Scheduling:

In Kubernetes, scheduling refers to the process of assigning Pods to Nodes so that the Kubelet can run them.

Scheduling Workflow:

1. Pod Creation: When you deploy a pod using a manifest file or kubectl commands, it enters an unscheduled state.
2. Scheduler Evaluation: The scheduler continuously scans the list of unscheduled pods and available nodes.
3. Predicate Checks: For each pod, the scheduler evaluates the predicates against all nodes in the cluster. Nodes that don't meet the pod's requirements are filtered out.
4. Priority Scoring: If multiple nodes pass the predicate checks, the scheduler assigns a priority score to each based on pre-defined rules (e.g., node with most free resources gets a higher score).
5. Pod Binding: The scheduler selects the node with the highest priority score and binds the pod to that node. This makes the pod "scheduled."
6. Kubelet Communication: The Kubernetes API server informs the kubelet (agent running on each node) about the scheduled pod.
7. Pod Execution: The Kubelet on the designated node pulls the container image(s) from the container registry, creates and starts the container(s) within the pod, and allocates the necessary resources.

Ways to Achive Scheduling:

By default, Kubernetes implements a built-in scheduling framework that you don't need to configure manually. However, you can influence the scheduling process in a few ways.

1. Manual Scheduling
2. Pod Annotation and Labels
3. Node Selectors
4. Node Affinities
5. Taint and Tolerations
6. Custom Schedulers
7. DaemonSets

Manual Scheduling

It’s like manually binding pods with specific labels or names to a specific Node

Example: manual-scheduling.yaml (in this directory)

Taints and Tolerations

Taint-Effects are:

1. NoSchedule (Pods without a toleration for the taint will not be scheduled on the tainted node.)
2. PreferNoSchedule (Pods without a toleration are still eligible for scheduling but penalized during the process, making other untainted nodes more favorable)
3. NoExcuse (Pod is evicted from the node if it is already running on the node, and is not scheduled onto the node if it is not yet running on the node.)

Tainting a node commands

- `kubectl taint nodes node-name key=value:taint-effect`
- `kubectl taint nodes node02 color=blue:NoSchedule`

Toleration of Pod definition file:

In this current directory there's a file called "pod-tolerations" which will explain the formats.

Node Selectors:

We can run a pod on a specific Node by labelling a Node and then mentioning it in the nodeSelector section of the pod definition file. By doing this the pod will directly assign to the specific Node.

This is helpful in case of basic pod assignment to node.

To achieve this we need to label the node first by this command:

`kubectl label nodes node-name key=value`

After this we can mention the labels in pod definition file in nodeSelector section

Example: node-selector.yaml in this directory

Node Affinity:

Primary reason for this way is to make sure pod launches on specific Node. It features advanced Node selection capabilities then Node Selectors. It's little complex then nodeSelector but it provides additional features then nodeSelector.

Example: node-affinity.yaml

Taints and Tolerations vs Node Affinity

Feature	Node Affinity	Taints and Tolerations
Purpose	Preference for scheduling	Mandatory scheduling rules or isolation
Implementation	Defined within pod spec	Taints applied to nodes, tolerations in pods
Approach	Positive selection (what's desired)	Restrictive (what's not allowed, exceptions)
Flexibility	More flexible for various needs	More strict for enforcing specific rules

DaemonSets

It ensures that one copy of the pod run on every nodes.

Example: monitoring tools or Log Viewer etc. another example is Kube-proxy which runs on every node.

Definition file present in daemonSet.yaml

Commands:

```
Kubectl create daemonsets name --image image-name

Kubectl get daemonsets

Kubectl describe daemonsets name
```