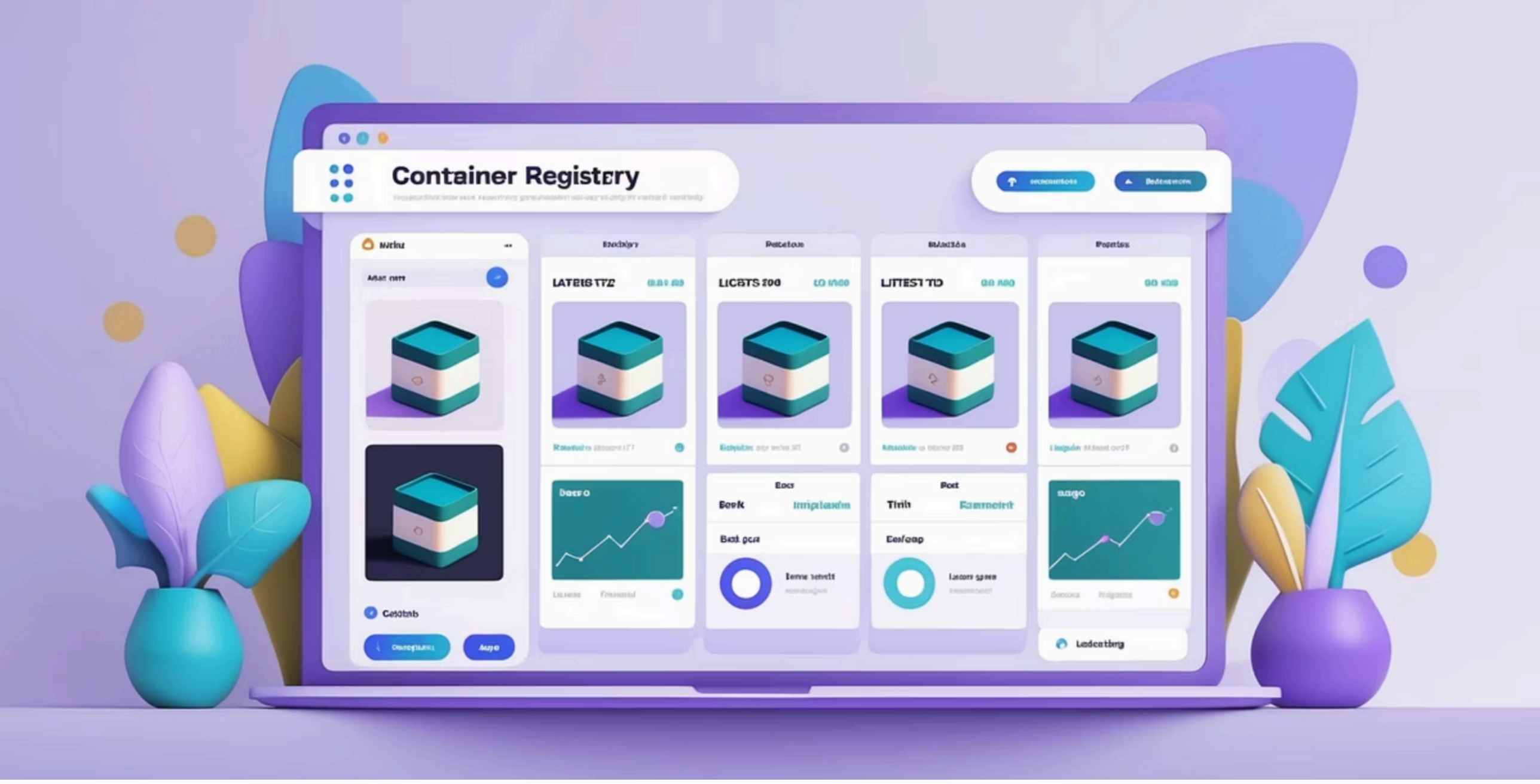
10 Kubernetes Design Errors You're Making (And How to Fix Them!)

Swipe to avoid the most common mistakes that are crashing your cluster!

RANGARAJ BALAKRISHNAN

@RAJCLOUDX



The "Latest" Tag

What's the problem?

