WHAT IS TROUBLESHOOTING:

- Kubernetes troubleshooting is the process of identifying, diagnosing, and resolving issues in Kubernetes clusters, nodes, pods, or containers.
- Kubernetes troubleshooting also includes effective ongoing management of faults and taking measures to prevent issues in Kubernetes components.
- when i give kubectl run pod it will go to
  api scheduler -- > etcd -- > scheduler -- > kubelet -- > container
- There are three aspects to effective troubleshooting in a Kubernetes cluster

1. understanding the problem,
2. managing and remediating the problem.
3. preventing the problem from recurring.

EXIT CODE:

0 = then app is running fine

1 = something issues will be there

UNDERSTANDING:

- Reviewing recent changes of cluster, pod, or node, to see what caused the failure.
- Analyzing YAML configurations, Github repositories, and logs for VMs or bare metal machines running the malfunctioning components.
- Looking at Kubernetes events and metrics such as disk pressure, memory pressure, and utilization.
- In a mature environment, you should have access to dashboards that show important metrics for clusters, nodes, pods, and containers over time.
- Comparing similar components behaving the same way, and analyzing dependencies between components, to see if they are related to the failure.

MANAGEMENT:

- In a microservices architecture, each component to be developed and managed by a separate team. Because production incidents often involve multiple components, collaboration is essential to remediate problems fast.
- Once the issue is understood, there are three approaches to remediating it:
- **Ad hoc solutions**—based on tribal knowledge by the teams working on the affected components.
- **Manual runbooks**—a clear, documented procedure showing how to resolve each type of incident.
- **Automated runbooks**—an automated process, which could be implemented as a script, infrastructure as code (IaC) template, or Kubernetes operator, and is triggered automatically when the issue is detected. It can be challenging to automate responses to all common incidents, but it can be highly beneficial, reducing downtime and eliminating human error.

POD LEVEL ISSUES:

- There are several pod-level issues that can occur in Kubernetes (K8s). Here are some common pod-level issues and their possible causes:

1. CreateContainerConfigError
2. ImagePullBackOff or ErrorImagePull
3. CrashLoopBackOff

CONTAINERCONFIGERROR:

- This error is usually the result of a missing Secret or ConfigMap. Secrets are Kubernetes objects used to store sensitive information like database credentials.
- ConfigMaps store data as key-value pairs, and are typically used to hold configuration information used by multiple pods.

- kubectl create deploy raham2 --image=mariadb
- kubectl get po - o wide
- raham2 0/1 Error 3 (37s ago) 73s 10.244.0.9 minikube <none> <none>
- you will get error after some time
- kubectl describe pod raham2-66c4648c74-hn45t
- now you will gte exit code as one : that means some issue will be there
- kubectl logs raham2-66c4648c74-hn45t
- kubectl set env deploy raham2 MARIADB_ROOT_PASSWORD=raham123
- kubectl get po - o wide
- Now it will run

IMAGEPULLBACKOFF OR ERRORIMAGEPULL:

- This status means that a pod could not run because it attempted to pull a container image from a registry, and failed.
- This error typically indicates a problem with accessing the container registry or retrieving the image.
- The pod refuses to start because it cannot create one or more containers defined in its manifest.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
  - name: my-app-container
    image: nginxx
```

```
kubectl apply -f abc.yml
kubectl get po
kubectl describe pod my-app
```

CRASHLOOPBACKOFF:

- This issue indicates a pod cannot be scheduled on a node.
- This could happen because the node does not have sufficient resources to run the pod, or because the pod did not succeed in mounting the requested volumes.
- if cluster is resarted or if password is changed and if i dont have req resources

CAUSES:

- Image availability
- Image registry authentication
- Image registry connectivity (Firewall rules or network restrictions of Cluster)
- Image registry permissions
- Image pull rate limits
- Network connectivity and DNS resolution
- Proxy configurations
- Resource constraints

SERVICE LEVEL ISSUES:

- If your service is not running properly
- first, we need to check labels & selectors
- next check the ports open or not
- next check the ingress and Network policy enbale
- kubectl get networkpolicy -A

- kubectl create deploy raham3 --image=httpd
- kubectl expose deploy raham3 --port=80 --type=NodePort
- kubectl get pods,svc
- kubectl get ep
- kubectl get pods -o wide
- kubectl get svc -o wide
- minikube ip

- curl 192.168.49.2:30933
- curl (minikubeip):(ports) : ports will be on svc
- kubectl get svc
- kubectl edit svc raham3
- app: raham3 -- > raham4 -- > selector -- > app: raham3 -- > raham4 -- > save
- kubectl get ep
- curl 192.168.49.2:30933
- curl: (7) Failed to connect to 192.168.49.2 port 30933: Connection refused
- now change both of them it will work, use sed command for better purpose
- kubectl get ep
- curl 192.168.49.2:30933
- kubectl get events : describe all events
- kubectl get events -o wide : to see more info
- kubectl describe deploy : object level events
- kubectl get pod: app level events

CLUSTER LEVEL ISSUES:

- When a worker node shuts down or crashes, all stateful pods that reside on it become unavailable, and the node status appears as NotReady.
- If a node has a NotReady status for over five minutes (by default), Kubernetes changes the status of pods scheduled on it to Unknown, and attempts to schedule it on another node, with status ContainerCreating.
- Failed node is able to recover or is rebooted by the user, the issue will resolve itself. Once the failed node recovers and joins the cluster, it will follow below:
- The pod with Unknown status is deleted, and volumes are detached from the failed node.
- The pod is rescheduled on the new node, its status changes from Unknown to ContainerCreating and required volumes are attached.
- Kubernetes uses a five-minute timeout (by default), after which the pod will run on the node, and its status changes from ContainerCreating to Running.

- If you have no time to wait, or the node does not recover, you'll need to help Kubernetes reschedule the stateful pods on another, working node. There are two ways to achieve this:
- **Remove failed node from the cluster**—using the command kubectl delete node [name]
- **Delete stateful pods with status unknown**—using the command kubectl delete pods [pod_name] --grace-period=0 --force -n [namespace]

CAUSES:

CONTROL PLANE COMPONENT FAILURES:

- Network connectivity
- Resource constraints:
- Misconfiguration or corrupted data.

NODE-RELATED ISSUES:

- Node failures:
- Resource exhaustion
- Node network connectivity
- Node eviction

POD LEVEL ISSUES:

- You might notice some similarities between CRDs and ConfigMap, a Kubernetes built-in object. Additionally, CRDs might actually accomplish the same thing if you use them in place of a custom controller. Both of them can be used to keep individual configurations. But there are observable distinctions between them.
- First of all, ConfigMaps are created with the intention of configuring your pods. They can be mounted into the pod as files or environment variables. If you have well-defined configuration files, such as the Apache or MySQL config, they function well.
- Pods can consume CRDs as well, but only after making a call to the Kubernetes API. Simply put, they serve a different function than ConfigMaps. They are not intended to be used to configure your pods, but rather to expand the Kubernetes API in order to create specialized automation.

SERVICE LEVEL ISSUES:

- If your service is not running properly first, we need to check labels & selectors
- next check the ports open or not & check the ingress and Network policy enbale
- kubectl get networkpolicy -A
- kubectl create deploy raham3 --image=httpd
- kubectl expose deploy raham3 --port=80 --type=NodePort
- kubectl get pods,svc
- kubectl get ep
- kubectl get pods -o wide
- kubectl get svc -o wide
- minikube ip
- curl 192.168.49.2:30933
- curl (minikubeip):(ports) : ports will be on svc