Welcome to IBM WebSphere Hybrid Edition

Introduction

Lets here about your,

- Name,
- Where you're from,
- Your IT Experience Background,
- Why you’re here.

Break Timings

- 11:00 AM IST – Morning Break (20 Mins)
- 13:00 PM IST - Lunch Break (60 Mins)
- 15:30 PM IST – Evening Break (20 Mins)

Some terms we use

- Theory
- Exercise
- Demonstration

Project 1

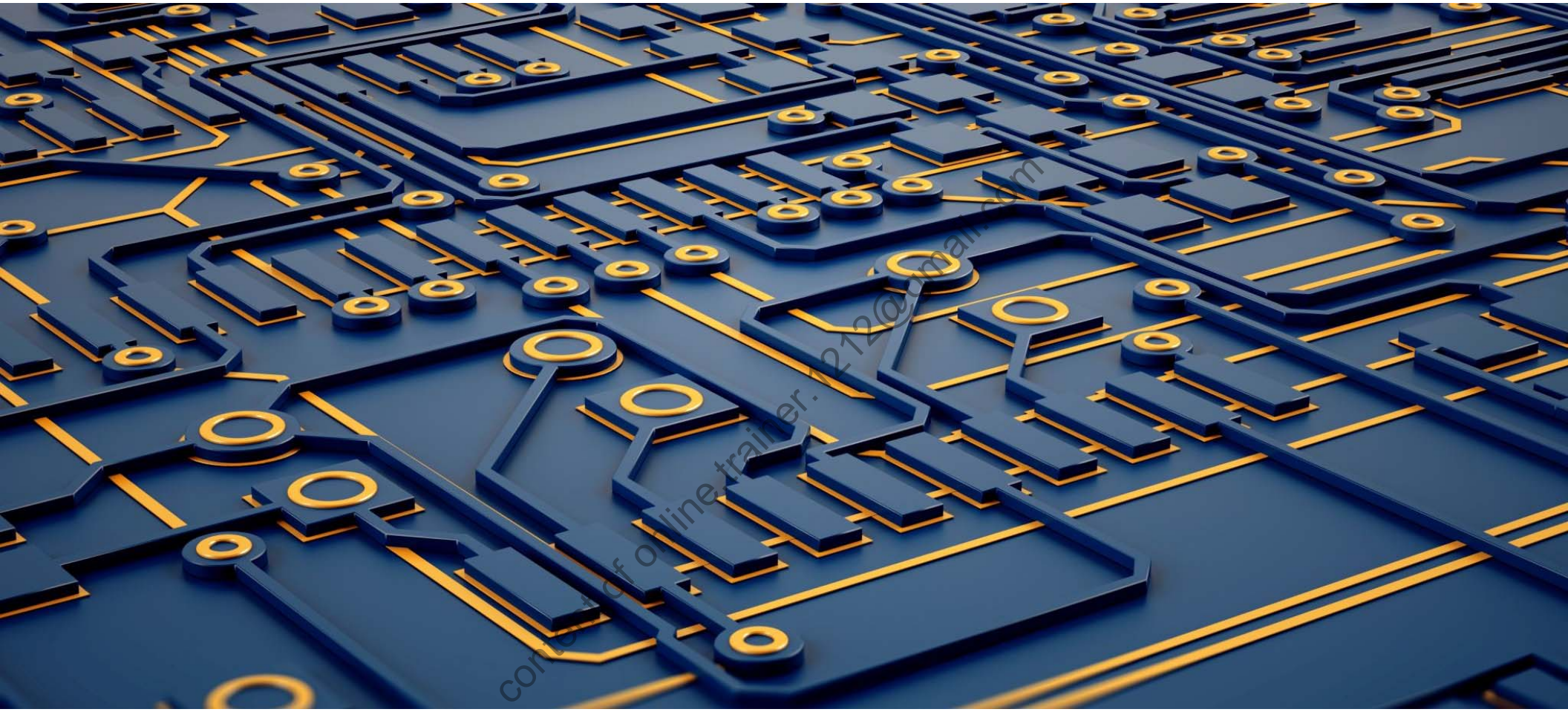
Project 1: Design & Architect A Cloud Native Solution

Sub-Projects

1. Understand the main elements of a cloud native solution
2. Design a Microservices architecture
3. Explain Containers and Container Orchestration
4. Understand the Cloud Native Reference Architecture
5. Exercise- Container Engine Labs Docker vs CRIO
6. Exercise- Container Orchestration -Kubernetes

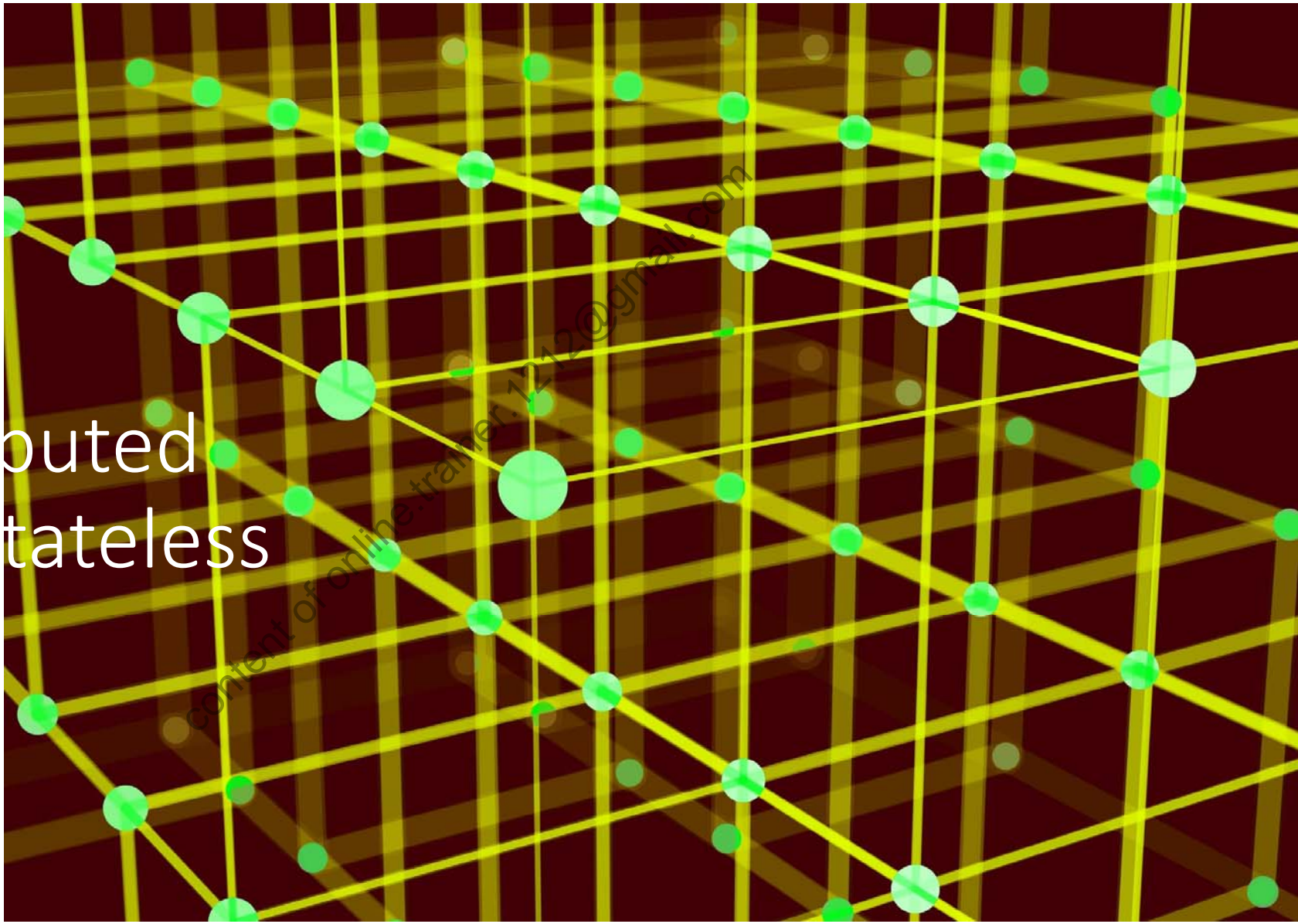
Sub Project 1.1

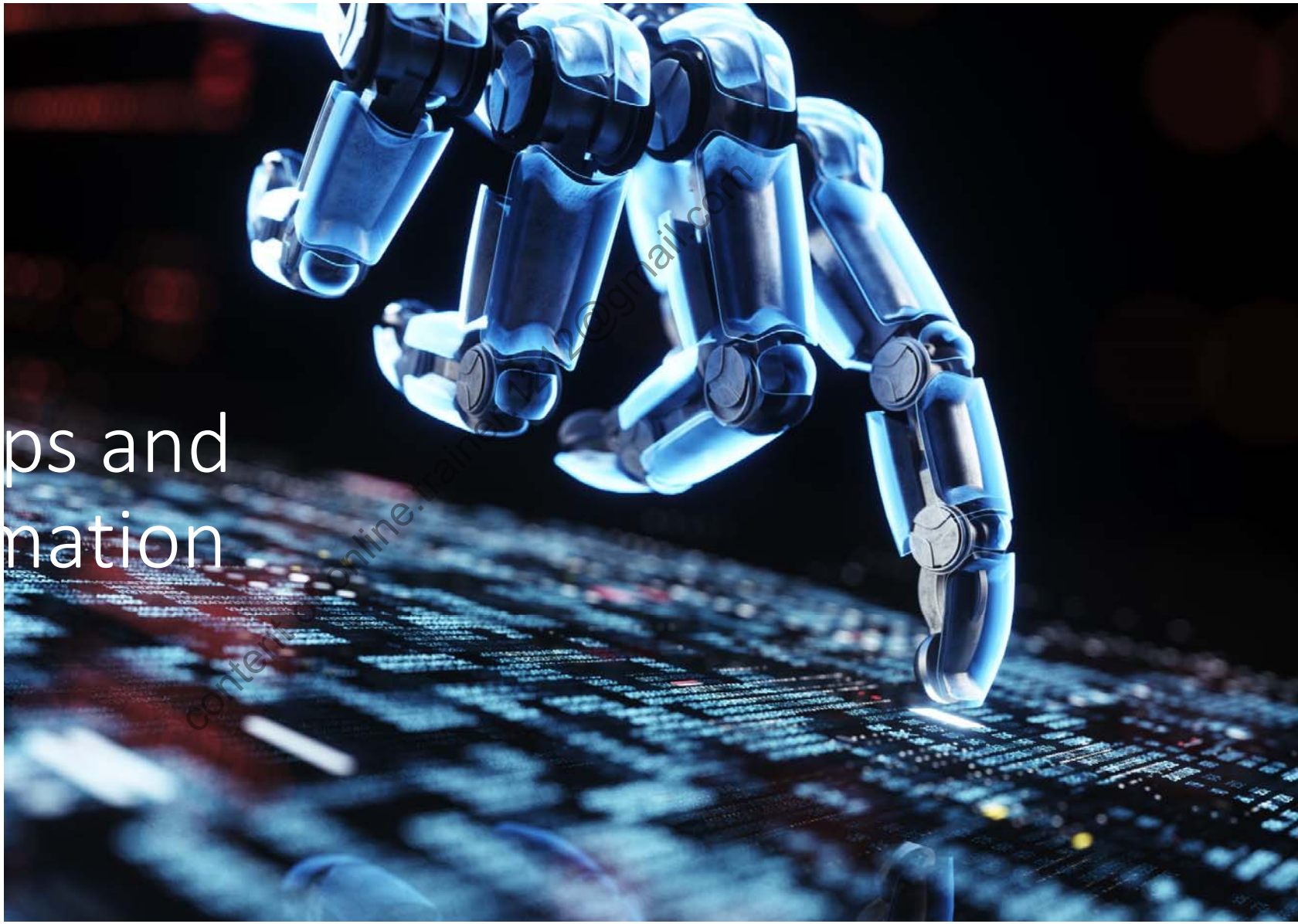
Elements of a cloud native solution.

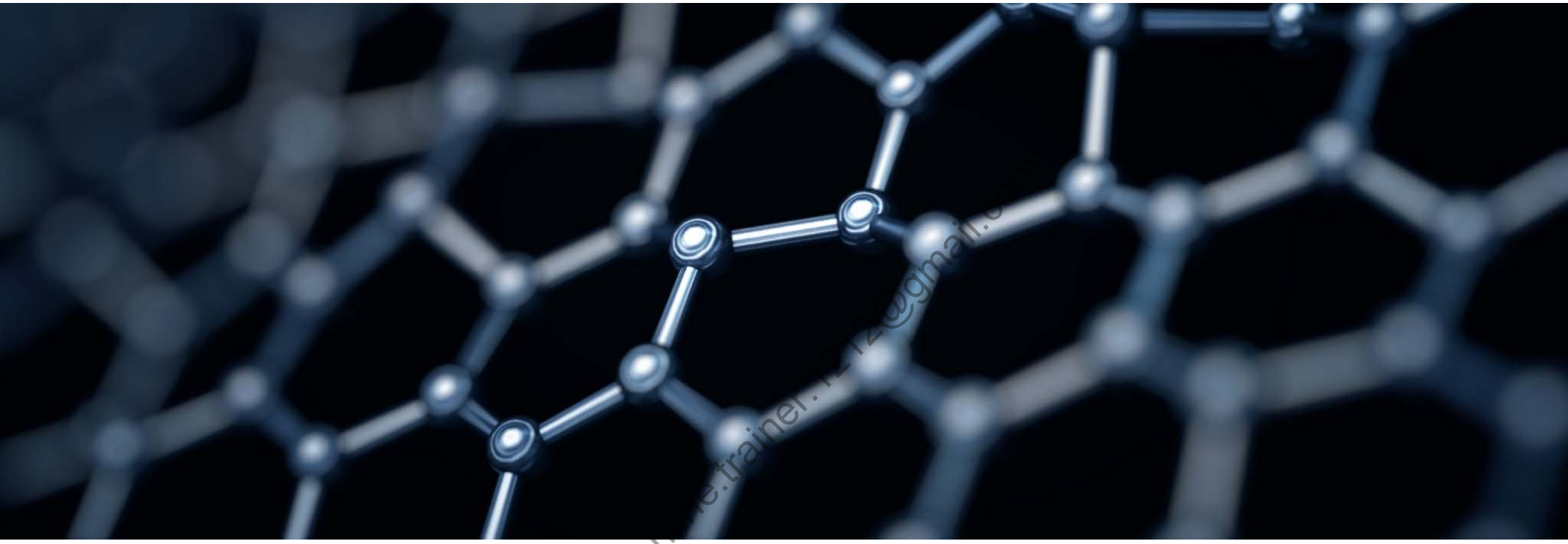


Microservices Architecture:

