Node selection in kube-scheduler:

kube-scheduler selects a node for the pod in a 2-step operation:

A. Filtering
B. Scoring

Filtering:

The filtering step finds the set of Nodes where it's feasible to schedule the Pod. For example, the PodFitsResources filter checks whether a candidate Node has enough available resource to meet a Pod's specific resource requests. After this step, the node list contains any suitable Nodes; often, there will be more than one. If the list is empty, that Pod isn't (yet) schedulable.

In the scoring step, the scheduler ranks the remaining nodes to choose the most suitable Pod placement. The scheduler assigns a score to each Node that survived filtering, basing this score on the active scoring rules.

Finally, kube-scheduler assigns the Pod to the Node with the highest ranking. If there is more than one node with equal scores, kube-scheduler selects one of these at random.

There are two supported ways to configure the filtering and scoring behavior of the scheduler:

Scheduling Policies allow you to configure Predicates for filtering and Priorities for scoring.
Scheduling Profiles allow you to configure Plugins that implement different scheduling stages, including: QueueSort, Filter, Score, Bind, Reserve, Permit, and others. You can also configure the kube-scheduler to run different profiles

Reference: https://kubernetes.io/docs/reference/scheduling/policies/

Predicates

The following predicates implement filtering:

- **PodFitsHostPorts:** Checks if a Node has free ports (the network protocol kind) for the Pod ports the Pod is requesting.
- **PodFitsHost:** Checks if a Pod specifies a specific Node by its hostname.
- **PodFitsResources:** Checks if the Node has free resources (eg, CPU and Memory) to meet the requirement of the Pod.
- **MatchNodeSelector:** Checks if a Pod's Node Selector matches the Node's label(s).
- **NoVolumeZoneConflict:** Evaluate if the Volumes that a Pod requests are available on the Node, given the failure zone restrictions for that storage.

- **NoDiskConflict:** Evaluates if a Pod can fit on a Node due to the volumes it requests, and those that are already mounted.
- **MaxCSIVolumeCount:** Decides how many CSI volumes should be attached, and whether that's over a configured limit.
- **PodToleratesNodeTaints:** checks if a Pod's tolerations can tolerate the Node's taints.
- **CheckVolumeBinding:** Evaluates if a Pod can fit due to the volumes it requests. This applies for both bound and unbound PVCs

Priorities:

The following priorities implement scoring:

- **SelectorSpreadPriority:** Spreads Pods across hosts, considering Pods that belong to the same Service, StatefulSet or ReplicaSet.

- **InterPodAffinityPriority:** Implements preferred inter pod affininity and antiaffinity.

- **LeastRequestedPriority:** Favors nodes with fewer requested resources. In other words, the more Pods that are placed on a Node, and the more resources those Pods use, the lower the ranking this policy will give.

- **MostRequestedPriority:** Favors nodes with most requested resources. This policy will fit the scheduled Pods onto the smallest number of Nodes needed to run your overall set of workloads.

- **RequestedToCapacityRatioPriority:** Creates a requested To Capacity based Resource Allocation Priority using default resource scoring function shape.

- **BalancedResourceAllocation:** Favors nodes with balanced resource usage.

- **NodePreferAvoidPodsPriority:** Prioritizes nodes according to the node annotation scheduler.alpha.kubernetes.io/preferAvoidPods. You can use this to hint that two different Pods shouldn't run on the same Node.

- **NodeAffinityPriority:** Prioritizes nodes according to node affinity scheduling preferences indicated in PreferredDuringSchedulingIgnoredDuringExecution. You can read more about this in Assigning Pods to Nodes.

- **TaintTolerationPriority:** Prepares the priority list for all the nodes, based on the number of intolerable taints on the node. This policy adjusts a node's rank taking that list into account.

- **ImageLocalityPriority:** Favors nodes that already have the container images for that Pod cached locally.

- **ServiceSpreadingPriority:** For a given Service, this policy aims to make sure that the Pods for the Service run on different nodes. It favours scheduling onto nodes that don't have Pods for the service already assigned there. The overall outcome is that the Service becomes more resilient to a single Node failure.
- **EqualPriority:** Gives an equal weight of one to all nodes.
- **EvenPodsSpreadPriority:** Implements preferred pod topology spread constraints.

Weight [Required] :
int64 the numeric multiplier for the node scores that the prioritize call generates. The weight should be a positive integer

Topology spread contstraints.

Reference: https://kubernetes.io/docs/concepts/workloads/pods/pod-topology-spread-constraints/