BITS Pilani presentation

BITS Pilani
Pilani Campus

Mridul Moitra
Cloud Computing

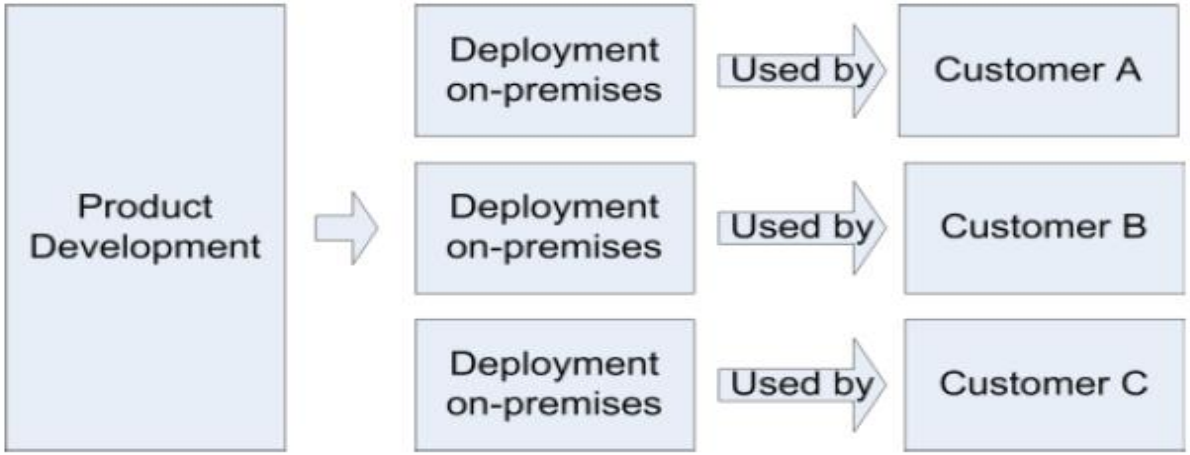
<CSI ZG527 / SS ZG527 / SE ZG527

Cloud Computing- Docker Technology

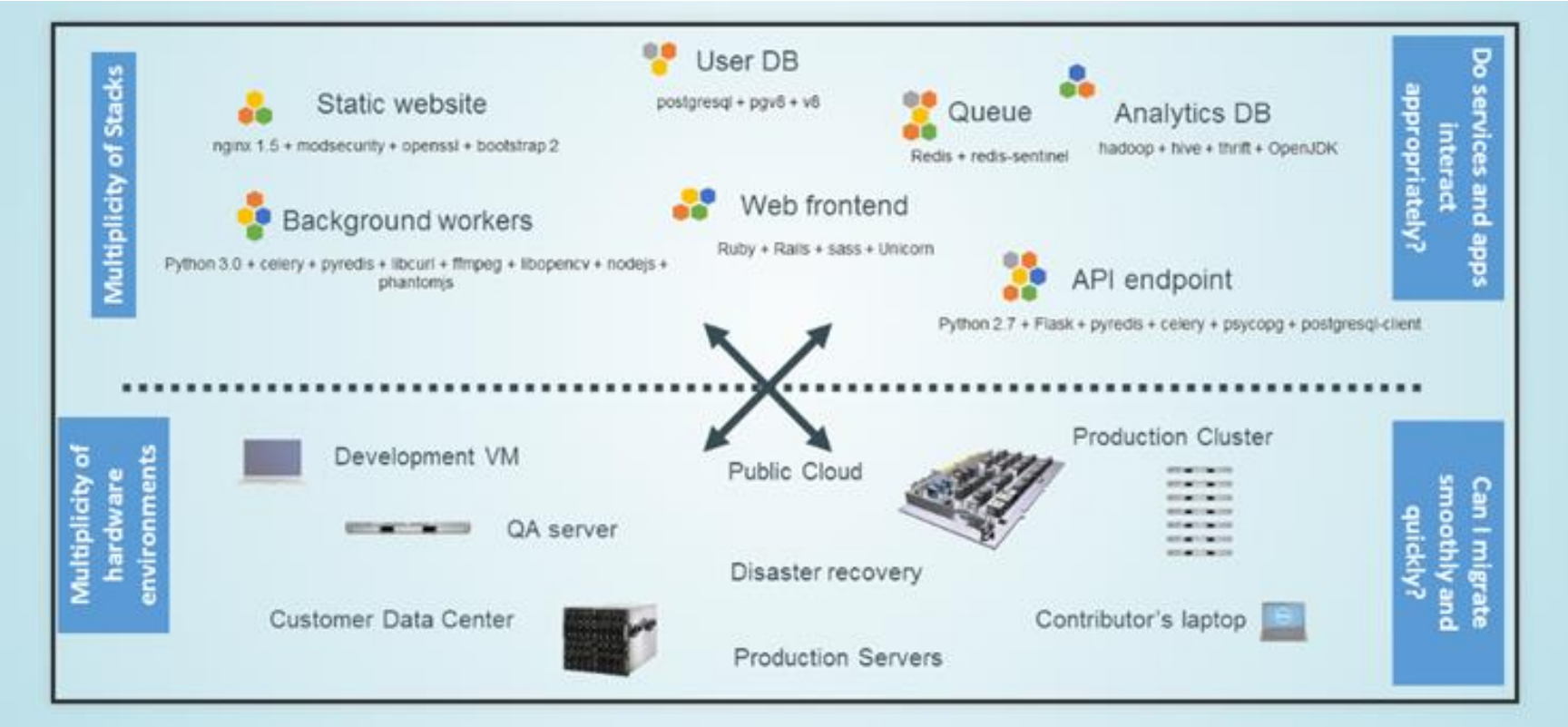
Lecture No. 5

BITS Pilani

Traditional Deployment Model



Challenges


Multiplicity of Stacks
Static website
nginx 1.5 + modsecurity + openssl + bootstrap 2
User DB
postgresql + pgv8 + v8
Queue
Redis + redis-sentinel
Analytics DB
hadoop + hive + thrift + OpenJDK
Do services and apps interact appropriately?
Background workers
Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs
Web frontend
Ruby + Rails + sass + Unicorn
API endpoint
Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client
Multiplicity of hardware environments
Development VM
QA server
Public Cloud
Production Cluster
Disaster recovery
Customer Data Center
Production Servers
Contributor's laptop
Can I migrate smoothly and quickly?

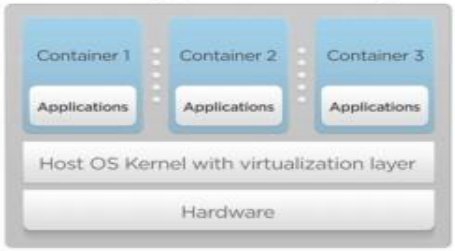
Introduction

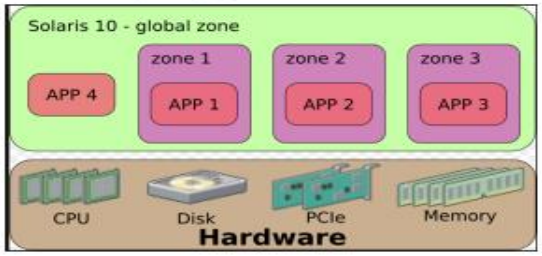
- Linux containers (LXC) are “lightweight” VMs
- Docker is a commoditized LXC technique that dramatically simplifies the use of LXC

OS Virtualization

OS Virtualization

- Emulate OS-level interface with native interface
- "Lightweight" virtual machines
  - No hypervisor, OS provides necessary support

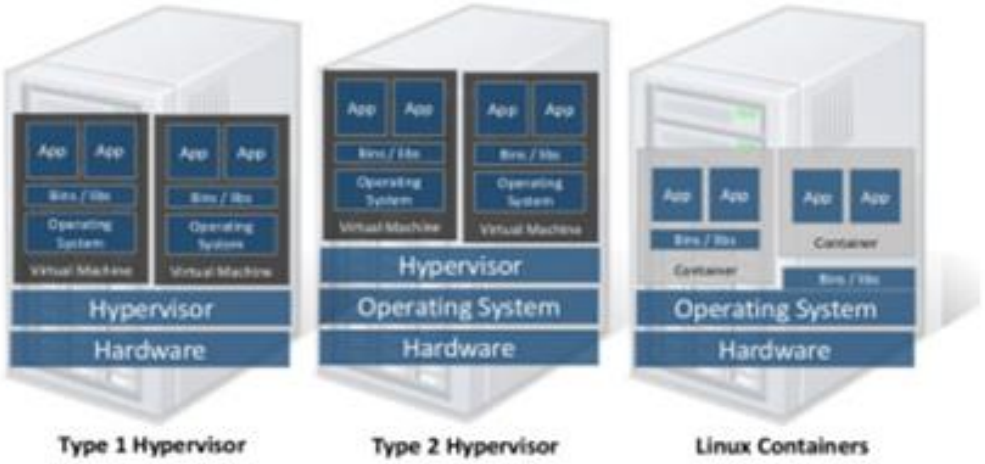




- Referred to as *containers*
  - Solaris containers, BSD jails, Linux containers

Linux Container

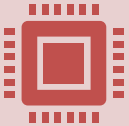
- Containers share OS kernel of the host
  - OS provides resource isolation
- Benefits
  - Fast provisioning, bare-metal like performance, lightweight



OS Mechanisms for LXC

- OS mechanisms for resource isolation and management
- namespaces: process-based resource isolation
- Cgroups: limits, prioritization, accounting, control
- chroot: apparent root directory
- Linux security module, access control
- Tools (e.g., docker) for easy management

Linux Namespaces



Linux kernel provides the “control groups” (cgroups) functionality

allows limitation and prioritization of resources (CPU, memory, block I/O, network, etc.) without the need for starting any VM

“namespace isolation” functionality

allows complete isolation of an applications' view of the operating environment,
including process trees, networking, user IDs and mounted file systems

Container Features

- Containers running in the user space
- Each container has
  - Own process space
  - Own network interface
  - Own /sbin/init (coordinates the rest of the boot process and configures the environment for the user)
  - Run stuff as root
- Share kernel with the host
- No device emulation

Isolation with cgroups

- Memory
- Cpu
- Blkio
- devices

Memory cgroup

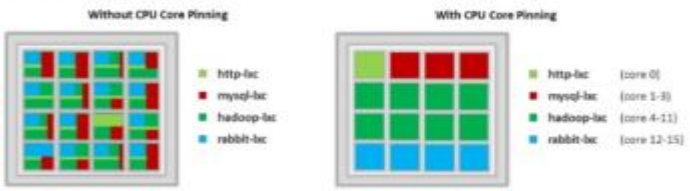
- keeps track pages used by each group:
  - file (read/write/mmap from block devices; swap)
  - anonymous (stack, heap, anonymous mmap)
  - active (recently accessed)
  - inactive (candidate for eviction)
- each page is charged to a group
- pages can be shared
- Individual (per-cgroup) limits and out-of-memory killer

CPU cgroup

- keep track of user/system CPU time
- set relative weight per group
- pin groups to specific CPU(s)
  - Can be used to reserve CPUs for some apps

Linux CGGROUPS

- Resource isolation
  - what and how much can a container use?
    - Set upper bounds (limits) on resources that can be used
    - Fair sharing of certain resources
- Examples:
  - cpu: weighted proportional share of CPU for a group
  - cpuset: cores that a group can access
  - block io: weighted proportional block IO access
  - memory: max memory limit for a group



Blkio cgroup

- keep track IOs for each block device
  - read vs write; sync vs async
- set relative weights
- set throttle (limits) for each block device
  - read vs write; bytes/sec vs operations/sec

Devices cgroup

- controls read/write/mknod permissions
- typically:
  - allow: /dev/{tty,zero,random,null}...
  - deny: everything else
  - maybe: /dev/net/tun, /dev/fuse, /dev/kvm, /dev/dri...
- fine-grained control for GPU, virtualization, etc

Almost no overhead

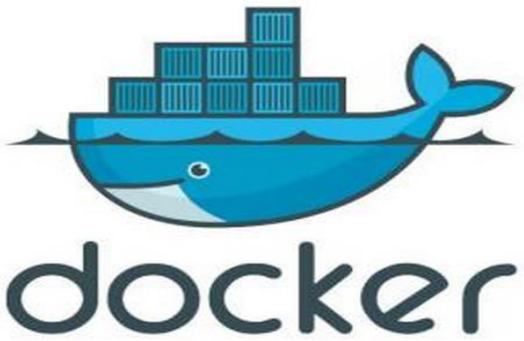
- Processes are isolated, but run straight on the host
- CPU performance = native performance
- Memory performance = a few % shaved off for (optional) accounting
- Network performance = small overhead; can be reduced to zero

Proportional Share Scheduling

  - Uses a variant of *proportional-share scheduling*
- *Share-based* scheduling:
  - Assign each process a weight w_i (a "share")
  - Allocation is in proportional to share
  - fairness: reused unused cycles to others in proportion to weight
  - Examples: fair queuing, start time fair queuing
- *Hard limits*: assign upper bounds (e.g., 30%), no reallocation
- Credit-based: allocate credits every time T, can accumulate credits, and can burst up-to credit limit
  - can a process starve other processes?

