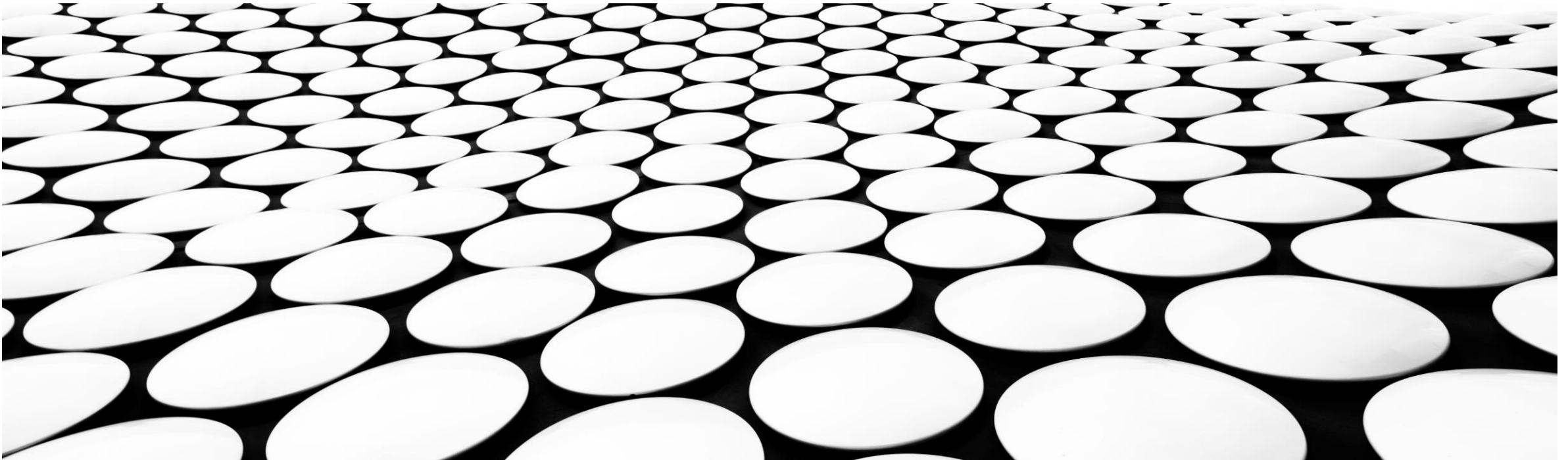

CONTAINER PIPELINES WITH ORACLE CLOUD

DJ (DHANANJAYAN)

DAY 1 – DAY 5

29TH JUNE – 03 JULY 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

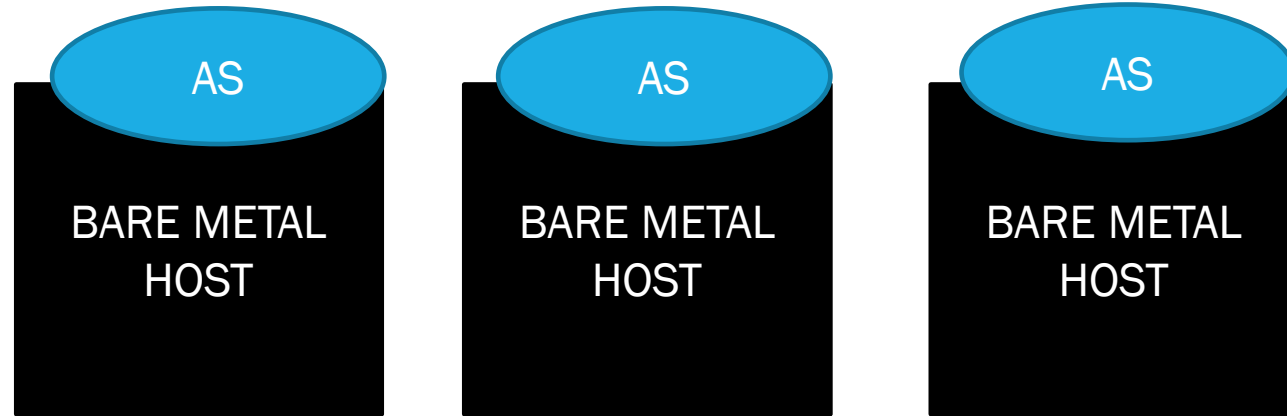
INFRASTRUCTURE

- 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)

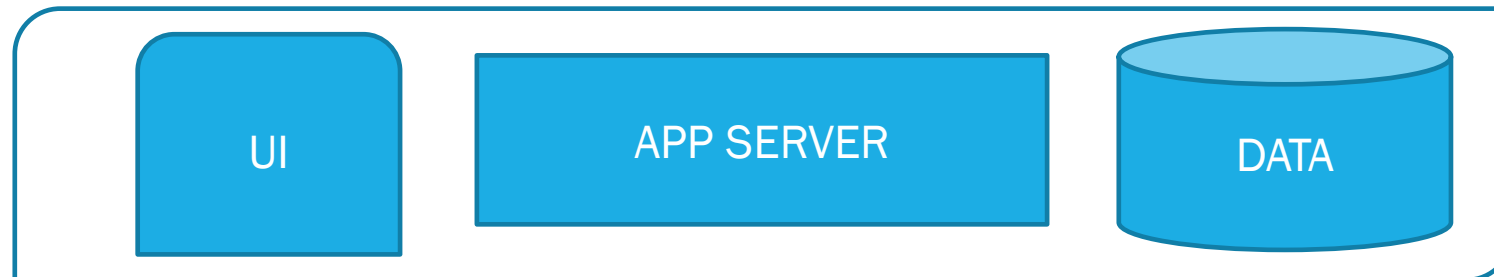
DEPLOYMENT

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

Linear Kind of Application



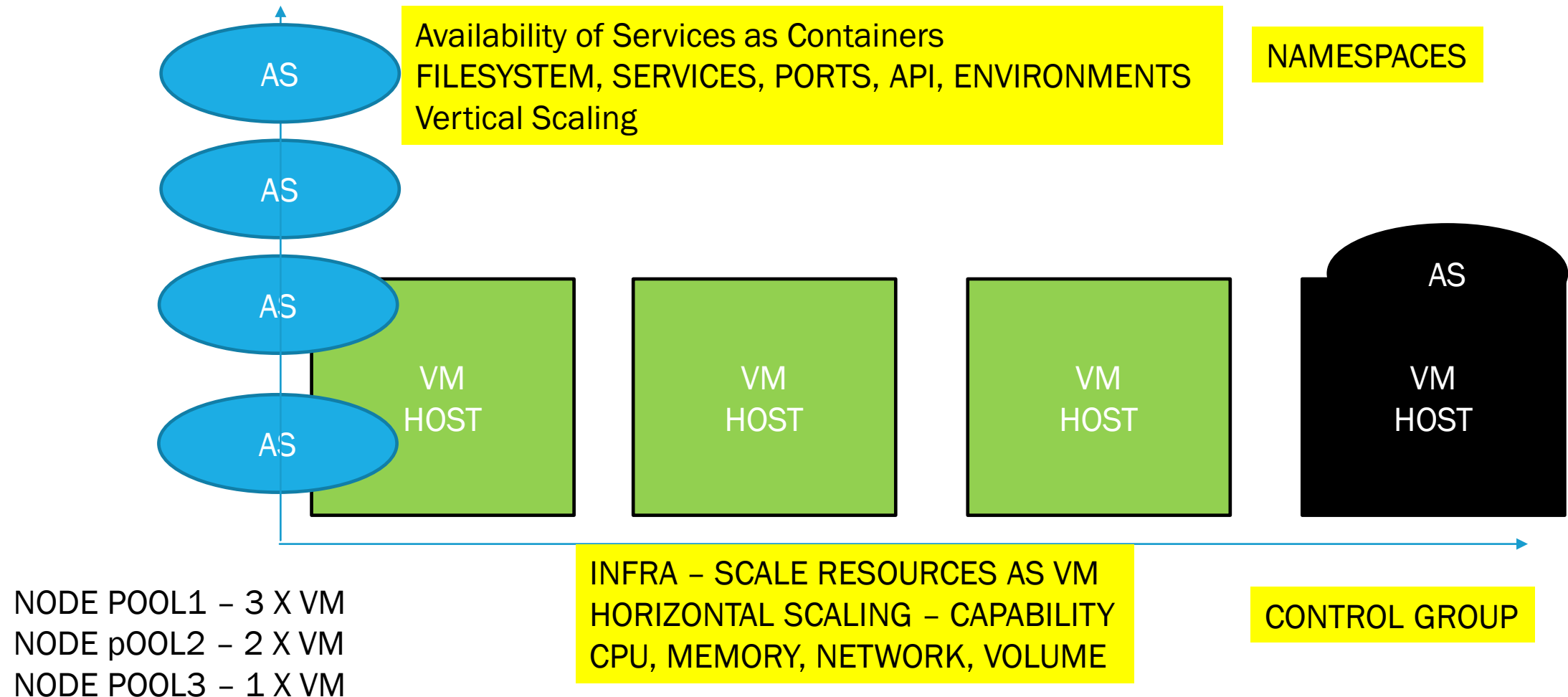
Service Layers
Workloads - VM
SOA



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



MONOLITHIC 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

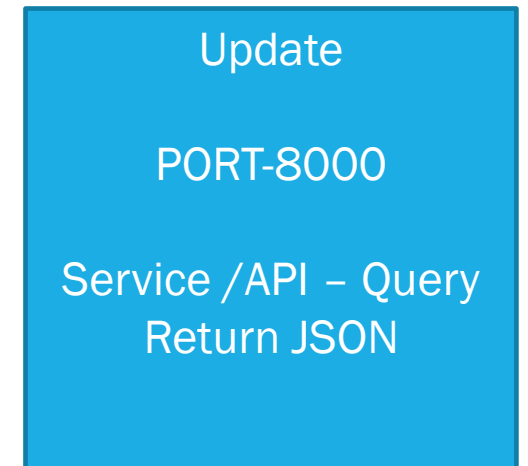
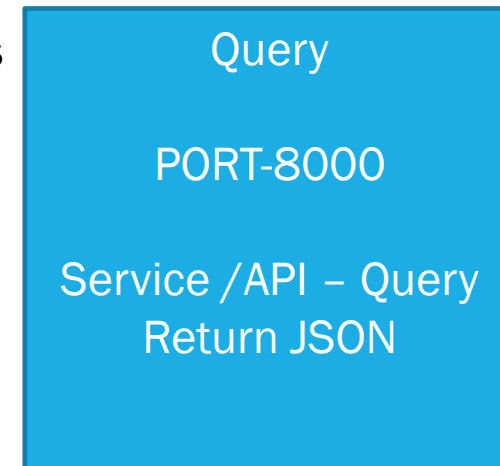
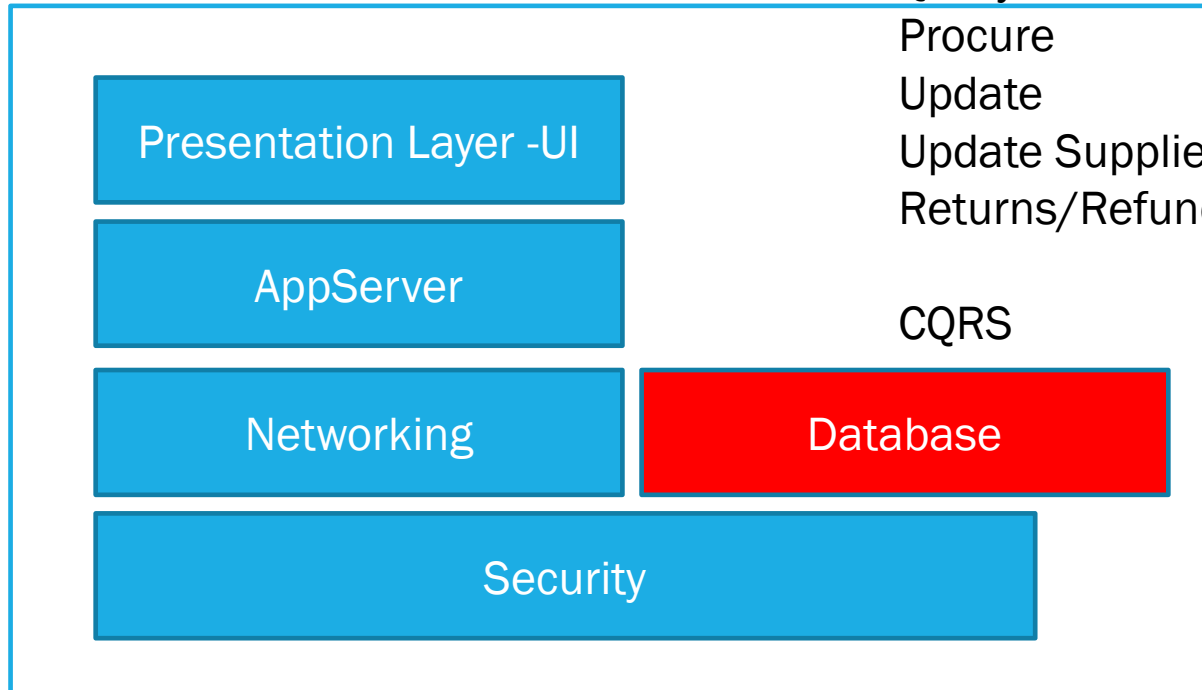
Query

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	Monolithic	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)

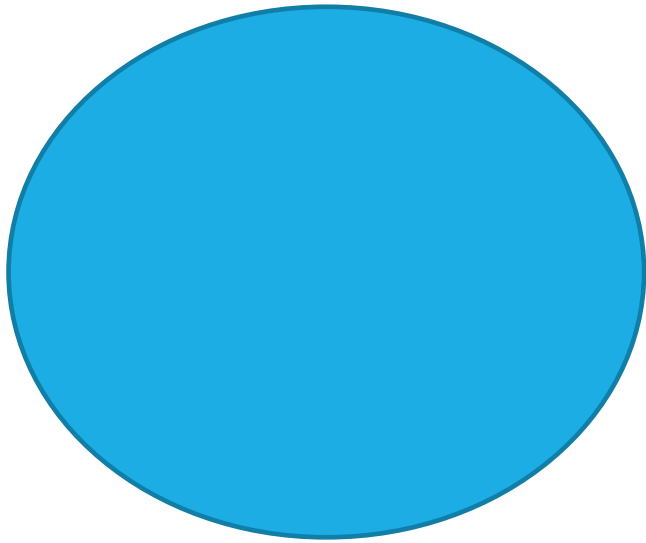
ARCHITECTURE

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

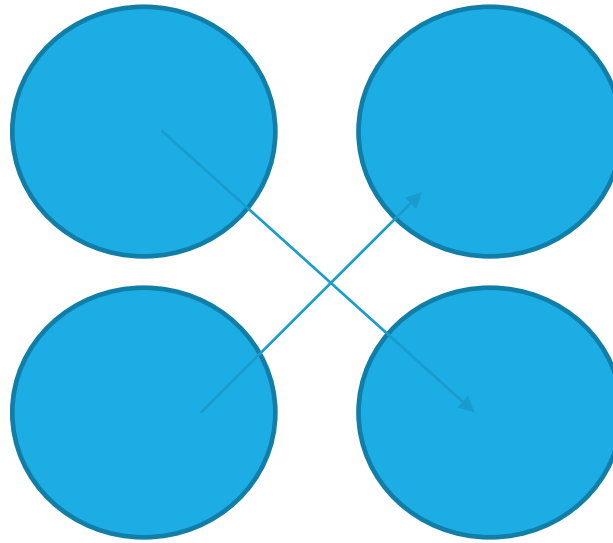
DOCKER VS K8S

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)

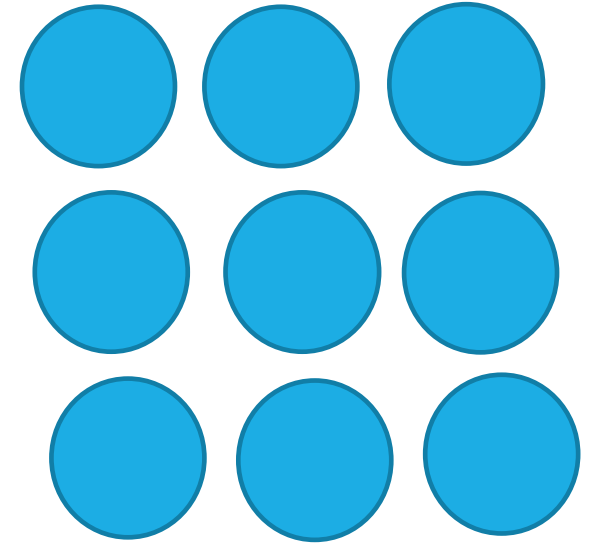
ARCHITECTURAL STYLES



Tighter Cohesion and Integration
Monolithic Architecture



Coarse Grained
Service Oriented Architecture



Fine Grained
MSA

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

Orchestrated Engine (Services/Infra Management)
K8s/Docker Swarm/Maesosphere

Monitoring – Logging – Reporting –Notification
(homegrown/open source)

Container – Container
CNI

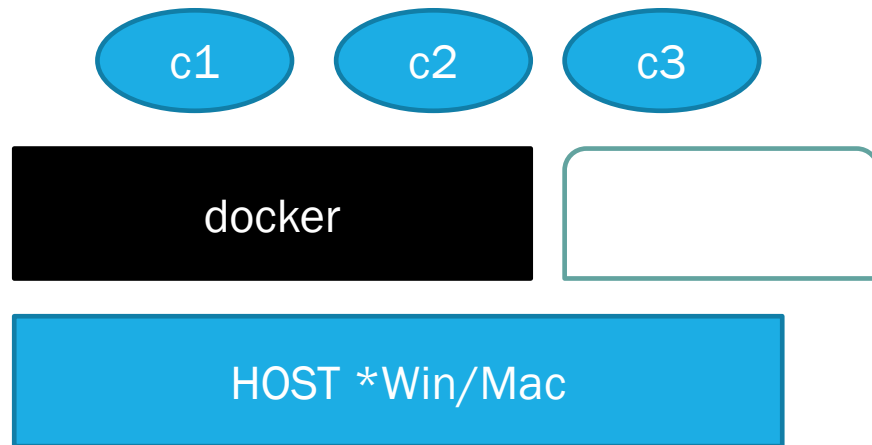
Storage –Medium
CSI

Runtime for Container → OCI (open Container Initiative)
Docker /Maesos

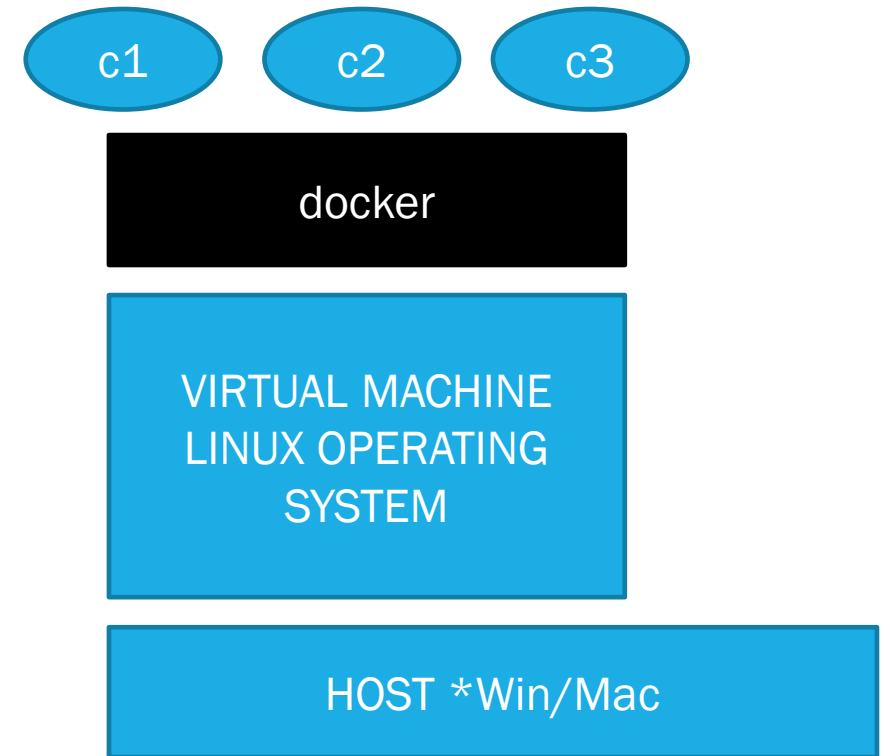
Deployment of Services (MSA) as Containers

