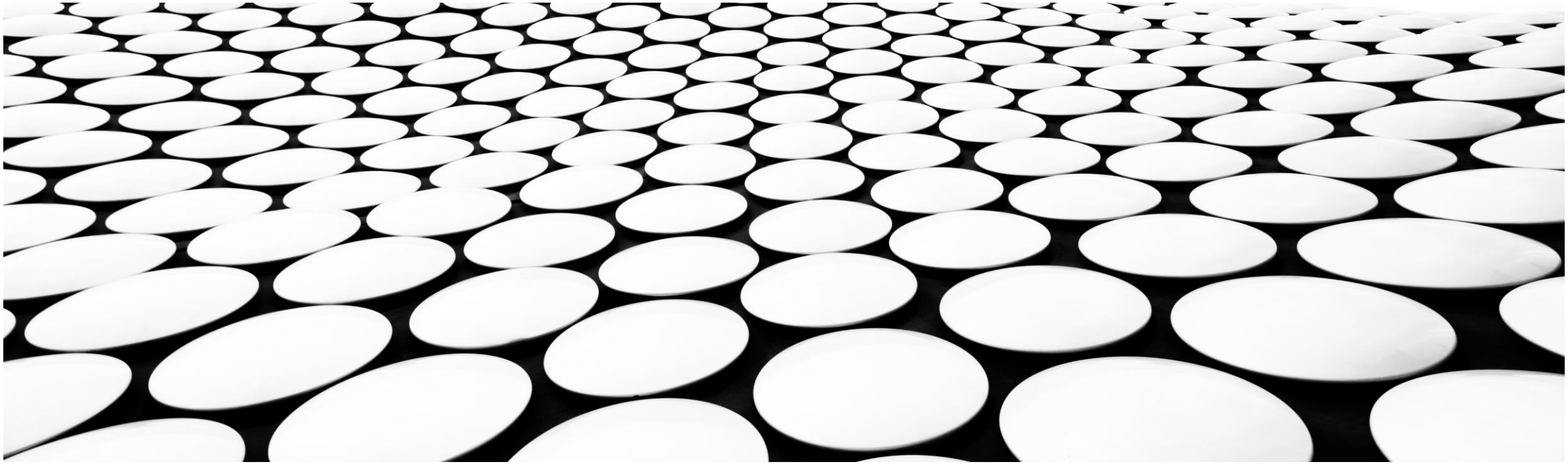
CONTAINER PIPELINES WITH ORACLE CLOUD

DJ (DHANANJAYAN)

29TH JUNE 2020



BREAK TIME

- 11.00-11:15 AM
- 1.00 – 2.00 PM
- 3.30 – 3.45 PM

INTRO

- NAME
- ROLE
- EXPERIENCE – Container and Docker Expertise ?
- Objective ?
- LINUX – Linux Architecture ?
- Cloud Computing ?

AGENDA

- MSA – Context and Architecture
- DEVOPS /Agenda , Containers
- Containerization of Solutions – USE CASES
- Docker Architecture – Implement – Best practices
- OCI (Repository)
- Orchestration – Ha of Services (Kubernetes)
- Implementation – USE CASE
- OKE (OCI)
- Automate Code – Deployment (Automation of Devops)
- OCP (Wercker)
- DEPLOY – OCI Interface

AGENDA FOR THE DAY

- USE CASE – DESIGN PATTERNS (MSA) – DEVOPS
- DOCKER – ARCHITECTURE
- OCI
- CONTAINERS AND DOCKER IMPLEMENTATION USE CASES

- ARCHITECTURE PATTERNS -0.5 DAY
- CONTAINERS AND DOCKERS – 1.5 DAYS
- OKE AND KUBERNETES – 1.5 DAYS
- CONTAINER PIPELINES – 1.5 DAYS

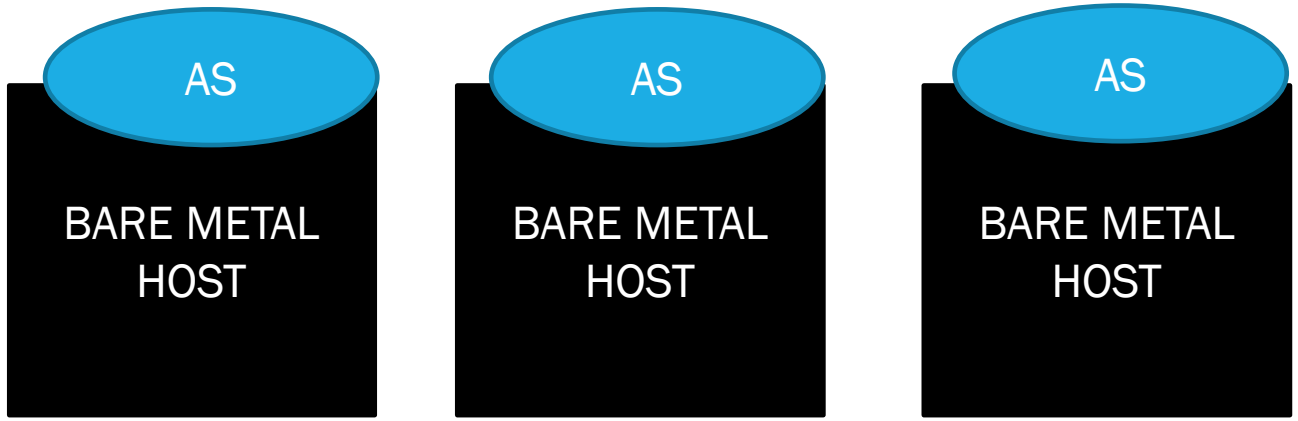
INFRASTRUCTURE

- DISCONNECT VPN
- ORACLE VIRTUALBOX 6.0 OR ABOVE (MANDATORY) → www.virtualbox.org
- Hub.docker.com
- Github.com
- App.wercker.com
- App.slack.com
- 16 GB RAM / 20 GB HD FREE /C Drive
- DOCKER INSTALLATION ? (Not Recommended Docker Desktop for Windows)
- **Virtualization /Enabled , Windows – HyperV (unchecked and should restart)**