Docker

Introduction and Demo



Agenda

01 What is Docker

02 Why use Docker

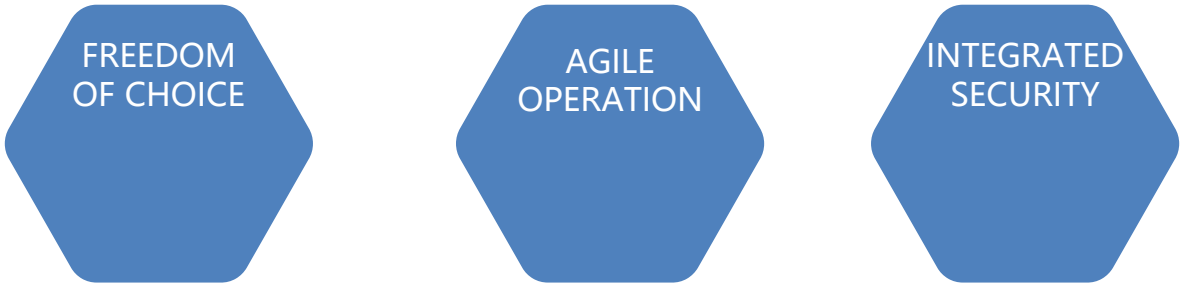
03 How to setup Docker in Linux

04 Commands & Refrences

01 What is Docker

What is Docker - Overview

Docker is the company driving the container movement and the only container platform provider to address every application across the hybrid cloud. Today's businesses are under pressure to digitally transform but are constrained by existing applications and infrastructure while rationalizing an increasingly diverse portfolio of clouds, datacenters and application architectures. Docker enables true independence between applications and infrastructure and developers and IT ops to unlock their potential and creates a model for better collaboration and innovation.



Docker history

- 2013-03: Releases as Open Source
- 2013-09: Red Hat collaboration (Fedora, RHEL, OpenShift)
- 2014-03: 34th most starred GitHub project
- 2014-05: JAX Innovation Award (most innovative open technology)

What is Docker?

Docker is a software platform that allows you to build, test, and deploy applications quickly, packaging software into standardized units called containers.

- Open Source engine to commoditize LXC
- using copy-on-write for quick provisioning
- allowing to **create and share** *images*
- **standard format** for containers
- standard, *reproducible* way to *easily* build *trusted* images (Dockerfile, Stackbrew...)

What is Docker – Basic Concepts

LXC

LXC(Linux Containers) is an operating-system-level virtualization method for running multiple isolated Linux systems(containers) on a control host using a single Linux kernel. Wikipedia

CGroups & Namespaces

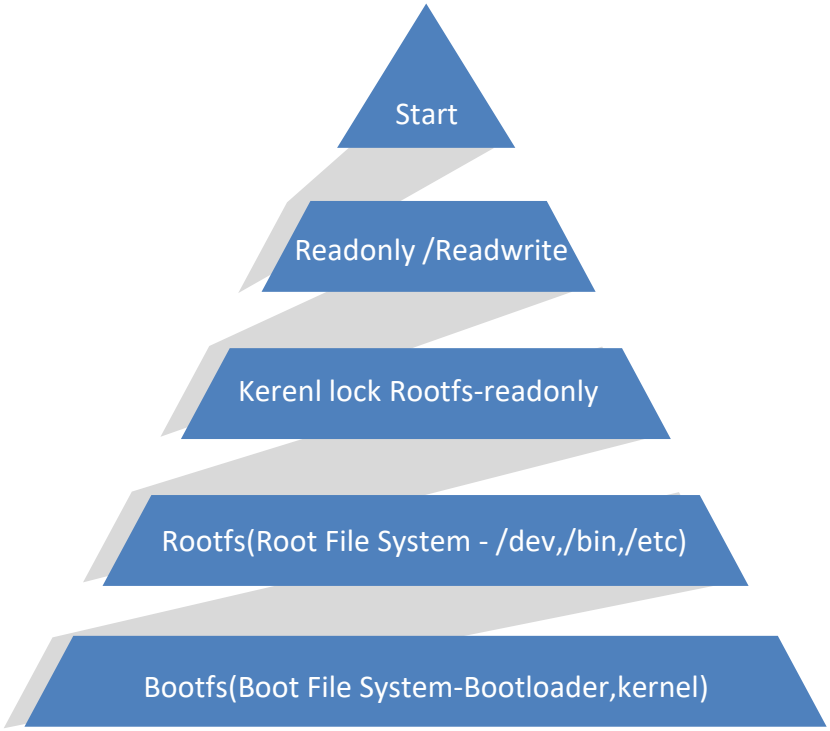
The Linux kernel provides the cgroups functionality that allows limitation and prioritization of resources (CPU, memory, block I/O, network, etc.) without the need for starting any virtual machines, and also namespace isolation functionality that allows complete isolation of an applications' view of the operating environment, including process trees,networking, user IDs and mounted file systems

AUFS

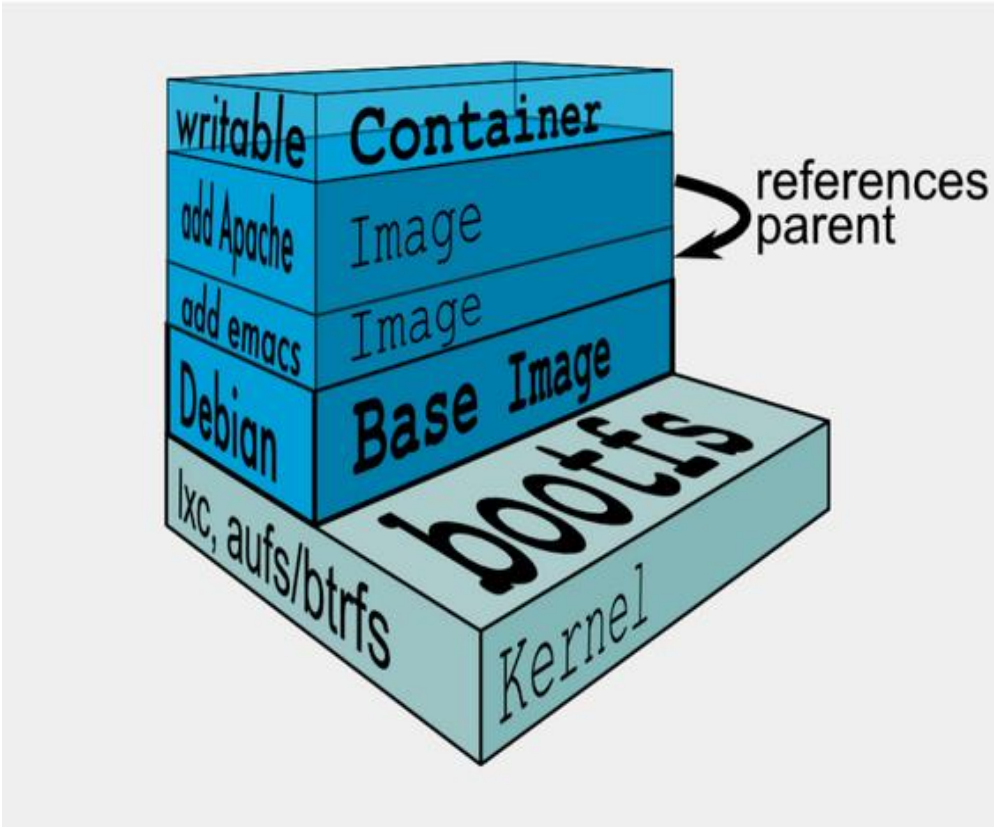
AUFS (short for advanced multi-layered unification filesystem) implements a union mount for Linux file systems

What is Docker – Basic Concepts

Linux Boot

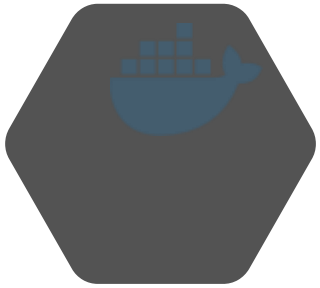


Docker Images and Uses



- Docker uses a union file system (AuFS)
  - allows containers to use host FS safely
- Essentially a copy-on-write file system
  - read-only files shared (e.g., share glibc)
  - make a copy upon write
- Allows for small efficient container images
- Docker Use Cases
  - "Run once, deploy anywhere"
  - Images can be pulled/pushed to repository
  - Containers can be a single process (useful for microservices) or a full OS