Develop Features (Services) – Recommended as MSA

DOCKER INSTALLATION



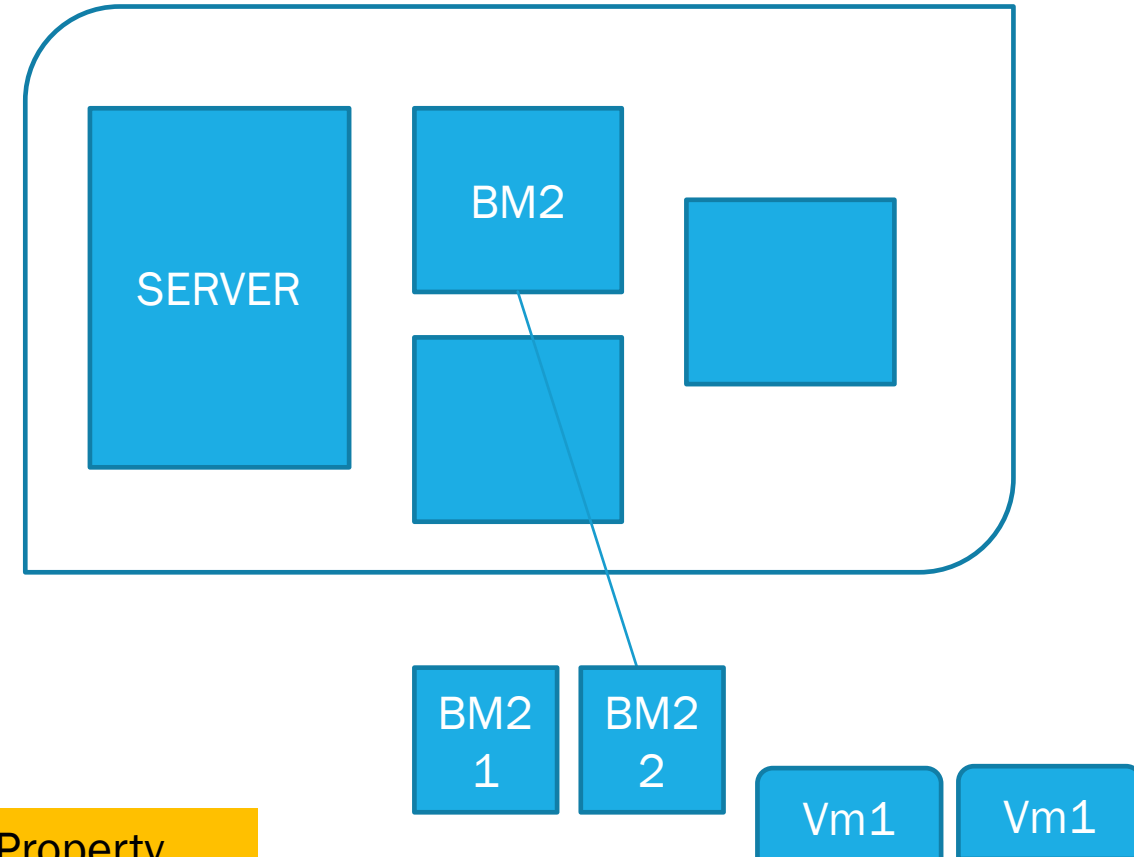
Docker Desktop for Windows



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

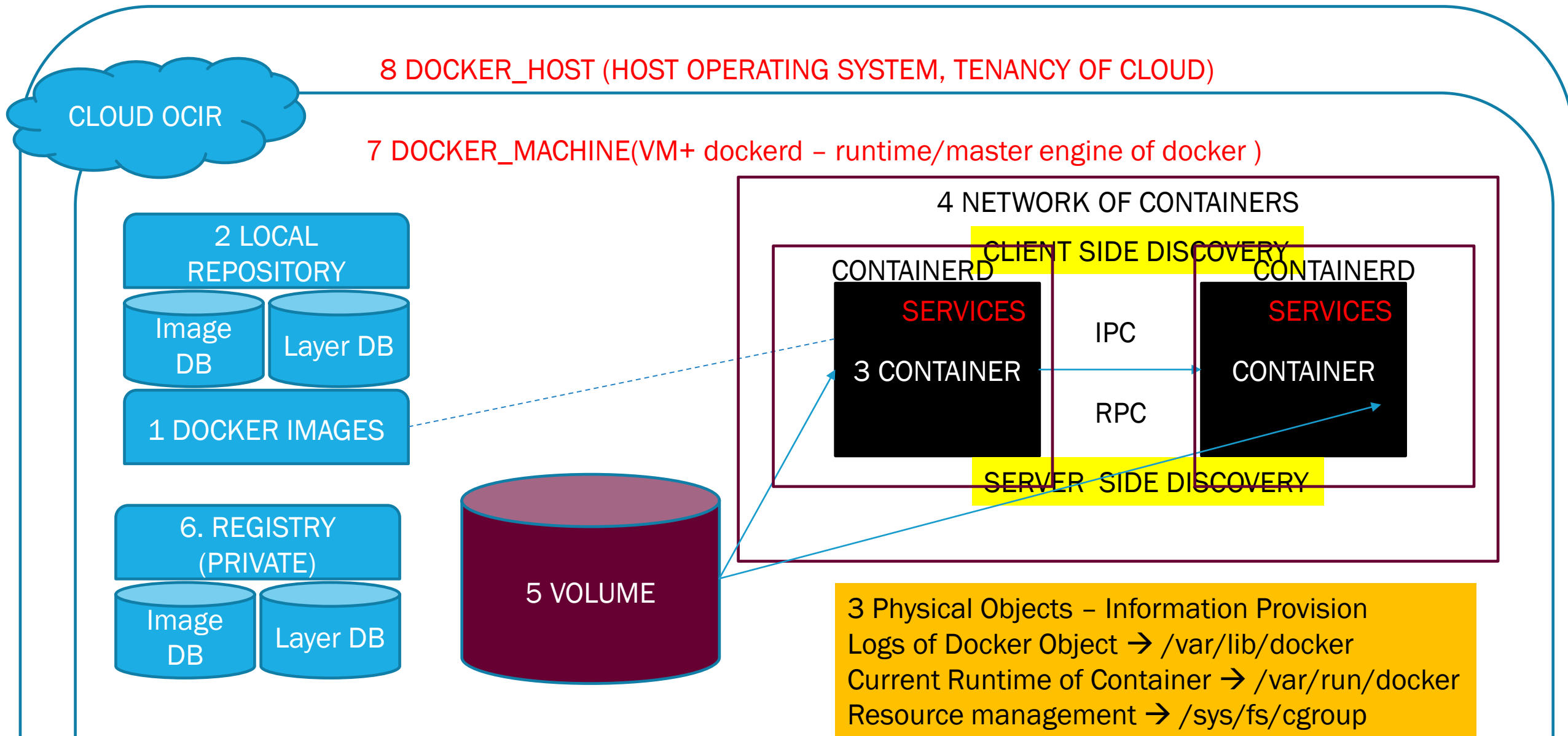
DOCKER CONTAINER IMAGE	VM IMAGE
PORTABLES, OBJECTS	FILE (ISO)
PORTABLE /COMPRESSED /SHA256	NOT PORTABLE / 32/64
OBJECT	FILE SYSTEM
Check Sum/Digest of Layers - One or more Layers	ISO 4.6 GB (OEL 7.5) ISO 4.7 GB (OEL 7,6)
If Present (Layers) – Incremental Update	FULL DOWNLOAD /UPLOAD
OBJECT STORAGE LIBRARY (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)
MONOLITHICS	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
→ -l (Small i) → Interactive (Run in Foregroun) → -t (tty device) → terminal → stdout	Background Process
→ -l -t	→ -d (detach)

DAY 2

- Manage Containers
- Check Logs /Troubleshooting
- Dockerfile
- Expose Services
- Use Case – MSA Object
- Use Case – Database as Service
- Health of Service
- Port Forwarding Service
- Monitoring Services ?
- Share images to HUB, OCIR and Data Center

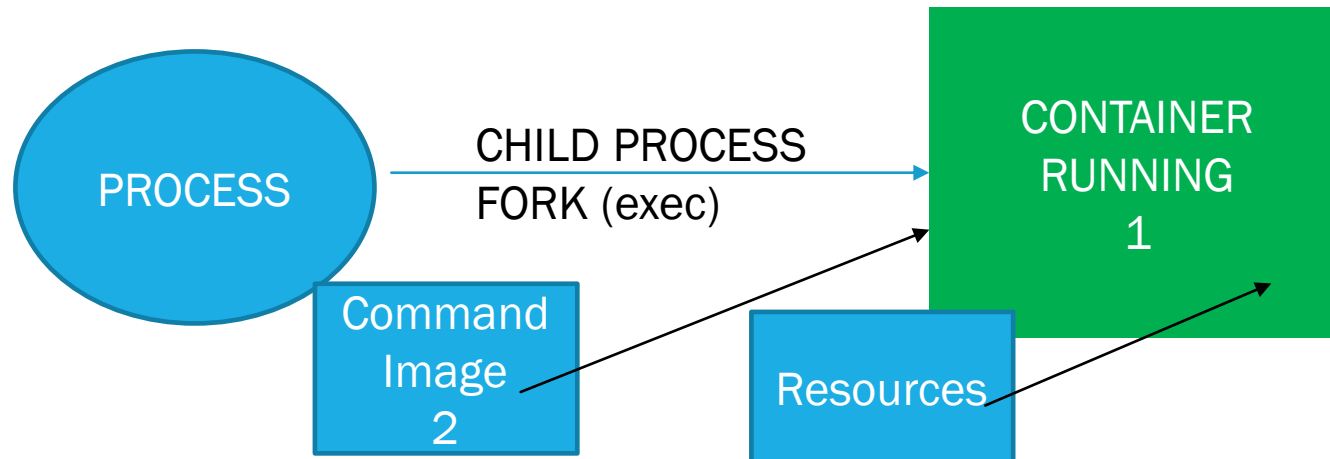
MANAGE CONTAINERS

- # docker start <container ID > or <container name>
 - `expression (back quote operator)` → UNIX
 - \$(expression)
- # -q (ids of the command)
- Extract from JSON
- → {{ Interpolation }}
- Case Sensitive , → “.” Attribute Operator
- {{.State.Status}}
- #docker inspect -f “{{ }}”

LINUX CONTAINERS

Read anything from Container
Read a Service/File
Monitor in Container
Explore Process in container.

- File for Image (Dockerfile)– Base Image
- /etc/resolv.conf → Nameserver file.
- /etc/hosts → information about host



RUNNING CONTAINER

PARENT PROCESS	FORKED PROCESS
Administer resources for Container Manage the Child Process of container Configure env.variables of container	Read a Log File Monitor a Process Report a Service File Systems.
Parent ps → Server Process → Listener to connect to Parent ps → user process → (bash)	#exec
# attach	Child process .. Exit – Context lies only to child process
# --detach-keys, impact the process within container	Parent process still continues to run

CUSTOM IMAGES

Container → Image	Dockerfile	VM → DIS
# docker commit	# docker commit as below SOP (detailed document..step..)	# rancher vm
Limited Layers, No API Support Synchronous, Manual Deletion Not supported by Automation	Choice of Layers (publisher) API Support → Timing Asynchronous , background Automation Tools, CNCF, Automatically GC	# not approved by cncf # not automated # cost

CUSTOM IMAGE – STAGE 1

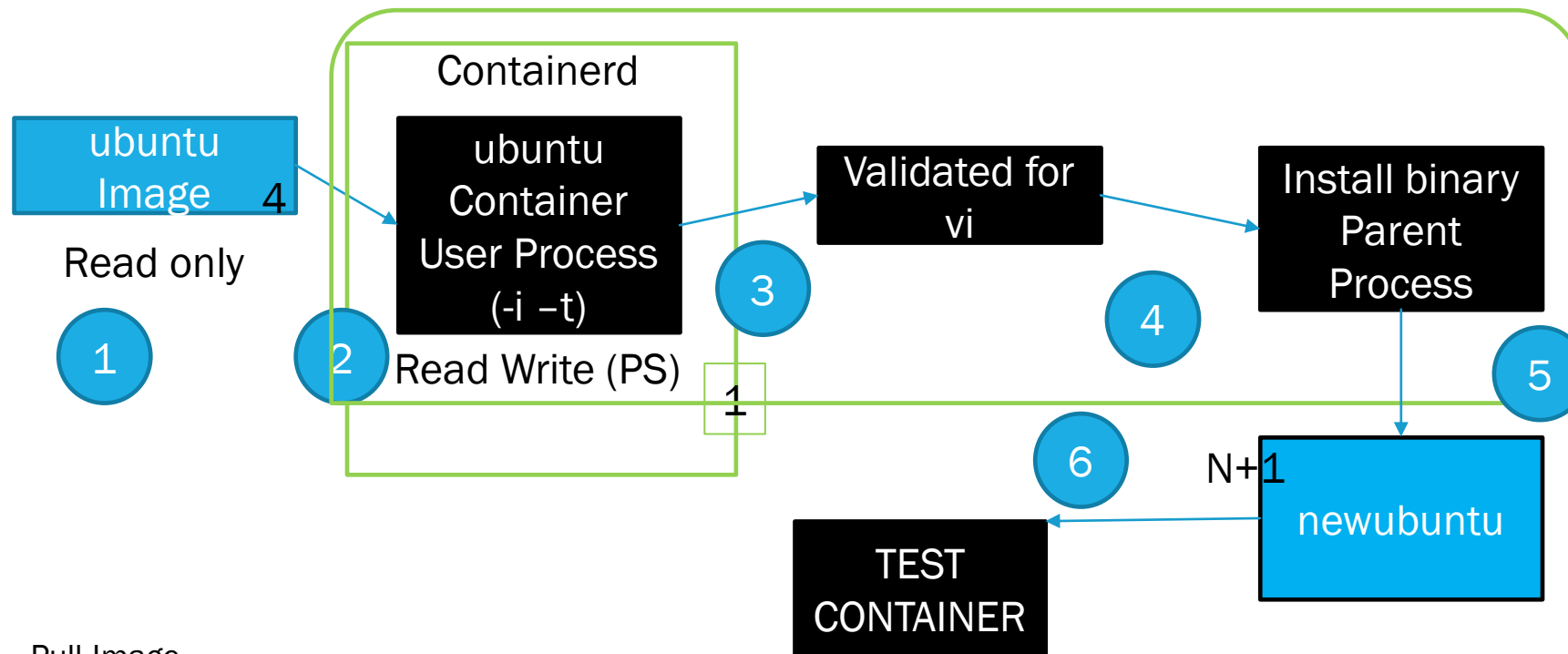
Entire CFS → Layer ?

Manually delete Containers

No API Support

Not supported by all cloud

Not supported by Automation



1 – Pull Image

2 – Create Container and start

3 – Exec into container and verify vi ? → Vi not found

4 – Parent Process – attach by detach keys → Install vi – apt-get install vim

5 – Container to Docker Image (ubuntu + vi) → docker commit container image-name

6 – pull, create cont, started, attach ... commands.. ..rm

DOCKER IMAGE

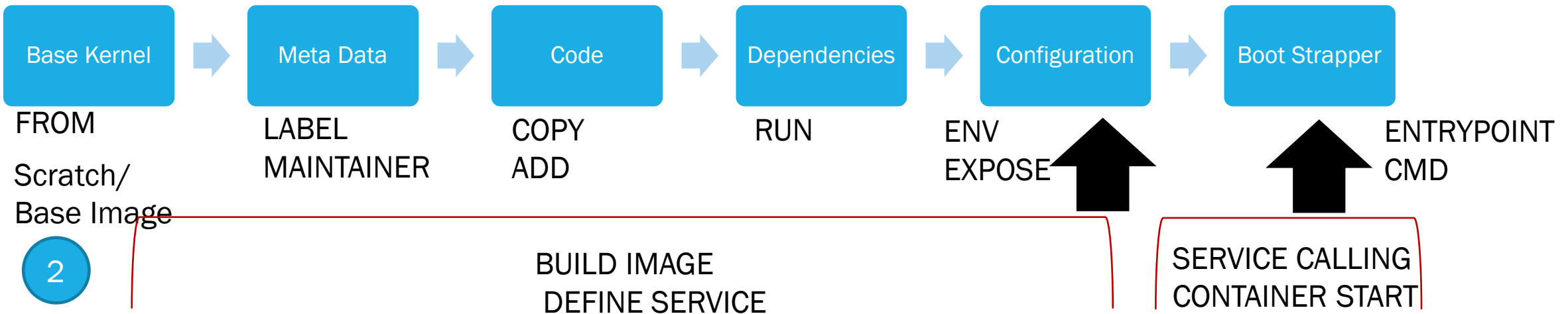
- MONOLOTHIC
- SERVICE ORINETED
- MICRO SERVICE ARCHITECTURE

1

- PACKAGING YOUR APPLICATION

- MAKEFILE
- ARCHIVE (JAR/TAR) -- RUNTIME

2



Dockerfile

Automation File to Build Docker Images
Choice of Layers
Asynchronous , CNCF – CSP will abide by rule.

```
# docker build -t <image> -f <Docker_file> <CONTEXT_ROOT>  
WEB_CONTEXT (CONTEXT_ROOT)  
Image name (-t) , Lowercase  
Dockerfile – custom Dockerfile (-f)  
Context ROOT →Directory path for dependencies for Dockerfile
```

USE CASE – SHELL APPLICATION

- SHELL APPLICATION → COMMAND LINE PARAMETER
- ARCHITECTURE – MONO
- PACKAGING is REQUIRED RUNTIME (BASH)
- DOCKERFILE
- BUILD DOCKERFILE (newubuntu:1)
- ENTRYPOINT [“sh”, “/code/Sample.sh”]
- CMD [“/etc/hosts”]
- INVALID FILENAME
- VALID FILENAME
- **NO PARAMETER FILENAME**

IMAGE → APPLICATION (SHELL)

Source Code – Path ?

