Kubernetes Jobs and Networking

starts at 9:05 pm

Agenda

1. Jobs
    1. Completions and Parallelism
2. Cron jobs
3. Services
    1. Types of Services
4. Network Policies

→ Jobs

Backup Tasks

- Periodic backups of a database (e.g., MySQL, PostgreSQL)
- Exporting logs or data snapshots to cloud storage

- **Batch Processing**
    - Generating reports (e.g., monthly sales reports)
    - Running analytics jobs on collected data

- **Cleanup Jobs**
    - Deleting old logs, cache, or expired records
    - Cleaning up orphaned resources

- **Data Import/Export**
    - Fetching external data and importing it into a database

- Sending bulk emails or notifications

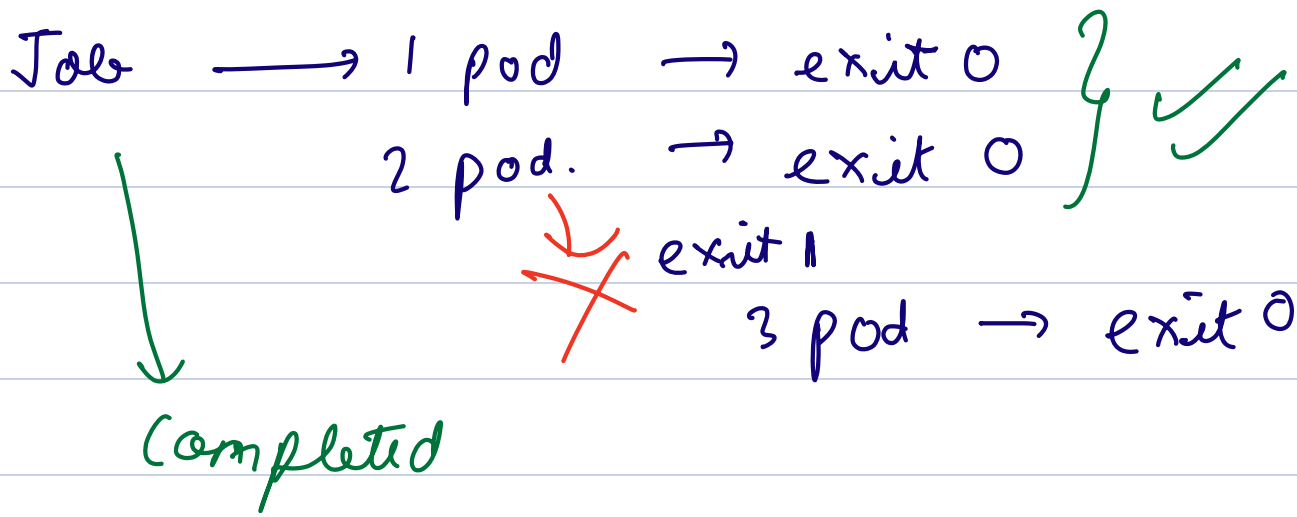
- **Security Scans & Compliance Checks**

  - Running vulnerability scans on application containers

  - Checking compliance rules on cloud resources

Jobs

In Kubernetes, **jobs** and **cron jobs** are essential for running tasks that are not part of the main application but need to be executed periodically or as one-off tasks.



backoff limit → 4

Container fails 4 times

Job → Failed

→ Completion and Parallelism

how many pods need to run?
how many times the job needs to complete successfully?

Completion →

Defines how many successful runs are required before the job is considered complete.

default → 1

Parallelism →
→ how many pods can run at a time
default → 1

spec:

completions: 5

parallelism: 1

2 → 2 pods.
2 pods
1 pod
} Completed

spec:

completions: 10

parallelism: 5

Scenario	Completions	Parallelism	Use Case
One-time job	`1`	`1`	Database migration, cleanup tasks
Fixed number of runs (sequential)	`N`	`1`	Data processing in sequence
Fixed number of runs (parallel)	`N`	`M`	Large-scale computations

Key Parameters:

- `completions`: The desired number of successful pod completions.

- `parallelism`: The number of pods to run in parallel.

- `backoffLimit`: The number of retries before the job is considered failed.

- `activeDeadlineSeconds`: The time duration before the job is considered failed, even if it's still running.