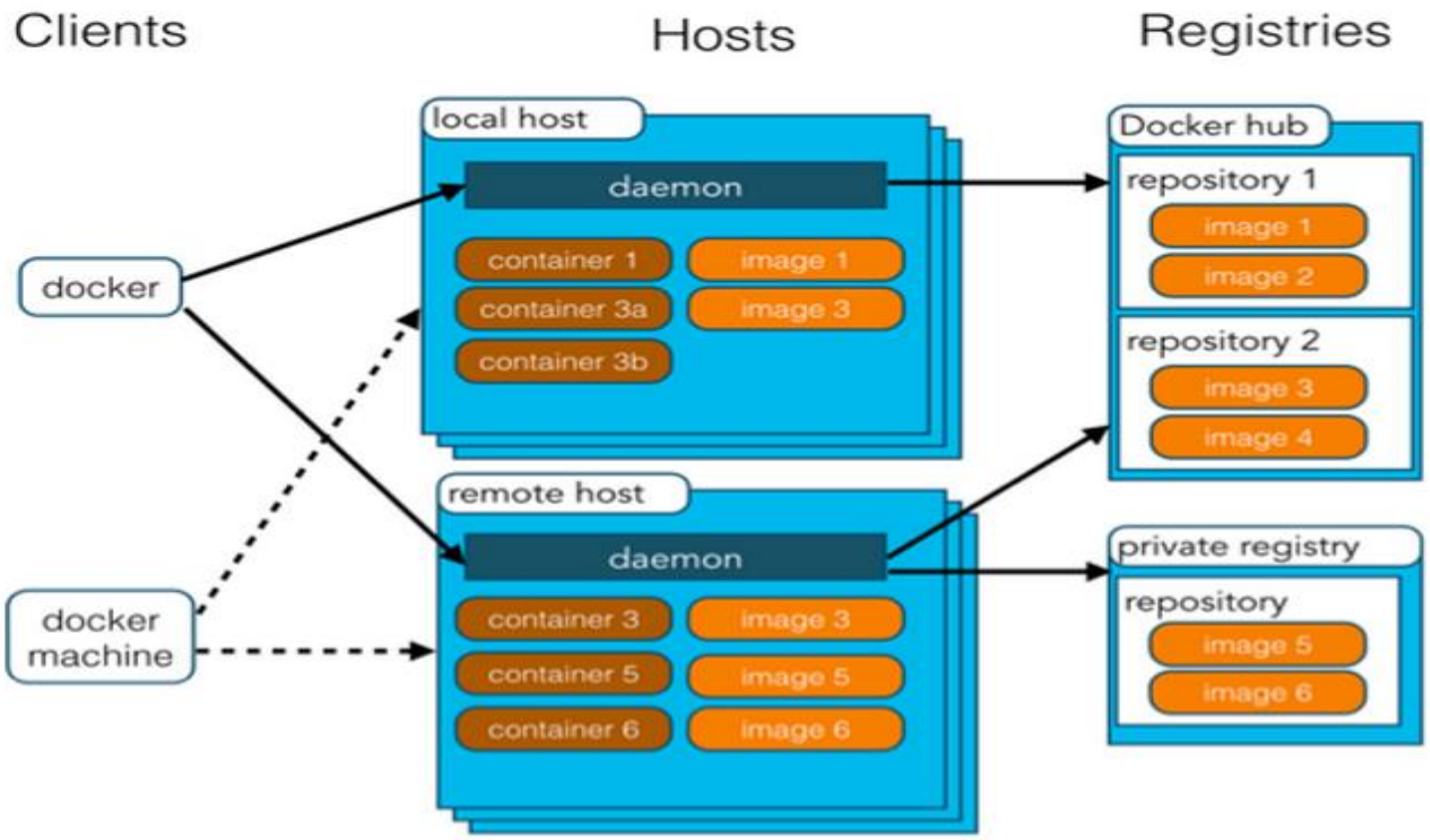
What is Docker - Contains



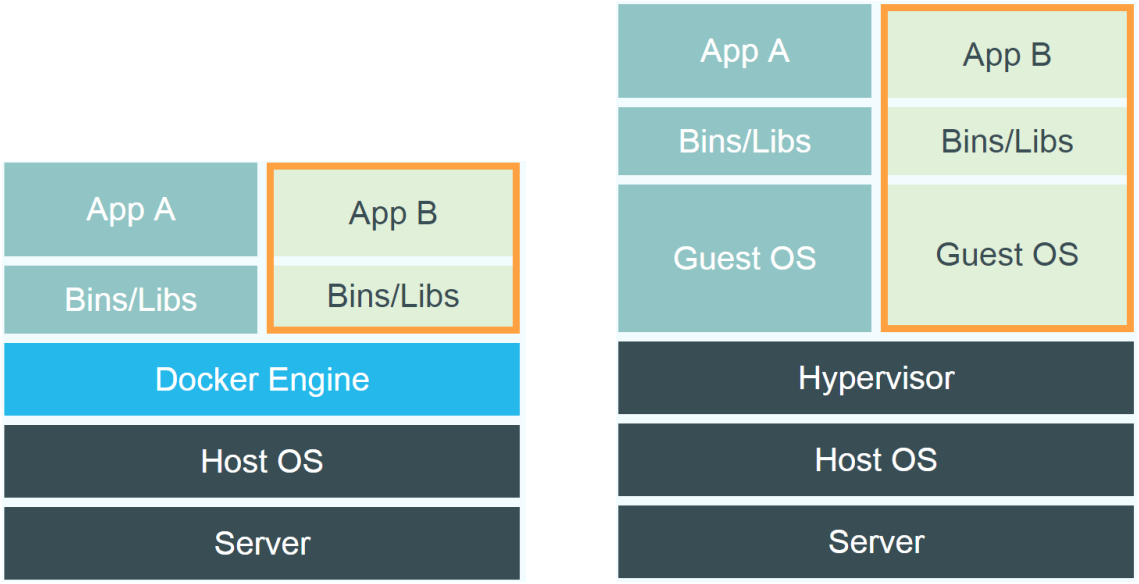
- Docker Host
- Docker Daemon, Image, Container
- Docker Client
- Docker Registry
- Docker Machine

What is Docker - Contains

THOMSON REUTERS

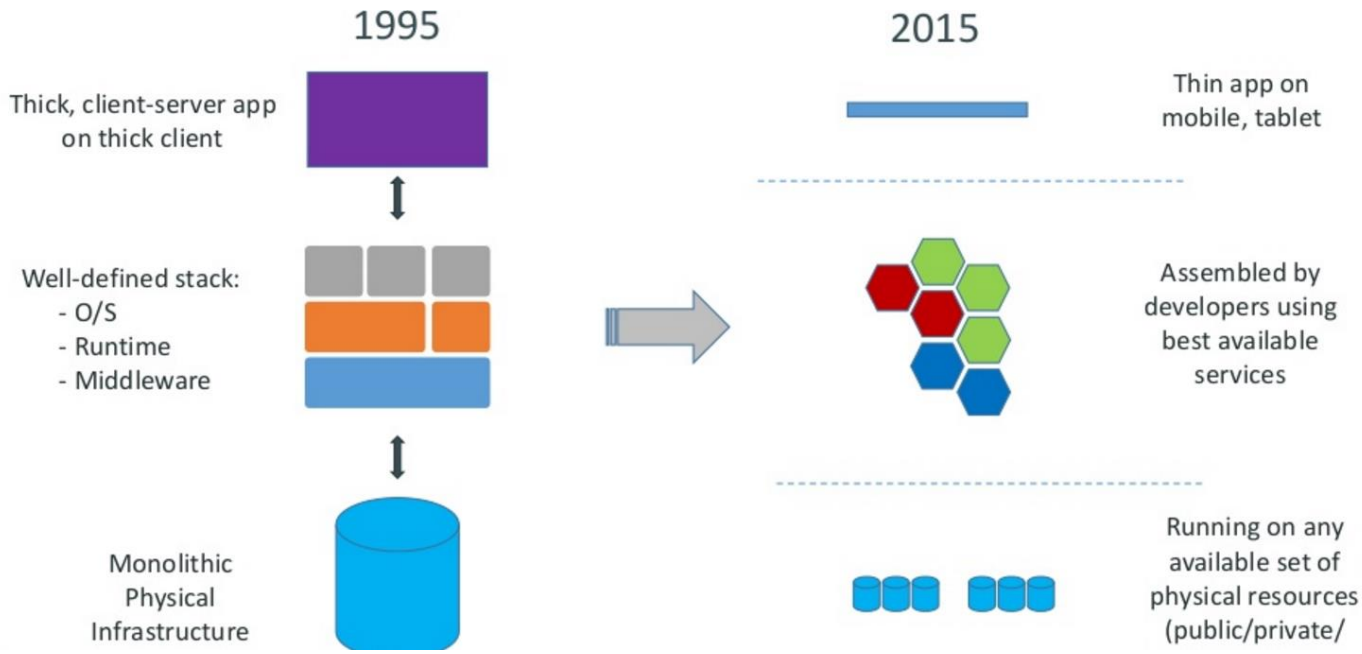


Comparison between LXC/docker and VM



02 Why use Docker

Motivation

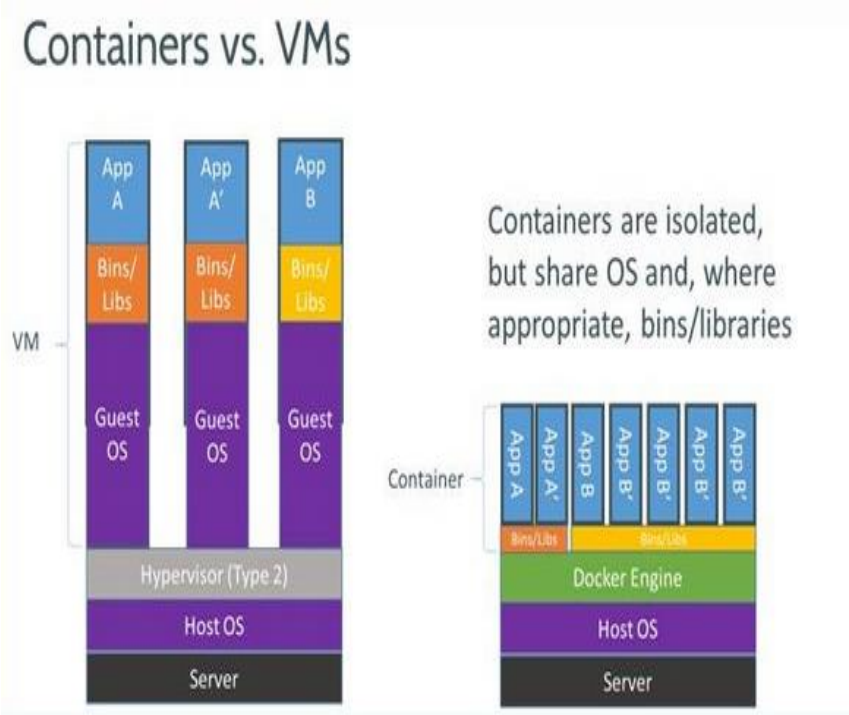

1995
2015
Thick, client-server app on thick client
Thin app on mobile, tablet
Well-defined stack:
- O/S
- Runtime
- Middleware
Assembled by developers using best available services
Monolithic Physical Infrastructure
Running on any available set of physical resources (public/private/

Docker

- Minimal learning curve
- Rebuilds are easy
- Caching system makes rebuilds faster
- Single file to define the whole environment!