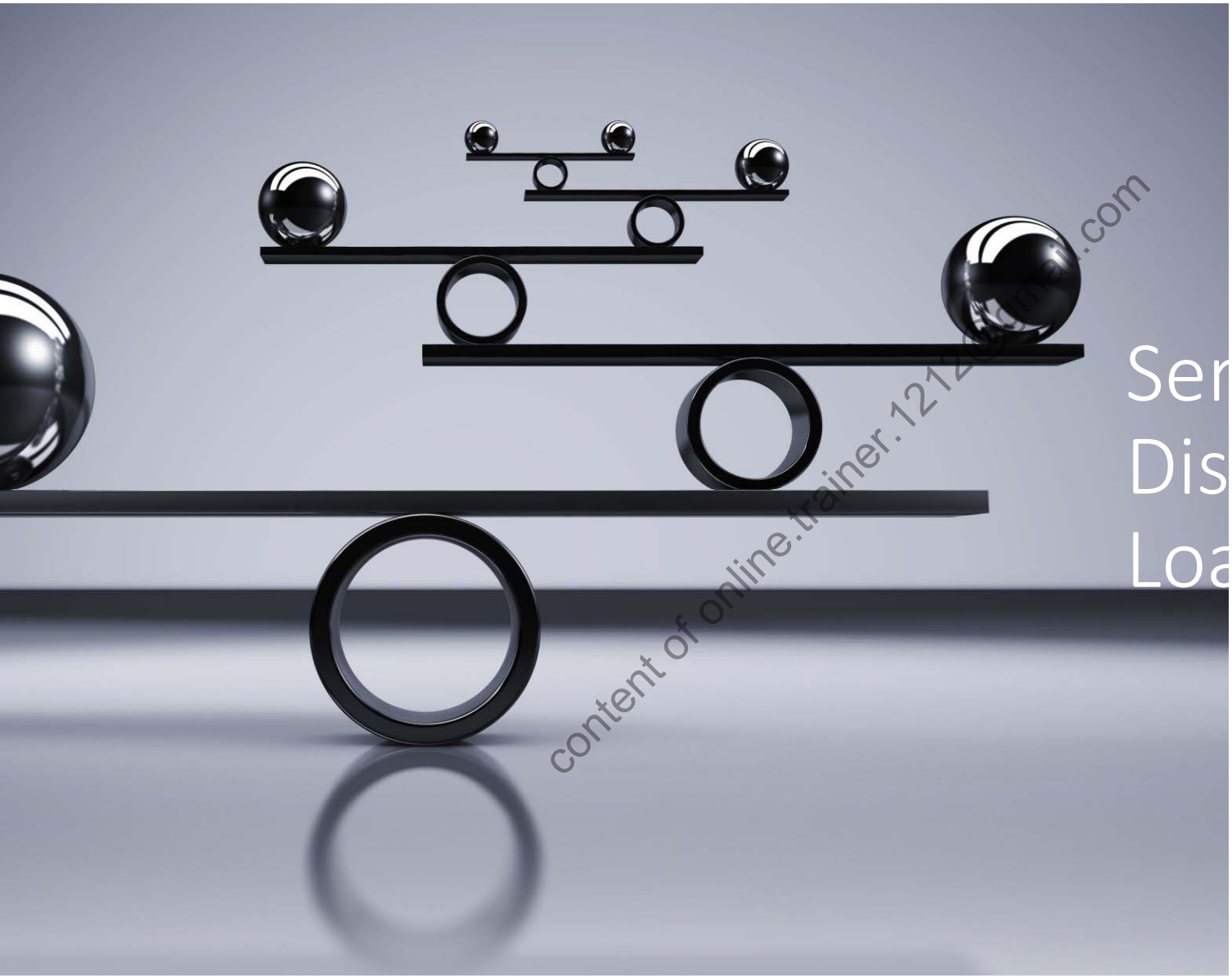
Co



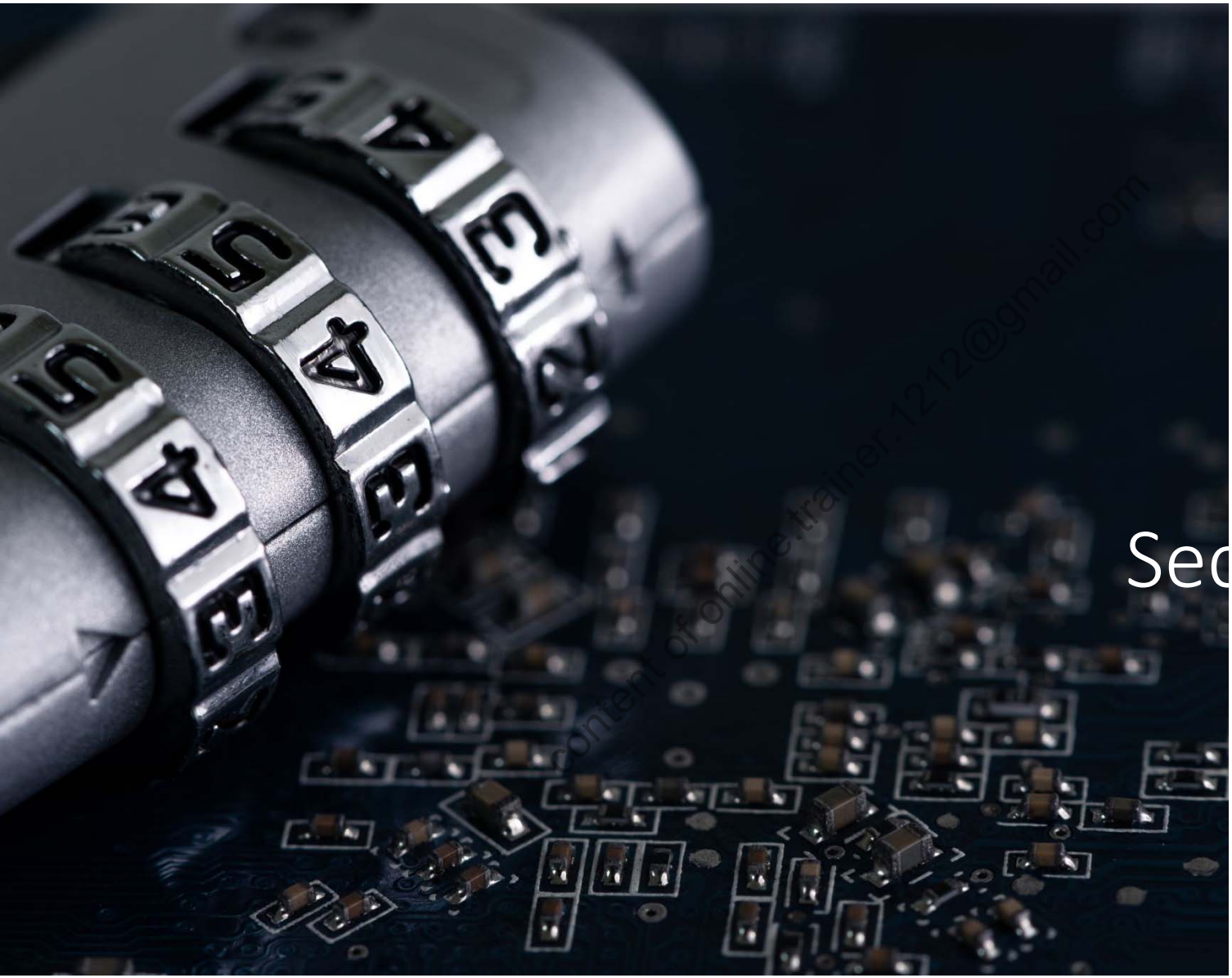

ps and
mation



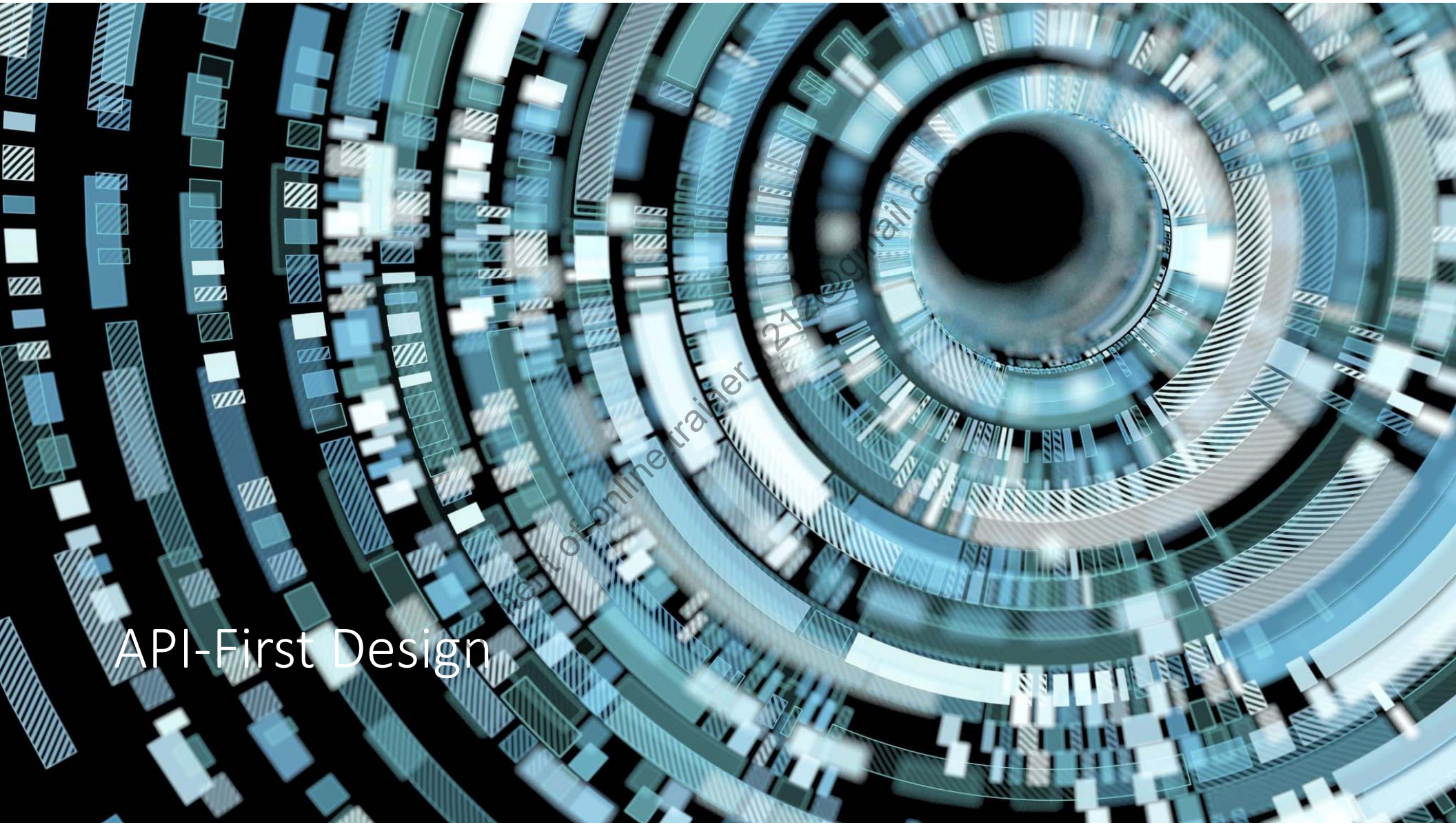
Scalability and Elasticity



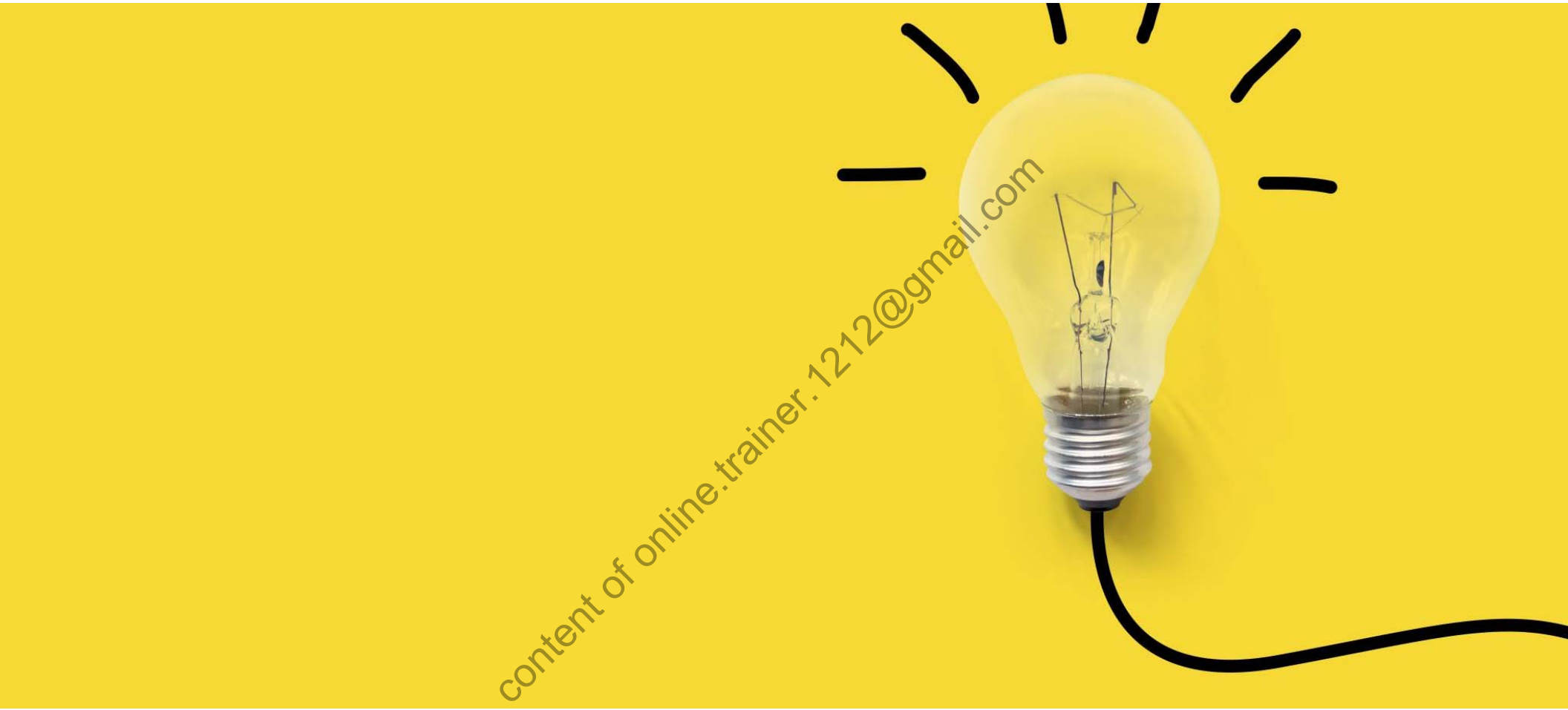
Ser
Dis
Loa



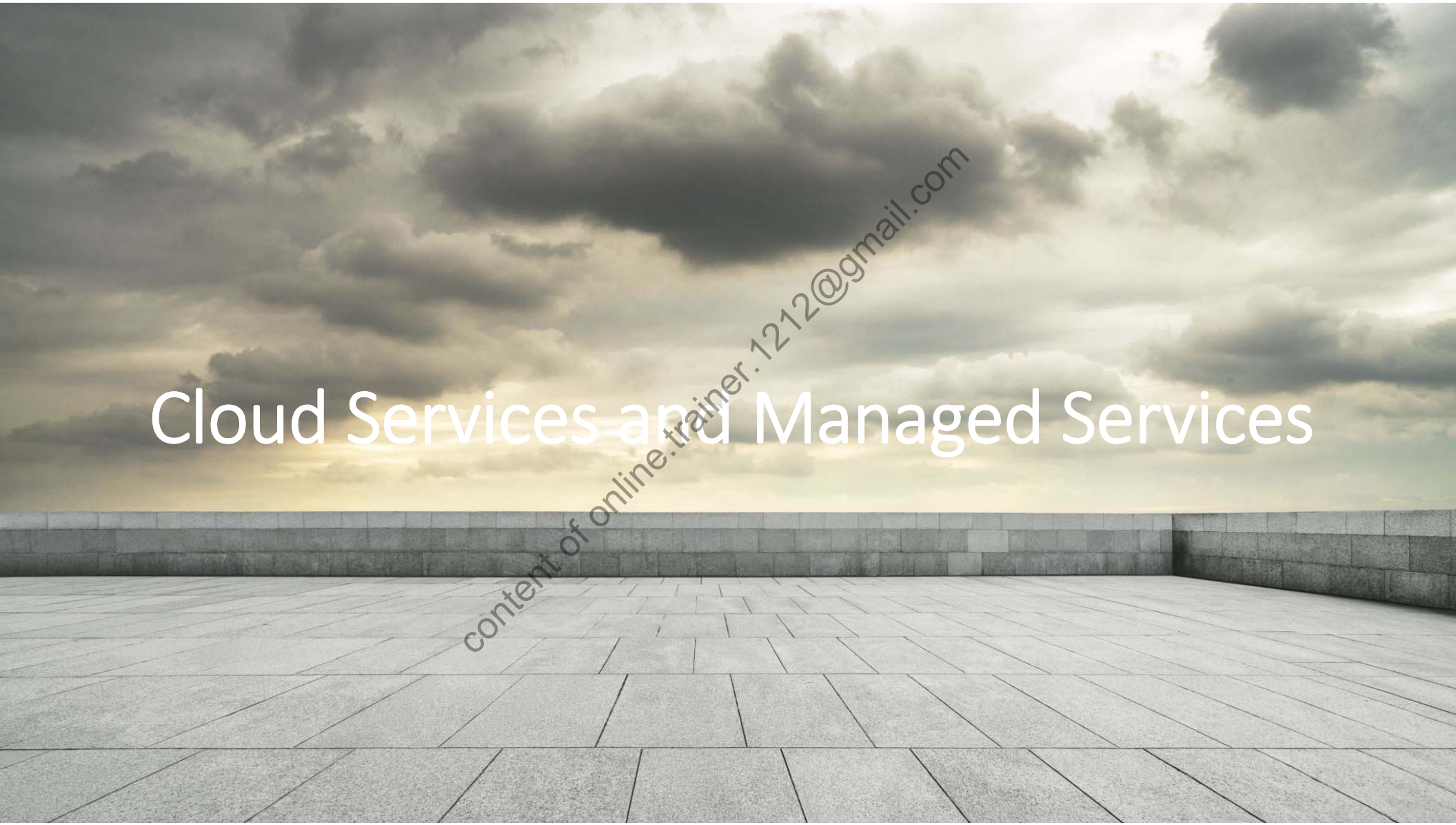
Sec



Continuous Improvement



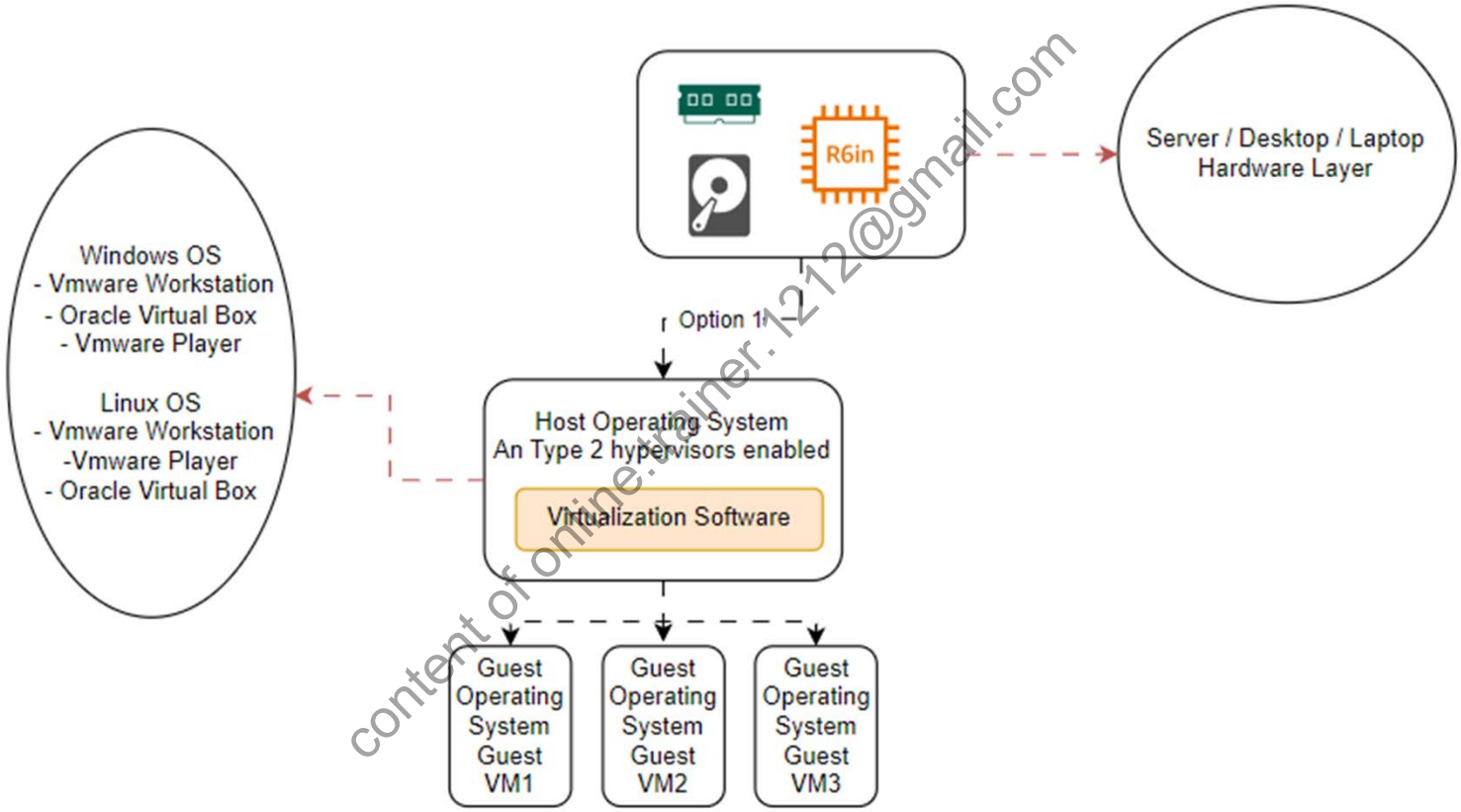
Resource Efficiency



Cloud Services and Managed Services

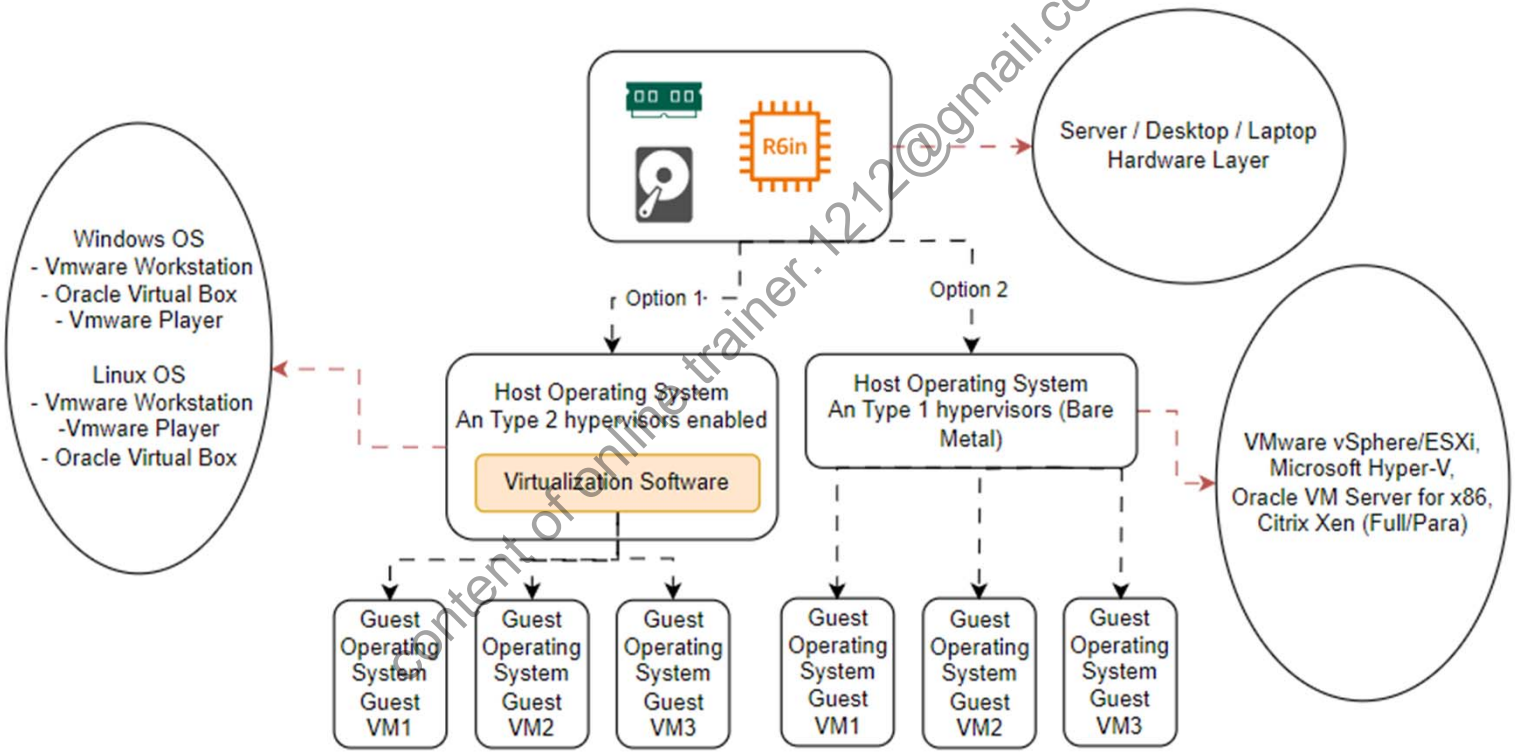
Sub Project 1.1.a

Virtualization Arch



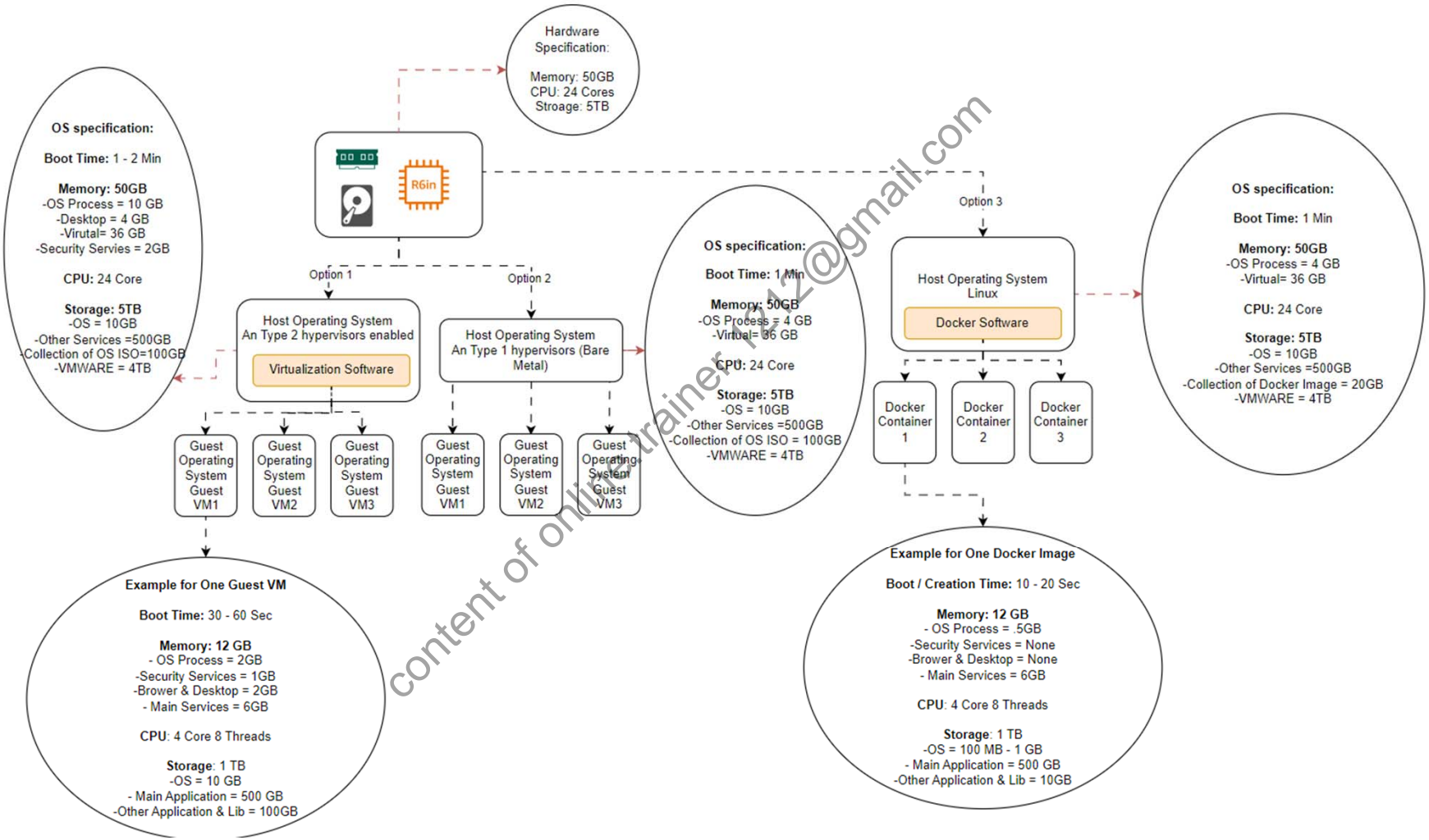
Sub Project 1.1.b

Type1 & 2 Virtualization



Sub Project 1.1.c

Difference btw, Virtualization & Containerization



Summary

Sub Project 1.2

Design a Microservices architecture

Sub Project 1.2.a

What is Microservices

