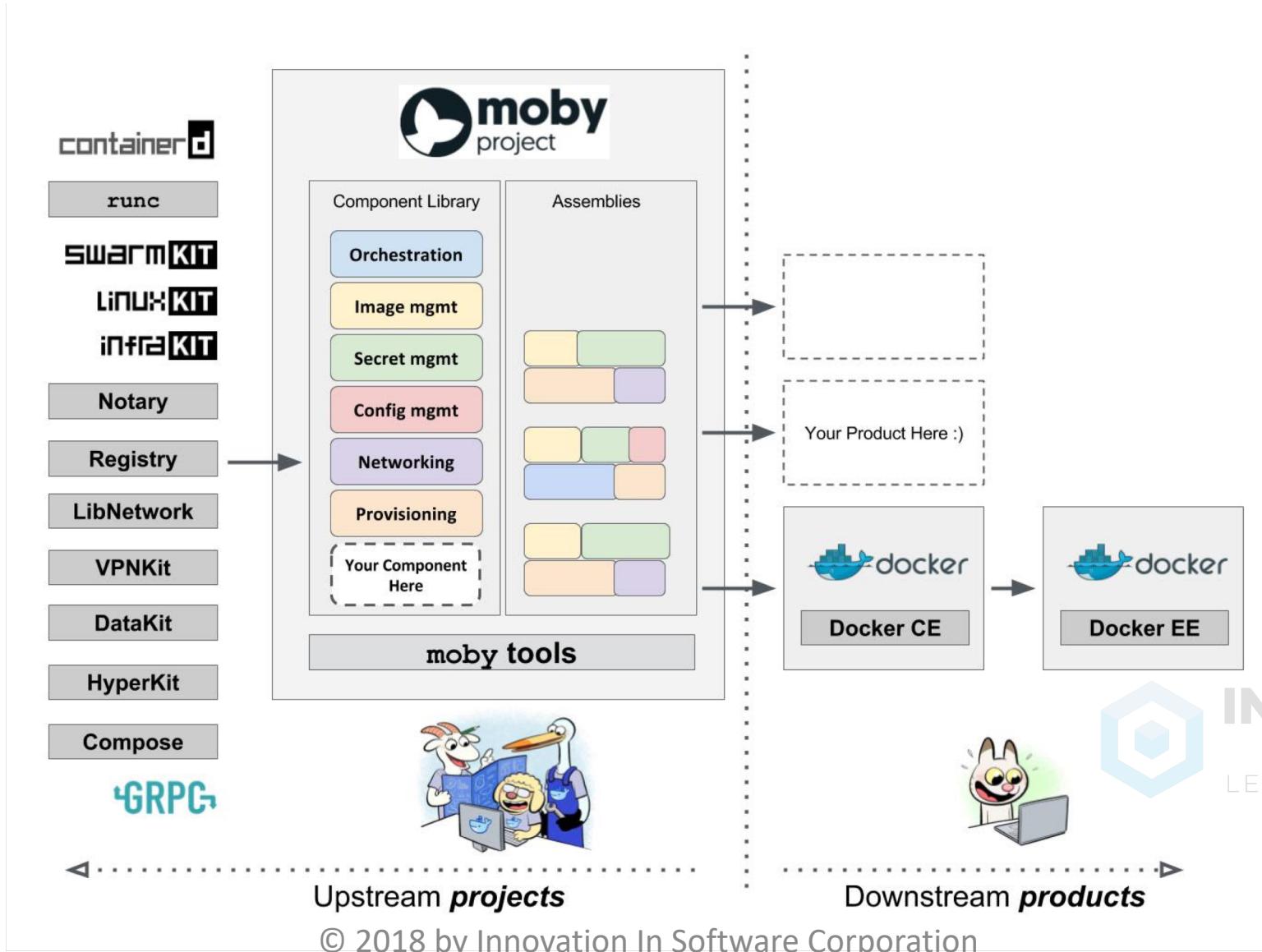
Docker Swarm

What is a Swarm?

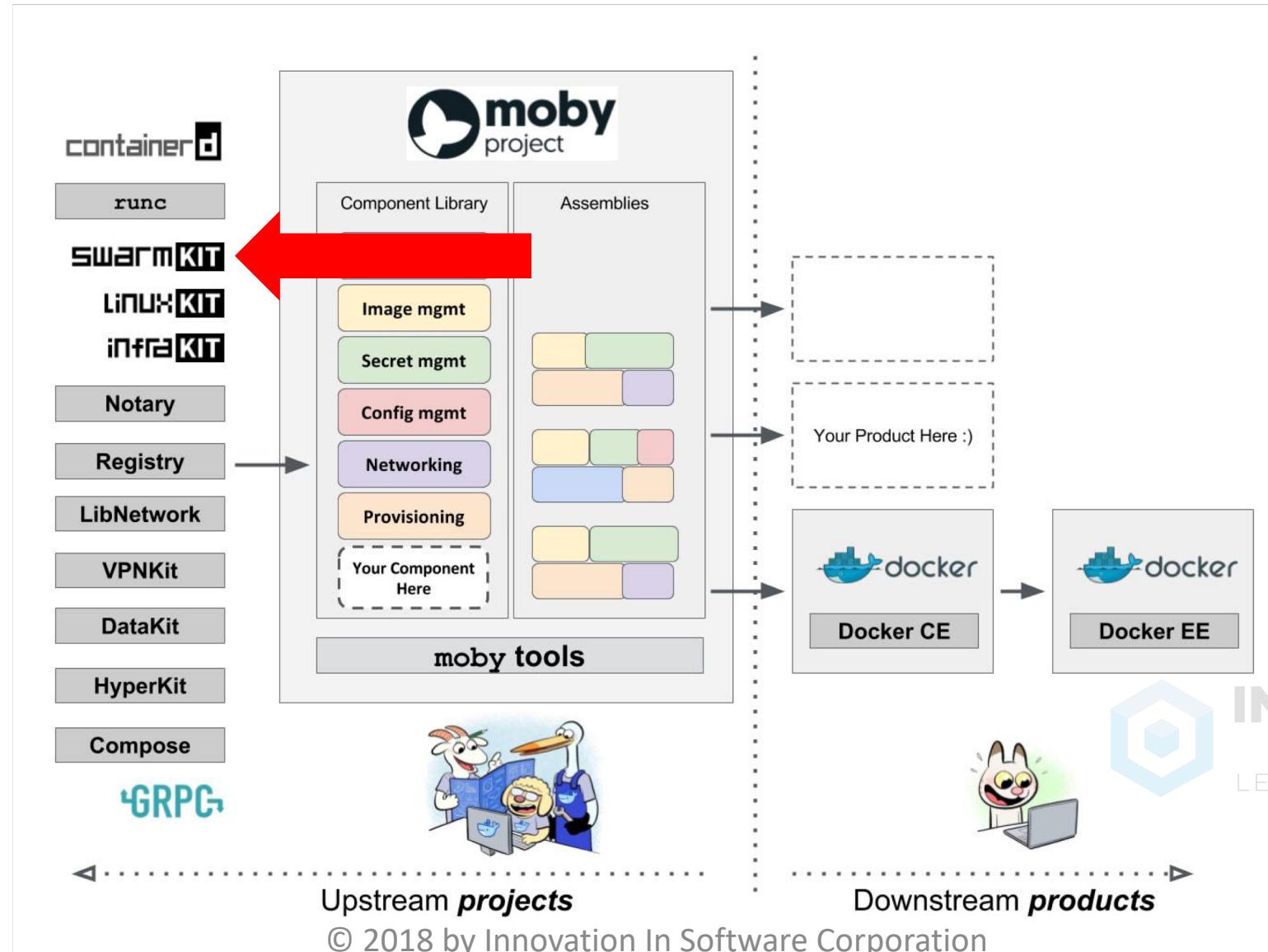
- A Swarm is a cluster of Docker Engines (nodes) where services are deployed.
- ***Cluster management*** and ***orchestration*** are built into Docker Engine using the Swarm Kit.
- Swarm mode is enabled using ***swarm init*** or ***swarm join***.
- Runs ***Container commands*** and ***Swarm commands*** at the same time
- Swarm mode enables the orchestration of services



Docker Swarm



Docker Swarm

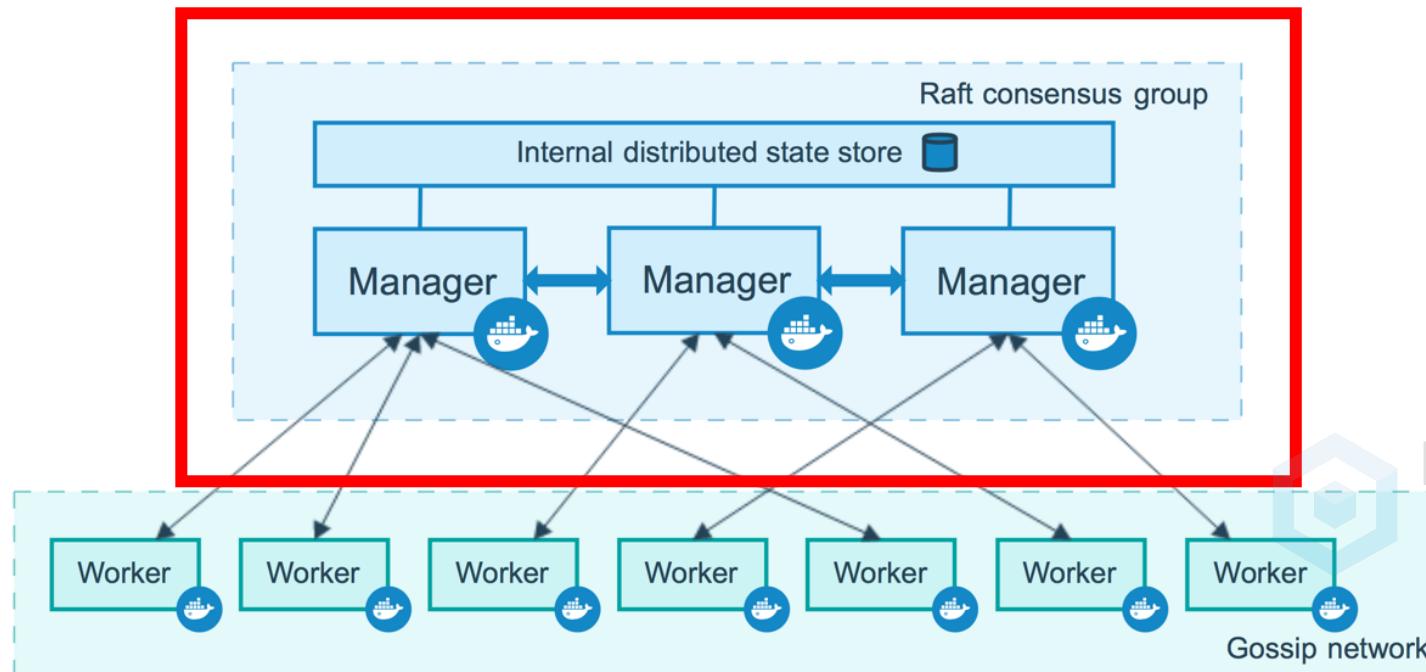


INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

Docker Swarm (Swarm Nodes)

Manager Nodes

- Maintains Cluster State
- Schedules Services
- Serves up swarm mode HTTP API endpoints



INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

Docker Swarm (Nodes)

What are Nodes?

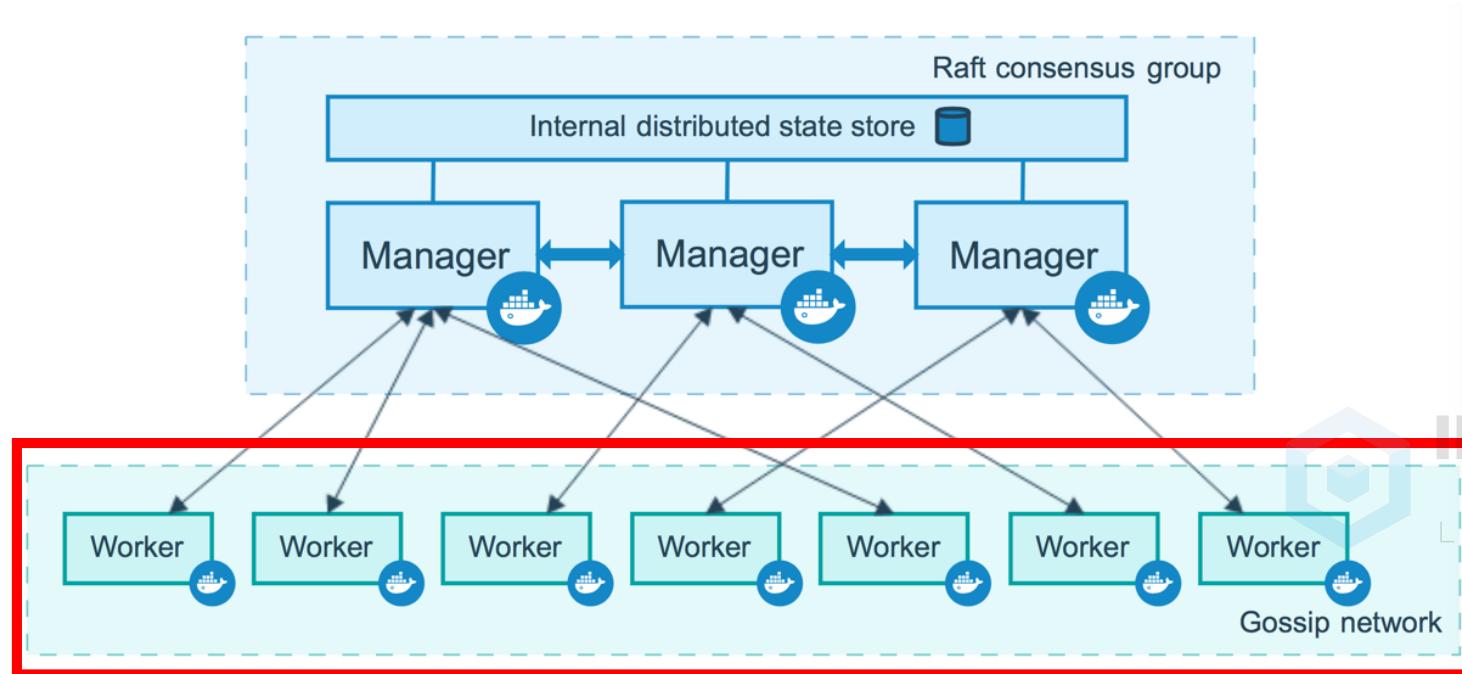
- An instance of the Docker Engine running in Swarm Mode
- Can run standalone (is that really a cluster?) or distributed across many machines
- Nodes are manager nodes or worker nodes
- Manager Nodes elect a single “Leader”
- Manager Nodes dispatch tasks to worker nodes
- Worker Nodes receive and execute tasks
- A Manager Node can also be a Worker Node



Docker Swarm (Worker Nodes)

Worker Nodes

- Executes containers (**Tasks**)
- Serve the swarm mode HTTP API
- Requires a Manager Node to function



Docker Swarm (Services and Tasks)

What are Services and Tasks?

- Service is a definition of the tasks to execute on the workers
- Primary root of user interaction with Swarm
- Two Types of **Services**:
 - Replicated Services - Distributed to a specific number of workers
 - Global Services - Distributed to all workers
- A **Task** is the atomic scheduling unit of swarm
- A **Task** can only run on one worker node at anytime
- A **Task** can only run or fail



Docker Swarm (Load Balancing)

What is Load Balancing?