Why use Docker – Compare with VM

THOMSON REUTERS



Hardware-level-virtualization 1

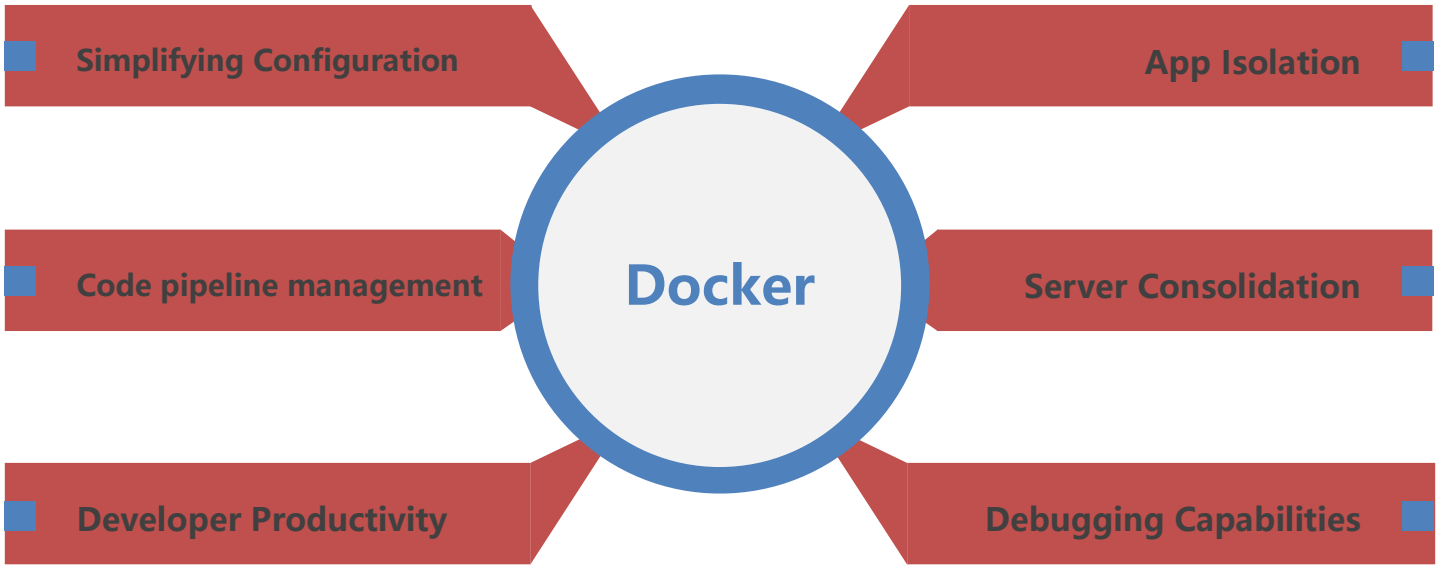
Hyper-V (OS atlease 5G)
VMWare, AWS EC2

OS-level-virtualization 2

Docker

Why use Docker – Advantages

THOMSON REUTERS



Deploy Reliability and Consistently

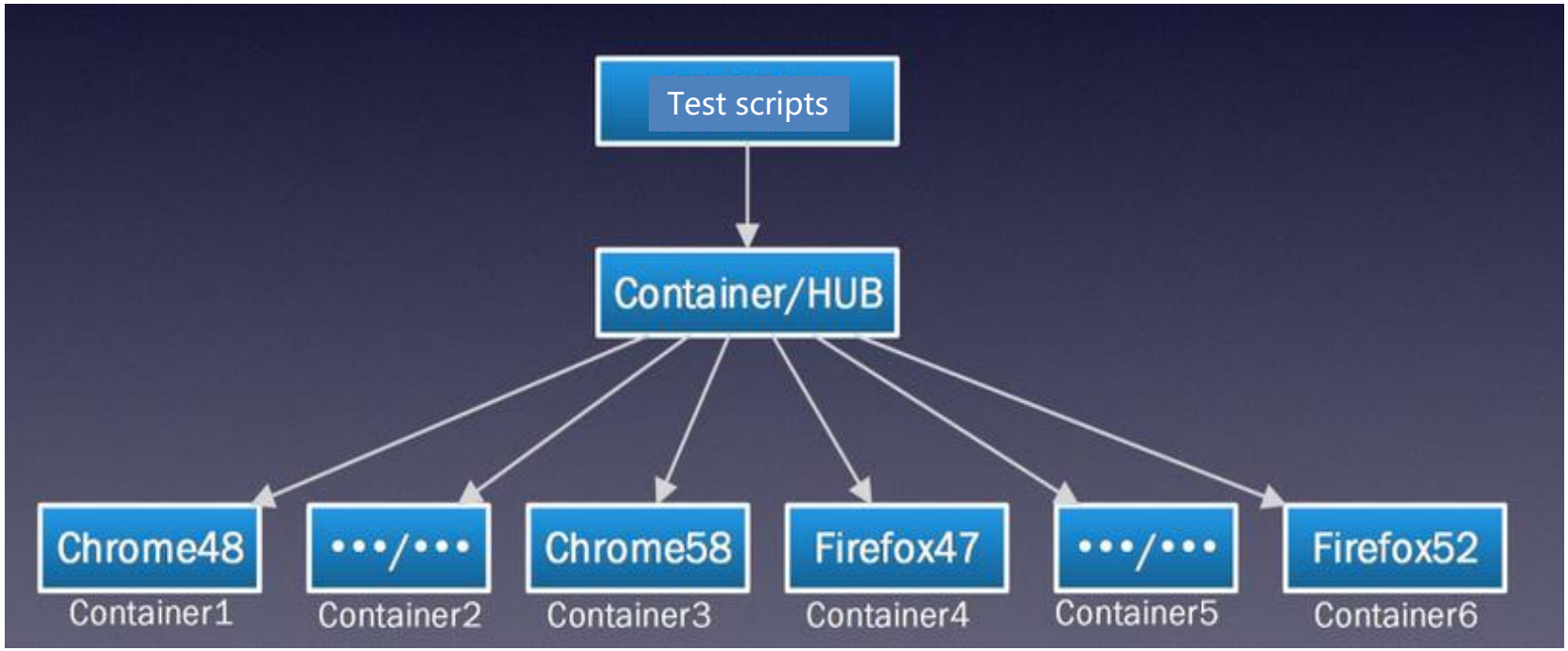
- If it works locally, it will work on the server
- *With exactly the same behavior*
- Regardless of versions
- Regardless of distros
- Regardless of dependencies

Deploy Efficiently

- Containers are lightweight
  - Typical laptop runs 10-100 containers easily
  - Typical server can run 100-1000 containers
- Containers can run at native speeds
  - Lies, damn lies, and other benchmarks:

Why use Docker – Docker in Test

THOMSON REUTERS



Docker container—developer viewpoint

Build once…run anywhere

- A clean, safe, hygienic and portable runtime environment for your app.
- No worries about missing dependencies, packages and other pain points during subsequent deployments.
- Run each app in its own isolated container, so you can run various versions of libraries and other dependencies for each app without worrying
- Automate testing, integration, packaging…anything you can script
- Reduce/eliminate concerns about compatibility on different platforms, either your own or your customers.
- Cheap, zero-penalty containers to deploy services? A VM without the overhead of a VM? Instant replay and reset of image snapshots? That's the power of Docker

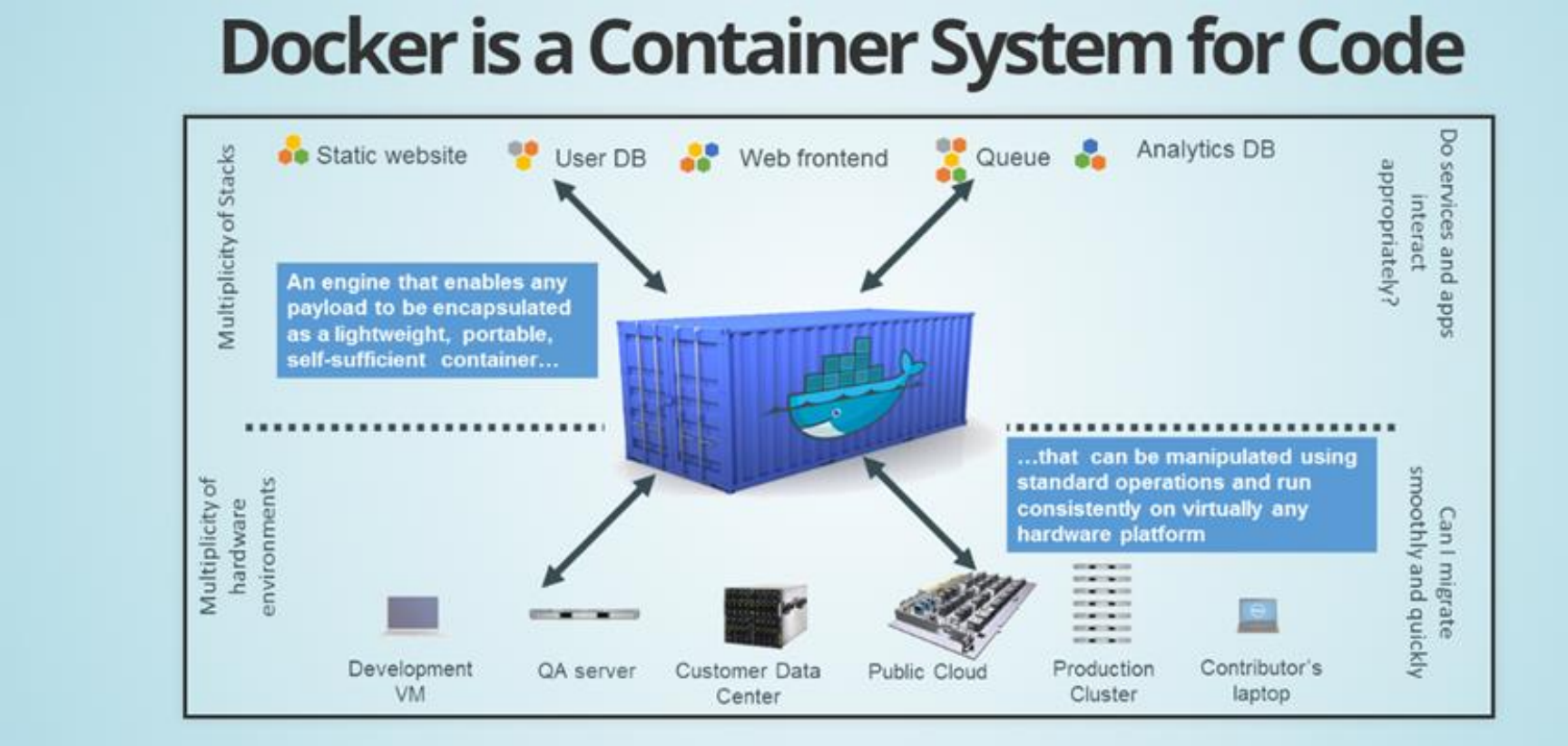
Administrative Benefits

Why Administrators Care

Configure once... run anything

- Make the entire lifecycle more efficient, consistent, and repeatable
- Increase the quality of code produced by developers.
- Eliminate inconsistencies between development, test, production, and customer environments.
- Support segregation of duties.
- Significantly improves the speed and reliability of continuous deployment and continuous integration systems.
- Because the containers are so lightweight, address significant performance, costs, deployment, and portability issues normally associated with VMs.

Docker Code Deployment



Docker Technical Details

More Technical Details

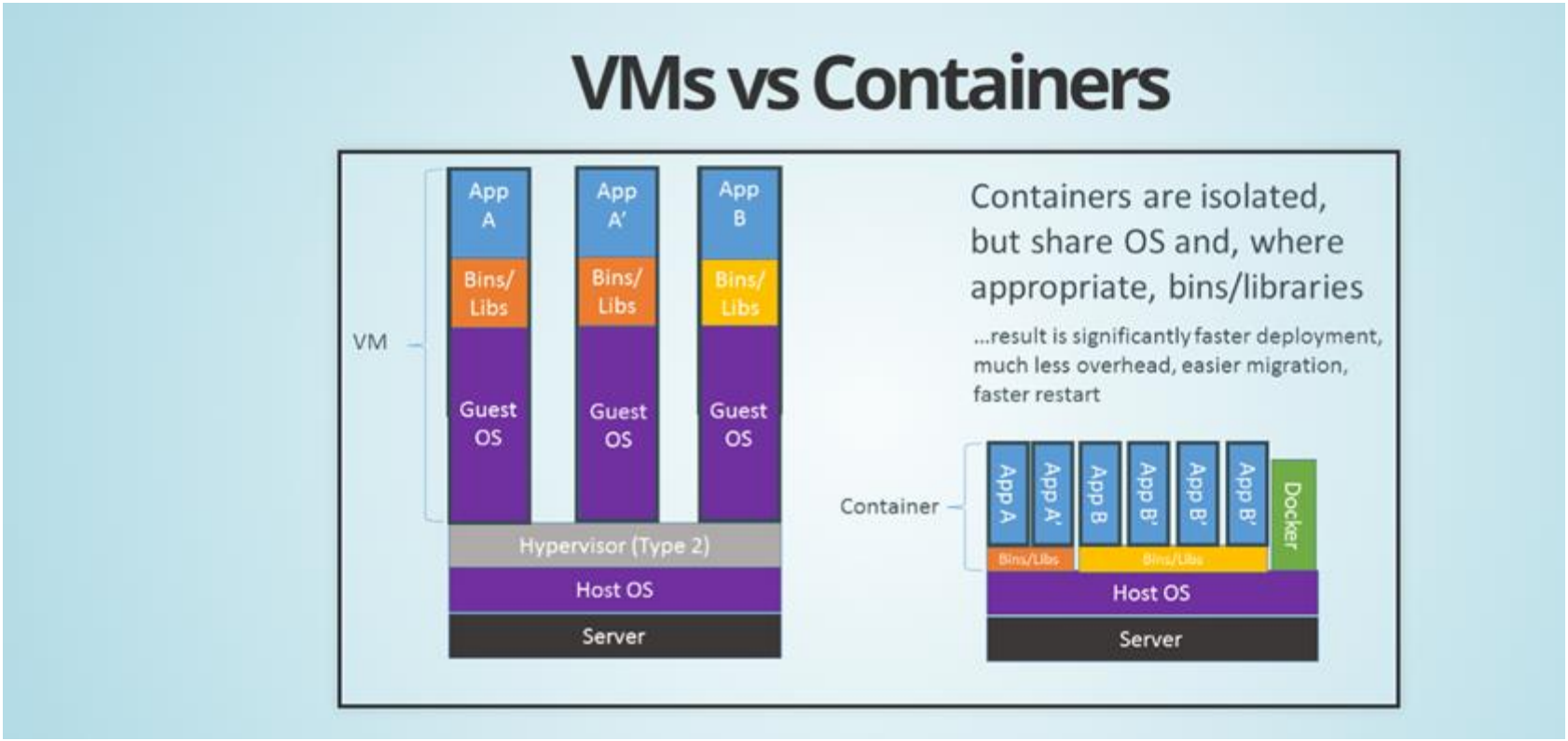
Why

- Run everywhere
  - Regardless of kernel version
  - Regardless of host distro
  - Physical or virtual, cloud or not
  - Container and host architecture must match...
- Run anything
  - If it can run on the host, it can run in the container
  - If it can on a Linux kernel, it can run

