KCNA - Kubernetes and Cloud Native Associate

By Riyaz Sayyad

RIZMAXed

START COURSE »

Disclaimer: This content is copyrighted and strictly for personal use only.

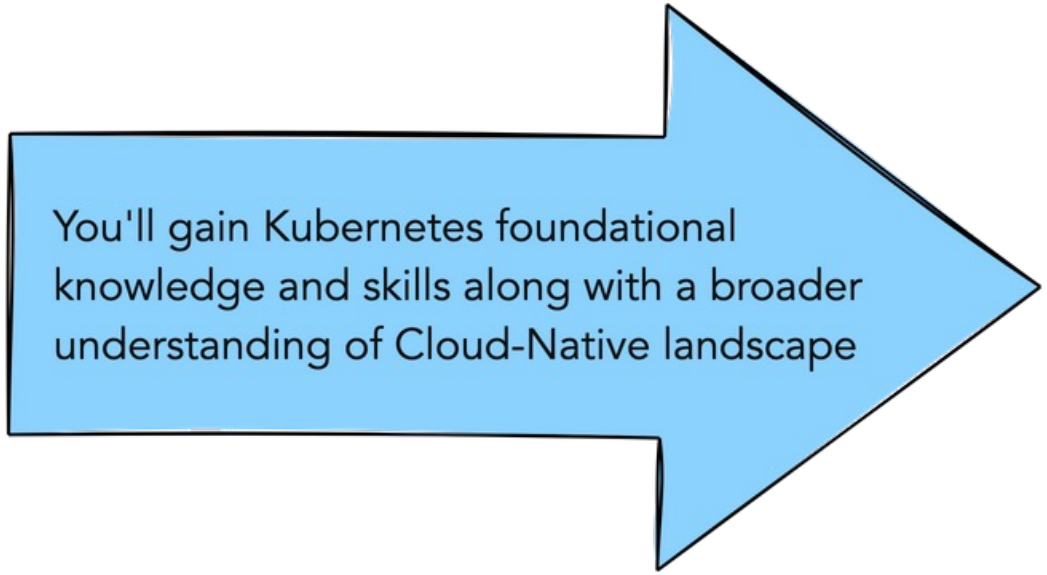
- This content is reserved for people enrolled into the KCNA - Kubernetes and Cloud Native Associate course by Riyaz Sayyad. It is intended for personal use and exam preparation only, please do not share this content.
- All trademarks, logos and brand names are the property of their respective owners.

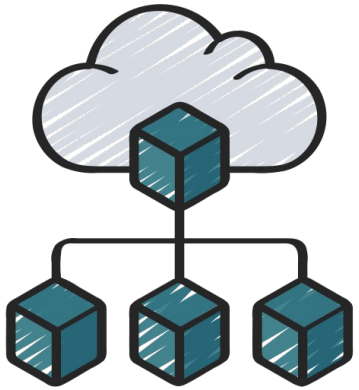
KCNA - Kubernetes and Cloud Native Associate

Hands-on Guide

Welcome!

- Let’s get ready for the KCNA exam





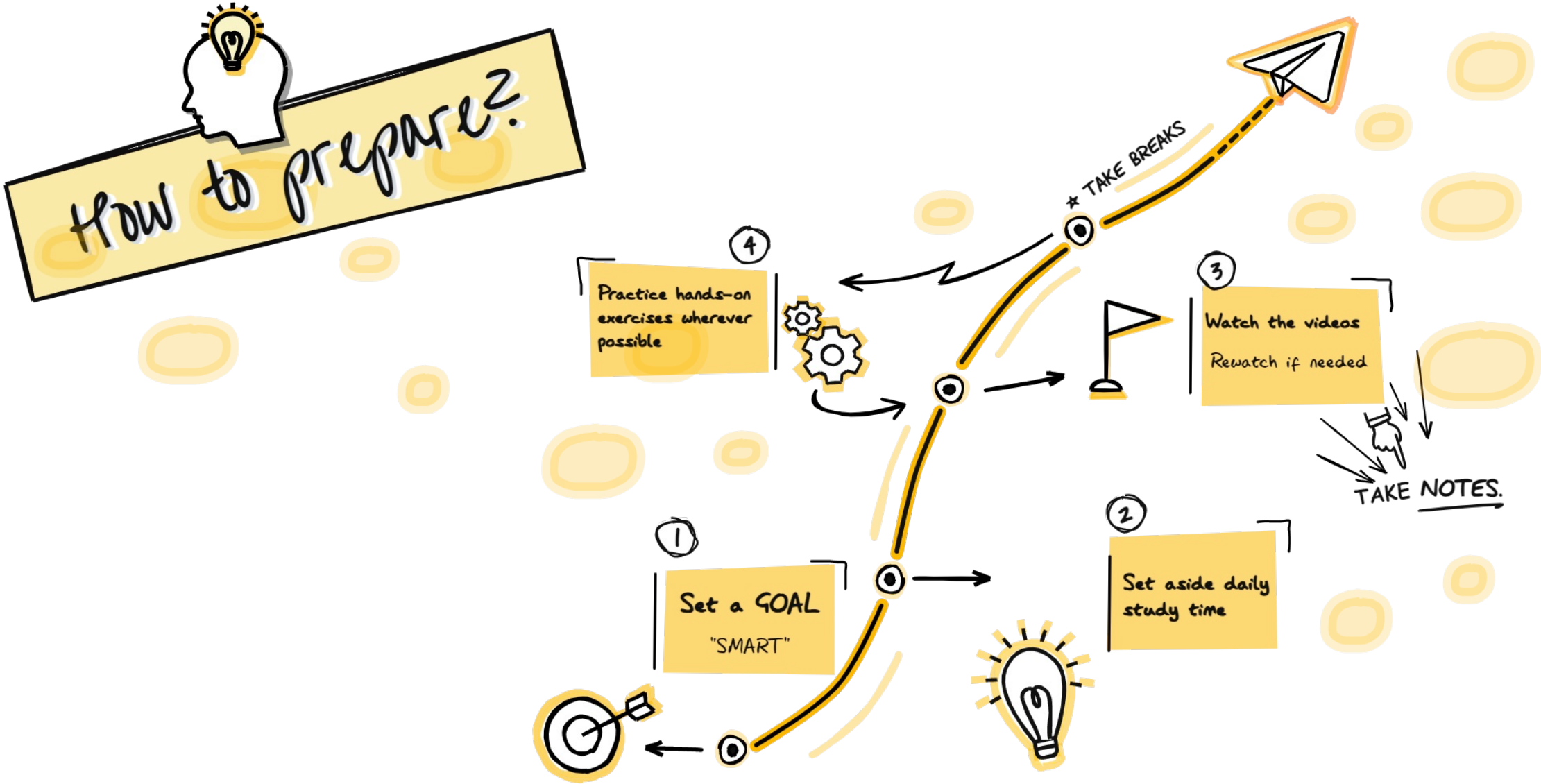
- Recommended IT experience
  - Basic Linux Knowledge (Good to have)

About the exam

- Exam Overview
  - 90 minutes
  - 60 questions (MCQs / MRQs)
  - Passing score – 75%
  - No negative marking
  - Free retake
- Key Focus Areas
  - Kubernetes Fundamentals
  - Container Orchestration
  - Cloud Native Architecture
  - Cloud Native Observability
  - Cloud Native Application Delivery

Put your best foot forward!

RIZMAXed



About me - I'm Riyaz!

RIZMAXed

- KCNA Certified
- 6x AWS Certified
- Over a decade of experience in AWS Cloud and Serverless
  - Built websites, apps, Cloud Native SaaS products
- Best-selling Instructor on Udemy
  - Serverless, Lambda, Databases
  - 100k+ students, 13k+ reviews
  - I love teaching!

Technology Overview

Containers, Kubernetes, and the cloud 🤝

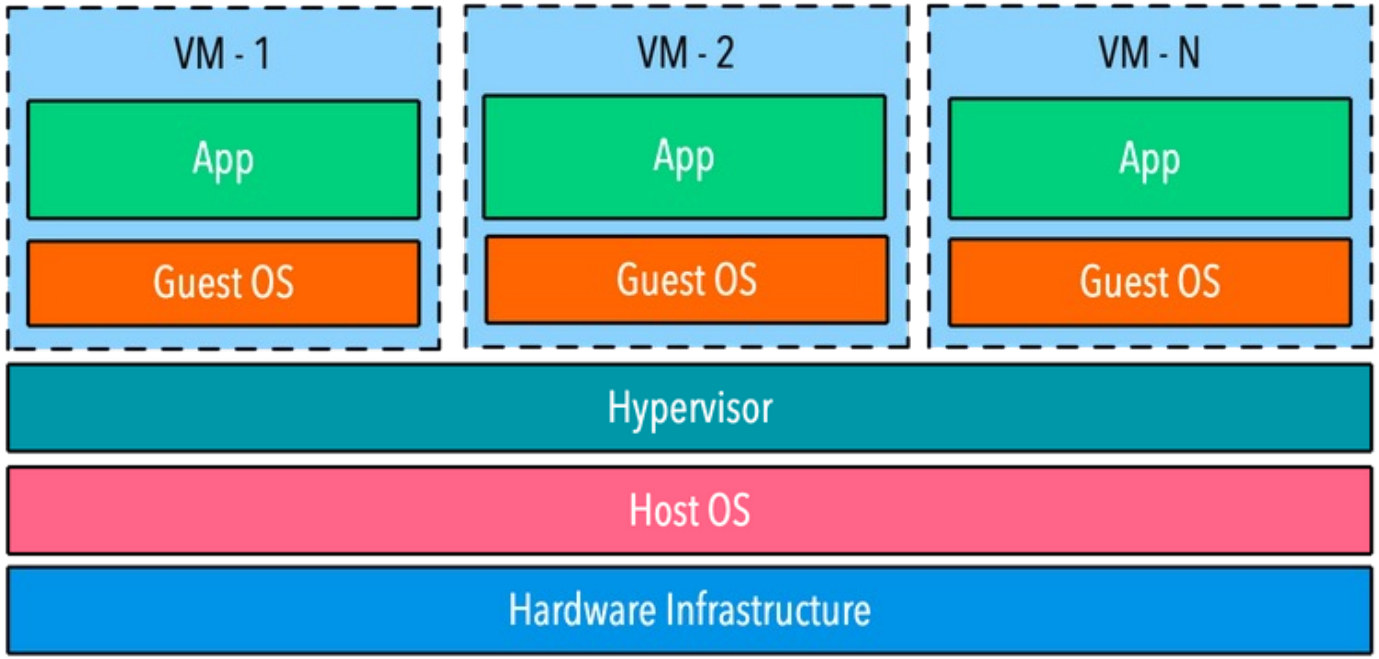
Cloud Computing

RIZMAXed

- On-demand, scalable IT resources
  - Compute, storage, databases, applications, etc.
- Pay-as-you-go (no upfront capital investment)
- Agility – deploy globally in minutes, faster TTM
- Main types of cloud computing – IaaS, PaaS, SaaS
- Cloud deployment models – Public, Private, Hybrid
- Public cloud – AWS, Azure, GCP
- Private cloud
  - Not same as the typical on-premise data center
  - On-premises data center w/ virtualization and resource management tools
- Hybrid = public + private cloud

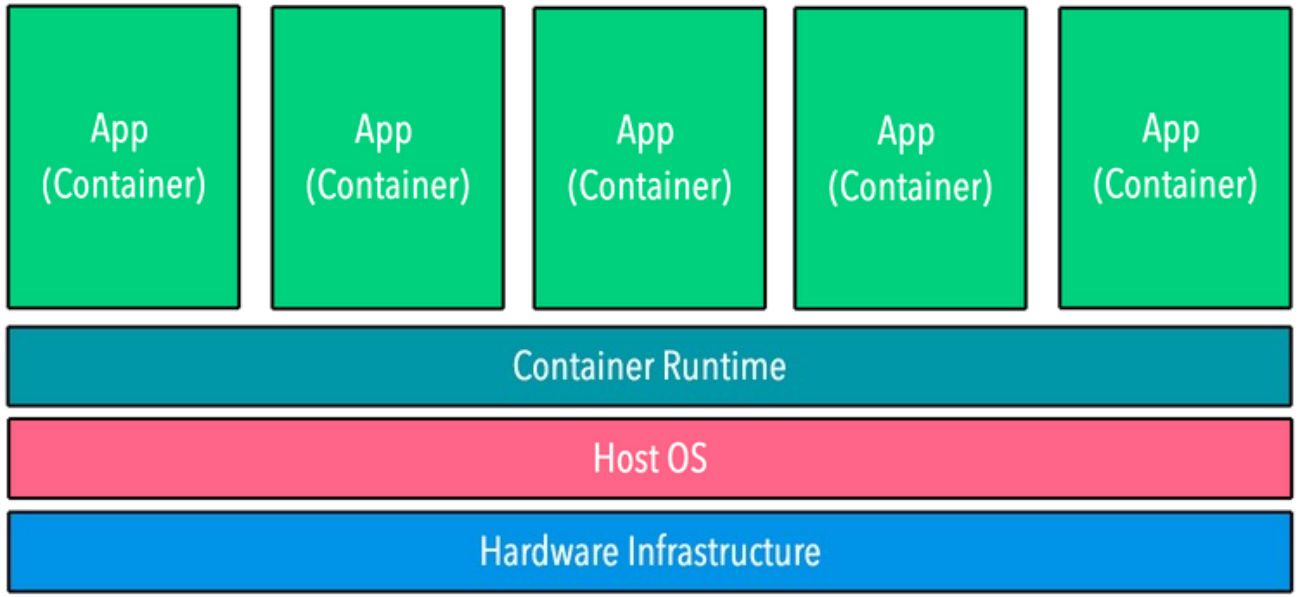
Virtualization

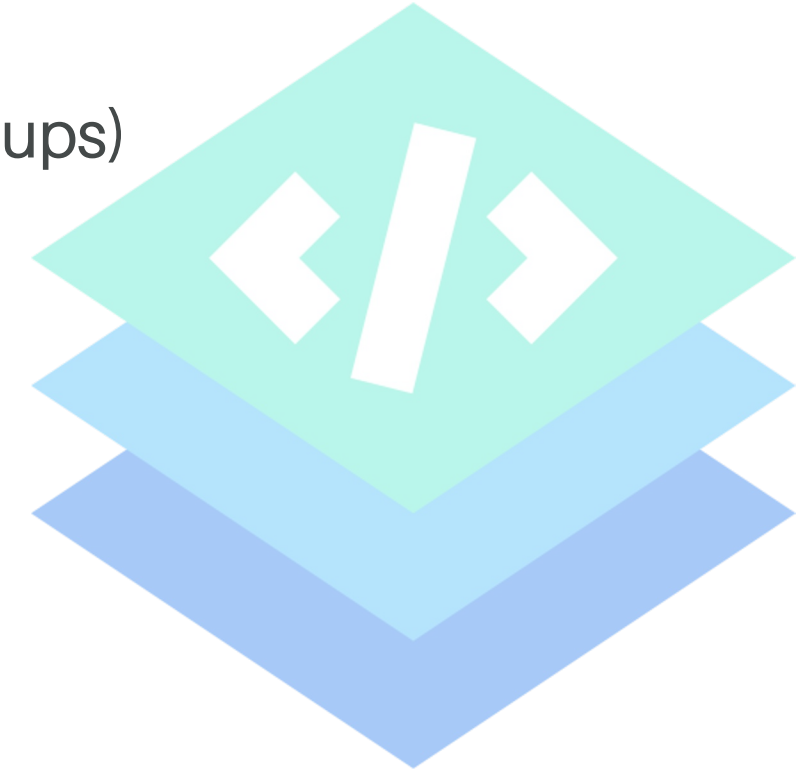
- running multiple operating systems on a single physical machine
- each operating system runs in a completely isolated environment
- managed or orchestrated by a hypervisor



Containerization

- container image is a snapshot of an application + dependencies
- container is a running instance of a container image, each runs as an isolated process
- are portable, smaller in size, offer more efficient resource utilization
- multiple applications can run on fewer virtual machines
- isolation provided by Linux control groups feature (cgroups)