Flexibility:

\$0 → FIXED

\$1 → Parameter 1

Fixed Boot Strapper

Sh /code/Sample.sh

Variable Parameter

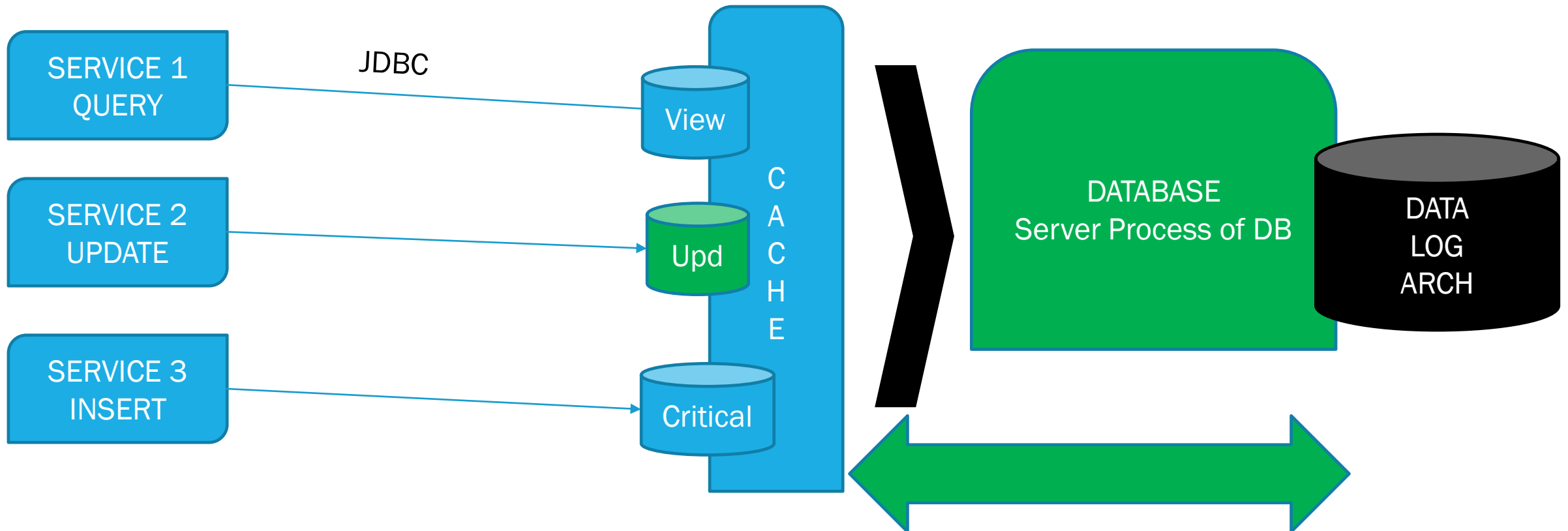
/etc/hosts

COMMAND LINE PARAMETER

[LIST OF ARGS]

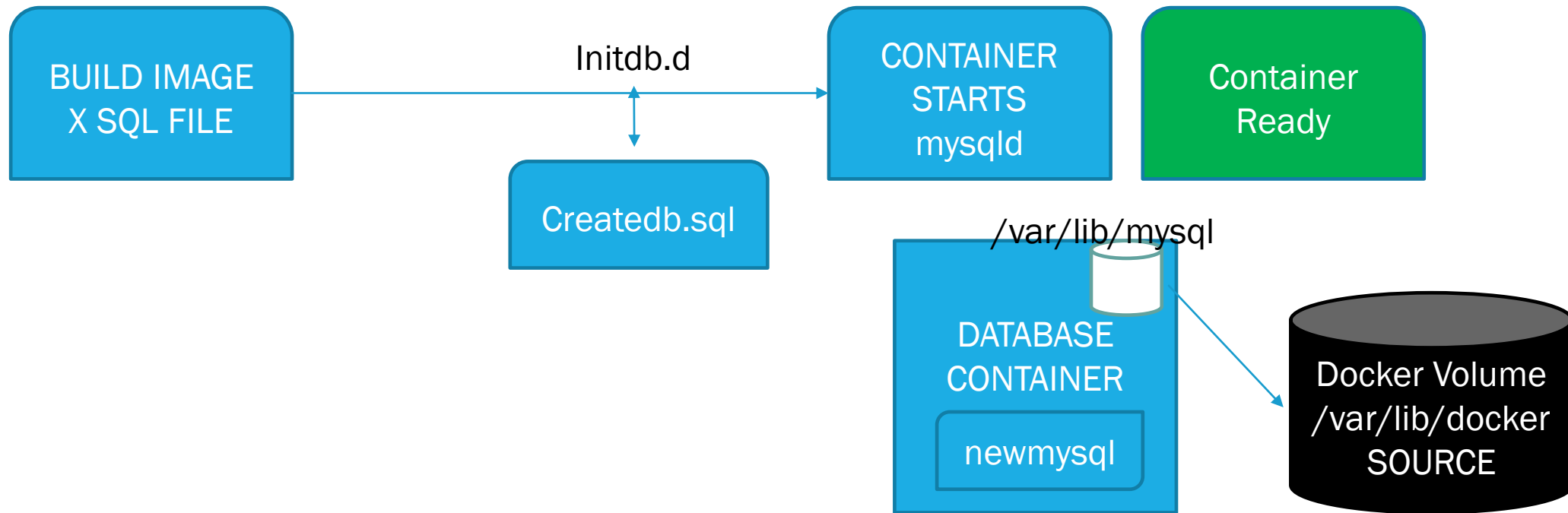
USE CASE (2) : DATABASE

- MICROSERVICES AND DATABASES

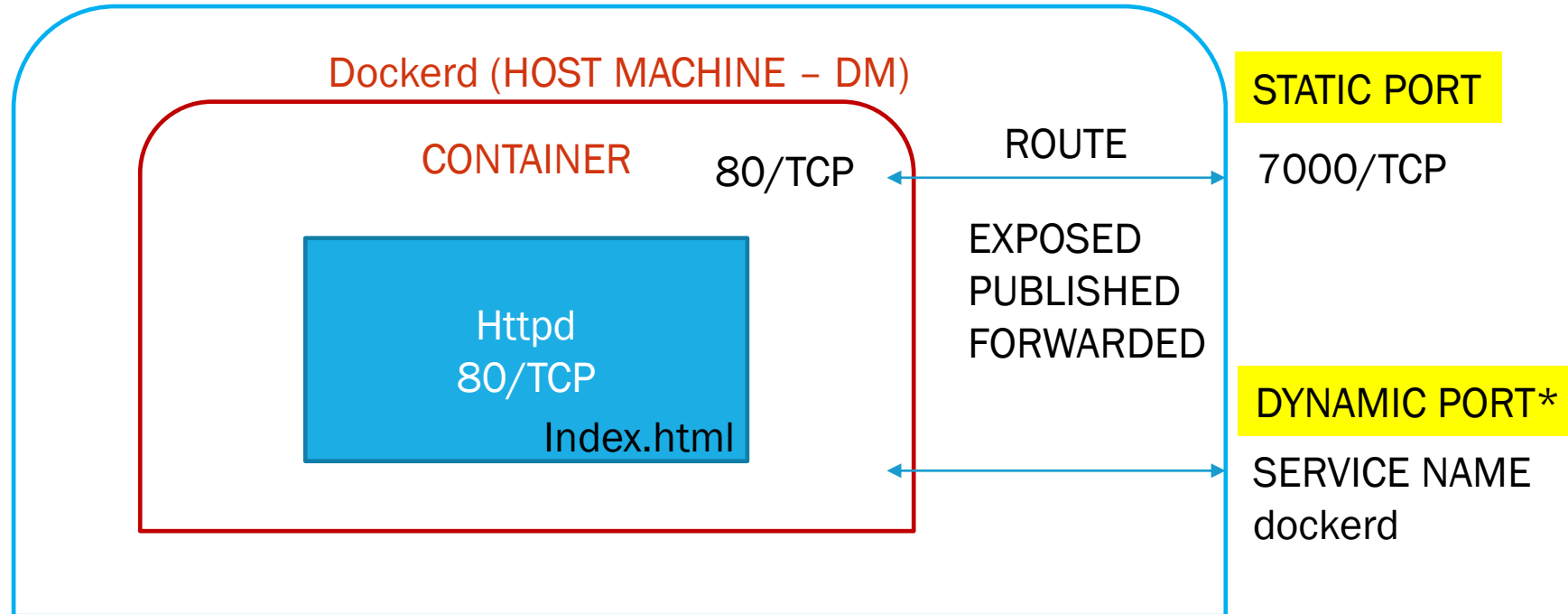


USE CASE : DATABASE CONTAINER

- MYSQL_ROOT_PASSWORD → TO START A DATABASE
- PATH DIRECTORY OF INITDB.D → /docker-entrypoint-initdb.d



SERVER PROCESS -- INTERACTION



Synchronous Call
REST API Call
1-1 Communication
Request - Response
Format - Developer
Indefinitely Block
Timeout=Output
100 - Information
200-300 - OK
300 - Circuit
400 - End point
500 - Server

ASynchronous Call
MQ Calls
0 - # Communication
Request - Response
Format - Developer
Timeout=Output
Pub - Subscriber

FORWARDS SERVICES THROUGH RPC

STATIC PORT FWD	DYNAMIC PORT FWD
Forwarded Port to Host – Decided by Ops (Devops)	Forwarded port to Host – Decided by dockerd
LESS THAN 30,000	32768 – 35999
ACCESS by port number	Access by Service Name
Knowledge Management – Allocate Free port Iptables –t nat –L*	Automatic Sequence.
Start/Stop Container – Restart – Forwarded port Fixed	Forwarded port Fluctuate /Service Name
-p HP : CP -p 7000:80 –p 7001:100	-P

USE CASE : REST API (MICROSERVICES)

- Independent Service

- Definition (Source Code)
- Deployment (Container or VM or Host)
- By Replication (SCALE)
- Private Data Store

NODE JS – Server Side Scripting
DATA Store (JSON)
9000/TCP
ListUsers → All users in JSON
Response → JSON

- PORT – 1 Primary (1 – Backup)

- MULTIPLE END POINT
 - <http://domain.abc.oracle.com:8000/abc> --> End point → Redirect to specific module

- LISTENING PORT (Primary + Backup Port number)

USE CASE : APP SERVER (REST API)

PREPARE
CODE



BUILD
Dockerfile



Docker
Image review



Container
Creation with
Port Forward



TEST
Container

--node.js
--data.json

```
FROM node
LABEL MAINTAINER dj@appserver.com
COPY node.js /code/node.js
COPY data.json /code/data.json
RUN npm install -y express body-parser
#metadata for Docker image. Does not validate.
EXPOSE 9000
CMD node /code/node.js
```

docker run...

USE CASE: HEALTH...

Dev (Define)

Container Running ?
Response for EndPoint ?
Format of Response

```
#curl for service  
# docker ps -a
```



Deployment
(Performance)

Response time within Threshold
Proactively being tested
output of threshold...
Services is Healthy

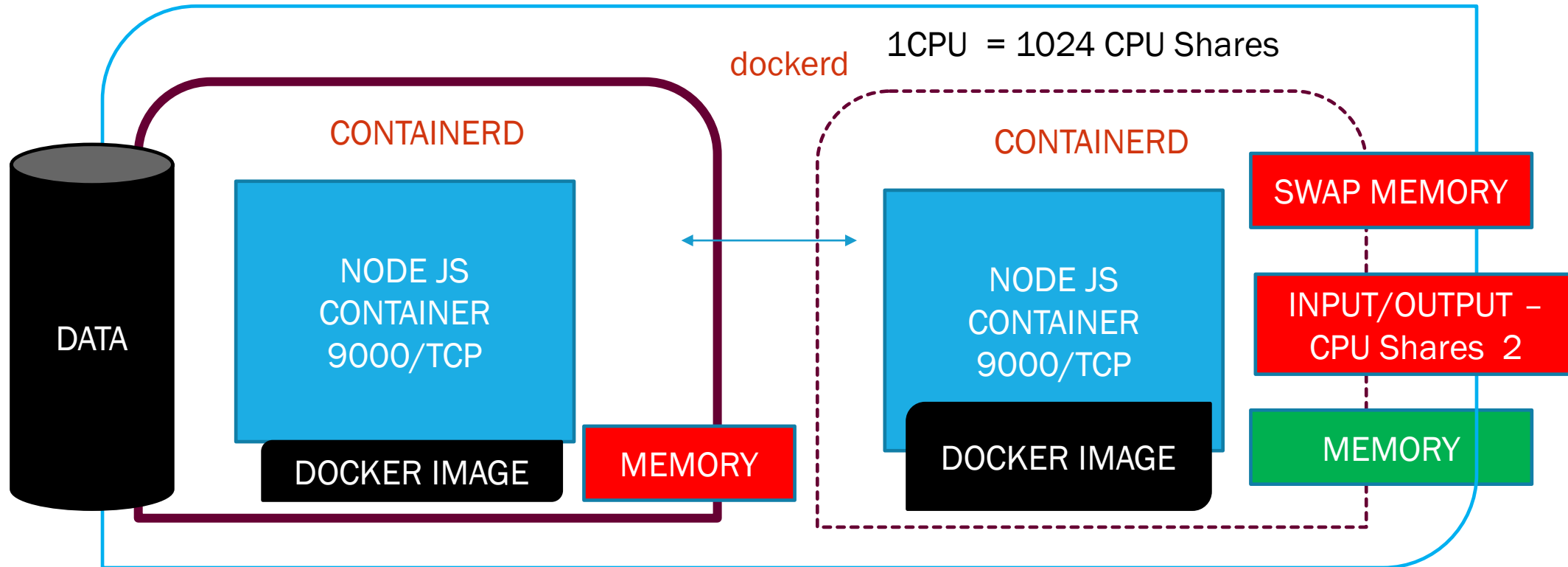


Operations
(Infra)

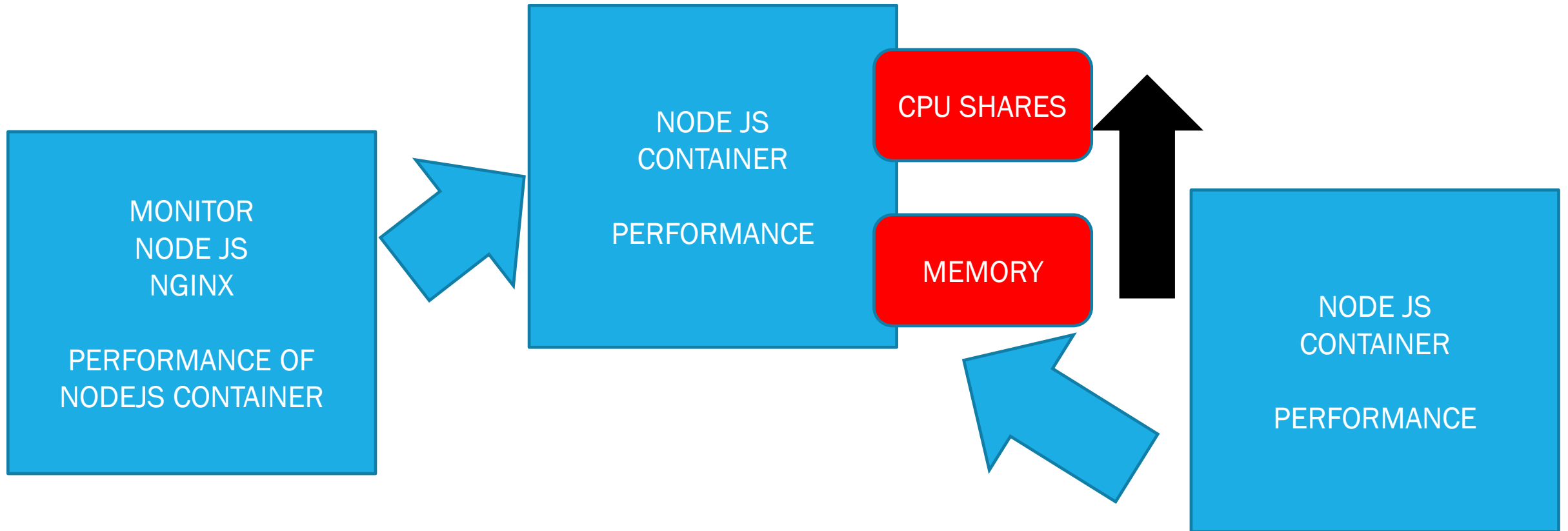
Infrastructure are resource
within acceptance of subscriptions
Resources are Healthy

```
#docker update..
```

USE CASE:...MSA FOR INFRA (RESOURCES) HEALTH ?



USE CASE : DEPLOYMENT HEALTH (SIDE CAR DESIGN PATTERN)



SHARE IMAGE

SAVE it in OCI	Save it in HUB (Docker HUB)	Data Canter (Backup)
<p>Tenancy – Share resources Tenancyname - ocuocictrng22</p> <ol style="list-style-type: none">1. MFA Auth Token (Oauth Token)2. SSO Repository: iad.ocir.io *use auth token as password * tenancy/username3. Prepare the Image repository/tenancy/image:tag <p>iad.ocir.io/ocuocictrng22/newubuntu01</p> <ol style="list-style-type: none">4. # docker push the tagged image	<p>Username /password – HUB</p> <ol style="list-style-type: none">1. SSO #docker login password as hub-password (account created for day 1)2 Prepare the image hub_username/image:tag3 # docker push <preparedimage>	<p>Image Object→ FILE (Serialization)</p> <ol style="list-style-type: none">1. # docker save -o <...tar> <docker image>

DAY 3

- RECOVER IMAGES
- EXPORT IMAGES (SINGLE LAYERED)
- RESTART POLICY FOR CONTAINERS
- NETWORKING FOR CONTAINERS – CNI
- TROUBLESHOOTING FOR DOCKER
- KUBERNETES ARCHITECTURE
- INSTALL KUBERNETES
- NODE ARCHITECTURE

RESTART POLICY

NO (NEVER)	ALWAYS	ON-FAILURE
Container will not restart when implicit stop happens	Container will restart always when implicit stop happens	Service Fails – RESTART Retries Fixed
No Restarts	Indefinite restarts	Exit Code (exit 0 – Success) (code non-zero – Failure)

Explicit Stop will not restart containers

docker stop

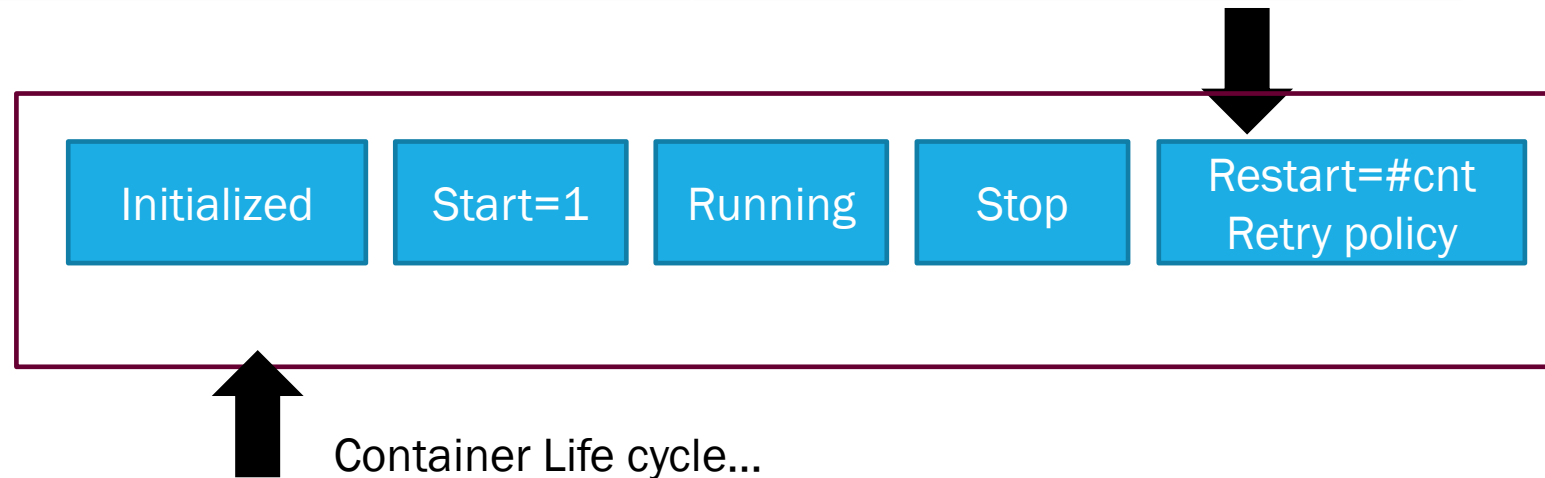
docker kill

Implicit Stop ?

