EKS Launch Template Configuration Cheat Sheet

Scenario	update_default_version	version	Behavior	When to Use
Initial Creation	0	"1" or "$Default"	Creates LT v1, nodes use v1	First-time setup
Safe Updates	0	aws_launch_template.example.latest_version	Creates new LT version, rolls out nodes	✅ Recommended for all updates
Dangerous	1	"$Latest"	❌ Race condition, suspended nodes	❌ Never use this
No Rollout	0	"5" (pinned)	Creates new LT version but no rollout	Testing changes safely
Broken	1	"5" (pinned)	Wasted change: updates default but no rollout	❌ Never use this

When we have new changes

- Do this:
  - update_default_version = false
  - version = aws_launch_template.example.latest_version
- Not This:
  - update_default_version = true
  - version = $Latest

Rollout Behavior:

Configuration	Creates New Version	Rolls Out Nodes	Safe
update_default_version=false + version=latest_version	✅ Yes	✅ Yes	✅ Yes
update_default_version=false + version="5"	✅ Yes	❌ No	✅ Yes
update_default_version=true + version="5"	✅ Yes	❌ No	❌ No
update_default_version=true + version="$Latest"	✅ Yes	✅ Yes	❌ No

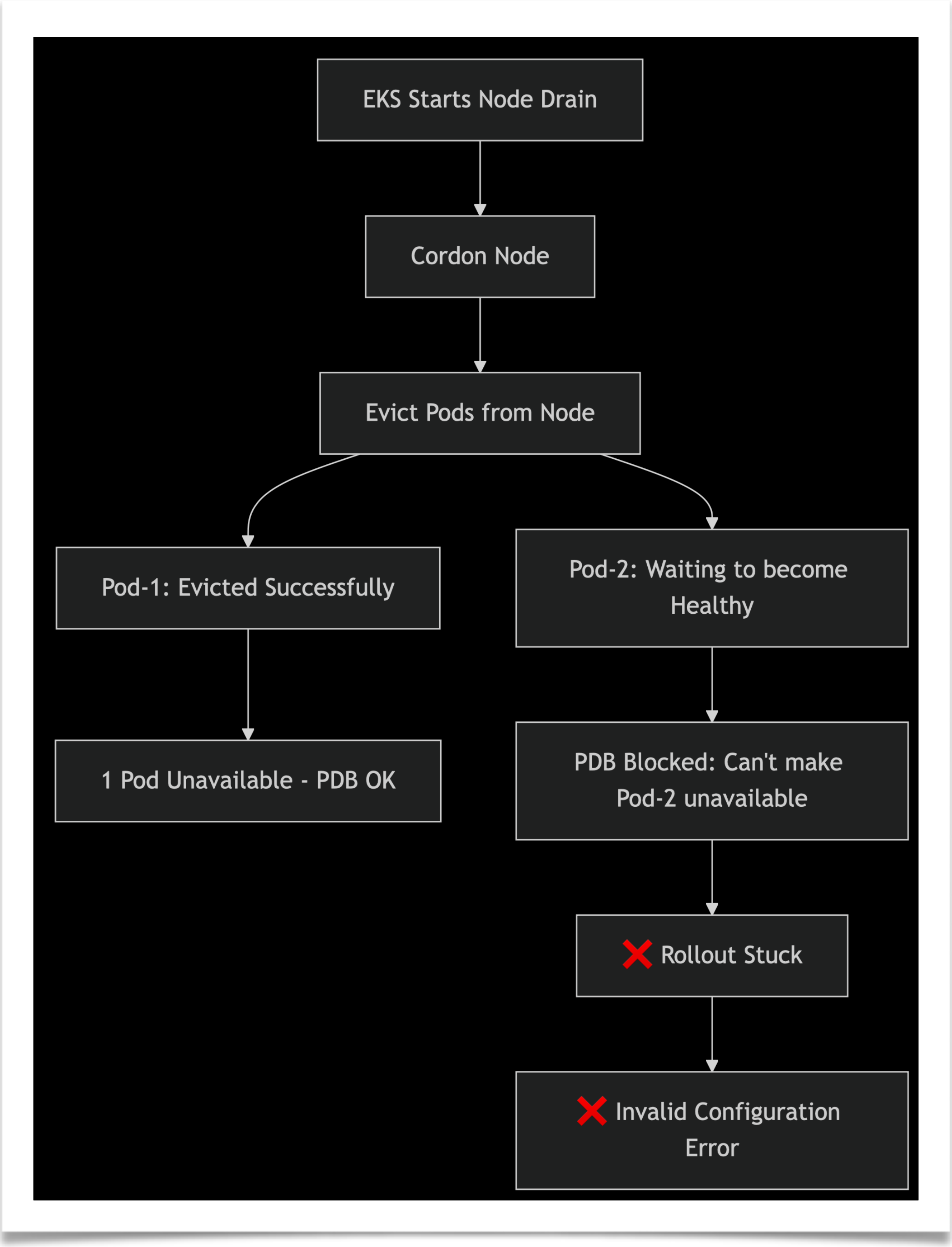
Recovery from suspended state

```
# 1. Check status
aws eks describe-nodegroup --cluster-name my-cluster --nodegroup-name my-ng

# 2. If suspended, update to known good version
aws eks update-nodegroup-config --cluster-name my-cluster --nodegroup-name my-ng   --launch-template version="32"  # Previous known good

# 3. Wait for recovery
```