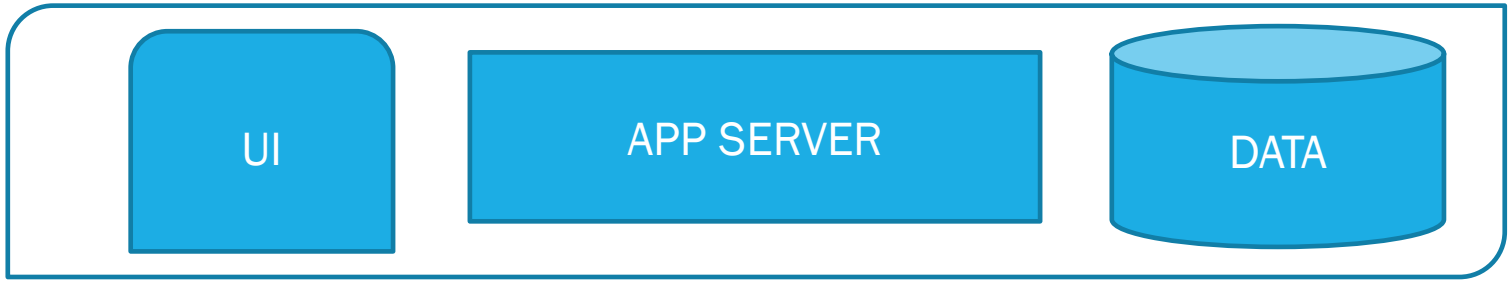
DEPLOYMENT

1 Package → Application + Environment → MAKEFILE (Executable) – Runtime and Code.
2 Package → Features and Dependencies → JAR /WAR/EAR → JRE (Runtime requirement)

Linear Kind of Application



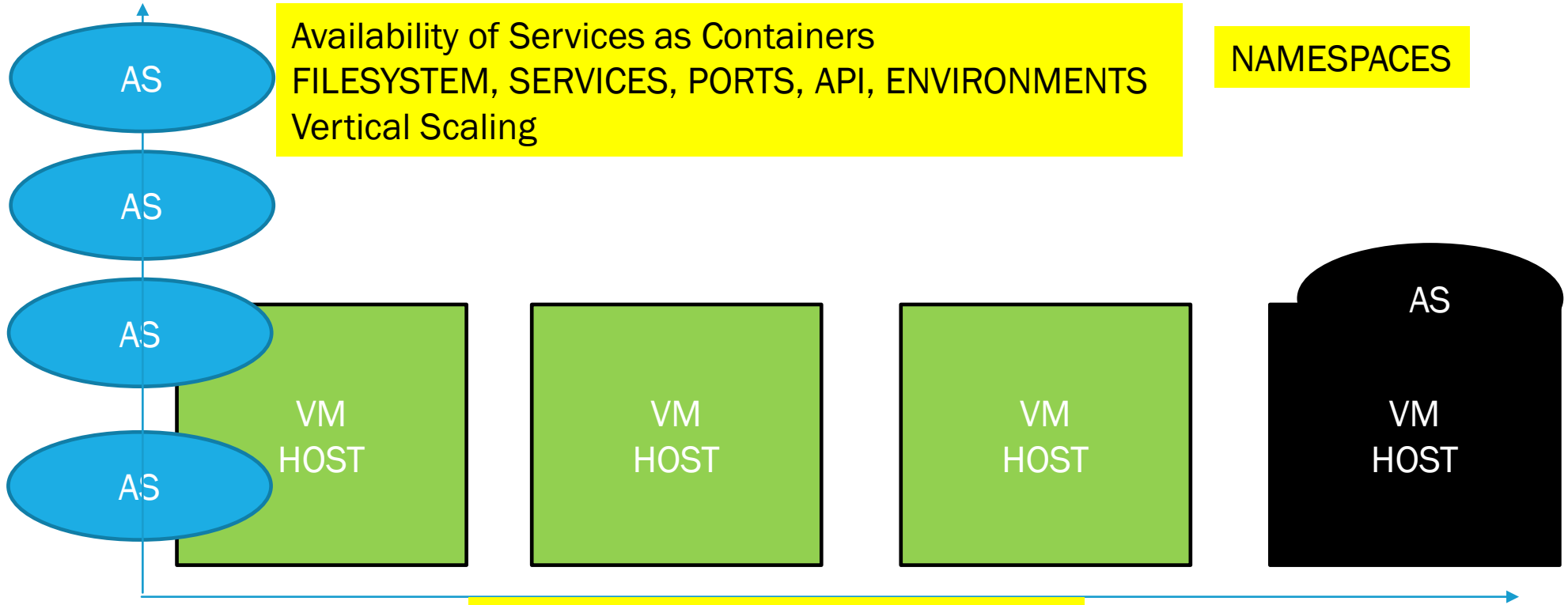
Service Layers
Workloads - VM
SOA



START TIME: Mins
Manage resources Dynamically
Fixed Operating System

DEPLOYMENT – SCALE CUBE

Images for Containers as Portable
Containers are Light weight- Performance
Scalability – Service/Infrastructure