Microservices architecture is an approach in which a single application is composed of many loosely coupled and independently deployable smaller services

Microservices typically…

- Have their own stack, inclusive of the database and data model
- Are organized by business capability, with the line separating services often referred to as a bounded context

Microservices are not…

- Large tightly-coupled application
- Services talk to and integrate with each other in a standardized way

Microservices principles

- One job
- Separate Process
- Execution Scope
- CI/CD
- Resiliency

Benefits of using microservices

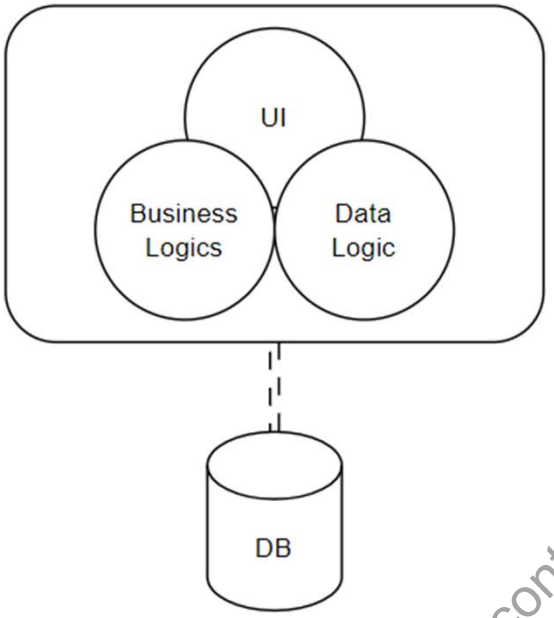
- Simplified deployment
- Improved application quality
- Easier scalability
- More efficient teams