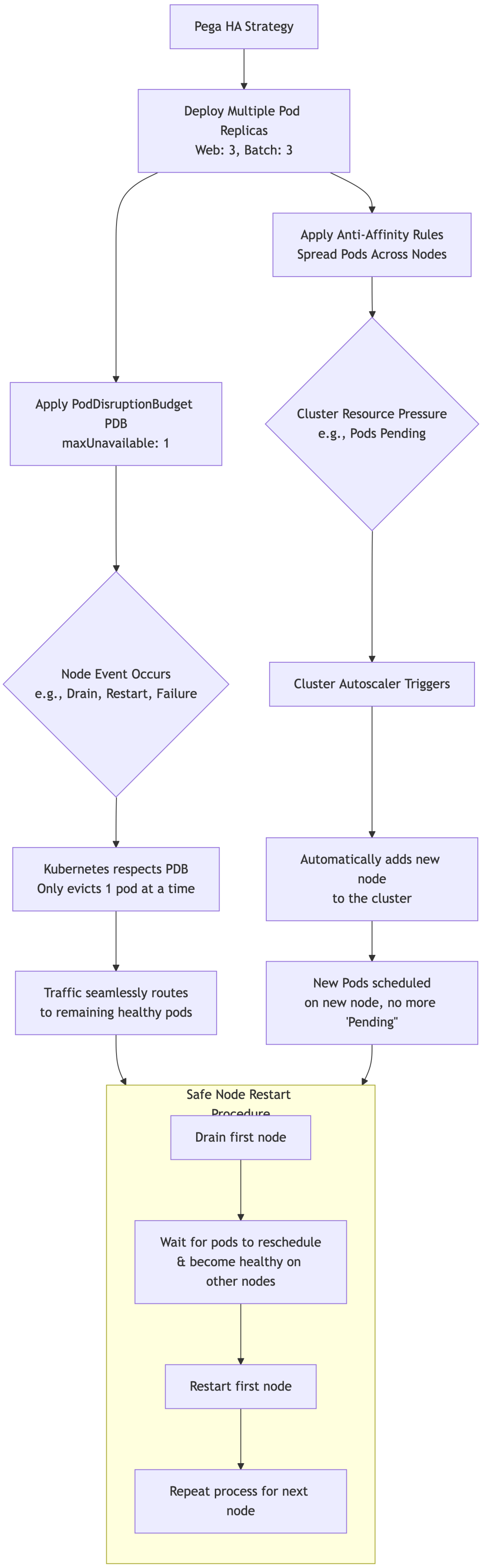
Workflow of Nodes restart


EKS Starts Node Drain
Cordon Node
Evict Pods from Node
Pod-1: Evicted Successfully
Pod-2: Waiting to become Healthy
1 Pod Unavailable - PDB OK
PDB Blocked: Can't make Pod-2 unavailable
❌ Rollout Stuck
❌ Invalid Configuration Error

EKS HA : Workflow diagram that visually explains the high-availability strategy, followed by a detailed breakdown of each component.