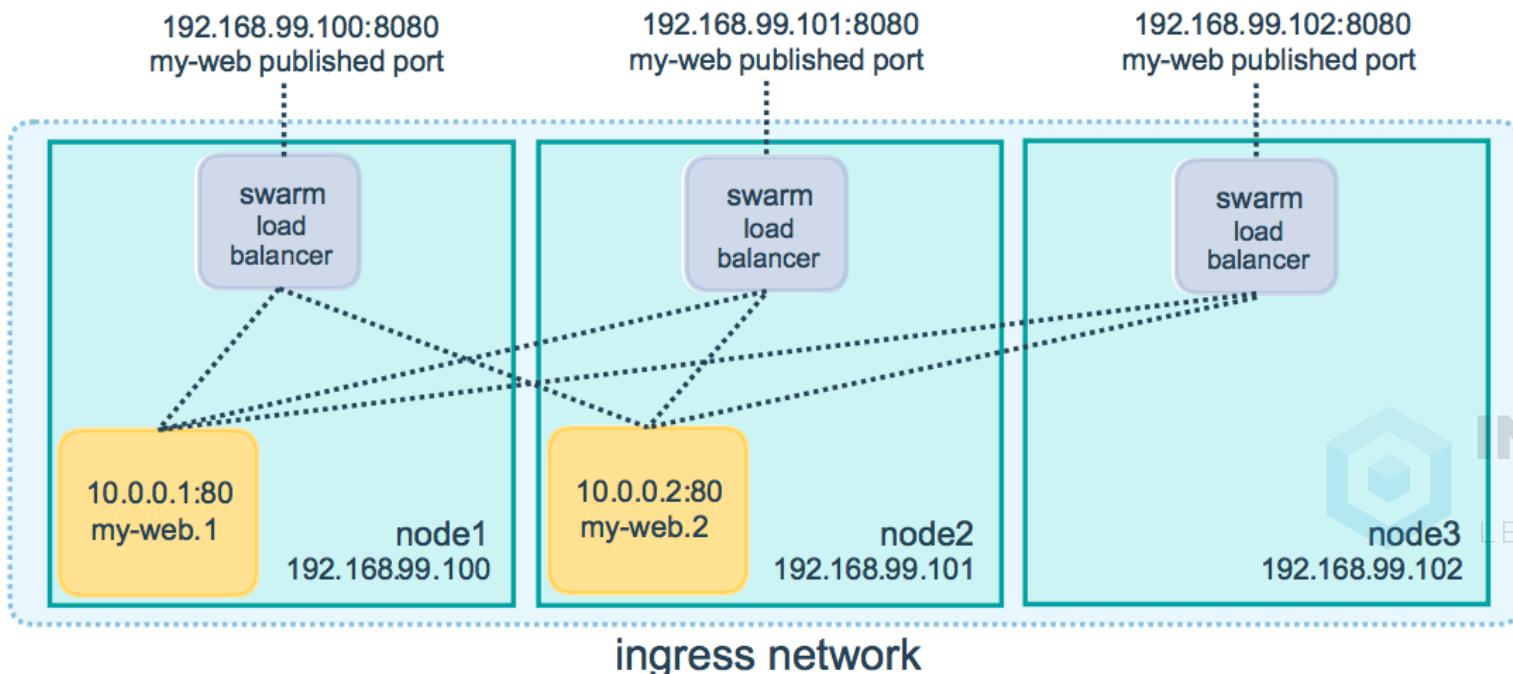
- Swarm uses ingress load balancing
- Swarm automatically assigns PublishedPorts
- Swarm can dynamically assign ports (30000 - 32767)
- External Load Balancers access PublishedPorts of Nodes
- ALL nodes route ingress traffic to the correct Task
- Swarm has internal DNS (routing mesh)



Docker Swarm (Load Balancing)

Swarm Load Balancing Routing Mesh

- All nodes are aware of **TASK** locations in Swarm
- Can send traffic to any node
- Traffic automatically routed to correct node

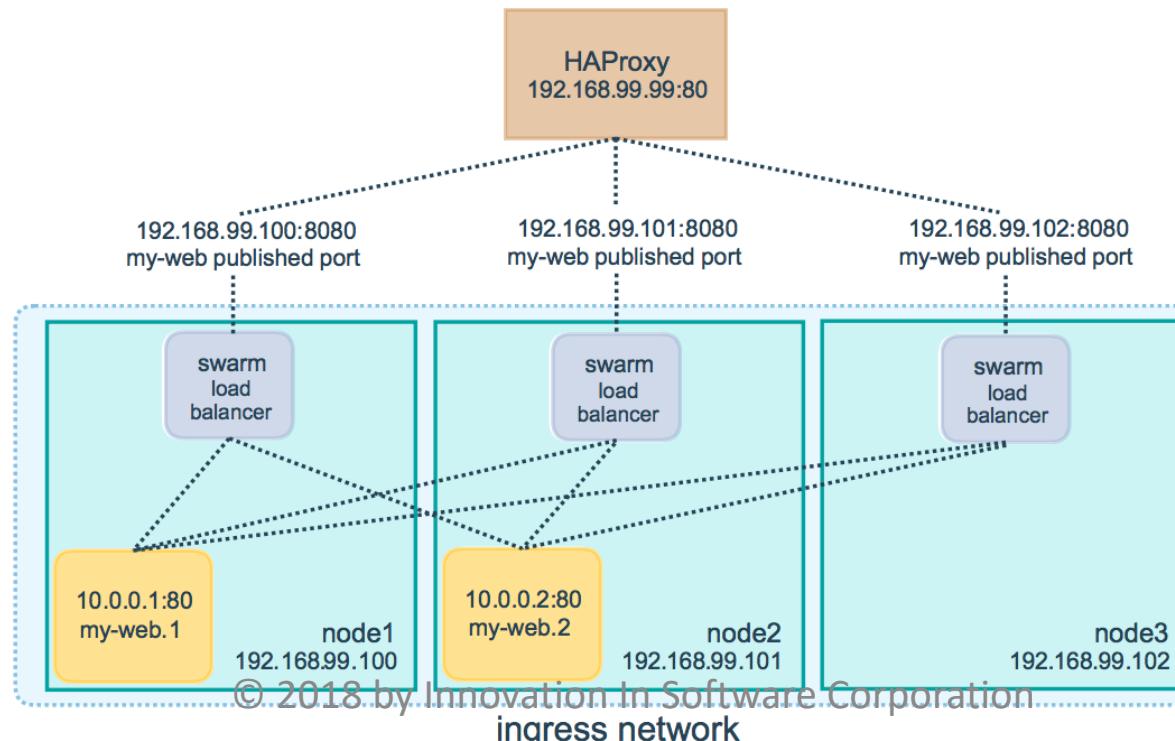


INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

Docker Swarm (Load Balancing)

Using an External Load Balancer

- External Load Balancer on TCP 80
- Internal Swarm Routing Mesh on TCP 8080
- All nodes are aware of **TASK** locations in Swarm



INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

Swarm Scheduler

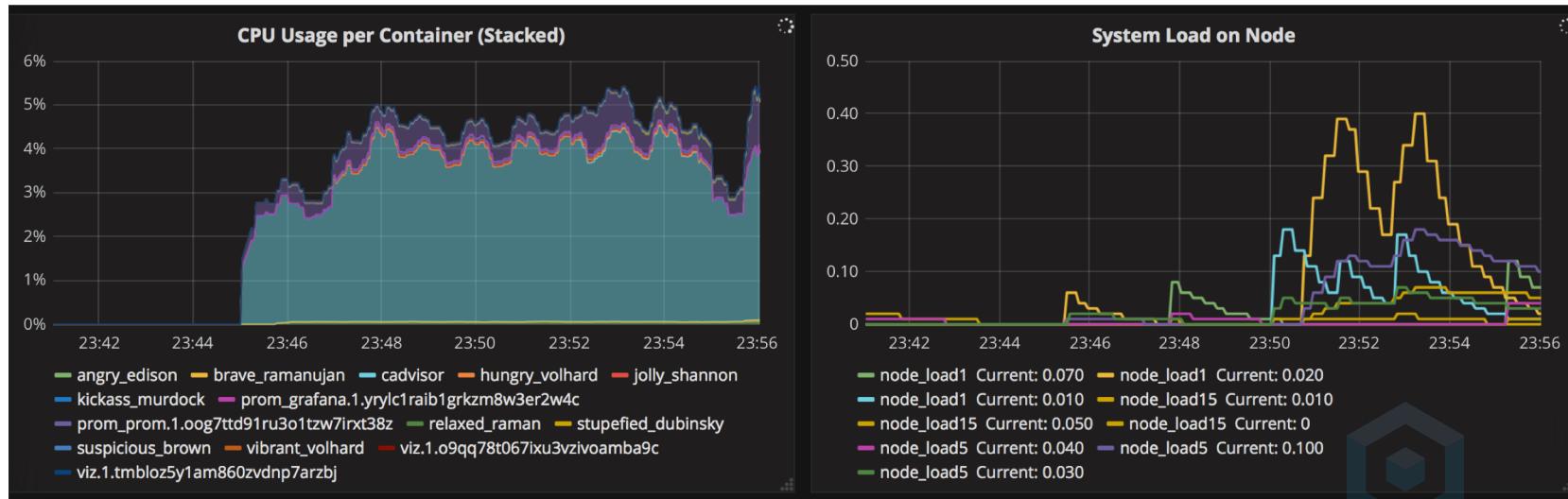


© 2018 by Innovation In Software Corporation



Swarm Scheduler - Resource Availability

Resource Availability is tracked by the Docker Engine on each node. Swarm Scheduler is aware of available resources on all Worker Nodes



INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

Swarm Scheduler - Labels

Labels can be specified to record attributes of a Swarm node
(e.g. role = manager, env = dev, storage = ssd)

```
# $ docker node update [OPTIONS] NODE
```

```
# $ docker node update --label-add env=dev node-1
```

```
# $ docker node inspect node-1
```

```
# $ docker node inspect --pretty node-1
```

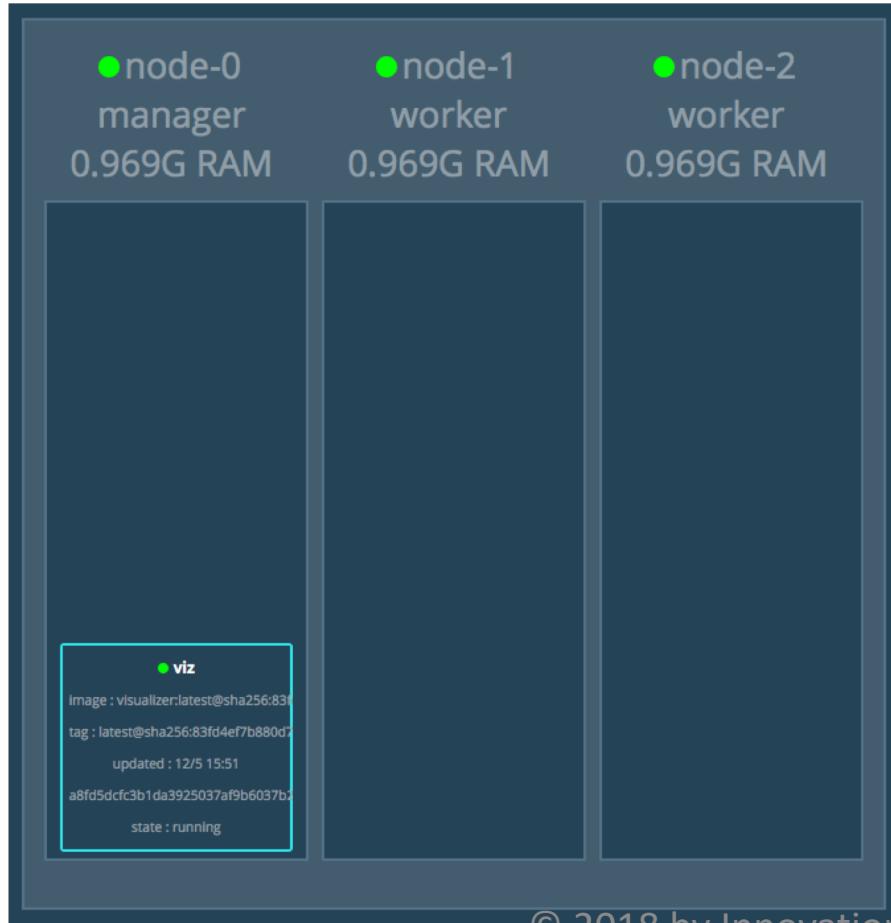
```
# $ docker node update --label-add disk=ssd node-1
```



INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

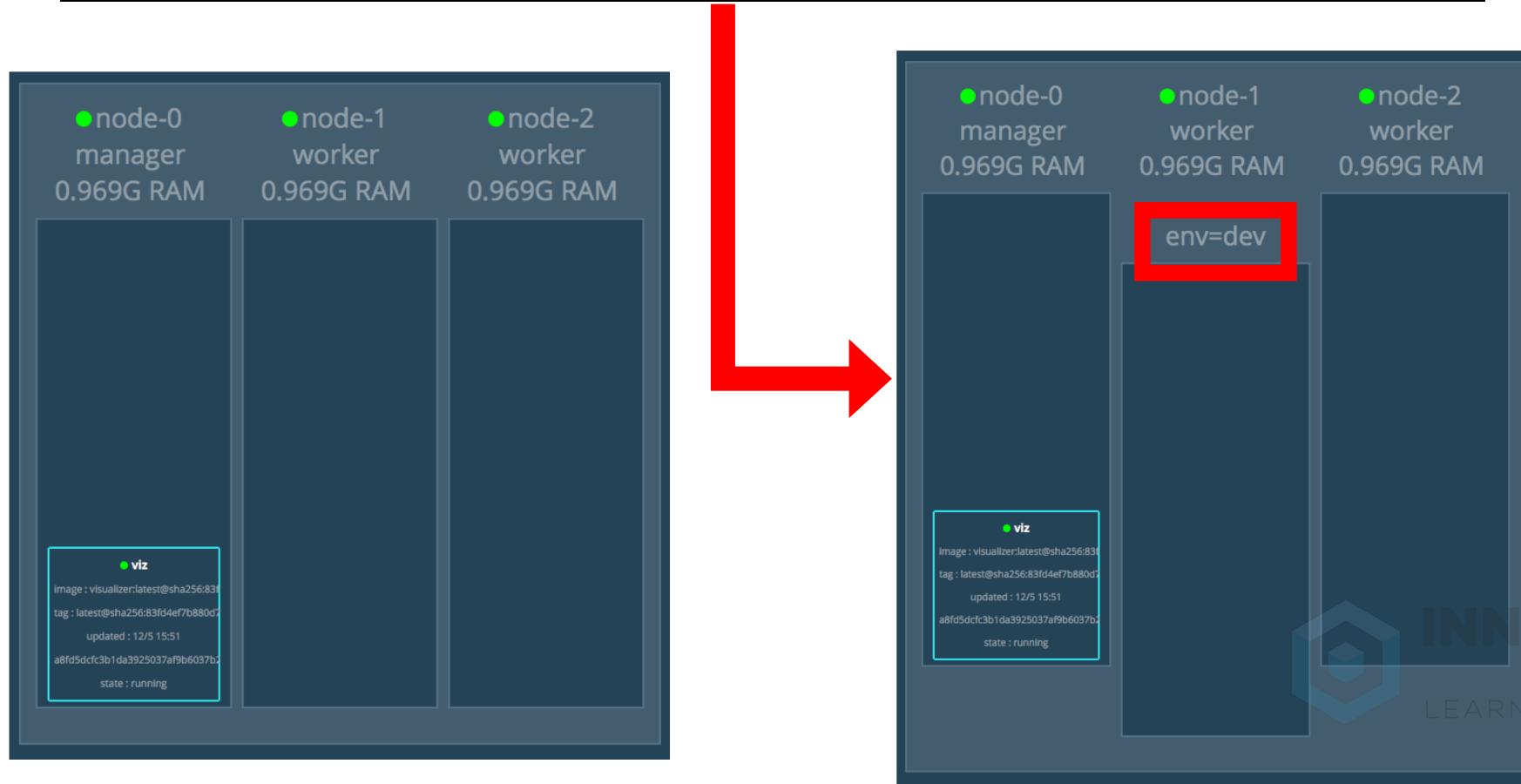
Swarm Scheduler - Labels

```
# $ docker node update --label-add env=dev node-1
```



Swarm Scheduler - Labels

```
# $ docker node update --label-add env=dev node-1
```



Swarm Scheduler - Constraints

Constraints specified on a Service to restrict task scheduling to nodes with certain attributes, e.g. nodes with specific label

```
# $ docker service create --name nginx-dev \
#   --constraint 'node.labels.env == dev' \
#   nginx:1.11-alpine
```

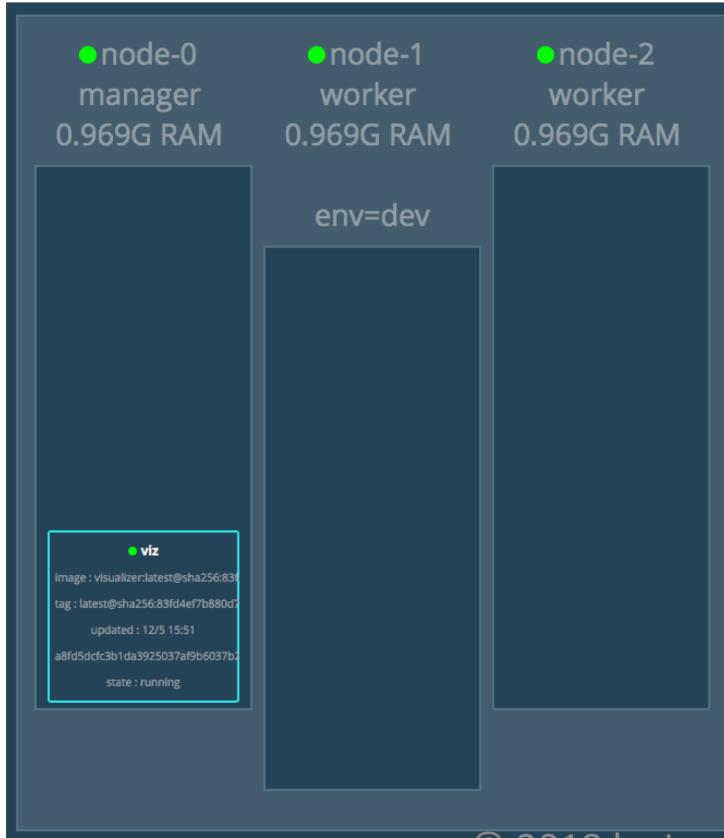
```
# $ docker node inspect --pretty node-1
#   Labels:
#     - disk = ssd
#     - env = dev
#   Hostname:          node-1
```



INNOVATION
SOFTWARE
N. EMPOWER. INNOVATE.