Summary

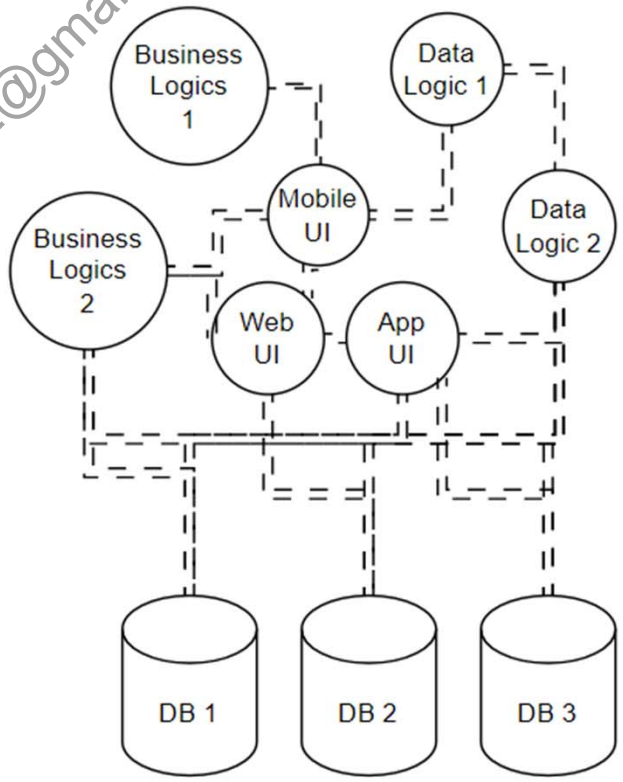
Sub Project 1.2.b

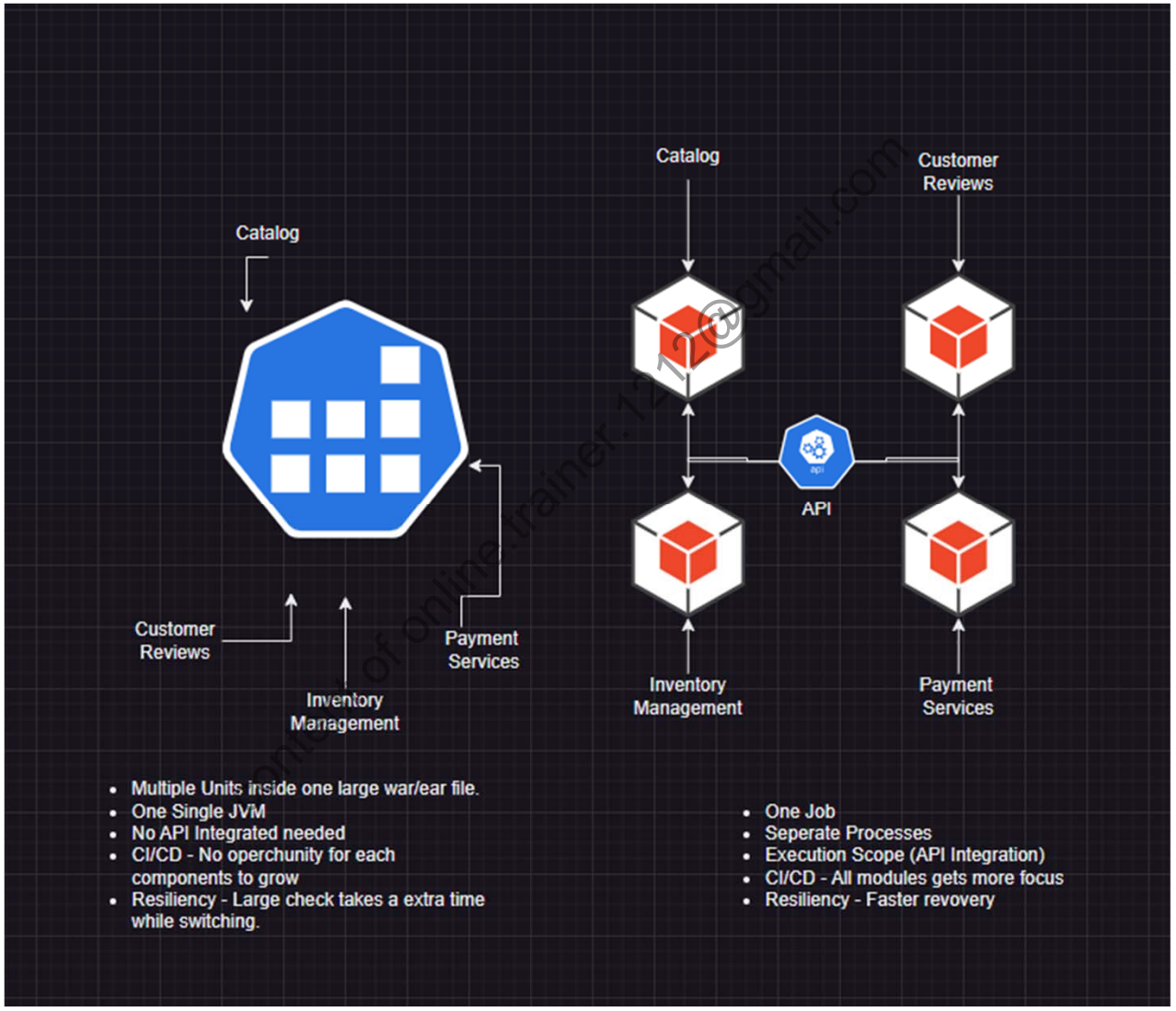
Microservices reference architecture

Monolithic
Architecture



Microservices
Architecture

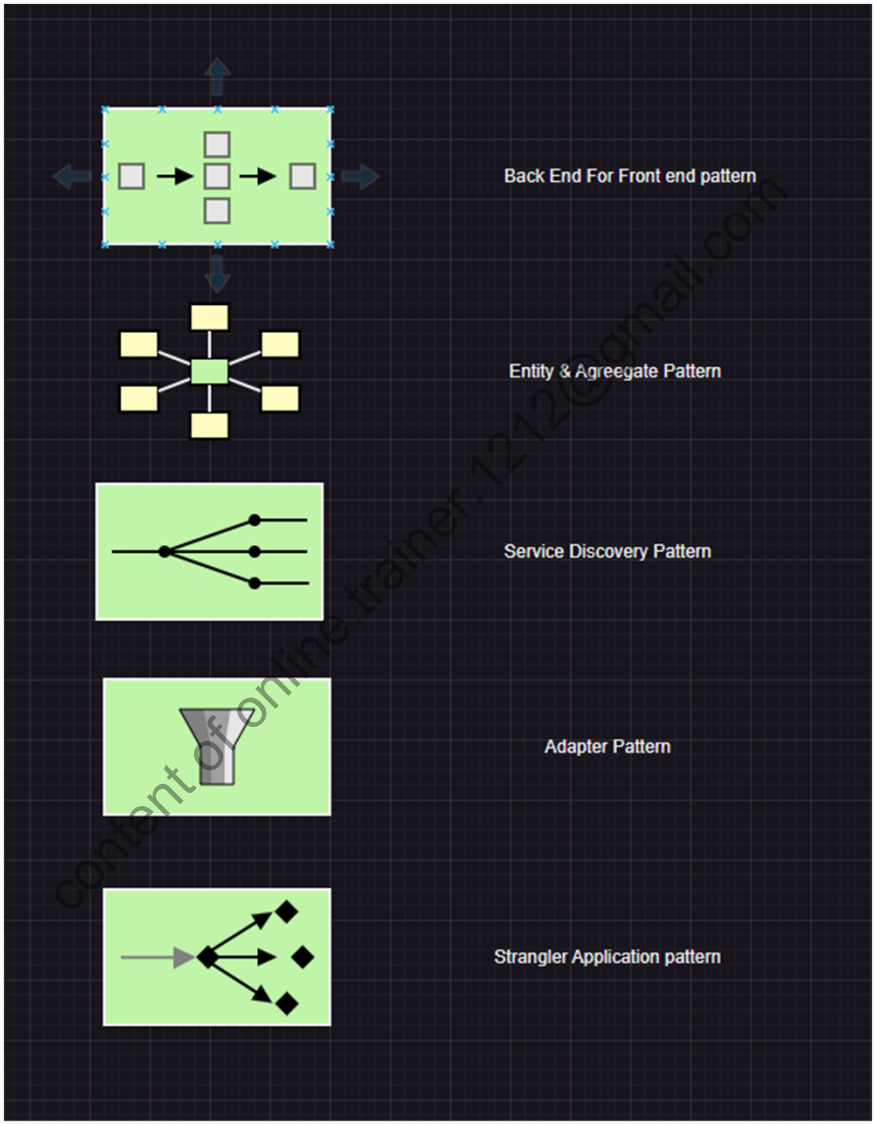



Catalog
Customer Reviews
Inventory Management
Payment Services
Catalog
Customer Reviews
API
Inventory Management
Payment Services
Multiple Units inside one large war/ear file.
One Single JVM
No API Integrated needed
CI/CD - No operchunity for each components to grow
Resiliency - Large check takes a extra time while switching.
One Job
Seperate Processes
Execution Scope (API Integration)
CI/CD - All modules gets more focus
Resiliency - Faster revovery

Summary

Sub Project 1.2.c

Microservices: common patterns



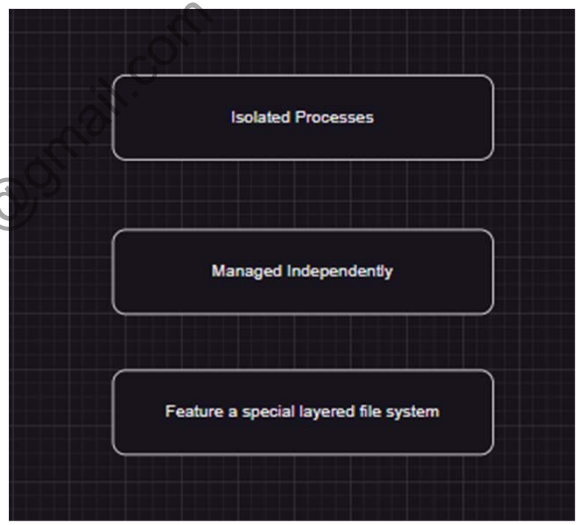
Summary

Sub Project 1.3

Understanding about Containers

What are containers?

- Make efficient use of resources
- Small, fast, and portable



Benefits of using containers

- Lightweight
- Portable, and platform independent
- Supports modern development architecture
- Improves utilization