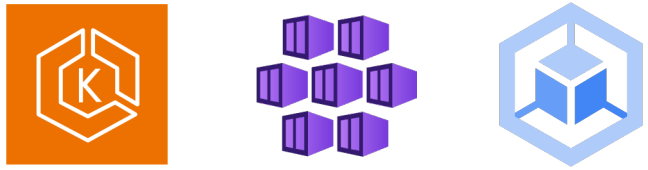


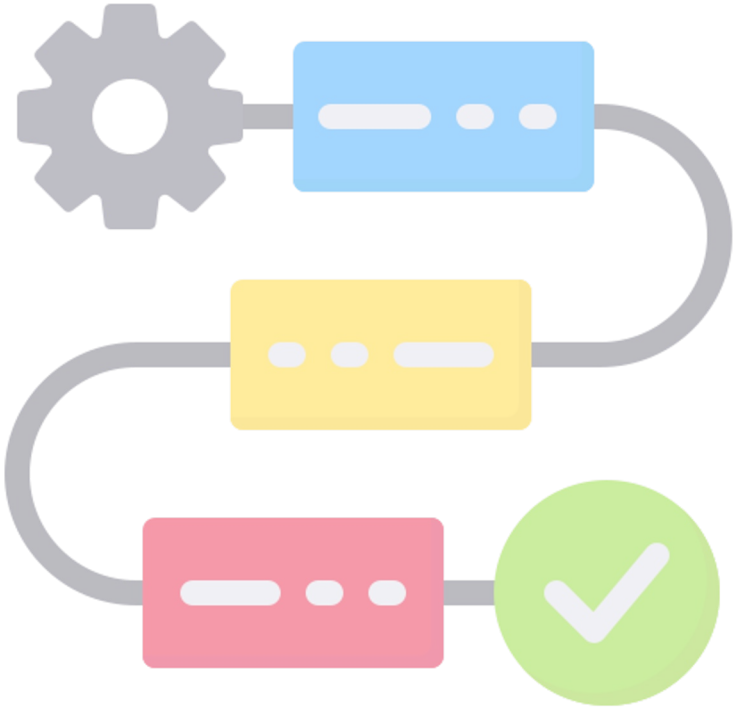
Docker

- provides a set of tools for developing, shipping, and running containerized applications
- a Docker image is defined using a **Dockerfile**
- **Docker Hub** is a central repository where users can store, share, and distribute Docker images
- container runtimes created by Docker
  - **runc** (lightweight, and portable)
  - **containerd** (industry-standard, simple, robust, and portable)
- runc was donated to OCI to help establish standardization
  - **OCI** = **Open Container Initiative**
- containerd was donated to CNCF
  - **CNCF** = **Cloud Native Computing Foundation**

Container Orchestration

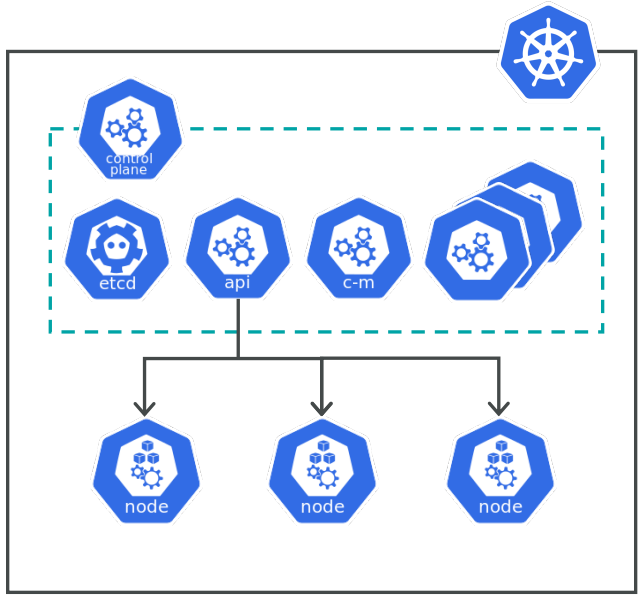
- automated management of containerized applications
  - including deployment, scaling, networking, and availability
- streamlines deploying and managing containers at scale
  - improves efficiency, reliability, and scalability
- Orchestration Platforms
  - Kubernetes
  - Docker Swarm
  - Apache Mesos
  - Amazon ECS
- Managed Kubernetes
  - EKS
  - AKS
  - GKE





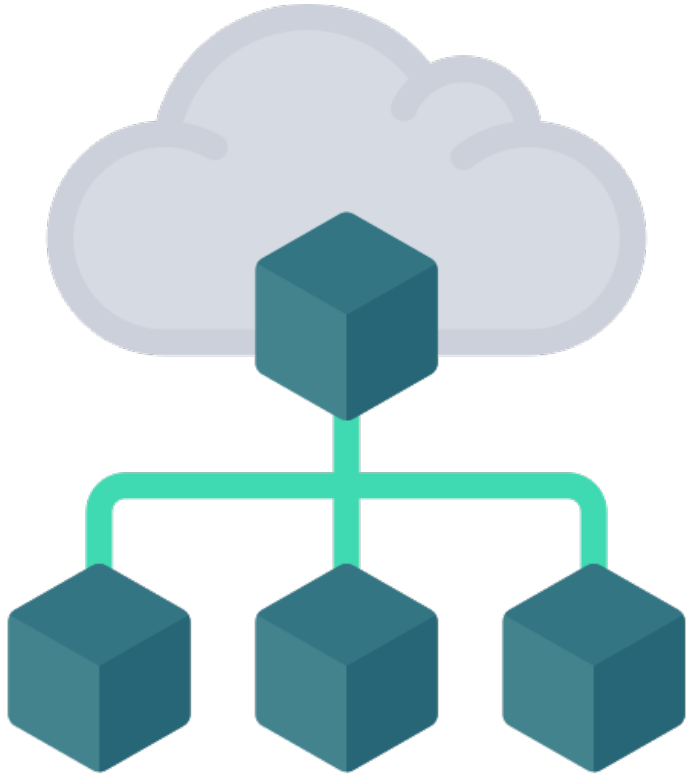
Kubernetes (K8s)

- open-source container orchestration platform
- developed by Google and donated to CNCF in 2014
- incubating 2016, graduated 2018
- Key features
  - automates the container scheduling and management
  - automatic scaling and self-healing
  - service discovery and load balancing
  - rolling updates and rollbacks
- master node controls and manages the cluster (control-plane)
- worker nodes host the containers and run the applications
- etcd acts as a distributed key value store to store cluster config data



Cloud Native Computing

- = design and deployment of applications that fully leverage the benefits of cloud computing models (public, private, or hybrid)
- => scalable, resilient, agile, cost-effective, portable applications
- Key guiding principles
  - containerization and orchestration
  - microservices architecture
  - continuous integration/delivery/deployment
  - DevOps, GitOps, FinOps
- Use cases
  - distributed web apps
  - IoT and edge computing
  - big data and analytics



Environment Setup

- Install **Docker Desktop**
- Set up Multi-node Kubernetes cluster using **Minikube**
- Install **VS Code** (optional)

JSON 101

RIZMAXed

JAVASCRIPT OBJECT NOTATION

- Simple and popular format
- Used to exchange data over the APIs

Hands-on Demo

RIZMAXed

YAML 101

RIZMAXed

YAML AIN'T MARKUP LANGUAGE

- Data serialization standard
- Human readable and computationally powerful
- Simple and popular text format
- Commonly used for writing templates and configuration files
- Do not use TAB characters for indentation. Use series of spaces instead. Modern IDEs like VS Code take care of this.

cfn-template.yaml

```
  1   Metadata:
  2     'AWS::CloudFormation::Interface':
  3   >   ParameterGroups: …
 69   >   ParameterLabels: …
170   > Parameters: …
492   > Conditions: …
630   > Mappings: …
704   Resources:
705   >   HANAIAMProfile: …
715   >   HanaDataVolume: …
757   >   HanaLogVolume: …
836   >   HanaOthersVolume: …
880     HanaSingleInstance:
881       Type: 'AWS::EC2::Instance'
882       Properties:
883   >     BlockDeviceMappings: …
896         IamInstanceProfile: !Ref HANAIAMProfile
897         ImageId: !Ref HANAAMIID
898         InstanceType: !Ref HANAInstanceType
899         KeyName: !Ref KeyName
900         Monitoring: true
901   >     NetworkInterfaces: …
904   >     Tags: …
927   >     UserData: !Base64 …
1296  >   AutoRecoverAlarm: …
```

Hands-on Demo

RIZMAXed

Linux Commands 101

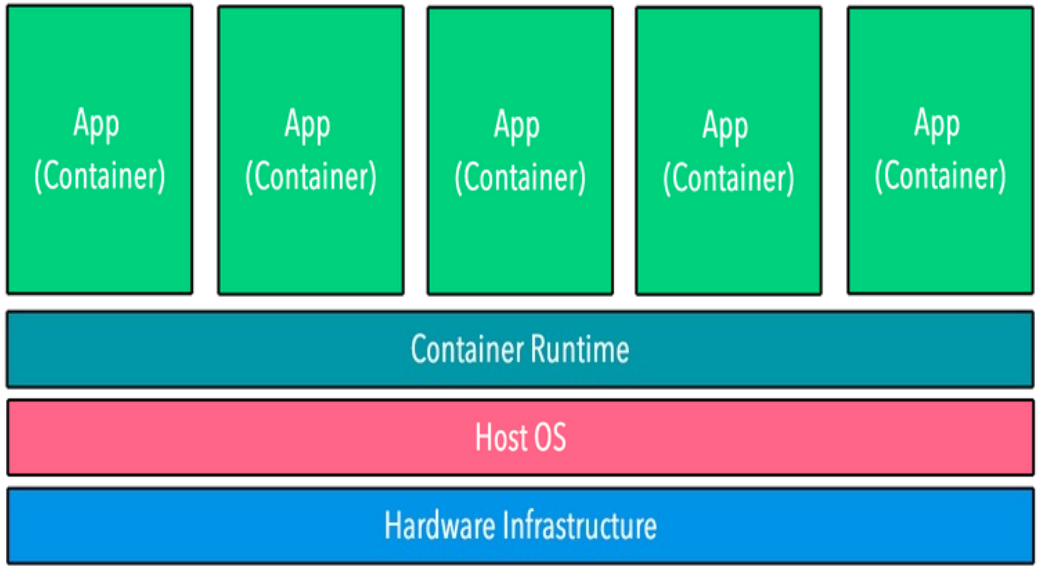
- Quick refresher on Linux commands
  - we'll spin up Ubuntu container using Docker
- We'll cover
  - Basic navigation with Linux CLI
  - File and directory manipulation
  - Search and replace operations
  - File viewing and editing
  - Other common commands

Working with Containers

Docker fundamentals

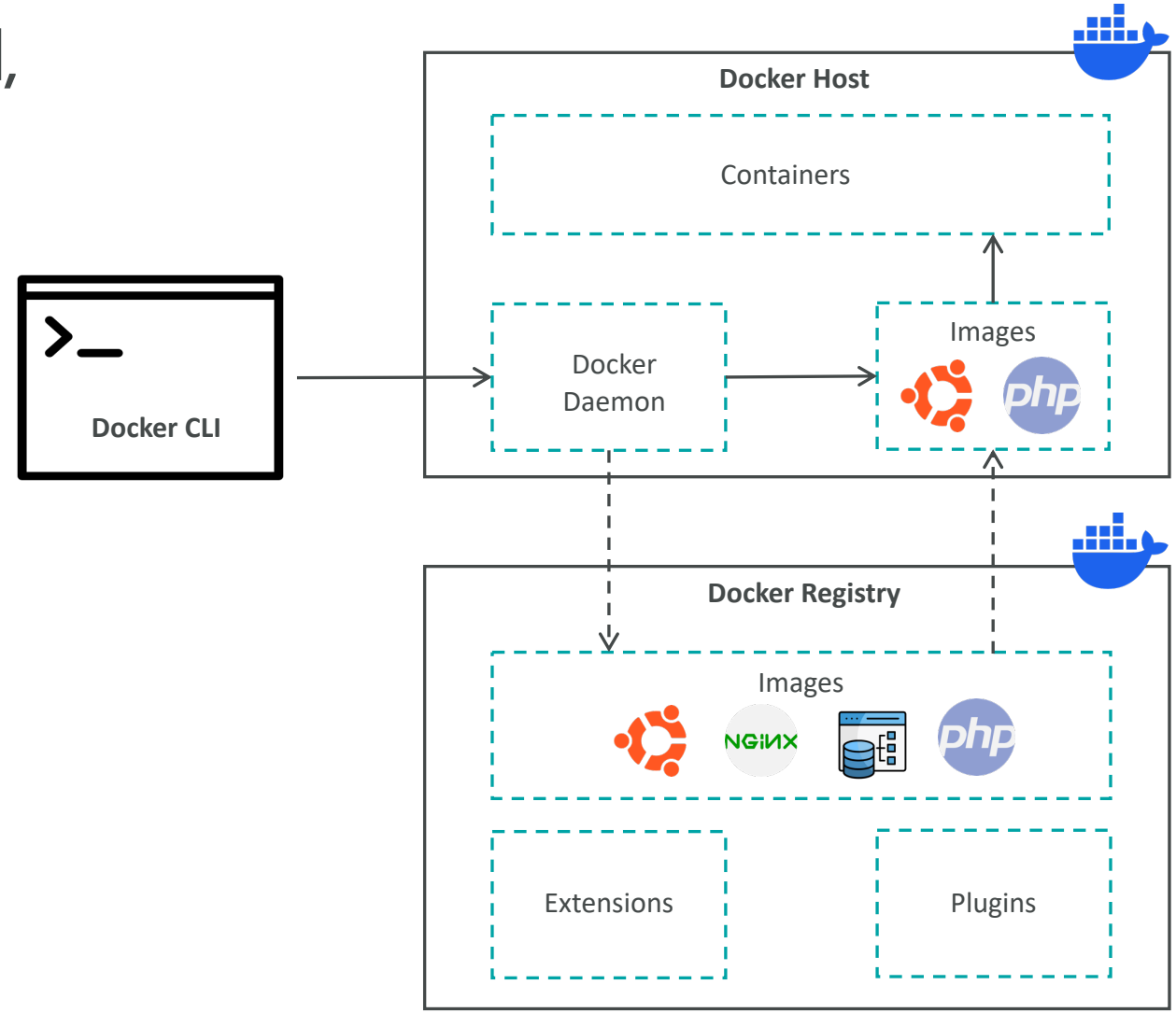
Container Basics

- = packaged applications
  - application + dependencies + libs + config
  - consistent behavior across environments
  - lightweight, standalone, and portable
  - containers share the kernel of the OS
- Why use them?
  - Consistency, Isolation, Scalability
- Docker
  - Founded in 2010 with name dotCloud
  - Renamed and open-sourced in 2013
  - Most popular container platform
  - Easily create, deploy, manage containers



How Docker Works

- Docker Engine helps you build, ship, and run containers
- Key components
  - Docker Daemon
    - runs in background
  - Docker CLI
    - allows you to interact with Docker
  - Docker Registry
    - repository for Docker images
    - aka Docker Hub



Hands-on Demo - Docker Basics

RIZMAXed

Docker Commands Cheat Sheet

RIZMAXed

- **docker run <image>**
  - Create and start a new container from an image.
- **docker ps**
  - List all running containers.
- **docker ps -a**
  - List all containers (including stopped ones).
- **docker stop <container>**
  - Stop a running container.
- **docker start <container>**
  - Start a stopped container.
- **docker restart <container>**
  - Restart a container.
- **docker rm <container>**
  - Remove a stopped container.
- **docker images**
  - List all available images.
- **docker rmi <image>**
  - Remove an image.
- **docker pull <image>**
  - Download an image from a repository.
- **docker exec -it <container> <command>**
  - Run a command inside a running container.
- **docker logs <container>**
  - View logs from a container.
- **docker inspect <container>**
  - Display detailed information about a container.
- **docker network ls**
  - List all Docker networks.
- **docker volume ls**
  - List all Docker volumes.
- **docker build -t <tag> <path>**
  - Build an image from a Dockerfile.
- **docker login**
  - Log in to a Docker registry.
- **docker push <image>**
  - Push an image to a registry.
- **docker system info**
  - Display system-wide information about the Docker installation
- **docker <command> --help**
  - Look up help information on any docker command