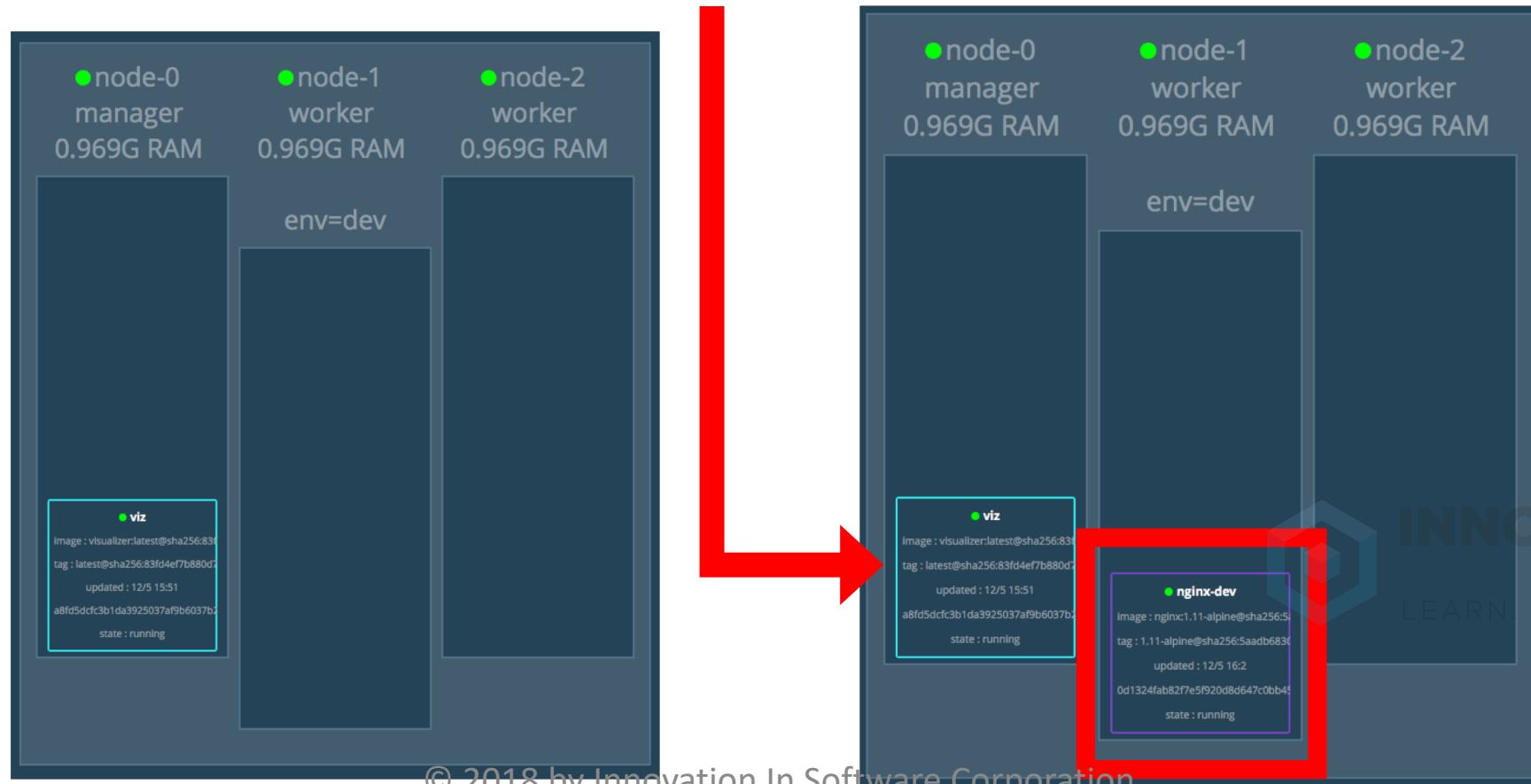
Swarm Scheduler - Constraints

```
# $ docker service create --name nginx-dev \
--constraint 'node.labels.env == dev' \
nginx:1.11-alpine
```



Swarm Scheduler - Constraints

```
# $ docker service create --name nginx-dev \
--constraint 'node.labels.env == dev' \
nginx:1.11-alpine
```



Swarm Scheduler - Pre-Defined Attributes

Constraints can reference a range of other pre-defined attributes on nodes, in addition to user-defined labels

node attribute	matches	example
node.id	Node ID	node.id == 2ivku8v2gvtg4
node.hostname	Node hostname	node.hostname != node-2
node.role	Node role	node.role == manager
node.labels	user defined node labels	node.labels.security == high
engine.labels	Docker Engine's labels	engine.labels.operatingsystem == ubuntu 14.04



Deploy & Configure Swarm

© 2018 by Innovation In Software Corporation



Deploy & Configure Swarm

Prerequisites

Run Swarm Init

Run Swarm Join

- 1 Docker Nodes (minimum)
 - Swarm Manager/Node
- Docker Engine 17.03CE installed



INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

Deploy & Configure Swarm

Prerequisites

Run Swarm Init

Run Swarm Join

```
# $ docker swarm init  
# Swarm initialized: current node (p0lpp0knm362nipb6jxlq06a6) is now a manager.
```

To add a worker to this swarm, run the following command:
docker swarm join \
--token SWMTKN-1-5bf0v9rruiau0a4a4ov2d4927abe9w31zyegr \
10.0.22.10:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

```
# $
```



INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

Source: <https://docs.docker.com/swarm/install-manual/>

Deploy & Configure Swarm

Prerequisites

Run Swarm Init

Run Swarm Join

\$ docker swarm join

This node joined a swarm as a worker.



INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

Deploy a Service with Swarm

© 2018 by Innovation In Software Corporation



Deploy a Service with Swarm

Deploy an nginx application to the Swarm.

Deploy service on manager:

```
$ docker service create --name nginx \  
  --replicas 2 nginx
```



Deploy a Service with Swarm

List containers in service

```
$ docker service ps nginx
```

ID	NAME	IMAGE	NODE	DESIRED	STATE
rzybb	nginx.1	nginx:latest	node-2	Running	
agoth	nginx.2	nginx:latest	node-0	Running	



Scale a Service with Swarm

Scale nginx application to the Swarm.

Scale nginx service up

```
$ docker service update nginx --replicas 3
```



Scale a Service with Swarm

New container created for service

```
$ docker service ps nginx
```

ID	NAME	IMAGE	NODE	DESIRED	STATE
rzybb	nginx.1	nginx:latest	node-2	Running	
agoth	nginx.2	nginx:latest	node-0	Running	
0yfp0	nginx.3	nginx:latest	node-1	Running	



Clustering Lab

© 2018 by Innovation In Software Corporation



Summary, Review & QA

© 2018 by Innovation In Software Corporation



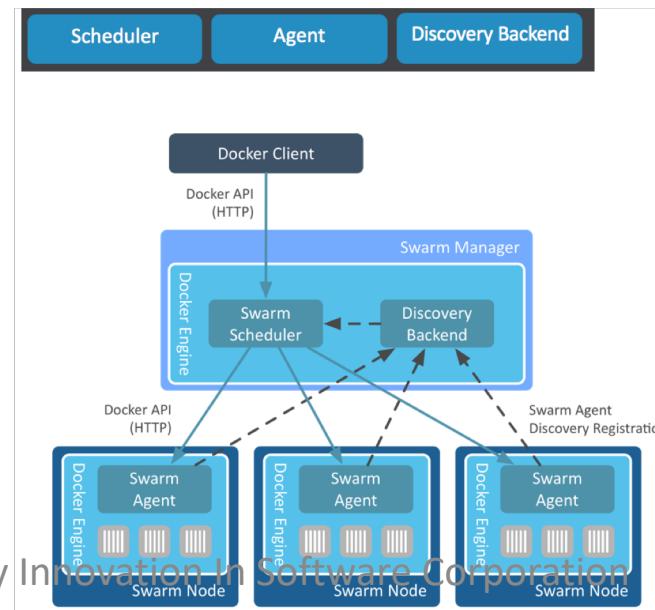
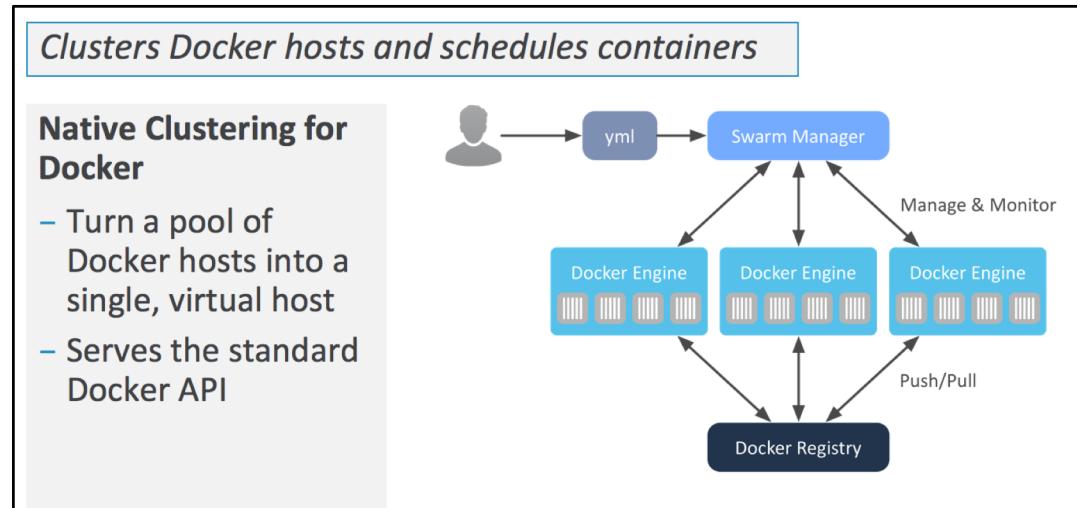
Summary

- Docker Swarm Overview
- Swarm Scheduler Strategies
- Discovery Backend
- Deploy and Configure Swarm
- Deploy a Container with Swarm



Review

- Docker Swarm Overview
- Swarm Scheduler
- Deploy and Configure Swarm
- Deploy a Service with Swarm



Review

- Docker Swarm Overview
- Swarm Scheduler
- Deploy and Configure Swarm
- Deploy a Service with Swarm



Review

- Docker Swarm Overview
- Swarm Scheduler
- Deploy and Configure Swarm
- Deploy a Container with Swarm

```
# $ docker swarm init
# Swarm initialized: current node (p0lpp0knm362nipb6jxlq06a6) is now a
# manager.

To add a worker to this swarm, run the following command:
  docker swarm join \
    --token SWMTKN-1-5bf0v9rruiau0a4a4ov2d4927abe9w31zyegr \
    10.0.22.10:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and
follow the instructions.

# $
```



Review

Docker Swarm
Overview
Swarm Scheduler
Deploy and
Configure Swarm
**Deploy a Service
with Swarm**

```
$ docker service create [options] image
```



Containers at Scale:Kubernetes

© 2018 by Innovation In Software Corporation



Agenda

- What is “Web Scale”?
- Container Scaling Challenges
- Overview of Kubernetes



What is “Web Scale”?



© 2018 by Innovation In Software Corporation

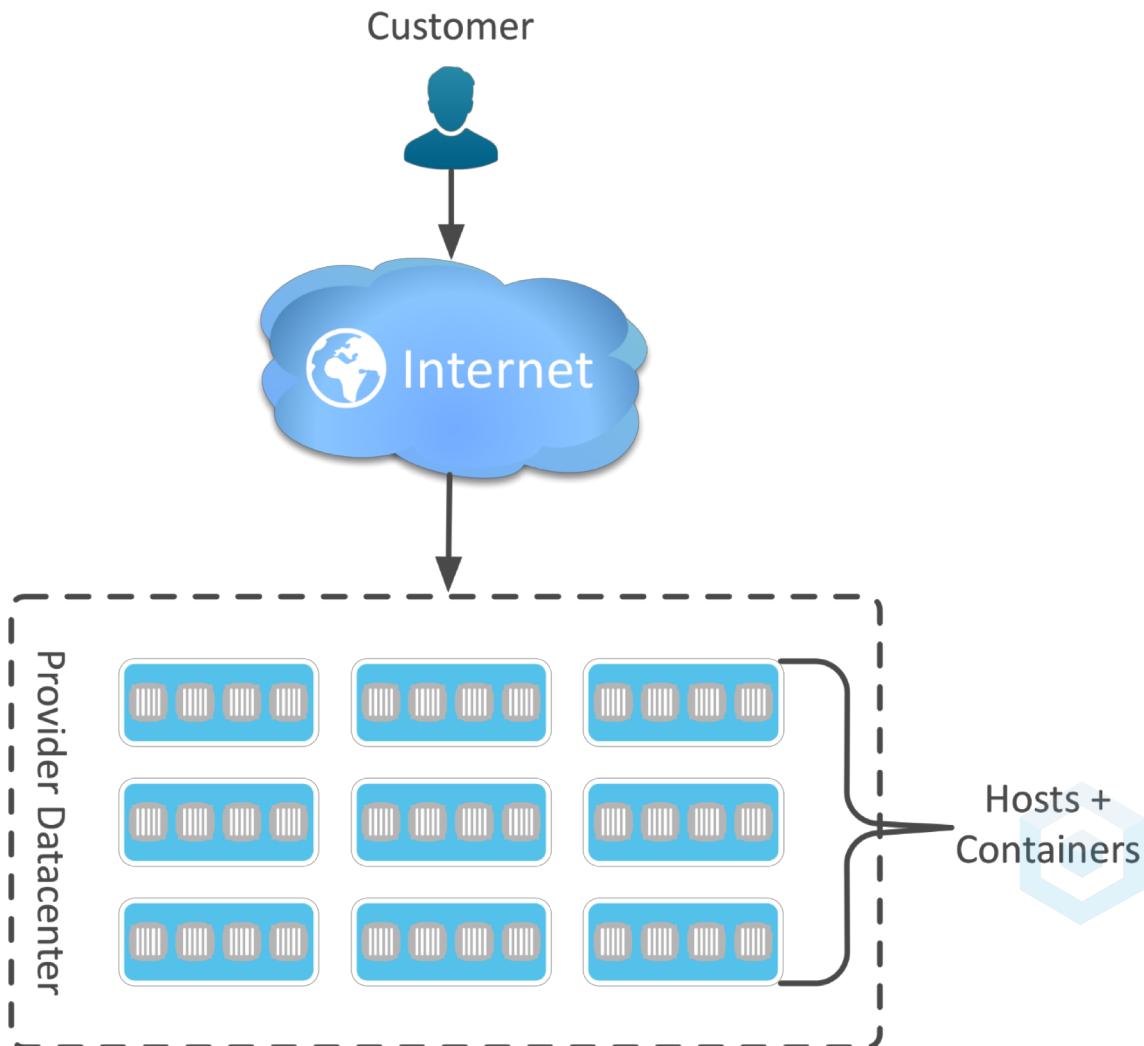


Changing Application Landscape

- Widespread Adoption of “Everything-as-a-Service”
- Rapid Feature Delivery
- Rapid Scale-Up, Scale-Down
- Limited to No Downtime
- Mostly Automated Processes

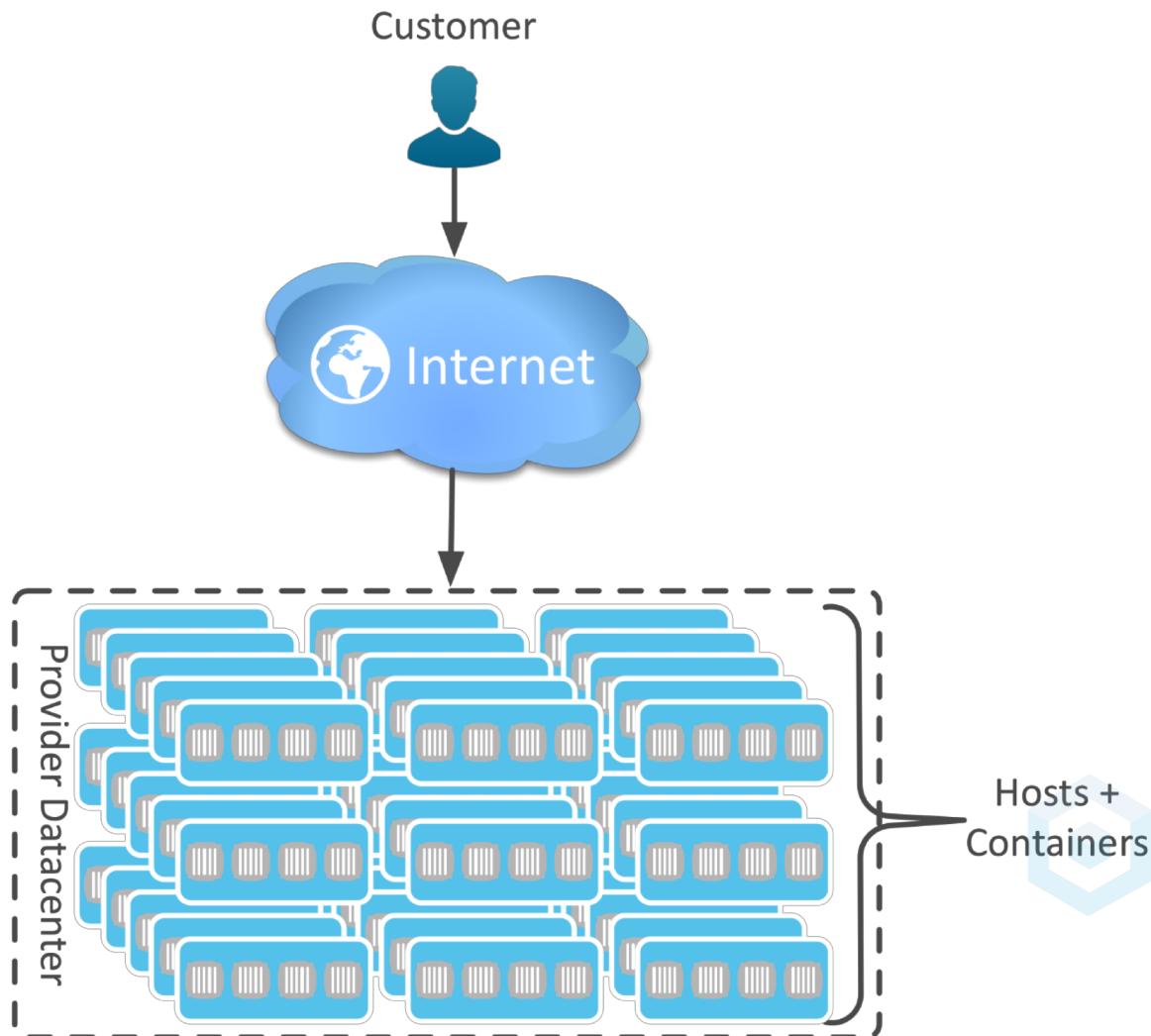


What is “Web Scale”?



INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

What is “Web Scale”?



© 2018 by Innovation In Software Corporation

INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

What is “Web Scale”?

YAHOO!