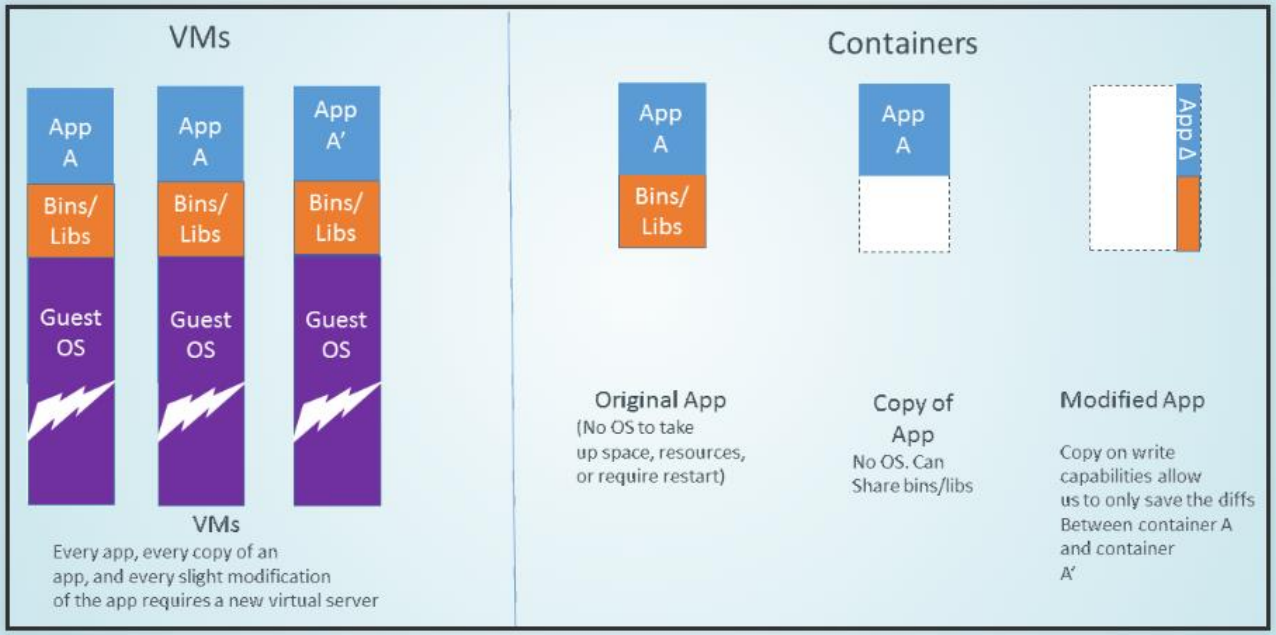
What

- High level: a lightweight VM
  - Own process space
  - Own network interface
  - Can run stuff as root
  - Can have its own /sbin/init (different from host)
  - <<machine container>>
- Low level: chroot on steroids
  - Can also *not* have its own /sbin/init
  - Container = isolated processes
  - Share kernel with host
  - <<application container>>

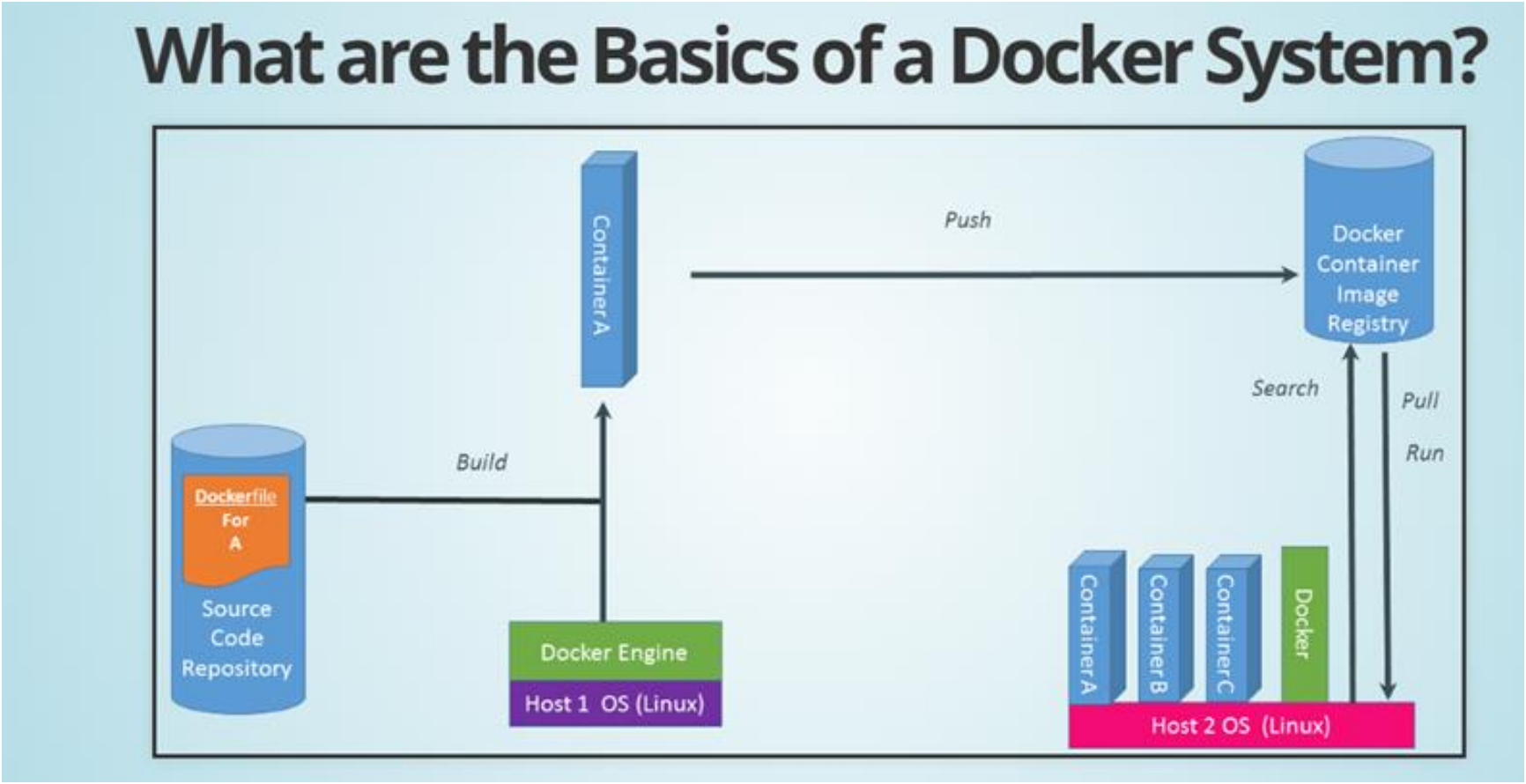
Comparison between VMS vs Containers



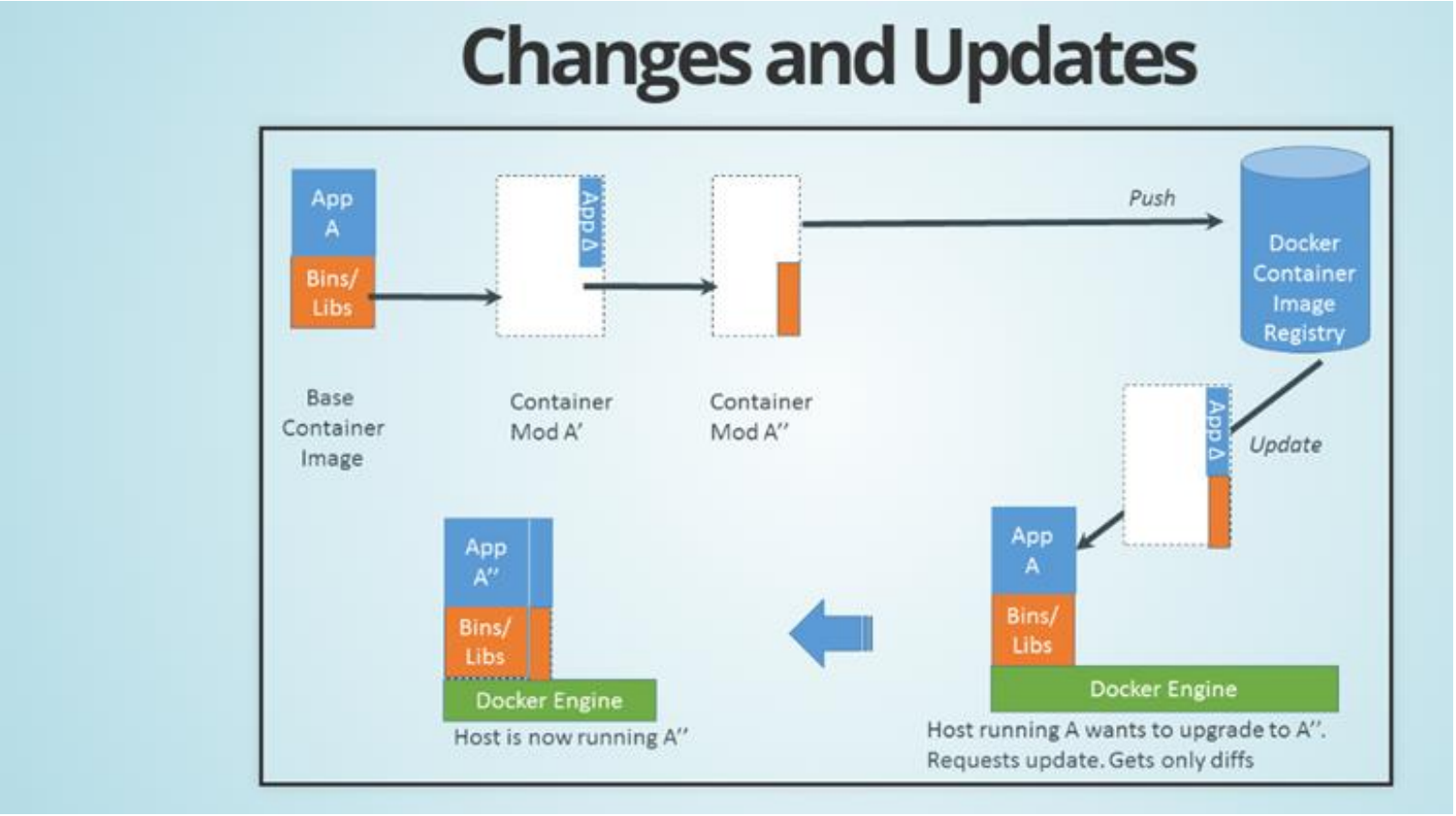
Why are Docker Containers Lightweight?