Using `:latest` tag causes unpredictable rollbacks and upgrades

Why it's dangerous

Pulls different versions and breaks deployment repeatability

CareerByteCode
Learning Made simple

Fix: Use Specific Version

Pin Exact Tags

Always use specific tags like `myapp:v1.3.0`

Example YAML

image: myapp:v1.3.0 in your Deployment manifest

Best Practice

Update image tag with every code release

CareerByteCode
Learning Made simple

Missing Health

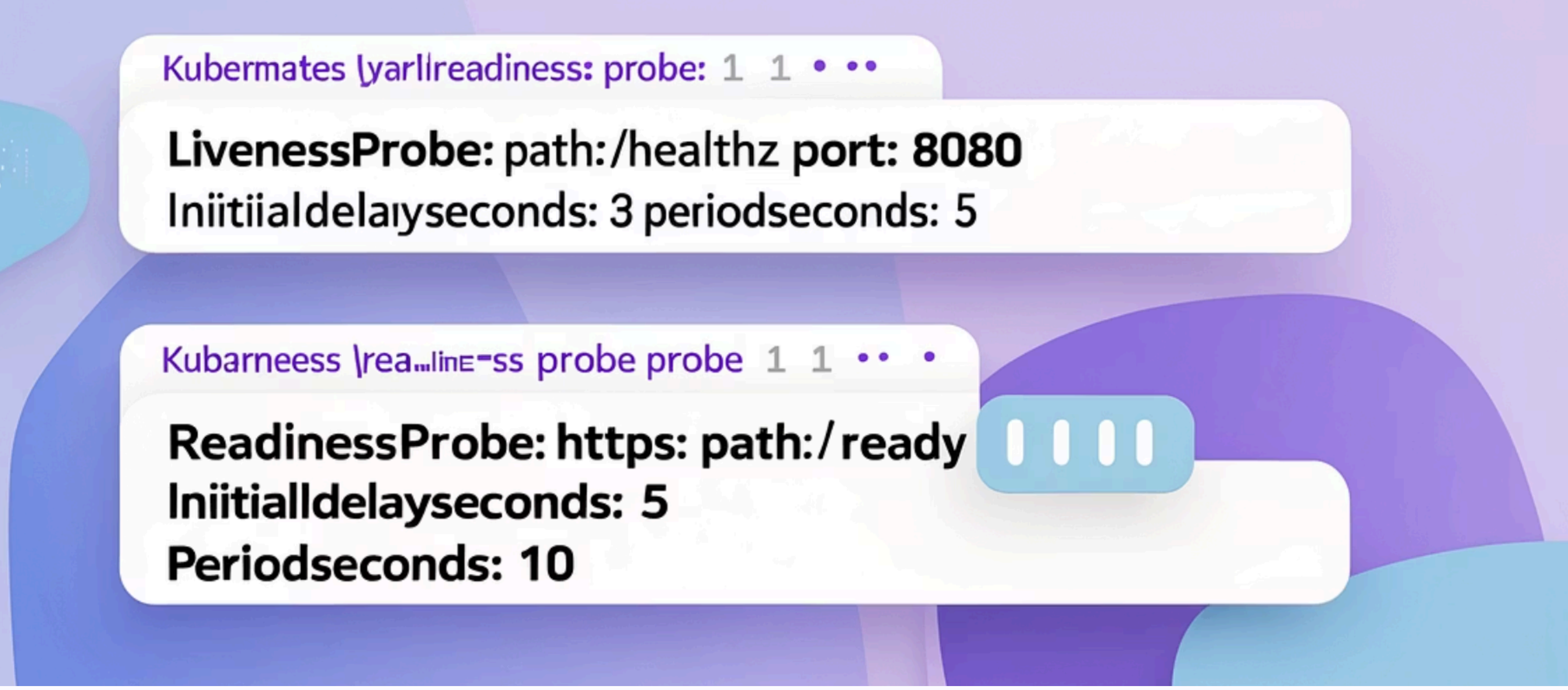
Dead Pods Stay in Service

Without probes, Kubernetes can't detect unhealthy pods

Slow Startups Cause Failures

Initializing pods receive traffic before they're ready

CareerByteCode
Learning Made simple



Fix: Implement Both Types of Probes

Liveness Probe

Detects if your pod is running but deadlocked

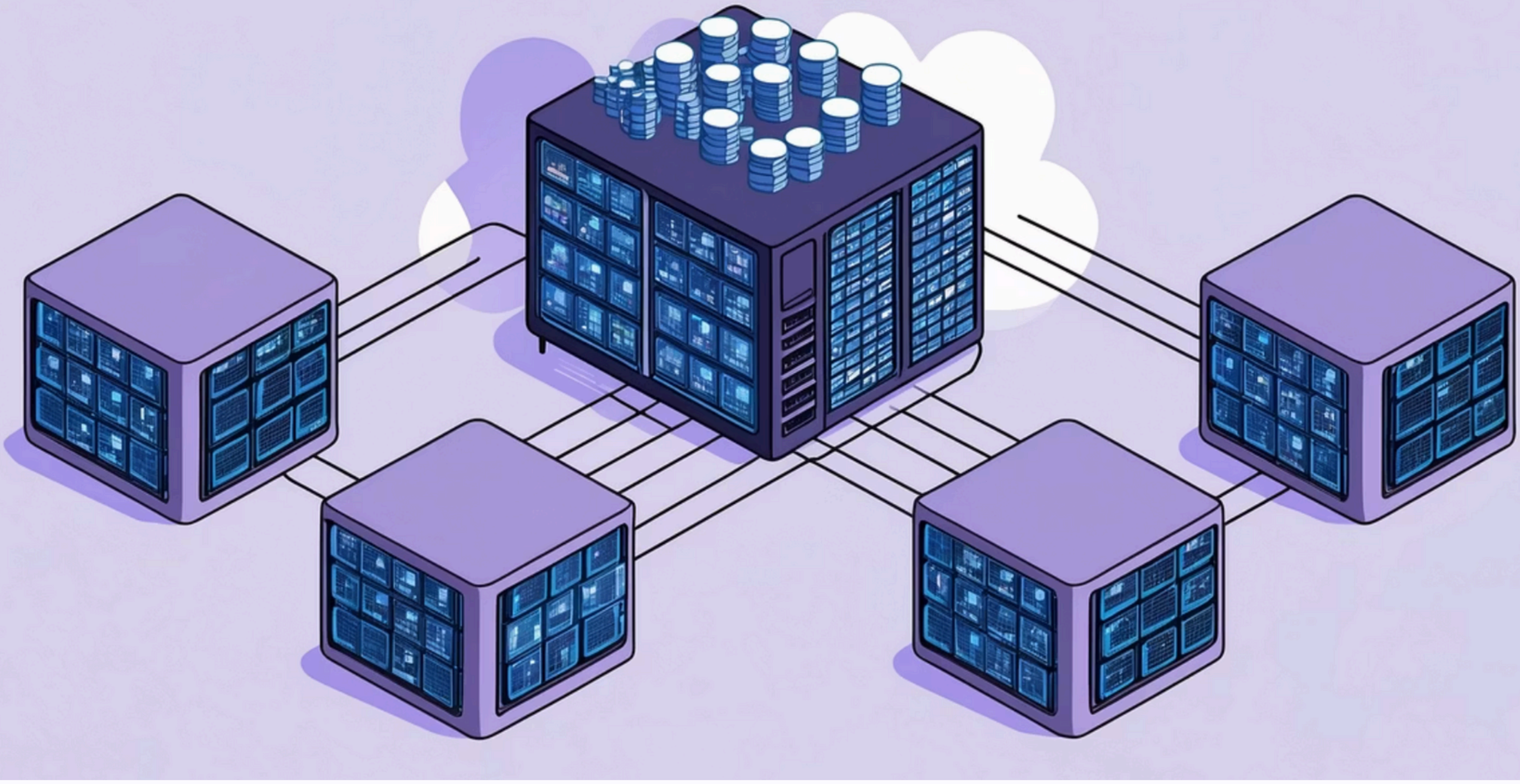
Readiness Probe

Checks if your pod is ready to receive traffic

Use HTTP or Exec

GET /health or command-line checks work best

CareerByteCode
Learning Made simple



Pods Piling Up on One

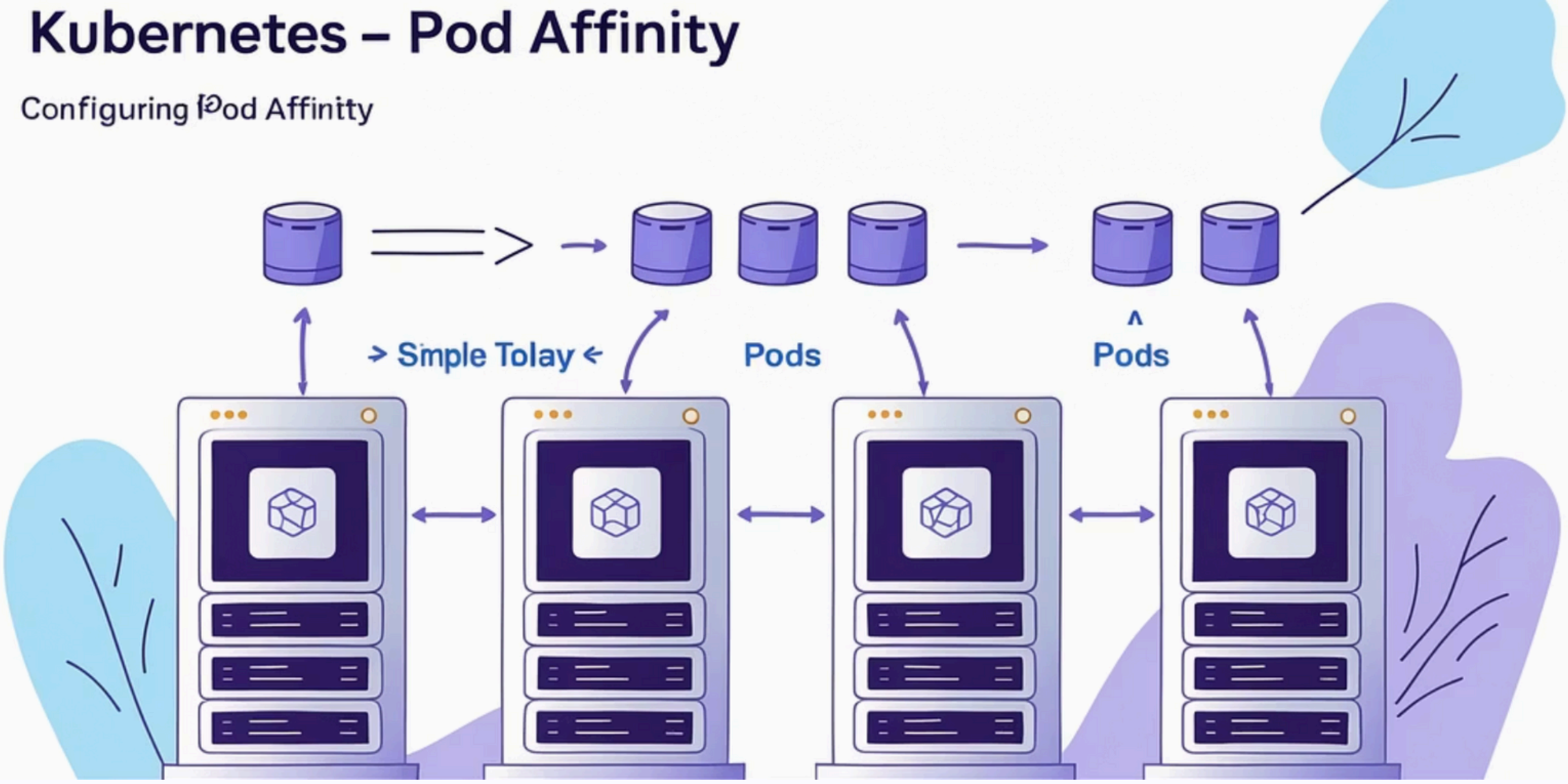