→ Memory Resources (Lack of)

→ Service abruptly stops

→ Attach → exit without detach keys



LOGS FOR CONTAINERS...

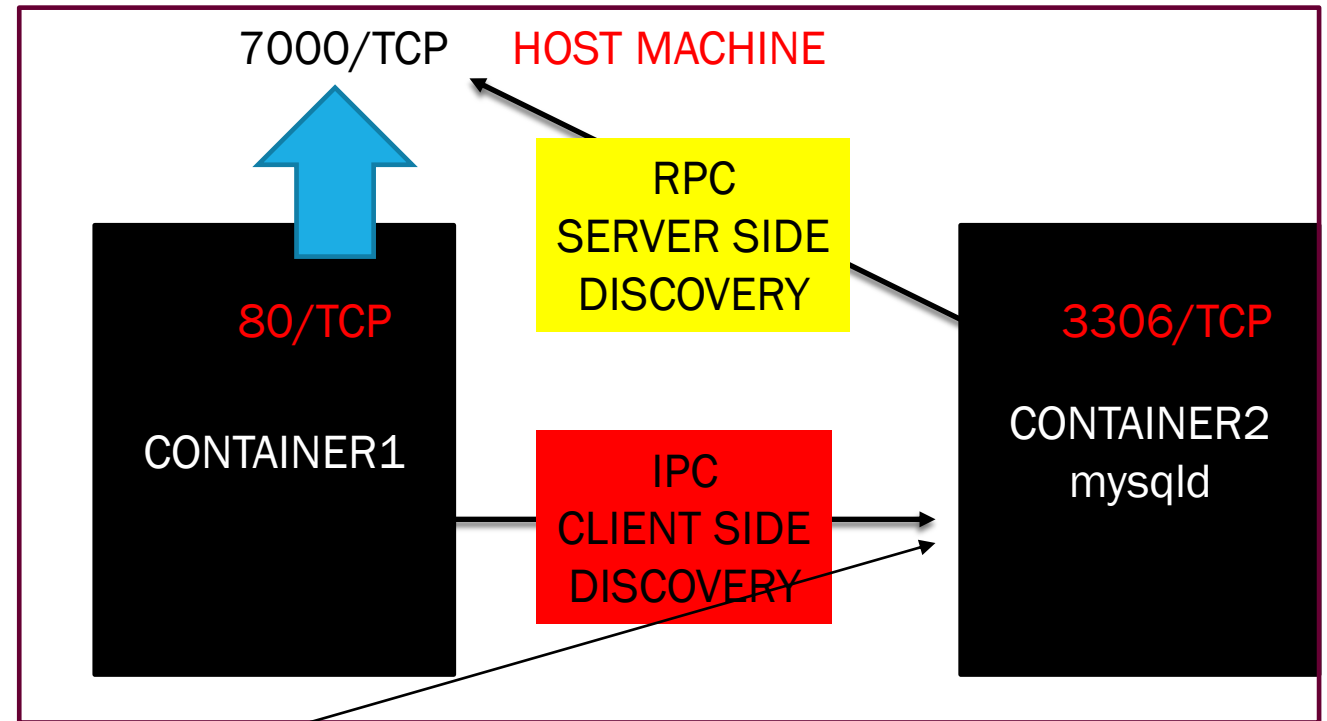
Image Logs → History of Image
Event Logs → dockerd logs (events)

Dev / QA	Administration	Support
Service UP ? Container Running? Container Parent Process Running?	Infrastructure Resources Scrutiny and Timestamp Who Executed ?	Diagnose Container Vs Image File System Change A Added C Changed D Deleted
# docker logs	# docker inspect LogPath	# docker diff <containername>
Parent process of container Environmental Variable change	Timestamp, user and application /sys/fs/cgroup/cpu/docker /sys/fs/cgroup/memory/docker Resource changes ?	File System change.

NETWORKING AS SERVICE

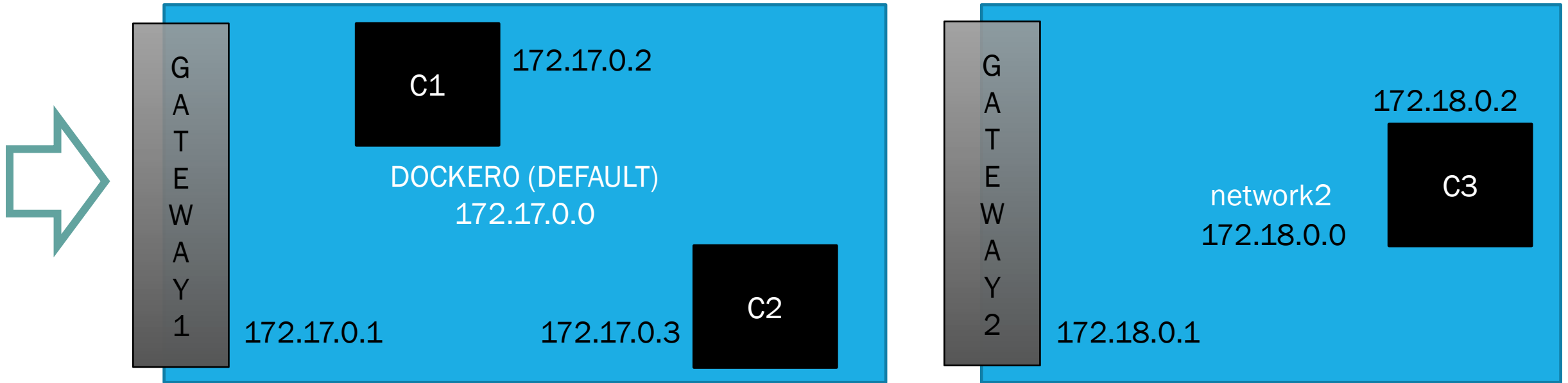
- COMMUNICATION AS SERVICE
- SUBNET OF RUNNING CONTAINERS
- IP RANGE – IPV4 /IPV6 (CIDR)
- LOCAL
 - WITHIN MACHINE
 - BRIDGE, HOST (LINUX), NONE (DEPRECATED)
- VAN
 - BETWEEN MACHINES
 - OUTSIDE MACHINES
 - OVERLAY (CLUSTER)

Container ID
Container Name
IPV4 ADDRESS
MAC ADDRESS (Router)
EndPoint Name (i-node)



172.X.Y.Z /16
256 x256 x 256 - 3
172.100.0.0 – 172.100.255.255
172.100.0.0 - Net Mask
172.100.255.255 – Net Mask
172.100.0.1 - Gateway

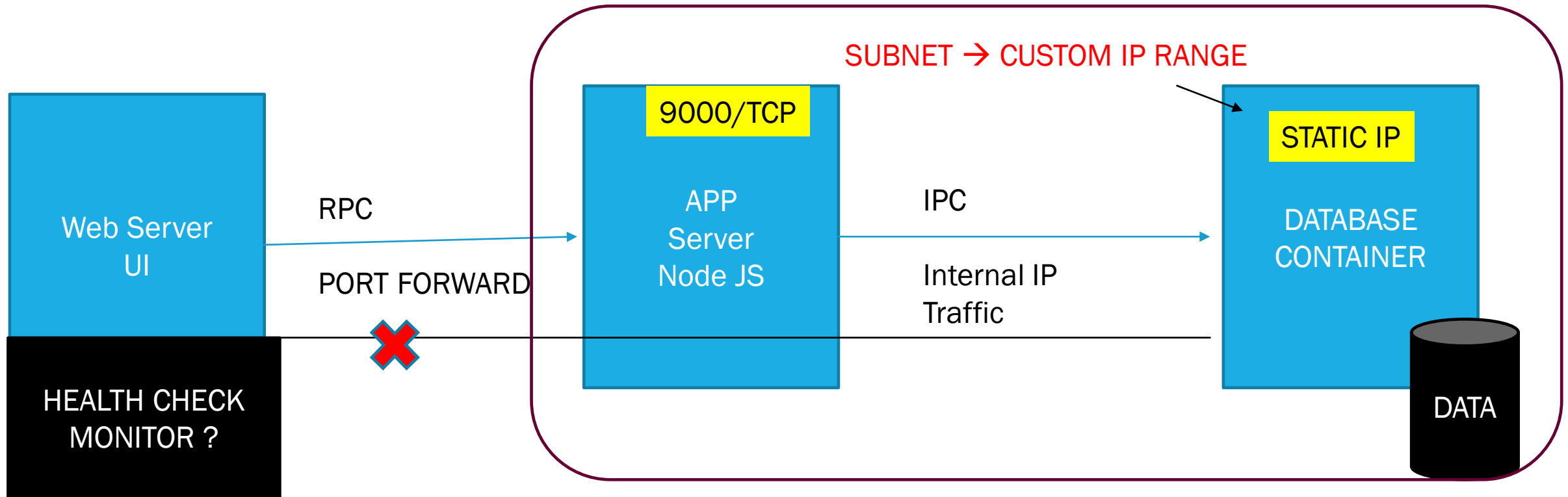
SCENARIO : 1



BOUNDARY OF RUNNING CONTAINERS LIMITED by SUBNET
GATEWAYS ARE PUBLIC
CONTAINERS WITHIN GATEWAY ARE PRIVATE

C1 → C2 (SAME SUBNET)
C1 → GATEWAY1 (PUBLIC)
C1 → GATEWAY 2 (PUBLIC)
C1 → C3 (DIFF SUBNET.. FAILURE)

USE CASE SCENARIO : 2



IMPLEMENTATION

- Step 1 → Create a Network – CIDR Subnet - Custom Subnet – 172.100.0.0/16
- Step 2 → Build Docker Image for Database Container
- Step 3 → Create a Database Container with Static IP Address , Assigning to Network (Step 1) – 172.100.100.100
- Step 4 → Verify whether Database is running ?
- Step 5 → Seed Database IP in Node JS Code
- Step 6 → Build Docker Image for App Server
- Step 7 → Assign Node JS Container to Network (Step 1) , Port Forwarding
- Step 8 → Verify Service Output
- Step 9 → Create a Service Health Container.. With –health-cmd, --health-retries – health-interval –health-timeout=1s
- Step 10 → Verify Health of Service.

EXTRACT IMAGES

Data Center Backup	Release Management
Images with Layers Full Backup – Save and Load Retains with Layers as it is saved	Single Layer Architecture # docker import Compress all Layers into one layer
Backup / CI , CD	Release / CF

EASY SETUP...

Before	During	After
Setting up Application	Troubleshooting	Uninstalling
Shell Scripting (Batch) TerraForm (Script)	CLI - docker, docker-machine	TerraForm
YAML	CLI of docker	YAML

#docker-compose

Yaml – open sources, preferred by CNCF, approved by CNCF Providers

OCI , OEL, GIT, Platforms, Languages, Docker, Kubernetes, CNI, CSI , CM , DevSecops, Troubleshooting , CSP

RULES IN YAML

- Indentation (Spacing)
- Case Sensitive
- Key: "Value"
- JSON :
- key: value → Scalar
- key : { rsa: xxx, pem: yyy } -> "|"
- key : [collection] → "-"
- Version : 3
- services

#docker-compose.yml

```
version: 3
services:
  database:
    image: newmysql
    environments:
      - MYSQL_ROOT_PASSWORD=admin
  web:
    image: httpd
    ports:
      - "8001:80"
    requests:
      memory: 200M
      cpus: 5
```

CONTAINER RUNTIME – MAKE UP FOR SERVICE

COMPOSE YAML → SERVICE MANAGEMENT


CONTAINER

Docker Image

Base Kernel
Dependency
Configuration
Boot Strapper

Dockerfile → IMAGE MANAGEMENT

CODE (PACKAGE)
→ Server Side Discovery
(Applications)



Configuration
ENV FILES
Resource
mgmt.
CPU
Memory
Swap
Memory

Set up
Communication
Network
Volume
Resource mgmt

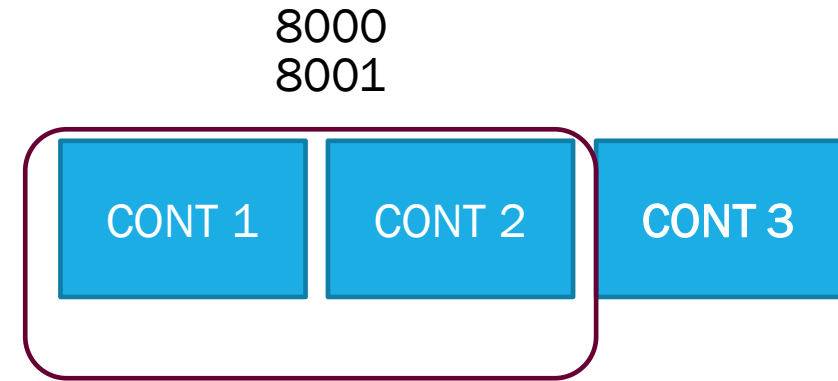
HEALTH OF
SERVICE

CONTEXT

DOCKER	K8s
MANAGE CONTAINERS – CONTAINERS ARE INDEPENDENT	MANAGE PODS – CONTAINERS ARE DEPENDENT ON PODS
CLI → Containers (docker CLI)	CLI → Manage (kubectl)
Restart Policy → NEVER (NO)	Restart Policy → ALWAYS , Self Heal (Recover, Repair)
Properties → JSON	Properties → Key Value Pair Format
Container Runtime → Docker	Any Container Runtime
Automation → YAML (Optional) , Manage CLI	CLI – 35-40% , Setup Applications (YAML)
No Scalability	Scalability - Horizontal or Vertical Scaling
Define Services as Docker Images/ Deploy as Containers	Define Services as Docker Images / Deploy them as PODS (which will internally contain containers)
Dev/QA (Development)	Operations/Administration

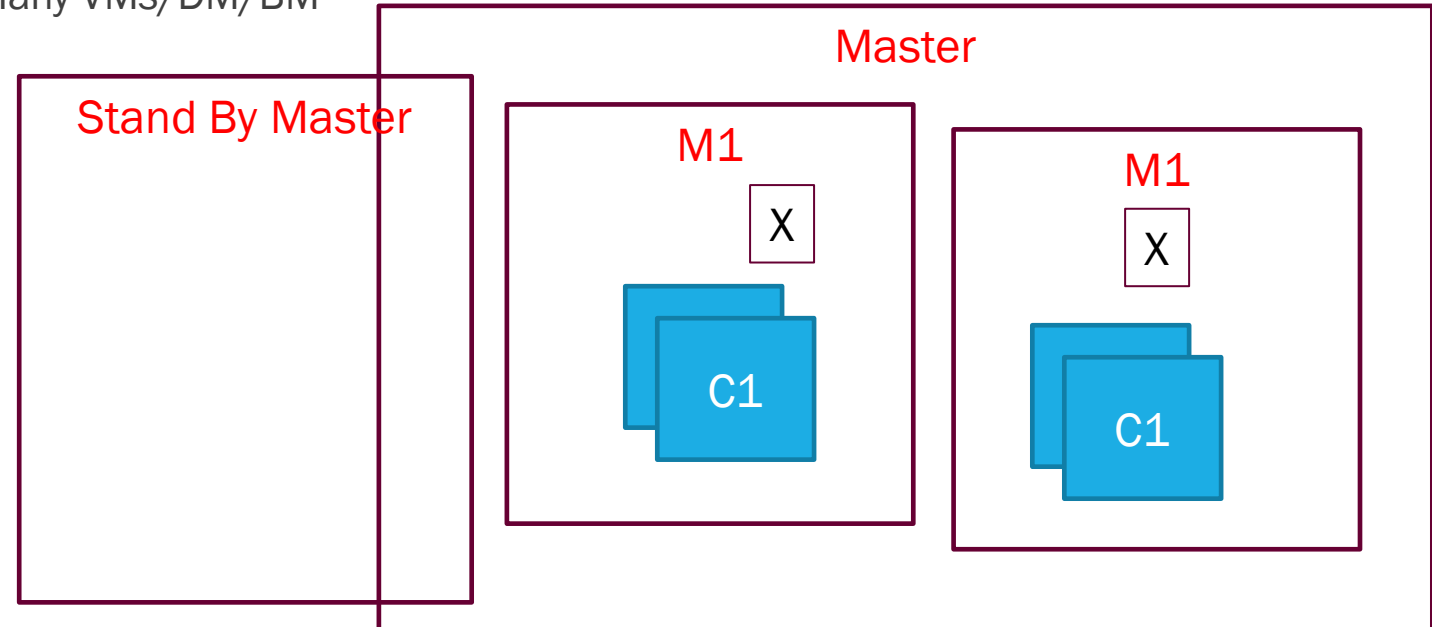
KUBERNETES

- Cluster of Container runtimes.
- Maintain High Availability
 - Services
 - Infrastructure
- Infrastructure – Replicate Infrastructure (SCALING OF VM/HOST)
- Replicate Services as Containers (Scaling of Containers) - End point of 8000 → Multiple Container Services
- Collection of Machine(s) – Networked together - Cluster
- Cluster is a collection of container runtimes .