Docker Deployment



Docker Changes

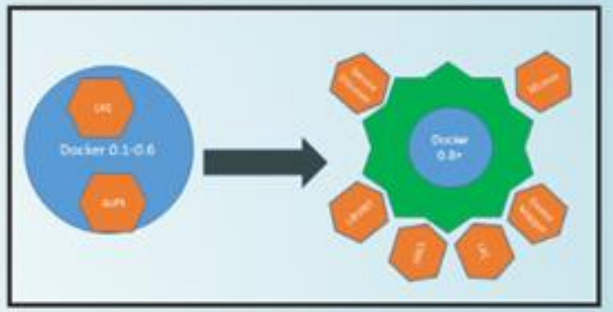


Ecosystem Supports

- Operating systems
  - Virtually any distribution with a 2.6.32+ kernel
  - Red Hat/Docker collaboration to make work across RHEL 6.4+, Fedora, and other members of the family (2.6.32 +)
  - CoreOS—Small core OS purpose built with Docker
- OpenStack
  - Docker integration into NOVA (& compatibility with Glance, Horizon, etc.) accepted for Havana release
- Private PaaS
  - OpenShift, Solum (Rackspace, OpenStack), Other TBA
- Public PaaS
  - Deis, Voxoz, Cocaine (Yandex), Baidu PaaS
- Public IaaS
  - Native support in Rackspace, Digital Ocean,+++
  - AMI (or equivalent) available for AWS & other
- DevOps Tools
  - Integrations with Chef, Puppet, Jenkins, Travis, Salt, Ansible +++
- Orchestration tools
  - Mesos, Heat, ++
  - Shipyard & others purpose built for Docker
- Applications
  - 1000's of Dockerized applications available at index.docker.io

Docker Futures

- Docker 0.7 (current release)
  - Fedora compatibility
  - Reduce kernel dependencies
  - Device mapper
  - Container linking
- Docker 0.8 (Dec)
  - Shrink and stabilize Core
  - Provide stable, pluggable API
  - RHEL compatibility
  - Nested containers
  - Beam: Introspection API based on Redis
  - Expand snapshot management features for data volumes
  - Will consider this "production ready"
- Docker 0.9 (Jan)
- Docker 1.0 (Feb)
  - Will offer support for this product



Dockerfile

It is possible to build your own images reading instructions from a `Dockerfile`

```
FROM centos:7
RUN yum install -y python-devel python-virtualenv
RUN virtualenv /opt/indico/venv
RUN pip install indico
COPY entrypoint.sh /opt/indico/entrypoint.sh
EXPOSE 8000
ENTRYPOINT /opt/indico/entrypoint.sh
```

docker-compose

Allows to run multi-container Docker applications reading instructions from a `docker-compose.yml` file

```
version: "2"
services:
  my-application:
    build: ./
    ports:
      - "8000:8000"
    environment:
      - CONFIG_FILE
  db:
    image: postgres
  redis:
    image: redis
    command: redis-server --save "" --appendonly no
    ports:
      - "6379"
```

Docker use cases

- Development Environment
- Environments for Integration Tests
- Quick evaluation of software
- Microservices
- Multi-Tenancy
- Unified execution environment (dev test prod (local, VM, cloud, ...)

03

How to setup Docker in Linux

How to setup Docker in Linux – Oracle linux 7.x

1. Touch file /etc/yum.repos.d/docker.repo

name=Docker Repository

baseurl=https://yum.dockerproject.org/repo/main/oraclelinux/7

enabled=1

gpgcheck=1

gpgkey=https://yum.dockerproject.org/gpg

2. Yum install -y docker-engine

3. Configure Firewall ：

systemctl disable firewalld

yum install -y iptables-services

systemctl enable iptables

systemctl start iptables

4. Automatic start: systemctl enable docker.service

5. Manually start: systemctl start docker.service

6. Check status: systemctl status docker.service

How to setup Docker in Linux – RHEL 7.4 build

THOMSON REUTERS

```
[root@sg1i-ddncaut03a ~]# uname -r
3.10.0-693.5.2.el7.x86_64
[root@sg1i-ddncaut03a ~]# yum install docker
Loaded plugins: product-id, search-disabled-repos
```

```
Installed:
  docker.x86_64 2:1.12.6-61.git85d7426.el7

Dependency Installed:
  atomic-registries.x86_64 1:1.19.1-5.git48c224b.el7
  audit-libs-python.x86_64 0:2.7.6-3.el7
  checkpolicy.x86_64 0:2.5-4.el7
  container-selinux.noarch 2:2.28-1.git85ce147.el7
  container-storage-setup.noarch 0:0.7.0-1.git4ca59c5.el7
  docker-client.x86_64 2:1.12.6-61.git85d7426.el7
  docker-common.x86_64 2:1.12.6-61.git85d7426.el7
  docker-rhel-push-plugin.x86_64 2:1.12.6-61.git85d7426.el7
  json-glib.x86_64 0:1.2.6-1.el7
  libcgroup.x86_64 0:0.41-13.el7
  libsemanage-python.x86_64 0:2.5-8.el7
  oci-register-machine.x86_64 1:0-3.13.gitcd1e331.el7
  oci-systemd-hook.x86_64 1:0.1.14-1.git1ba44c6.el7
  oci-umount.x86_64 2:2.0.0-1.git299e781.el7
  policycoreutils-python.x86_64 0:2.5-17.1.el7
  python-IPy.noarch 0:0.75-6.el7
  setools-libs.x86_64 0:3.3.8-1.1.el7
  skopeo-containers.x86_64 1:0.1.24-1.dev.git28d4e08.el7
  subscription-manager-plugin-container.x86_64 0:1.19.23-1.el7_4
  yajl.x86_64 0:2.0.4-4.el7

Complete!
[root@sg1i-ddncaut03a ~]#
```

How to setup Docker in Linux – RHEL 7.4 build

THOMSON REUTERS

```
[root@sg1i-ddncaut03a ~]# docker version
Client:
 Version:         1.12.6
 API version:     1.24
 Package version: docker-1.12.6-61.git85d7426.el7.x86_64
 Go version:      go1.8.3
 Git commit:      85d7426/1.12.6
 Built:           Tue Sep 26 15:30:51 2017
 OS/Arch:         linux/amd64

Server:
 Version:         1.12.6
 API version:     1.24
 Package version: docker-1.12.6-61.git85d7426.el7.x86_64
 Go version:      go1.8.3
 Git commit:      85d7426/1.12.6
 Built:           Tue Sep 26 15:30:51 2017
 OS/Arch:         linux/amd64
[root@sg1i-ddncaut03a ~]#
```

How to setup Docker in Linux – RHEL 7.4 build

THOMSON REUTERS

```
[root@sg1i-ddncaut03a ~]# docker search oracle
INDEX       NAME                                      DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
docker.io   docker.io/oraclelinux                     Official Docker builds of Oracle Linux.         448       [OK]
docker.io   docker.io/frolvlad/alpine-oraclejdk8      The smallest Docker image with OracleJDK 8...   303                  [OK]
docker.io   docker.io/sath89/oracle-12c               Oracle Standard Edition 12c Release 1 with...   291                  [OK]
docker.io   docker.io/alexeiled/docker-oracle-xe-11g  This is a working (hopefully) Oracle XE 11...   251                  [OK]
docker.io   docker.io/sath89/oracle-xe-11g            Oracle xe 11g with database files mount su...   180                  [OK]
docker.io   docker.io/jaspeen/oracle-11g              Docker image for Oracle 11g database            63                   [OK]
docker.io   docker.io/isuper/java-oracle              This repository contains all java releases...   55                   [OK]
docker.io   docker.io/wnameless/oracle-xe-11g         Dockerfile of Oracle Database Express Edit...   51                   [OK]
docker.io   docker.io/oracle/glassfish                GlassFish Java EE Application Server on Or...   42                   [OK]
docker.io   docker.io/oracle/openjdk                  Docker images containing OpenJDK Oracle Linux   37                   [OK]
docker.io   docker.io/airdock/oracle-jdk              Docker Image for Oracle Java SDK (8 and 7)...   30                   [OK]
docker.io   docker.io/ingensi/oracle-jdk              Official Oracle JDK installed on centos.        21                   [OK]
docker.io   docker.io/cogniteev/oracle-java           Oracle JDK 6, 7, 8, and 9 based on Ubuntu ...   20                   [OK]
docker.io   docker.io/n3ziniuka5/ubuntu-oracle-jdk    Ubuntu with Oracle JDK. Check tags for ver...   16                   [OK]
docker.io   docker.io/oracle/nosql                    Oracle NoSQL on a Docker Image with Oracle...   16                   [OK]
docker.io   docker.io/sgrio/java-oracle               Docker images of Java 7/8/9 provided by Or...   11                   [OK]
docker.io   docker.io/andreptb/oracle-java            Debian Jessie based image with Oracle JDK ...   7                    [OK]
docker.io   docker.io/openweb/oracle-tomcat           A fork off of Official tomcat image with O...   7                    [OK]
docker.io   docker.io/flurdy/oracle-java7             Base image containing Oracle's Java 7 JDK       5                    [OK]
docker.io   docker.io/martinseeler/oracle-server-jre  Oracle's Java 8 as 61 MB Docker container.      4                    [OK]
docker.io   docker.io/davidcaste/debian-oracle-java   Oracle Java 8 (and 7) over Debian Jessie        3                    [OK]
docker.io   docker.io/teradatalabs/centos6-java8-oracle  Docker image of CentOS 6 with Oracle JDK 8...  3
docker.io   docker.io/spansari/nodejs-oracledb        nodejs with oracledb installed globally on...   2
docker.io   docker.io/publicisworldwide/oracle-core   This is the core image based on Oracle Lin...   1                    [OK]
docker.io   docker.io/softwareplant/oracle            oracle db                                       0                    [OK]
[root@sg1i-ddncaut03a ~]#
```

```
[root@sg1i-ddncaut03a ~]# docker pull docker.io/oraclelinux
Using default tag: latest
Trying to pull repository docker.io/library/oraclelinux ...
latest: Pulling from docker.io/library/oraclelinux
1b19d6599a70: Downloading [==============================================>  ] 82.89 MB/86.06 MB
```

```
[root@sg1i-ddncaut03a ~]# docker pull docker.io/oraclelinux
Using default tag: latest
Trying to pull repository docker.io/library/oraclelinux ...
latest: Pulling from docker.io/library/oraclelinux
1b19d6599a70: Pull complete
Digest: sha256:6067eb9ac8edc2042508e2adfd00b9fb1cc1d38e708d0f5f98058a8740ba0661
[root@sg1i-ddncaut03a ~]#
```

How to setup Docker in Linux – RHEL 7.4 build

THOMSON REUTERS

```
[root@sg1i-ddncaut03a ~]# docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
docker.io/oraclelinux   latest              af8cf7fc5b7e        2 weeks ago         233.7 MB
[root@sg1i-ddncaut03a ~]#
```

```
[root@sg1i-ddncaut03a ~]# docker inspect af8cf7fc5b7e
[
    {
        "Id": "sha256:af8cf7fc5b7e9e4dcee6db022f23118d98ff54bfea1458bfb2d2ad7fad61770f",
        "RepoTags": [
            "docker.io/oraclelinux:latest"
        ],
        "RepoDigests": [
            "docker.io/oraclelinux@sha256:6067eb9ac8edc2042508e2adfd00b9fb1cc1d38e708d0f5f98058a8740ba0661"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2018-04-18T18:40:12.935217168Z",
        "Container": "4b74c7f189be3a3f2b318729e06b963df935520e5544520c2b26c17fddf63f55",
        "ContainerConfig": {
            "Hostname": "4b74c7f189be",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/sh",
                "-c",
                "#(nop) ",
                "CMD [\"/bin/bash\"]"
```