NODE POOL1 – 3 X VM
NODE pOOL2 – 2 X VM
NODE POOL3 – 1 X VM

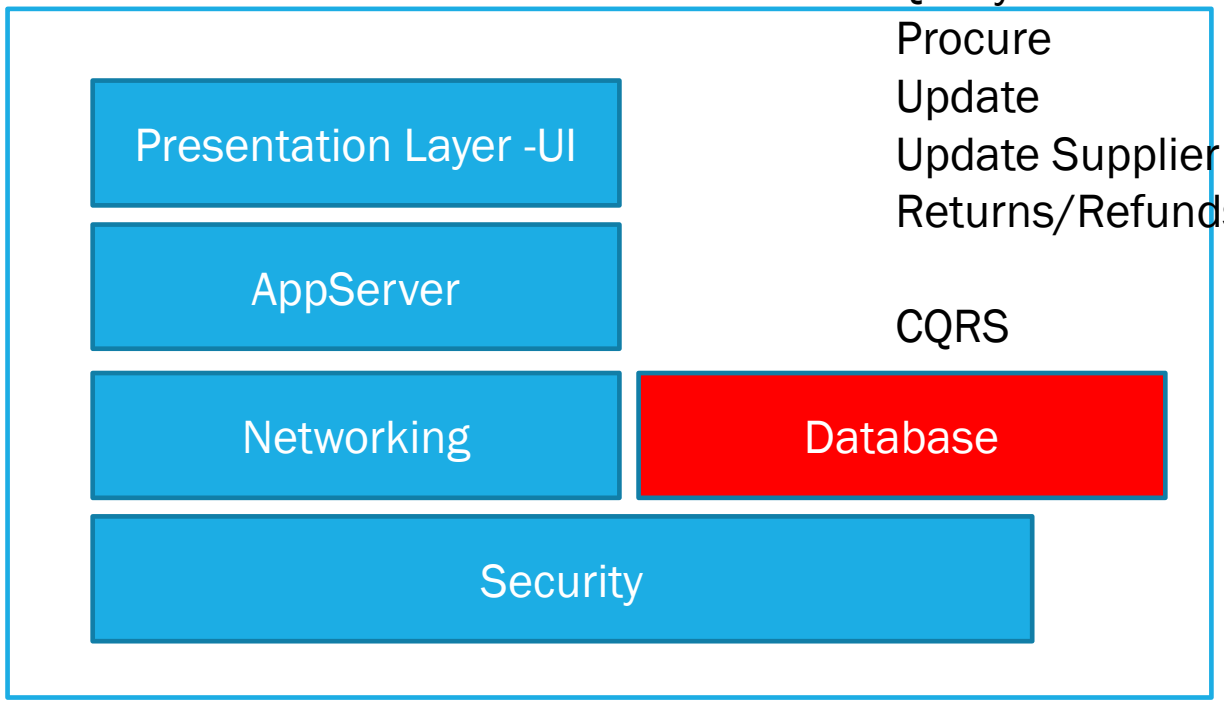
MONOLOTHIC BECOMES OLD STONE AGE

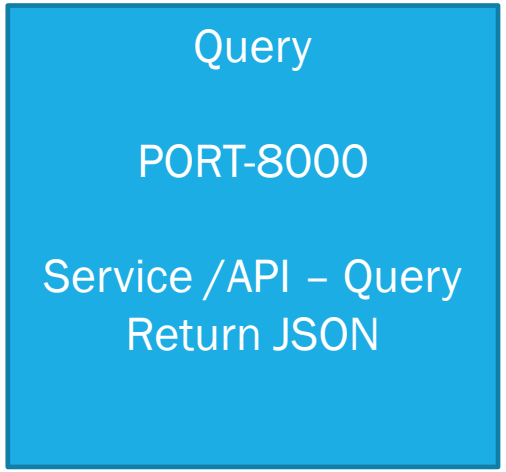
- END POINTS /Custom_Image?product&id=&abc
- DEPLOYED as MONOLOTHIC UNDER VM/HOST

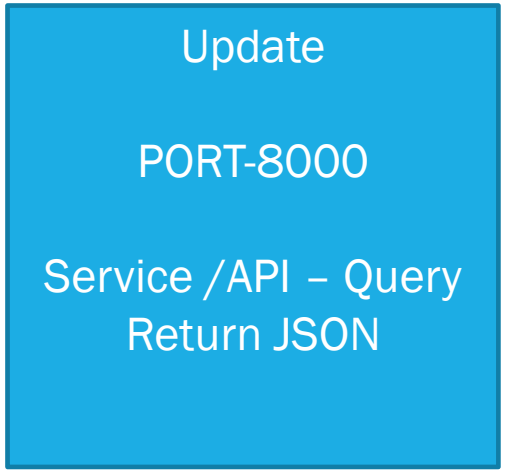
Change management → Difficult
Integration Challenges
Cloud
→ Rollouts → High Availability
→ Ha for Infra/ Business Services
→ Performance of Application
→ Cost of Infra → CSP

Divide them into Smaller Services
Query
Procure
Update
Update Supplier
Returns/Refunds

CQRS







MICROSERVICE ARCHITECTURE

- *Independently Scale*
- *Independently Deploy (Containers)*
- *Independent Data Stores (no SQL …)*
- *Independently Defined*

- Rollouts are easier (changes)
- Performance management – Less expensive
- Automate Applications (Abstracted from Infrastructure)