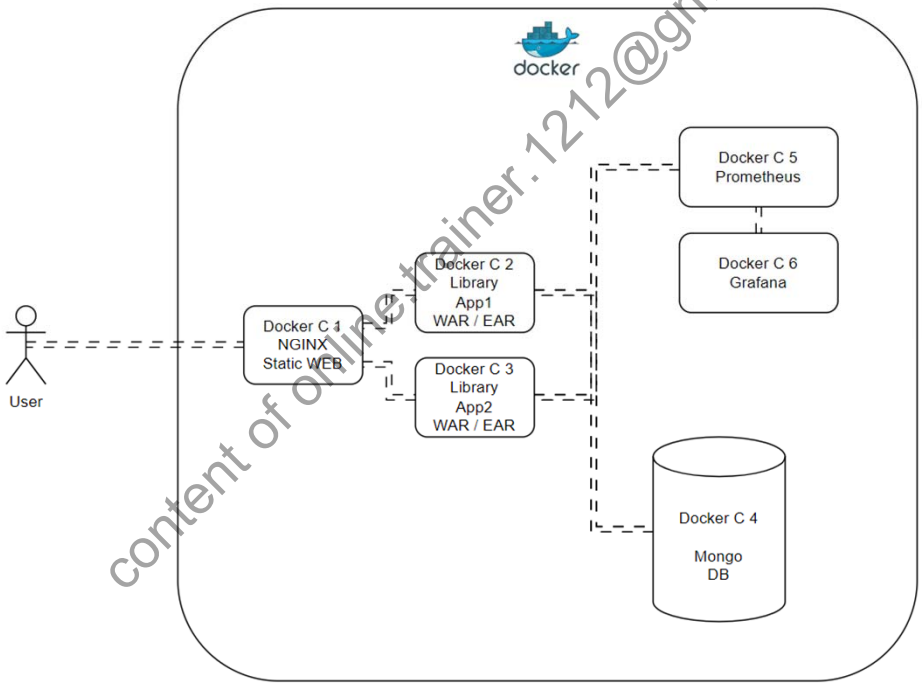
Container lifecycle

Acquire -> Build -> Deliver -> Deploy -> Run -> Maintain

Summary

Sub Project 1.3.a

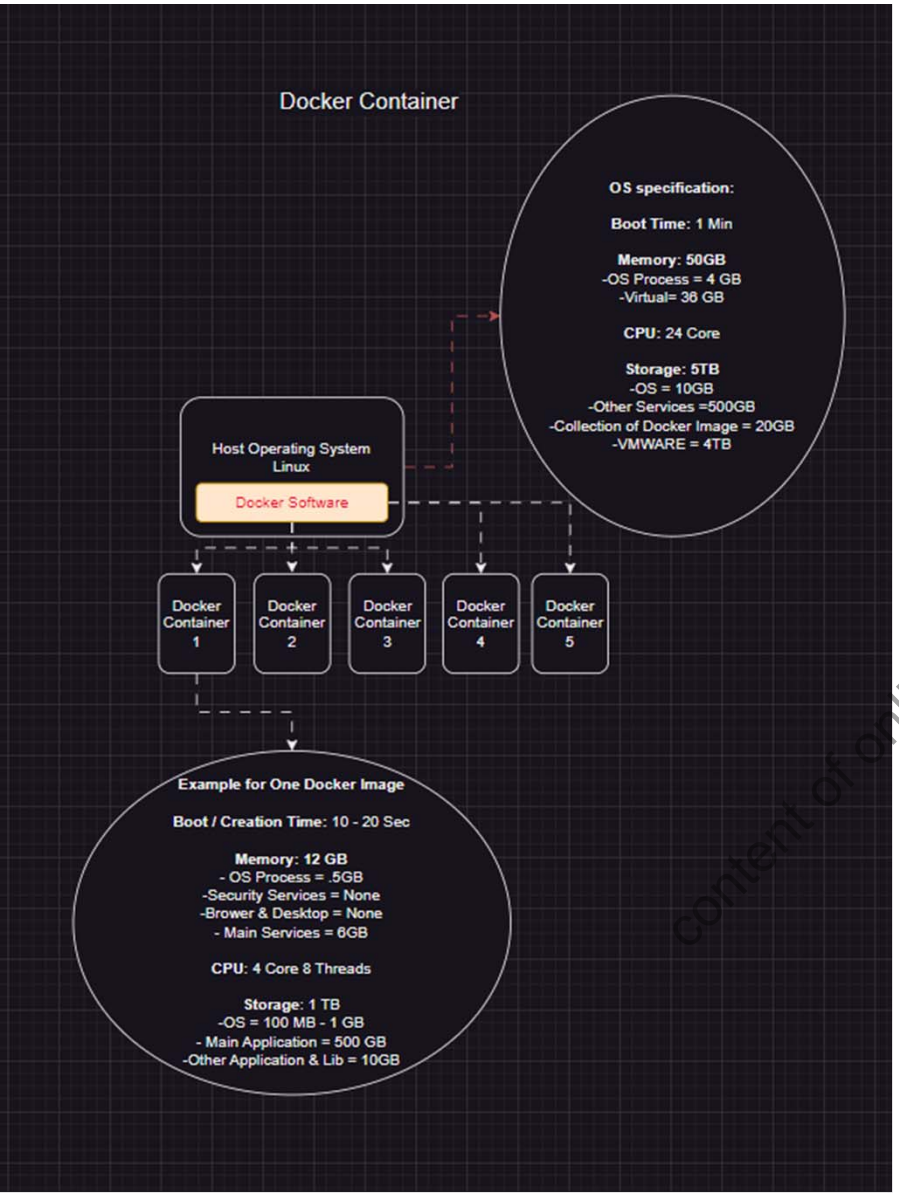
Simple Container Arch

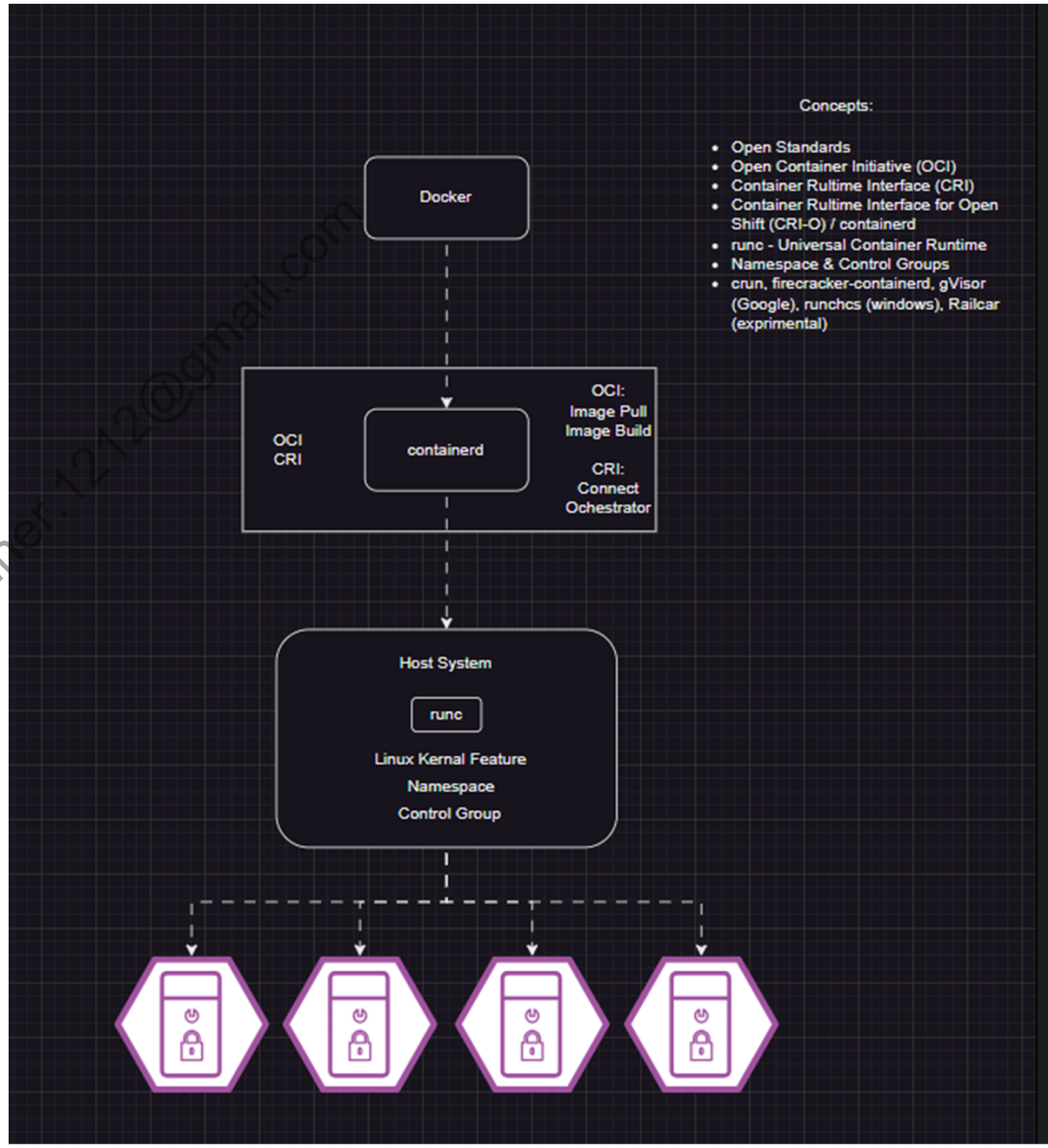


Summary

Sub Project 1.3.a

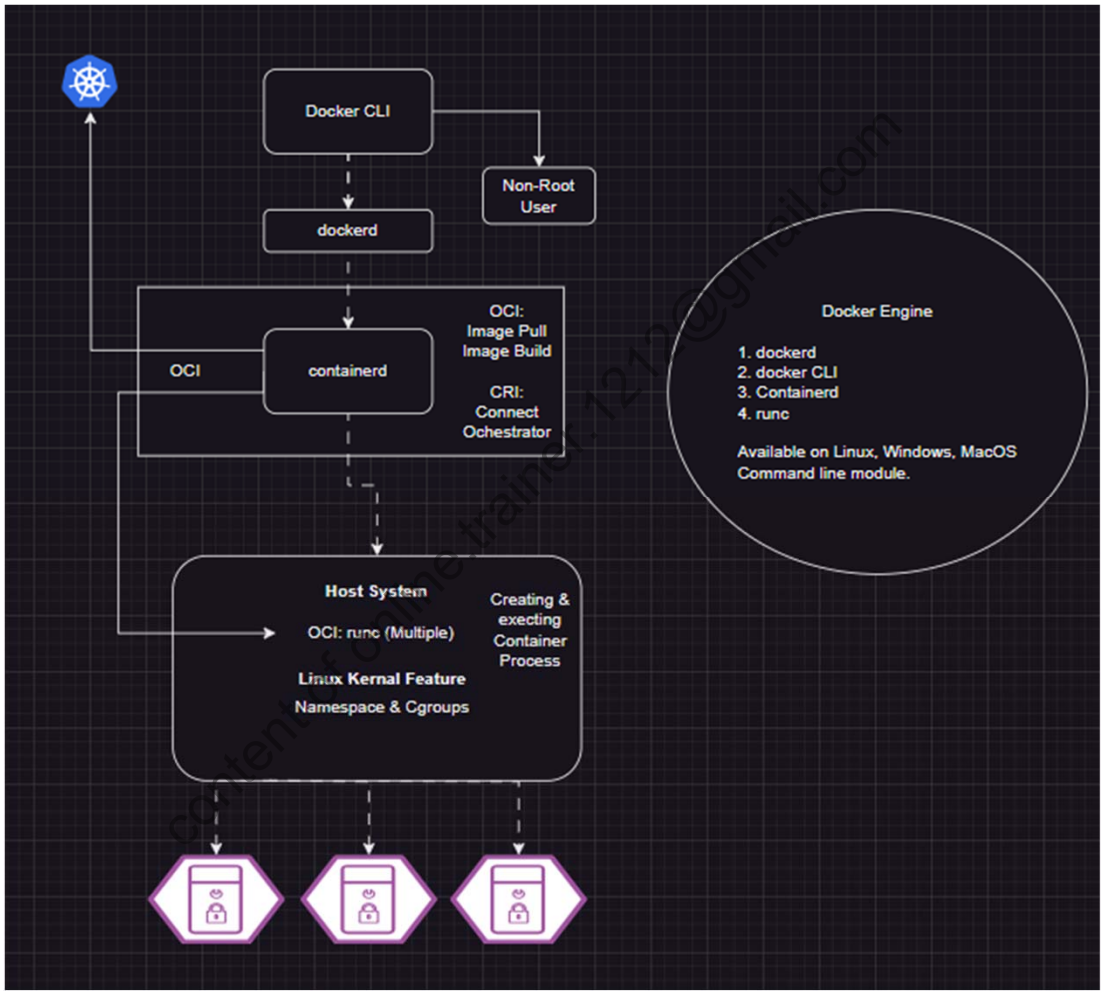
Container Arch





Sub Project 1.3.a

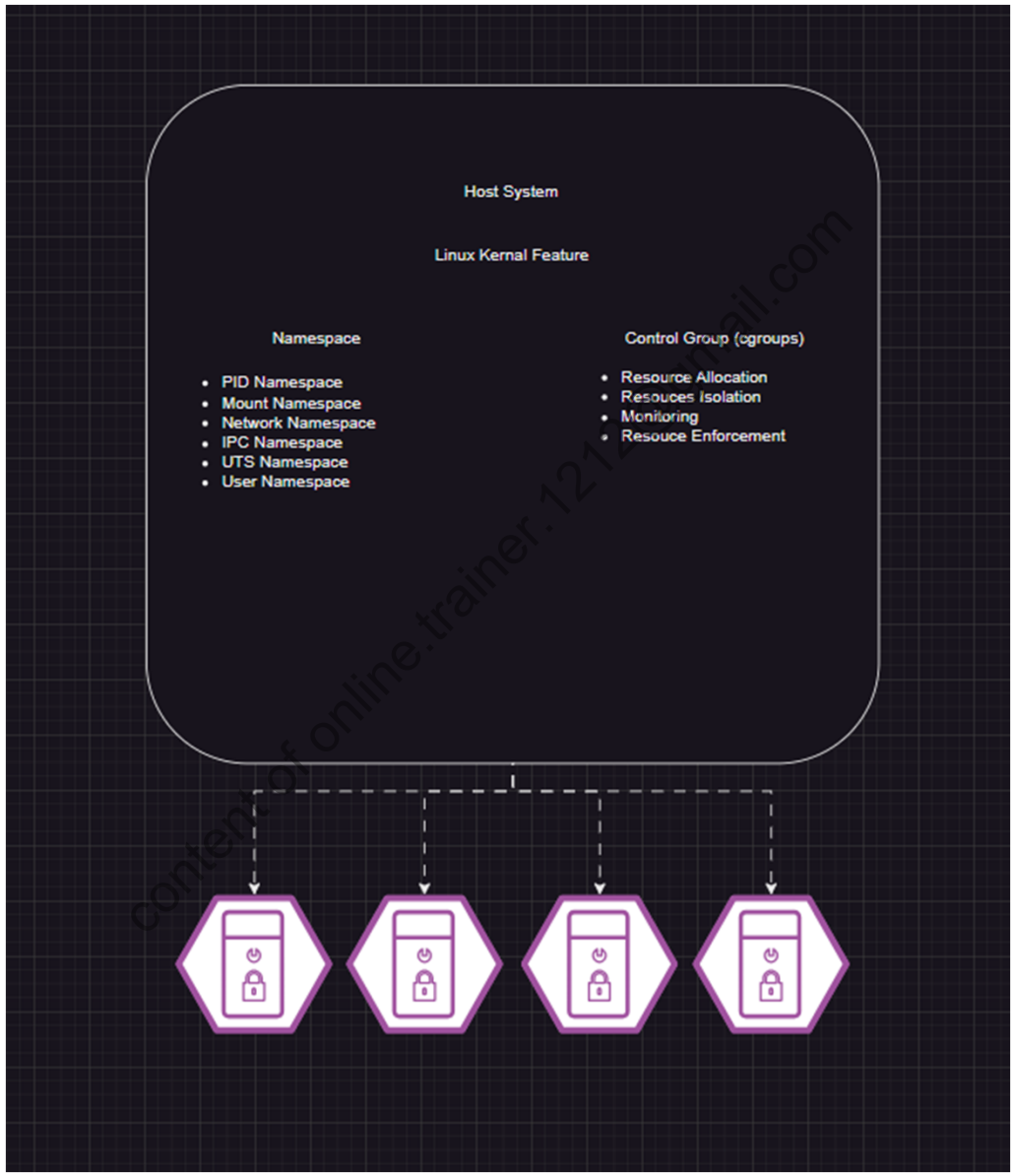
Internal Working Container Arch


Docker CLI
Non-Root User
dockerd
OCI
containerd
OCI:
Image Pull
Image Build
CRI:
Connect
Ochestrator
Host System
OCI: runc (Multiple)
Linux Kernal Feature
Namespace & Cgroups
Creating & execting Container Process
Docker Engine
1. dockerd
2. docker CLI
3. Containerd
4. runc
Available on Linux, Windows, MacOS
Command line module.