How to setup Docker in Linux – RHEL 7.4 build

THOMSON REUTERS

```
[root@sg1i-ddncaut03a ~]# docker run --help

Usage:  docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

Run a command in a new container
```

```
[root@sg1i-ddncaut03a ~]# docker run -i -t docker.io/oraclelinux /bin/bash
[root@9524b56297f5 /]# echo "hello docker"
hello docker
[root@9524b56297f5 /]# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
```

```
[root@sg1i-ddncaut03a ~]# docker ps
CONTAINER ID        IMAGE                  COMMAND             CREATED             STATUS              PORTS               NAMES
9524b56297f5        docker.io/oraclelinux  "/bin/bash"         About a minute ago  Up About a minute                       zen_chandrasekhar
[root@sg1i-ddncaut03a ~]# docker ps -a
CONTAINER ID        IMAGE                  COMMAND             CREATED             STATUS              PORTS               NAMES
9524b56297f5        docker.io/oraclelinux  "/bin/bash"         About a minute ago  Up About a minute                       zen_chandrasekhar
[root@sg1i-ddncaut03a ~]#
```

Commands & Reference – Docker Commands

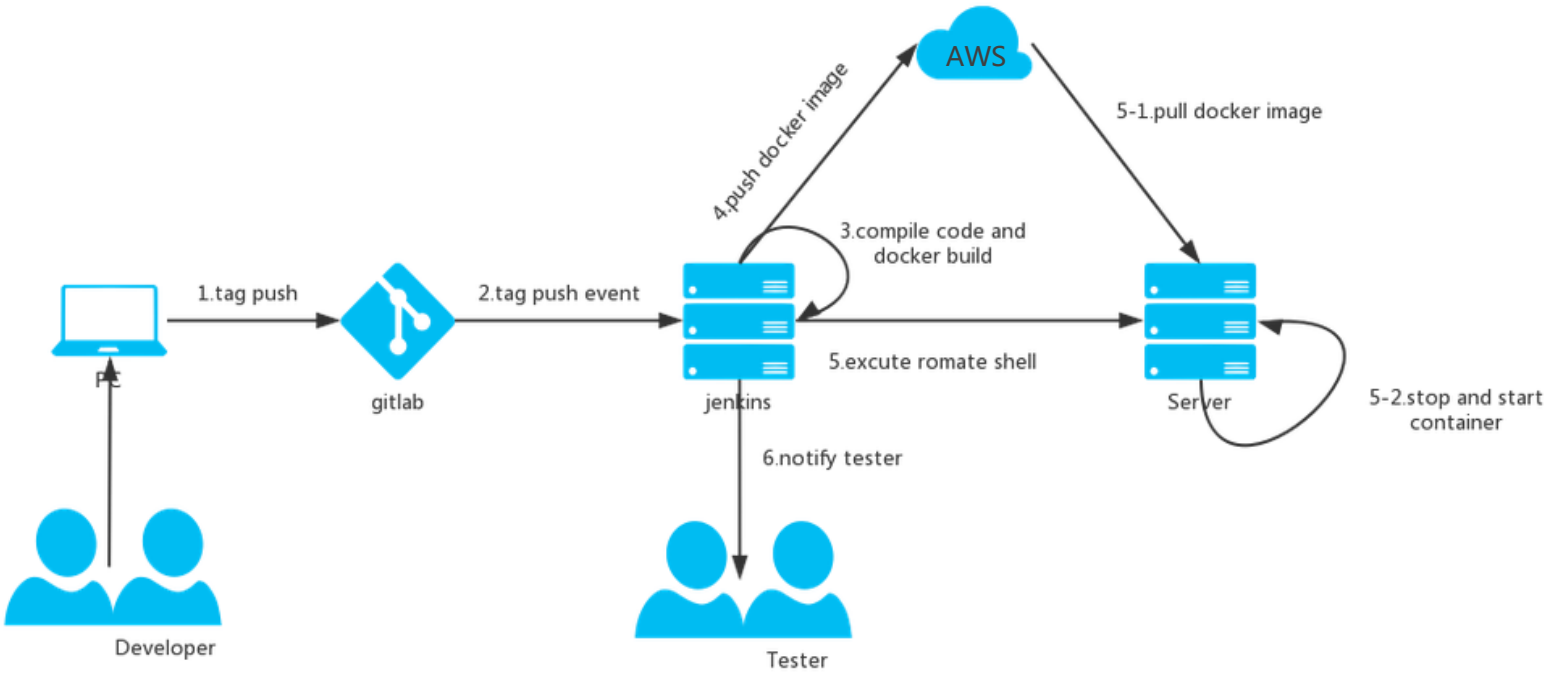
THOMSON REUTERS

docker run [OPTIONS] IMAGE [COMMAND] [ARG...]	runoob@runoob:~$ docker run -it nginx:latest /bin/bash
docker start [OPTIONS] CONTAINER [CONTAINER...]	docker start myrunoob
docker stop [OPTIONS] CONTAINER [CONTAINER...]	docker stop myrunoob
docker restart [OPTIONS] CONTAINER [CONTAINER...]	docker restart myrunoob
docker kill [OPTIONS] CONTAINER [CONTAINER...]	runoob@runoob:~$ docker kill -s KILL mynginx
docker rm [OPTIONS] CONTAINER [CONTAINER...]	docker rm -f db01、db02
docker create [OPTIONS] IMAGE [COMMAND] [ARG...]	runoob@runoob:~$ docker create --name myrunoob nginx:latest
docker ps [OPTIONS]	runoob@runoob:~$ docker ps -a -q
docker inspect [OPTIONS] NAME|ID [NAME|ID...]	runoob@runoob:~$ docker inspect mysql:5.6
docker top [OPTIONS] CONTAINER [ps OPTIONS]	runoob@runoob:~/mysql$ docker top mymysql
docker logs [OPTIONS] CONTAINER	runoob@runoob:~$ docker logs -f mynginx
docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]	runoob@runoob:~$ docker commit -a "runoob.com" -m "my apache" a404c6c174a2 mymysql:v1
docker diff [OPTIONS] CONTAINER	runoob@runoob:~$ docker diff mymysql
docker login [OPTIONS] [SERVER]	docker login -u username -p password
docker pull [OPTIONS] NAME[:TAG|@DIGEST]	docker pull java
docker push [OPTIONS] NAME[:TAG]	docker push myapache:v1
docker search [OPTIONS] TERM	runoob@runoob:~$ docker search -s 10 java
docker images [OPTIONS] [REPOSITORY[:TAG]]	runoob@runoob:~$ docker images
docker build [OPTIONS] PATH | URL | -	docker build -t runoob/ubuntu:v1 .

Commands & Reference – Dockerfile Commands

FROM	FROM <image>:<tag>
MAINTAINER	MAINTAINER <name>
RUN	RUN <command> (the command is run in a shell - `/bin/sh -c`)
CMD	CMD ["executable","param1","param2"] (like an exec, this is the preferred form)
ENTRYPOINT	ENTRYPOINT ["executable", "param1", "param2"] (like an exec, the preferred form)
USER	ENTRYPOINT ["memcached", "-u", "daemon"]
EXPOSE	EXPOSE <port> [<port>...]
ENV	ENV <key> <value>
ADD	ADD <src> <dest>
VOLUME	VOLUME ["<mountpoint>"]

Why use Docker – Docker in Devopes



Commands & References – References

http://www.docker.com

https://docs.docker.com/linux/

https://docs.docker.com/engine/userguide/

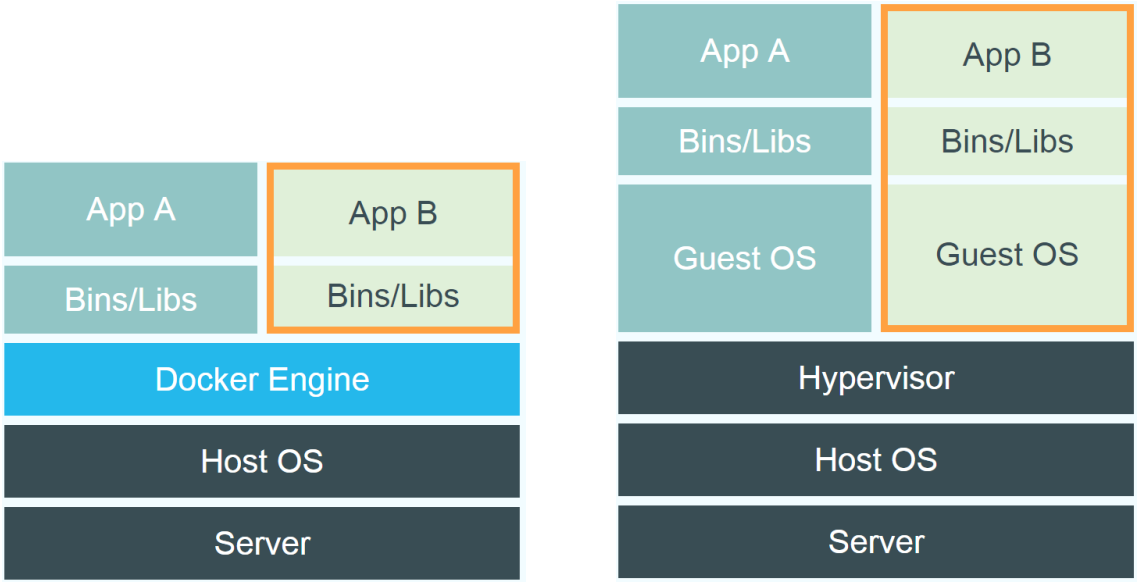
https://www.docker.com/open-source

https://hub.docker.com/

https://www.docker-cn.com/

https://thehub.thomsonreuters.com/docs/DOC-2572951

Comparison between LXC/docker and VM



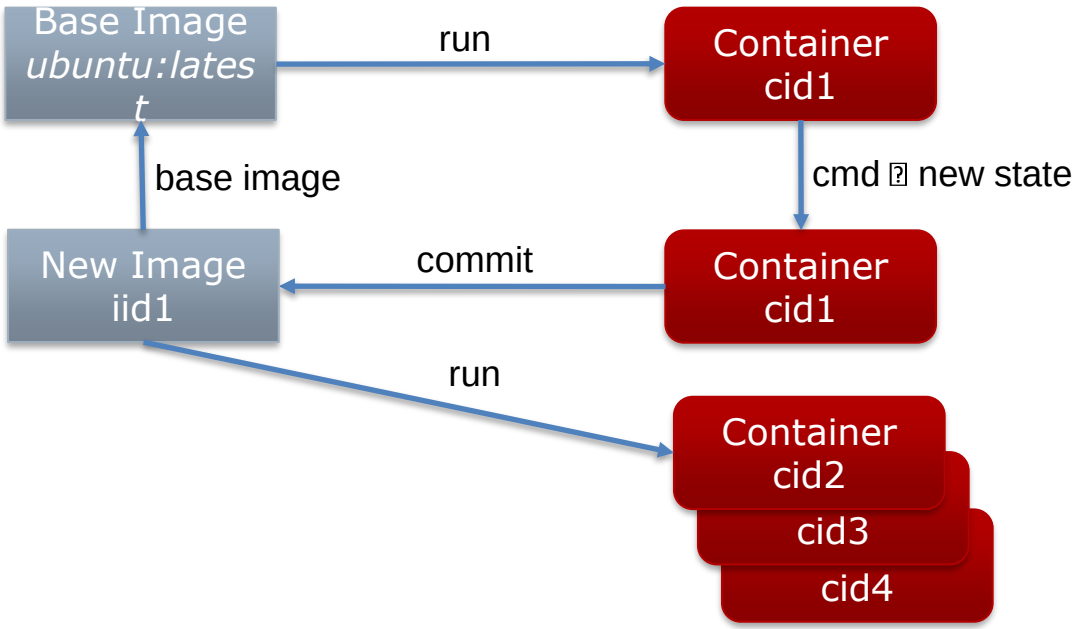
Docker Engine

- the Docker engine runs in the background
  - manages containers, images, and builds
  - HTTP API (over UNIX or TCP socket)
  - embedded CLI talking to the API

