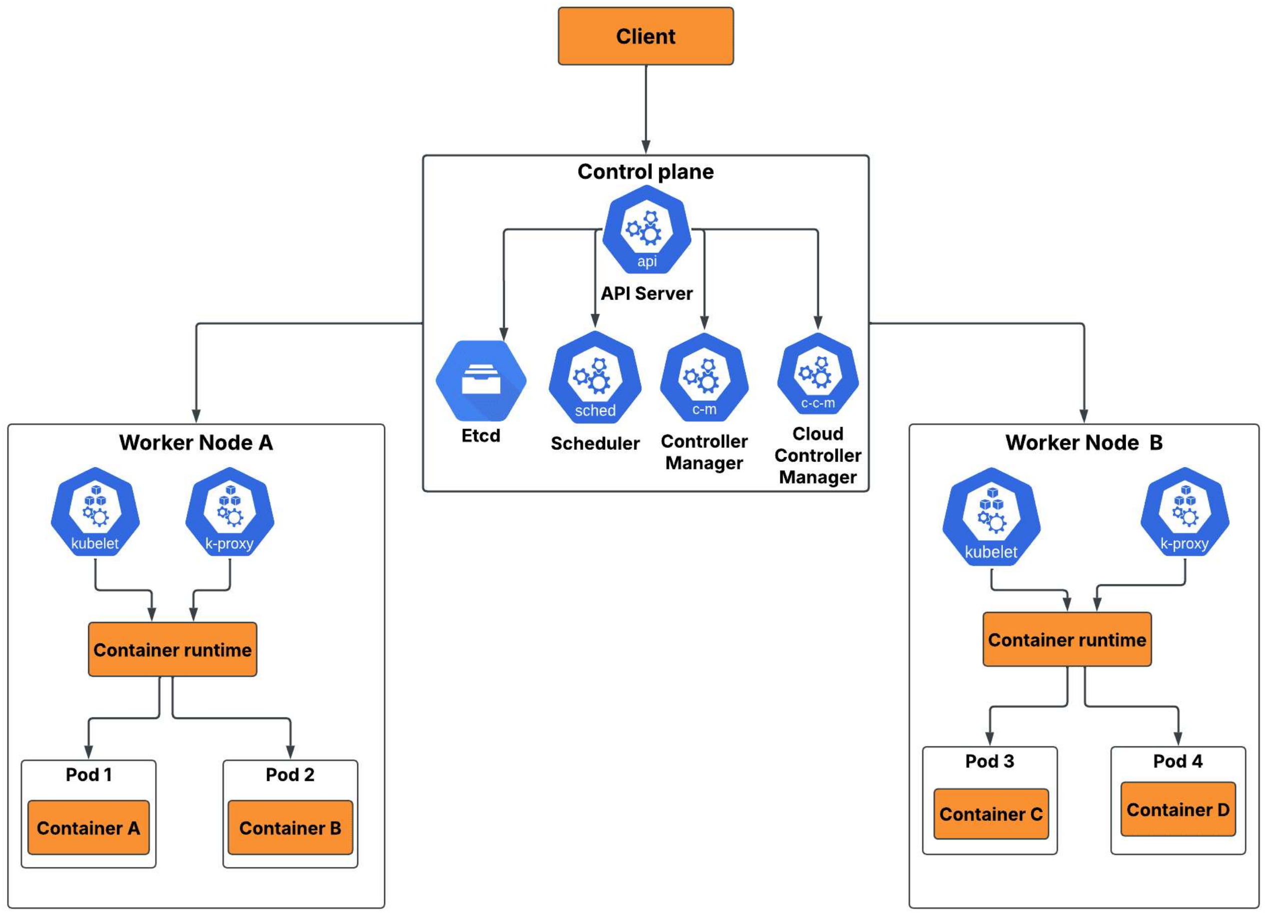