Poor Scheduling Rules

Without affinity settings, pods cluster on single nodes

Availability Risk

If that node fails, multiple services go down simultaneously

CareerByteCode
Learning Made simple



Fix: Set Pod Affinity

Balanced Scheduling

Use podAntiAffinity to spread replicas across nodes

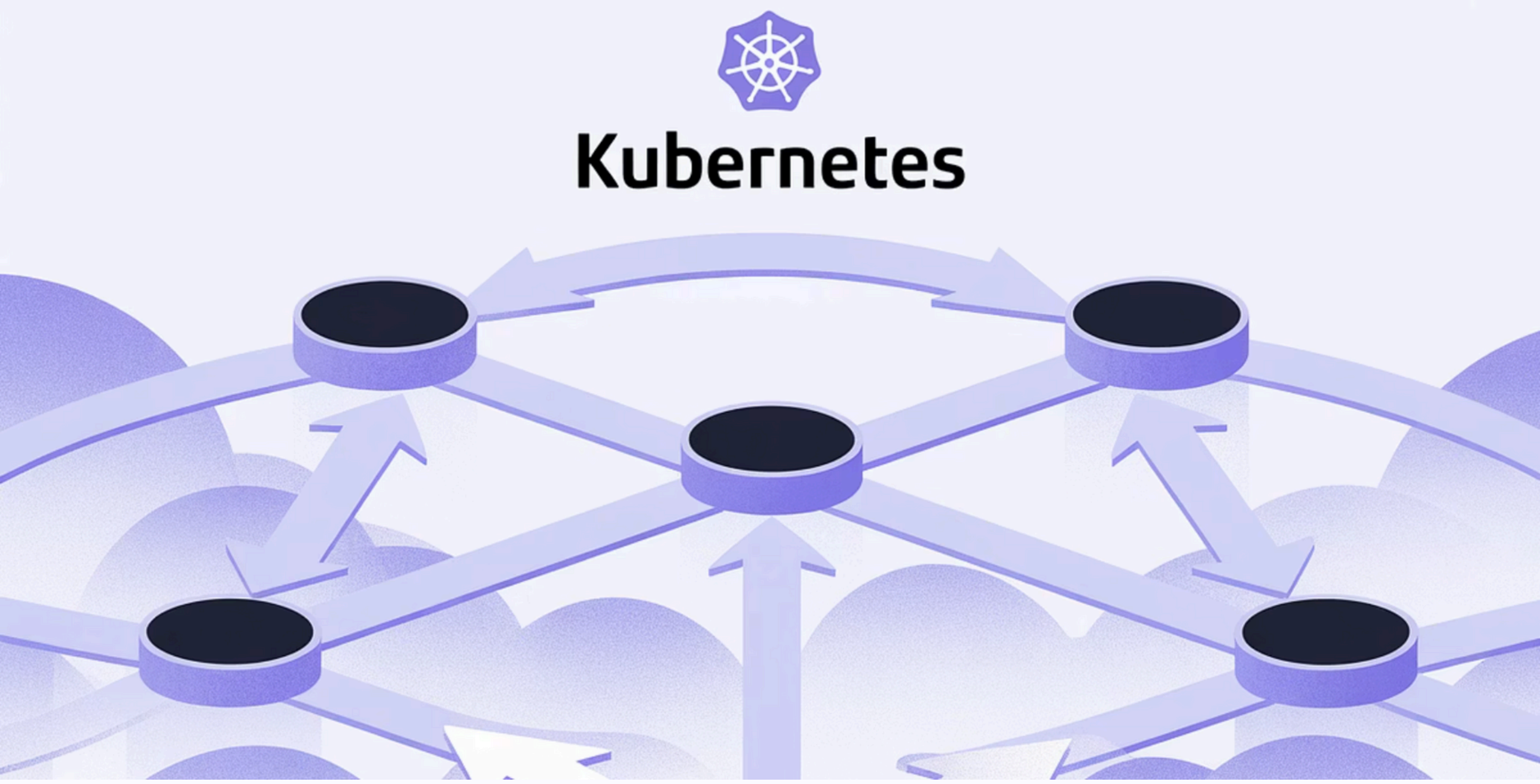
Example YAML

requiredDuringSchedulingIgnoredDuringExecution with matching labels

Result

Pods evenly distributed for better resilience

CareerByteCode
Learning Made simple



No Network Policies = Security Risk

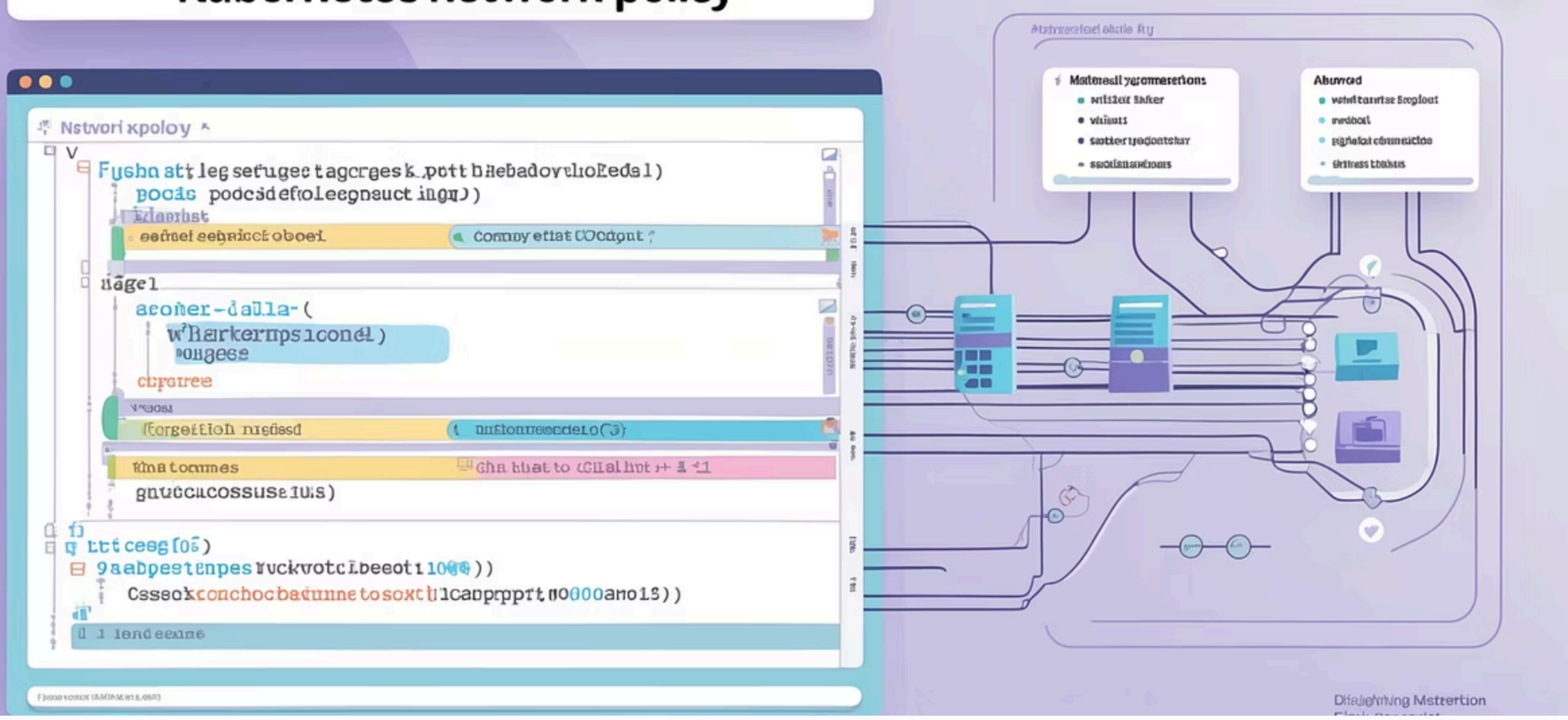
Default: Allow All

Every pod can communicate with every other pod

Zero Segmentation

Compromised pod becomes entry point to entire cluster

CareerByteCode
Learning Made simple



Fix: Apply Network

Restrict

Only allow necessary communication paths

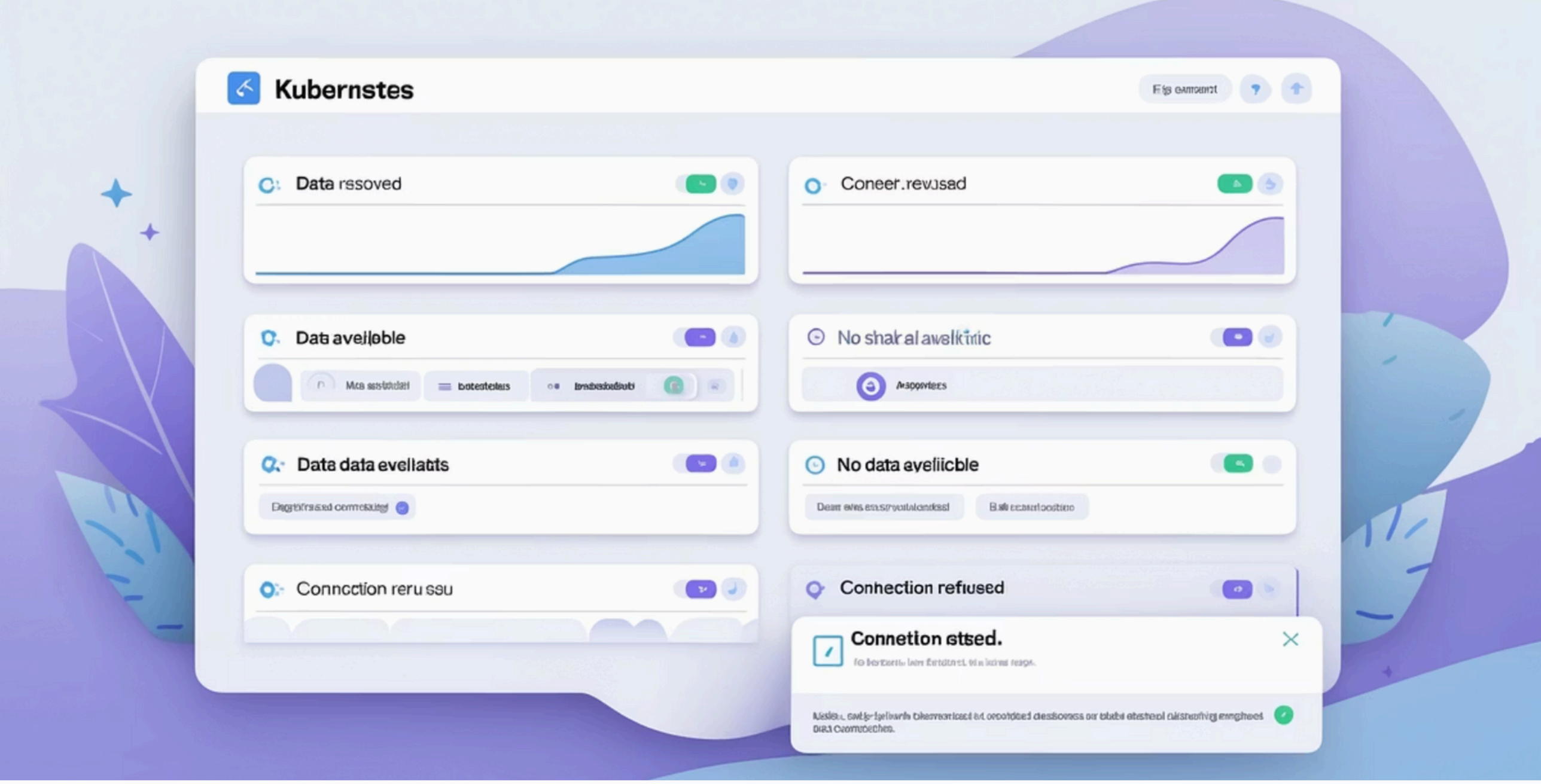
Example

Frontend only accepts web traffic; backend only accepts frontend

Verification

Test with kubectl exec curl commands between pods

CareerByteCode