CLUSTER

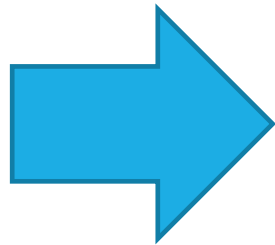
- Collection of Container runtime (dockerd) -- VM -- Many VMs/DM/BM
- Leader of Cluster -- Orchestrate
 - Monitor Utilization
 - Monitor Resources of Machines
 - Monitor Services
 - Monitor Health of Services
- Stand by Orchestrators
 - Leader Failure -- Orchestator
- Machines (node)
 - Orchestrated Node
 - Worker Node



ROLES

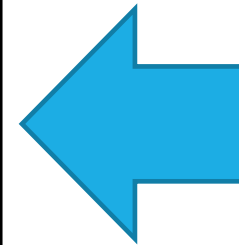
1. Master
2. How many Node Pool
3. Configuration?
4. Nodes?
5. Configuration of nodes (Reserve)
6. Availability Domain

Define Cluster



1. Services
2. How many Replicas
3. Security of Services
4. Network Policy
5. Router Policy
6. Service Exposure
7. Service Maintenance

Scalability
Services - Infrastructure

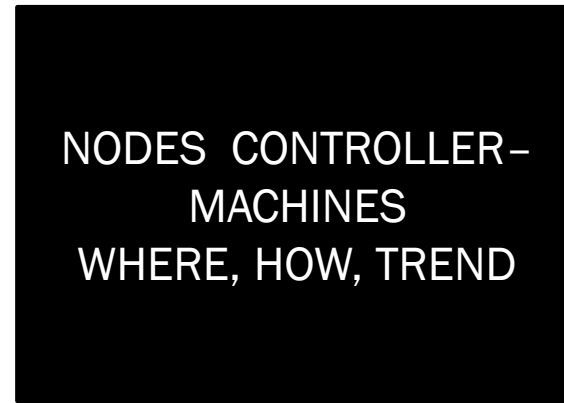


1. Define Service
2. Docker Image
3. Validate Service
4. Validate Port
5. Health of Service
6. Healthy ?

Service Definition - Docker

ORCHESTRATOR NODE

VERTICAL SCALING – SERVICE DISCOVERY /SERVICE REGISTRY - POD SCALING

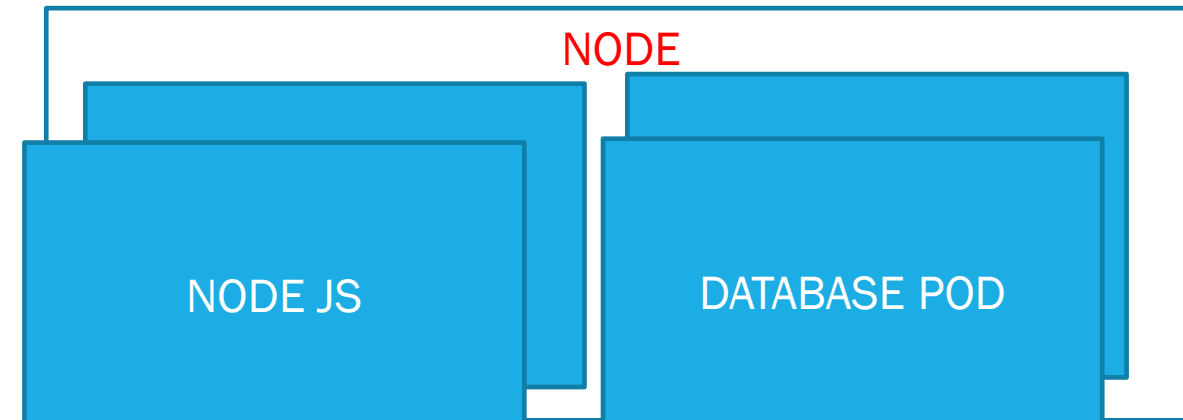


S1 - TRUE

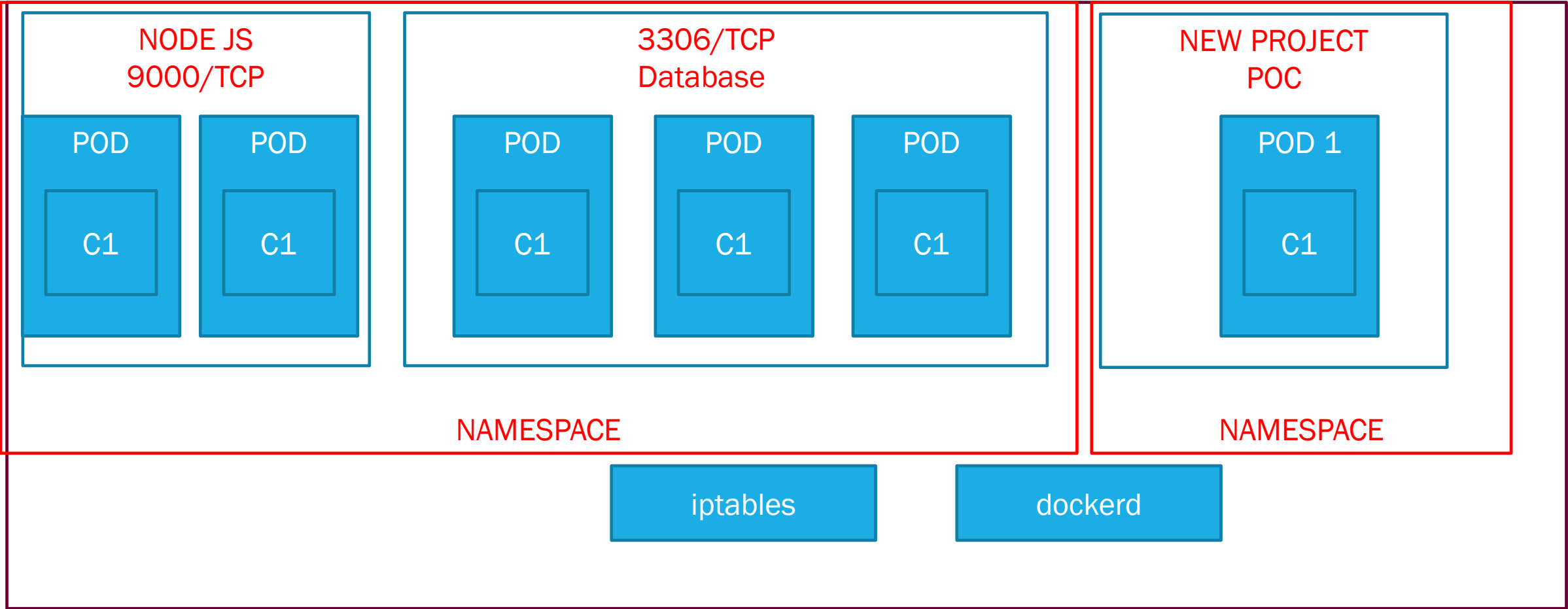
S1 – N1 – ENDPOINT (HE)
S1 – N2 – ENDPOINT (UNH)

HORIZONTAL SCALING – NODE CONTROLLER

ANYTHING RUNNING IS KUBERNETES IS POD
K8s → POD MANAGEMENT / NODE MANAGEMENT



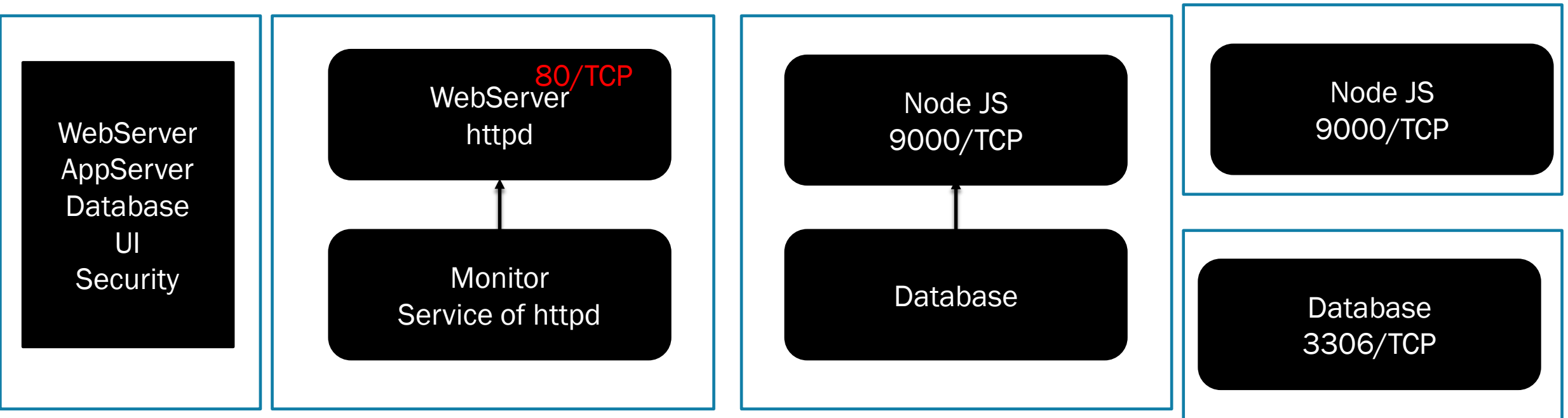
WORKER NODE



OBJECT OF ABSTRACTION

- MASTER
 - STAND BY MASTER
- NODE POOL
 - **NODES (VM)**
 - NAMESPACES
 - SERVICE (DNS ENTRY)
 - DEPLOYMENT
 - PODS
 - PODS
 - CONTAINERS
 - **IMAGE (CODE)**
 - LAYERS

UNIT OF SCALE (ABSTRACTION) - POD



SCALE A PORT (SERVICE) → MANY CONTAINERS

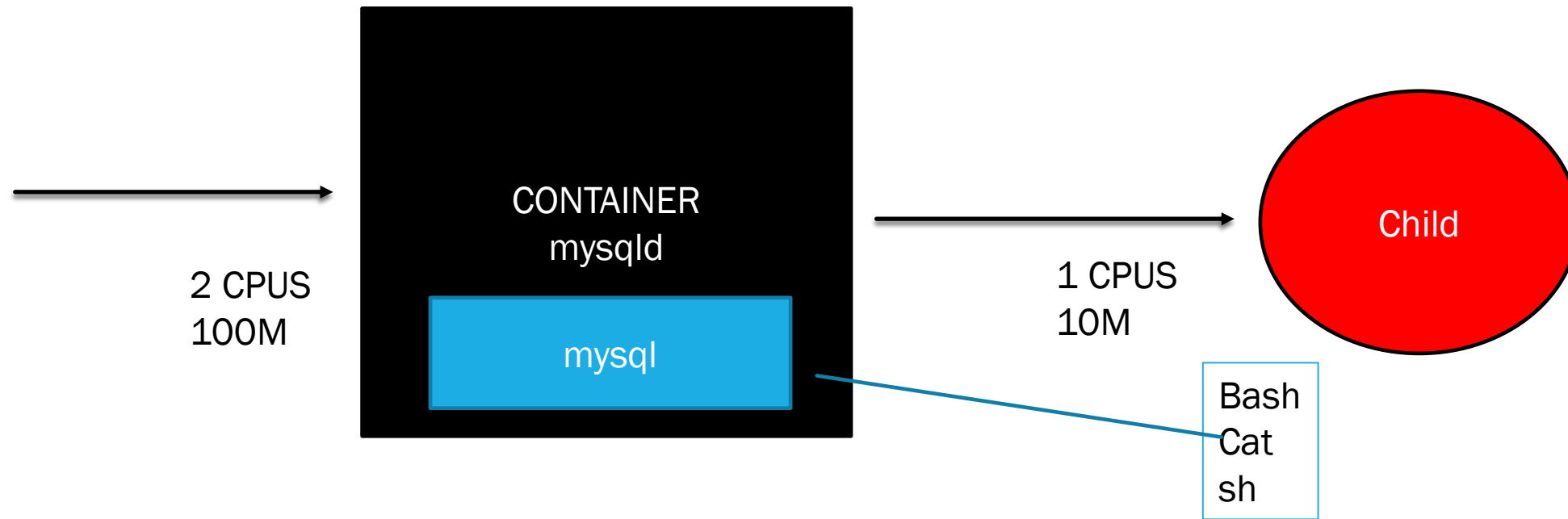
SCALE A SERVICE → ONE CONTAINER

SCALE A SERVICE → MANY PORTS – ONE OR MORE CONTAINERS

Java – Jar → Code ->>>. WebLogic (Jar) → Database → Cloud (on Prem)

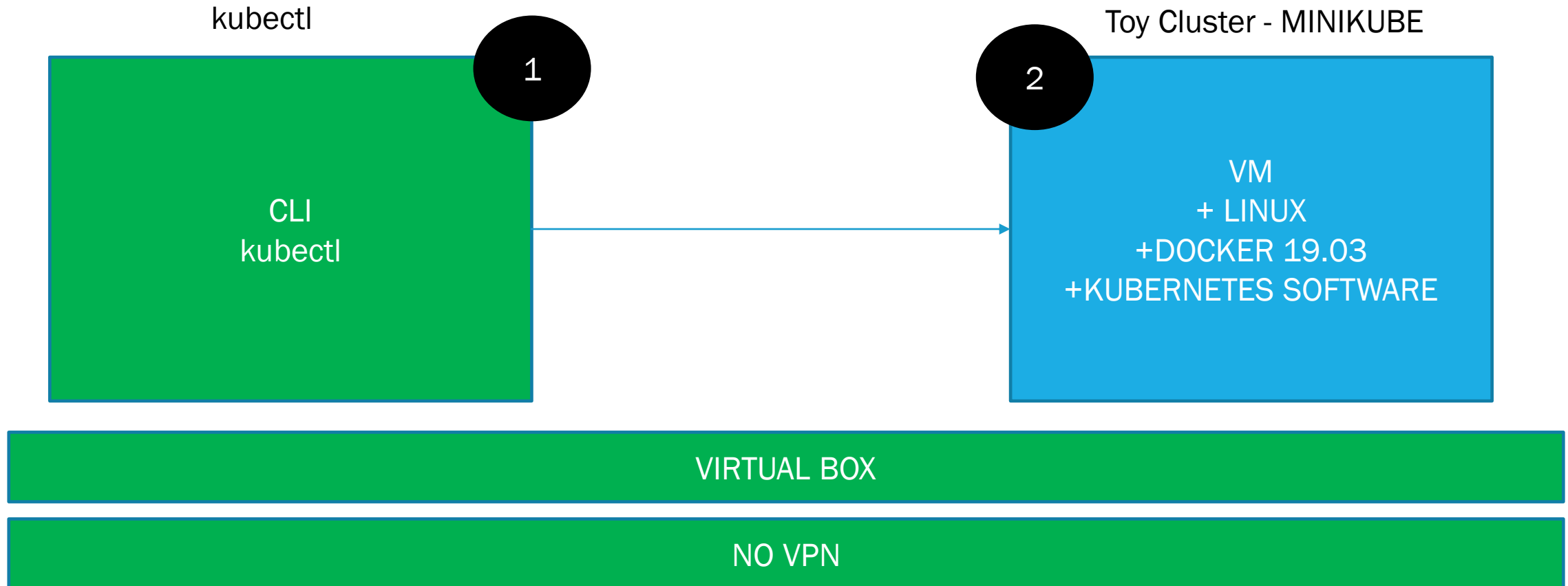
Docker images (Jar) → JRE . Database (Cache). Web logic (Cloud)

TESTING ON CONTAINER.. EXEC CHILD PROCESS



SIMPLE STEPS

```
#minikube start -cpus=2 -memory=3072 -vm-driver=virtualbox
```



DAY 4

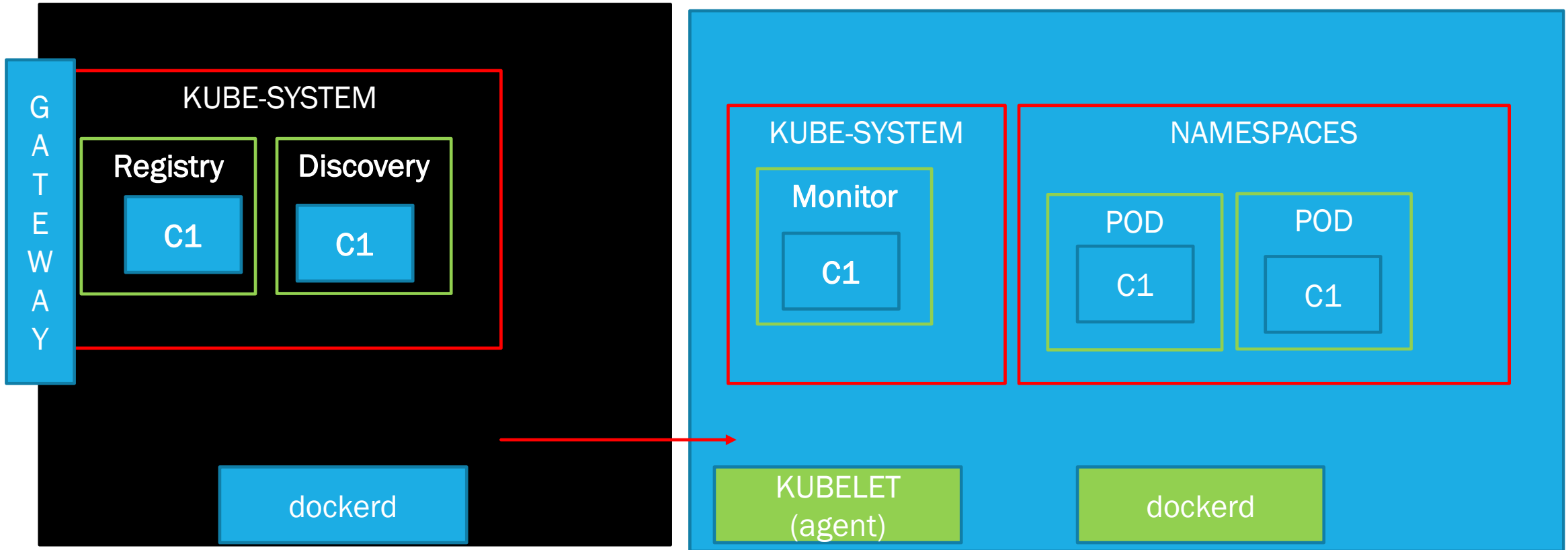
- POD Architecture
- Services Architecture
- Replication
- Communication – Discovery
- Configurations
- Service management and Scope
- Logging and Troubleshooting
- Use case – Communication

KUBERNETES ARCHITECTURE

ORCHESTRATED NODE 8443/6443/16443

HEART BEAT

WORKER NODE - 443



ORCHESTRATOR SERVICES

Gateway	Registry	Discovery	Router
KUBE DNS (Core DNS)	-ETCD key value pair registry	API SERVER (Trace where services)	DNS – Router
NGINX PLUS	Zoo Keeper (Big Data)		Nginx
JETTY	Kong (rdbms..)		VOYAGER – ENVOY
VERTIX	Eureka (no SQL)	Ribbon (Client Side)	ISTIO. LINKERD
OTD	Consul IO (no SQL)		TRAFTEIK

#KUBECTL API-RESOURCES

Version of kubectl → Server Version ?