Docker Port Mapping

RIZMAXed

- Publish all exposed ports to random ports

```
riyaz@RizmaxMBP ~ % docker run --rm -d -P nginx:latest
73c5df201d68c8ac99421b3672121d01be49f3af02910bf5813d6ef6304eba7b
riyaz@RizmaxMBP ~ %
riyaz@RizmaxMBP ~ % docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                     NAMES
73c5df201d68   nginx:latest   "/docker-entrypoint.…"   6 seconds ago   Up 5 seconds   0.0.0.0:55000->80/tcp   kind_wiles
riyaz@RizmaxMBP ~ %
```

- Publish container port(s) to specific port(s) on host

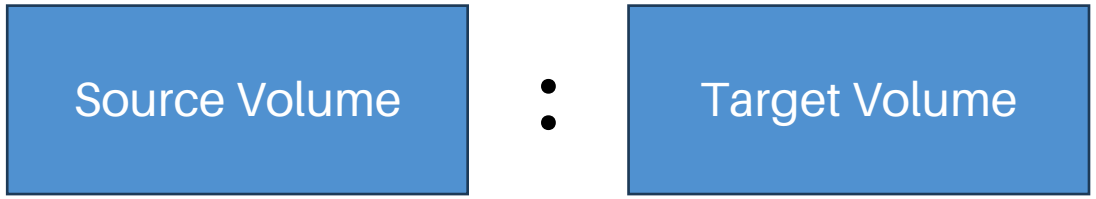
```
riyaz@RizmaxMBP ~ % docker run --rm -d -p 8888:80 nginx:latest
16bb0d90b84c1e0e507fc224353f52abe76c063541d8a0d41d74dd1663661d0d
riyaz@RizmaxMBP ~ %
riyaz@RizmaxMBP ~ % docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                     NAMES
16bb0d90b84c   nginx:latest   "/docker-entrypoint.…"   39 seconds ago   Up 39 seconds   0.0.0.0:8888->80/tcp   charming_gr
riyaz@RizmaxMBP ~ %
riyaz@RizmaxMBP ~ %
```

Docker Volume Binding

RIZMAXed

- Mount a file or directory on the host machine into a container

```
riyaz@RizmaxMBP static-website % docker run --rm -d -p 8888:80 -v /Users/riyaz/sandbox/projects/static-website:/usr/share/nginx/html nginx:latest
413e11b46e6e3a327824ab13644d2f85bed5877dafbc2063ab0406e938a1d4c1
riyaz@RizmaxMBP static-website % docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                   NAMES
413e11b46e6e   nginx:latest   "/docker-entrypoint.…"   12 seconds ago   Up 11 seconds   0.0.0.0:8888->80/tcp   competent_franklin
riyaz@RizmaxMBP static-website %
```



Create a Container Image using Dockerfile

RIZMAXed

Dockerfile

Dockerfile > ...

```
1   FROM nginx:latest
2   COPY index.html /usr/share/nginx/html
3   EXPOSE 80
4   CMD [ "nginx", "-g", "daemon off;" ]
5
6
7
8
```

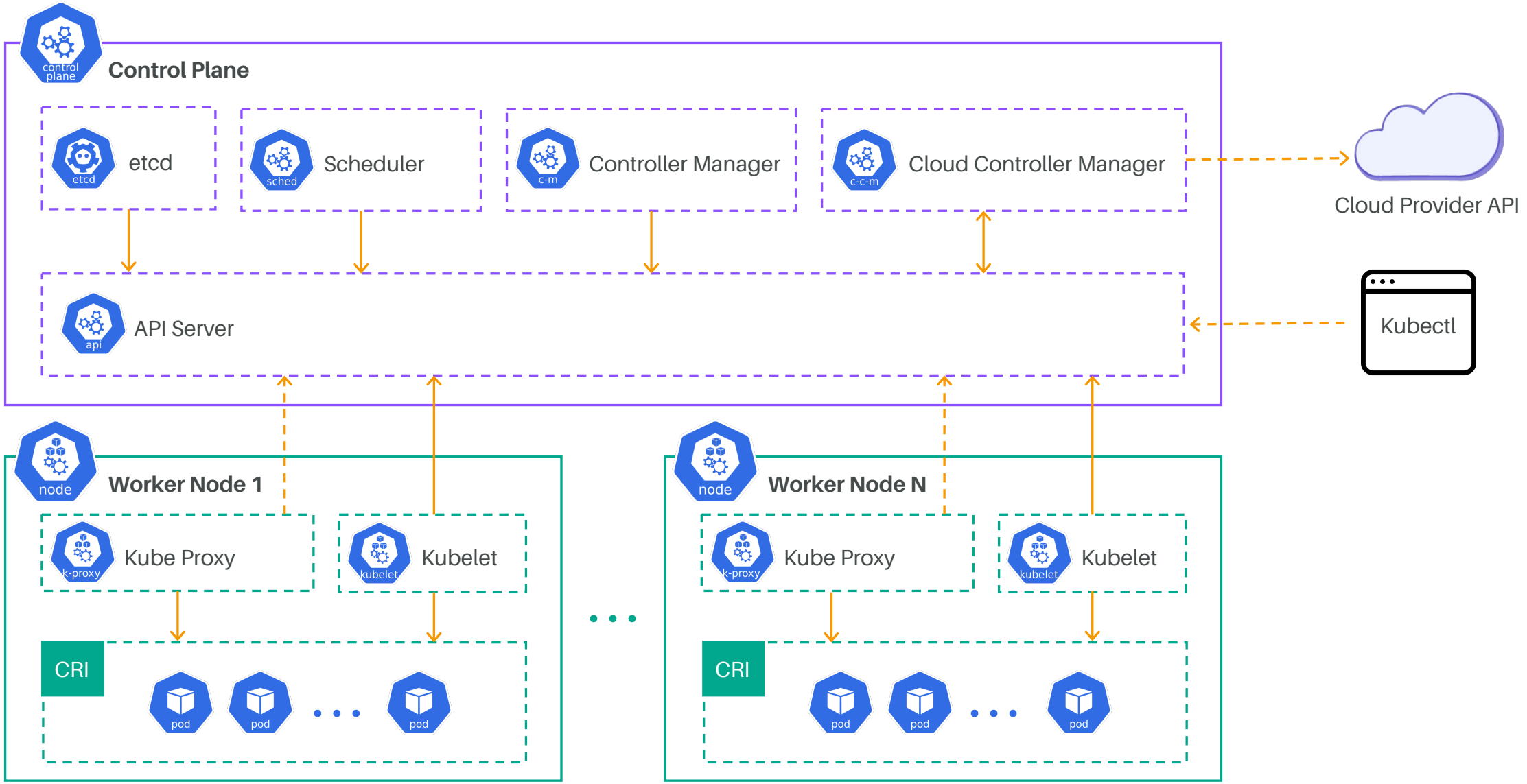
```
$ docker build -t myapp .
```

Kubernetes Overview

Get started with Kubernetes

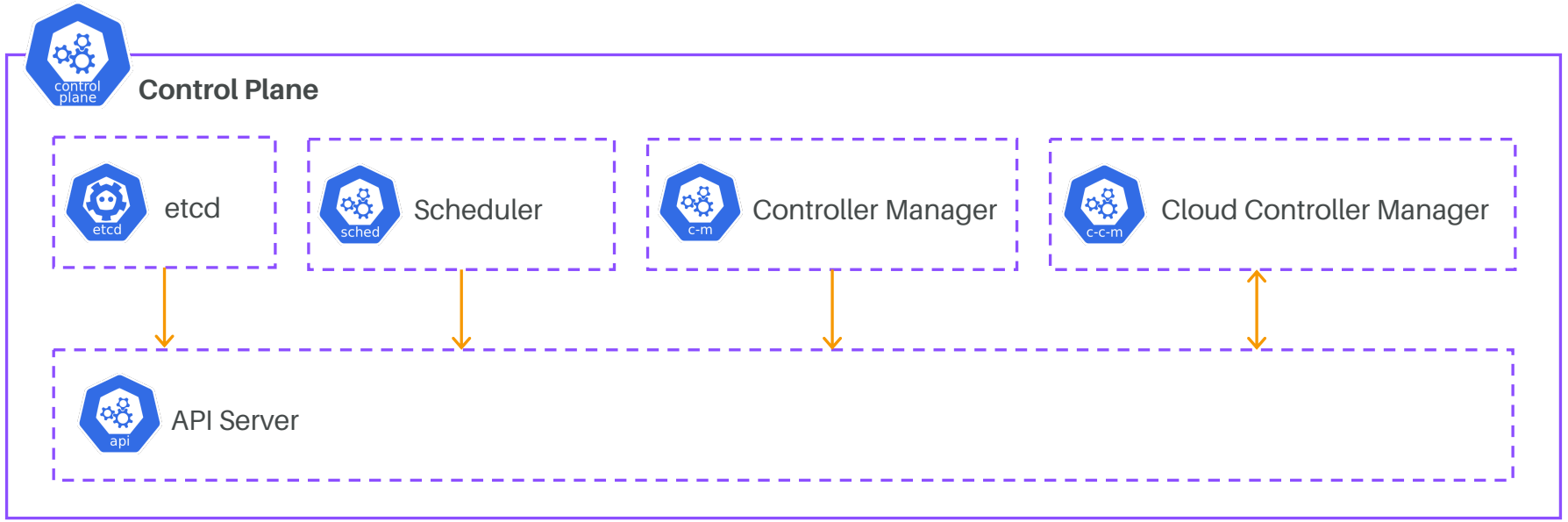
Kubernetes Architecture

RIZMAXed



Control Plane

RIZMAXed



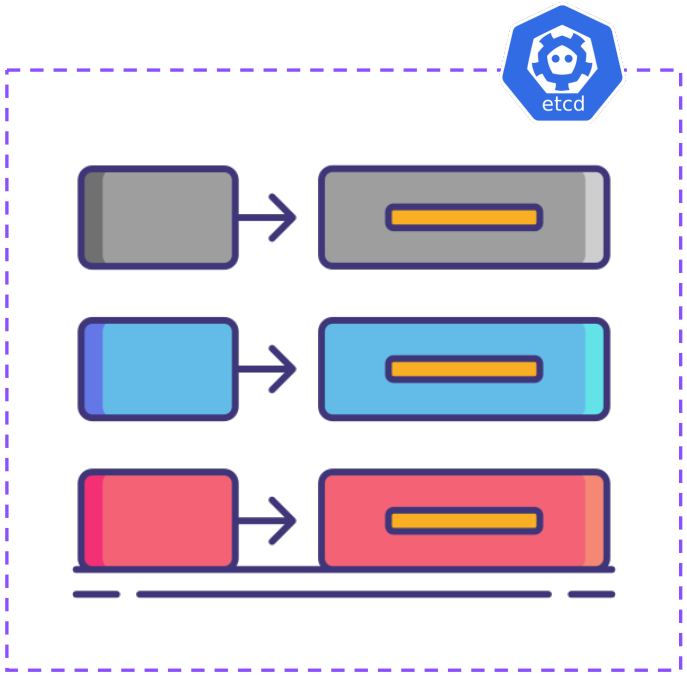
- Brain of the k8s cluster
- Manages and maintains the desired state of the cluster
  - resource management, scheduling, scaling, high-availability
- One or more control plane nodes
  - Single Control Plane Node – smaller or dev clusters
  - Highly Available Control Plane – production-grade clusters

Kube API Server

- Primary interface for interacting with a k8s cluster
- RESTful API – supports HTTP CRUD operations
- Resources are organized into API Groups
  - /api/v1 – core group `apiVersion: v1`
  - /apis/$GROUP_NAME/$VERSION- named groups `apiVersion: apps/v1`
- Custom resources can be defined using CRD
  - **CRD = Custom Resource Definition**
- Authentication – tokens, certificates, etc.
- Authorization – RBAC, ABAC, etc.
- Reconciles the desired state defined in the YAML or JSON manifests

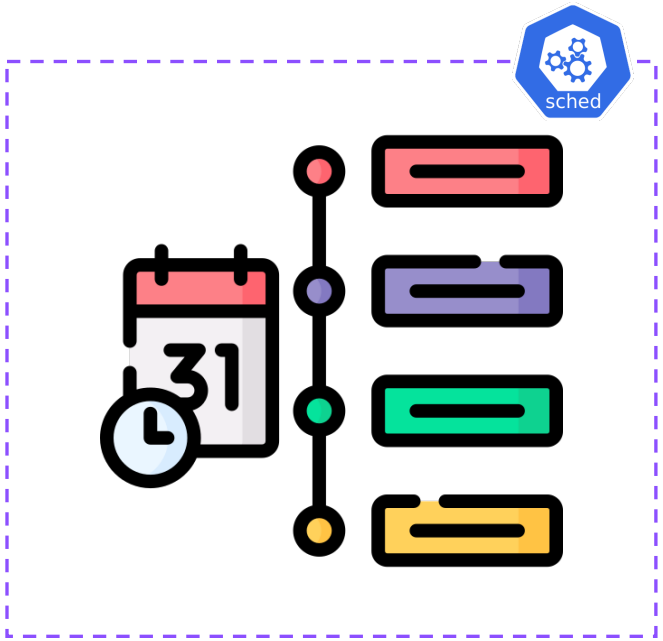
etcd

- Distributed key-value store
- Primary datastore to store all cluster data
  - config data, state, and metadata of k8s objects
  - stores ConfigMaps and Secrets
- Reliable and consistent
  - highly available and fault-tolerant
  - implements Raft consensus algorithm
- Offers TLS encryption and authentication features
- Typically placed on control plane nodes
  - one etcd instance on each control plane node
- Can be placed on dedicated etcd cluster
  - multiple etcd instances in the etcd cluster



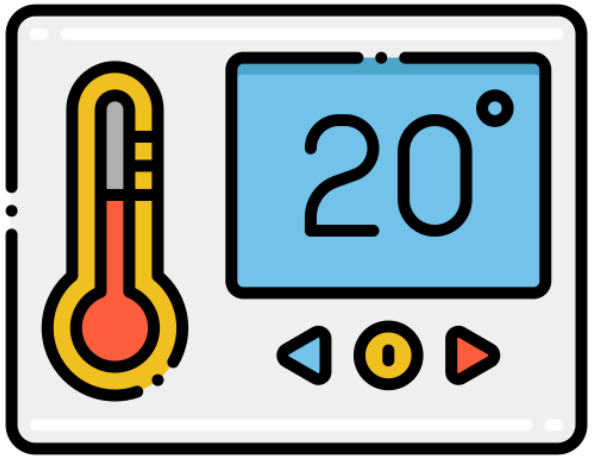
Kube Scheduler

- Scheduling
  - process to determine the best node to run a pod
- Kube Scheduler assigns pods to nodes
  - ensuring optimal resource utilization and
  - meeting application requirements
- 3-step operation
  - filtering – finds suitable nodes as per resource requirements
  - scoring – ranks found nodes
  - assignment – assigns pod to the highest-ranking node
- Pod placement can be controlled using
  - node name, node selectors, affinities, and taints/tolerations

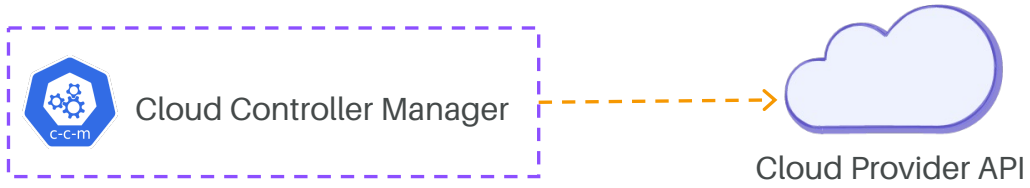


Controller Manager

- Controllers
  - implement control loops (like a thermostat in a room)
  - continuously watch the cluster state via the API server
  - act to reconcile the current state with the desired state
- Controller Manager operates different controllers
  - Deployment Controller
  - Job controller etc.
- Can define custom controllers



Cloud Controller Manager (CCM)