Learning Made simple



Flying Blind Without Monitoring

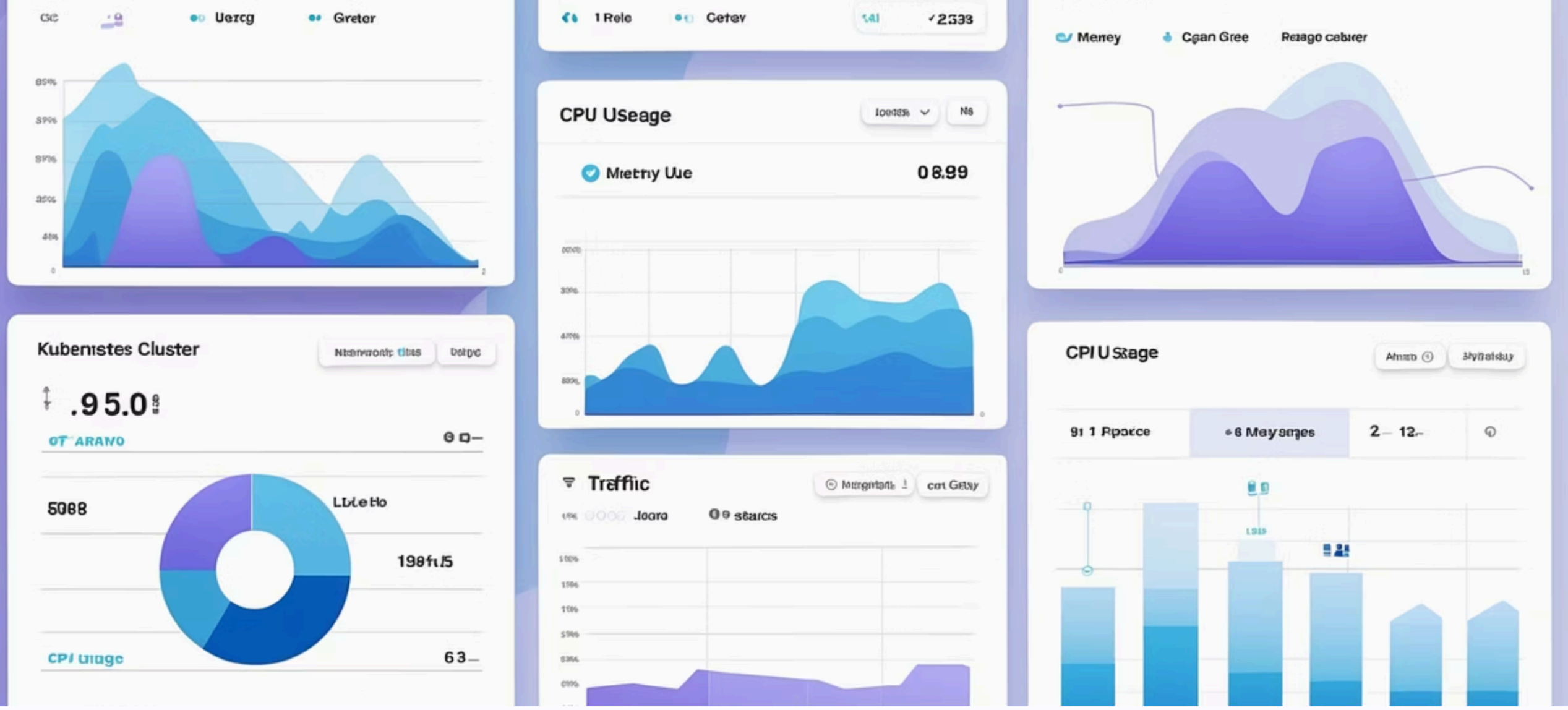
No Performance Metrics

Can't detect resource bottlenecks before they cause outages

Missing Log Aggregation

Debugging requires manual pod-by-pod log inspection

CareerByteCode
Learning Made simple



Fix: Setup Monitoring Stack

Metrics

Implement Prometheus/Grafana for resource visualization

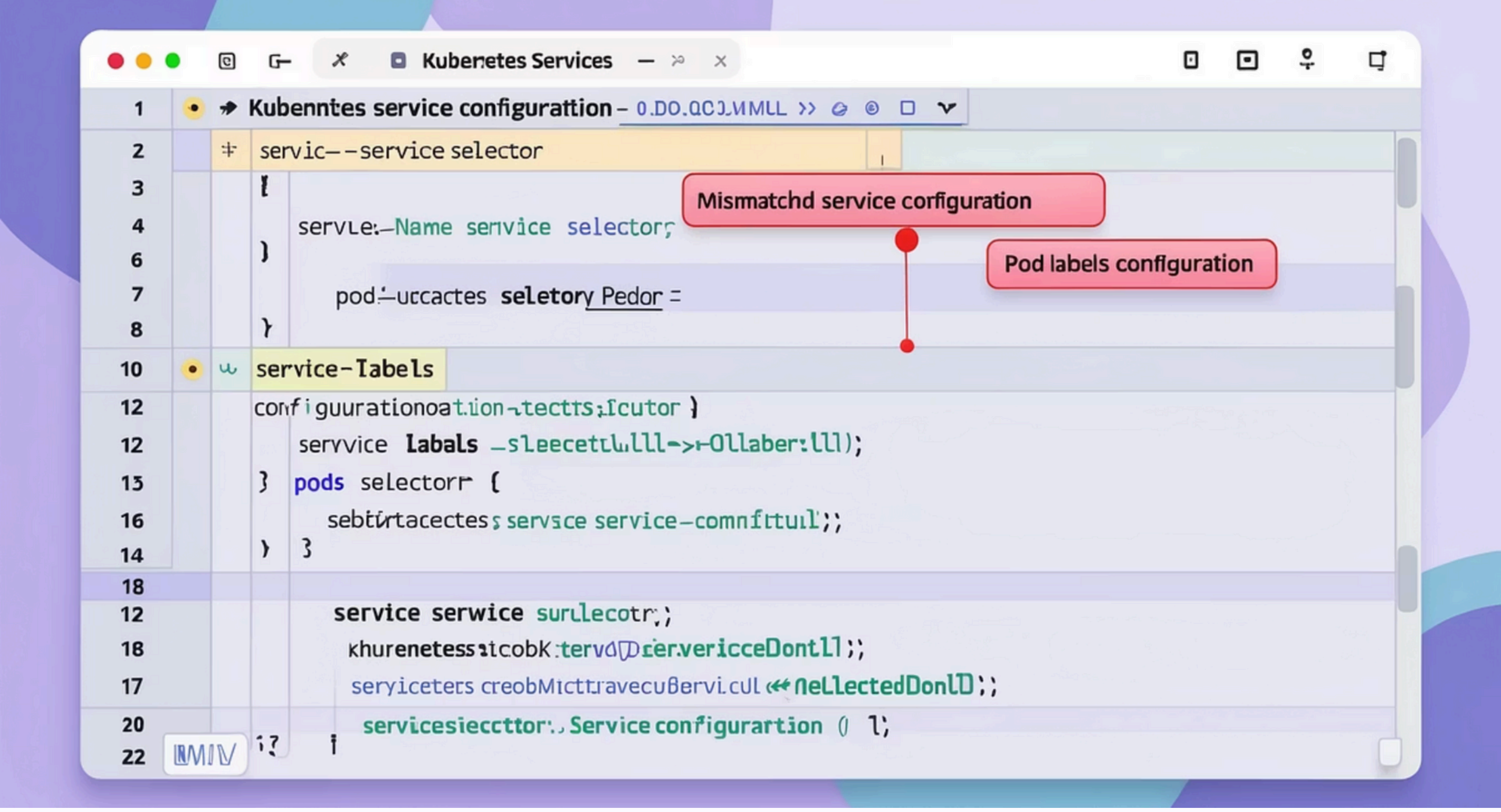
Logging

Deploy ELK or EFK stack for centralized log collection

Configuration

Add prometheus.io/scrape: "true" annotations to pods

CareerByteCode
Learning Made simple



Label Selector Mismatches

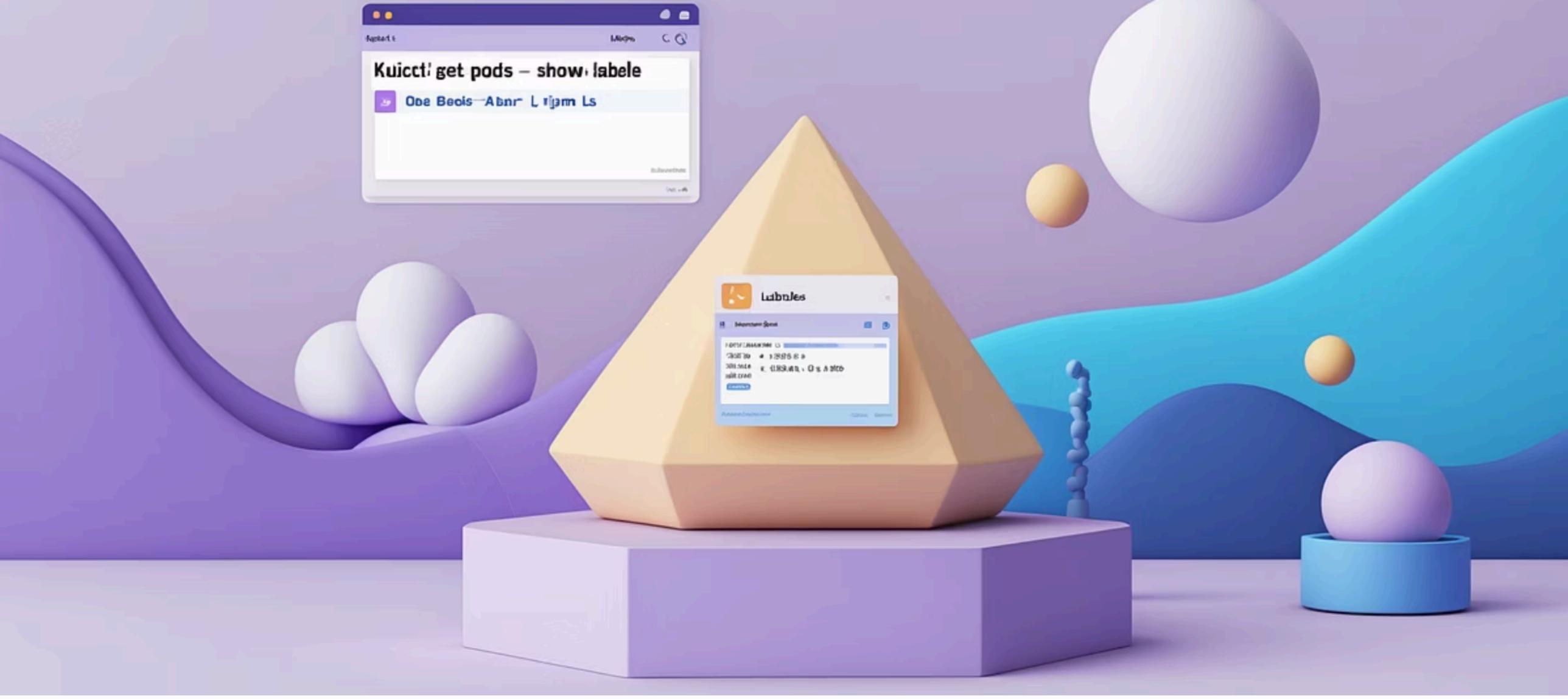
Mislabeled Resources

Services can't find pods due to typos or inconsistent labels

Broken Connections

Traffic routes to nowhere despite running pods

CareerByteCode
Learning Made simple



Fix: Align Labels and Selectors

Match Exactly

Ensure app: myapp on both pod template and service selector

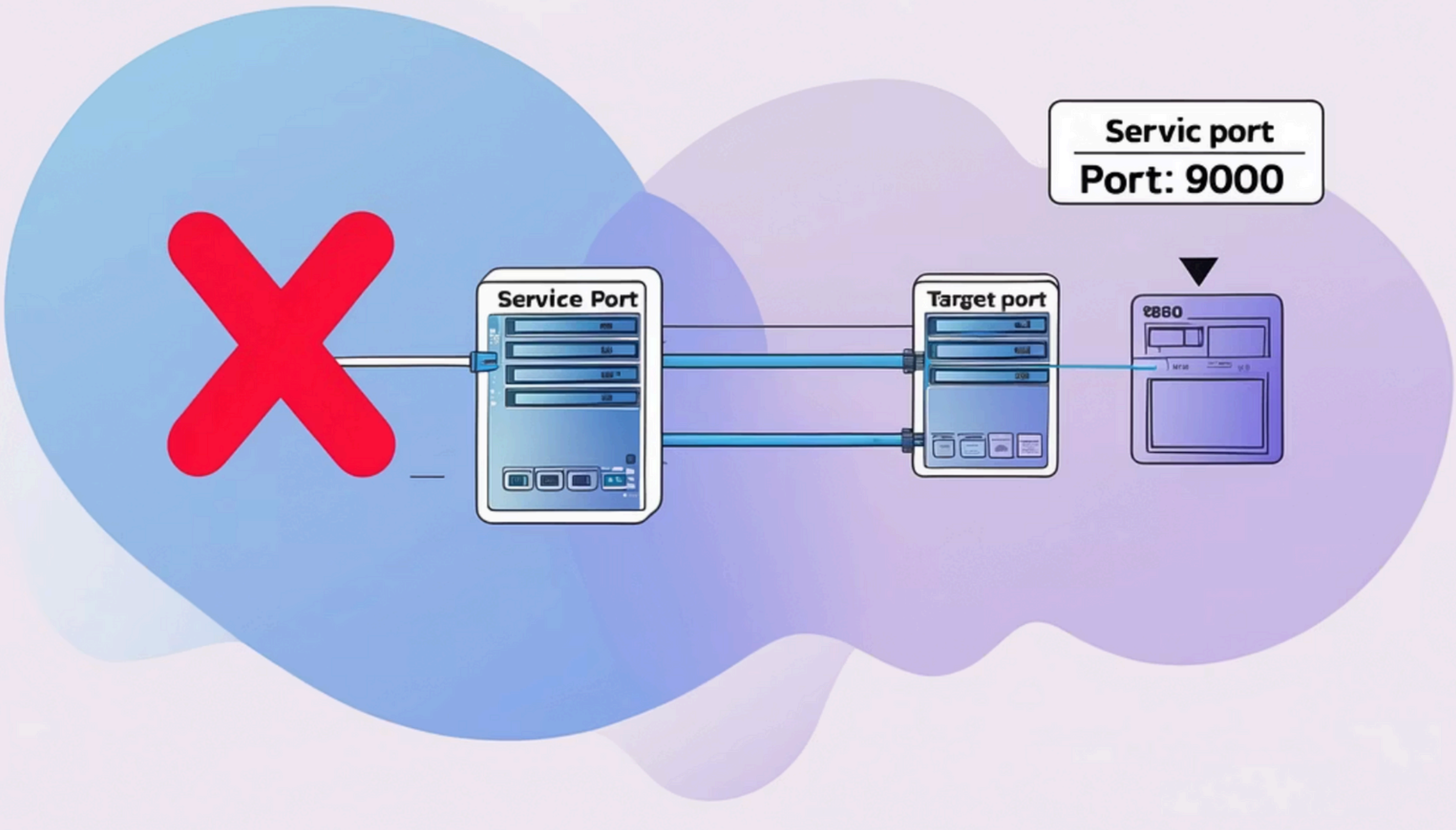
Verification