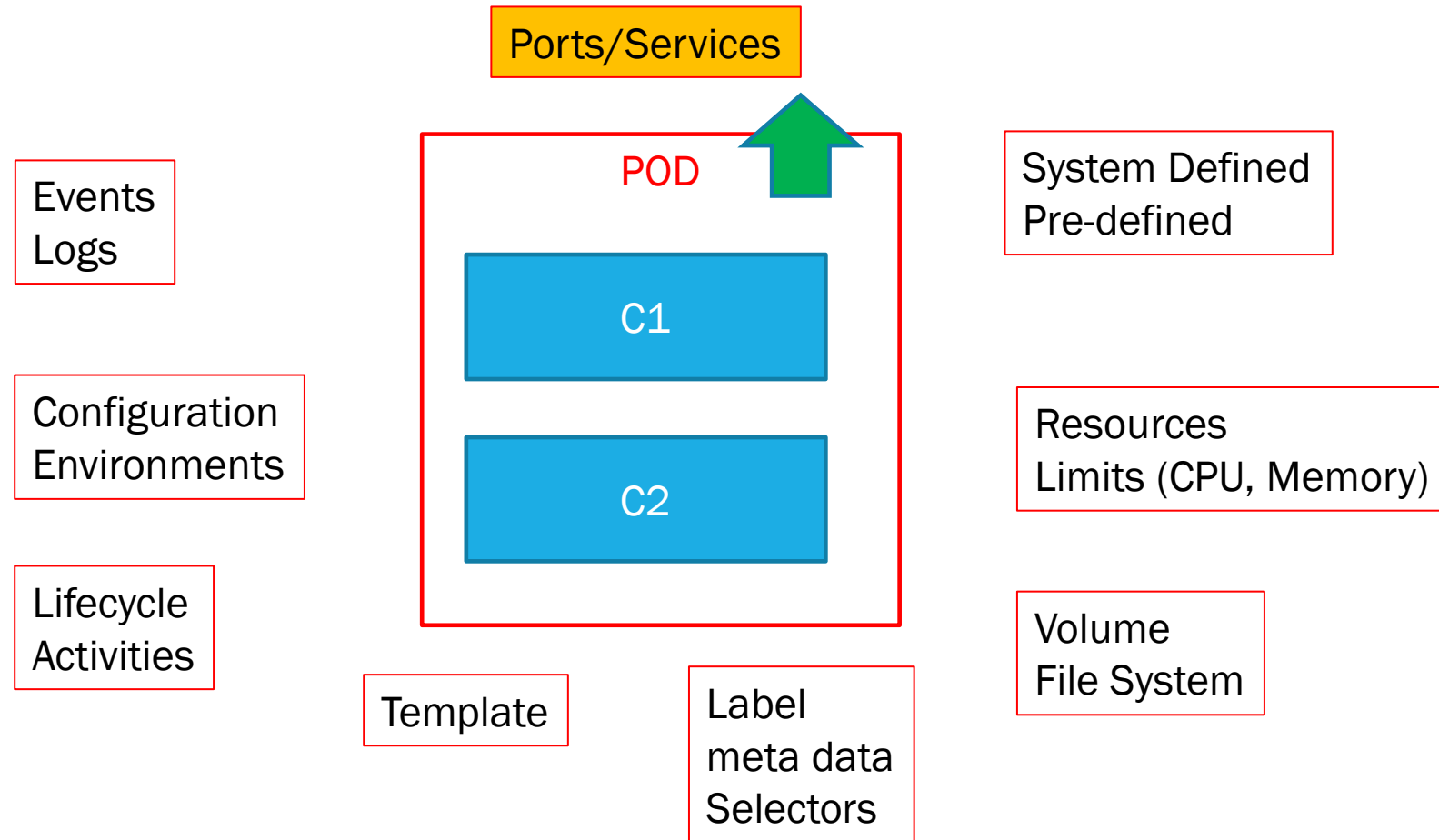
- Name of the object → CLI
- Short Description of the Object → CLI for Abbreviated object
- API Library – YAML (Object Source)
- Namespaced – True/False → CLI /YAML
- Kind → YAML

POD DEFINITION

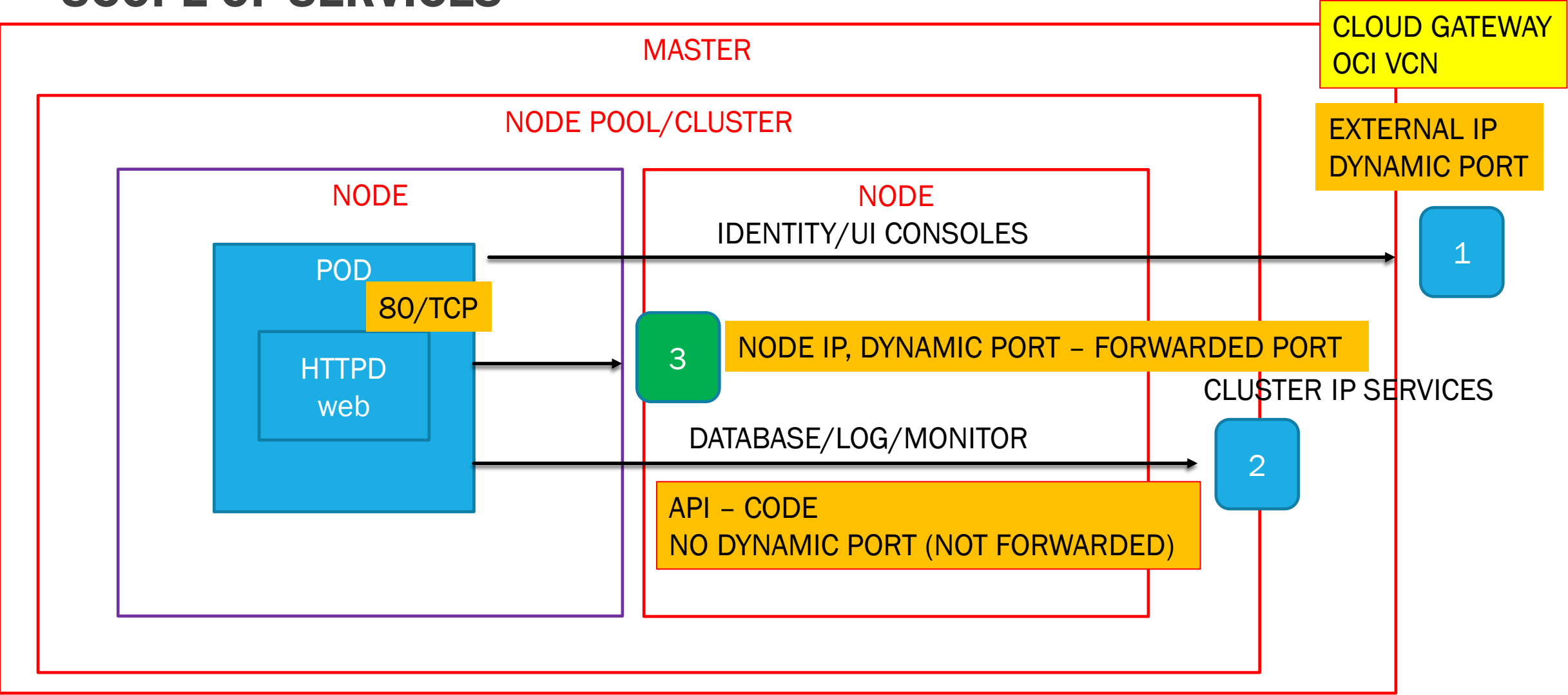
Stand Alone Pods	Deployment
Simple , 1x1 Pods , POC or testing	Replicated Pods High Availability of Service
Name of Pod, Name of Container, Name of Image	Collection of Similar pods –httpd (Service)
#kubectl run	
1 Service → 1 POD (End Point – 1 Pod)	1 Service → replicas pod (# end points...)

CLI → Create, Maintain, Delete (CRUD)
YAML → Create and Delete (Declarative)

POD ARCHITECTURE



SCOPE OF SERVICES

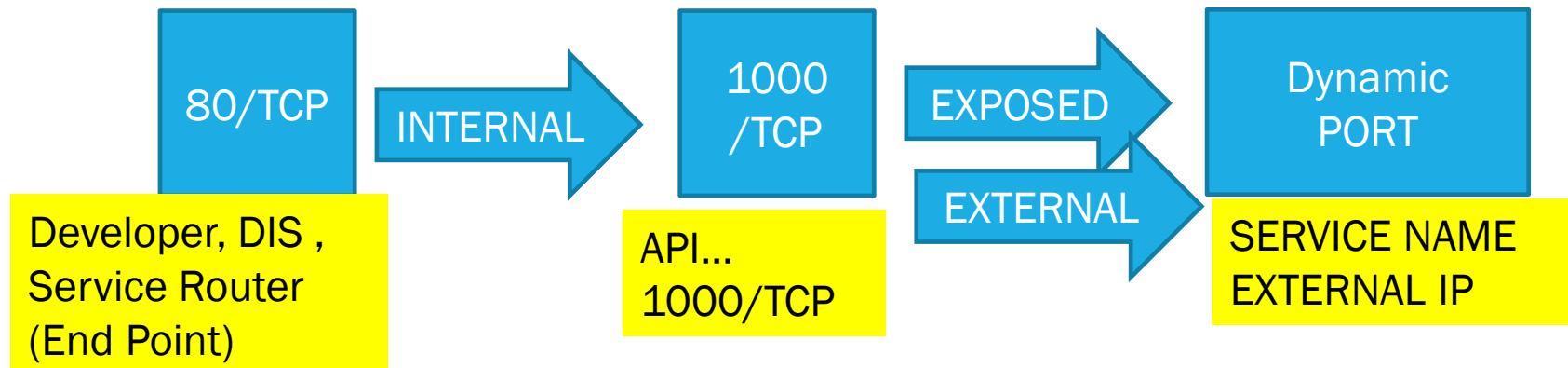


SERVICE TYPE

- SERVICES ARE PUBLIC WITHIN CLUSTER
 - LIMIT SERVICE FOR SELECTIVE (NETWORK POLICY)
- SERVICE NAME
- FORWARDED PORT*
- SERVICE DNS ENTRY (SERVICE DISCOVERY)
 - SERVICE_NAME.NAMESPACE_NAME.SVC.CLUSTER.LOCAL
- SERVICE IP (PRIMITIVE APPS)

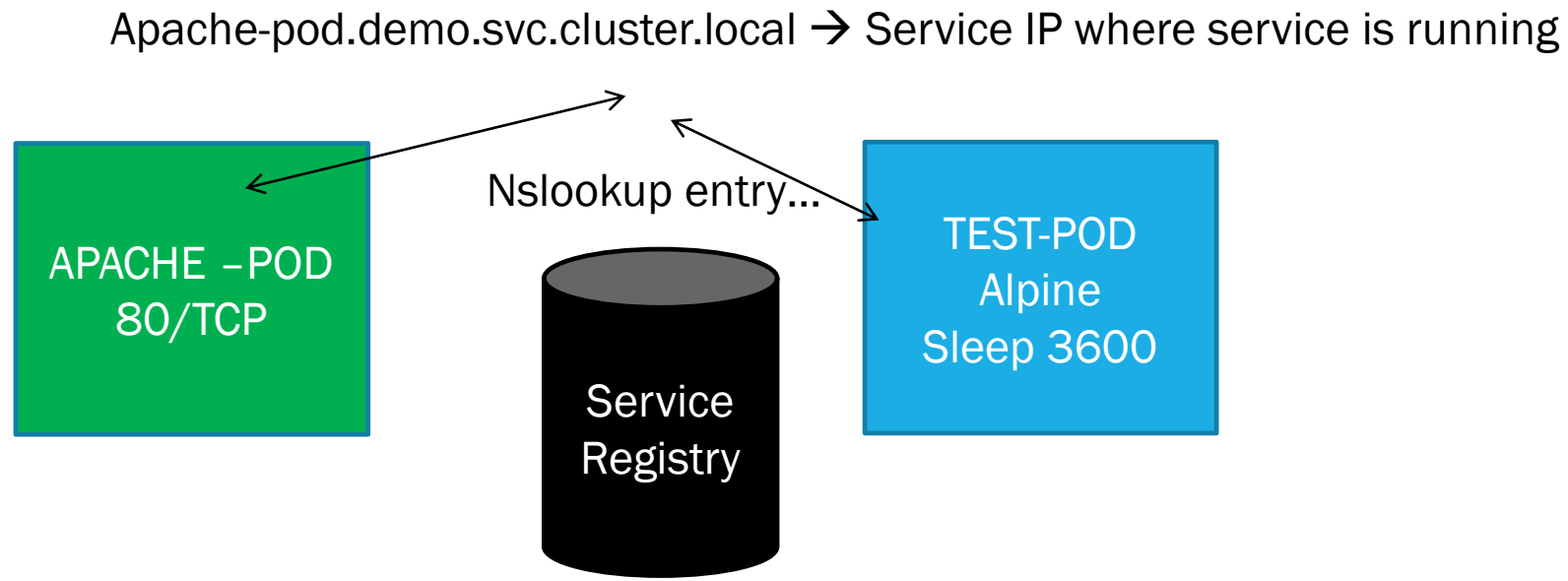
TARGET-PORT : IMAGE_PORT: 80/TCP
FORWARDED_PORT: DYNAMIC PORT
(30,000 – 32767)
MASKED_PORT : 100/TCP
(Internally 80 , Externally 100)

Alpine → #nslookup...
Sleep 3600

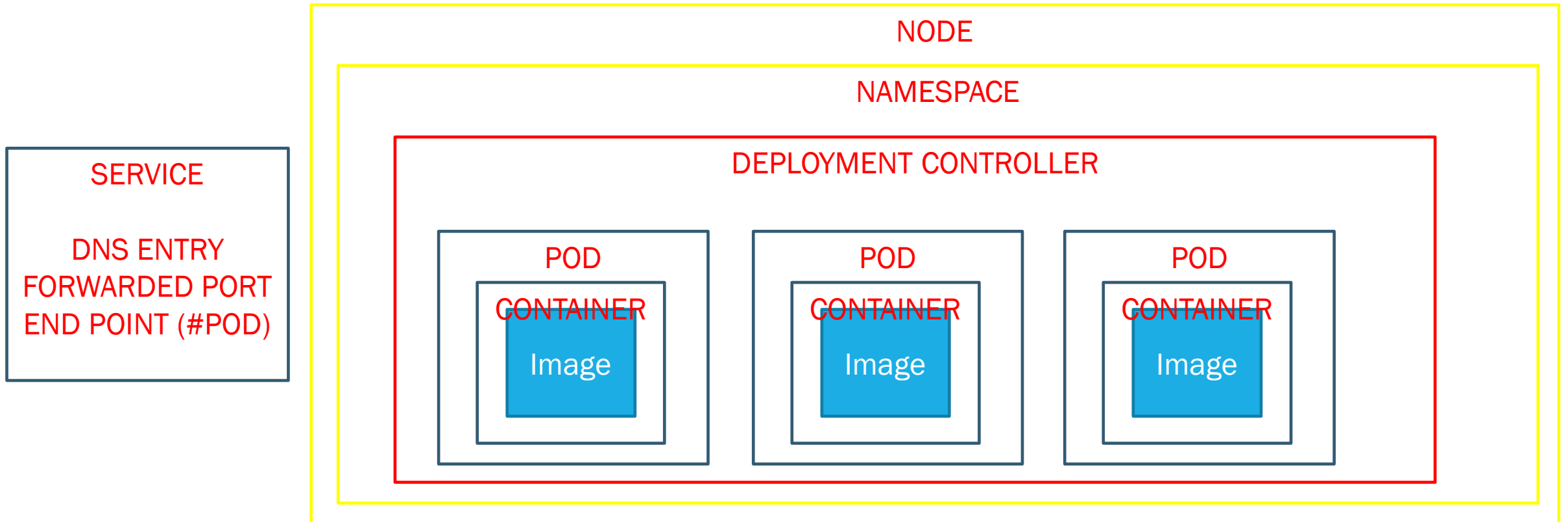


EXPOSURE TYPES – COMPARISON

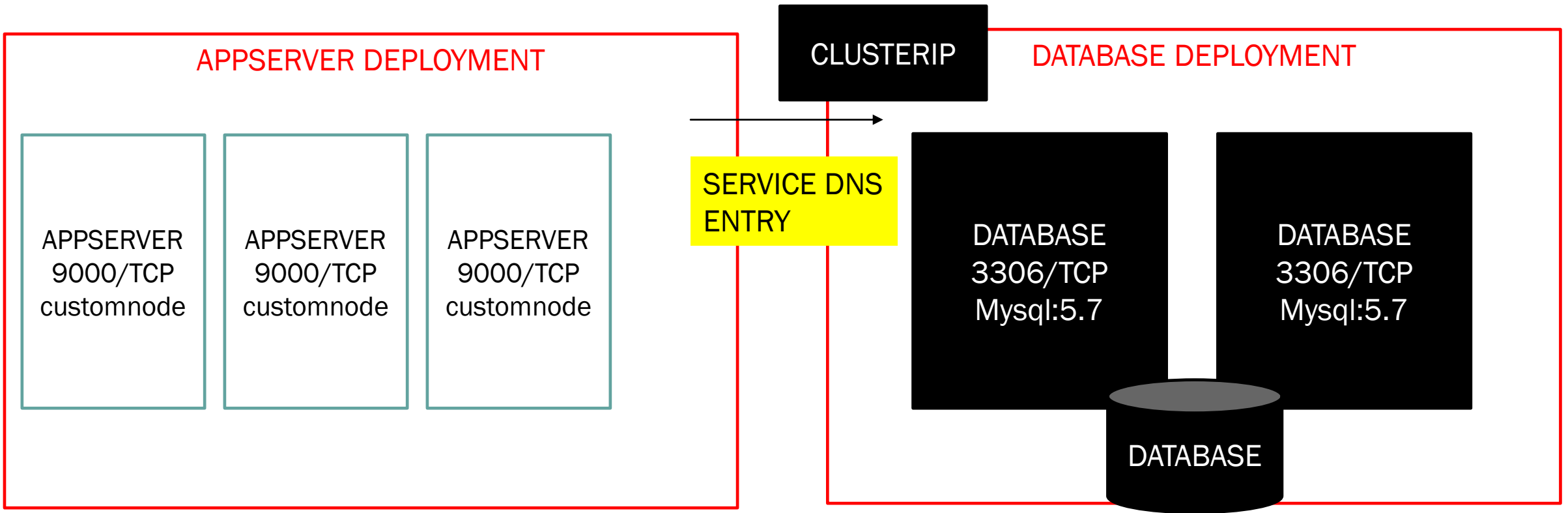
1. Internal Service	2. Cloud Exposure	3 . Node Port Exposure
Internal to Cluster (ClusterIP)	External to Gateway	External to Node (NodePort)
Service available within SSH , Within API , Within POD	Requires Cloud Creates External IP Accessible Public outside the cluster	Internal via Service IP.. Access it externally via Node IP (Curl...) Not necessary within POD.. Only for those who have access to Node IP !



ABSTRACTION– DEPLOYMENTS...



USE CASE WITH DEPLOYMENTS



IMPLEMENTATION :

- ACCESS DOCKER ENVIRONMENT
- BUILD DOCKER IMAGE FOR DATABASE
- CREATE DEPLOYMENT FOR DATABASE
- EXPOSE DATABASE AS INTERNAL SERVICE
- VERIFY DATABASE IS RUNNING WITH YOUR SCHEMAS
- SCALE DATABASE DEPLOYMENT to 2 REPLICAS
- RESERVER SERVICE DNS ENTRY FOR DATABASE
mysql.sample.svc.cluster.local

- ACCESS DOCKER ENVIRONMENT
- SEED SERVICE DNS ENTRY INTO APP CODE
- CREATE DOCKER IMAGE FOR APPSERVER
- CREATE DEPLOYMENT FOR APPSERVER
- EXPOSE APPSERVER AS EXTERNAL SERVICE
- VERIFY APPSERVER IS RUNNING
- SCALE APPSERVER DEPLOYMENT to 3 REPLICAS
- VERIFY OUTPUT

PERSISTENT VOLUME

STORAGE CLASS
READ/WRITE

VOLUME
WHERE ?
SIZE
SC
MODE:

CLAIM
LIMIT
MODE : R/W
VOLUME ?