Too Many Rollouts → 14 days /Release
Vertical Scaling (Services)
Horizontal Scaling (On Demand)

Many Changes
Development Perspective – Continuous Development
Continuous Testing
Continuous Deployment
Continuous Release (Rollouts frequent)
==========
No Impact to Business
No Down time to Infra/App Services

EVOLUTION

App Delivery	App Architecture	App Deployment	App Integrate
Iterative	Monolothic	Bare Metal (Host)	Data Centers
Agile (Dev, QA)	Service Oriented Architectures	Virtual Machines	On Premises
Devops (Dev + Ops) CI/CD/CF	Microservices Architecture	Containers Deployments	Cloud (OCI)

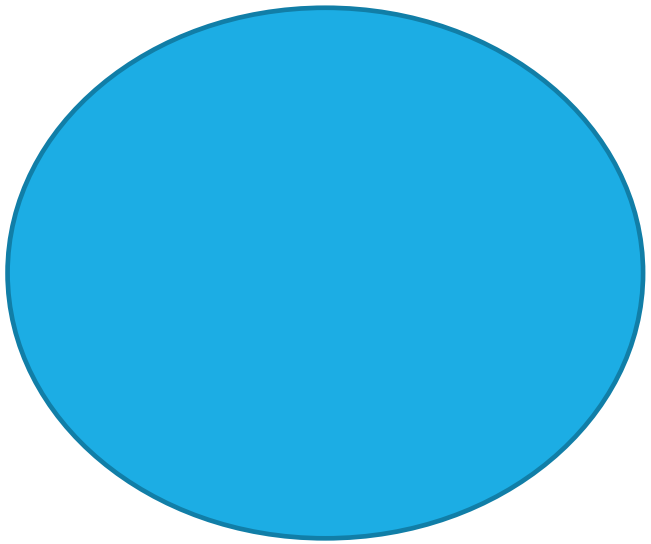
ARCHITECTURE

Develop Features	Deploy Features	Maintain Features	Change Features	Hosting Features
Code Style of Architecture	Architectures	High Availability	Rollouts	Security Identity Connections
MakeFile Packaging Archive	VM Container	Manual Automation	Manual (Downtime) Automated (No downtime)	Infra – Hosting Setup Manual/Automated
Java/Python/Spring	Docker /Maesos	Automate Services – Orchestrator		Cloud Infrastructure
MSA	Container as process	Kubernetes as software		Cloud Service provider

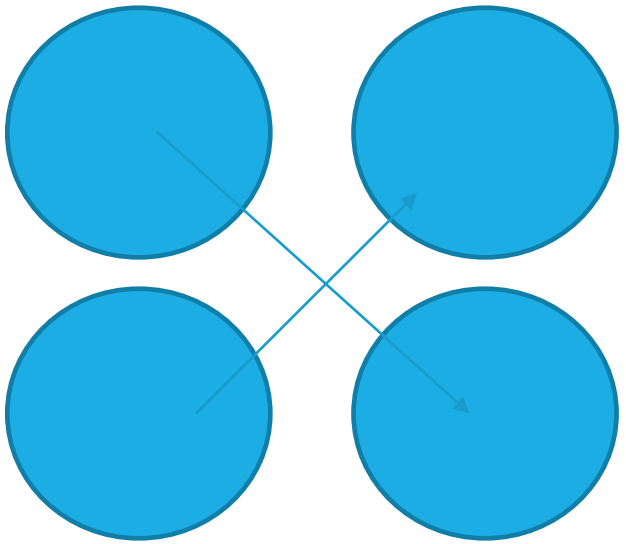
DOCKER VS K8S

Container Runtime	Cluster Container Runtime
Abstract Infrastructure from Services	Scale of Infra/Services (Horizontal/Vertical) on Demand
Images associated with Services → Portable across Platform	Self Heal – Services /Infrastructure Monitor Utilization /Services and Ascertain action – Repair
Light weight process – Dynamic resource management	Orchestrating my Services/Infrastructure
Runtime which will manage resources for all containers- provide infra, maintain logs like events..	Cluster of Container runtime / Infrastructure. – Networking, Identity and Application Support
Logs, Monitoring , Troubleshooting (CLI)– runtime	Any Container runtime
CRUD for Services (Images), Containers (process)	On demand scale /Customize , Open Source
Define Services , Deploy them as Containers	Scale Services on Demand (High Availability)
Docker , Maesos, Zones, Rkt, Rancher, LXC, OCI (open container initiative)	Maesosphere (Apache), Vagrant Cloud (Vagrant), Swarm (Docker), **Kubernetes (k8s)**

