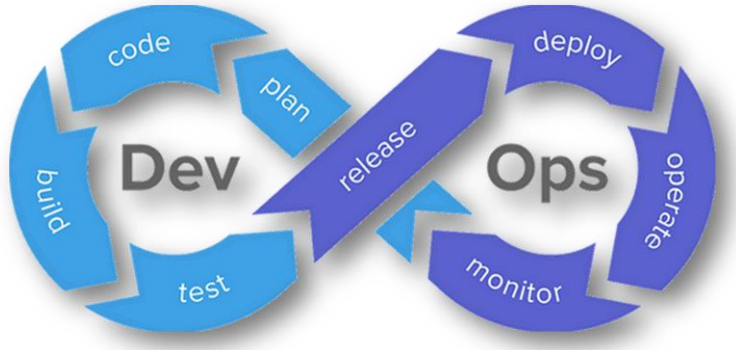
Thinknyx®
You Trust, We Deliver!

DevOps
24 hours
Day6



Kubernetes

Monitoring in K8S

Monitoring ensures visibility into **cluster health, workloads, and resource usage**.

Key Areas
Cluster monitoring → Nodes, kubelet, API server, etcd
Pod/Container monitoring → CPU, memory, network usage, restarts
Application monitoring → Business metrics exposed by the app

Monitoring in K8S

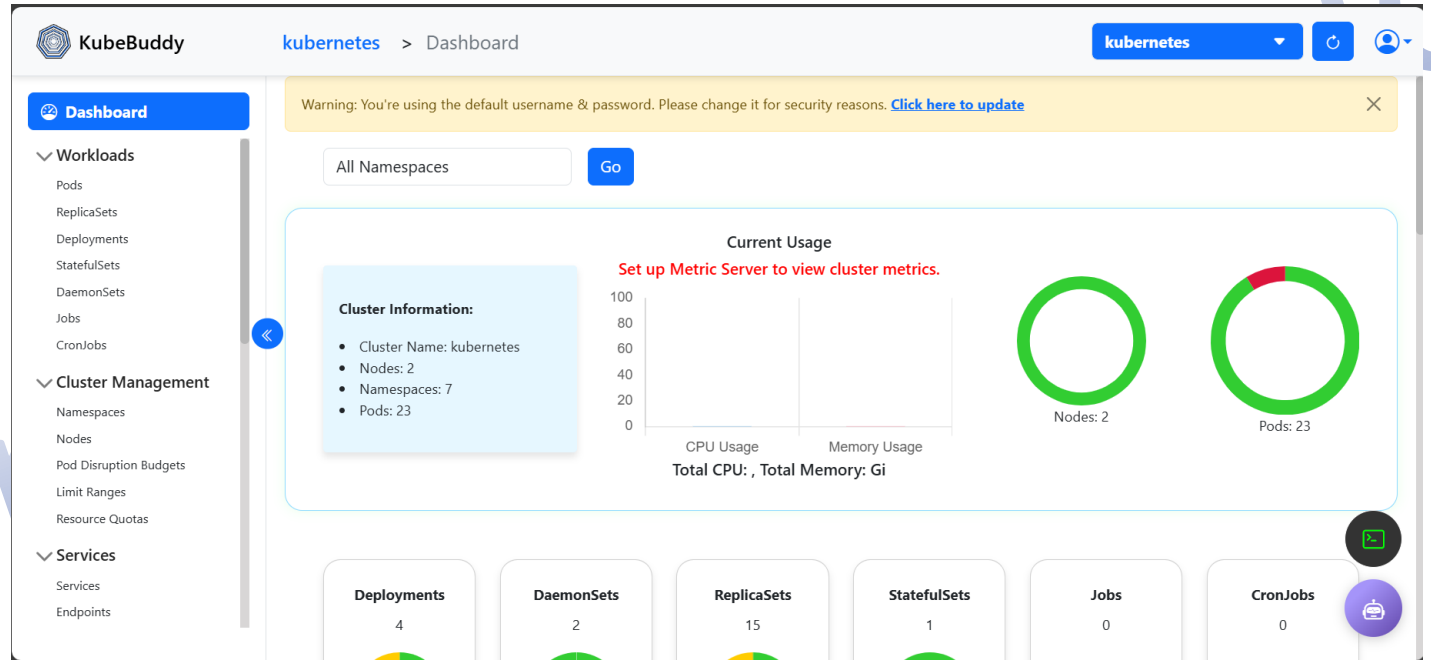
Tools & Approaches

- **Metrics Server**
  - Collects resource usage (CPU, memory).
  - Used for **Horizontal Pod Autoscaler (HPA)**.
  - Limited, not for long-term storage.

- **Prometheus** (most common)
  - Scrapes metrics exposed by kubelets, cAdvisor, apps, etc.
  - Stores time-series data.
  - Supports alerting (Alertmanager).

- **Grafana**
  - Visualizes metrics from Prometheus or Loki.