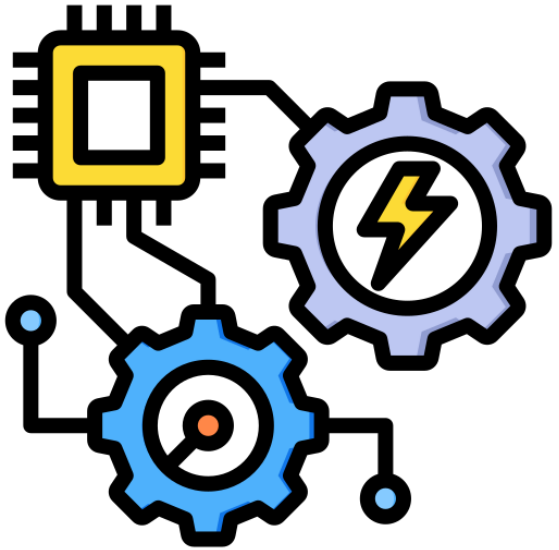
- bridge between Kubernetes and cloud providers
  - AWS, Azure, GCP, and more
- Abstracts cloud provider APIs
  - to manage resources like load balancers, volumes, instances
  - allows any cloud provider to integrate with k8s
- Key functions
  - Resource provisioning – persistent volumes, load balancers etc.
  - Node management – create/delete instances, auto-scaling
  - Networking – communication between pods and external services

Kube Proxy

- Runs on all worker nodes
- Acts as a Network Proxy
  - managing network connectivity to k8s services within the cluster
- Abstracts the internal network details
  - providing a consistent view of services to applications
- Enforces network policies
  - to control traffic between pods and services
- Implements load balancing for services
  - routing traffic to healthy pods
- Facilitates service discovery

Kubelet

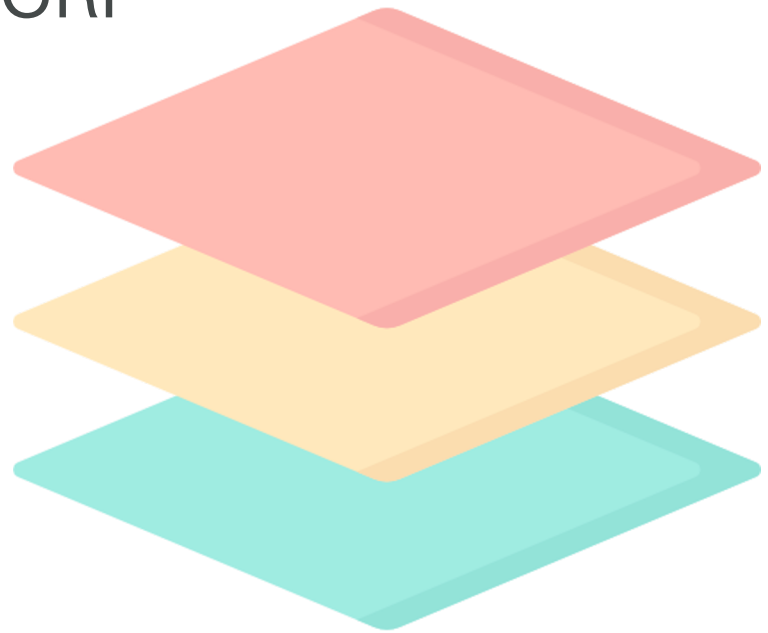
- Primary **node agent** that runs on each node
- Ensures pods are instantiated, running, and terminated
  - according to the desired state defined in pod specifications
- Handles resource allocation to pods
  - CPU, memory, and storage
- Monitors node and container health
  - uses liveness probes to check container health
    - e.g. HTTP Probe, Exec Probe, TCP probe
  - restarts containers that fail the liveness probe
- Interfaces with container runtime to manage containers within pods



Container Runtime Interface (CRI)

- Container runtime runs containers within pods
- CRI is an abstraction layer on top of Container Runtime
  - enables the kubelet to use a wide variety of container runtimes
  - => flexibility and interoperability across runtime environments
- Supports any container runtime that implements CRI
  - e.g. containerd, CRI-O, Kata Containers, Firecracker
- Docker (dockershim) is no longer supported

containerd cri-o kata containers Firecracker



Kubeconfig

- Configuration file
  - cluster info, auth credentials, and context settings
- Clusters
  - cluster endpoint, Server CA info
- Users
  - user authentication creds (tokens, certs)
- Contexts
  - defines cluster access configuration
  - cluster, user, namespace, + other settings
- Typically stored in
  - **~/.kube/config**
  - or in **KUBECONFIG** env var

```
Rizmax ~ $ cat ~/.kube/config
apiVersion: v1
clusters:
- cluster:
    certificate-authority: /Users/riyaz/.minikube/ca.crt
    extensions:
    - extension:
        last-update: Wed, 13 Mar 2024 07:03:53 EDT
        provider: minikube.sigs.k8s.io
        version: v1.32.0
      name: cluster_info
    server: https://127.0.0.1:59873
  name: minikube
contexts:
- context:
    cluster: minikube
    extensions:
    - extension:
        last-update: Wed, 13 Mar 2024 07:03:53 EDT
        provider: minikube.sigs.k8s.io
        version: v1.32.0
      name: context_info
    namespace: default
    user: minikube
  name: minikube
current-context: minikube
kind: Config
preferences: {}
users:
- name: minikube
  user:
    client-certificate: /Users/riyaz/.minikube/profiles/minikube/client.crt
    client-key: /Users/riyaz/.minikube/profiles/minikube/client.key
```

Semantic Versioning (SemVer)

- Kubernetes follows **SemVer**
  - Kubernetes Releases
  - API versions
  - Resource Definitions
  - Component Versions
- Kubernetes releases follow **MAJOR.MINOR.PATCH** format
  - major releases may introduce breaking changes
    - 1.0.0, 2.0.0
  - minor releases introduce new features
    - in a backward-compatible manner
    - 1.19.0, 1.20.0
  - patch releases contain bug fixes and minor enhancements
    - 1.19.1, 1.19.2
- API versions may be labeled as alpha, beta, or stable

Kubernetes Resources

RIZMAXed

- = API objects in k8s
  - Pod, Service, Deployment, ReplicaSet, etc.
- Scoped by **namespaces**
  - divide cluster resources into virtual clusters
    - isolate resources of different projects / teams / environments within a cluster
  - can set resource quotas at namespace level
    - CPU, memory, etc.
  - if no namespace is specified, **default** namespace is used
- Defined using YAML or JSON manifests
  - => **Declarative, IaC**
- Can also be created using CLI commands
  - => **Imperative**
- Use CRD to define custom resources
  - **CRD = Custom Resource Definition**

```
$ kubectl api-resources
```

```
[Rizmax ~ $ kubectl api-resources
NAME                              SHORTNAMES   APIVERSION   NAMESPACED   KIND
bindings                                       v1           true         Binding
componentstatuses                 cs           v1           false        ComponentStatus
configmaps                        cm           v1           true         ConfigMap
endpoints                         ep           v1           true         Endpoints
events                            ev           v1           true         Event
limitranges                       limits       v1           true         LimitRange
namespaces                        ns           v1           false        Namespace
nodes                             no           v1           false        Node
persistentvolumeclaims            pvc          v1           true         PersistentVolumeClaim
persistentvolumes                 pv           v1           false        PersistentVolume
pods                              po           v1           true         Pod
podtemplates                                   v1           true         PodTemplate
replicationcontrollers            rc           v1           true         ReplicationController
resourcequotas                    quota        v1           true         ResourceQuota
secrets                                        v1           true         Secret
serviceaccounts                   sa           v1           true         ServiceAccount
services                          svc          v1           true         Service
mutatingwebhookconfigu                         admissionreg false        MutatingWebhookConfigu
validatingwebhookconfi                         admissionreg false        ValidatingWebhookConfi
customresourcedefiniti            crd,crds     apiextension false        CustomResourceDefiniti
apiservices                                    apiregistrat false        APIService
controllerrevisions                            apps/v1      true         ControllerRevision
daemonsets                        ds           apps/v1      true         DaemonSet
deployments                       deploy       apps/v1      true         Deployment
replicasets                       rs           apps/v1      true         ReplicaSet
statefulsets                      sts          apps/v1      true         StatefulSet
```

Hands-on Demo

RIZMAXed

Imperative vs Declarative

RIZMAXed

- Imperative approach
  - specify exact sequence of steps to provision resources
  - give explicit instructions for implementation
  - e.g. creating resources with CLI commands

```
$ kubectl run mypod --image=nginx:stable
```

- Declarative approach – IaC, preferred
  - specify desired end state
  - define desired state using YAML manifests
  - k8s reconciles current state with desired state
  - required top-level attributes
    - apiVersion
    - kind
    - metadata
    - spec

first-pod.yaml

first-pod.yaml > {} spec > [ ] containers > {} 0 > [ ] ports > {} 0

io.k8s.api.core.v1.Pod (v1@pod.json)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
  labels:
    name: mypod
spec:
  containers:
    - name: nginx
      image: nginx:stable
      resources:
        limits:
          memory: "128Mi"
          cpu: "500m"
      ports:
        - containerPort: 80
```

Pods

- The smallest deployable unit in k8s
- One pod runs one or more containers
  - typically, one app container => single-container pod
  - can have multiple app containers, rare
- Multi-container pods
  - app container + helper or sidecar containers
- All containers in a pod share the same
  - IP address, hostname, storage volumes
- Pod Lifecycle
  - Pending > Running > Succeeded
  - Failed, Unknown
- Restart Policy
  - kubelet is designed to restart exited containers
  - restartPolicy: Always | OnFailure | Never
  - default restartPolicy = Always
  - can be defined at pod-level or container-level

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
  labels:
    name: mypod
spec:
  containers:
    - name: nginx
      image: nginx:stable
      resources:
        limits:
          memory: "128Mi"
          cpu: "500m"
      ports:
        - containerPort: 80
  restartPolicy: OnFailure
```

Labels and Annotations

RIZMAXed

- Labels
  - key-value pairs attached to k8s objects
  - for identification, grouping, and selection
  - enable flexible querying, filtering, and organizing of resources
  - used with selectors for resource targeting

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: myapp
    environment: production
  annotations:
    description: "Frontend for production environment"
    buildVersion: "v1.2.3"
    imageregistry: "https://hub.docker.com/"
    kubernetes.io/change-cause: "Bug fix 871"
```

- Annotations
  - additional metadata attached to k8s objects
  - for providing extra information
  - for documentation, debugging, or tooling purposes
  - limit the use of annotations to essential metadata to avoid cluttering

ReplicaSet

- Ensures that a specified # of identical pods are running at any given time
- => high availability and scalability
- Used for **stateless** apps
- Can scale down to zero
- selector is used to identify pods to run replicas
- Deployment is k8s object used to manage ReplicaSets
- Recommended to use Deployments
- Use ReplicaSets only if you need to manage pod replicas directly

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
        - name: myapp-container
          image: myapp-image
```

Deployment

- A Deployment manages ReplicaSets
- declarative updates to pods
- => high availability and scalability
- + updates and rollbacks
- Update Strategies
  - RollingUpdate (default)
  - Recreate
  - use maxUnavailable and maxSurge params to control behavior in case of rolling updates
- Can implement canary deployments by creating multiple deployments
- Deployments are **stateless**

```
$ kubectl apply -f myapp-deployment.yaml
$ kubectl scale deployment/myapp-deployment --replicas=10
$ kubectl rollout undo deployment/myapp-deployment
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
        - name: myapp-container
          image: myapp-image
```

Init and Sidecar Containers

RIZMAXed

- Init Container
  - runs before the app containers start
  - to run any utilities or setup scripts
    - database schema initialization
    - data pre-processing
    - configuration preparation
  - always run to completion
  - must complete successfully before the next one starts
- Sidecar Container
  - run alongside the main app container
  - ignores the Pod-level restartPolicy
  - to provide additional functionality
    - logging, monitoring, or proxying for the main app
    - e.g. Prometheus as a sidecar to collect metrics

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  labels:
    app: myapp
spec:
  replicas: 2
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
        - name: myapp-container
          image: myapp-image
      initContainers:
        - name: init-setup
          image: busybox
          command: ["sh", "-c", "wget -O /config/config.yaml http://config-server/config.yaml"]
        - name: wait-for-db
          image: busybox
          command: ["sh", "-c", "until nc -z -v -w30 db-hostname 3306; do echo 'Waiting for DB connection...'; sleep 5; done"]
        - name: logshipper
          image: busybox
          restartPolicy: Always
          command: ['sh', '-c', 'tail -F /opt/logs.txt']
```

StatefulSet

- Used to manage deployment and scaling of stateful apps
- Runs identical pods just like Deployment/ReplicaSets
- Maintains sticky identity for each pod
  - pod identifier(name/network hostname) remains the same between rescheduling
- used for **stateful** applications
- Ensures stable persistent storage for each pod in the set
- selector is used to identify pods to run replicas
- Currently requires a **Headless Service**
- Ideal for running databases, caching systems, messaging queues
- Update Strategies
  - RollingUpdate (default)
  - OnDelete

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: my-statefulset
spec:
  replicas: 3
  serviceName: my-statefulset
  selector:
    matchLabels:
      app: my-statefulset
  template:
    metadata:
      labels:
        app: my-statefulset
    spec:
      containers:
        - name: my-container
          image: my-image
          ports:
            - containerPort: 80
```

DaemonSet

- ensures that a specified pod runs on every node in a cluster
  - or on some nodes (using filters)
- Ideal for
  - system daemons
  - log collectors
  - monitoring agents
  - other tasks that should run on all nodes
- selector is used to identify pods to run daemons
- Update Strategies
  - RollingUpdate (default)
  - OnDelete

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: logshipper
  labels:
    app: logshipper
spec:
  selector:
    matchLabels:
      app: logshipper
  template:
    metadata:
      labels:
        app: logshipper
    spec:
      containers:
        - name: logshipper
          image: busybox
          command: ["sh", "-c", "tail -F /opt/logs.txt"]
```

Job

- Used to execute one-time tasks that run to completion
- Creates one or more pods to execute tasks
- Retries execution of pods until a specified # of them successfully terminate
- for one-time tasks like batch processing, data processing, etc.
- Pod restartPolicy can only be Never or OnFailure
- Deleting a Job deletes the pods created

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: my-job
spec:
  template:
    spec:
      containers:
        - name: my-container
          image: my-image
          command: ["echo", "Hello from my job!"]
      restartPolicy: Never
  backoffLimit: 4
```

CronJob

- Creates Jobs on a predefined schedule
- Similar to Unix Cron jobs
- Schedule is defined using a crontab expression
- For automation of repetitive tasks, or scheduled tasks

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: my-cronjob
spec:
  schedule: "0 22 * * 1-5"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: my-container
              image: my-image
              imagePullPolicy: IfNotPresent
              command: ["echo", "Hello from my cronjob!"]
          restartPolicy: OnFailure
```

ConfigMaps and Secrets

- Store data as key-value pairs
- ConfigMaps store non-sensitive config data
- Secrets store sensitive data
  - not encrypted, only base64 encoded
  - can be encrypted with additional config
- Consumed by pods
- Can be mounted as data volumes or exposed as env vars
- Decouples config data from application code
- Can be immutable after creation

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-configmap
data:
  key1: value1
  key2: value2
immutable: true
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  username: <base64-encoded-value>
  password: <base64-encoded-value>
immutable: true
```

Hands-on Demo

RIZMAXed

Kubernetes Deep Dive

Container Orchestration

Scheduling

RIZMAXed

- Scheduling
  - process to determine the best node to run a pod
- Kube Scheduler assigns pods to nodes
  - ensuring optimal resource utilization and
  - meeting application requirements
- 3-step operation
  - filtering – finds suitable nodes as per resource requirements
  - scoring – ranks found nodes
  - assignment – assigns pod to the highest-ranking node
- Pod placement can be controlled using
  - node name, node selectors, affinities, and taints/tolerations

first-pod.yaml

first-pod.yaml > {} spec > [ ] containers > {} 0 > [ ] ports > {} 0

io.k8s.api.core.v1.Pod (v1@pod.json)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
  labels:
    name: mypod
spec:
  containers:
    - name: nginx
      image: nginx:stable
      resources:
        limits:
          memory: "128Mi"
          cpu: "500m"
      ports:
        - containerPort: 80
```

Controlling Pod Placement

RIZMAXed

- Specify **nodeName** in your pod spec
- Specify node labels with **nodeSelector**

```
Rizmax ~ $ kubectl label node minikube-m02 env=dev
node/minikube-m02 labeled
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - name: nginx
      image: nginx
  nodeSelector:
    env: dev
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - name: nginx
      image: nginx
  nodeName: worker-01
```

Controlling Pod Placement

RIZMAXed

- **Node Affinity and Anti-affinity**
  - similar to nodeSelector
  - provides more control node selection
  - use logical operators to select nodes
- Node Affinity
  - ensures that pods are scheduled on nodes with certain labels
  - Use operators like In, Exists, Gt, Lt
- Node Anti-affinity
  - prevents pods from being scheduled on nodes with certain labels
  - use operators like NotIn, DoesNotExist

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
            - key: env
              operator: In
              values:
                - dev
                - test
            - key: env
              operator: NotIn
              values:
                - prod
  containers:
    - name: nginx
      image: nginx
```

requiredDuringSchedulingIgnoredDuringExecution
=> enforce the rule

preferredDuringSchedulingIgnoredDuringExecution
=> suggest using the rule

Similar to nodeAffinity, use **podAffinity** to co-locate pods or **podAntiAffinity** to prevent them from being scheduled together.