Summary

Sub Project 1.3.c

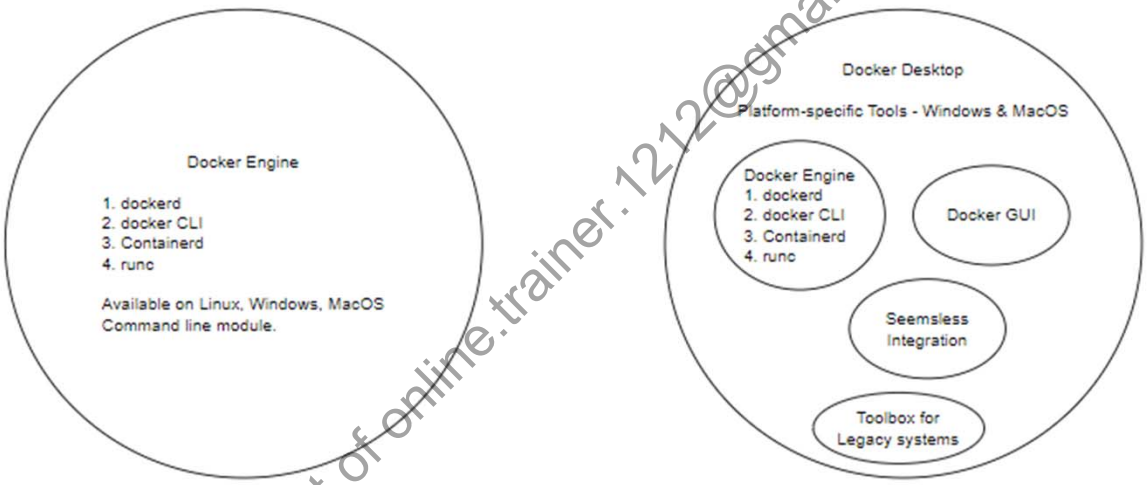
Understanding Namespace & Cgroups



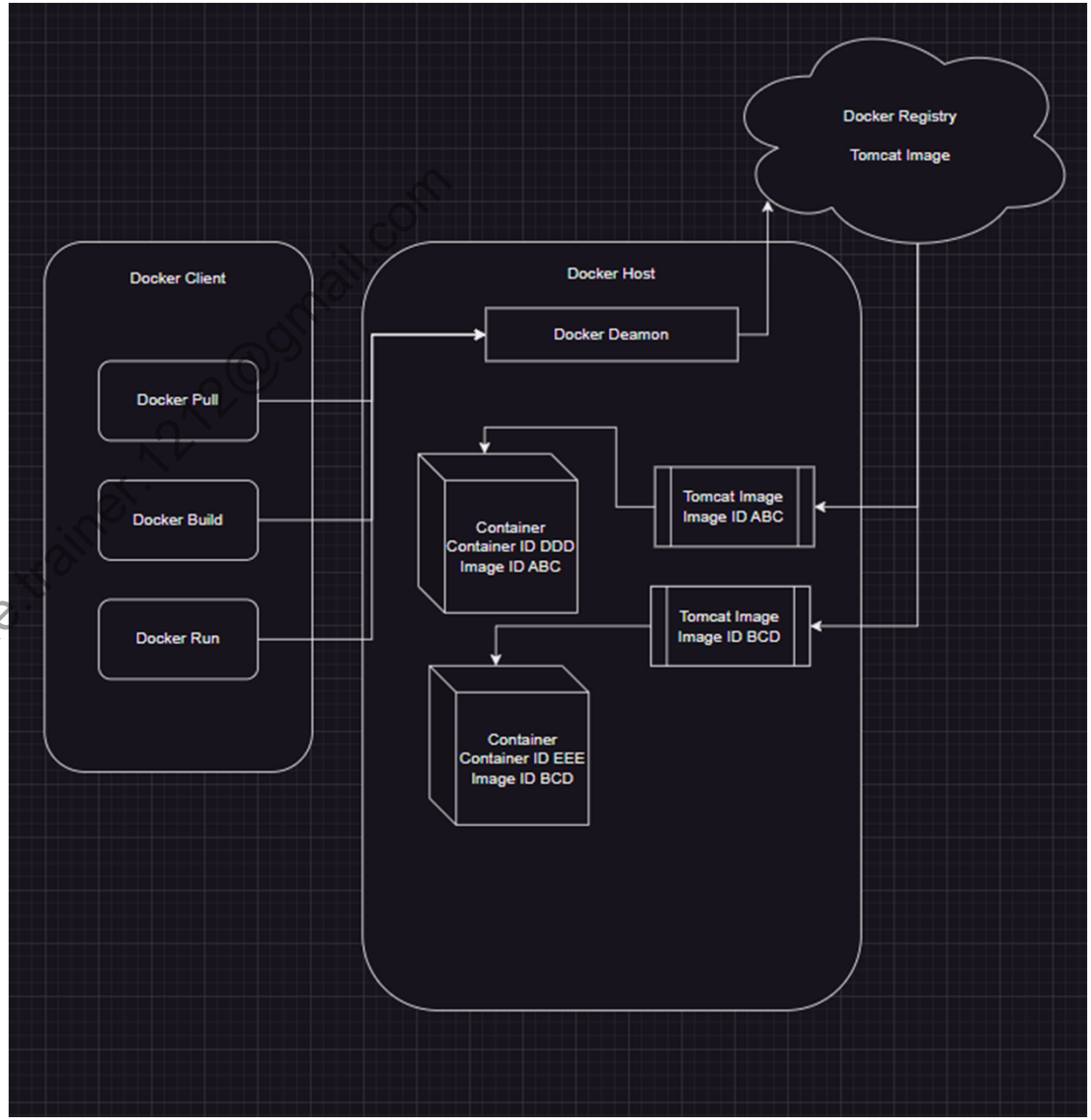
Summary

Sub Project 1.3.d

Docker Desktop VS Docker Engine



How Docker Works in Real Time

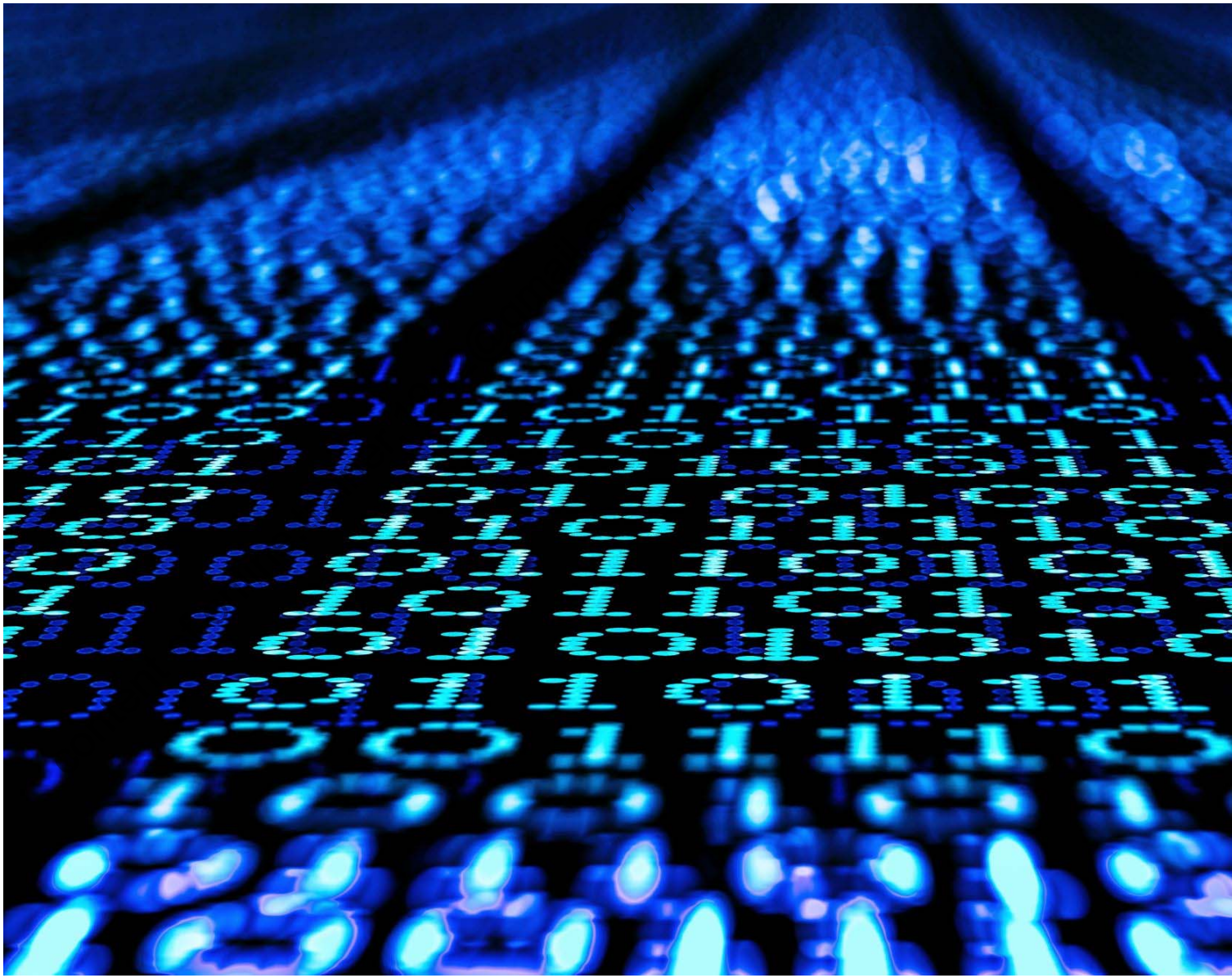


Summary

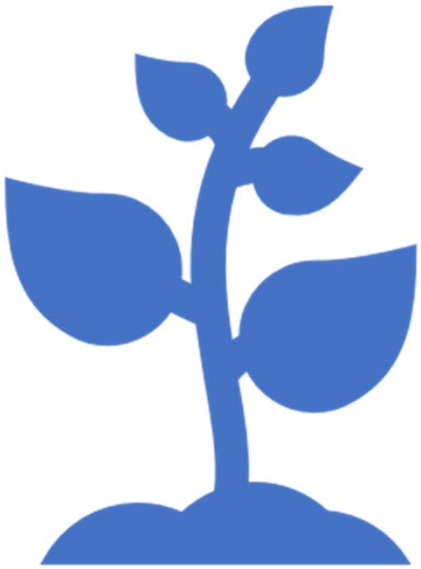
Sub Project 1.3.g

Understanding Kubernetes

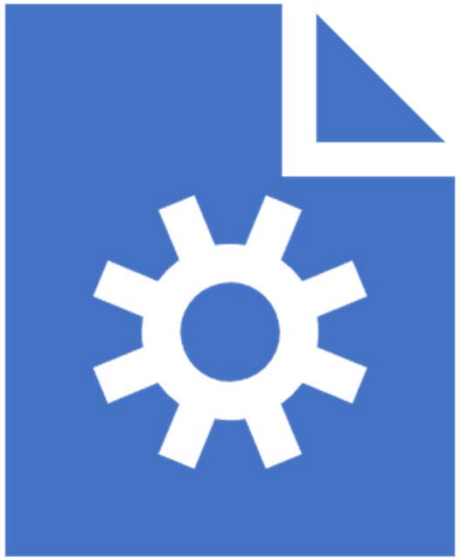
n

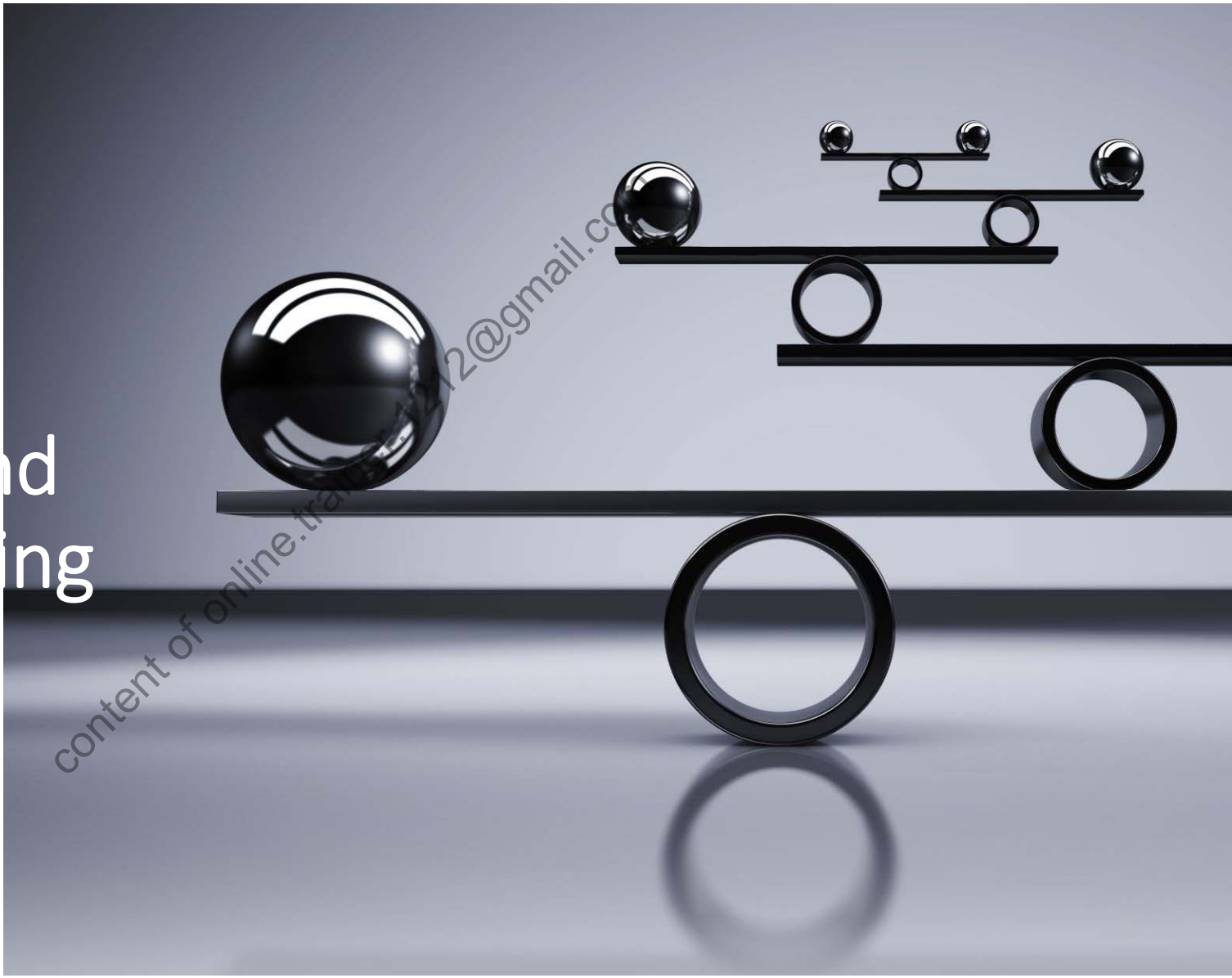


Self-Healing



Declarative Configuration:





nd
ing

Rol
and

Cloud-Native and Microservices

Cost-Efficiency


Multi-Cloud and Hybrid Cloud:

Summary

Sub Project 1.3.h

Kubernetes:

What is Kubernetes?

A popular choice for container orchestrator that handles these container-related tasks:

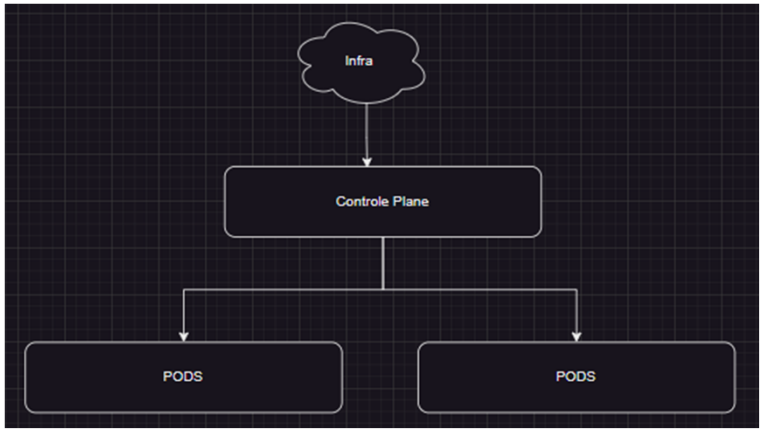
- Deployment
- Rollouts and Rollbacks
- Service discovery
- Storage provisioning
- Load balancing and scaling
- Self-healing for high availability

Kubernetes cluster architecture

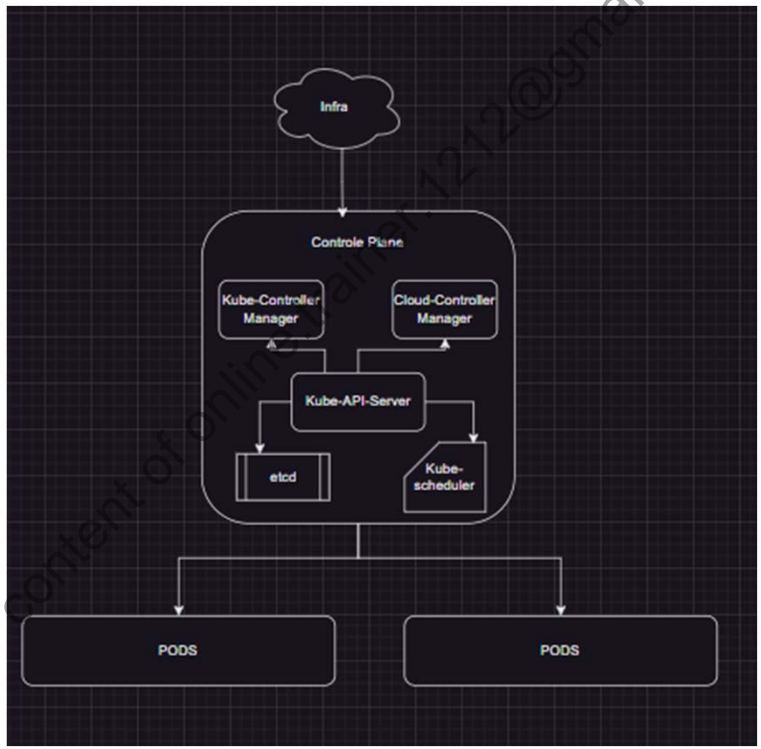
Clusters are the building blocks of Kubernetes architecture. Clusters are made up of nodes, each of which represents a single compute host that runs containerized applications.

Nodes host the Pods that are the components of the application workload

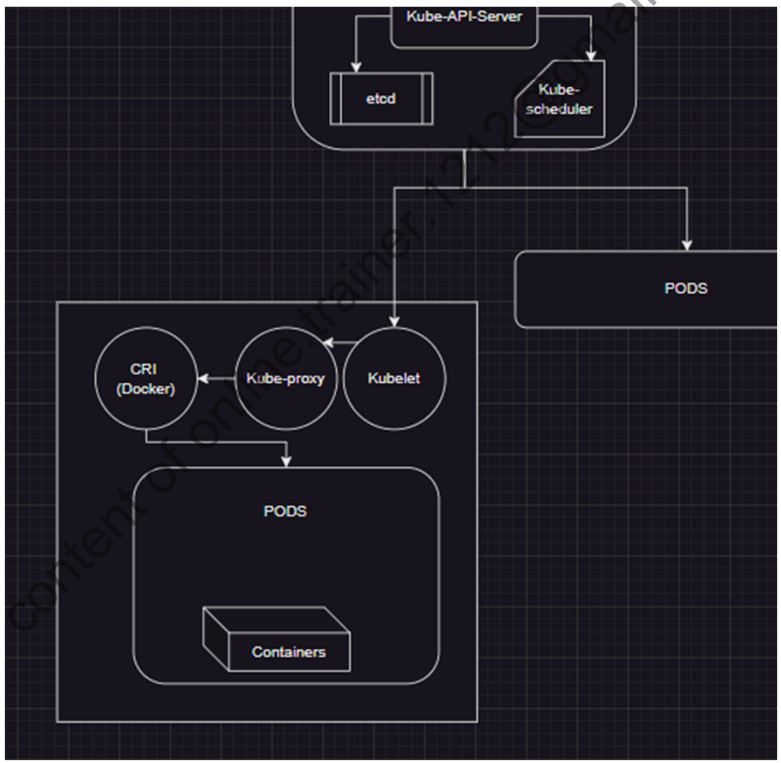
The control plane manages the nodes and the Pods in the cluster. It schedules containers onto the nodes according to available capacity and user-defined configuration.



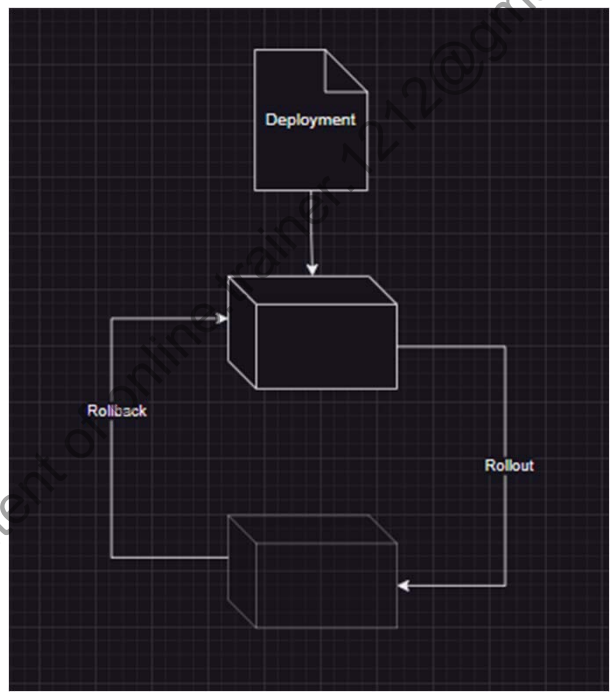
Kubernetes control plane



Worker Node components



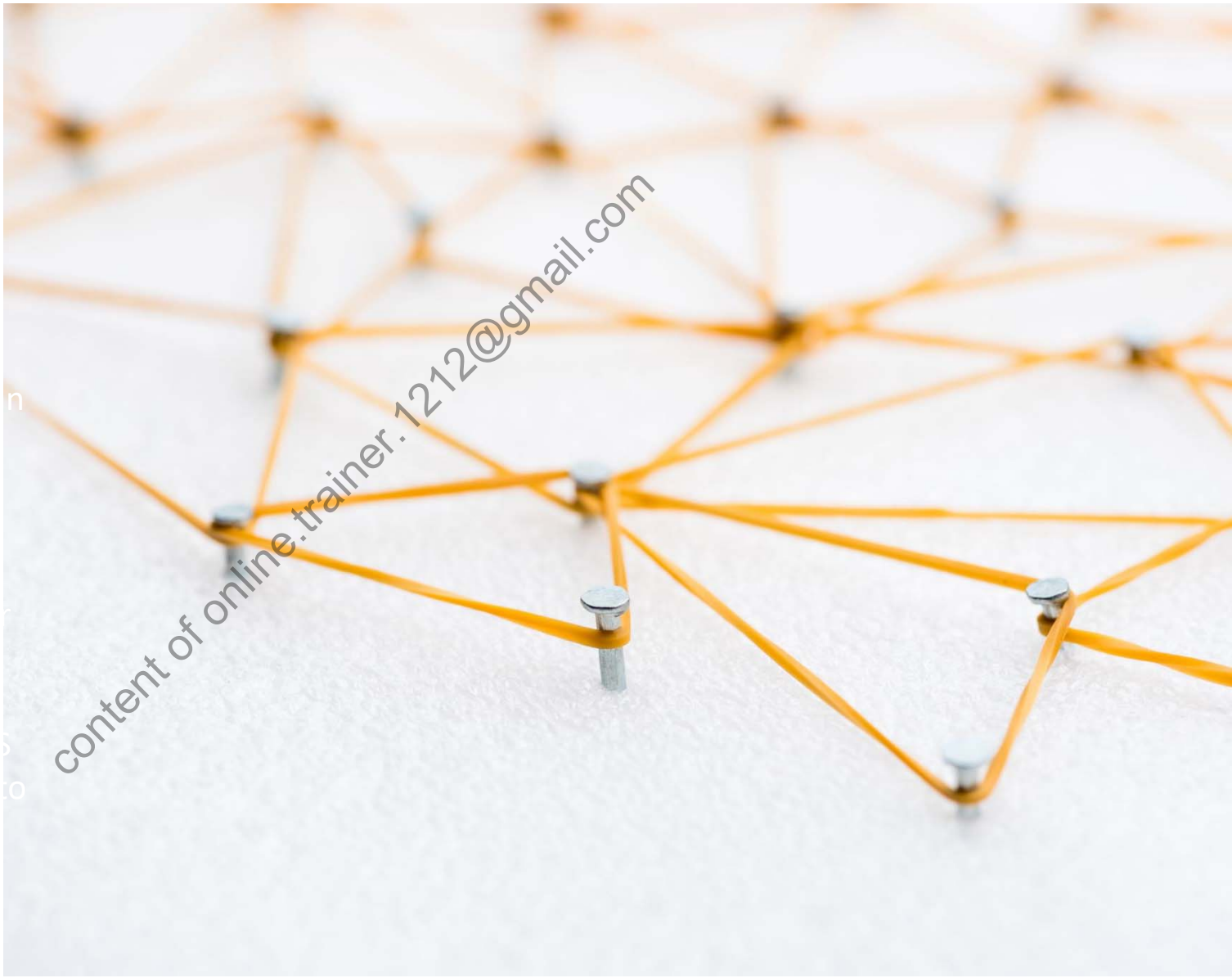
Deployments



Kube-Controller VS Cloud-Controller

S


NIZZA
WIEN
LUXEMBURG
TURIN
AMSTERDAM
VERONA
BERLIN-TEGEL
WASHINGTON
PHILADELPHIA
OTTAWA
LEIPZIG HALLE
DUESSELDORF
GENF
LINZ
BREMEN
NUERNBERG
HANNOVER
BRUESSEL
1060
1238
9304
298
992
262
182
418
426
839
160
078
1218
1252
038
144
H
LH