Case 1 backofflimit

apiVersion: batch/v1

kind: Job

metadata:

  name: demo-job

spec:

  completions: 5   # Total 5 successful runs required

  parallelism: 2   # Run 2 pods at a time

  backoffLimit: 3   # Retry a failed pod 3 times before marking job as failed

```
  activeDeadlineSeconds: 60  # Job must finish within 60 seconds
  template:
    spec:
      containers:
      - name: worker
        image: busybox
        command: ["sh", "-c", "echo Running task; sleep 10; exit 1"]
      restartPolicy: Never
apiVersion: batch/v1
kind: Job
metadata:
  name: demo-job
spec:
  completions: 5       # Total 5 successful runs required
  parallelism: 2       # Run 2 pods at a time
  backoffLimit: 3      # Retry a failed pod 3 times before marking job as failed
  activeDeadlineSeconds: 10  # Job must finish within 60 seconds




  template:
    spec:
      containers:
```

```
    - name: worker
      image: busybox
      command: ["sh", "-c", "echo Running task; sleep 10; exit 0"]
    restartPolicy: Neve
r
apiVersion: batch/v1
kind: Job
metadata:
  name: demo-job
spec:
  completions: 5       # Total 5 successful runs required
  parallelism: 2       # Run 2 pods at a time
  backoffLimit: 3      # Retry a failed pod 3 times before marking job as failed
  activeDeadlineSeconds: 60  # Job must finish within 60 seconds


  template:
    spec:
      containers
 name: worker
      image: busybox
      command: ["sh", "-c", "echo Running task; sleep 10; exit 0"]
    restartPolicy: Never
```

Cron Job

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: demo-cronjob
spec:
  schedule: "*/2 * * * *"  # Runs every 2 minutes
  jobTemplate:
    spec:
      completions: 5        # Each Job requires 5 successful pods
      parallelism: 2        # Run 2 pods at a time
      backoffLimit: 3       # Retry a failed pod 3 times
      activeDeadlineSeconds: 60  # Each job must complete in 60 seconds
      template:
        spec:
          containers:
          - name: worker
            image: busybox
            command: ["sh", "-c", "echo Running task; sleep 10"]
          restartPolicy: Never
```

Cronjob → schedule.

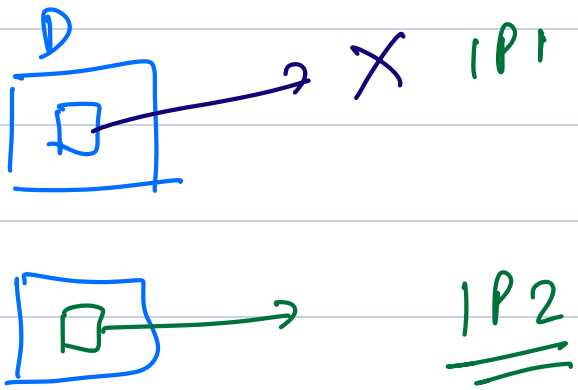
↓ 2 mins.

→ Job → Pod

kubectl patch cronjob demo-cronjob -p '{"spec": {"suspend": true}}'

Break → 10:20 pm

→ Services



a **Service** is an abstraction that provides **stable networking** for a group of Pods.

Office Building → K8s cluster

multiple employees → Pods.

↓

desk phone → IP

helpline number → Service.

Route ↓ ↓ Route

correct employee Pods

Types of services.

① Cluster IP

→ used for communication only within the cluster

```
metadata:
  labels:
    app: myapp
```

} → Pod.

```
selector:
  app: myapp
```

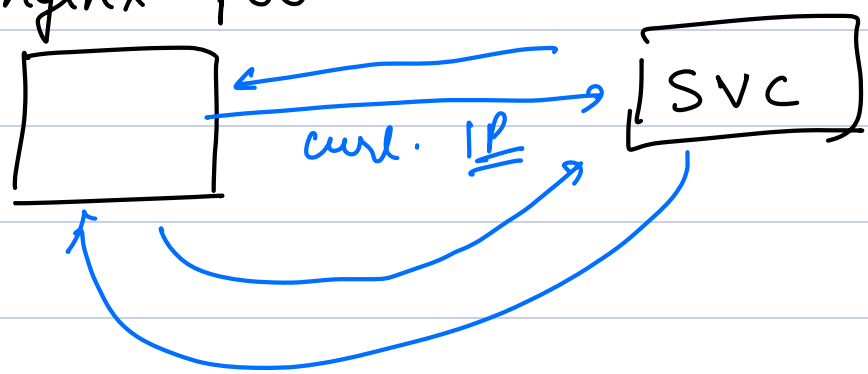
} → service.

→ Pod

→ CM

→ Service

→ nginx-pod.



kubectl port-forward resource LOCAL_PORT:TARGET_PORT

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
```

```
    image: nginx
    ports:
    - containerPort: 80
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
  volumes:
  - name: html
    configMap:
      name: nginx-html
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-html
data:
  index.html: |
    <html><body><h1>Welcome to My Nginx Pod</h1></body></html>
---
apiVersion: v1
kind: Service
metadata:
```

```
  name: nginx-service
spec:
  selector:
    app: nginx  # Matches the pod label
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: ClusterIP  # Internal service
```