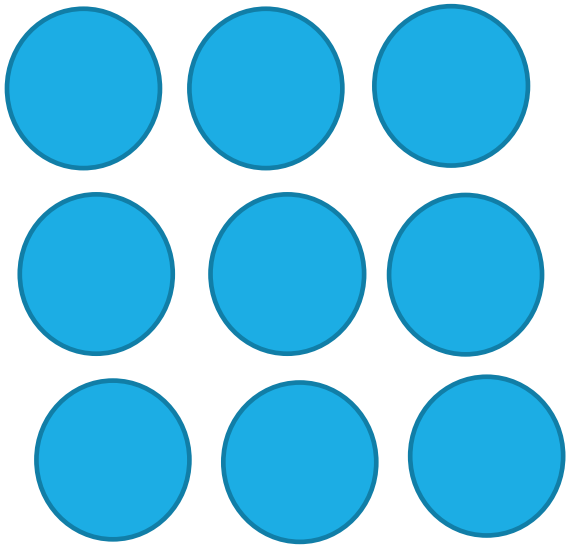
ARCHITECTURAL STYLES



Tighter Cohesion and Integration
Monolithic Architecture



Coarse Grained
Service Oriented Architecture

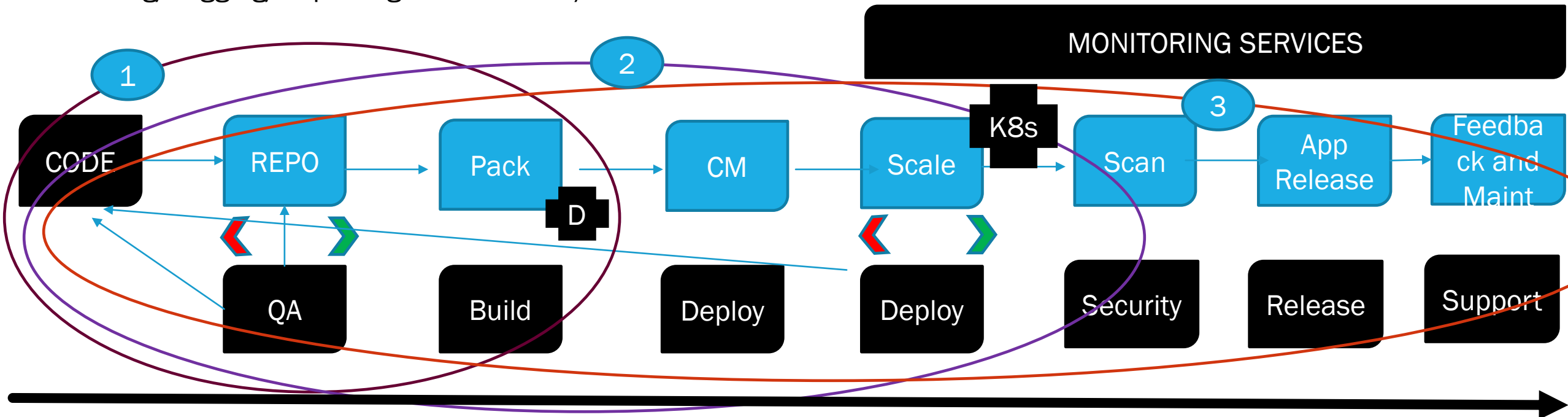


Fine Grained
MSA

DEVOPS = AUTOMATION OF CI+CD+CF

Architecture Style → Type of Services (Mono/SOA/MSA)
Deployment Style → Containers/VM
Orchestrate /Administer → Manual/Automation
Source code Storage → Repository (SCM) – File /Directory
Storage Servers → Volume of Data
Hosting → Cloud / Data Center/On Prem (Security)
Monitoring/Logging/Reporting -> Dashboard/review

1 Continuous Integration (QA and Dev) - Black Circle
Code – Repo – QA – No – Yes – Build : BOX (Deployable)
2 Continuous Deployment (Ops, Performance, Scale)
Boxable Build – CM Tools – Deployment – Scale – PT – Releas
3 Continuous Feedback (Release, BRM, DevSecops, Support)
Release – Security – Support – Customer (Feedback)