Kubernetes – Production Implementation

- 1,000 Hosts
- 42,000 Containers
- ~2,500 Applications



What is “Web Scale”?



Kubernetes – Production Implementation

- 430 Development Teams
- 2,000 Applications
- Cluster/Container Size
 - Educated Guess
 - ~1,000 Hosts, ~40,000 Containers

“The size of the JPMC application portfolio is large. We needed a proven, enterprise grade private cloud PaaS that could handle our scale for both .NET and Java. Apprenda has the technology that could deliver on the private PaaS vision of savings and agility, transforming the way we develop and run applications firmwide.”

– Ian Penny, JPMorgan Chase

INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

Container Scaling Challenges

© 2018 by Innovation In Software Corporation



Scaling Challenges

- **Automation, Automation, Automation**
- **Centralized Management**
 - Endpoints for Scheduling Containers
 - Declarative Deployment Definitions
- **Monitoring**
 - What's Running Where?
 - Event Correlation
 - Host/Cluster Health
 - Container/Application Health
- **Logs**
 - Centralized
 - Indexed



Overview of Kubernetes

© 2018 by Innovation In Software Corporation



What is Kubernetes



Kubernetes is an open source orchestration system for Docker containers.

Handles scheduling onto nodes in a compute cluster and actively manages workloads to ensure their state matches the users declared intention.

- Provides:
 - Discovery
 - Container grouping
 - Load balancing
 - Auto healing
 - Scaling
- Inspired by the technology that drives Google

Kubernetes Architecture

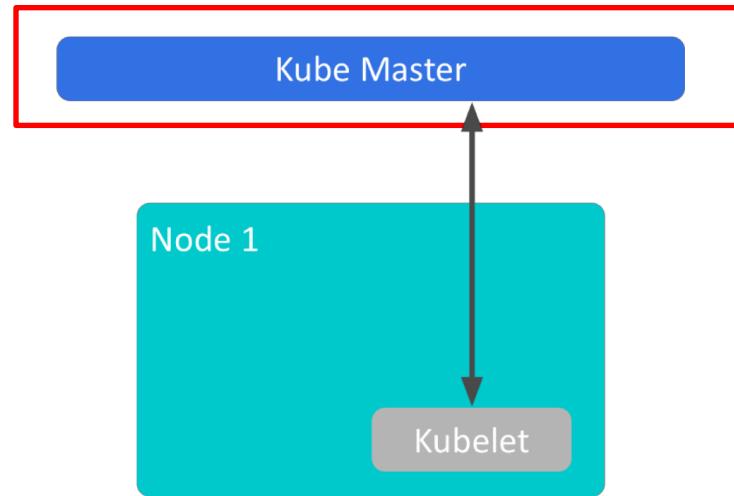
Main Components:

Master:

- This machine (or machines) manages and oversees Nodes, as well as hosts the Kubernetes APIs

Nodes:

- Worker units in the Kubernetes cluster. Responsible for running containers as instructed



Kubernetes Architecture

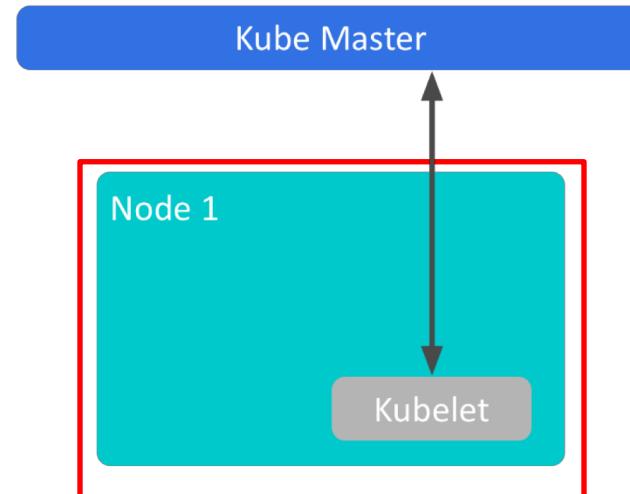
Main Components:

Master:

- This machine (or machines) manages and oversees Nodes, as well as hosts the Kubernetes APIs

Nodes:

- Worker units in the Kubernetes cluster. Responsible for running containers as instructed



Kubernetes Architecture

Main Components:

Master:

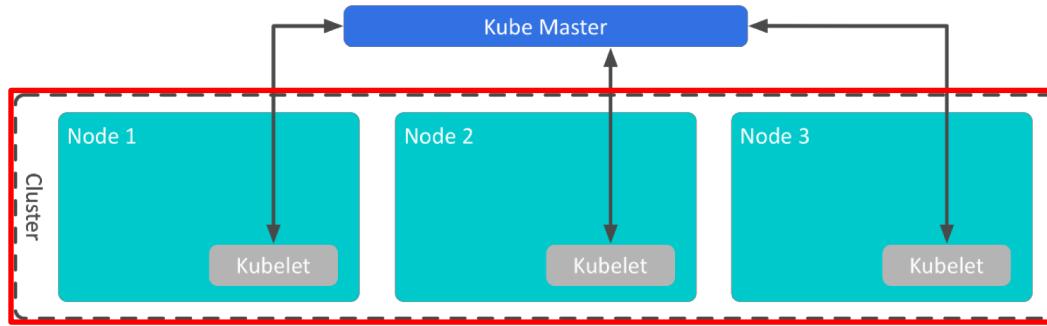
- This machine (or machines) manages and oversees Nodes, as well as hosts the Kubernetes APIs

Nodes:

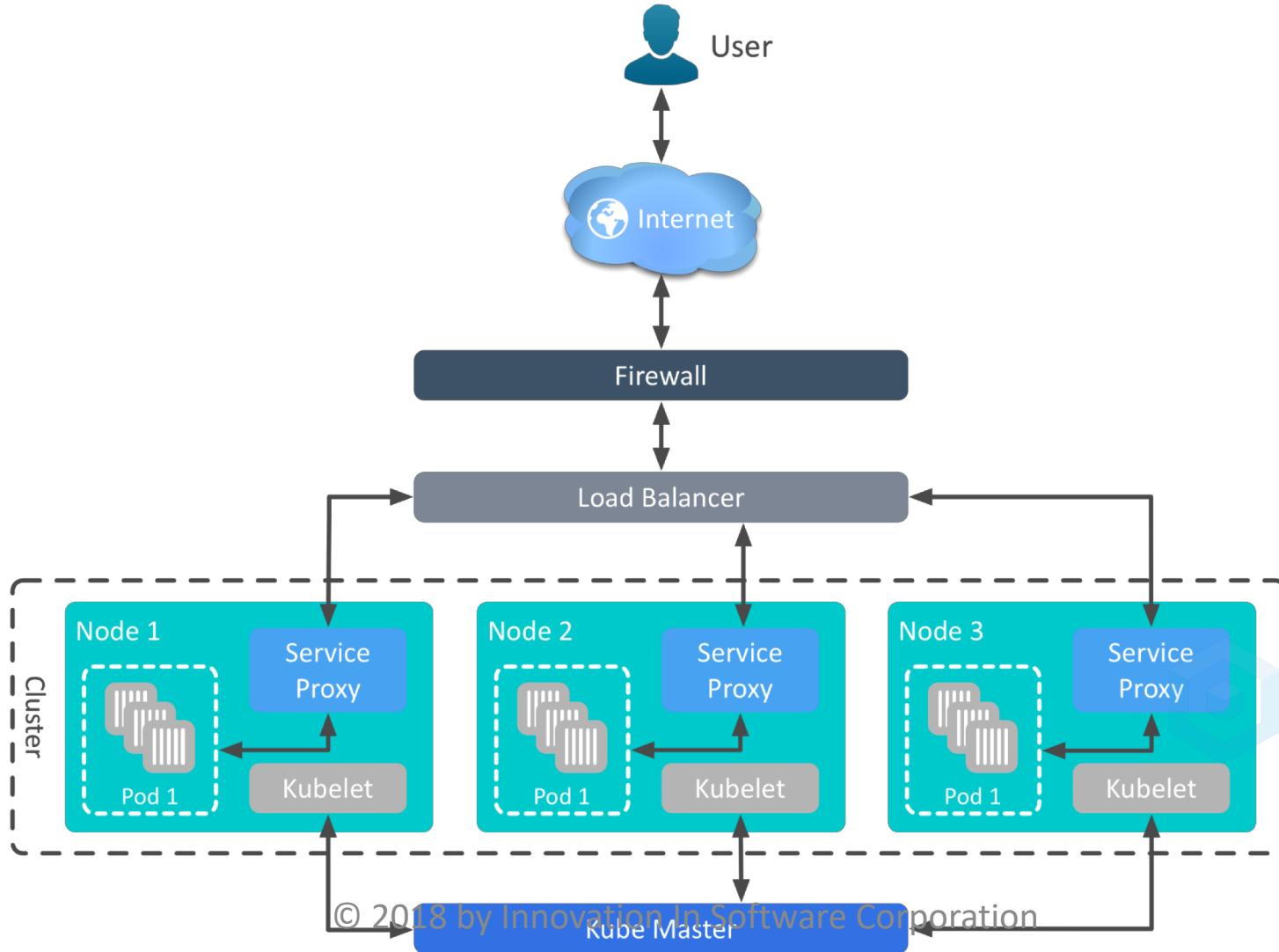
- Worker units in the Kubernetes cluster. Responsible for running containers as instructed

Clusters:

- A collection of nodes bound to a Kube Master. Managed as a single logical unit of capacity



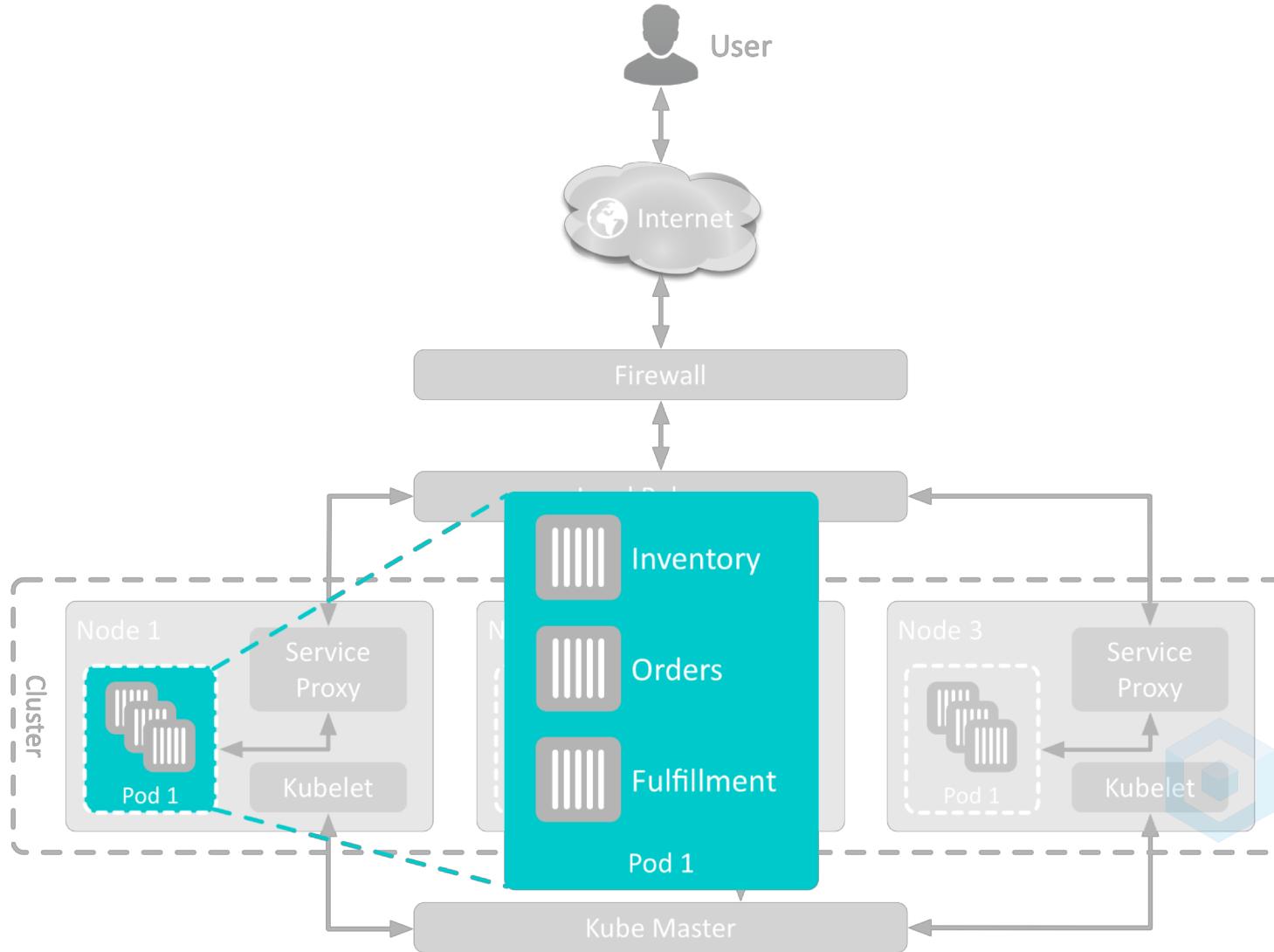
Kubernetes Architecture



INNOVATION
SOFTWARE

LEARN. EMPOWER. INNOVATE.

Work Units - Pods



INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

Work Units - Pods

The basic work unit in Kubernetes

- One or more containers to be orchestrated together
- All containers in a pod share environment
 - IP
 - Volumes
- Pod definitions are declarative

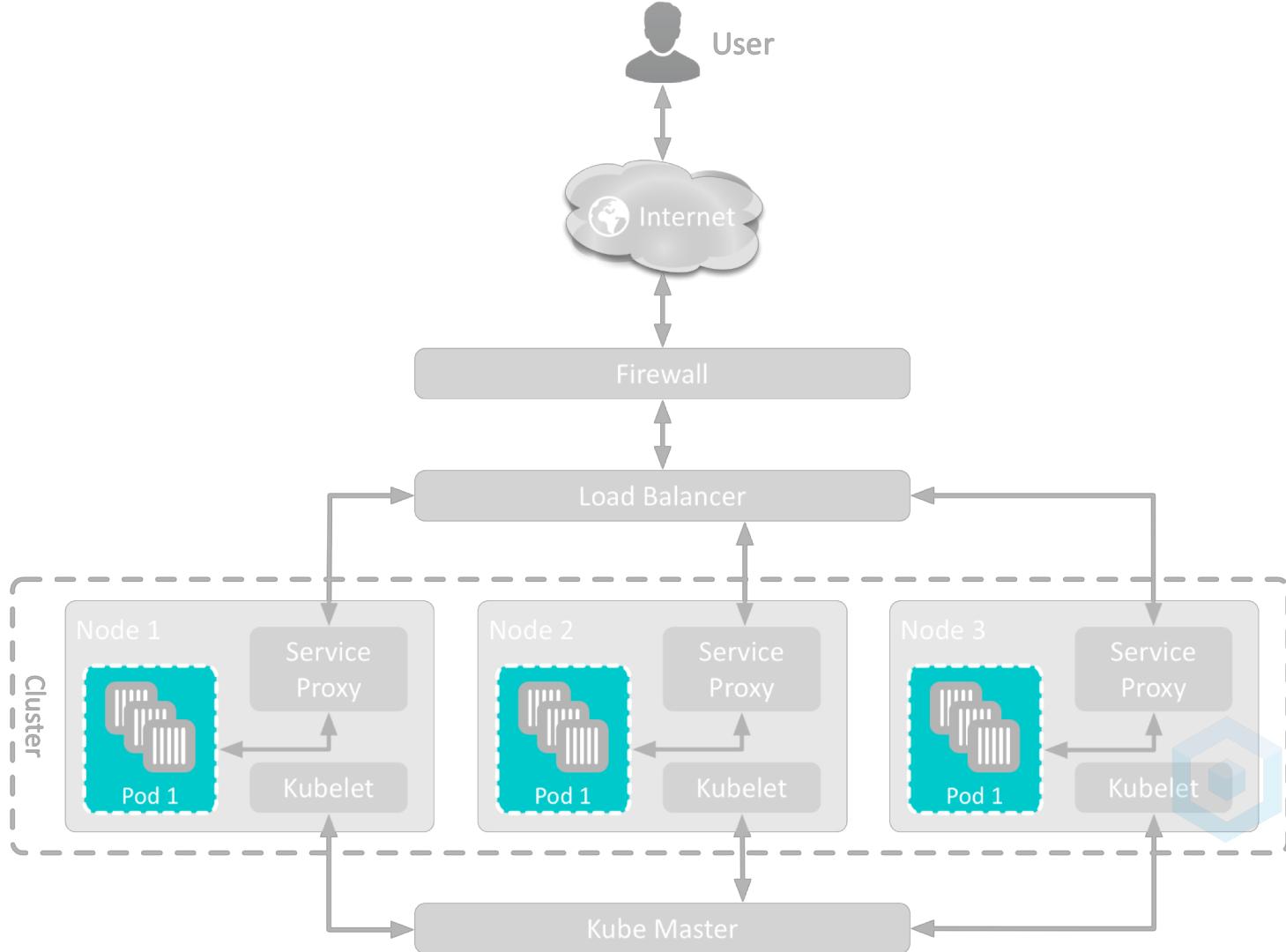
```
apiVersion: v1
kind: Pod
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  containers:
    - image: wordpress:4.4-apache
      name: wordpress
```



INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

Pod definition

Work Unit - Deployments



INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

Work Unit - Deployments

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: Wordpress
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: wordpress
    spec:
      containers:
        - name: wordpress
          image: wordpress:4.4-apache
          ports:
            - containerPort: 80
```

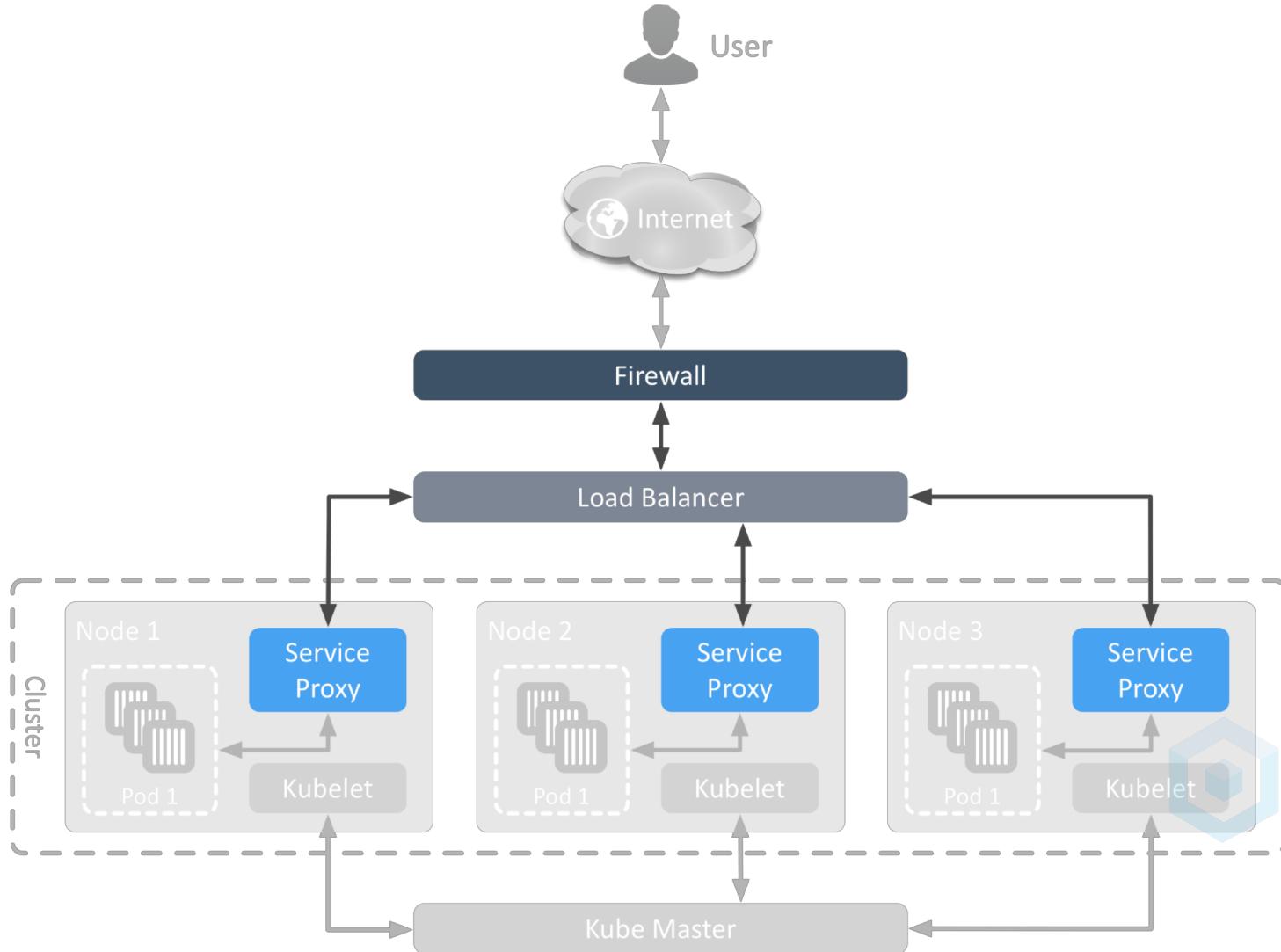
Deployment Definition

Deployments control a **set of pod replicas**

- Meant to be horizontally scaled
- Deployments will maintain desired number of copies
- If Pods go down, they are recreated automatically



Work Unit - Services



INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

Work Unit - Services

Acts as an abstraction between layers of your application

- Tracks all pods matching specified label selector
- Provides stable IP as named load balancer for pod endpoints

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  ports:
    - port: 80
  selector:
    app: wordpress
    tier: frontend
  type: LoadBalancer
```

Service Definition



**INNOVATION
SOFTWARE**
LEARN. EMPOWER. INNOVATE.

Summary, Review & QA

© 2018 by Innovation In Software Corporation



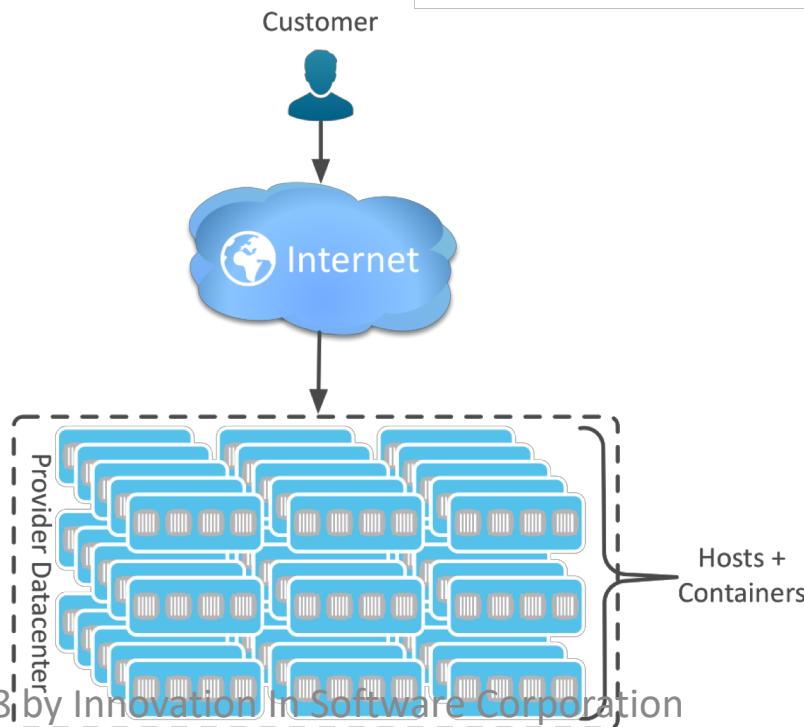
Summary

- What is “Web Scale”?
- Container Scaling Challenges
- Overview of Kubernetes



Review

- What is "Web Scale"?
- Container Scaling Challenges
- Overview of Kubernetes



Review

- What is "Web Scale"?
- Container Scaling Challenges
- Overview of Kubernetes

- **Automation, Automation, Automation**
- **Centralized Management**
 - Endpoints for Scheduling Containers
 - Declarative Deployment Definitions
- **Monitoring**
 - What's Running Where?
 - Event Correlation
 - Host/Cluster Health
 - Container/Application Health
- **Logs**
 - Centralized
 - Indexed



**INNOVATION
SOFTWARE**
LEARN. EMPOWER. INNOVATE.

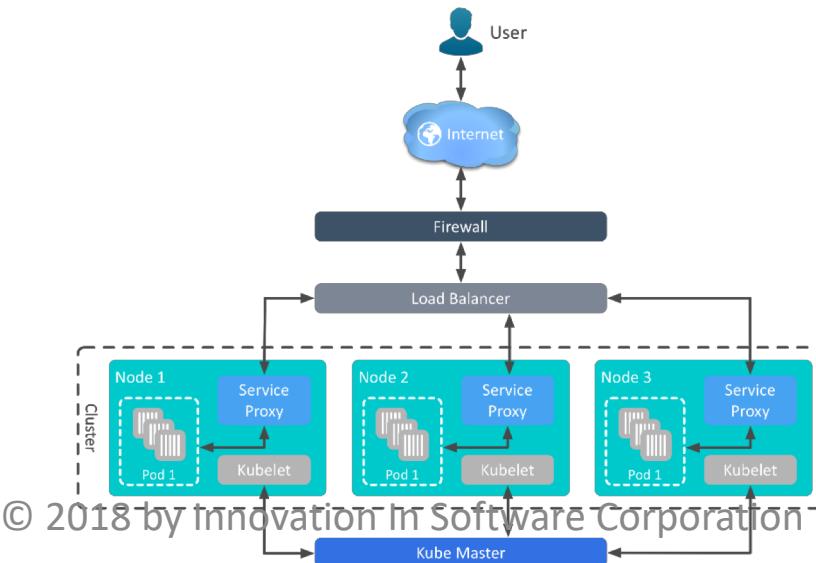
Review

- What is "Web Scale"?
- Container Scaling Challenges
- Overview of Kubernetes

 **Kubernetes** is an open source orchestration system for Docker containers.

Handles scheduling onto nodes in a compute cluster and actively manages workloads to ensure their state matches the users declared intention.

- Provides:
 - Discovery
 - Container grouping
 - Load balancing
 - Auto healing
 - Scaling
- Inspired by the technology that drives Google





Questions



Enterprise Implementation



© 2018 by Innovation In Software Corporation



How to Get Started

There are many ways to implement the Docker platform. Your selection should be based on the architecture and functional requirements of your application.

Identifying the right applications and the right environment is the first step.



Agenda

- Docker Ready Workloads
- Docker &
 - Private Cloud
 - Public Cloud
 - Hybrid Cloud



Docker Ready Workloads

Legacy

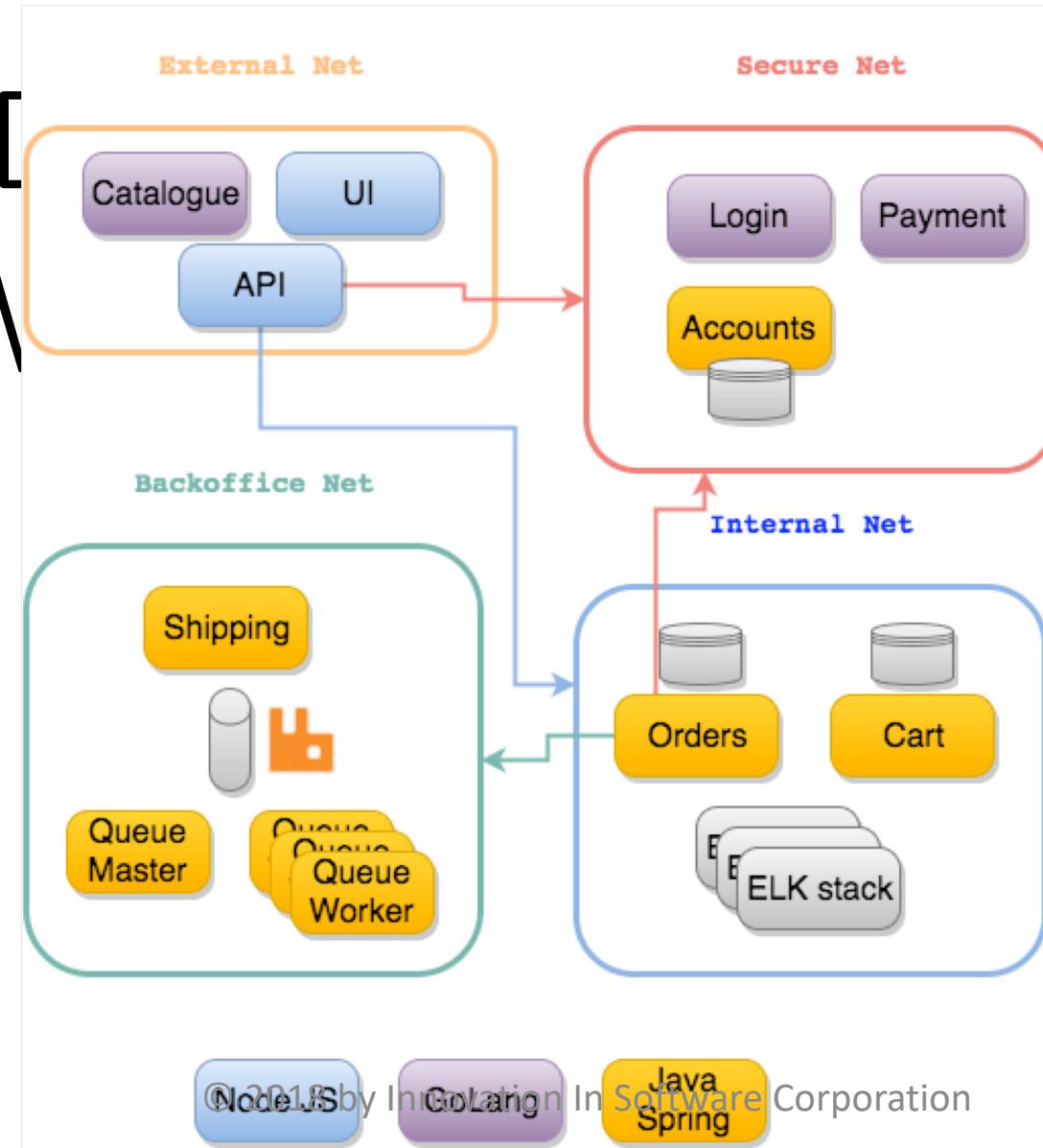
- Long Lived
- Stateful
- Dependent
- Monolithic
- Built Manually

Docker Ready

- Ephemeral
- Stateless
- Self-Contained
- Atomic
- CI Builds



Docker Ready Workloads



Containers & Cloud

© 2018 by Innovation In Software Corporation



Containers & Cloud

IaaS

- Many Providers
- Flexible Configuration
- Requires OS Management



and many, many more...



Containers & Cloud

IaaS

- Many Providers
- Flexible Configuration
- Requires OS Management



Google Cloud Platform



PaaS

- Fewer Providers
- Fewer Configuration Options
- No OS Management



IBM Bluemix



Pivotal
Web Services



INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

Docker & Cloud IaaS

- Assess the Network Impact of Containerized Application Components
 - Communication Patterns
 - Security Boundaries
- Assess the Monitoring Impact of Containerized Applications Components
- Start Small, Fail Fast, Iterate Rapidly (Agile)
 - Begin in DEV, Test & Promote
- Standardize Your Docker Host OS
 - Minimal is Better
- Standardize on a Base Image
 - Version Control the Dockerfile



Docker & Cloud PaaS

- Understand the Platform Limits
 - Can't Alter Certain Environment Settings
- Thoroughly Evaluate Your Monitoring Options
- Assess Security Impact/Compliance Impact