Kubernetes command-line interface

- Command syntax:

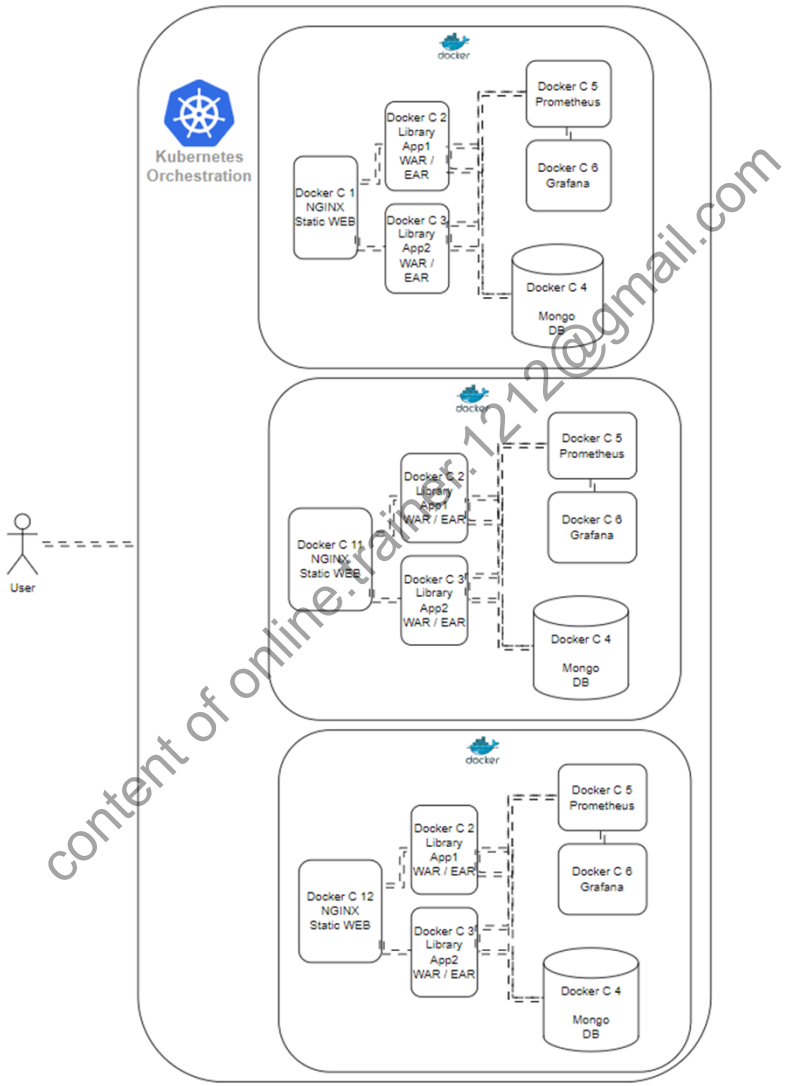
  kubectl [command] [TYPE] [NAME] [flags]

- Where command, TYPE, NAME, and flags are:
- command: Specifies the operation that you want to perform on one or more resources, for example create, get, describe, delete.
- TYPE: Specifies the resource type.
- NAME: Specifies the name of the resource.
- flags: Specifies optional flags. For example, you can use the -s or --server flags to specify the address and port of the Kubernetes API server.

Summary

Sub Project 1.3.I

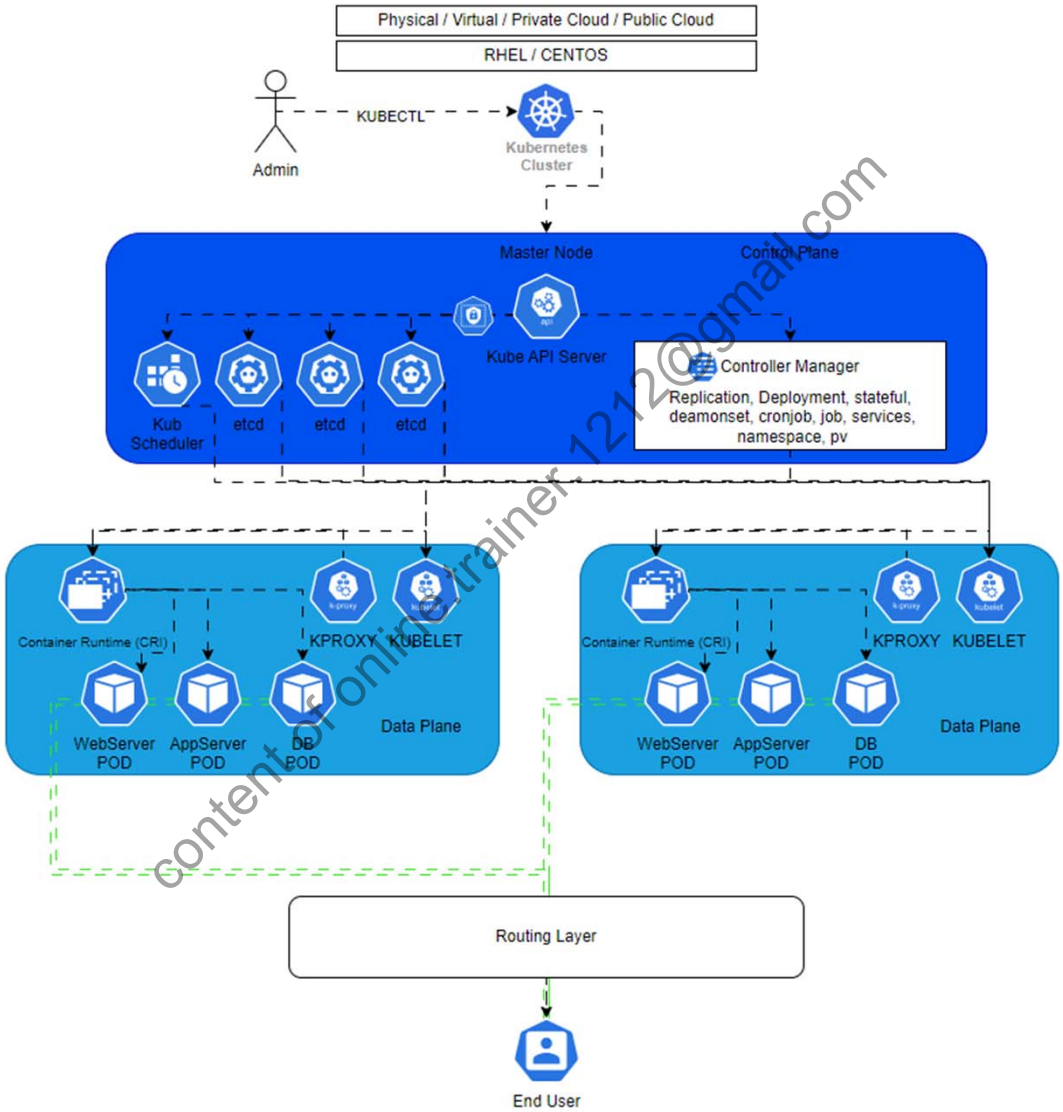
Explain Containers and Container Orchestration



Summary

Sub Project 1.3.J

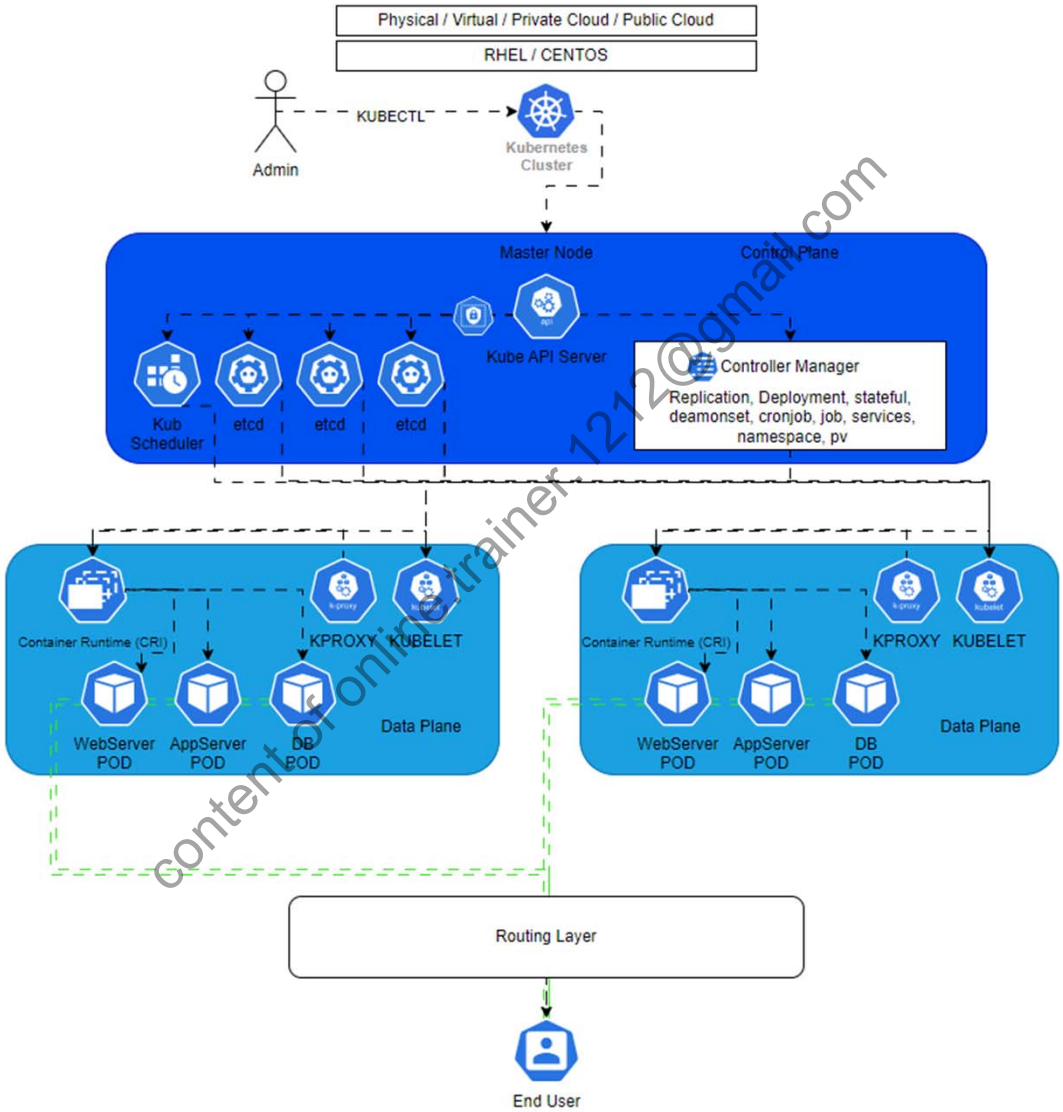
Kubernetes Architecture



Summary

Sub Project 1.3.k

Kubernetes Operations flow



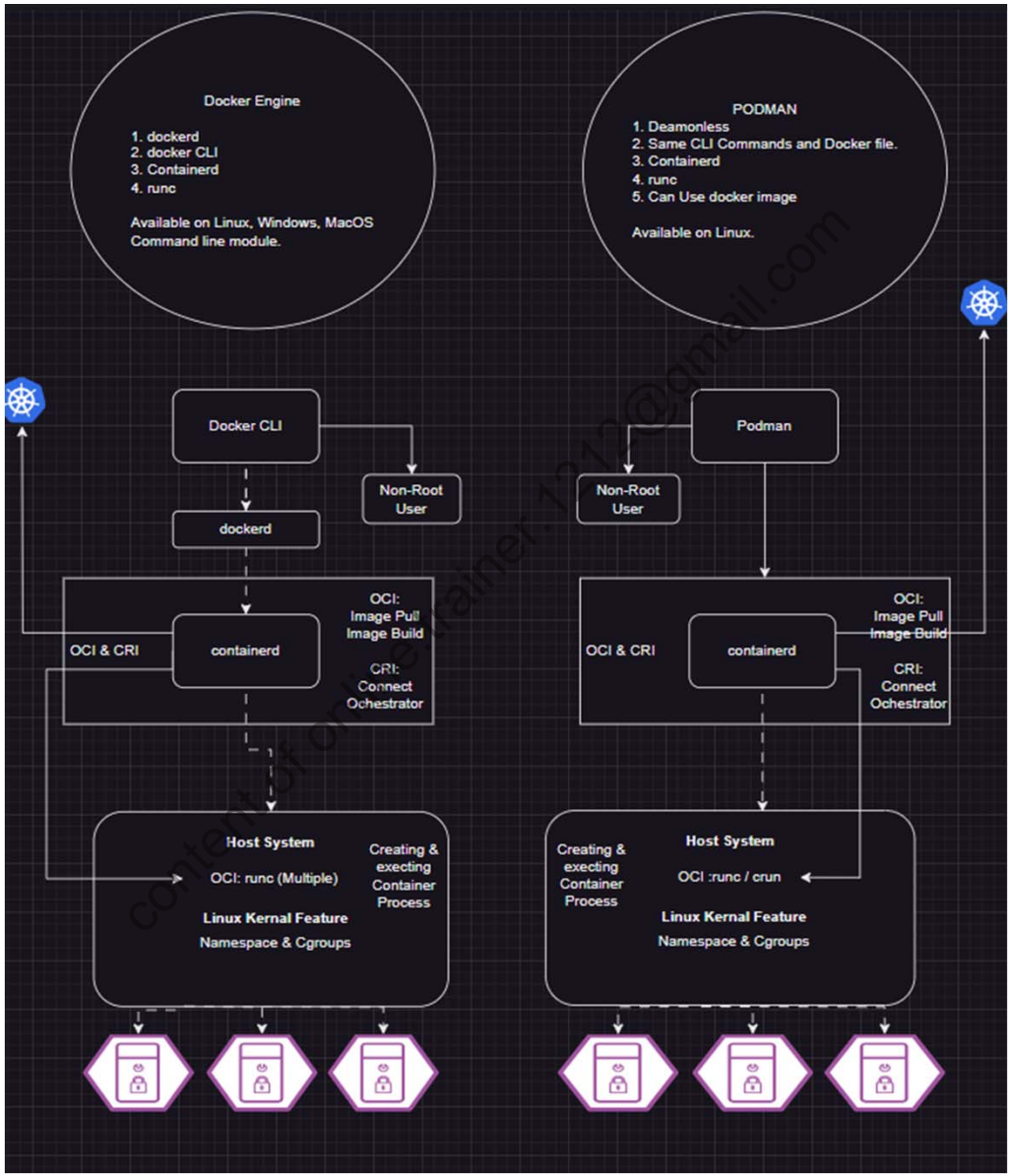
Summary

Sub Project 1.3.L

Docker VS Podman

What is Podman?