Controlling Pod Placement

RIZMAXed

- **Taints and tolerations**
  - ensure that pods are not scheduled onto inappropriate node
- Taints
  - labels applied to nodes that repel certain pods
  - kubectl taint nodes <node-name> <key>=<value>:<effect>
  - effect: NoExecute | NoSchedule | PreferNoSchedule
- Tolerations
  - applied to pods
  - allow pods to tolerate node taints and still be scheduled on tainted nodes

```
Rizmax ~ $ kubectl taint nodes minikube-m02 team=noram:NoSchedule

node/minikube-m02 tainted
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
    - name: nginx
      image: nginx
  tolerations:
    - key: "team"
      operator: "Equal"
      value: "noram"
      effect: "NoSchedule"
```

Static Pods

- = pods managed directly by kubelet on a node
  - not by the Kubernetes API server
  - managed outside the control plane (local management)
- Suitable for lightweight, node-specific tasks
- Simply place the pod manifest YAML or JSON files inside

  `/etc/kubernetes/manifests`
- Kubelet automatically detects and manages static pods
- Lack features like scaling, rolling updates, or advanced scheduling options
- Use judiciously considering trade-offs in manageability and flexibility

Pod Disruption Budgets (PDB)

RIZMAXed

- Ensure pod availability during voluntary disruptions
  - i.e. maintain high availability during updates, maintenance, or failures
- voluntary disruptions by app owners or cluster admins
  - accidental pod deletion
  - updating pod template causing restart
  - draining a node for repair or upgrade
  - cluster scale down operation

can specify only one of **minAvailable** and **maxUnavailable** in a single PodDisruptionBudget.

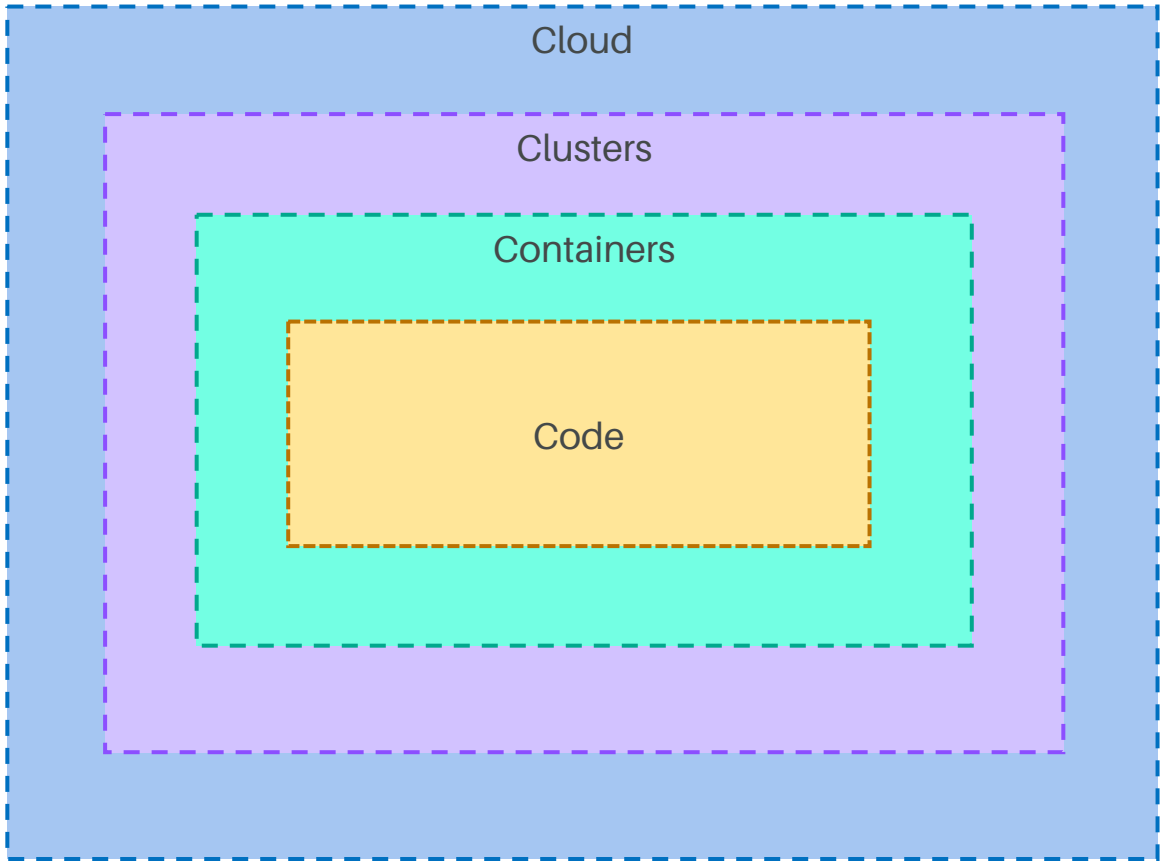
```yaml
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: zk-pdb
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: zookeeper
```

```yaml
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: zk-pdb
spec:
  maxUnavailable: 1
  selector:
    matchLabels:
      app: zookeeper
```

Kubernetes Security

RIZMAXed

- Kubernetes is based on cloud-native architecture
- Adheres to CNCF guidelines for best practices in cloud-native information security
- The 4C's of Cloud Native security
  - Cloud
  - Clusters
  - Containers
  - Code
- Falco
  - Cloud native runtime security tool
  - Real-time detection and alerts on abnormal behavior and potential security threats
  - CNCF graduated project



Authentication

- Checks who has access to the k8s API
  - process of verifying the identity of entities accessing the cluster
  - users, processes, or components
- Kubernetes doesn't handle user or group management
- Authentication methods
  - X.509 Certificates
    - authenticating users and components using TLS client certificates
    - e.g. kubectl use this method
  - Static tokens
    - using bearer tokens for non-interactive authentication
  - Service Account Tokens
    - for pods and services to access the k8s API
    - allows to authenticate with API server from inside of our pod
  - OIDC Authentication
    - to integrate external identity providers using OpenID Connect
    - uses **JWT** (JSON Web Tokens)

Authorization

RIZMAXed

- Determines what actions the authenticated entity can perform
- Controls access to k8s resources based on defined policies
- Authorization Methods
  - Node
    - controlling access to nodes and their resources
  - ABAC (Attribute Based Access Control)
    - defining access policies based on attributes of users, resources, or environments
  - RBAC (Role Based Access Control)
    - assigning roles to users and service accounts
    - defining permissions based on those roles
    - allows for fine-grained access-control
    - recommended
  - Webhook
    - checking for permissions using a webhook
- OPA Gatekeeper (Open Policy Agent Gatekeeper)
  - a policy engine for Kubernetes, an open-source project
  - admission controller that enforces policies on resources during the admission process

RBAC Authorization

- RBAC API declares four kinds of objects
  - Role
    - defines a set of permissions that can be applied to resources within a namespace.
  - ClusterRole
    - defines a set of permissions that apply across the entire cluster.
  - RoleBinding
    - associates a role with a user, group, or service account within a namespace.
  - ClusterRoleBinding
    - associates a cluster-wide role with a user, group, or service account.
- Best practice = rule of **lease privilege**
  - grant only the permissions necessary for each user or component to perform their tasks

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
  - apiGroups: [""] # "" indicates the core API group
    resources: ["pods"]
    verbs: ["get", "watch", "list"]
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
  - kind: User
    name: jane
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

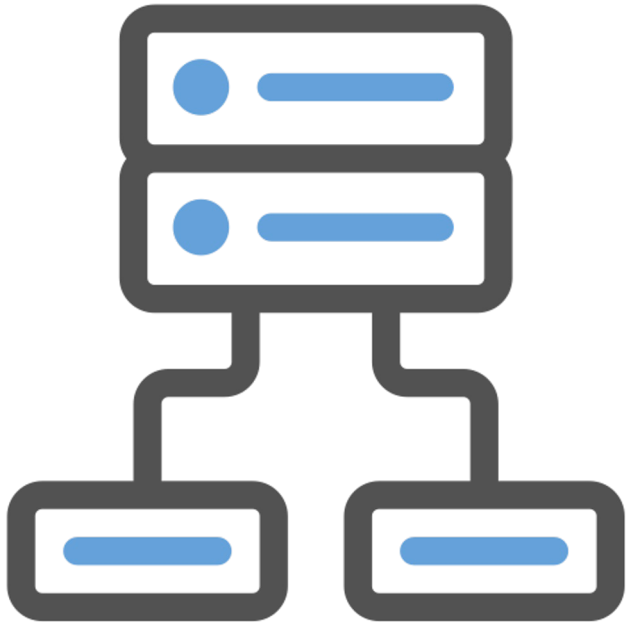
Hands-on Demo

RIZMAXed

Networking in Kubernetes

RIZMAXed

- Kubernetes uses CNI plugins for cluster networking
  - CNI = Container Network Interface
- Cluster DNS
  - built-in DNS service that provides DNS resolution for services and pods within a cluster
- Each pod gets its own IP address within the cluster.
- Pods can communicate with each other without NAT
- Services
  - allow you to expose an app running in pods
- Ingress
  - used to manage external access to services in a cluster
- Network Policies
  - for controlling traffic between pods and external networks



Network Policies

- Rules for controlling traffic flow between pods and external networks
- Specify which pods can communicate with each other and on which ports
- Helps enforce security and segmentation within the cluster.

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - ipBlock:
            cidr: 172.17.0.0/16
            except:
              - 172.17.1.0/24
        - namespaceSelector:
            matchLabels:
              project: myproject
        - podSelector:
            matchLabels:
              role: frontend
      ports:
        - protocol: TCP
          port: 6379
  egress:
    - to:
        - ipBlock:
            cidr: 10.0.0.0/24
      ports:
        - protocol: TCP
          port: 5978
```

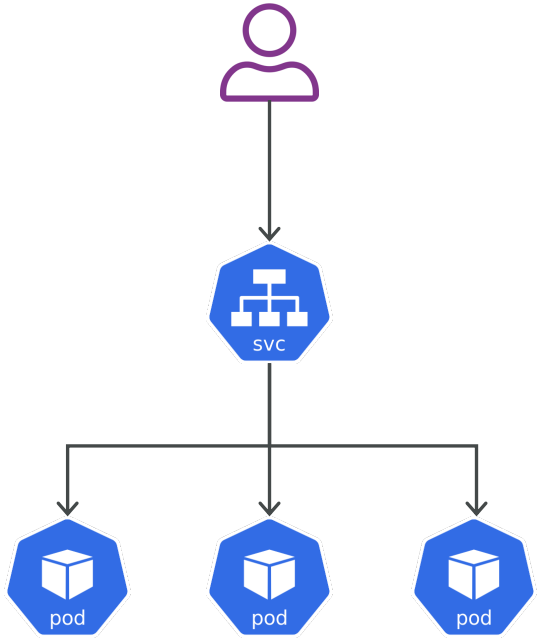
Hands-on Demo

RIZMAXed

Kubernetes Services

RIZMAXed

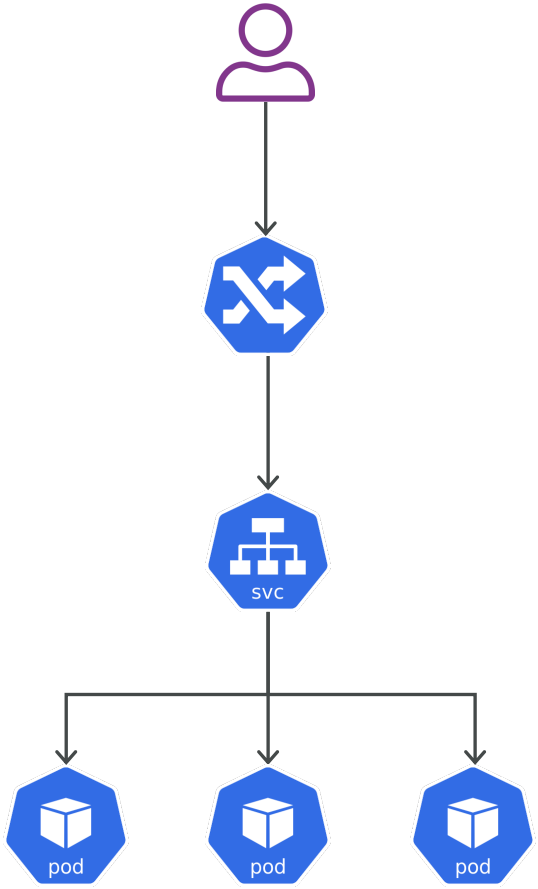
- Service allows you to expose an app running in a set of pods
  - provides a stable endpoint to access apps
  - abstracts away the underlying pod IP addresses
  - uses labels and selectors to route traffic to pods
  - supports load balancing and service discovery
- Service types
  - Cluster IP (default)
    - exposes service internally on an IP within cluster
  - NodePort
    - exposes service externally on a static port on each node
  - LoadBalancer
    - exposes service externally using an external load balancer
    - e.g. AWS ELB
  - ExternalName
    - exposes service externally using provided external name or hostname
    - maps a service to a DNS hostname using a CNAME record
- Headless Service (clusterIP=None)
    - Cluster IP service without an IP address
    - used with StatefulSets



If you create a service without selectors, you must create corresponding endpoint objects manually.

Ingress

- Used to manage external access to services in a cluster
- Specifically, HTTP/S applications, websites and APIs
- Supports
  - TLS termination
  - path-based routing
  - virtual hosting