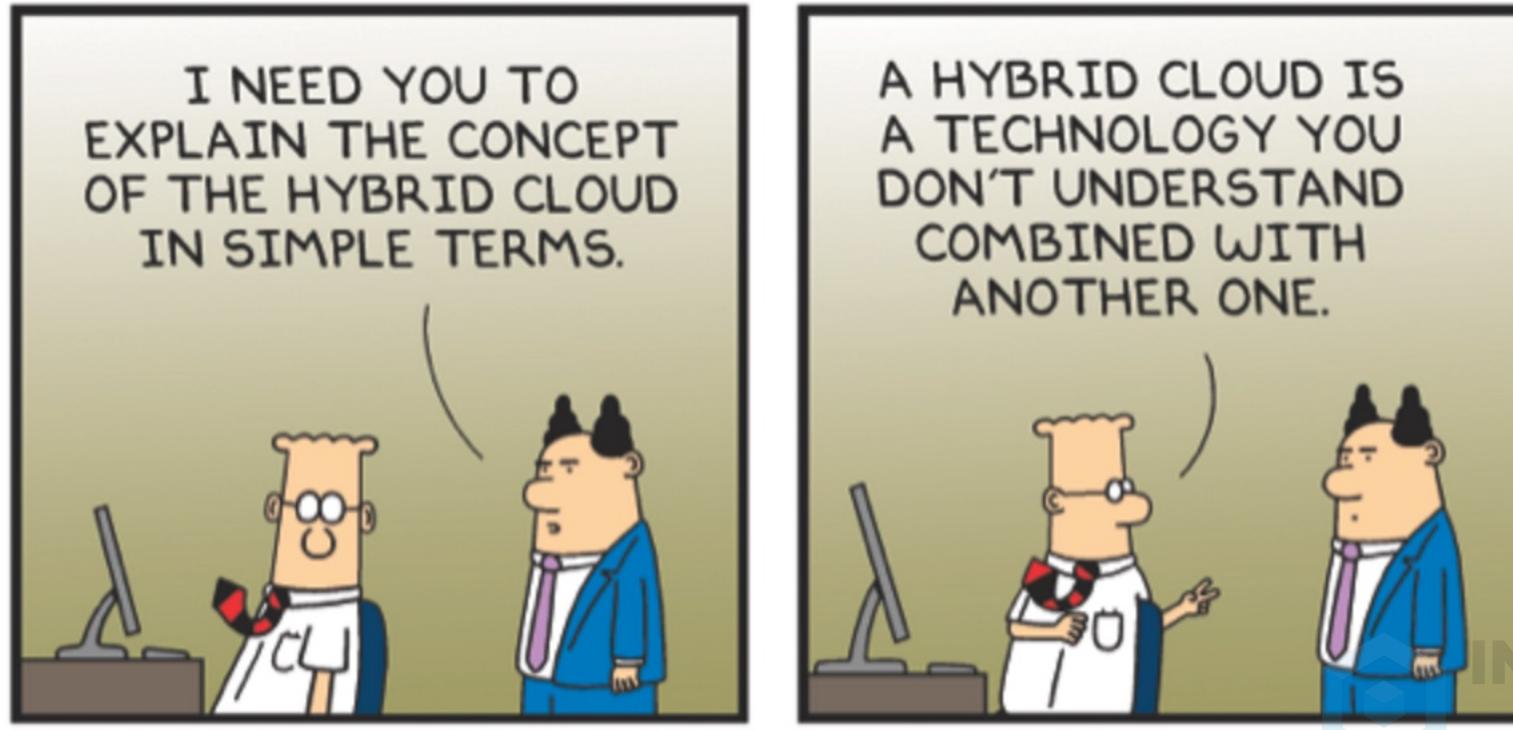


Containers & Hybrid Cloud

© 2018 by Innovation In Software Corporation

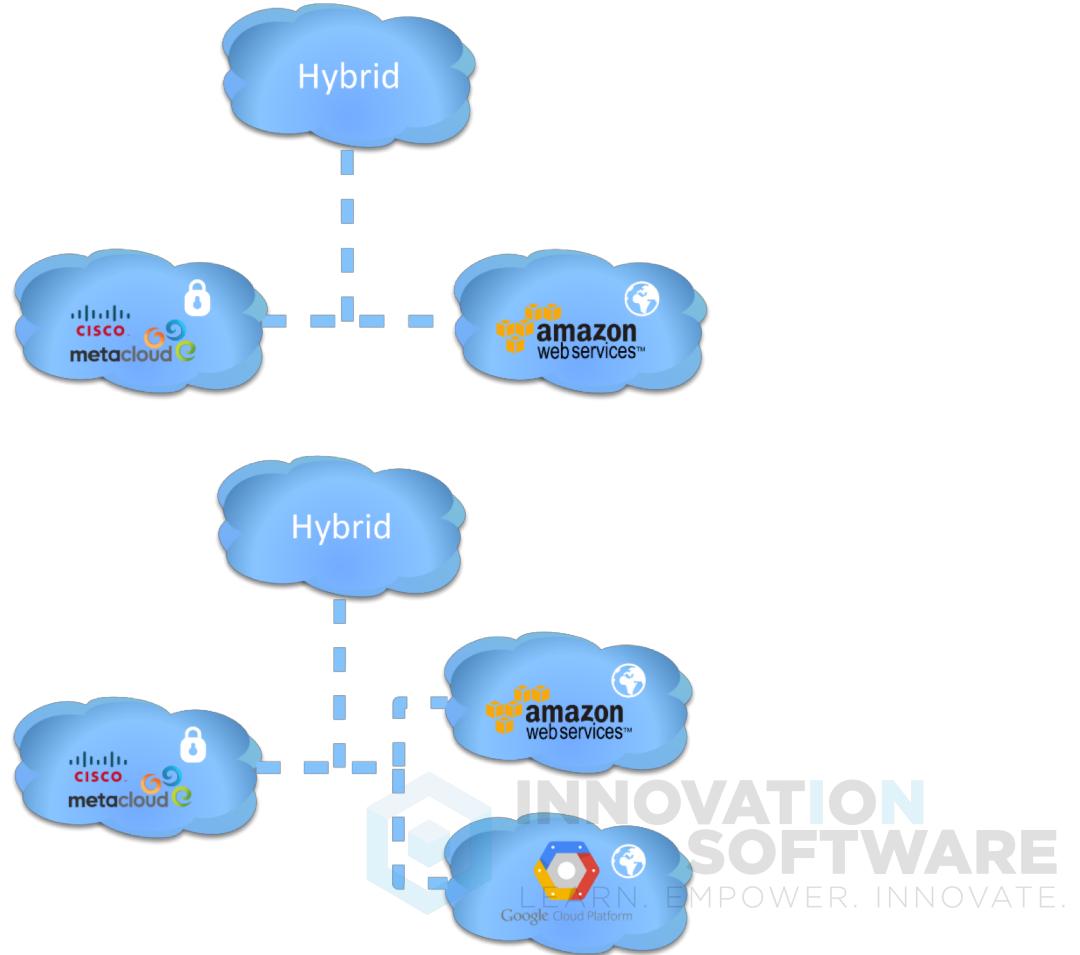


Hybrid Cloud – what does this mean?



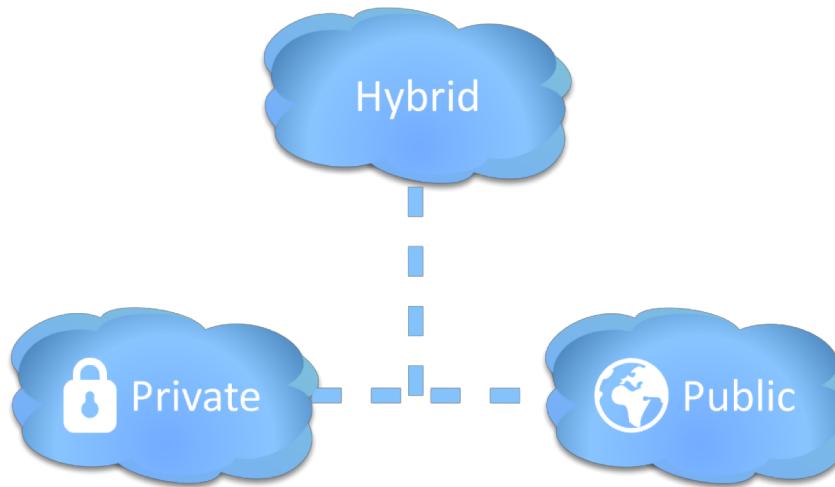
Hybrid Cloud – Advantages

- **Scalability**
 - Elastic Capacity
- **Best-Fit Placement**
 - Place Work on the Cloud with the Best Feature Set
- **Cost Optimization**
 - Pick the Most Cost-Effective Cloud
 - Repatriating High Cost Workloads
- **Risk Mitigation**
 - Place Public-Facing Systems Outside the Private Cloud



Hybrid Cloud – Disadvantages

- **Visibility**
 - What's Running Where?
- **Cost**
 - Quotas
 - Projections
- **Placement**
 - What's the Best Fit?
- **Application Modeling**
 - Many DSLs



Global Media Company

Global Media Company

Challenge	Solution	Outcomes
<ul style="list-style-type: none">Inability to scale on-demand in line with programming and customer trendsLegacy infrastructure inadequate to support growthOrganizational and legacy processes constrain adoption of new technology	<ul style="list-style-type: none">Hybrid solution: private cloud with elastic scaling to the public cloudImplemented DevOps processes and toolchain to accelerate application deploymentPrioritized apps for cloud deployment based on quick wins and long-term gains	<ul style="list-style-type: none">Integrated into current process controls to meet security, compliance and auditExtended the DevOps toolchain to include containers and microservicesTrained engineers, admins & operators to operate and extend the hybrid cloud
<ul style="list-style-type: none">The upgrade process for their tightly coupled applications was expanding the deployment times.Development teams were on hold while waiting for other services to be completedQA / Test cycles were growing.	<ul style="list-style-type: none">Capitalize on their CI/CD investment.Java based microservices that are deployed in containers and orchestrated via Kubernetes.Deployments run against Amazon Web Services as well as OpenStack.	<ul style="list-style-type: none">The loosely coupled framework has reduced the dependencies in the application stack and greatly streamlined deployments and upgrades.Streamlined deployments now that a single artifact moves from Dev to Production



Questions



Docker for CI & CD

© 2018 by Innovation In Software Corporation



Agenda

- What is CI & CD?
- Docker Containers for CI
- Docker Containers for CD



What is CI & CD?



© 2018 by Innovation In Software Corporation





Development

Focus on what is inside
the container

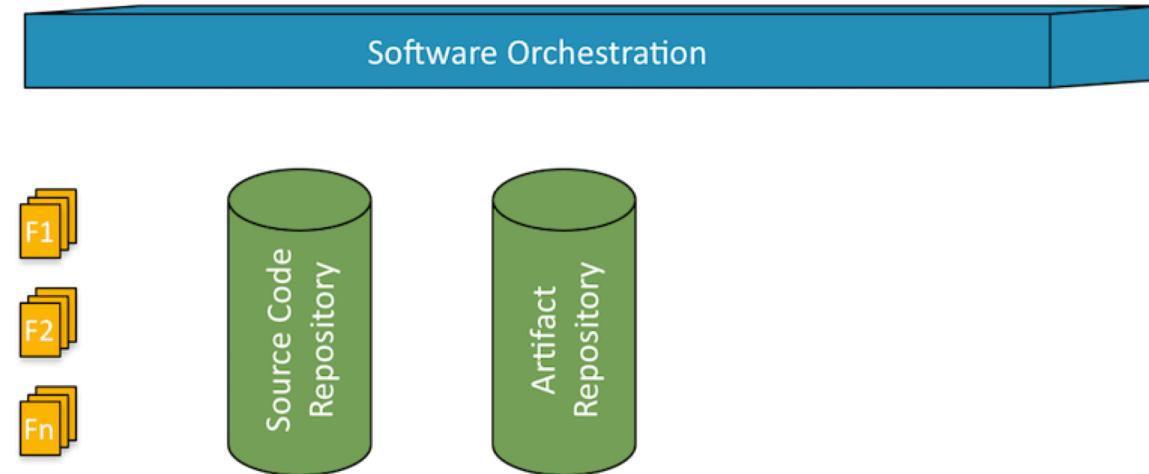


Operations

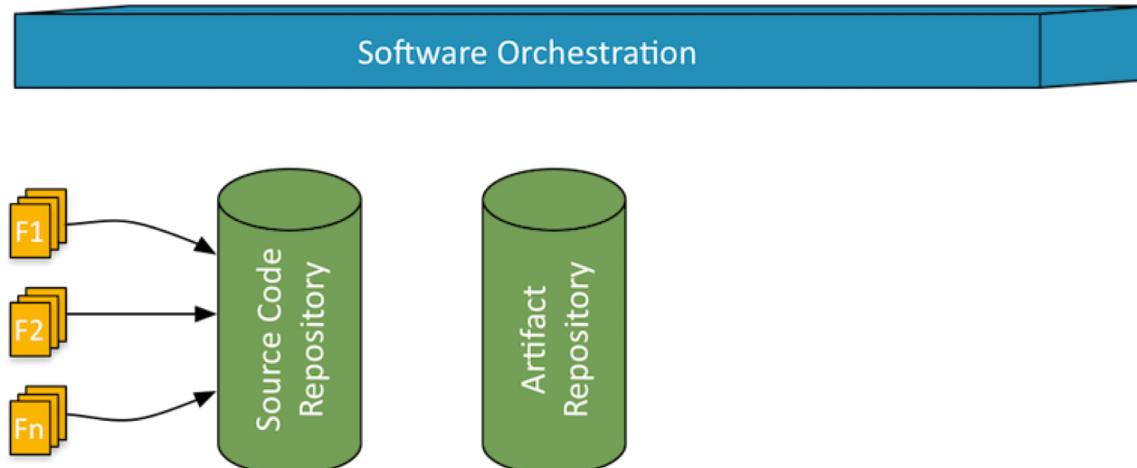
Focus on placing and
maintaining containerized
code in production



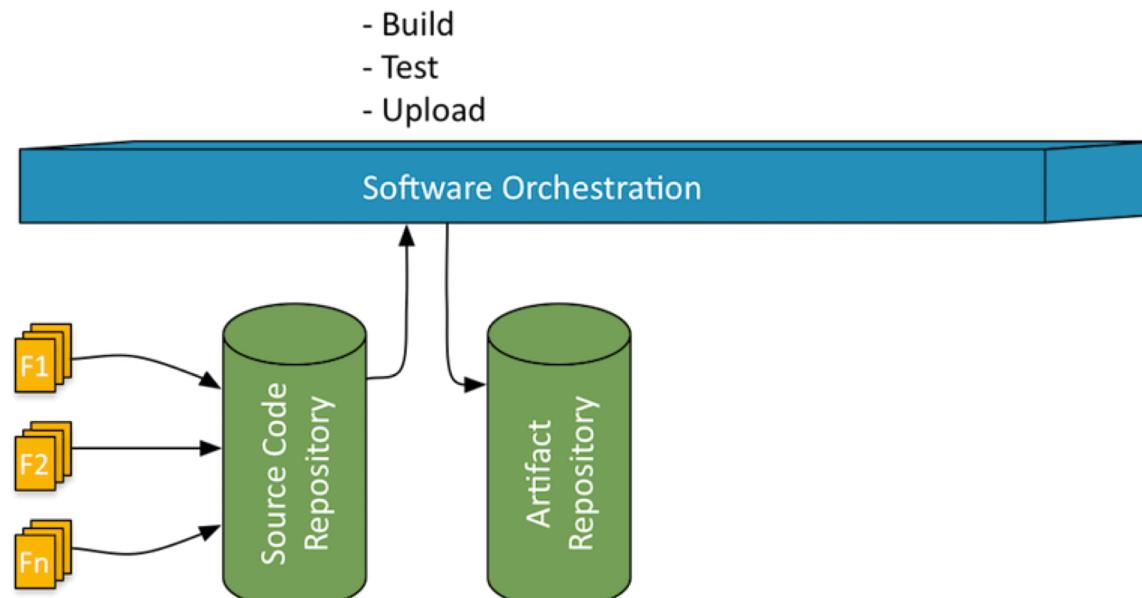
What is Continuous Integration (CI)?



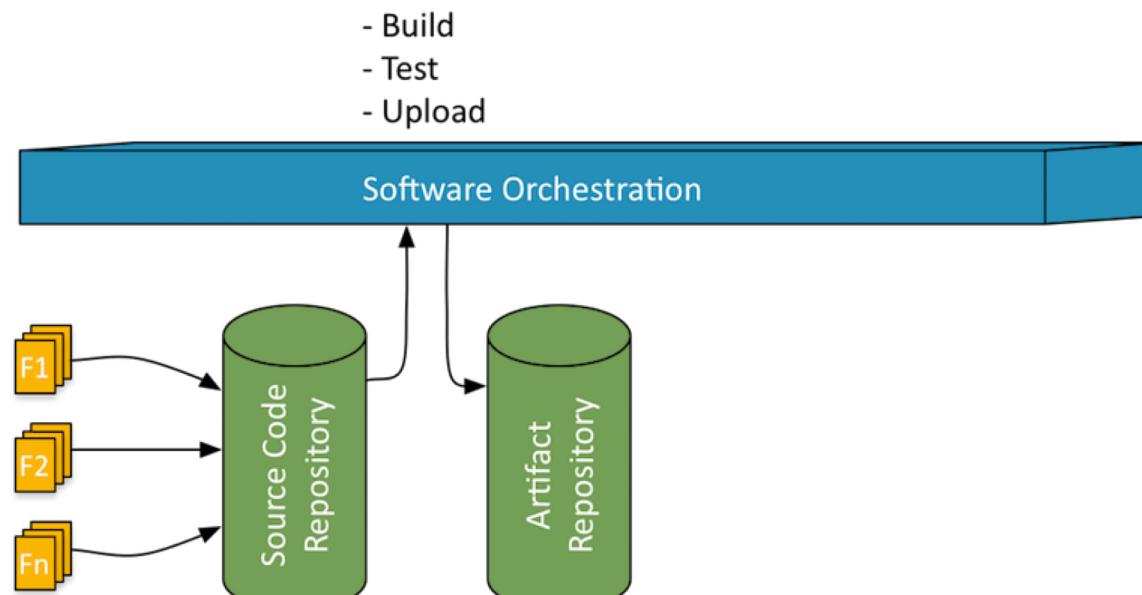
What is Continuous Integration (CI)?



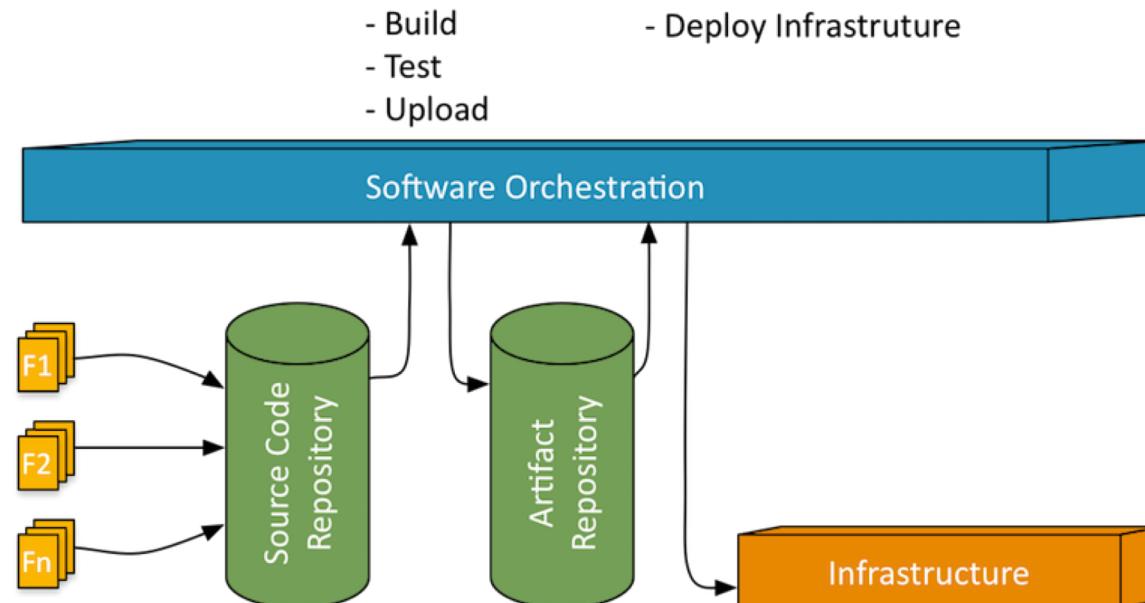
What is Continuous Integration (CI)?