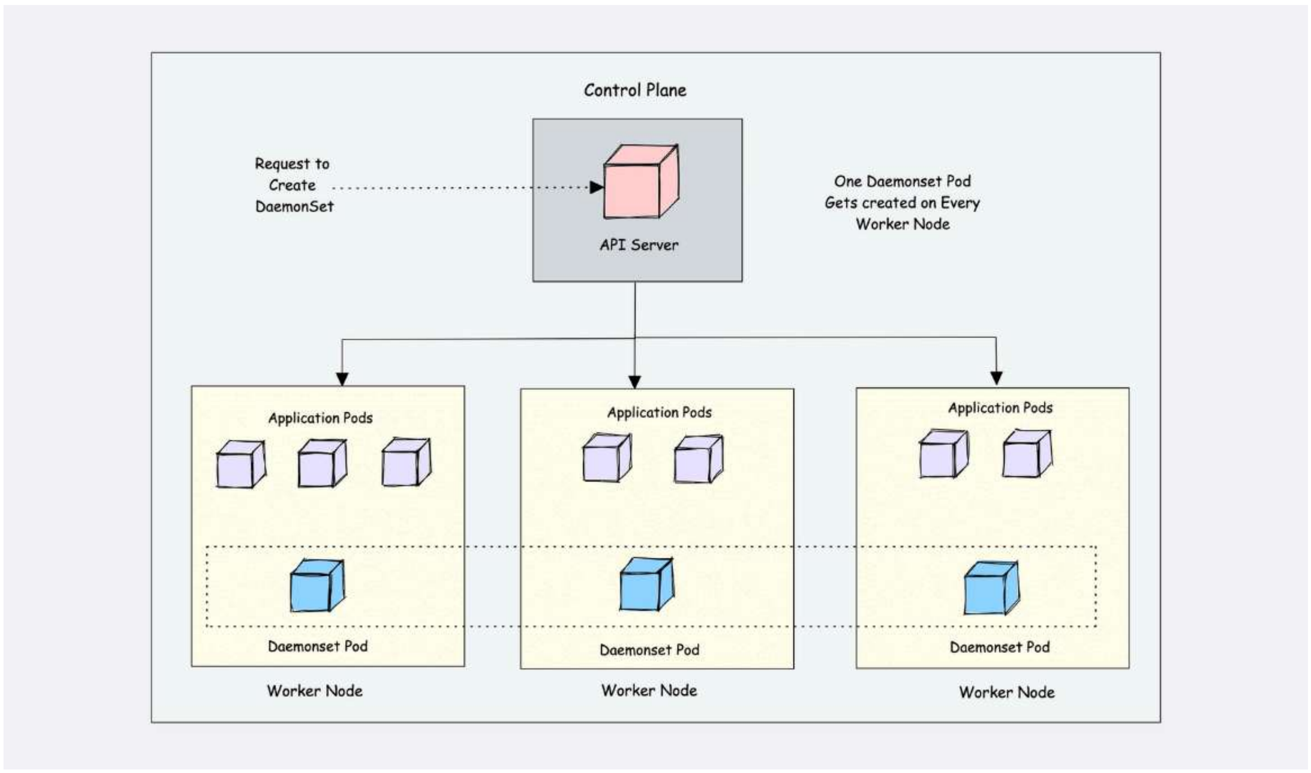
🧠 Kubernetes Workloads – Overview