```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
spec:
  tls:
    - hosts:
        - example.com
      secretName: example-tls
  rules:
    - host: example.com
      http:
        paths:
          - path: /app
            pathType: Prefix
            backend:
              service:
                name: webapp-service
                port:
                  number: 80
          - path: /api
            pathType: Prefix
            backend:
              service:
                name: api-service
                port:
                  number: 80
```

Hands-on Demo

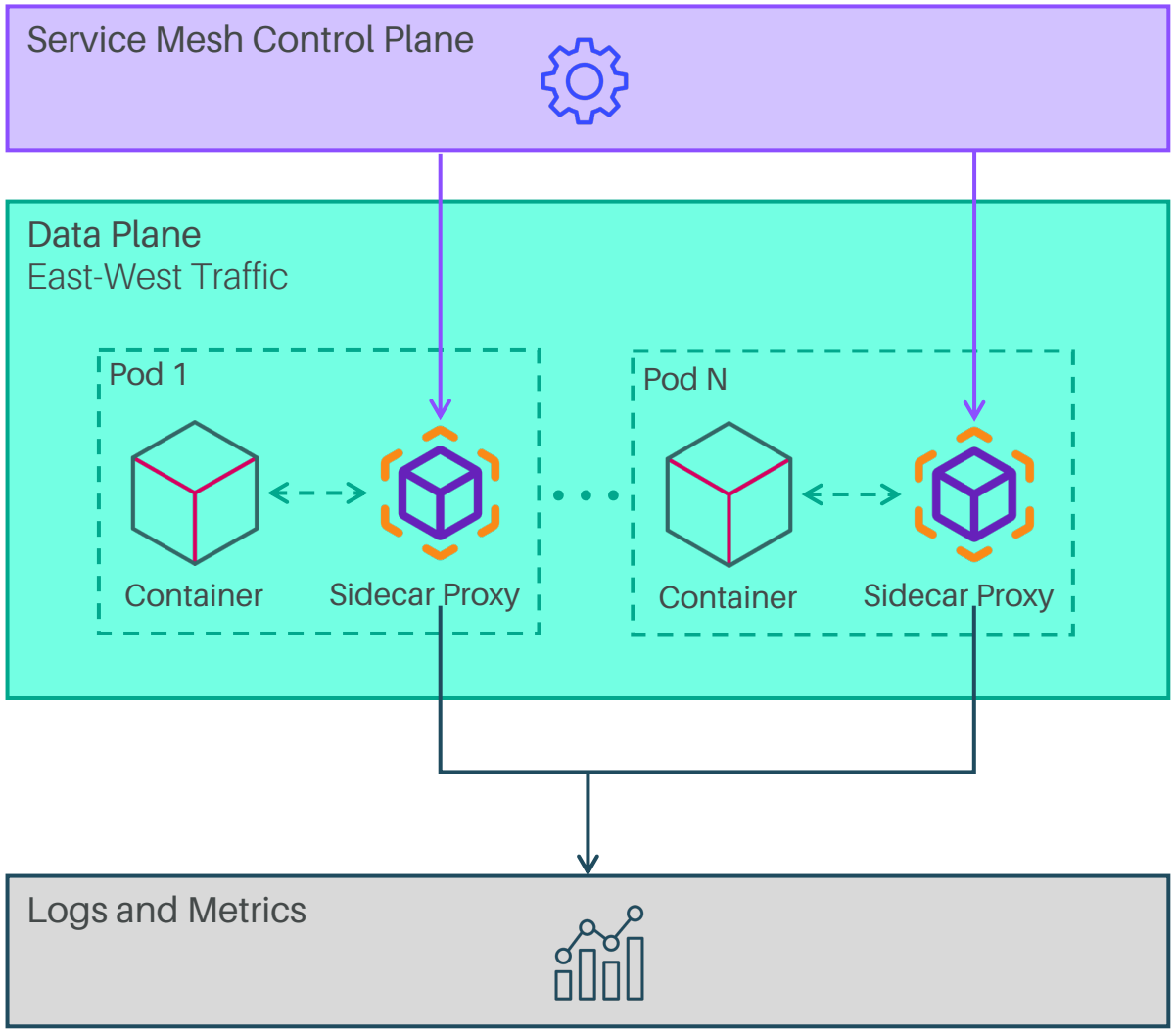
RIZMAXed

Service Mesh

RIZMAXed

- Dedicated infrastructure layer
- Manages communication between microservices within a cluster
- Provides traffic management, observability, and security features
  - routing, load balancing, and retries
  - centralized logging, and metrics
  - implements security policies for inter service communication
- Uses sidecar proxies to intercept and control network traffic
- Kubernetes provides SMI
  - **SMI = Service Mesh Interface**
- Popular solutions that support SMI
  - **Linkerd, Istio, Consul**

Service Mesh Architecture



Kubernetes Storage

- Ephemeral volume types have a lifetime of a pod
- Persistent volumes exist beyond the lifetime of a pod
- emptyDir volume
  - initially empty, used as a scratch disk
  - all containers in the pod can read/write the same files in the volume
  - erased when a Pod is removed
- iscsi volume
  - to mount an existing iscsi volume
  - contents are preserved when pod is removed
- nfs volume
  - to mount an existing NFS share
  - contents are preserved when pod is removed

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
    - image: registry.k8s.io/test-webserver
      name: test-container
      volumeMounts:
        - mountPath: /cache
          name: cache-volume
  volumes:
    - name: cache-volume
      emptyDir:
        sizeLimit: 500Mi
```

Persistent Volumes

- PV = PersistentVolume
  - k8s objects representing storage volumes within the cluster
- PVC = PersistentVolumeClaim
  - k8s object used to request storage from a PV
  - binds a PV and allows you to mount it on a pod
- Reclaim Policy defines action to take on PV when PVC is deleted
  - Retain – manual reclamation
  - Recycle – automatic reclamation using basic scrub (rm -rf /thevolume/*)
  - Delete – delete the volume
- Storage Classes
  - define the type and properties of storage volumes available within the cluster

Persistent Volumes - Example

RIZMAXed

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv
spec:
  capacity:
    storage: 1Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: manual
  hostPath:
    path: /data/example-pv
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: manual
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
    - name: app-container
      image: nginx
      volumeMounts:
        - name: data-volume
          mountPath: /data
  volumes:
    - name: data-volume
      persistentVolumeClaim:
        claimName: example-pvc
```

Container Storage Interface

- Kubernetes provides CSI to interact with storage systems
  - **CSI = Container Storage Interface**
- allows integration between containerized apps and storage platforms
- Popular solutions that support CSI 
  - **Rook**
    - CNCF graduated project
    - Storage Orchestration for Kubernetes with support for **Ceph** storage
    - automates deployment and management of Ceph
    - provide self-managing, self-scaling, and self-healing storage
  - **Longhorn**
    - CNCF graduated project
    - Cloud-native distributed storage for Kubernetes

Autoscaling K8S Workloads

RIZMAXed

- HPA = Horizontal Pod Autoscaler
  - scales pod replicas automatically
  - based on CPU utilization or other metrics
- VPA = Vertical Pod Autoscaler
  - scales CPU and memory reservations of your pods automatically
  - based on resource utilization
- Cluster Autoscaler
  - scales # of nodes in the cluster automatically
  - based on resource demands

```yaml
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: myapp-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: "Deployment"
    name: "myapp-deployment"
  updatePolicy:
    updateMode: "Auto"
```

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
```

Autoscaling K8S Workloads (contd.)

- KEDA
  - Kubernetes Event-driven Autoscaling
  - KEDA is a CNCF graduated project
  - scale automatically based on event sources
  - e.g. scale based on # of messages in a queue
    - Azure Queue, Kafka topics, or AWS SQS
- Karpenter
  - open-source cluster Autoscaler
  - created by AWS
  - automatically provisions new nodes in response to unschedulable pods
- Autoscaling Best Practices
  - Use the Right Metrics
  - Test Your Autoscaling
  - Monitor and Adjust

```yaml
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: sqs-scaledobject
spec:
  scaleTargetRef:
    deploymentName: myapp-deployment
  pollingInterval: 5
  minReplicaCount: 1
  maxReplicaCount: 10
  triggers:
    - type: aws-sqs
      metadata:
        queueName: my-sqs-queue
        awsRegion: us-west-2
        identityResourceID: my-aws-iam-role-arn
        queueLength: "5"
```

Hands-on Demo

RIZMAXed

Helm and Helm Charts

RIZMAXed

- Helm
  - package manager for Kubernetes
  - CNCF graduated project
  - simplifies the process of deploying, managing, and upgrading applications
- Helm Chart
  - packaged Kubernetes resources
  - reusable, versioned bundles containing
    - templates for Kubernetes manifests
    - default config
    - optional dependencies
- How it works
  - Developers create Helm charts to package k8s app
  - Users install Helm chart onto their k8s cluster using helm install
  - User can manage k8s apps using helm upgrade, and helm rollback


```
$ helm install my-web-app ./path/to/chart
```

```
$ helm upgrade my-web-app ./path/to/updated/chart
```

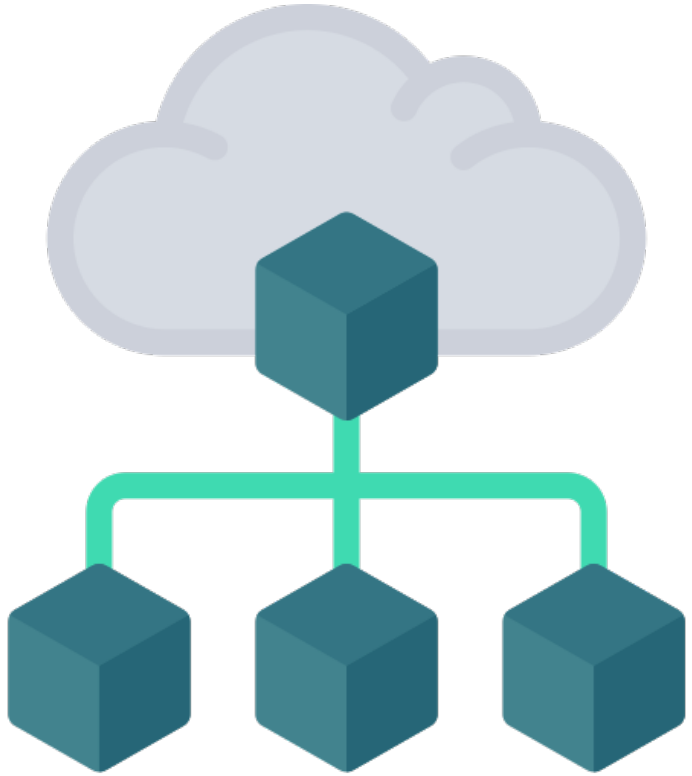
```
$ helm rollback my-web-app
```

Cloud Native Computing

Cloud Native, Serverless, and Microservices

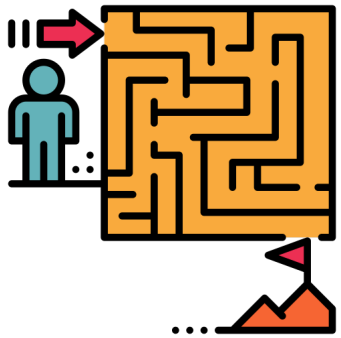
Cloud Native Computing

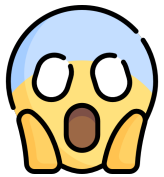
- = design and deployment of applications that fully leverage the benefits of cloud computing models (public, private, or hybrid)
- => scalable, resilient, agile, cost-effective, portable applications
- Key guiding principles
  - containerization and orchestration
  - microservices architecture
  - continuous integration/delivery/deployment
  - DevOps, GitOps, FinOps
- Use cases
  - distributed web apps
  - IoT and edge computing
  - big data and analytics

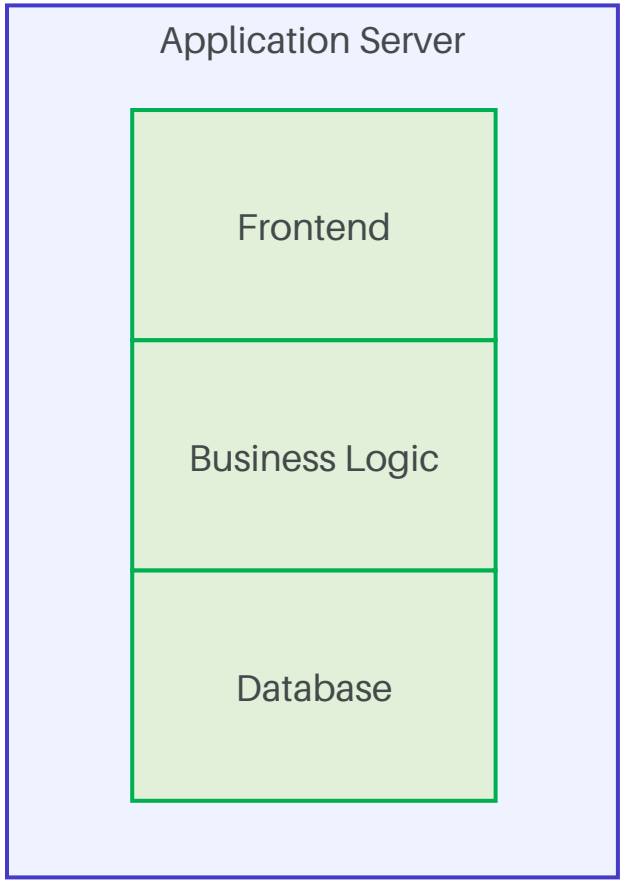


Traditional Approach to Software Development

- Monolithic architecture
  - All components run as a closely-knit unit => tightly-coupled
  - Not highly scalable
  - Not flexible
  - Not agile
  - Complex deployment (not automated)
  - Difficult to maintain



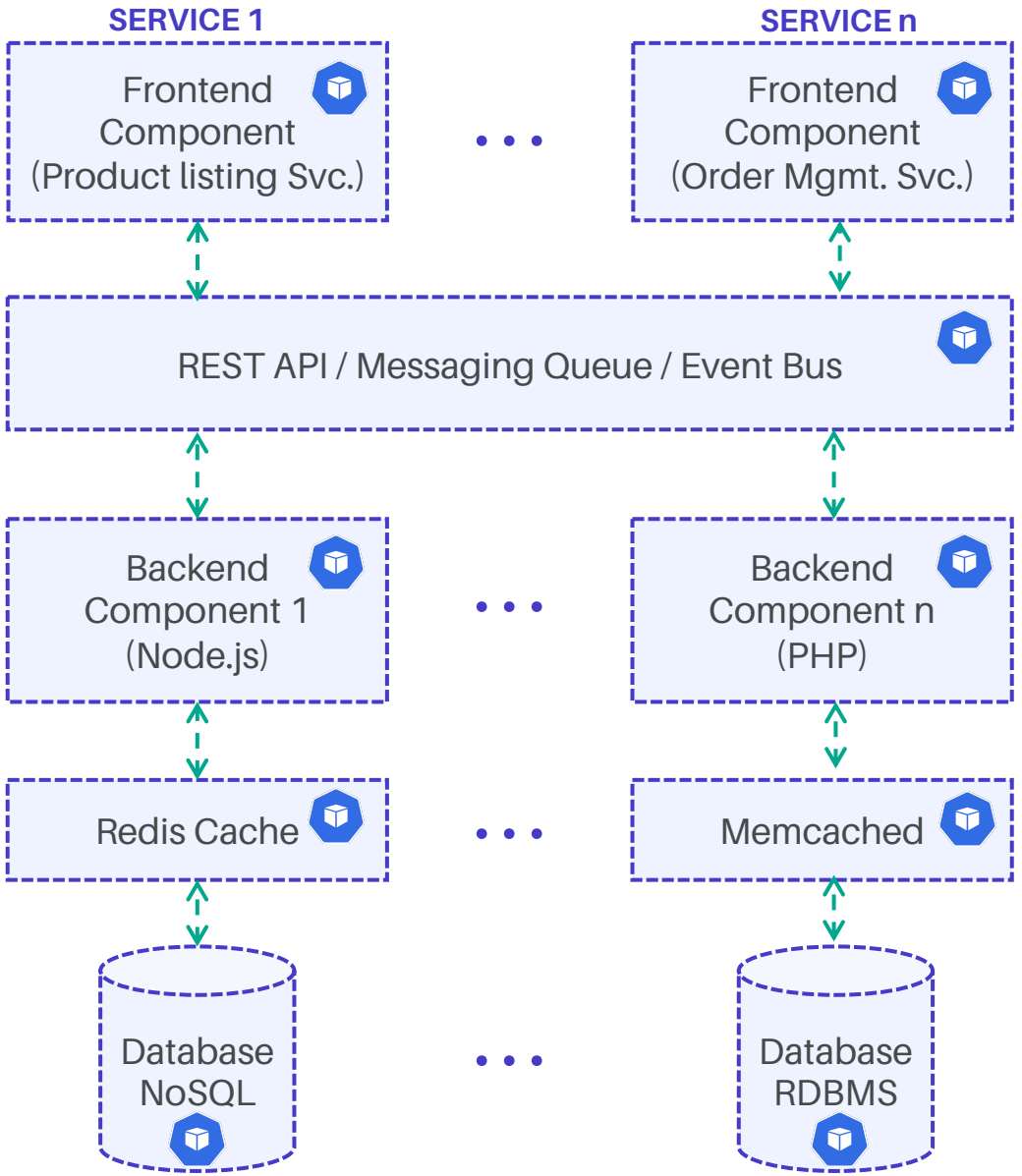




Microservices Architecture

RIZMAXed

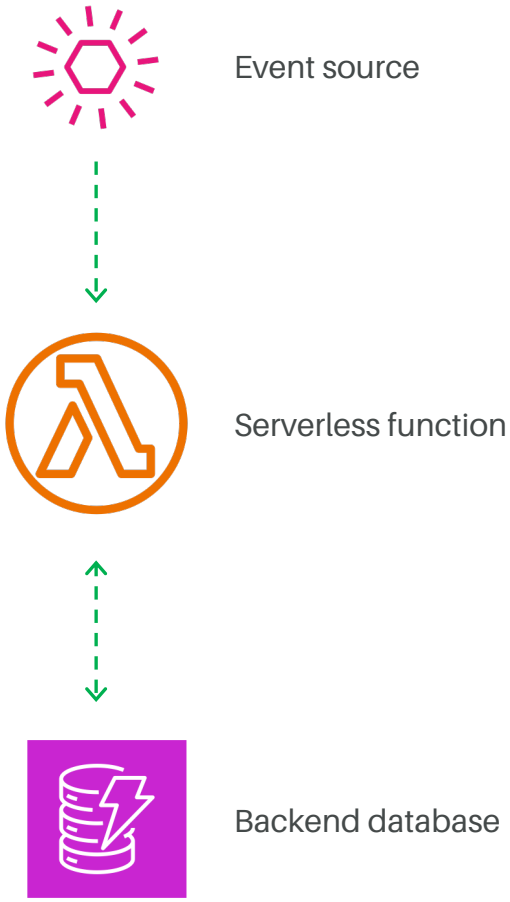
- Structures an application as a collection of services
- Each service is
  - independent unit
  - can be deployed on its own
  - often owned by a single, small team
  - highly and independently scalable
  - supports automated deployments
  - independently maintainable
- Loosely coupled architecture
  - allows for distributed computing
- Flexible, agile (allows for rapid, frequent changes)