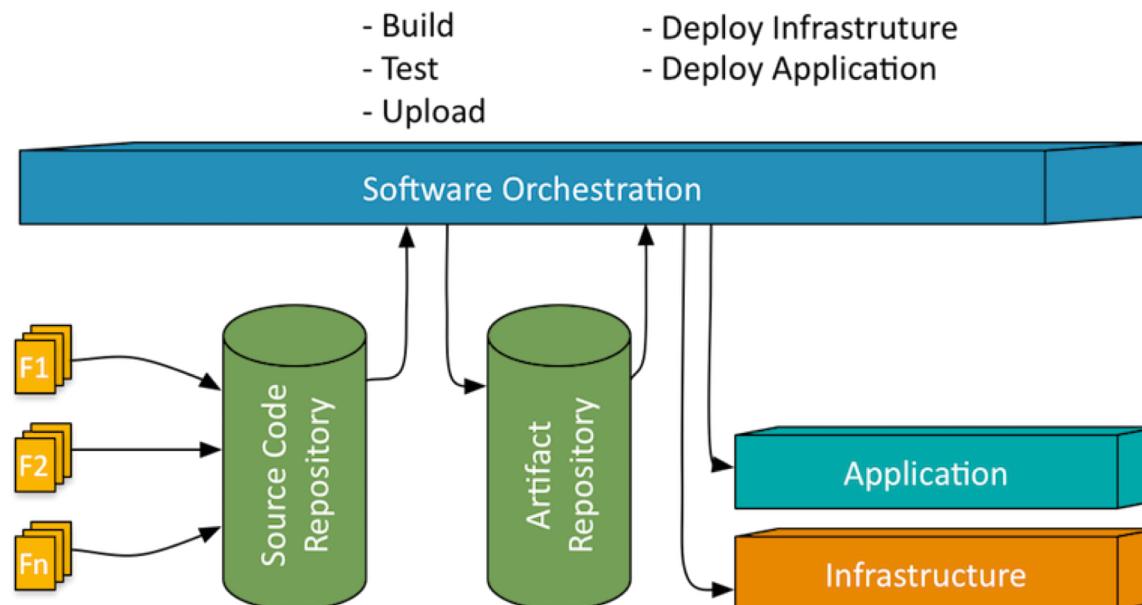
What is Continuous Delivery (CD)?



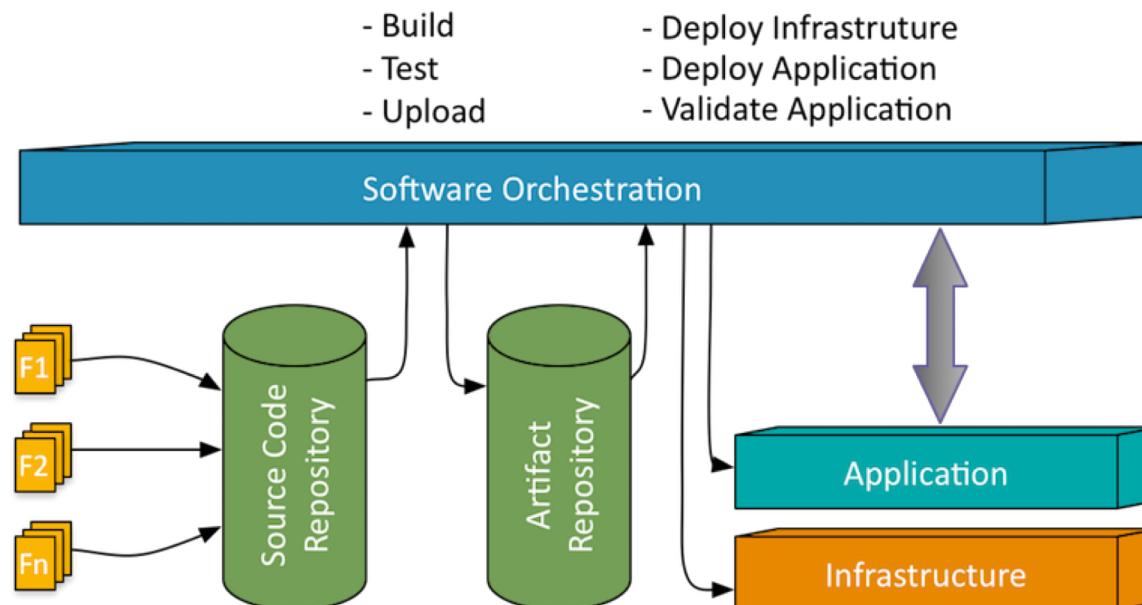
What is Continuous Delivery (CD)?



What is Continuous Delivery (CD)?



What is Continuous Delivery (CD)?



What's the Difference?

Continuous Integration:

- Frequent integrations of the code base
- Small, iterative change sets
- Automated builds for each commit
- Automated testing for validation
- Automated packaging/storage of applications

Continuous Delivery:

- Frequent deliveries
- Small, integrated change sets
- Automated deployment of Application Infrastructure
- Automated deployment of Applications
- Automated Application Testing/Validation



Docker Containers for CI

© 2018 by Innovation In Software Corporation

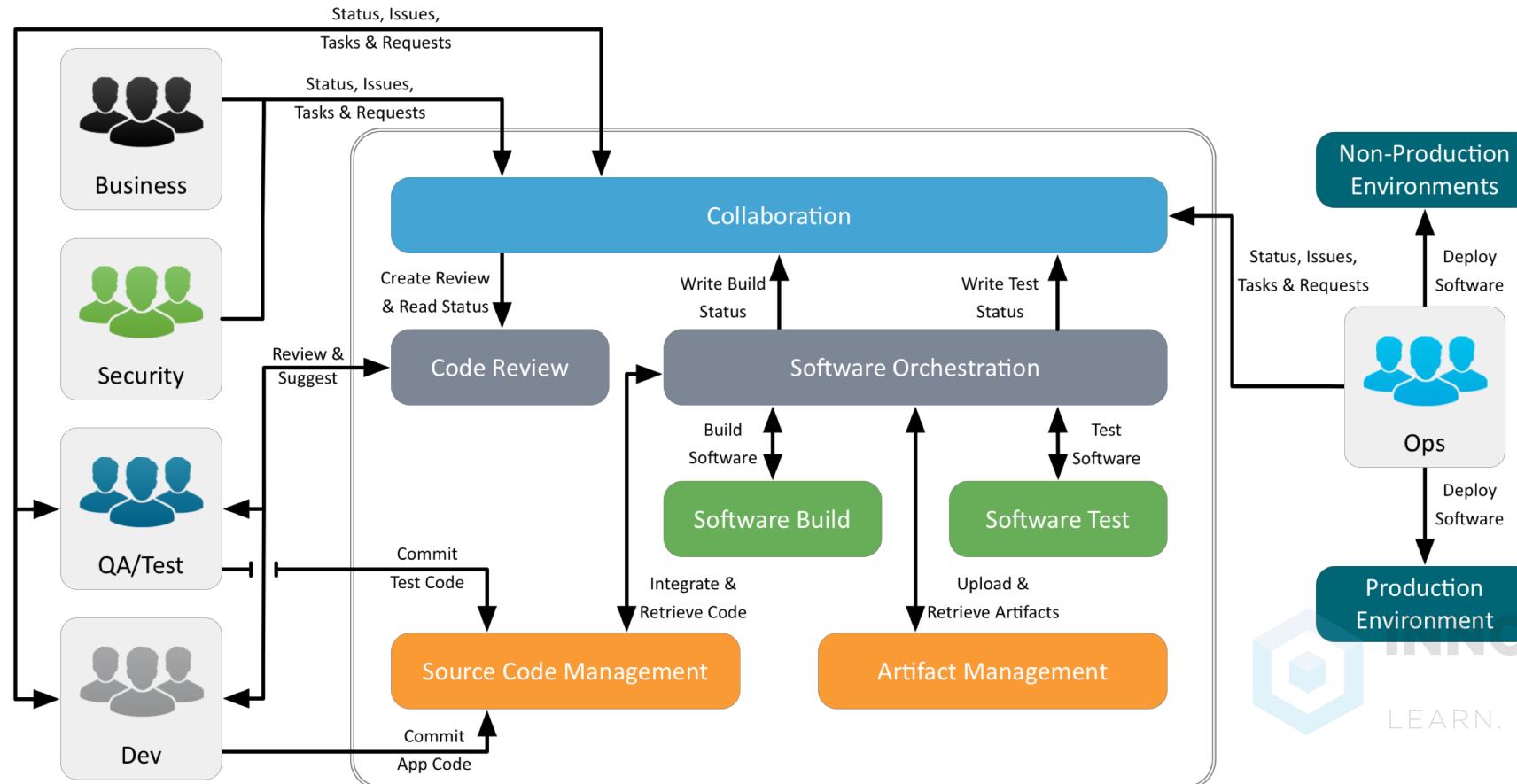


Docker for Continuous Integration

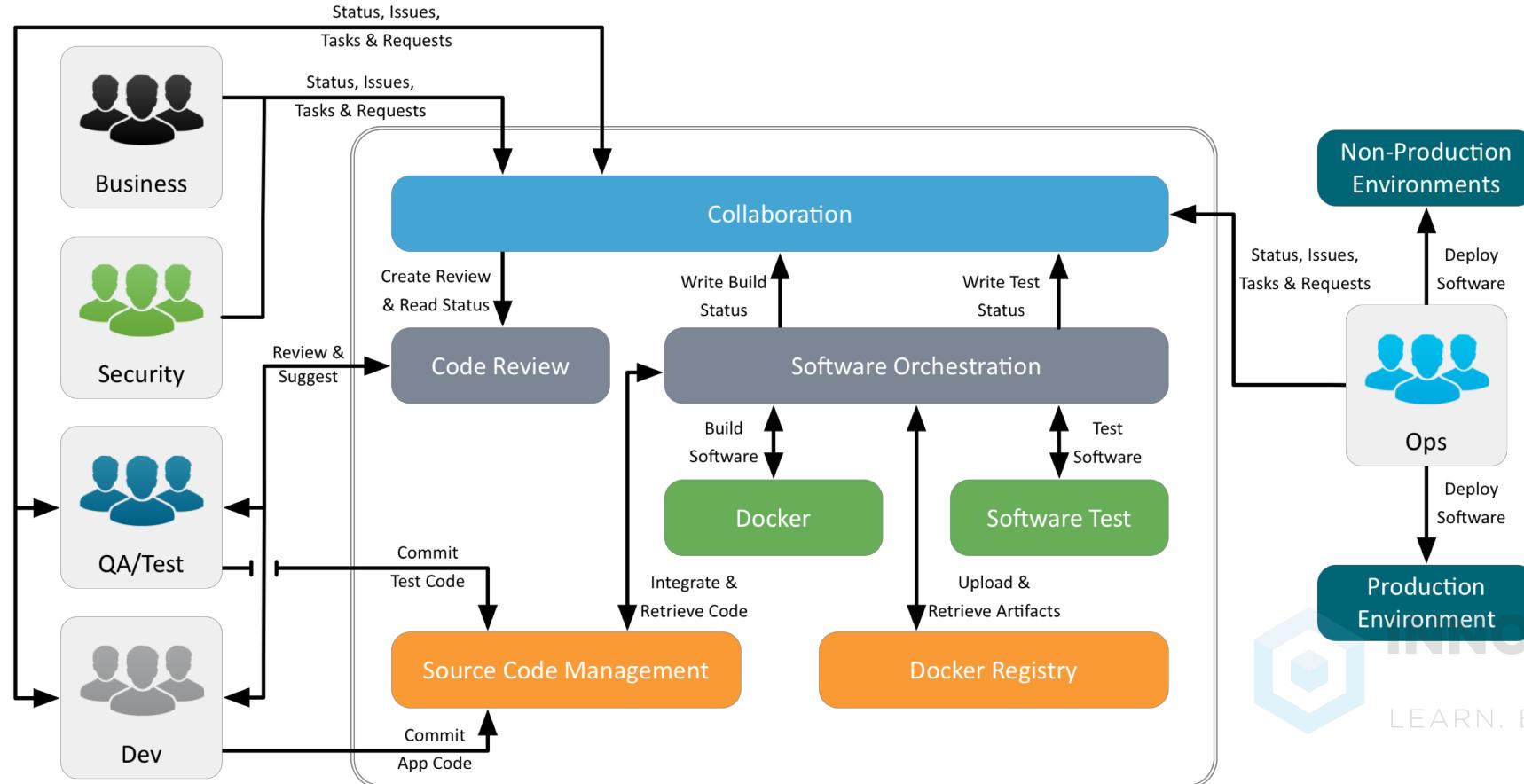
- Frequent deliveries
 - Small, integrated change sets
- Automated deployment of Application Infrastructure
- Automated deployment of Applications
- **Automated Application Testing/Validation**



Docker for Continuous Integration

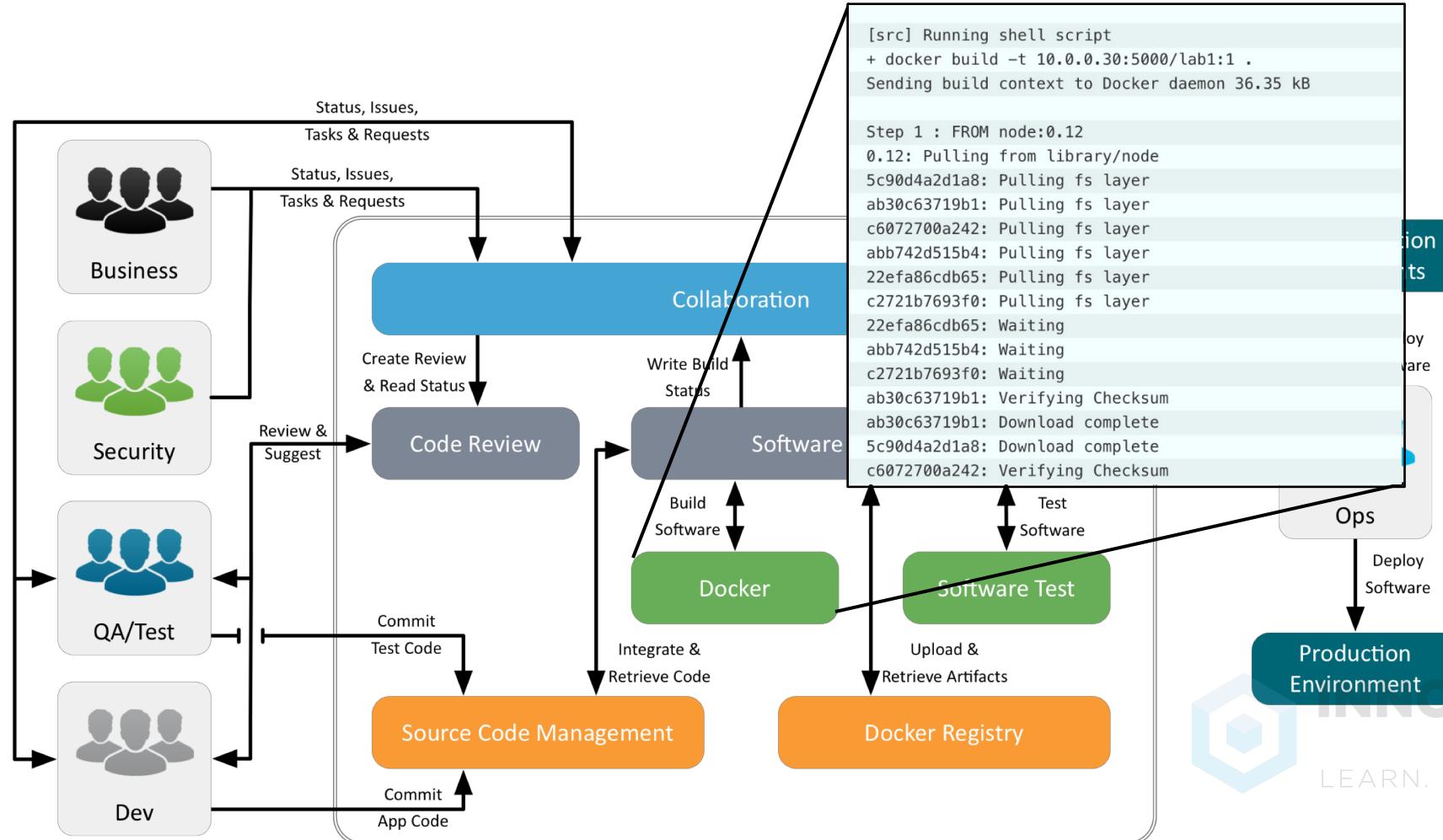


Docker for Continuous Integration

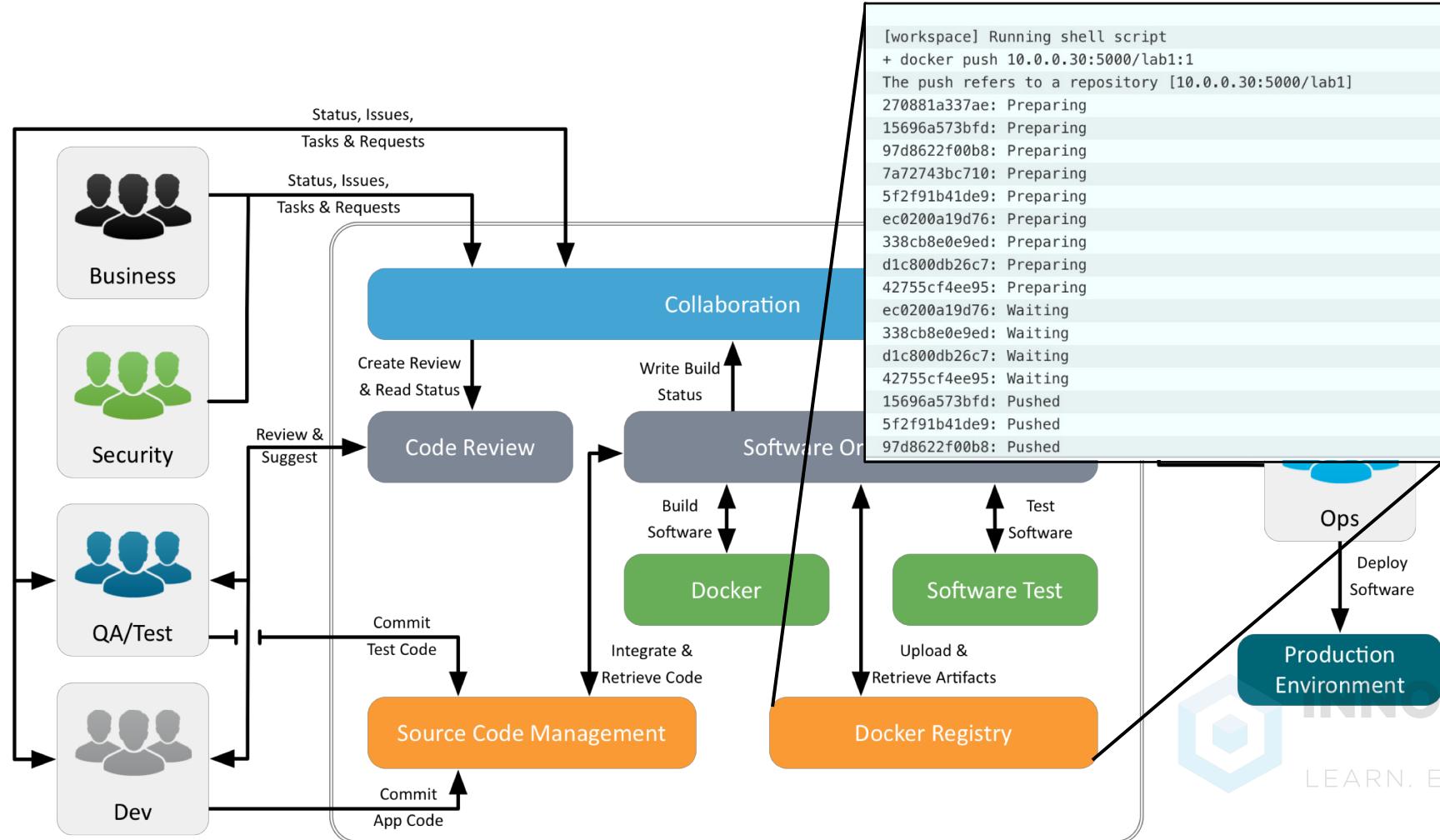


INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

Docker for Continuous Integration



Docker for Continuous Integration



Docker for Continuous Integration

Jenkins Solinea Student | log out [ENABLE AUTO REFRESH](#)

New Item People Build History Manage Jenkins My Views Credentials

All +

S	W	Name ↓	Last Success	Last Failure	Last Duration
		CI_Demo	1 min 35 sec - #3	N/A	2.5 sec

Icon: S M L Legend RSS for all RSS for failures RSS for just latest builds

Build Queue No builds in the queue.