Use kubectl get pods --show-labels to confirm labels

Best Practice

Use consistent labeling scheme across all resources

CareerByteCode
Learning Made simple



Service Port

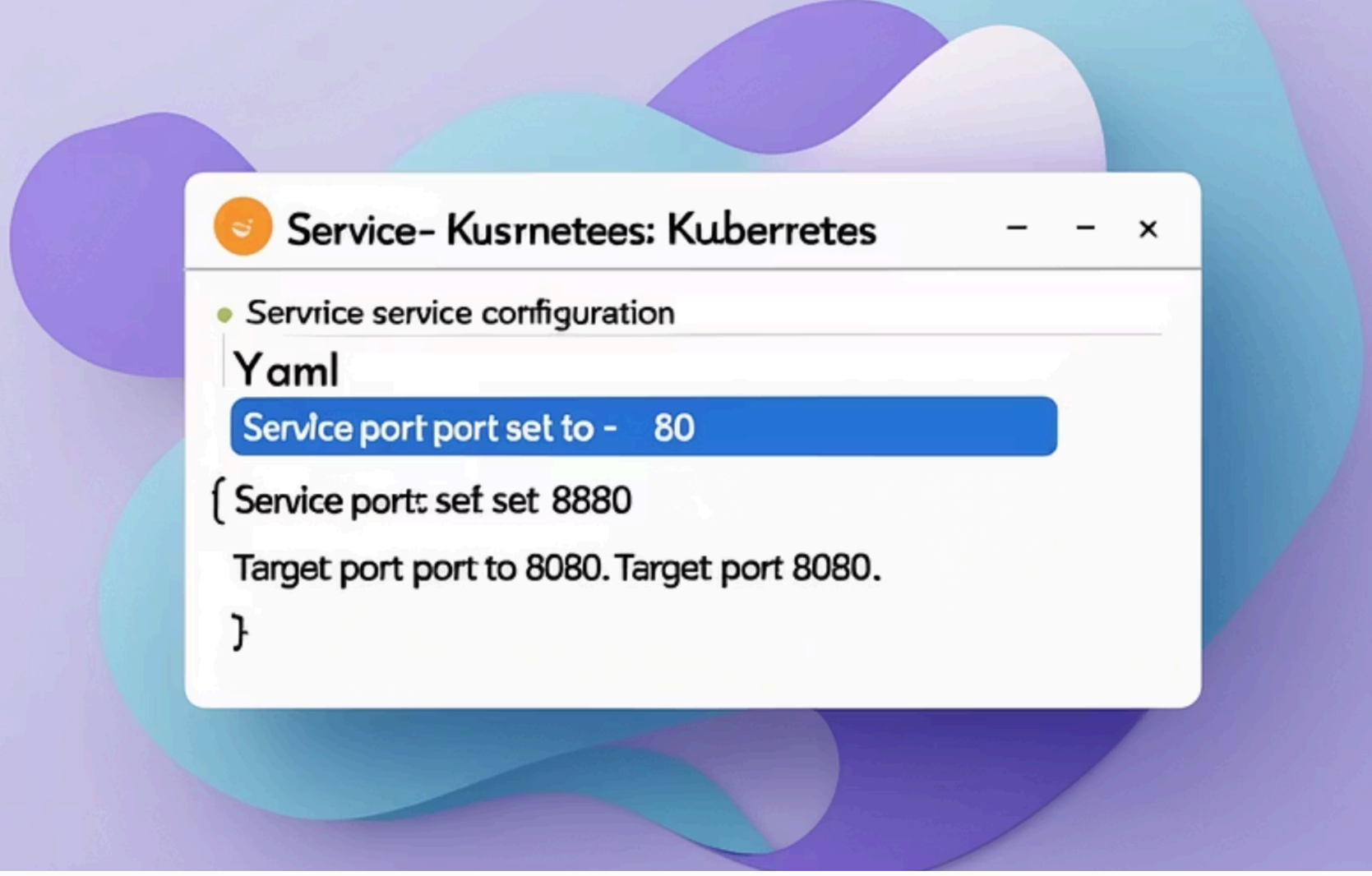
Port Mismapping

Service port doesn't match container's listening port

Connection Failures

Traffic arrives but application doesn't respond

CareerByteCode
Learning Made simple



Fix: Align Service and Container Ports

Port vs TargetPort

Service port is what clients connect to

Example

Service exposes port 80, targets container port 8080

Verification

Test connectivity with kubectl port-forward

CareerByteCode
Learning Made simple

Load Balancer Overuse

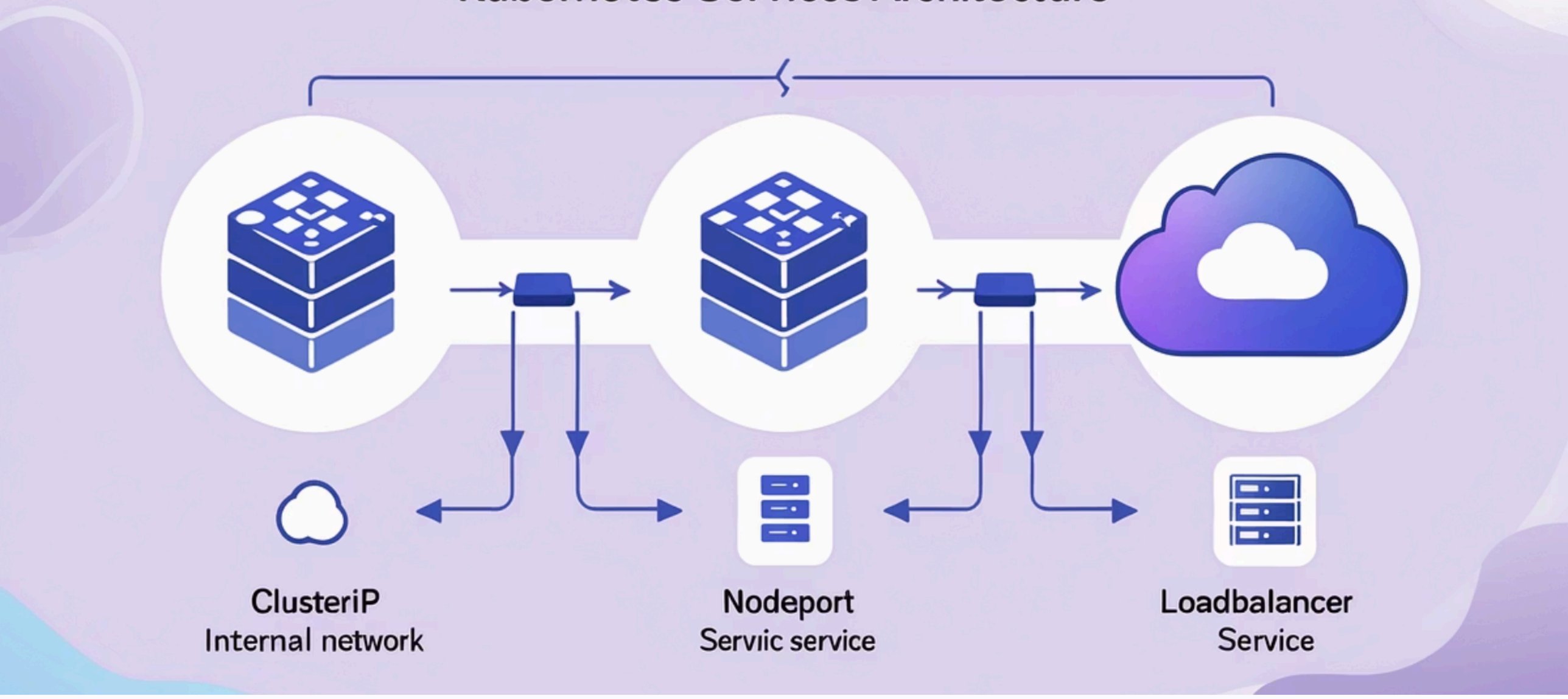
Cost Impact

Each LoadBalancer service incurs cloud provider charges

DNS Confusion

Multiple public endpoints for one app cause routing complexity

CareerByteCode
Learning Made simple



Fix: Use Load Balancers Strategically

Internal: ClusterIP

For pod-to-pod communication within cluster

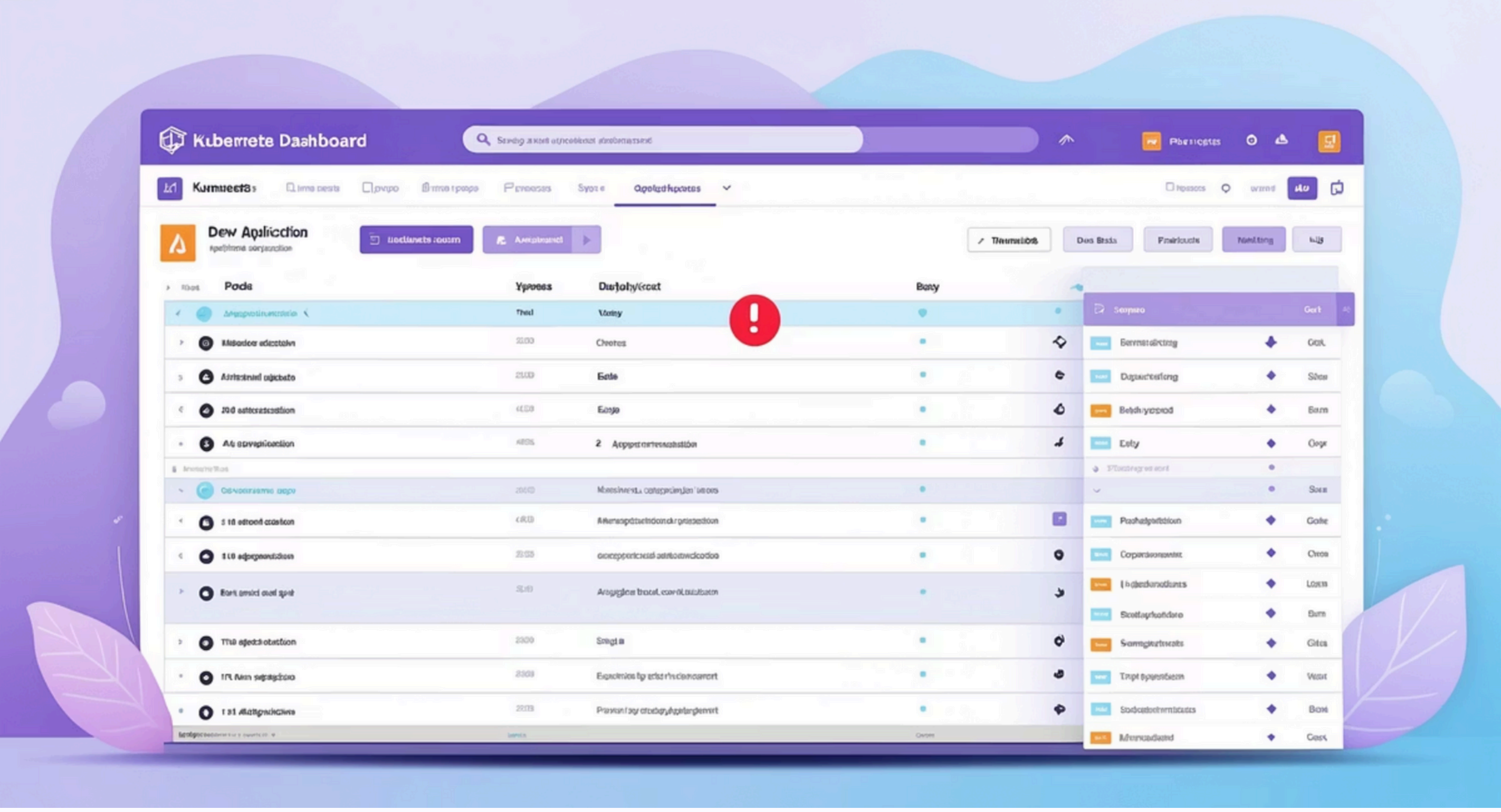
Semi-Public:

For dev/test or behind another LB

Public: LoadBalancer

Only for external-facing endpoints

CareerByteCode
Learning Made simple



Wrong Namespace Deployments

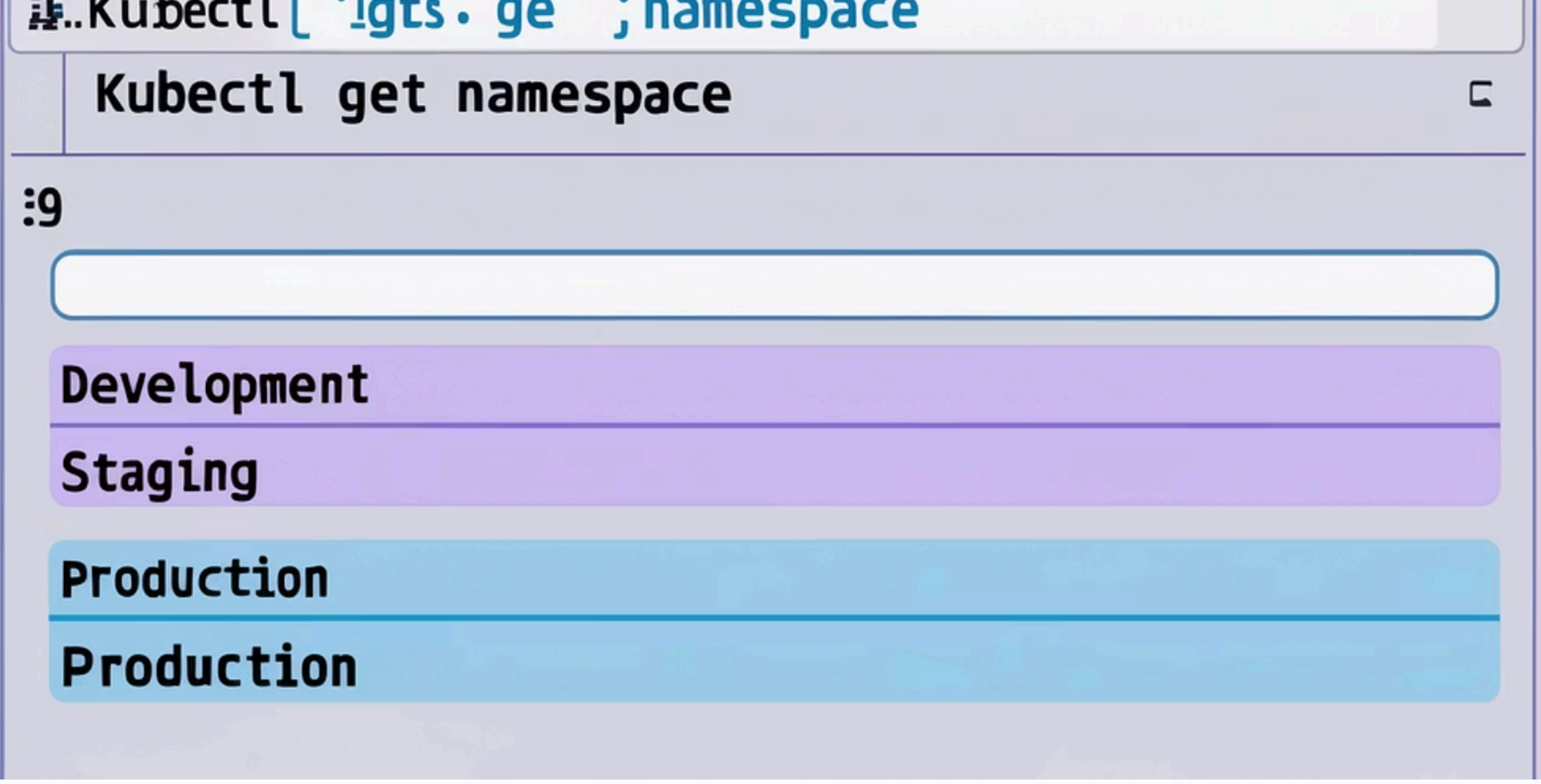
Environment Mixups

Test apps deployed to production or vice versa

Secret Exposure

Apps access secrets from wrong environment

CareerByteCode
Learning Made simple



Fix: Explicit Namespace Management

In YAML

Always include namespace: prod in resource definitions

Command Line

Use kubectl config set-context to set default namespace

Verification

Double-check with kubectl get pods -n staging

CareerByteCode
Learning Made simple

Unlimited Resource Consumption

Resource Hogging

One pod can consume all CPU/memory on a node

Eviction Risk

Pods killed when nodes run out of resources

CareerByteCode
Learning Made simple

Fix: Define Resource Boundaries

Requests

Minimum resources pod needs: CPU: 100m, memory: 256Mi

Limits

Maximum resources pod can use: CPU: 500m, memory: 1Gi

Benefits

Better scheduling, predictable performance, reliable autoscaling

CareerByteCode
Learning Made simple

Maintenance Downtime Risk

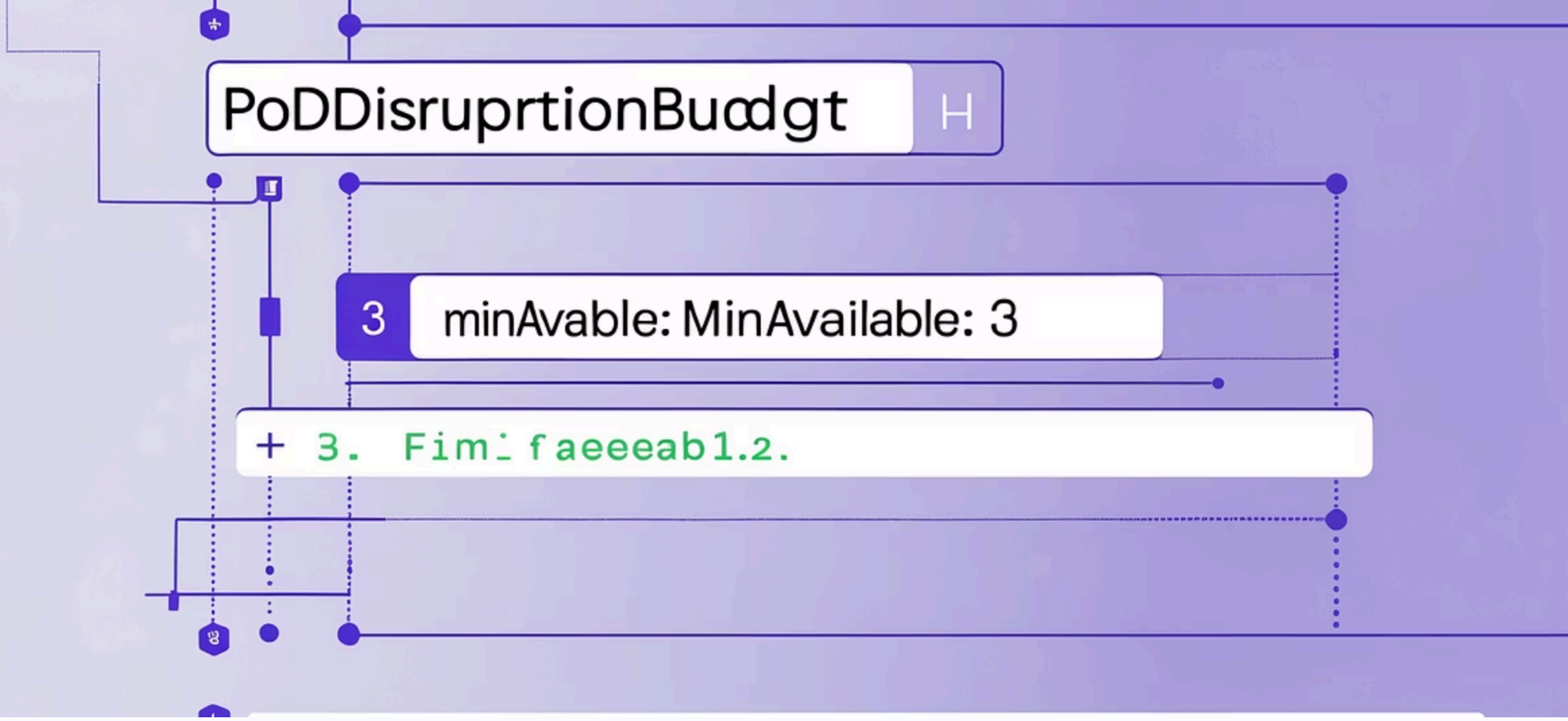
Node Maintenance

Cluster upgrades can kill all replicas simultaneously

No Safety Net

Without PDB, kubectl drain can cause outages

CareerByteCode
Learning Made simple