Serverless Computing

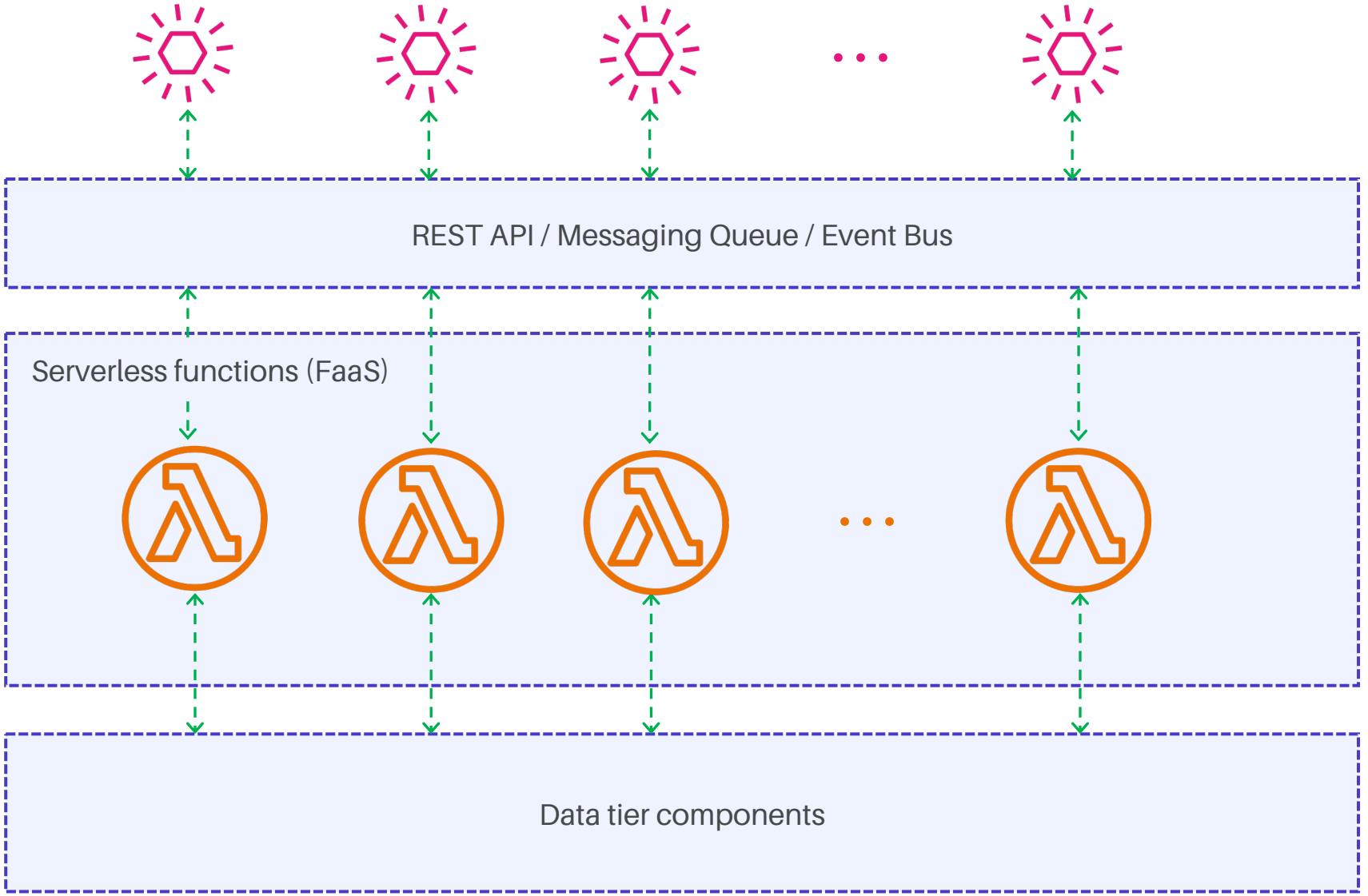
RIZMAXed

- Cloud provider manages servers
- Developers focus on writing code (functions)
  - without worrying about underlying infrastructure
  - => FaaS (Function as a Service)
- Serverless functions
  - event-driven
  - auto-scaling
  - pay-per-use
- Benefits
  - No servers to manage
  - Cost-effective
  - High scalability
  - Fast time-to-market
- FaaS Solutions on public cloud
  - AWS Lambda
  - Azure Functions
  - Google Cloud Functions



Serverless Microservices Architecture

RIZMAXed



Kubernetes Serverless Solutions

RIZMAXed

- Knative
  - serverless application layer built on top of Kubernetes
  - three components
    - Knative Serving – HTTP-triggered autoscaling container runtime
    - Knative Eventing – CloudEvents-over-HTTP asynchronous routing layer
    - Knative Functions – Serverless function framework

- OpenFaaS
  - serverless framework for Kubernetes and Docker Swarm
  - platform for building serverless functions with containers
  - write services and functions in multiple languages
  - build and ship your code in a container image

Community and Governance

RIZMAXed

- CNCF = Cloud Native Computing Foundation
  - is a vendor-neutral organization
  - that focuses on growth and adoption of cloud-native technologies
  - through open-source collaboration and community engagement
- Community
  - diverse ecosystem of contributors, users, and enthusiasts worldwide
  - collaborative development, knowledge sharing, and innovation
  - regular events, meetups, and conferences for networking, learning and engagement
    - e.g. KubeCon + CloudNativeCon, Contributor Summits
- Governance
  - open feedback channels enable continuous improvement
  - transparent and inclusive governance models
  - decisions are made openly and democratically
  - enforced code of conduct

CNCF Committees and Groups

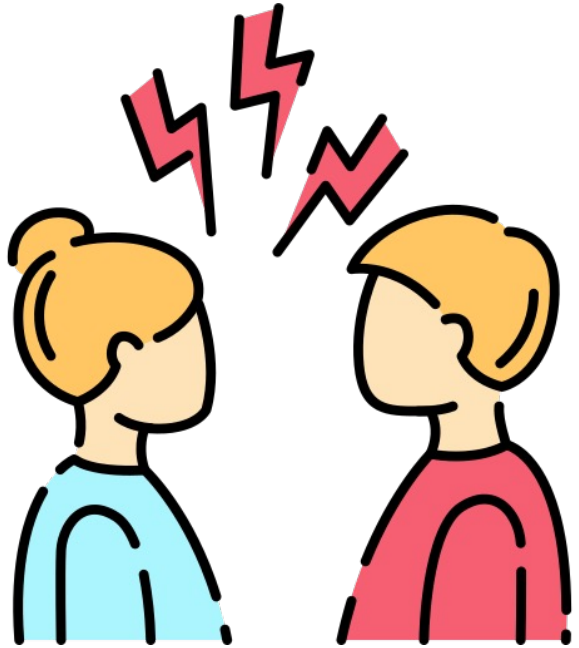
RIZMAXed

- GB = Governing Board
  - responsible for marketing, budget and other business oversight decisions
- TOC = Technical Oversight Committee
  - defines and maintains the technical vision
  - provides technical leadership and oversight for CNCF and its projects
  - reviews/approves new projects
  - governance discussions and decision-making
- SIG = Special Interest Group
  - focus on specific domains, technologies, or use cases within cloud native ecosystem
- WG = Working Group
  - SIGs may organize working groups to address specific challenges/initiatives
- EUC = End User Community
  - End User SIGs and User Groups

Conflict resolution within CNCF

RIZMAXed

- Open **Discussion** 💡
  - open and transparent communication within its groups
- Mediation
  - group facilitators/leaders may act as mediators to discuss and reach a resolution
- Escalation
  - unresolved conflicts may be escalated to TOC, GB for review and resolution
- Consensus building
  - among group members and stakeholders
  - though **elections and voting** 💡
- Code of Conduct enforcement
  - to ensure a respectful and inclusive environment

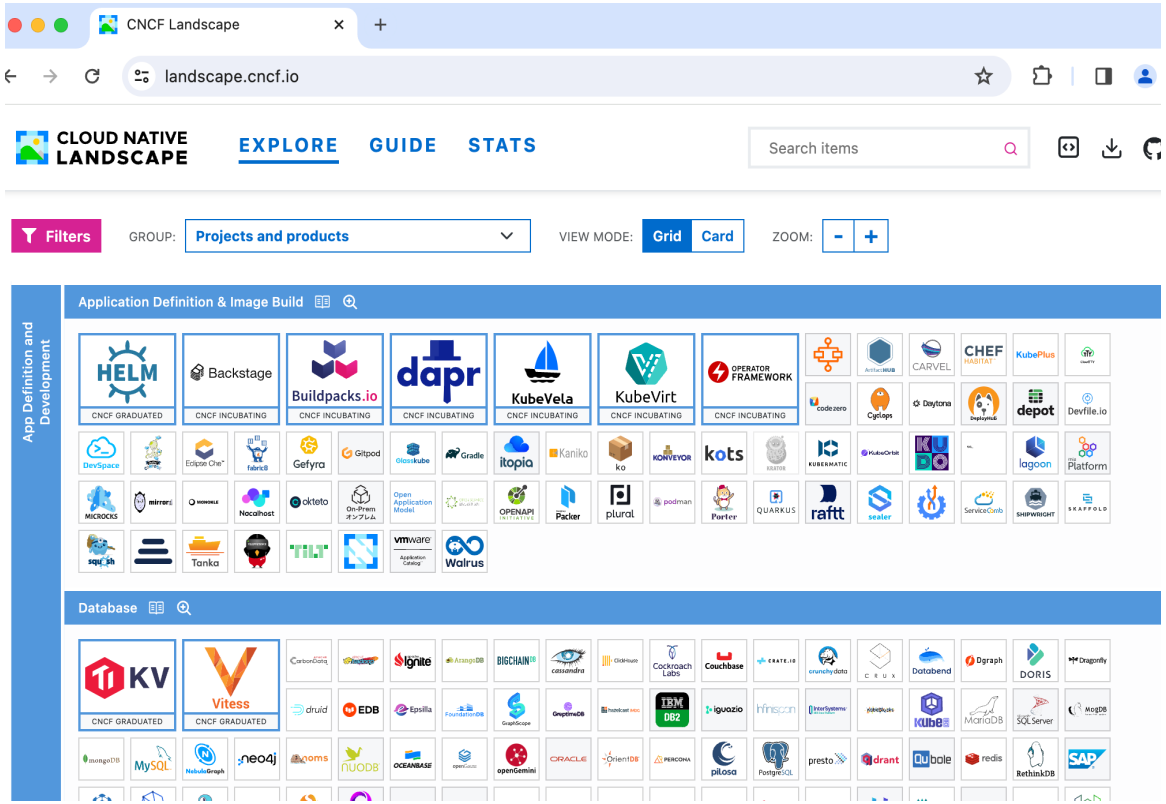


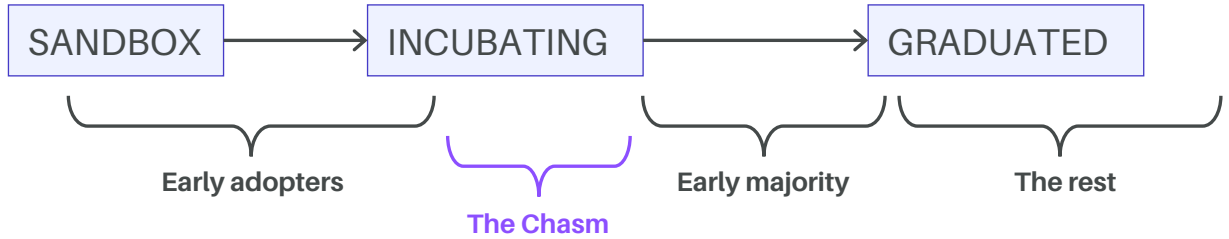
CNCF Landscape

RIZMAXed

- Comprehensive visualization of the cloud-native ecosystem
- Showcases projects and tools
  - exploration and discovery
  - reference
  - community engagement
- Well-defined categories and filtering options
- CNCF projects are tagged as per their maturity levels
  - Sandbox
  - Incubating
  - Graduated
- Crossing the Chasm
  - => moving from early adopters to mainstream adoption
  - i.e. from Incubating to Graduated

CNCF Landscape - https://landscape.cncf.io/





Open Standards

- OCI = Open Container Initiative
  - Defines open standards around container formats and runtimes
    - Runtime Spec (based on runc)
    - Image Spec
    - Distribution Spec
- CRI = Container Runtime Interface
- CNI = Container Network Interface
- CSI = Container Storage Interface
- SMI = Service Mesh Interface
- Enable diverse stakeholders to collaborate and innovate
- Interoperability
  - between different systems and components within the cloud-native ecosystem
- Portability
  - across different environments and platforms
  - reduces vendor lock-in
  - increases flexibility

Cloud Native Roles and Personas

RIZMAXed

- Application Developer
- DevOps Engineer
- Site Reliability Engineer (SRE)
- Security Engineer
- DevSecOps Engineer
- Cloud Architect
- FinOps Consultant

Application Developer

- Responsibilities
  - Design, develop, deploy cloud-native apps
    - using modern technologies like microservices, containers, and serverless
    - using agile practices
- Skills
  - Programming languages
  - Containerization
  - Git and version-control systems
  - CI/CD pipelines

DevOps Engineer

- Responsibilities
  - Automate the deployment, scaling, and management
    - of cloud-native apps and infrastructure
    - using infrastructure-as-code (IaC)
    - using CI/CD pipelines
    - using GitOps tools
- Skills
  - Cloud platforms
  - Container orchestration
  - Configuration management
  - Scripting languages

Site Reliability Engineer (SRE)

RIZMAXed

- Crossover role with DevOps
  - with focus on reliability and operational excellence
- Responsibilities
  - Ensure reliability, availability, and performance
    - of cloud-native applications and infrastructure
    - through monitoring, alerting, incident response, and capacity planning
  - Meet SLAs, with SLOs, SLIs
    - SLA = Service Level Agreement
    - LO = Service Level Objective
    - SLI = Service Level Indicator
- Skills
  - Observability and Telemetry tools
  - Incident management
  - Performance optimization
  - Distributed systems architecture

SLA, SLO, and SLI

RIZMAXed

- SLA = Service Level Agreement
  - contract with the client defining the level of service
  - based on measurable metrics
    - uptime, availability, performance, and response times
  - e.g. service uptime SLA = 99.95%
- SLO = Service Level Objective
  - defines the acceptable level of service
  - goals defined to meet each measurable metric in the SL
  - e.g. SLO = 99.97% uptime
- SLI = Service Level Indicator
  - Measure of actual performance against the SLOs
  - Used to monitor and track SLA compliance
  - e.g. SLI = 99.99% uptime