Building docker image

- With run/commit commands
  1) docker run ubuntu bash
  2) apt-get install this and that
  3) docker commit <containerid> <imagename>
  4) docker run <imagename> bash
  5) git clone git://.../mycode
  6) pip install -r requirements.txt
  7) docker commit <containerid> <imagename>
  8) repeat steps 4-7 as necessary
  9) docker tag <imagename> <user/image>
  10) docker push <user/image>

Docker


Base Image
ubuntu:latest
run
Container
cid1
base image
cmd new state
New Image
iid1
commit
Container
cid1
run
Container
cid2
cid3
cid4

Authoring image with a dockerfile

- A sample dockerfile

FROM ubuntu

```
RUN apt-get -y update
RUN apt-get install -y g++
RUN apt-get install -y erlang-dev erlang-manpages erlang-base-
hipe ...
RUN apt-get install -y libmozjs185-dev libicu-dev libtool ...
RUN apt-get install -y make wget
RUN wget http://.../apache-couchdb-1.3.1.tar.gz | tar -C /tmp -
zxf-
RUN cd /tmp/apache-couchdb-* && ./configure && make install
RUN printf "[httpd]\nport = 8101\nbind_address = 0.0.0.0" >
/usr/local/etc/couchdb/local.d/docker.ini
```

EXPOSE 8101

CMD ["/usr/local/bin/couchdb"]

Run the command to build:
docker build -t your_account/couchdb .

Docker Hub

- Public repository of Docker images
  - https://hub.docker.com/
  - docker search [term]
- Automated: Has been automatically built from Dockerfile
  - Source for build is available on GitHub

Dev-> test->production

- code in local environment
  (« dockerized » or not)
- each push to the git repo triggers a hook
- the hook tells a build server to clone the code and run « docker build » (using the Dockerfile)
- the containers are tested (nosetests, Jenkins...), and if the tests pass, pushed to the registry
- production servers pull the containers and run them
- for network services, load balancers are updated

Docker has a repository like github

you can push and pull container images to/from the Docker registry

which is something like a “GitHub” for Docker container images.

Docker has a repository like github

you can push and pull container images to/from the Docker registry

which is something like a “GitHub” for Docker container images.

- Multi-tenancy is an architecture in which a single instance of a software application serves multiple customers. Each customer is called a tenant. Tenants may be given the ability to customize some parts of the application, such as color of the user interface (UI) or business rules, but they cannot customize the application's code.
- A software-as-a-service (SaaS) provider, for example, can run one instance of its application on one instance of a database and provide web access to multiple customers. In such a scenario, each tenant's data is isolated and remains invisible to other tenants.

- Multi-tenancy is an architectural pattern
- A single instance of the software is run on the service provider's infrastructure
- Multiple tenants access the same instance.
- In contrast to the multi-user model, multi-tenancy requires customizing the single instance according to the multi-faceted requirements of many tenants.

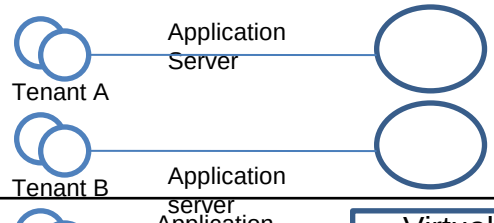
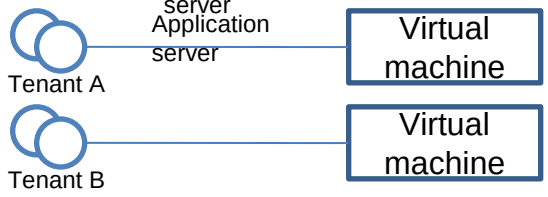
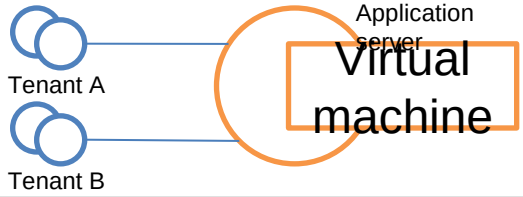
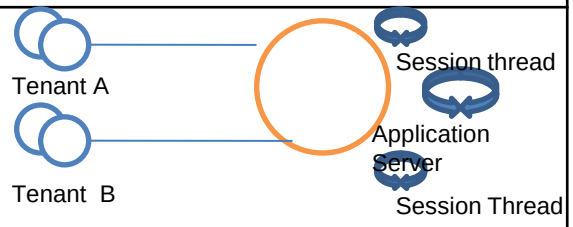
Multitenancy – key aspects

A Multi-tenants application lets customers (tenants) share the same hardware resources, by offering them one shared application and database instance ,while allowing them to configure the application to fit there needs as if it runs on dedicated environment.

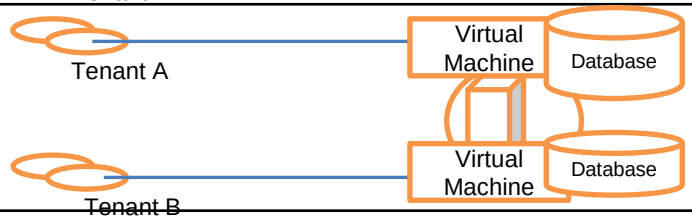
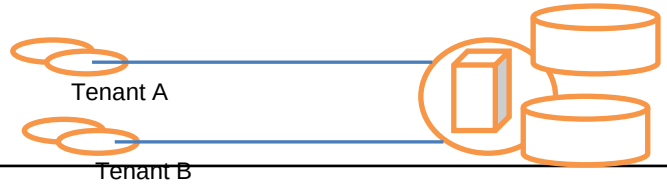
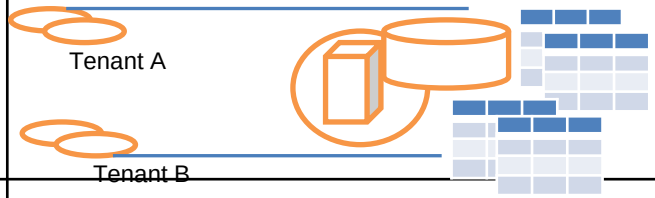
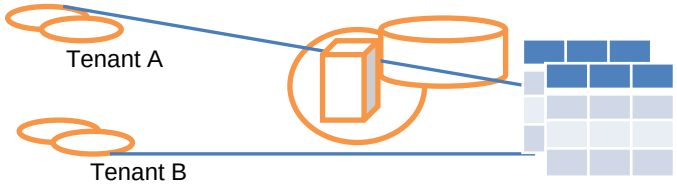
These definition focus on what we believe to be the key aspects of multi tenancy:

1. The ability of the application to share hardware resources.
2. The offering of a high degree of configurability of the software.
3. The architectural approach in which the tenants make use of a single application and database instance.

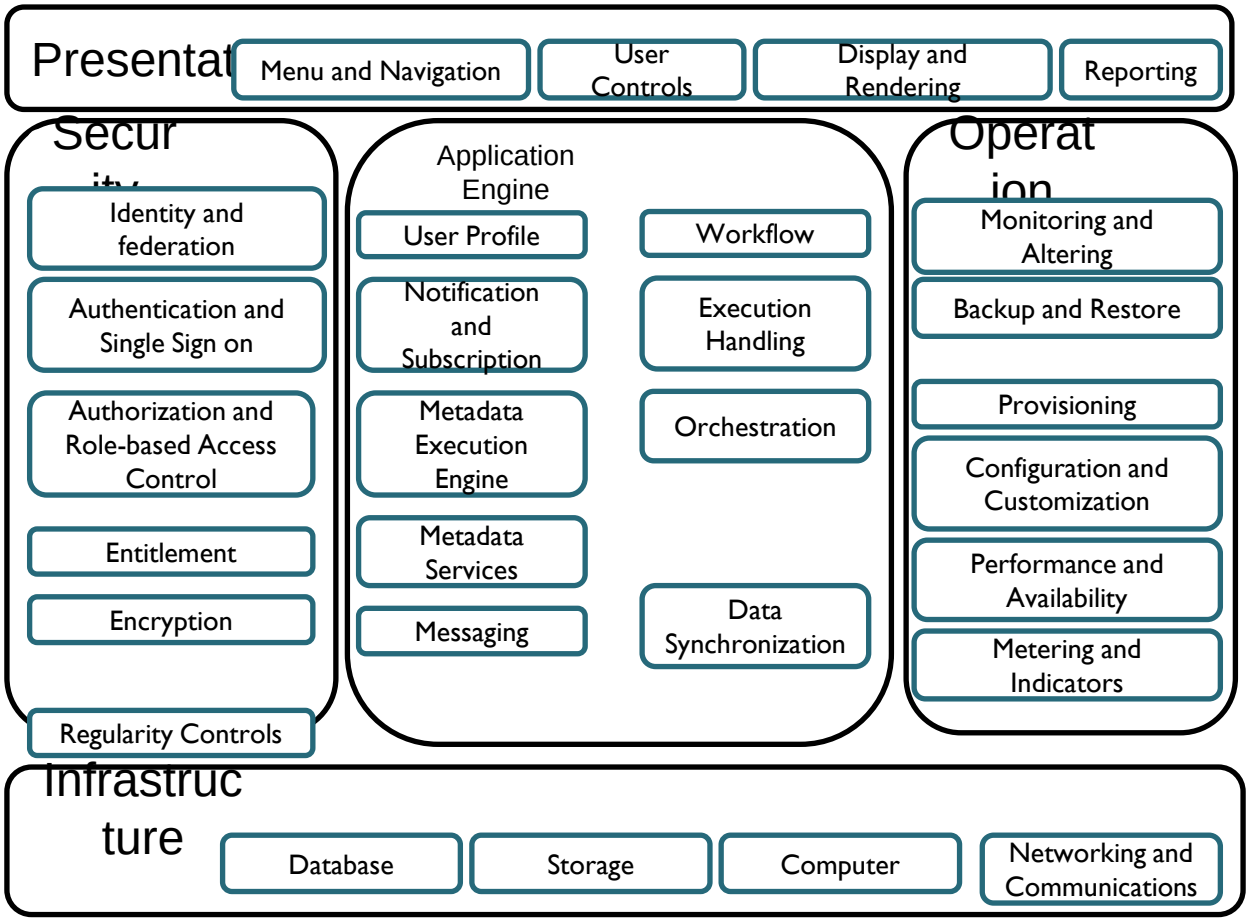
Multi-tenants Deployment Modes for Application Server

Fully isolated Application server Each tenant accesses an application server running on a dedicated servers.	
Virtualized Application Server Each tenant accesses a dedicated application running on a separate virtual machine.	
Shared Virtual Server Each tenant accesses a dedicated application server running on a shared virtual machine.	
Shared Application Server The tenant shared the application server and access application resources through separate session or threads.	

Multi-tenants Deployment Modes in Data Centers

Fully isolated data center **The tenants do not share any data center resources**	
Virtualized servers **The tenants share the same host but access different databases running on separate virtual machines**	
Shared Server **The tenants share the same server (Hostname or IP) but access different databases**	
Shared Database **The tenants share the same server and database (shared or different ports) but access different schema(tables)**	
Shared Schema **The tenants share the same server, database and schema (tables). The irrespective data is segregated by key and rows.**	

Conceptual framework of Software as a Service



Docker needs containers