POD
DATA OF VOLUME

POD
DATA OF VOLUME

NAMESPACE : QUOTA

- LIMIT NAMESPACE BY BUDGET
 - INFRA BUDGET – MEMORY, RESOURCE
 - COUNT – POD, SERVICES, DEPLOYMENTS..
- CROSS LIMIT → FORBIDDEN
- QUOTA : 4 → SAMPLE NS (VIOLATING QUOTA) → NEW OBJECTS.
- NEW QUOTA FOR NAMESPACE → 6 PODS (MAX):
- CREATE DEPLOYMENT FOR 2 PODS REPLICAS ?

TROUBLESHOOT KUBERNETES !

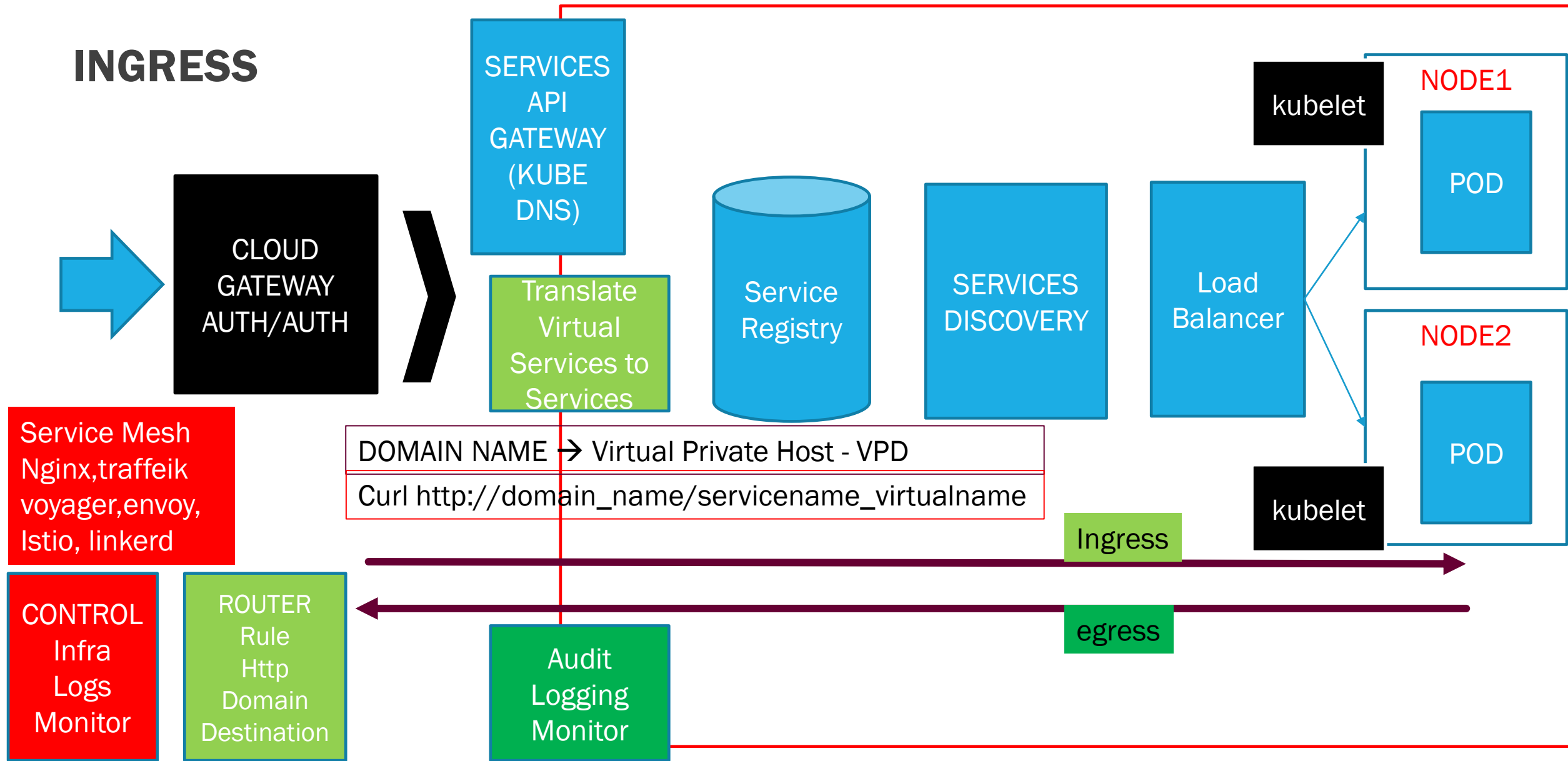
Command Issued – No Syntax Error Object is not created	Command Issued – Object Created, Object is not in Desirable State	Command Issued , Object Created, Object Desired State, Service not running as per desired State
Quota Exceeded for NS Nodes do not have Taint.. Locked Nodes for Deployments. Quota Exceeded for Cluster	Object Created. ImagePullBackOff Error → Network ,Cannot connect to Image Repository. CrashLoopBackoff → Restart Service ? (BootStrapping Issue)→ Docker Image ? ErrImagePull → Invalid Image in Repository (Check name of Image)	Service Issue Resource Issue Code Issue
# kubectl get events (Event Log)	#kubectl describe <object>	#kubectl logs <object>



DAY 5

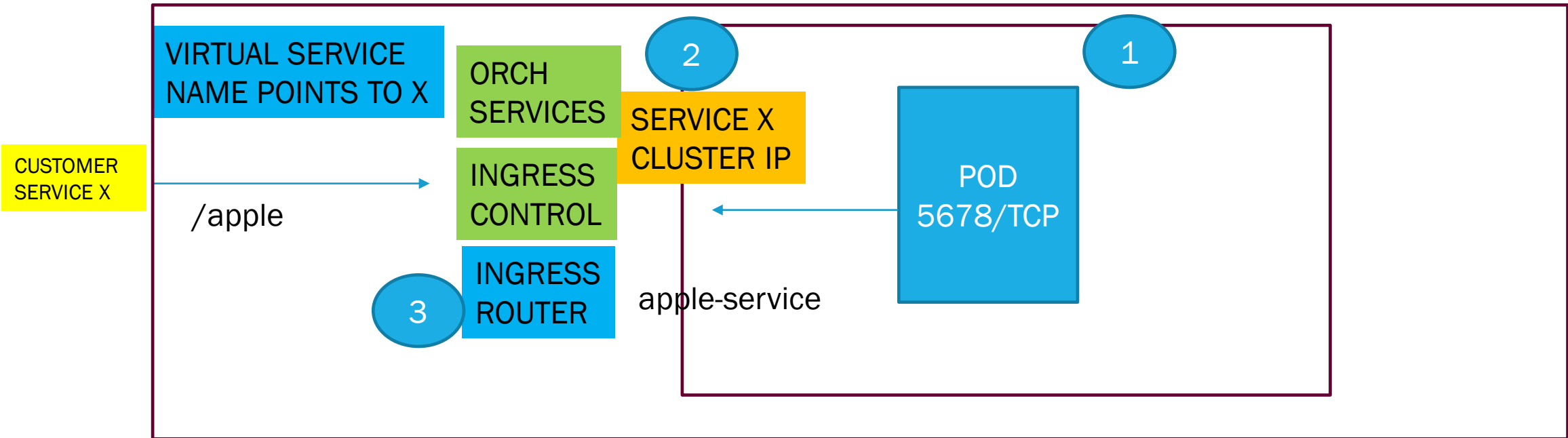
- Ingress as domain
- Bringing Automation
- Container – Automation
- Build
- Deploy
- Notify
- Package

INGRESS



INGRESS DEMONSTRATION

Green Color → ORCHESTRATOR (ADMIN)
DARK BLUE → OPERATIONS (ROUTER)

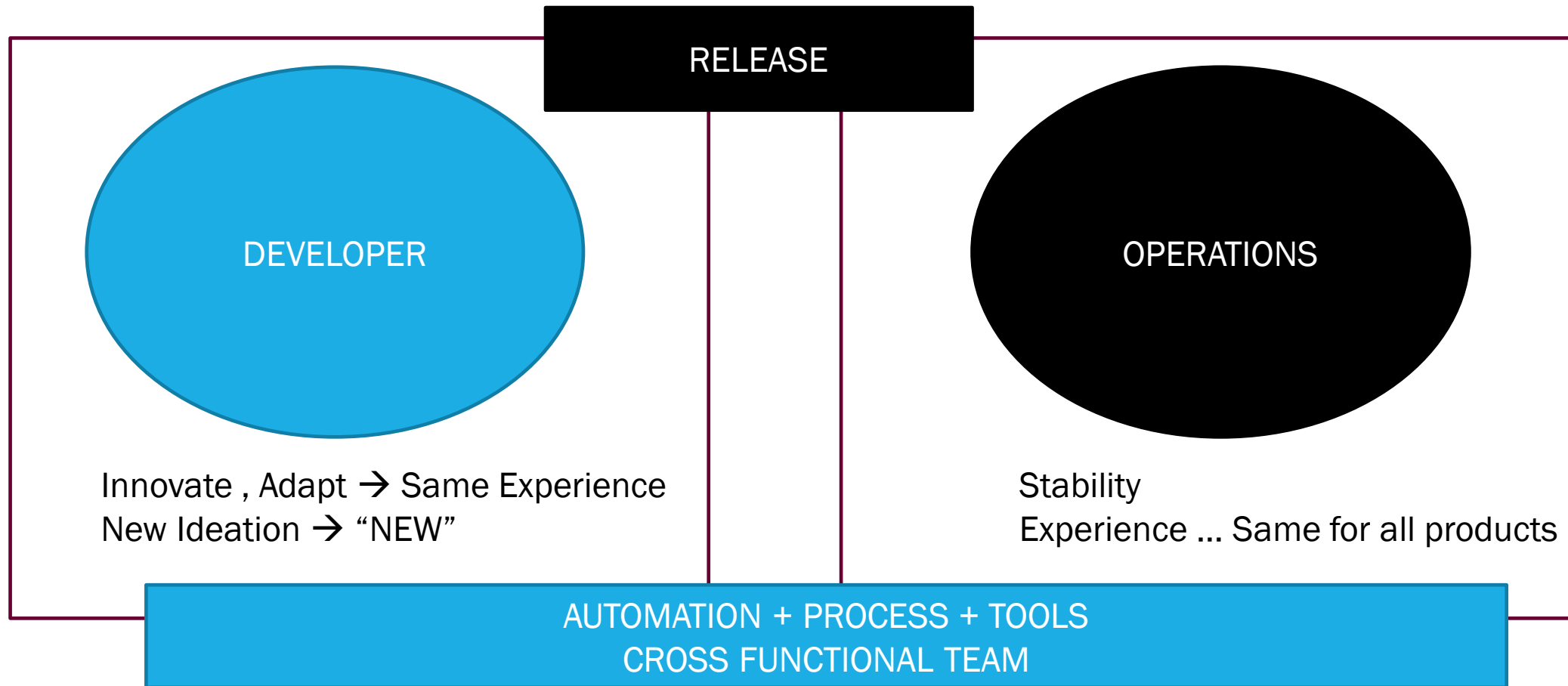




ACCOUNTS PRE_REQUISITE

- WERCKER ACCOUNT → app.wercker.com
- GITHUB → github.com
- SLACK NOTIFICATION → app.slack.com
- Cloud Account → Oracle Cloud Subscription

DEVOPS REDEFINED...



DEVOPS PROCESS → ORACLE DEVCS

1 PLATFORM FOR AUTOMATION → YAML

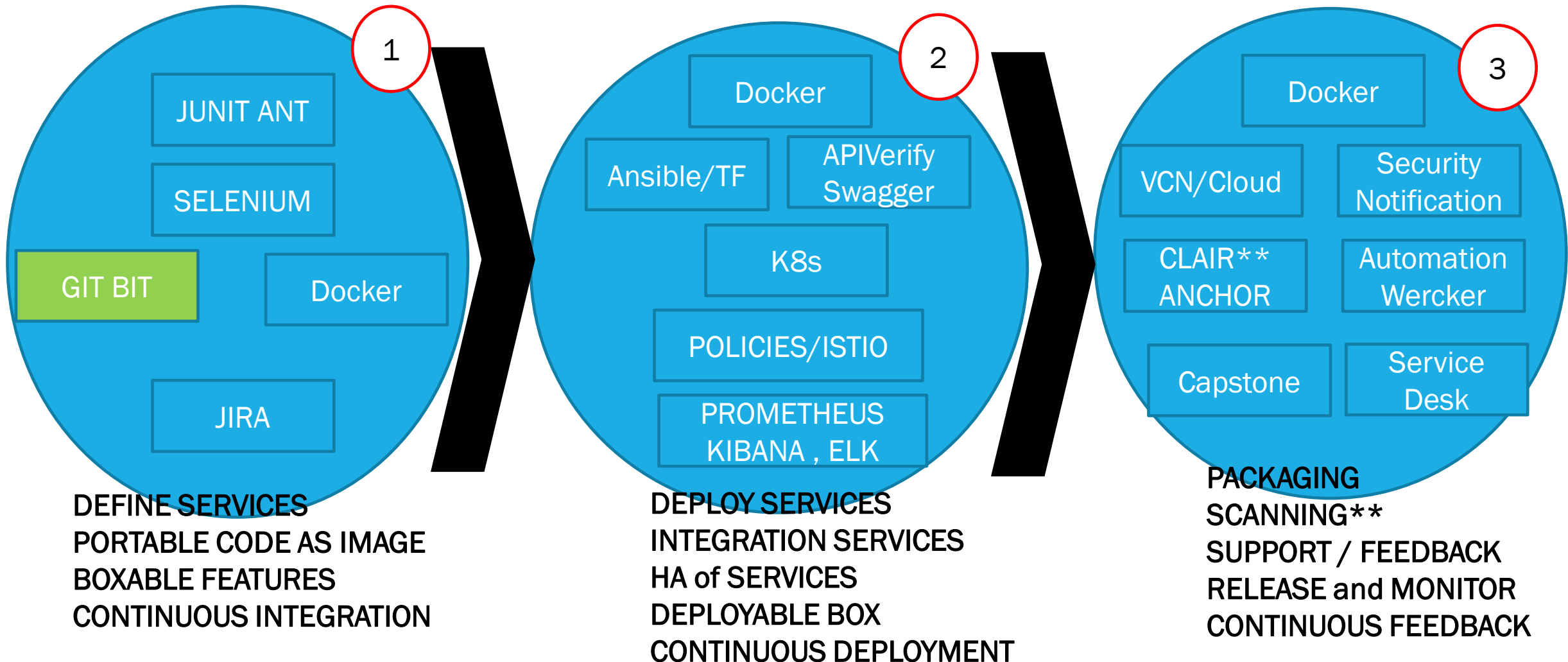
TOOL → CI , CD , CF ? – FULLY AUTOMATED

ARCHITECTURE – SOA /MSA / MONO – EASY SUPPORT

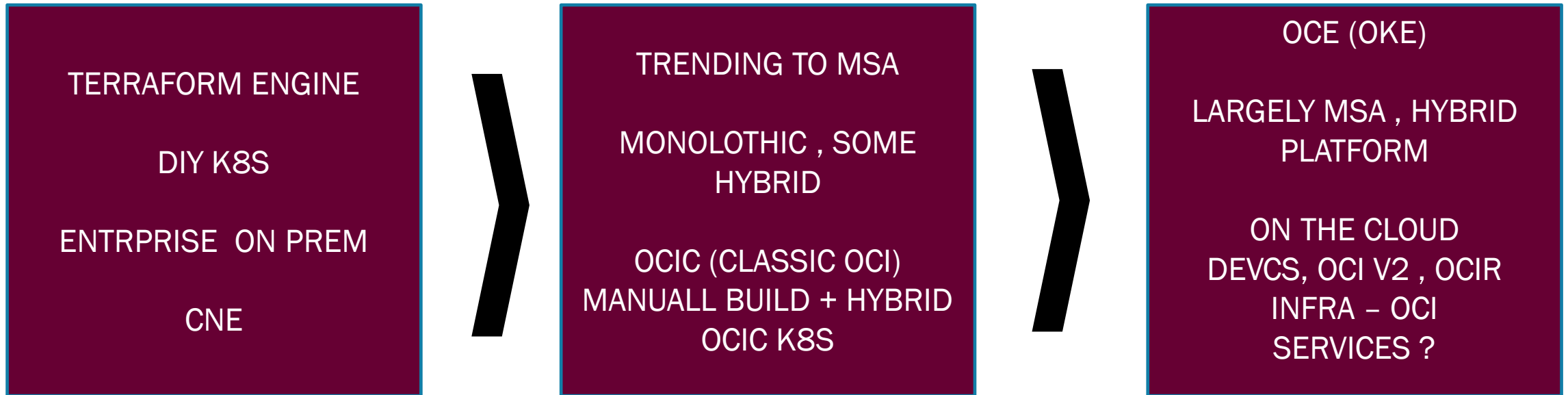
OPEN SOURCE - AUTOMATE SECTIONS / CUSTOMIZE, CLOUD FRIENDLY

CONTAINERISED - DOCKER FRIENDLY → WERCKER

VM BASED → JENKINS (CI , PARTIAL CD)