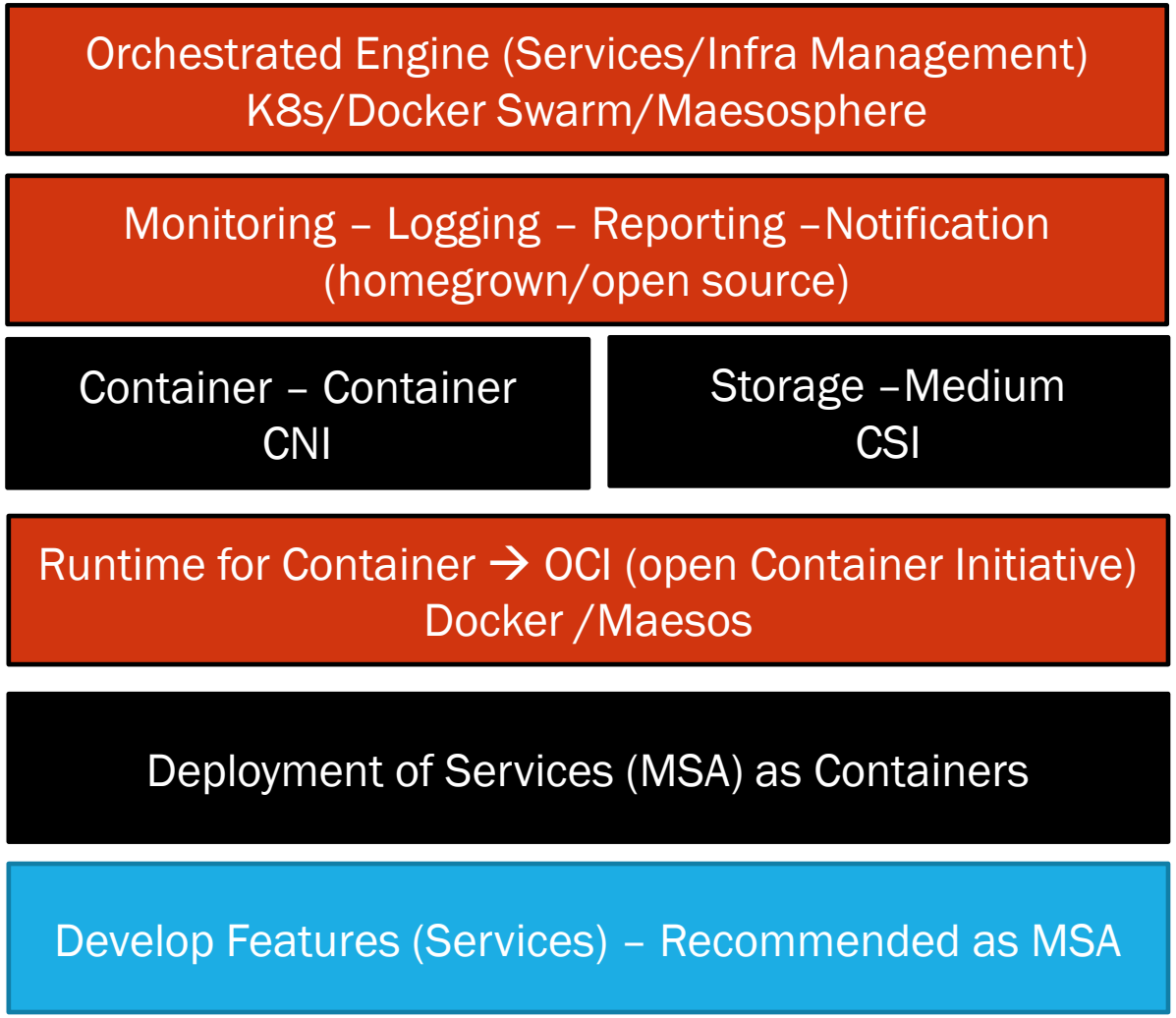


CNCF

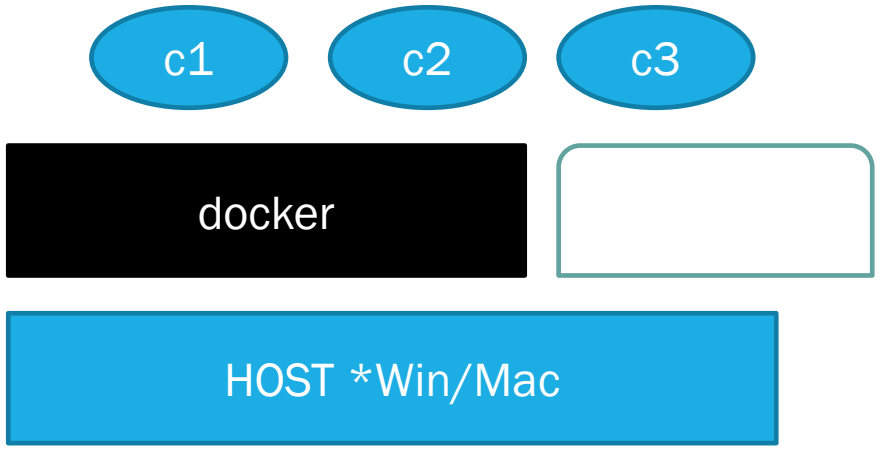
All CSP to standardize services

Most CSP – Extend beyond these Frameworks, OKE , AKE, AzKe, GKE

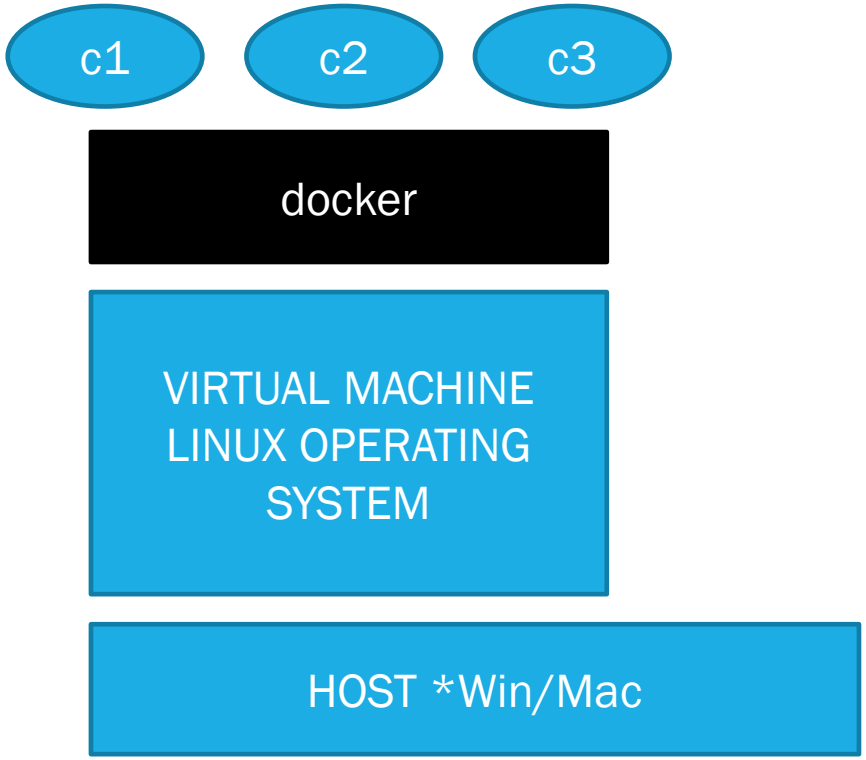
Open Source Tools
CNI , CSI , Container Runtime, testing application, Monitoring Tools