Alternative way to expose → Nodeport.

nodeport range. 30000 – 32767

Receptionist → Nodeport.

http://Node_IP:NodePort

```
ports:
  - protocol: TCP
    port: 80
    targetPort: 80
    nodePort: 30007
```

→ curl

↓

reaches nodeport on your node.

↓

:80 of the service.

↓

target port 80 → on the container.

Demo.

3 pods. 3 CMs.

→ NP service.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod-1
  labels:
    app: nginx
    pod: "1"
spec:
  containers:
```

```
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
  volumes:
  - name: html
    configMap:
      name: nginx-html-1
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-html-1
data:
  index.html: |
    <html><body><h1>Welcome to Pod 1</h1></body></html>
---
apiVersion: v1
kind: Pod
```

```
metadata:
  name: nginx-pod-2
  labels:
    app: nginx
    pod: "2"
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
  volumes:
  - name: html
    configMap:
      name: nginx-html-2
---
apiVersion: v1
kind: ConfigMap
metadata:
```

```
  name: nginx-html-2
data:
  index.html: |
    <html><body><h1>Welcome to Pod 2</h1></body></html>
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod-3
  labels:
    app: nginx
    pod: "3"
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
  volumes:
```

```
  - name: html
    configMap:
      name: nginx-html-3
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-html-3
data:
  index.html: |
    <html><body><h1>Welcome to Pod 3</h1></body></html>
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx  # Selects all pods with the app label 'nginx'
  type: NodePort  # Exposes the service externally on every node
  ports:
    - protocol: TCP
```

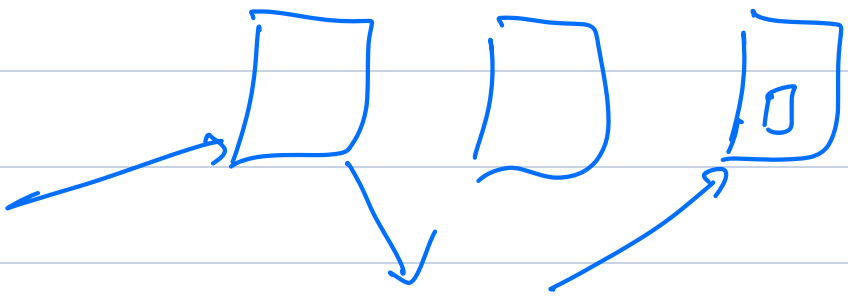
port: 80 # The service port

targetPort: 80 # The port on the pod where traffic is forwarded

nodePort: 30007 # The external port to access the service

→ Drawbacks.

① Limited Port Ranges.
② Requires access to node IPs.
③ Security.
④ Not Proper Load Balancing.



→ LoadBalancer Service.

Step-by-Step Flow of NLB Provisioning in EKS

1. **You define a Kubernetes `Service` of type `LoadBalancer`**

   - You include the annotation `service.beta.kubernetes.io/aws-load-balancer-type: "nlb"`

2. **Kubernetes Cloud Controller Manager (CCM) detects the LoadBalancer request**

   - The **EKS CCM** (built into the control plane) watches for services of type `LoadBalancer`.

- It **communicates with AWS APIs** to provision an NLB.

3. **AWS Automatically Creates and Configures an NLB**

   - An **AWS Network Load Balancer** is created.

   - A **static external IP (Elastic IP) is assigned** if needed.

   - The NLB forwards traffic to the Kubernetes **worker nodes** where the service's pods are running.

4. **Traffic is sent to pods**

   - If `target-type: "ip"` is specified, NLB directly forwards traffic to the pod IPs instead of NodePorts.

```
apiVersion: v1
kind: Service
metadata:
  name: my-nlb-service
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
    service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "ip"
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
```

```
selector:
  app: my-app
```