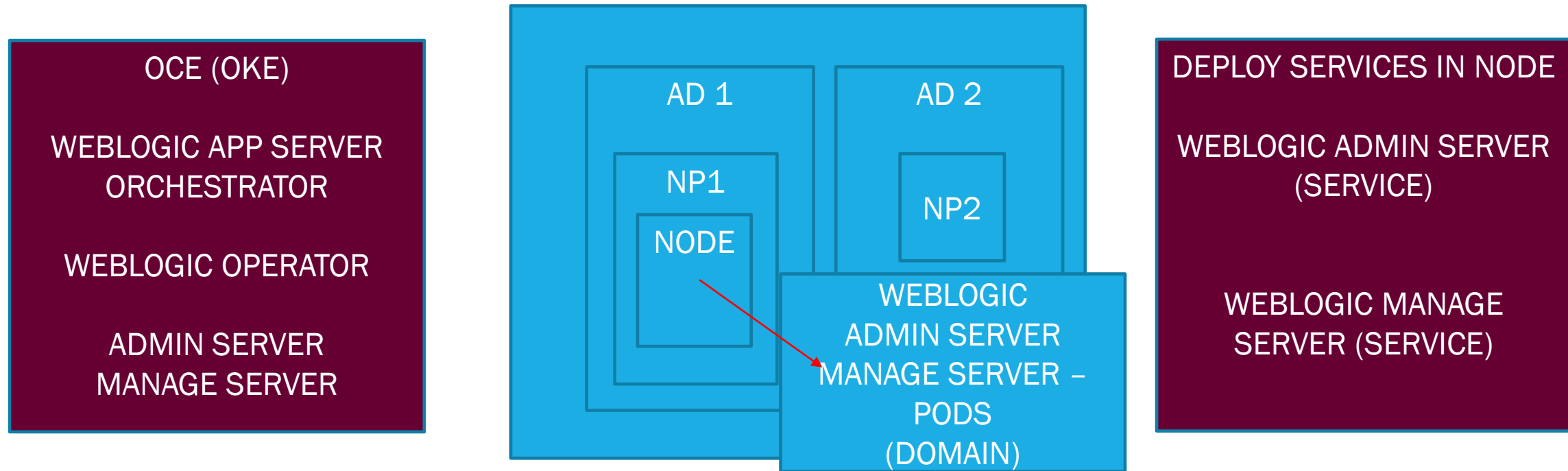


KUBERNETES... K8S

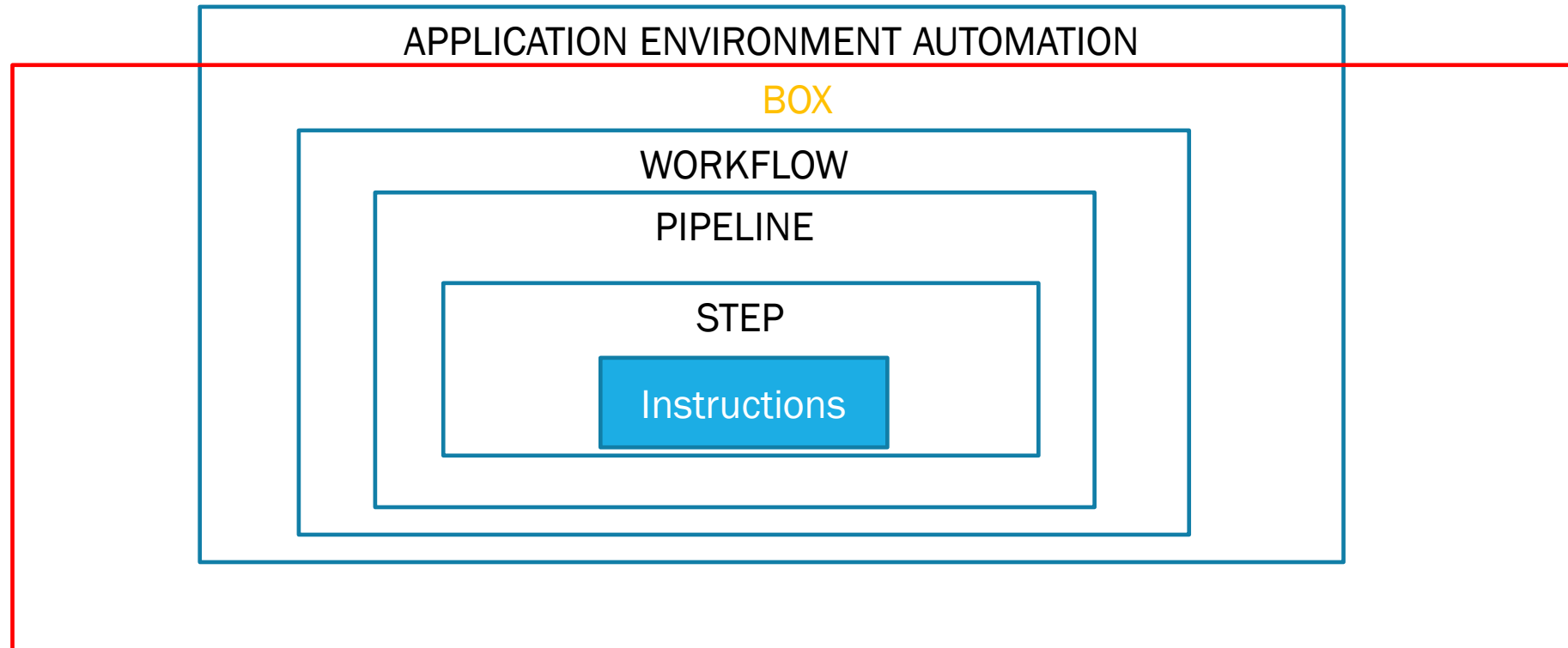


KUBERNETES WEB-LOGIC

LIFT AND SHIFT → DUMP IN CLOUD
OKE ?



WERCKER ARCHITECTURE



Wercker → Box is a CONTAINER → WORKFLOWS, PIPELINES AND STEPS
Box can be localized at Pipeline or can global at Workflow level or application level

STEP ?

- Git pull
- Docker-build
- Docker-push
- Docker-run
- Script
- Install-binaries
- Notify
- Kubectl ?
- Terraform script

Steps to be predefined..

Internal-steps:

internal/docker-push:

internal/docker-run:

internal/docker-build:

Scripts:

-- execute any command in box

-- execute any script in box

After-steps:

notification:

Step registry ->

Kubectl

Terraform

Install/packages

Minimum mandates:

Code → JAR/Packaged

Dockerfile

Environments

wercker.yml

WERCKER.YML

- PIPELINE → build (CI) → Root Process
- PIPELINE → deploy (CD)
 - Indent under pipeline as Steps (Step Registry)
 - Process
 - Linear Collection of Steps
- PIPELINE (2)
 - STEPS
- CHAIN PIPELINES (WORKFLOW)
- RUN PIPELINE

REPOSITORY CREDENTIALS
-- ENVIRONMENT VARIABLES
FOR WORKFLOW
-- ENVIRONMENTAL VARIABLES
FOR PIPELINE
-- ENVIRONMENTAL
VARIABLES FOR STEP

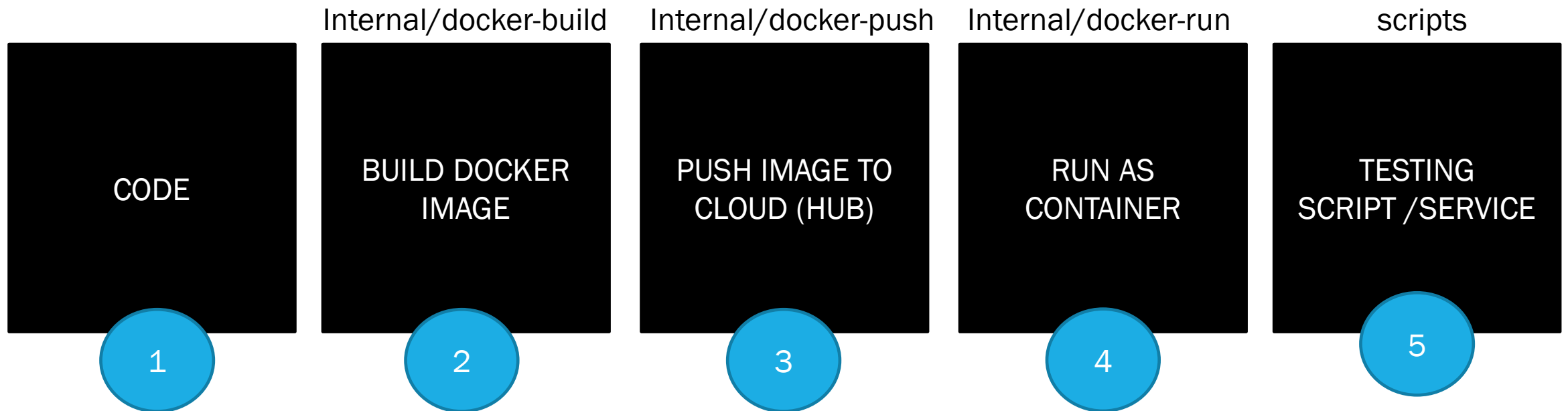
RUNTIME
→ BOX (dockerd)
→ Connection to Cloud
Repository
→ Logs



INTERACT WITH WERCKER

- WERCKER GUI
- WERCKER CLI
- HUB.DOCKER.COM
- WERCKER API (SCRIPT)
- ESSENTIAL → STEP REGISTRY

USE CASE: 1 (DEMONSTRATING CI) – WRITING YML



ENVIRONMENTS AS CAPITALS

USERNAME → HUB_USERNAME
PASSWORD → HUB_PASSWORD
REPO NAME → HUB_USERNAME/IMAGE_NAME
TAGNAME → IMAGE_TAGNAME
CONTAINER_NAME → NAME OF CONTAINER
IMAGE_BASE → Dockerfile build Image

FILES:
CODE FILES
DOCKERFILE
WERCKER.YML

STEPS TO IMPLEMENT

CODE
In
GIT
(Dockerfile,
Wercker.yml)

Wercker & GIT

Set up
ENVIRONMENTS

Verify Workflow
(Check for build)

Trigger RUN
Wercker.yml

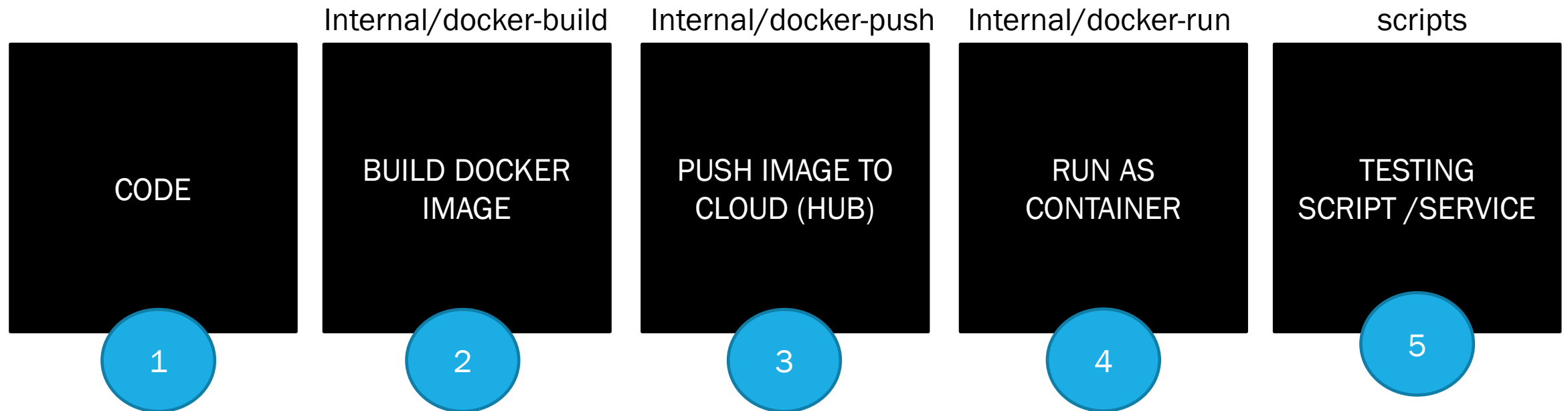
ROLLOUT..

- VERTICAL SCALING – DEPLOYMENT
- SPIN UP MORE PODS – OLD IMAGE PODS,
- SECTION OF PODS → TERMINATE , NEW PODS WITH NEW IMAGE. (PERCENTAGE OF PODS)
- PERCENTAGE _ MINIMUM
- ROLLOUT STRATEGY
 - PODS SHOULD GO THROUGH CHANGE
 - PODS SHOULD BE SPINNED MORE
- UNDO ROLLOUTS, GO TO VERSION



WERCKER – USE CASE

USE CASE: 2 (DEMONSTRATING CI) – WRITING YML FOR OCI



ENVIRONMENTS AS CAPITALS

USERNAME → TENACNY/OCI_USERNAME

PASSWORD → AUTHTOKEN

REPONAME → REPOURL/TENANCY_NAME/IMAGE_NAME

TAGNAME → IMAGE_TAGNAME

CONTAINER_NAME → NAME OF CONTAINER

IMAGE_BASE → Dockerfile build Image

FILES:

CODE FILES

DOCKERFILE

WERCKER.YML

STEPS TO IMPLEMENT FOR OCI

CODE
In
GIT
(Dockerfile,
Wercker.yml)

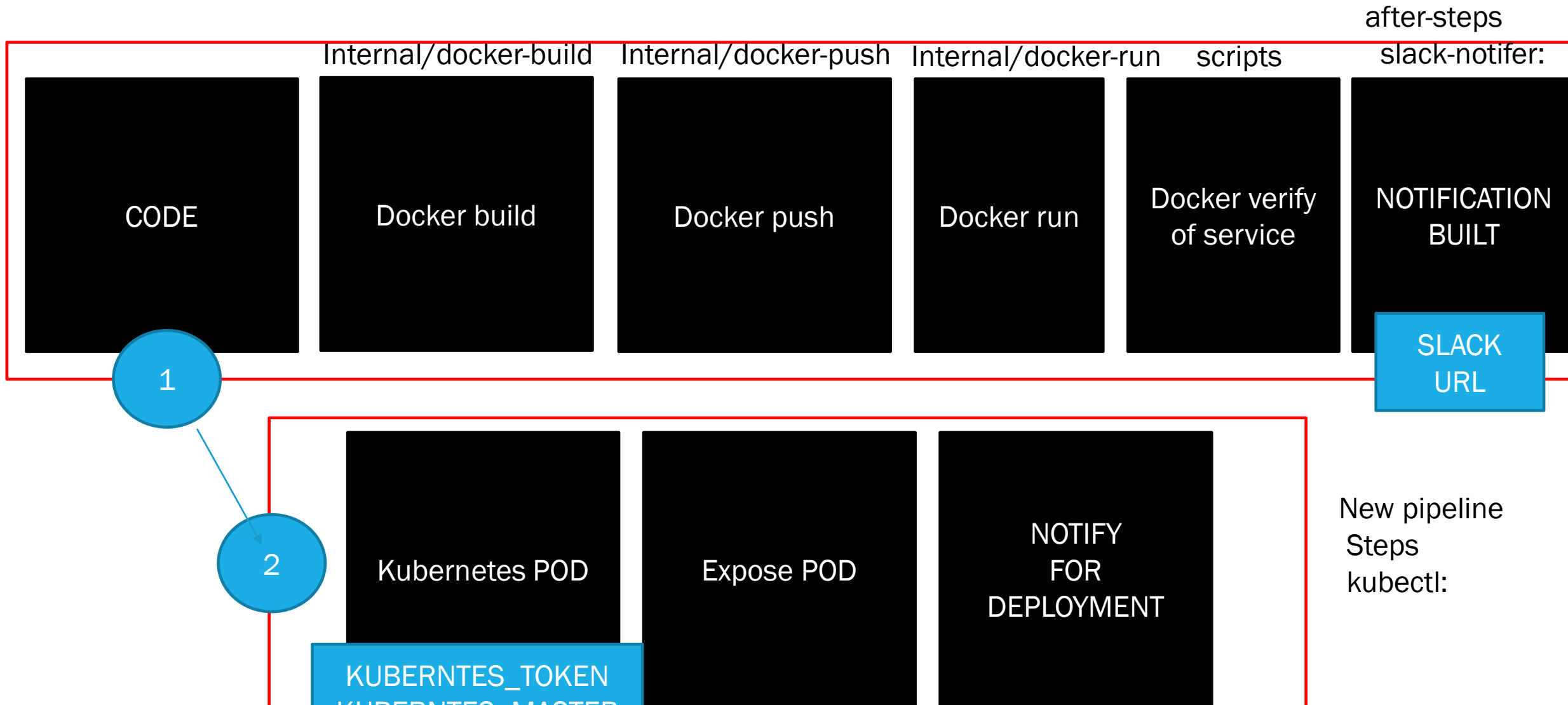
Wercker & GIT

Set up
ENVIRONMENTS

Verify Workflow
(Check for build)

Trigger RUN
Wercker.yml

USE CASE : 3 (DEPLOYMENT)



ISSUE 1 : AUTHENTICATION TO NAMESPACE IN OKE ?

- HUMAN USER (user01) → RBAC (Role Base Access Control)
- API (APPLICATION – WERCKER) → AUTHENTICATION to DEPLOY in NAMESPACE (Create, delete, Update)
 - SERVICE ACCOUNT → AUTHENTICATION API – AUTHENTICATING APPLICATIONS (API) - JWT TOKEN (JSON WEB TOKEN)
 - KUBERNETES_MASTER ?

ISSUE 2 : SLACK NOTIFICATION

- WERCKER → SLACK
- SLACK URL → TOKEN OF IDENTITY:
 - SLACK ACCOUNT
 - CHANNEL NOTIFY
- JSON URL (JSON WEB TOKEN – URL)

SLACK APPLICATION

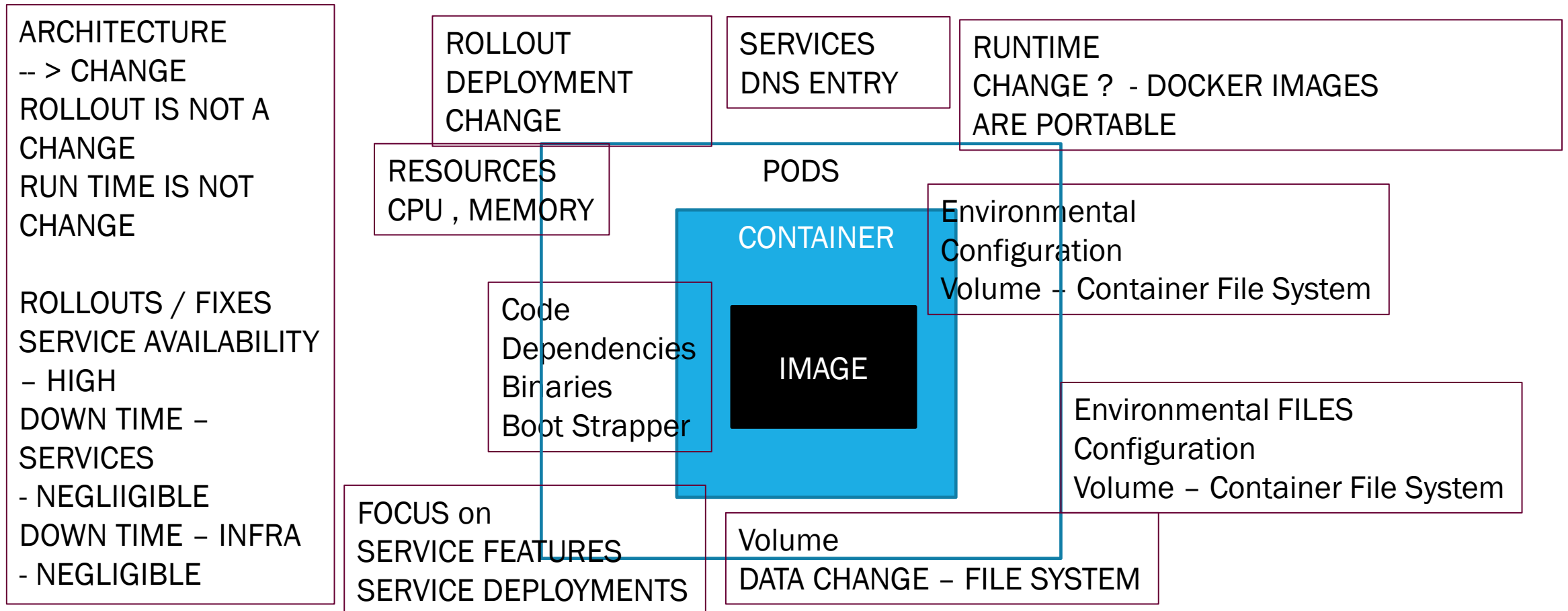
Incoming WebHooks – Authorizing Slack to make sure it can accept Wercker to post in Slack via The SLACK TOKEN

RECORD SLACK_URL FOR AUTOMATION

INTEGRATION... USE CASE DEPLOYMENT

- SOURCE CODE
- DOCKERFILE
- WERCKER.YML
- YML for Kubernetes Object
- REPO NAME
- USERNAME
- PASSWORD -AUTH TOKEN
- IMAGE_BASE
- CNT_NAME
- SLACK_URL
- KUBERNETES_TOKEN
- KUBERNETES_MASTER

CHANGE ?



DATABASE APPLICATION CONTAINER