Kubernetes workloads define **how containers run** inside the cluster.

Workload	Purpose
Pod	Smallest deployable unit
ReplicaController	Legacy replication
ReplicaSet	Replica management
Deployment	Rolling updates & scaling
Job	One-time task
CronJob	Scheduled task
DaemonSet	Runs on every node
Init Container	Pre-run setup
Sidecar	Supporting container

🏗️ Kubernetes Workloads Architecture





```
┌──────────────┐
│   kubectl    │
└─────┬────────┘
      │
┌─────▼─────────────────────────────────┐
│ Kubernetes Control Plane              │
│ ├─ Deployment → ReplicaSet → Pod      │
│ ├─ Job → Pod                           │
│ ├─ CronJob → Job → Pod                │
│ ├─ DaemonSet → Pod (per node)         │
└───────────────────────────────────────┘
```

📦 Workload Types Explained (With YAML)

🔹 1. Pod (`pod.yml`)

Smallest unit in Kubernetes

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
    - name: nginx
      image: httpd
      ports:
        - containerPort: 80
```

✔️ Not self-healing ❌ Not recommended for production

🔹 2. ReplicaController (`replicacontoller.yml`)

> ⚠️ **Deprecated** – replaced by ReplicaSet

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx-rc
spec:
  replicas: 3
  selector:
    app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
          ports:
            - containerPort: 80
```

✔️ Legacy clusters only

🔹 3. ReplicaSet (`replicaset.yml`)

Ensures **fixed number of pods**

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-rs
  labels:
    app: nginx
    env: dev
spec:
  replicas: 3
```

✔️ Used internally by Deployments

🔹 4. Deployment (`deployment.yml`)

Most common production workload

```
apiVersion: apps/v1
kind: Deployment
spec:
```

```yaml
  replicas: 3
  strategy:
    type: RollingUpdate
```

✓ Rolling updates ✓ Rollbacks ✓ Auto-healing

🔹 5. Job (`job.yml`)

Runs **once and exits**

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: monitored-job
spec:
  completions: 1
  template:
    spec:
      shareProcessNamespace: true # Share PID namespace between containers
      containers:
        - name: main-worker
          image: alpine:latest
          command: ["sh", "-c"]
          args:
            - |
              echo "Starting main task..."
              for i in $(seq 1 10); do
                echo "Processing item $$i"
                sleep 1
              done
              echo "Task completed"

        - name: monitor-sidecar
          image: busybox:latest
          command: ["sh", "-c"]
          args:
            - |
              echo "Monitoring main process..."
              # Monitor the main process
              while true; do
                if ps aux | grep -v grep | grep -q "main-worker"; then
                  echo "Main process is running"
                else
                  echo "Main process completed"
                  break
                fi
                sleep 5
              done

      restartPolicy: Never
```

✔ DB migrations ✔ Backup jobs

🔹 6. CronJob (`cron-job.yml`)

Scheduled Jobs (Linux cron style)

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: myjob
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox:1.28
            command:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

✔ Log cleanup ✔ Reports ✔ Batch processing

🔹 7. DaemonSet (`daemonset.yml`)

Runs **one pod per node**

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: node-exporter
  namespace: monitoring
  labels:
    k8s-app: node-exporter
spec:
  selector:
    matchLabels:
      k8s-app: node-exporter
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
  template:
    metadata:
      labels:
        k8s-app: node-exporter
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "9100"
        prometheus.io/path: "/metrics"
    spec:
      hostNetwork: true
      hostPID: true
      tolerations:
        - effect: NoSchedule
          operator: Exists
      containers:
        - name: node-exporter
          image: prom/node-exporter:v1.5.0
          args:
            - --path.rootfs=/host/root
            - --path.procfs=/host/proc
            - --path.sysfs=/host/sys
            - --web.listen-address=:9100
            - --collector.filesystem.mount-points-
exclude=^/(sys|proc|dev|host|etc)($$|/)
          ports:
            - containerPort: 9100
              hostPort: 9100
              name: metrics
              protocol: TCP
          resources:
            requests:
              memory: 100Mi
              cpu: 100m
            limits:
              memory: 200Mi
              cpu: 200m
          securityContext:
            runAsNonRoot: true
            runAsUser: 65534
          volumeMounts:
```