- A container engine, like Docker, but daemonless
- Podman commands and syntax are essentially the same (with few exceptions)
- Can be used to manage the entire container ecosystem including pods, containers, container images, and container volumes


Docker Engine
1. dockerd
2. docker CLI
3. Containerd
4. runc
Available on Linux, Windows, MacOS
Command line module.
PODMAN
1. Deamonless
2. Same CLI Commands and Docker file.
3. Containerd
4. runc
5. Can Use docker image
Available on Linux.
Docker CLI
Non-Root User
dockerd
OCI & CRI
containerd
OCI: Image Pull Image Build
CRI: Connect Ochestrator
Host System
OCI: runc (Multiple)
Linux Kernal Feature
Namespace & Cgroups
Creating & execting Container Process
Podman
Non-Root User
OCI & CRI
containerd
OCI: Image Pull Image Build
CRI: Connect Ochestrator
Creating & execting Container Process
Host System
OCI :runc / crun
Linux Kernal Feature
Namespace & Cgroups

Podman example commands

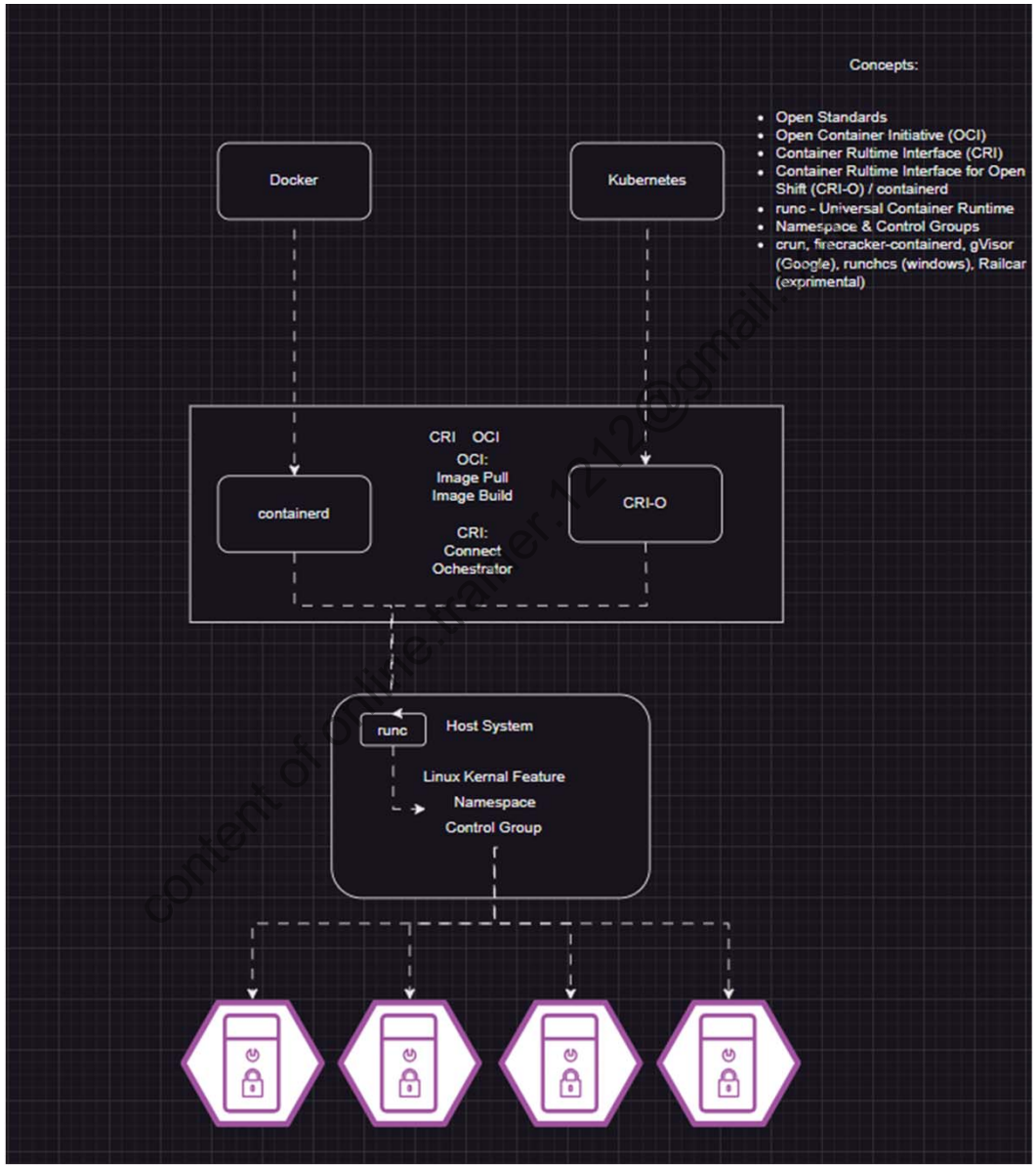
- podman search busybox

- podman run -it docker.io/library/busybox

Summary

Sub Project 1.5.M

Docker VS Kubernetes


Concepts:
Open Standards
Open Container Initiative (OCI)
Container Rultime Interface (CRI)
Container Rultime Interface for Open Shift (CRI-O) / containerd
runc - Universal Container Runtime
Namespace & Control Groups
crun, firecracker-containerd, gVisor (Google), runchcs (windows), Railcar (exprimental)
Docker
Kubernetes
CRI OCI
OCI:
Image Pull
Image Build
CRI:
Connect
Ochestrator
containerd
CRI-O
runc
Host System
Linux Kernal Feature
Namespace
Control Group

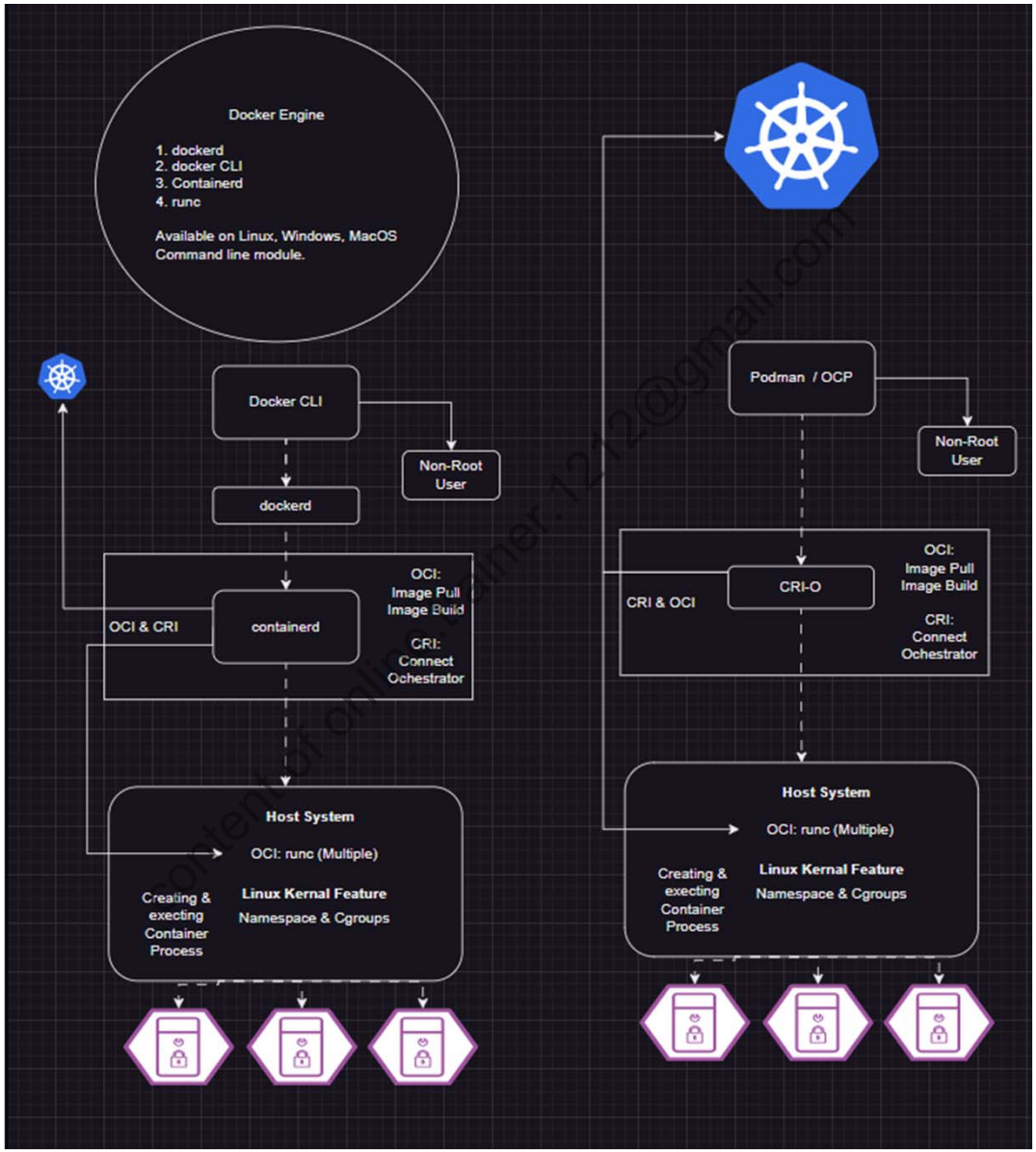
Summary

Sub Project 1.3.N

Containerd VS CRI-O

What is CRI-O

- Kubernetes Integration:
- Lightweight:
- OCI Compatibility:
- Security and Isolation:
- OCI-Compliant Container Images:
- Daemonless:
- Performance:


Docker Engine
1. dockerd
2. docker CLI
3. Containerd
4. runc
Available on Linux, Windows, MacOS
Command line module.
Docker CLI
Non-Root User
dockerd
containerd
OCI & CRI
OCI:
Image Pull
Image Build
CRI:
Connect
Ochestrator
Host System
OCI: runc (Multiple)
Creating & execting Container Process
Linux Kernal Feature
Namespace & Cgroups
Podman / OCP
Non-Root User
CRI-O
CRI & OCI
OCI:
Image Pull
Image Build
CRI:
Connect
Ochestrator
Host System
OCI: runc (Multiple)
Creating & execting Container Process
Linux Kernal Feature
Namespace & Cgroups

Summary

Sub Project 1.4

Understand the Cloud Native Reference Architecture

Cloud native is...

A cloud native application consists of discrete, reusable components known as microservices that are designed to integrate into any cloud environment.

Cloud native refers less to where an application resides and more to how it is built and deployed.

CNCF Cloud Native Definition v1.0

“Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach. These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.”

12-factor app methodology

I. Codebase
II. Dependencies
III. Config
IV. Backing services
V. Build, release, run
VI. Processes
VII. Port binding
VIII. Concurrency
IX. Disposability
X. Dev/prod parity
XI. Logs
XII. Admin processes

Cloud native and microservices

- A microservice is a small application with a small footprint that performs a specific function
- Microservices enable an architectural approach where a much larger application is composed of discrete, independently deployed components
- Microservices work well with cloud native apps because of their modular structure, compatibility with Agile and DevOps processes, and harmony with container use

Cloud Native versus…

- Cloud enabled: Developed for deployment in a traditional data center but later changed to run in a cloud environment
- Cloud ready: Services or software designed to work over the internet, or any app that can work in a cloud environment
- Cloud based: A general term that is applied to various cloud offerings that are delivered over the internet
- Cloud first: A business strategy in which organizations commit to using cloud resources first when launching new IT services, refreshing existing services, or replacing legacy technology

Cloud Native development principles

- Follow the microservices architectural approach
- Rely on containers for maximum flexibility and scalability
- Adopt Agile methods

Critical Cloud Native components

- Packaged as lightweight containers
- Designed as loosely-coupled microservices
- Deployed on a virtual, shared and elastic infrastructure (cloud) that supports horizontal scaling
- Use lightweight APIs that are based on protocols such as representational state transfer (REST)
- Use a service mesh for handling service-to-service communication
- Managed through agile DevOps processes or CI/CD pipelines

Other attributes of Cloud Native apps

- Polyglot; each service uses the language and framework best suited for the functionality
- Separation of stateful and stateless
- Policy-driven resource provisioning
- Zero downtime

Service mesh

A networking model that sits at a layer of abstraction above TCP/IP

Responsible for the reliable delivery of requests through the complex topology of services that comprise a modern, cloud native application

Cloud Native reference architecture

Step 1: Next Gen Dev Experience

Step 2: IBM Certified Containers

Step 3: Kubernetes Workloads

Cloud Native reference architecture

Security

Service Management

DevOps

Non-functional requirements

Strategize first for greatest success

1. Know the needs of your business
2. Choose the right cloud for the right workload
3. Have a knowledge of new technologies and methods
4. Get valuable experience and industry insights

Finding the freedom to innovate

5. Uncover new possibilities with open-based architecture
6. Create an infrastructure built for innovative technologies
7. Leverage the speed and agility of cloud native
8. Small and mighty microservices
9. Cut the coding with containers
10. Maintain order with orchestration

Iterate to innovate with DevOps

11. Transform into a culture of collaboration
12. Alleviate awkward handoffs with automation

Summary

Sub Project 1.5

Hands On - Container Engine Labs Docker vs CRIO

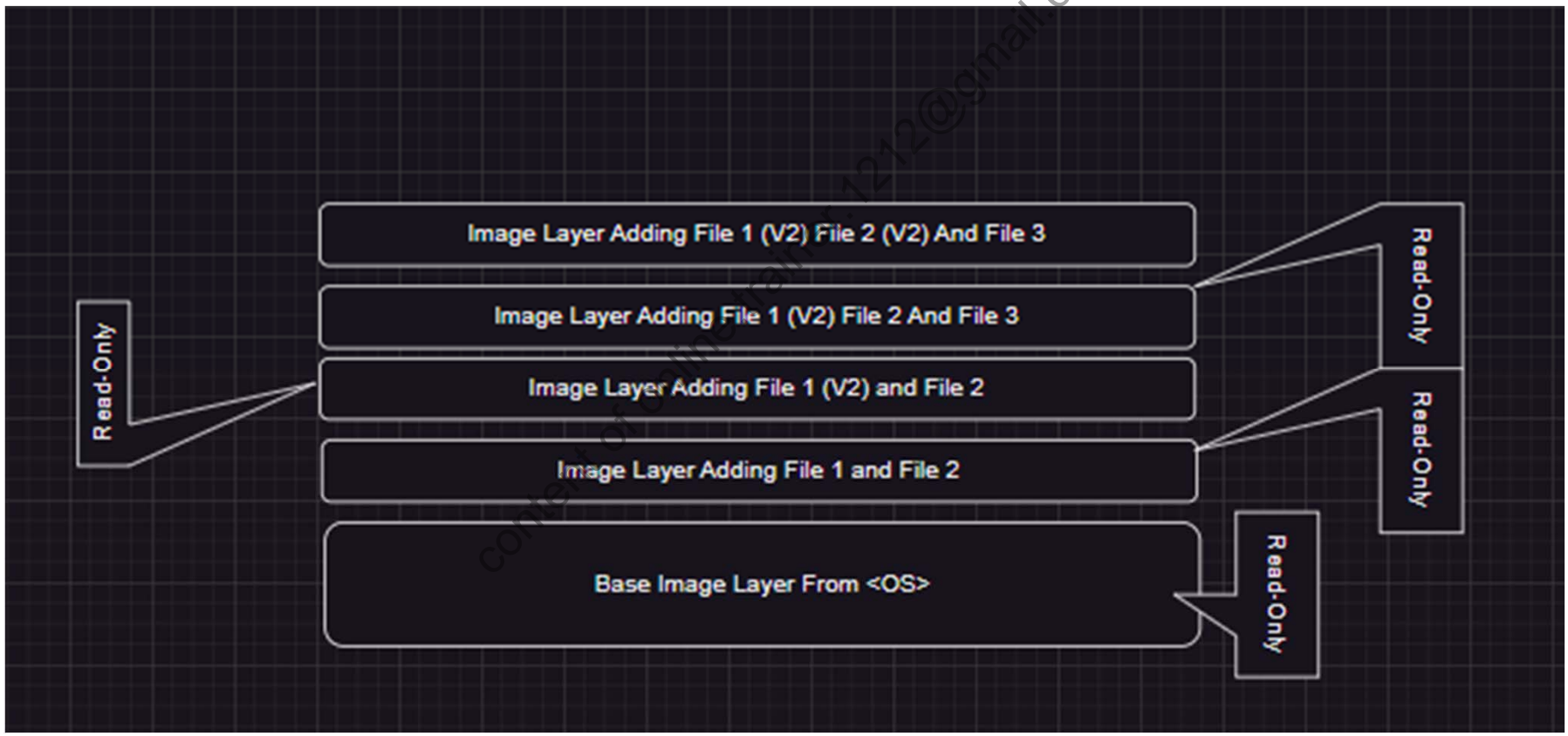
Sub Project 1.5.a

Hands On – Installation of Docker Engine

Working with Docker Commands

- docker --version
- docker search <software name>
- docker run hello-world
- Creating Dockerfile
- docker image build -t first_image .
- docker image ls
- docker ps –a / ps
- docker run -it --entrypoint /bin/bash <IMAGE>
- docker rm <ContainerID>
- docker image rm <IMAGENAME>
- docker run -p 80:8080 <IMAGENAME>
- docker run -it --ulimit nofile=122880:122880 -m 3G --rm -p 80:8080 <IMAGENAME>
- docker system prune -a

Layering in Docker



Creating A Layered Image

- Creating Hello World & Executing the same.

Sub Project 1.5.b

Hands On – Installation of CRI-O

Sub Project 1.5.c

Hands On – Installation Of PODMAN

Summary

Sub Project 1.6

Installing Kubernetes

Summary


CT