SLA compliance = SLI ≥ SLO

Security Engineer

RIZMAXed

- Responsibilities
  - Secure cloud-native applications and infrastructure
  - Implement security best practices
  - Implement access control mechanisms
  - Threat detection and vulnerability management
- Skills
  - Cloud native security principles
  - Encryption
  - Identity and access management (IAM)
  - Security compliance frameworks
  - Penetration testing

DevSecOps Engineer

- Responsibilities
  - Integrate security practices into the DevOps workflow
  - Implement security controls, tools, and processes
  - Collaborate with dev, ops, and security teams
  - Automate security testing, compliance checks, and remediation tasks
  - Monitor and analyze security metrics, logs, and alerts
- Skills
  - Security principles, standards, and best practices
  - Experience with DevOps practices
  - Scripting / programming skills
  - Cloud platforms

Cloud Architect

RIZMAXed

- Responsibilities
  - Design and implement cloud-native architectures
    - to meet scalability, reliability, and performance requirements
    - using cloud services, microservices, hybrid cloud solutions
- Skills
  - Cloud platforms
  - Networking
  - Containerization
  - Distributed systems
  - Architecture patterns
  - System Design skills
  - Communication skills

FinOps Consultant

RIZMAXed

- Responsibilities
  - Cloud cost optimization
  - Analyze cloud spending trends, identify cost-saving opportunities
  - Recommend strategies to improve cost-efficiency
  - Diagnose cost inefficiencies, identify root causes
  - Develop practical solutions to optimize cloud spending
- Skills
  - Financial management skills
    - budgeting, forecasting, and cost analysis
  - In-depth knowledge of cloud platforms
  - Familiarity with cloud billing and pricing mechanisms
  - Cloud cost management tools

Telemetry & Observability

Prometheus, Grafana, and FinOps

Telemetry and Observability

RIZMAXed

- Telemetry
  - collect, aggregate, and transmit data outputs from a system
  - for monitoring and performance analysis
- Observability
  - infer the internal state and behavior of a system
  - based on its external outputs i.e. using telemetry data
- Key components
  - **Metrics**
    - measurements over a time period e.g. CPU, memory utilization etc.
    - Categories – **Meters, Counters, Gauges, Histograms**
  - **Logs**
    - recorded messages generated by the systems and components
    - based on verbosity like error, warning, info, debug etc
  - **Trace**
    - represents journey of a request as it moves through a distributed system
    - made up of a series of tagged time intervals called **spans**
  - **Events**
    - Notifications and alerts generated by a system

OBSERVABILITY TOOLS

Prometheus
Monitoring & alerting toolkit

Grafana
Visualization and dashboarding platform

ELK stack
Elasticsearch, Logstash, Kibana Stack for collecting, indexing, and visualizing logs

Jaeger
Distributed tracing solution

Zipkin
Distributed tracing solution

Prometheus

- Open-source monitoring and alerting toolkit, CNCF graduated
- Widely used for monitoring Kubernetes
- Originally built at SoundCloud
- Collects and stores **time-series data** in a database
  - Can be queried using **PromQL** (Prometheus Query Language)
- **Prometheus Server**
  - Discovers targets (pods, services) using Kubernetes service discovery
  - Collects data from targets using **HTTP scraping** (**pull-based**)
- **Metrics Exporter**
  - gathers metrics from systems that do not natively expose Prometheus-compatible endpoints
  - agents deployed alongside Kubernetes applications
  - to expose metrics in Prometheus-compatible format (**push based**)
- **Alertmanager**
  - Handles alerts generated by Prometheus
  - Sends notifications via various channels (email, Slack, PagerDuty)

Grafana

- Open-source analytics and visualization platform
- Widely used for visualizing metrics and logs in real-time
  - System performance and health
- Flexible, interactive, customizable dashboards and panels
- Supports a wide range of data sources
  - Prometheus, Elasticsearch, InfluxDB, etc.
- Can set up alerts based on predefined thresholds and conditions
- Supports templating for dynamic filtering and grouping of data

Cost Management in Kubernetes

RIZMAXed

- Refers to optimizing costs on cloud resources
- Right sizing – use right sized resources e.g. CPU, memory
- Auto scaling – dynamically adjust resource capacity
- Resource tagging – to track and allocate costs accurately
- Cost allocation
  - allocating costs to teams, projects, or departments
  - for accountability and budgeting purposes
- Reserved instances
  - securing discounted pricing using long-term reservations
- Spot instances or preemptible VMs
  - to take advantage of discounted pricing
  - for non-critical workloads or batch processing tasks
- Cloud Anomaly Detection
  - to identify abnormal patterns or deviations in cloud infrastructure, apps and services
  - anomalies in resource utilization, or billing discrepancies

COST MANAGEMENT TOOLS

AWS Cost Explorer
AWS tool for visualizing and analyzing AWS cost and usage

Azure Cost Management
Azure's cost monitoring and optimization service

Google Cloud Billing
collection of tools to track and your Google Cloud spending

Kubecost
Cost monitoring for Kubernetes workloads and cloud costs

CloudHealth
Multi-cloud FinOps platform

Cloud Native Application Delivery

CICD, GitOps, and more...

Cloud Native Application Delivery

- Modern approach to building, deploying, and managing applications
- What?
  - Containerization and orchestration
  - Microservices architecture
  - Infrastructure as Code (IaC)
  - CI/CD automation and GitOps

Cloud Native Application Delivery

- Why?
  - Agility
    - rapid iteration, experimentation, and innovation
  - Scalability
    - apps can handle varying workloads and demands
  - Resilience
    - reliability with fault tolerance, and minimal disruptions
  - Cost-efficiency
    - pay-as-you-go, pay for what you use

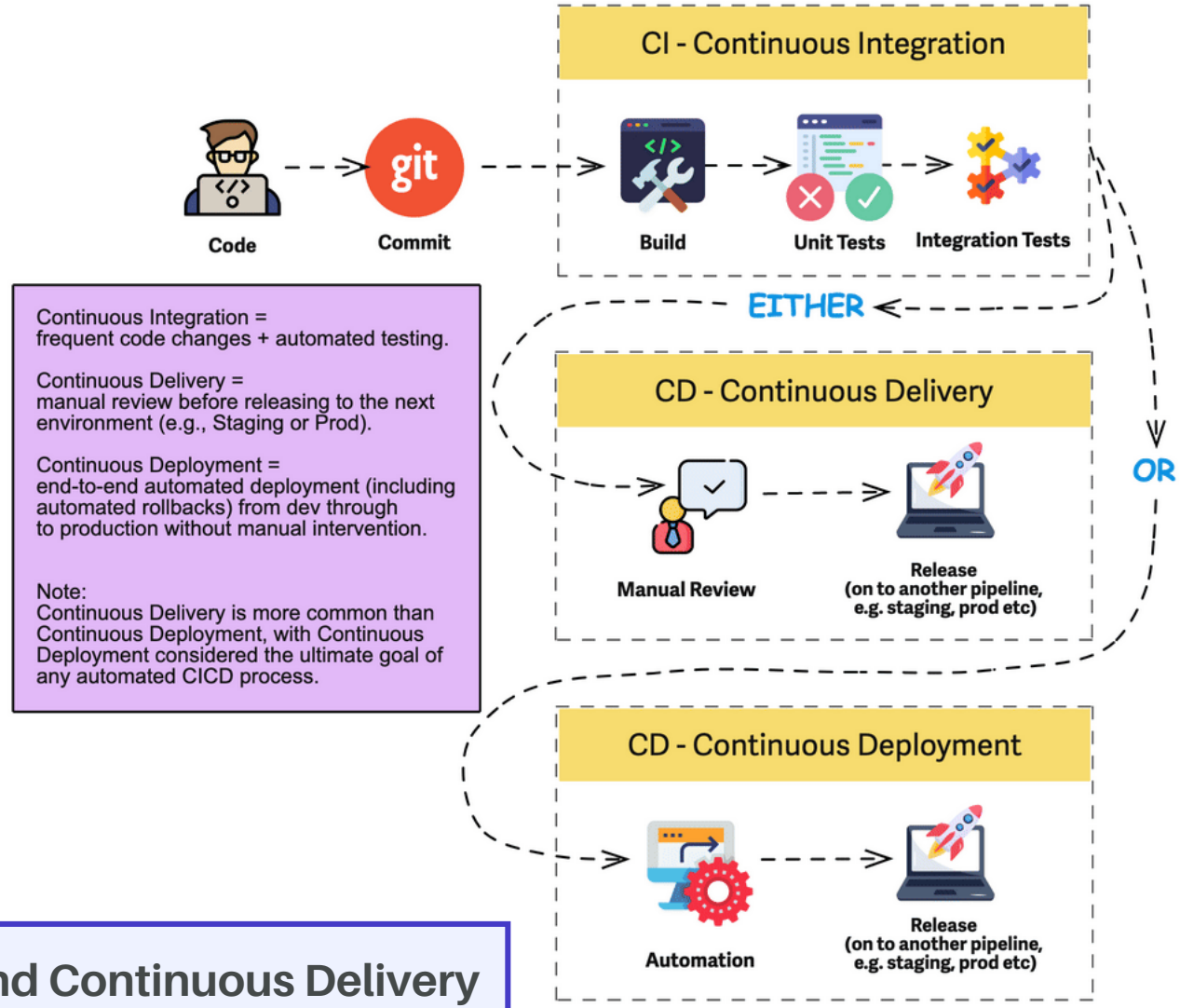
Cloud Native Application Delivery

- How?
  - Embrace agile methodologies
  - Design for resilience and scale
  - Embrace DevOps culture
  - Automate everything
  - Monitor and iterate
    - Telemetry and Observability

Continuous Integration -> Continuous Delivery -> Continuous Deployment

CI/CD

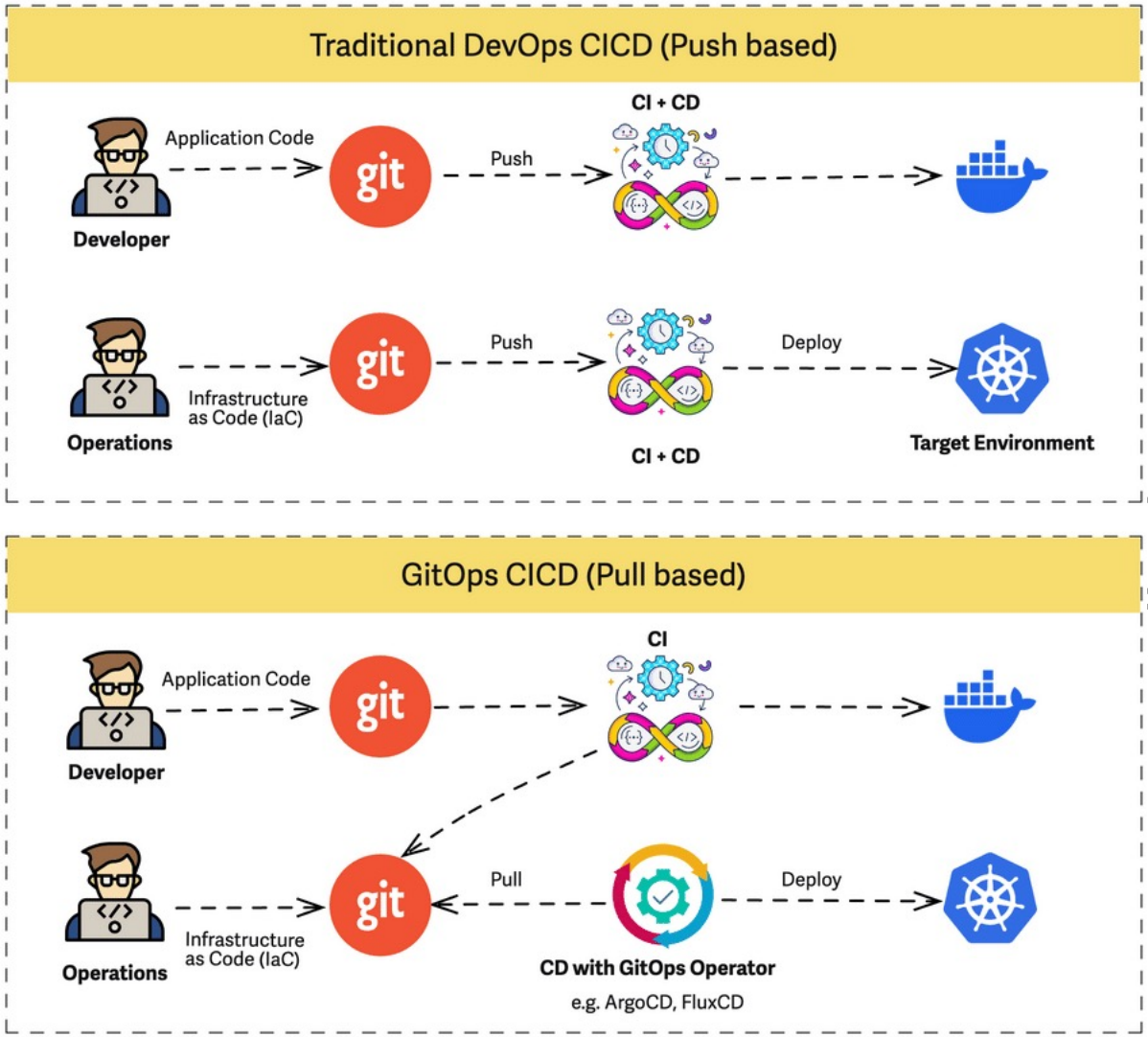
- **Automate build, test, and deploy** process
- Continuous Integration (CI)
  - automatically **build and test** code changes on commit to Git repo
- Continuous Delivery (CD)
  - automatically **deploy to staging** or pre-production
- Continuous Deployment (CD)
  - automatically **deploy to production** (without manual intervention)



CI/CD typically refers to: **Continuous Integration and Continuous Delivery**

GitOps

- DevOps
  - development (Dev) and operations (Ops) teams work together to deliver apps using deployment automation
  - CI/CD automation, IaC, Observability
  - Traditionally push-based deployment
- GitOps
  - extends DevOps principles
  - focus on automated continuous sync
  - uses a Git operator to reconcile
    - desired state in Git
    - with actual state of the cluster
  - GitOps operators – ArgoCD, FluxCD



ArgoCD and FluxCD

- ArgoCD and FluxCD are two popular GitOps Operators
- ArgoCD enables GitOps workflows
  - continuously monitors Git repositories for changes
  - automatically applies them to Kubernetes clusters
  - => ensures desired state convergence
  - provides Image Updater
    - to automatically update the container images used in k8s manifests
- FluxCD is built with the GitOps Toolkit
  - GitOps Toolkit is collection of open-source tools and best practices to implement GitOps workflows
  - continuously monitors git and applies changes to k8s
  - Image reflector and automation controllers
    - work together to update Git when container images are updated

Tackling the exam questions

"Perfect" practice makes perfect!

Exam Tips

RIZMAXed

- Focus on the question first, before reading the responses
  - try to answer the question if possible
  - then see if a there is a matching response
- Eliminate choices containing obviously incorrect statements, and using your knowledge
- Identify key phrases + look for qualifiers in the questions
  - the most operationally efficient
  - the most cost-effective option
  - the fastest
  - the least administrative overhead etc.

Exam Tips

- If you can’t figure out the correct answer
  - make a best guess (there is no penalty for incorrect answers, answer all the questions)
  - flag the question for later review
  - move on to the next question

Course Roundup

Woohoo! You did it!

Congratulations

- You’re all set!
- Revisit the lectures, review the course slides
- practice hands-on demos if you like
- Go ahead and take the final exam!

Some more resources

- Review Kubernetes documentation wherever possible
  - https://kubernetes.io/docs/home/
- Review the kubectl Cheat Sheet
  - https://kubernetes.io/docs/reference/kubectl/quick-reference/
- Go over the CNCF Landscape
  - https://landscape.cncf.io/
- Practice Kubernetes Tutorials if you like
  - https://kubernetes.io/docs/tutorials/
- Review the KCNA Exam Resources
  - https://training.linuxfoundation.org/certification/kubernetes-cloud-native-associate/