High-Level HA Workflow Diagram

This diagram shows the logical flow of how the different Kubernetes concepts work together to achieve high availability

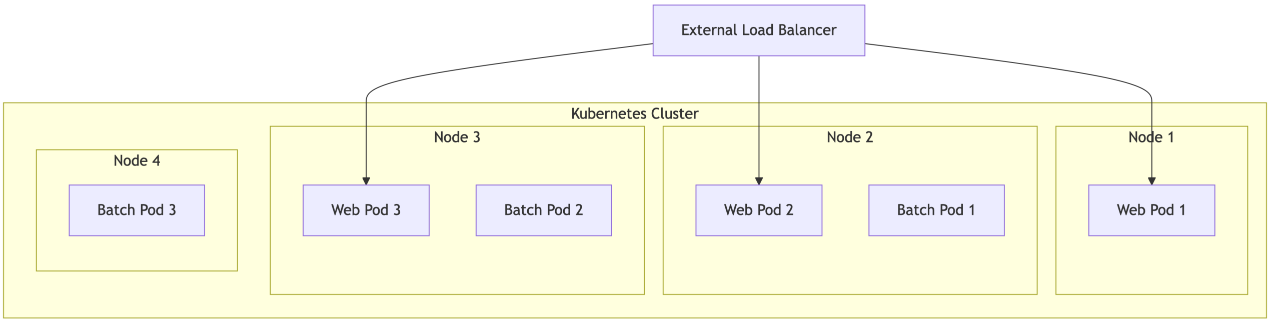

Pega HA Strategy
Deploy Multiple Pod Replicas Web: 3, Batch: 3
Apply Anti-Affinity Rules Spread Pods Across Nodes
Apply PodDisruptionBudget PDB maxUnavailable: 1
Cluster Resource Pressure e.g., Pods Pending
Node Event Occurs e.g., Drain, Restart, Failure
Cluster Autoscaler Triggers
Kubernetes respects PDB Only evicts 1 pod at a time
Automatically adds new node to the cluster
Traffic seamlessly routes to remaining healthy pods
New Pods scheduled on new node, no more 'Pending'
Safe Node Restart Procedure
Drain first node
Wait for pods to reschedule & become healthy on other nodes
Restart first node
Repeat process for next node

Detailed Breakdown of Components

The diagram above is built from the following key components working in concert.

1. Pod Replicas & Anti-Affinity: The Foundation

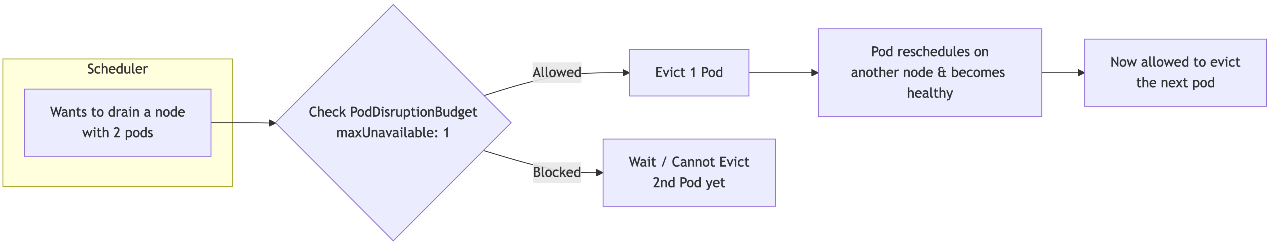
This is the base deployment, ensuring multiple copies of your service are running and placed on different physical nodes.



Key Insight: By using anti-affinity rules, you ensure no two web pods or two batch pods are on the same node. If **Node 2** fails, you only lose Web Pod 2 and Batch Pod 1. The other web and batch pods on Nodes 1, 3, and 4 continue serving traffic, preventing a full outage.