Fix: Add PodDisruptionBudget

MinAvailable

Set to keep at least 2 replicas running during maintenance

MaxUnavailable

Alternative: specify maximum pods that can be down

Implementation

Create PDB YAML that matches your deployment selector

CareerByteCode
Learning Made simple

Image Pull Errors

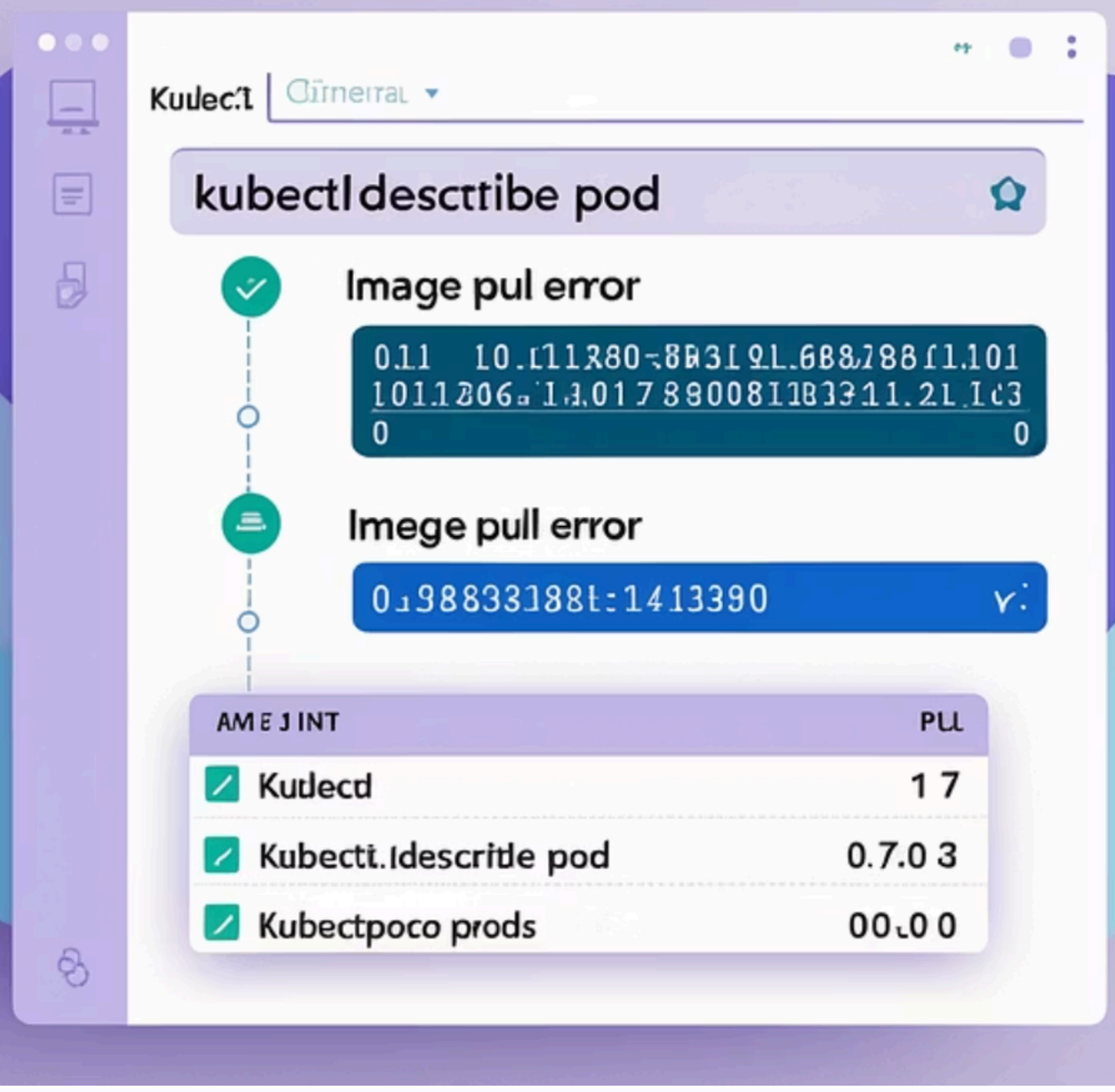
ErrImagePull / ImagePullBackOff

Pod can't download container image

Common Causes

Typos in image name, missing tags, or registry auth issues

CareerByteCode
Learning Made simple



Fix: Resolve Image

Verify Image

Double-check spelling and tag existence in registry

Check Credentials

Ensure imagePullSecrets are correct

Investigate

Run kubectl describe pod for detailed error message

CareerByteCode
Learning Made simple

CreateContainerConfigError

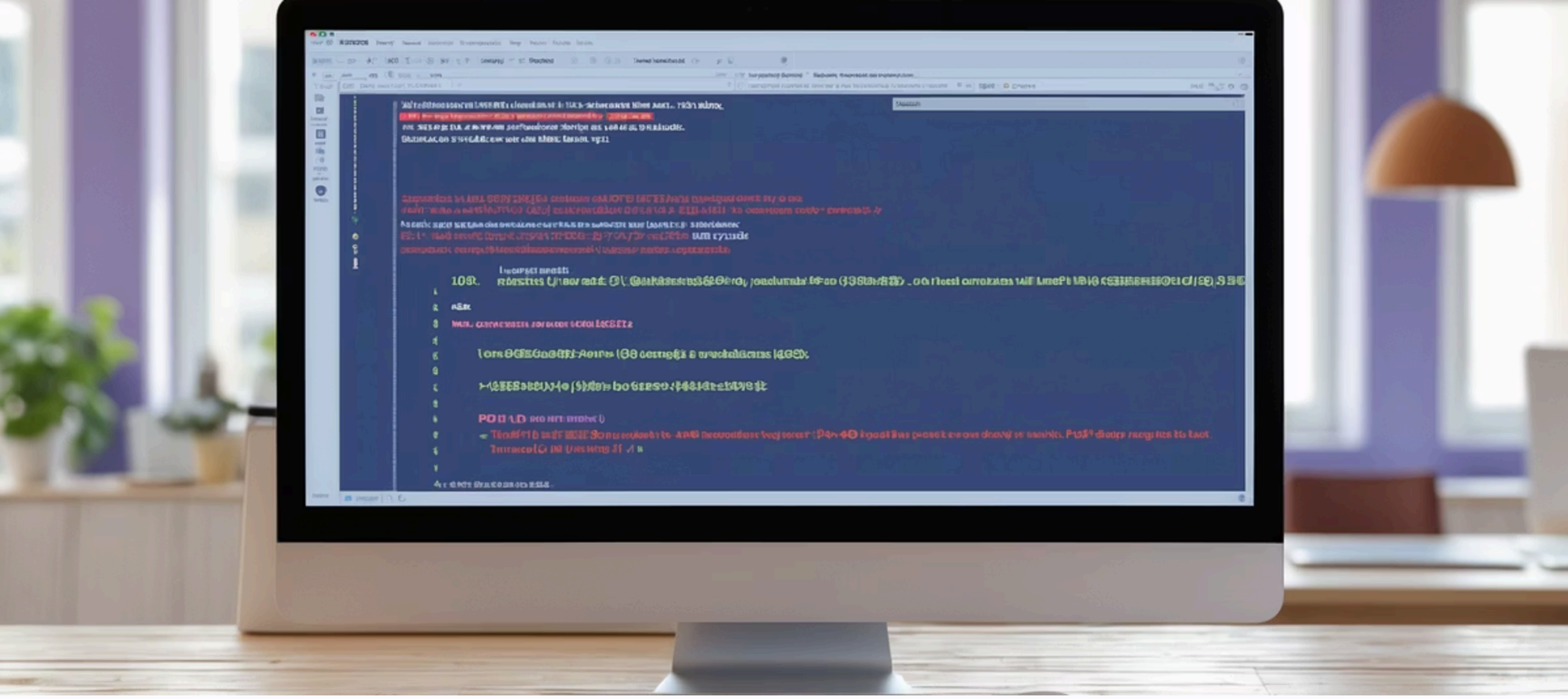
Config Problems

Pod fails to start due to specification errors

Common Causes

Missing environment variables, invalid mounts, or secret references

CareerByteCode
Learning Made simple



Fix: Correct Pod Configuration

API Validation

Use kubectl --validate to check manifest against schema

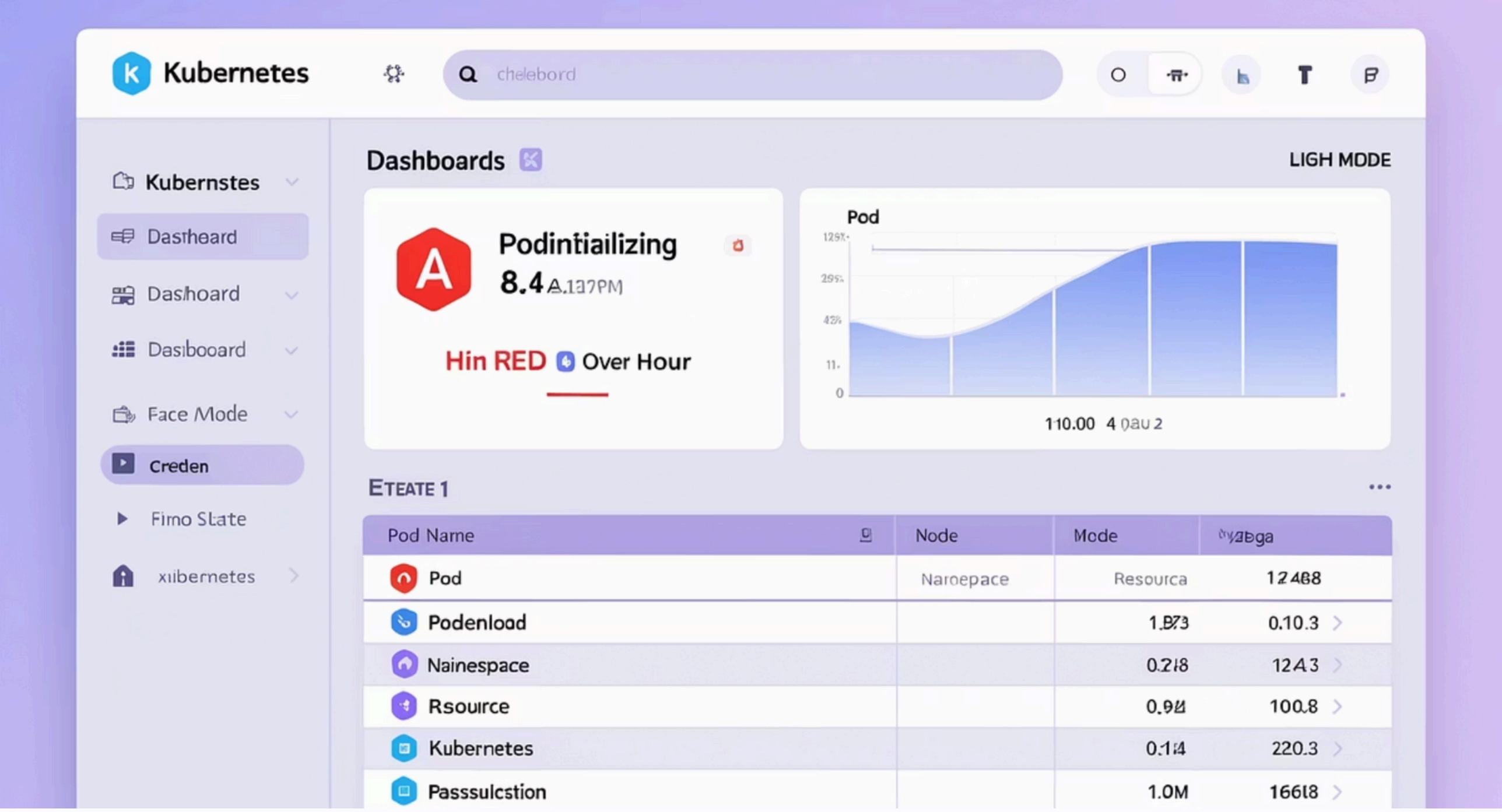
Inspect Errors

kubectl describe pod shows detailed configuration issues

Test Resources

Verify referenced ConfigMaps and Secrets exist

CareerByteCode
Learning Made simple



Stuck in