KubeBuddy Dashboard



https://kubebuddy.org/documents/#installation

Kubernetes Job

Kubernetes **Jobs** are super useful in real-world setups because many workloads aren't meant to run continuously like web servers. Some tasks need to **run once, finish, and exit** — and that's exactly what Jobs are built for.

A Kubernetes **Job** ensures that **a task runs to completion successfully**, even if pods crash or nodes fail.
If a pod fails, the Job controller automatically **retries** it until it succeeds or reaches a retry limit.
So instead of manually creating a pod, monitoring it, and cleaning it up, a Job **automates that lifecycle**.

Backup or Archival
Automate backups of databases, logs, or configurations.

Email or Report Generation
Some reports or notifications need to run only once or periodically.

File or Media Processing
When users upload files, you might need to resize images, transcode videos, or extract metadata.

Kubernetes Basic Job

job.yaml
========

```
apiVersion: batch/v1
kind: Job
metadata:
  name: hello-job
spec:
  template:
    spec:
      containers:
      - name: hello
        image: docker.io/library/busybox:latest
        command: ["echo", "Hello from Kubernetes Job!"]
      restartPolicy: Never
  backoffLimit: 3
```

```
kubectl apply -f job.yaml

kubectl get jobs

kubectl logs -l job-name=hello-job

kubectl delete job hello-job
```

Kubernetes Cron Job

cronjob.yaml
=======

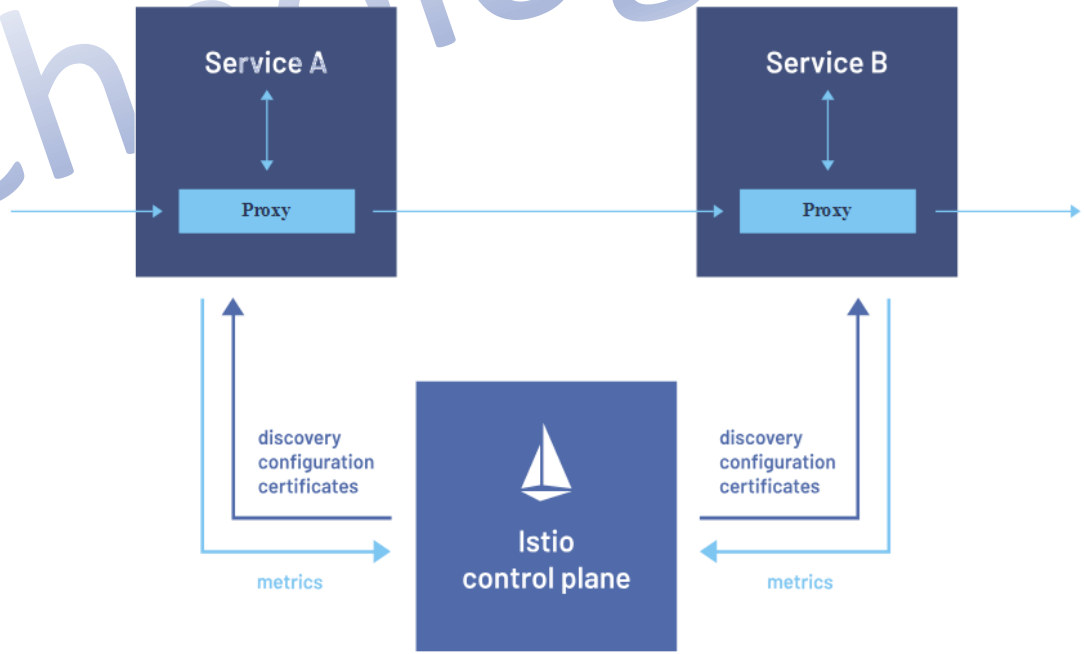
```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello-cron
spec:
  schedule: "*/1 * * * *"   # Every 1 minute
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: docker.io/library/busybox:latest
            command: ["date"]
          restartPolicy: Never
```

```
kubectl apply -f cronjob.yaml

kubectl get jobs

kubectl logs -l job-name=<job_name>

kubectl delete cronjob hello-cron
```

Istio

Istio is an **open-source service mesh** - a dedicated infrastructure layer that helps you **manage, secure, and observe communication between microservices** in a distributed application.

In simpler terms, if you have many services running (for example, in **Kubernetes**), Istio helps control how those services **talk to each other**, **securely** and **reliably**, without changing your application code.



Istio

Major Features

Traffic Management

Route traffic based on rules (e.g., version-based routing, canary deployments).

Perform load balancing, failover, and retries automatically.

Security

Enforces **mutual TLS (mTLS)** between services.

Provides authentication, authorization, and encryption of service-to-service communication.

Observability

Collects detailed telemetry data: metrics, logs, and traces.

Integrates with tools like **Prometheus** and **Grafana.**

Policy Enforcement

Control access, rate limits, quotas, and request validation centrally.