2. PodDisruptionBudget (PDB): The Safety Guard

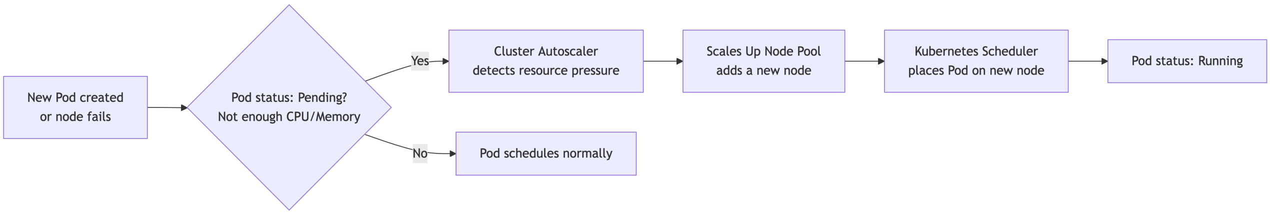
The PDB acts as a buffer during voluntary disruptions (like node restarts or upgrades).


Scheduler
Wants to drain a node with 2 pods
Check PodDisruptionBudget maxUnavailable: 1
Allowed
Evict 1 Pod
Pod reschedules on another node & becomes healthy
Now allowed to evict the next pod
Blocked
Wait / Cannot Evict 2nd Pod yet

Key Insight: The PDB prevents Kubernetes from accidentally taking down too many pods of a single type at once, ensuring minimum capacity is always maintained during operations.

3. Cluster Autoscaler: The Capacity Provider

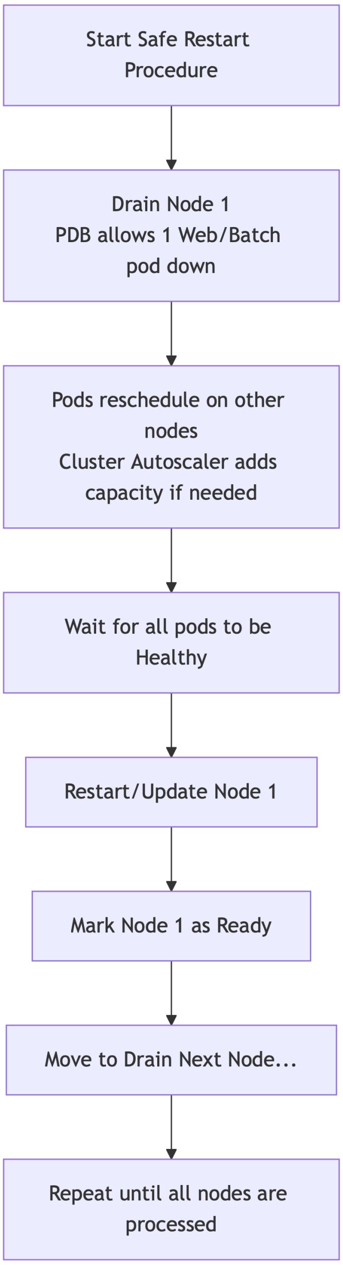
This ensures there is always enough infrastructure for the pods to run on.


New Pod created or node fails
Pod status: Pending? Not enough CPU/Memory
Yes
Cluster Autoscaler detects resource pressure
Scales Up Node Pool adds a new node
Kubernetes Scheduler places Pod on new node
Pod status: Running
No
Pod schedules normally

Key Insight: The autoscaler eliminates the "no room to schedule" failure mode, which is a common cause of downtime in poorly managed clusters.

4. Safe Node Restarts: The Procedure

This is the active process of applying all the above concepts to maintain the cluster without downtime.


Start Safe Restart Procedure
Drain Node 1
PDB allows 1 Web/Batch pod down
Pods reschedule on other nodes
Cluster Autoscaler adds capacity if needed
Wait for all pods to be Healthy
Restart/Update Node 1
Mark Node 1 as Ready
Move to Drain Next Node...
Repeat until all nodes are processed

Key Insight: This deliberate, one-by-one process leverages all the other HA mechanisms (Replicas, PDB, Anti-Affinity, Autoscaler) to guarantee service continuity.

One-Line Explanation for the Team

“We keep 3 pods for each service, spread them across nodes, only allow 1 pod down at a time, let the cluster add new nodes when needed, and restart nodes one by one. That’s how we get HA.”