DOCKER INSTALLATION



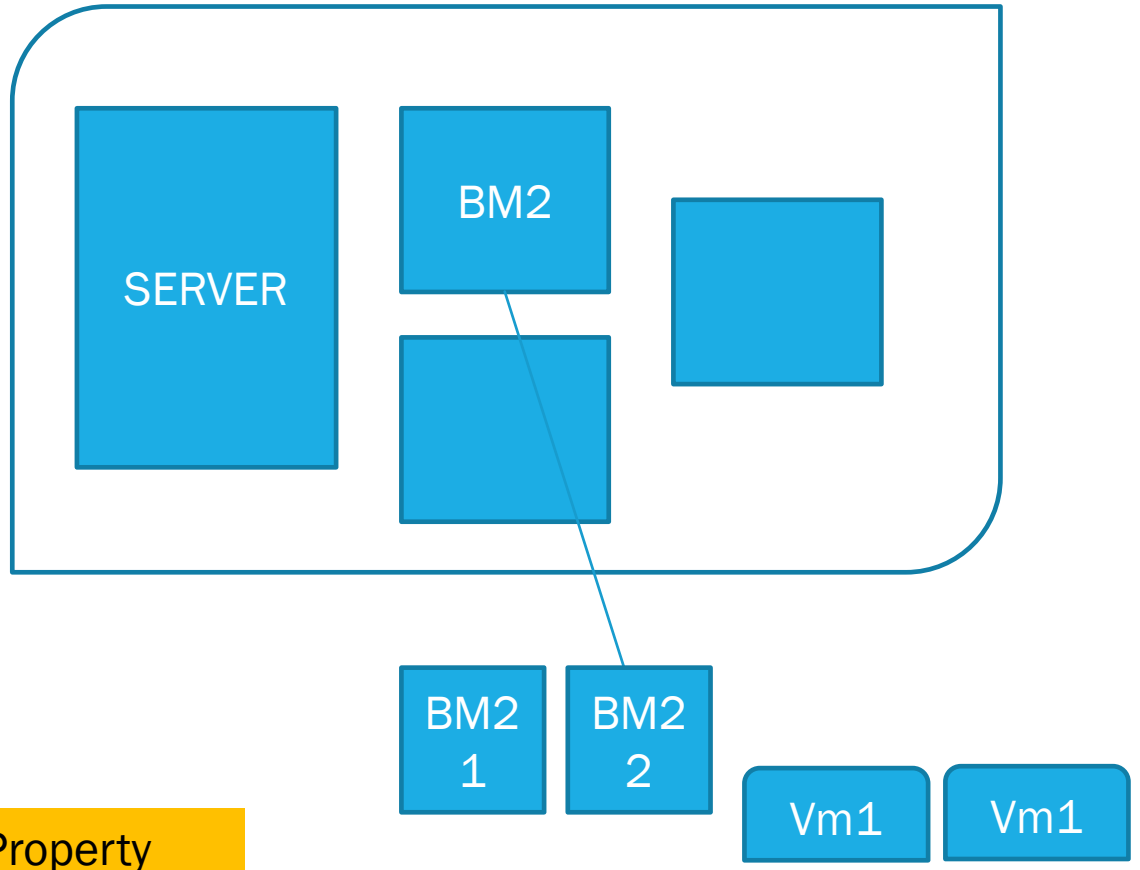
Docker Desktop for Windows



Docker ToolBox for Windows/MAC

CLOUD (OCI)

- BAREMETAL (SERVER BOX) – HOST
  - REGIONS (REALM)
    - AVAILABILITY DOMAIN (AD1)
    - AD2
    - AD3
  - TENANCY (LOGICAL COLLECTION OF RESOURCES)
    - ROOT COMPARTMENT
      - COMPARTMENTS (PRIVILEGES OR POLICY)
        - USERS