Build Executor Status 1 Idle 2 Idle

Help us localize this page Page generated: Jul 1, 2016 5:38:43 PM UTC REST API Jenkins ver. 2.7



Docker Containers for CD

© 2018 by Innovation In Software Corporation



Docker for Continuous Delivery

- Frequent deliveries
 - Small, integrated change sets
- Automated deployment of Application Infrastructure
- **Automated deployment of Applications**
- Automated Application Testing/Validation



Docker for Continuous Delivery

Development

- Allows the ability to define the entire project configuration and tear-down/recreate it easily
- Supports multiple versions of application simultaneously

Test Environments

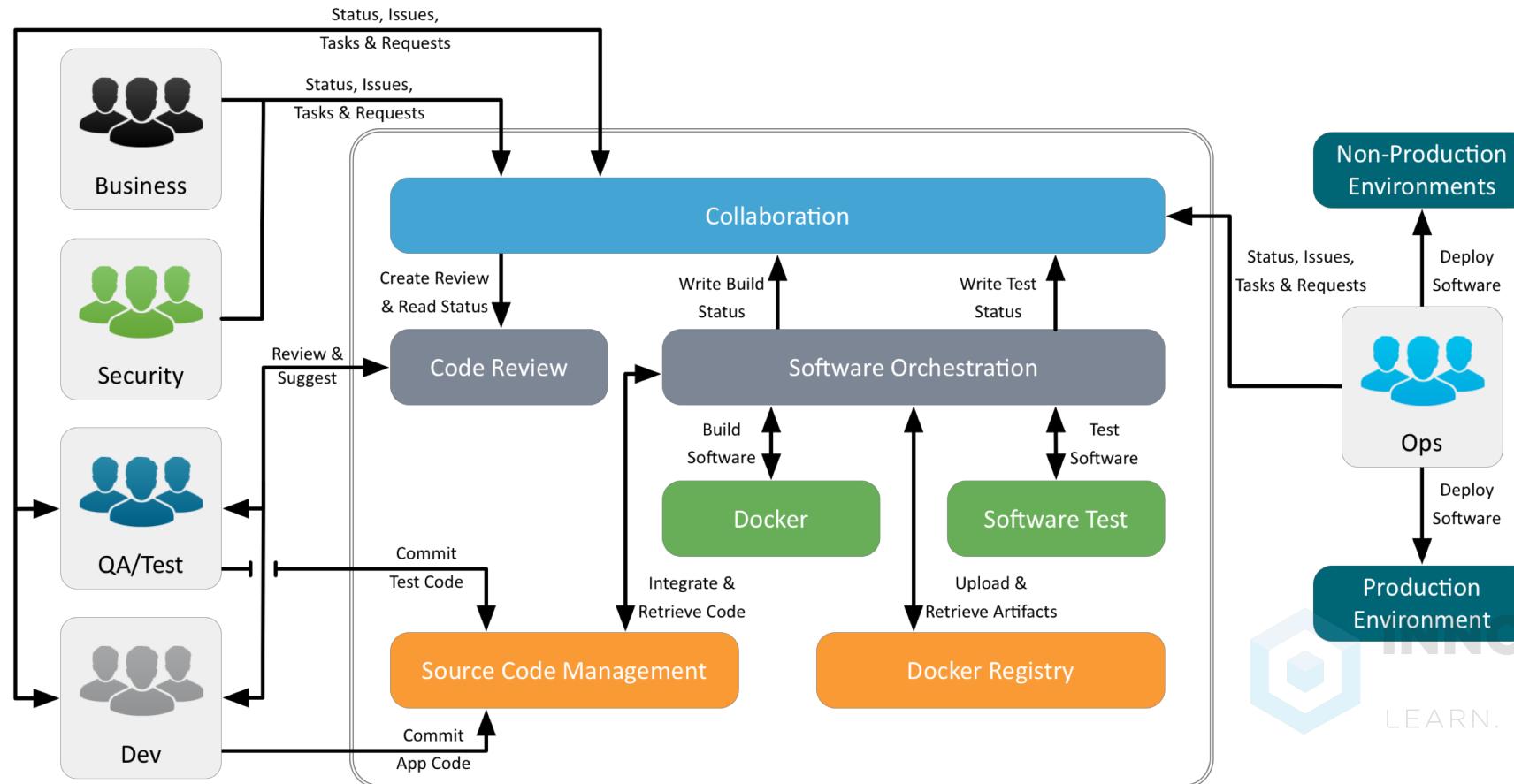
- Same container that developers run is the container that runs in test lab and production – includes all dependencies
- Well formed API allows for automated building and testing of new containers

Micro-Services

- Design applications as suites of services, each written in the best language for the task
- Better resource allocation
- One container per microservice vs. one VM per microservice
- Can define all interdependencies of services with templates

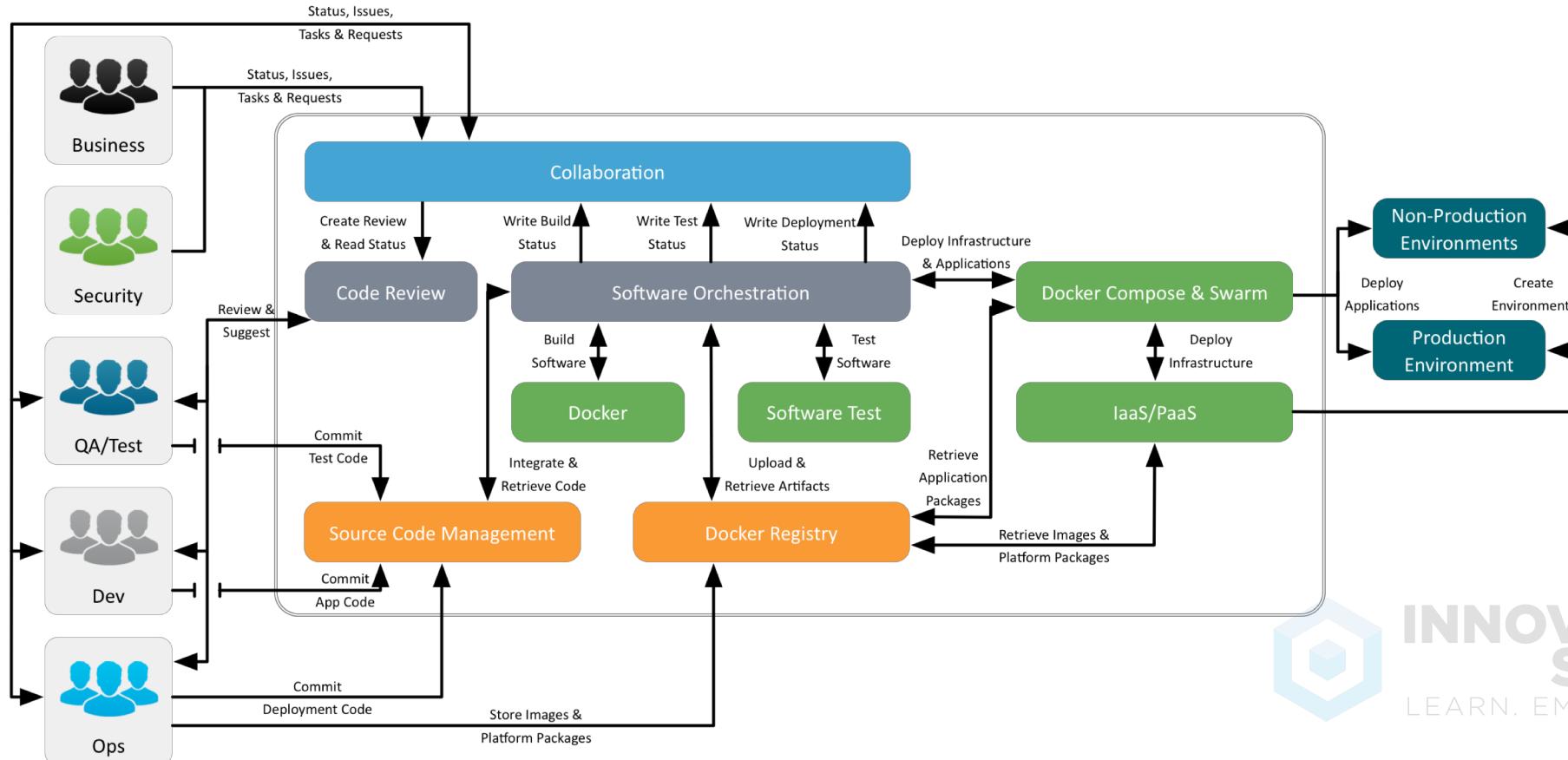


Docker for Continuous Delivery



INNOVATION
SOFTWARE
LEARN. EMPOWER. INNOVATE.

Docker for Continuous Delivery



INNOVATION SOFTWARE
LEARN. EMPOWER. INNOVATE.

Docker for Continuous Delivery

Jenkins Solinea Student log out

ENABLE AUTO REFRESH

Back to Dashboard Status Changes Build Now Delete Pipeline Configure Move Full Stage View GitHub Pipeline Syntax

Pipeline CD_Demo

Recent Changes add description

Average stage times: (Average full run time: ~18s)

Stage View

Pre-Build Housekeeping	Get Code	Run Unit Tests	Build Container	Upload Container	Deploy Container
53ms	640ms	627ms	313ms	564ms	16s
55ms	561ms	609ms	321ms	556ms	18s
56ms	610ms	611ms	296ms	555ms	15s
57ms	723ms	663ms	333ms	565ms	15s
44ms	669ms	625ms	304ms	581ms	15s

Build History trend →

#4 Jul 2, 2016 4:59 AM
#3 Jul 2, 2016 4:59 AM
#2 Jul 2, 2016 4:59 AM
#1 Jul 2, 2016 4:57 AM

RSS for all RSS for failures



INNOVATION SOFTWARE LEARN. 15S MPOWER. INNOVATE.

Summary, Review, & Q&A

© 2018 by Innovation In Software Corporation



Summary

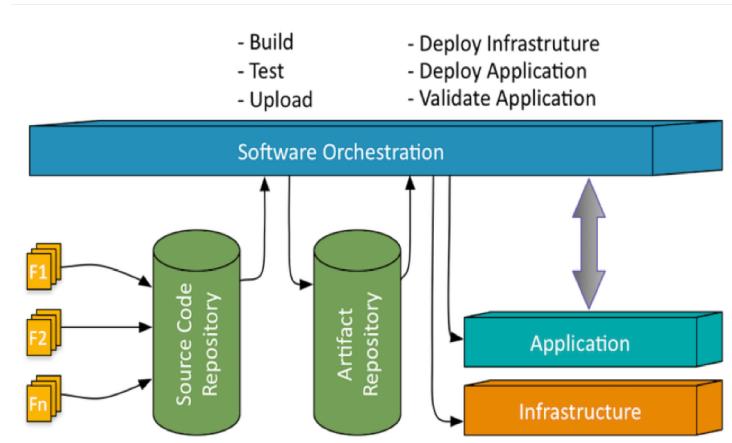
- What is CI & CD?
- Docker Containers for CI
- Docker Containers for CD



Review

What is CI & CD?

Docker Containers for
CI
Docker Containers for
CD



Continuous Integration:

- Frequent integrations of the code base
 - Small, iterative change sets
- Automated builds for each commit
- Automated testing for validation
- Automated packaging/storage of applications

Continuous Delivery:

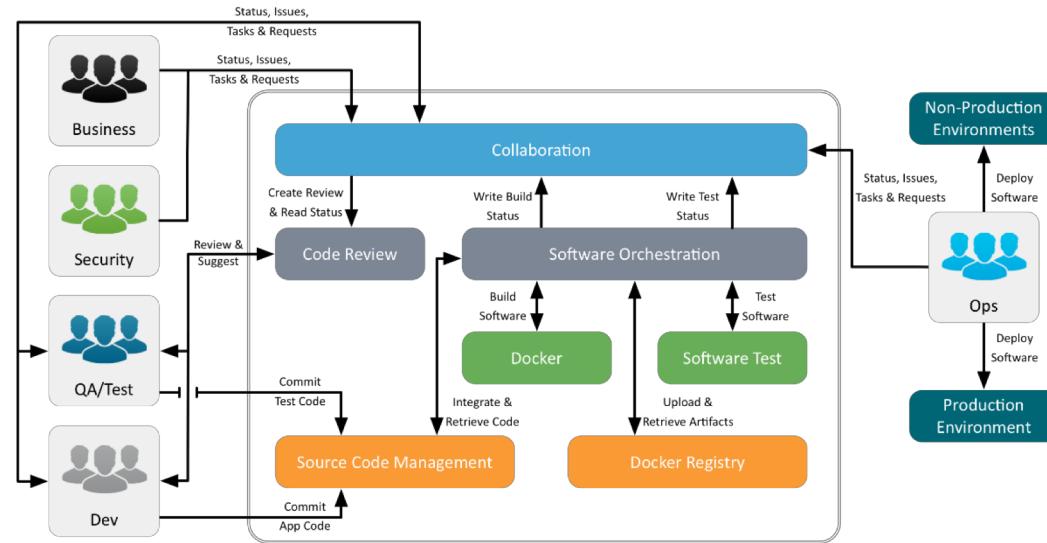
- Frequent deliveries
 - Small, integrated change sets
- Automated deployment of Application Infrastructure
- Automated deployment of Applications
- Automated Application Testing/Validation

Review

What is CI & CD?

Docker Containers for
CI

Docker Containers for
CD



Review

What is CI & CD?
Docker Containers for
CI
Docker Containers for
CD

Development

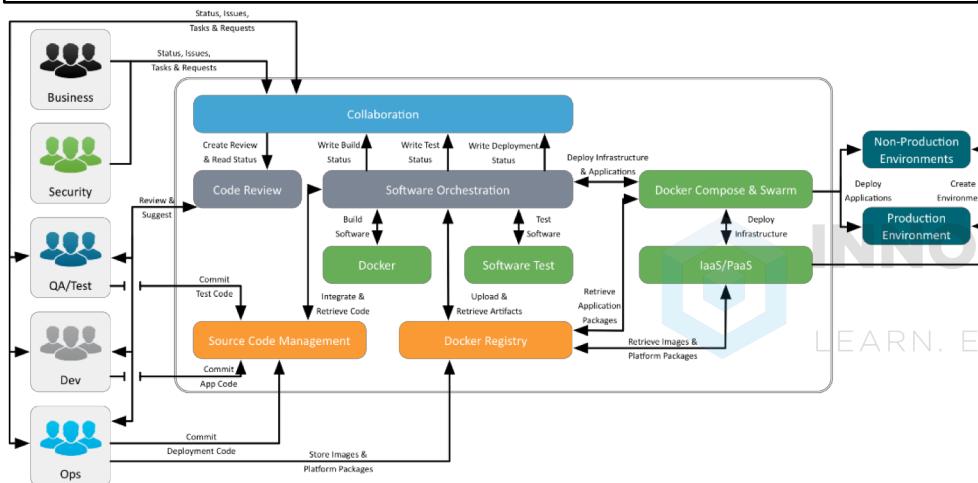
- Allows the ability to define the entire project configuration and tear-down/recreate it easily
- Supports multiple versions of application simultaneously

Test Environments

- Same container that developers run is the container that runs in test lab and production – includes all dependencies
- Well formed API allows for automated building and testing of new containers

Micro-services

- Design applications as suites of services, each written in the best language for the task
- Better resource allocation
 - One container per microservice vs. one VM per microservice
- Can define all interdependencies of services with templates





Questions





© 2018 by Innovation In Software Corporation