```yaml
            - name: rootfs
              mountPath: /host/root
              readOnly: true
            - name: proc
              mountPath: /host/proc
              readOnly: true
            - name: sys
              mountPath: /host/sys
              readOnly: true
          livenessProbe:
            httpGet:
              path: /
              port: 9100
            initialDelaySeconds: 30
            timeoutSeconds: 5
          readinessProbe:
            httpGet:
              path: /
              port: 9100
            initialDelaySeconds: 30
            timeoutSeconds: 5
      volumes:
        - name: rootfs
          hostPath:
            path: /
        - name: proc
          hostPath:
            path: /proc
        - name: sys
          hostPath:
            path: /sys
```

✔ Log collectors ✔ Monitoring agents ✔ Security agents

◆ 8. Init Container (`init-container.yml`)

Runs **before app container starts**

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: init-container-app
  labels:
    app: init-container-app
spec:
  replicas: 3
```

```yaml
    selector:
      matchLabels:
        app: init-container-app
    template:
      metadata:
        labels:
          app: init-container-app
      spec:
        initContainers:
          - name: wait-for-backend
            image: busybox
            command: ["/bin/sh", "-c"]
            args: ["until nslookup google.com; do sleep 2 || true; done"]
        containers:
          - name: nginx-container
            image: nginx
            ports:
              - containerPort: 80
            resources:
              limits:
                memory: "128Mi"
                cpu: "500m"
```

✔ DB wait ✔ Config generation

🔹 9. Sidecar Container (`sidecar.yml`)

Runs **alongside main container**

```yaml
apiVersion: v1
kind: Deployment
metadata:
  name: app-with-sidecar
  labels:
    app: myapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
```

```yaml
      # Main application container
      - name: main-app
        image: nginx:latest
        ports:
          - containerPort: 80
    volumeMounts:
      - name: shared-logs
        mountPath: /var/log/nginx
      - name: shared-data
        mountPath: /shared-data

  # Sidecar container
  - name: log-collector-sidecar
    image: fluent/fluentd:latest
    volumeMounts:
      - name: shared-logs
        mountPath: /var/log/nginx
      - name: config-volume
        mountPath: /fluentd/etc
    command: ["/bin/sh", "-c"]
    args:
      - fluentd -c /fluentd/etc/fluentd.conf

  # Another sidecar example (for metrics)
  - name: metrics-exporter-sidecar
    image: prom/node-exporter:latest
    ports:
      - containerPort: 9100
    securityContext:
      runAsUser: 65534 # nobody user for security

volumes:
  - name: shared-logs
    emptyDir: {}
  - name: shared-data
    emptyDir: {}
  - name: config-volume
    configMap:
      name: fluentd-config
```

✔️ Logging ✔️ Proxy ✔️ Metrics exporter

🔄 Execution Flow (Very Important)

```
Init Container → App Container + Sidecar
                          |
              Liveness / Readiness
```

```
CronJob → Job → Pod → Complete → Exit
```

🚀 Apply All Manifests

```
kubectl apply -f pod.yml
kubectl apply -f replicaset.yml
kubectl apply -f deployment.yml
kubectl apply -f job.yml
kubectl apply -f cron-job.yml
kubectl apply -f daemonset.yml
kubectl apply -f init-container.yml
kubectl apply -f sidecar.yml
```

🔍 Verification (SRE Commands)

```
kubectl get pods
kubectl get deploy
kubectl get rs
kubectl get jobs
kubectl get cronjobs
kubectl get ds
```

Describe deeply:

```
kubectl describe pod <pod-name>
```

🌍 Real-World Use Cases

Scenario	Workload
Web App	Deployment
Database Migration	Job
Daily Backup	CronJob
Logging Agent	DaemonSet
App Bootstrap	Init Container
Log Shipping	Sidecar

✅ Best Practices (Production)

✔️ Never use Pods directly in prod ✔️ Always use Deployments ✔️ Prefer CronJob over OS cron ✔️ DaemonSet for node-level tasks ✔️ Init containers for dependencies ✔️ Sidecars for observability