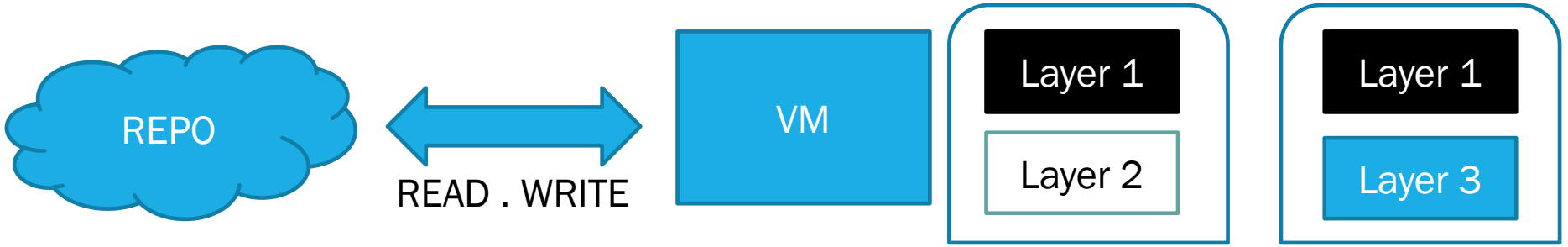


Mi,, Gi, Ti, Ki ---------- K , M , G, T

JSON Notable Property
Digest of Layer in SHA256
Automation in YAML

IMAGE … READONLY .. STATIC

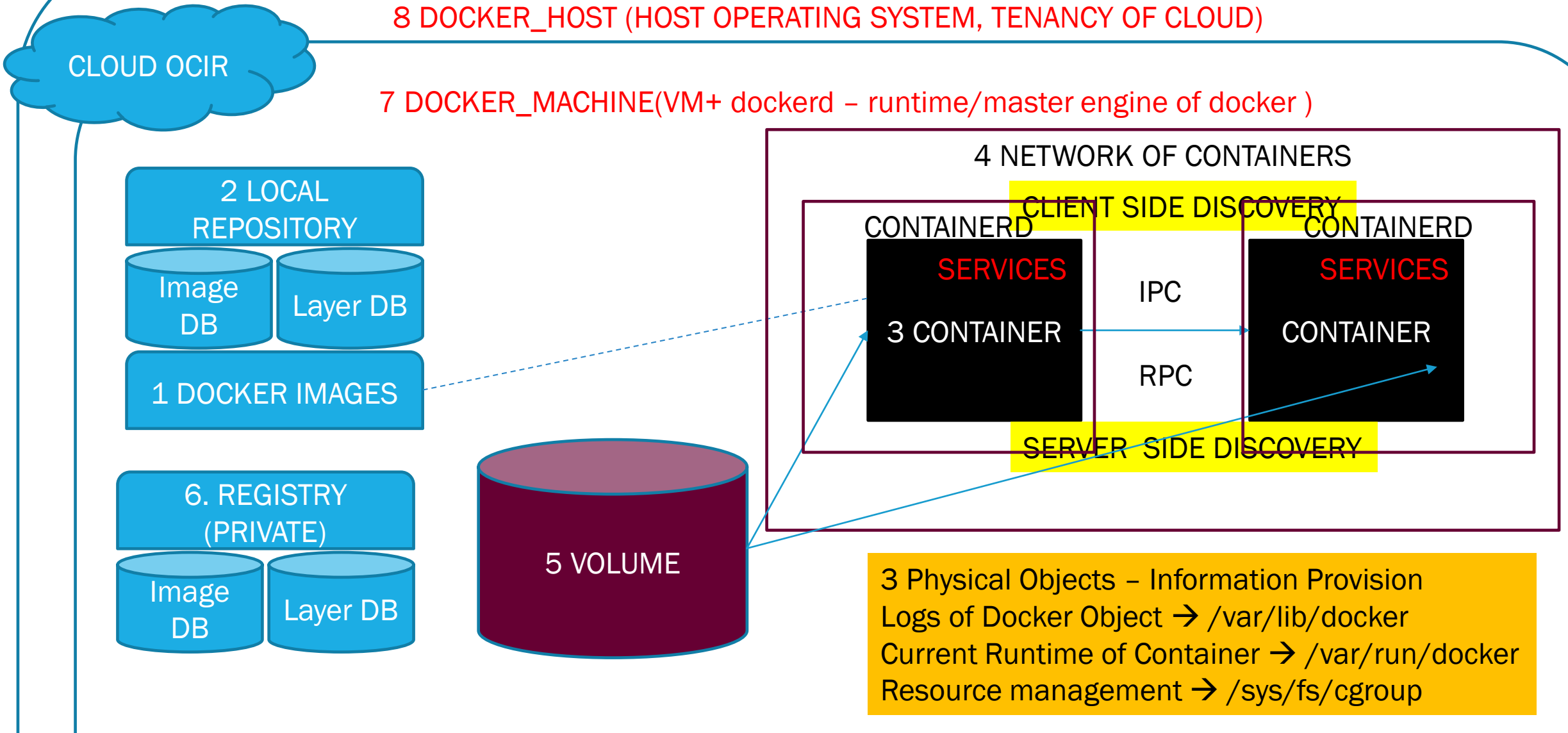
DOCKER CONTAINER IMAGE	VM IMAGE
PORTABLES, OBJECTS	FILE (ISO)
PORTABLE /COMPRESSED /SHA256	NOT PORTABLE / 32/64
OBJECT	FILE SYSTEM
Check Sum/Digest of Layers - One or more Layers	**ISO 4.6 GB (OEL 7.5)** ISO 4.7 GB (OEL 7,6)
If Present (Layers) – Incremental Update	FULL DOWNLOAD /UPLOAD
OBJECT STORAGE LIBRARY (IMAGE REPOSITORY)	REPOSITORY (SCM)



MSA → CONTAINERS…

VM	DOCKER CONTAINER
INSTANCE OF ISO	INSTANCE OF DOCKER CONTAINER IMAGE
STATE	STATE – RUNNING, EXITED, PAUSED , RESTARTING
BOOT OF MACHINE – LOAD FILE SYSTEM	LOAD ONLY REQUIRED – LAZY LOADING
RESOURCE MANAGEMENT STATIC	RESOURCE MANAGEMENT ARE DYNAMIC
HORIZONTAL SCALING (CAPACITY)	VERTICAL SCALING (SERVICE AVAILABILITY)
VM- VM TRAFFIC	CONTAINER – CONTAINER TRAFFIC
IP ADDRESS – SSH KEYS (RSA) , CERTIFICATES(PEM)	IP ADDRESS (RUNNING)
MONOLOTHICS	MSA/SOA/MONO

DOCKER ARCHITECTURE



INSPECT IMAGE

- PARENT
- TIMESTAMP / AUTHOR (META DATA)
- CONFIG
  - EXPOSE PORTS → SERVICE API
  - CMD → DEFAULT COMMAND TO GET EXECUTED WHEN A CONTAINER IS STARTED
  - ENVIRONMENTS (KEY CONFIGURATION)
- LAYERS

PROCESSES

USER PROCESS	SERVER PROCESS
SHELL PROCESS , IDE SHELL	WEBSERVER, DATABASE SERVER, APP SERVER DAEMON PROCESS – INTERACTING WITH FILESYSTEM AND SOCKET
ENTRY , STDIN . STDOUT	LISTENING TO PROCESS , CONNECTING VIA SOCKET/PORT
→ - I (Small i) → Interactive (Run in Foregroun) → -t (tty device) → terminal → stdout	Background Process
→ -I –t	→ -d (detach)