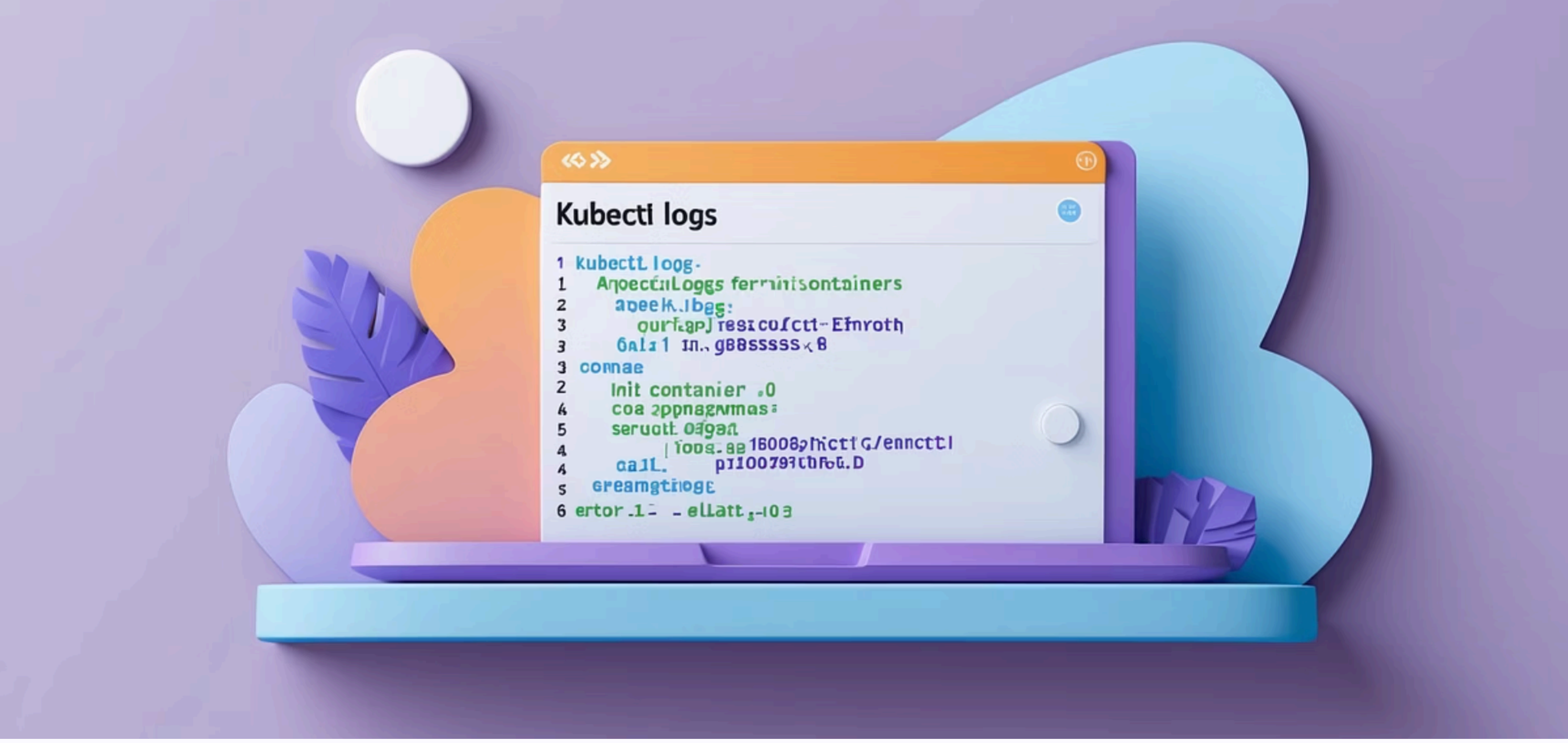
Initialization Freeze

Pod never progresses past init phase

Common Causes

Failed init containers, missing volumes, or resource constraints

CareerByteCode
Learning Made simple



Fix: Investigate Init Failures

Check Init Logs

kubectl logs pod-name -c init-container-name

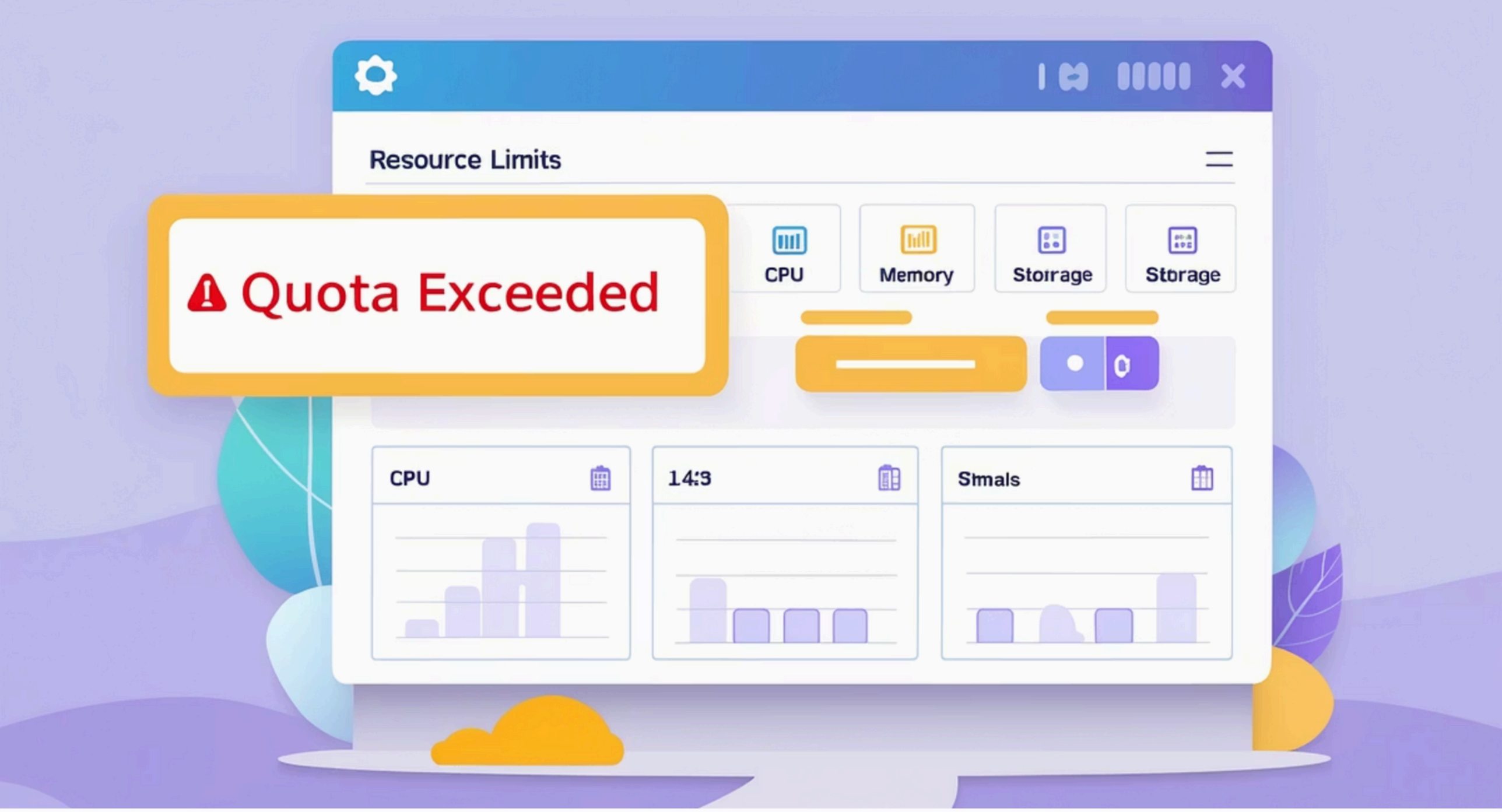
Verify Volumes

Ensure PVCs are bound and readable

Review Resources

Check if init containers have sufficient CPU/memory

CareerByteCode
Learning Made simple



Resource Quota

Hitting Namespace Limits

Prevented from creating new resources

Common Quota

Pod count, CPU, memory, storage, or custom resources

CareerByteCode
Learning Made simple

Fix: Manage Resource Quotas

Check Current

Run kubectl get resourcequota -n namespace

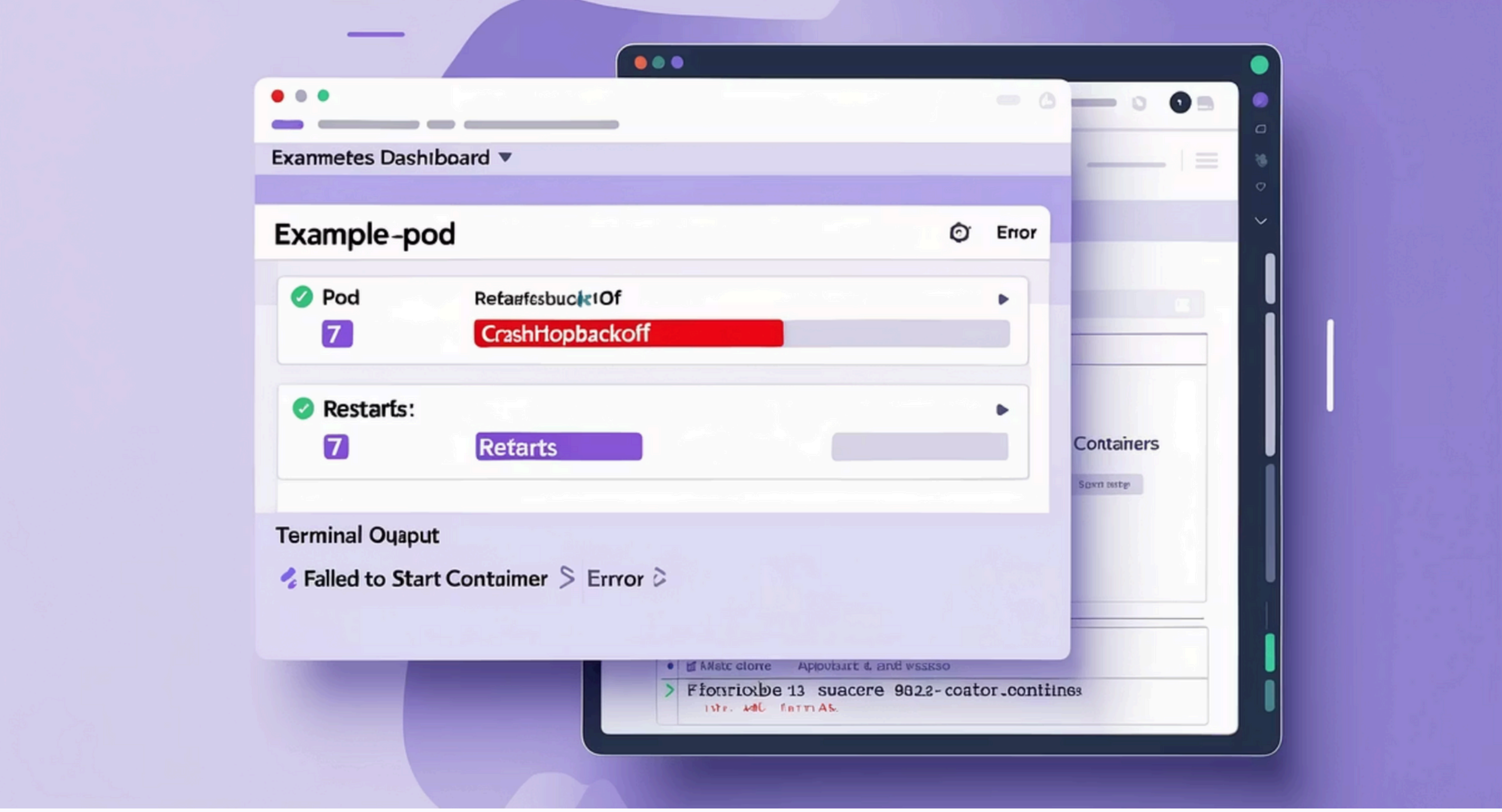
Adjust Usage

Reduce replica count or size resource requests

Request Increase

Ask cluster admin to raise quota limits if needed

CareerByteCode
Learning Made simple



CrashLoopBackOff

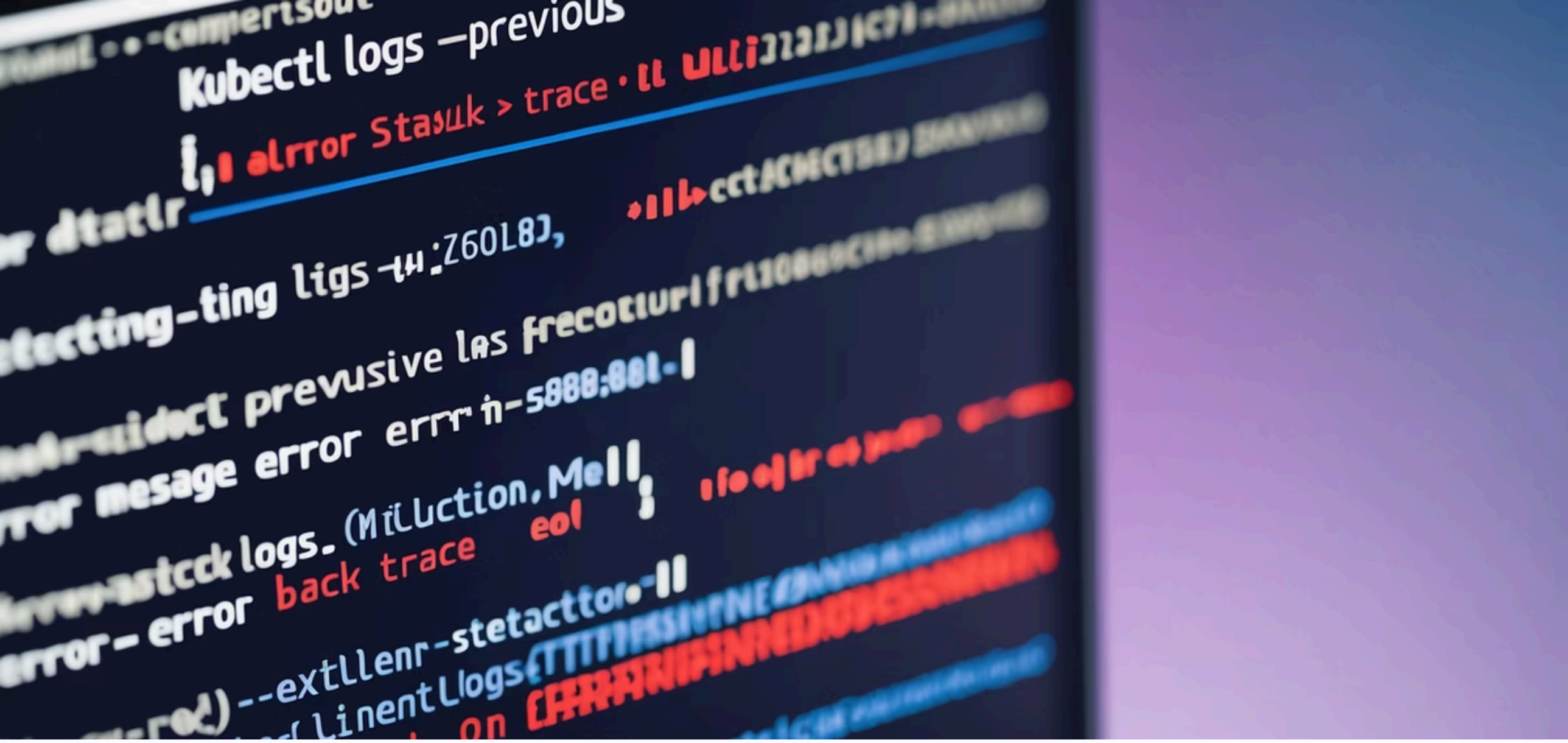
Endless Restarts

Container keeps failing and Kubernetes keeps restarting it

Common Causes

Missing files, bad commands, or configuration issues

CareerByteCode
Learning Made simple



Fix: Break the Crash Loop

Get Crash Logs

kubectl logs pod-name --previous to see failure reason

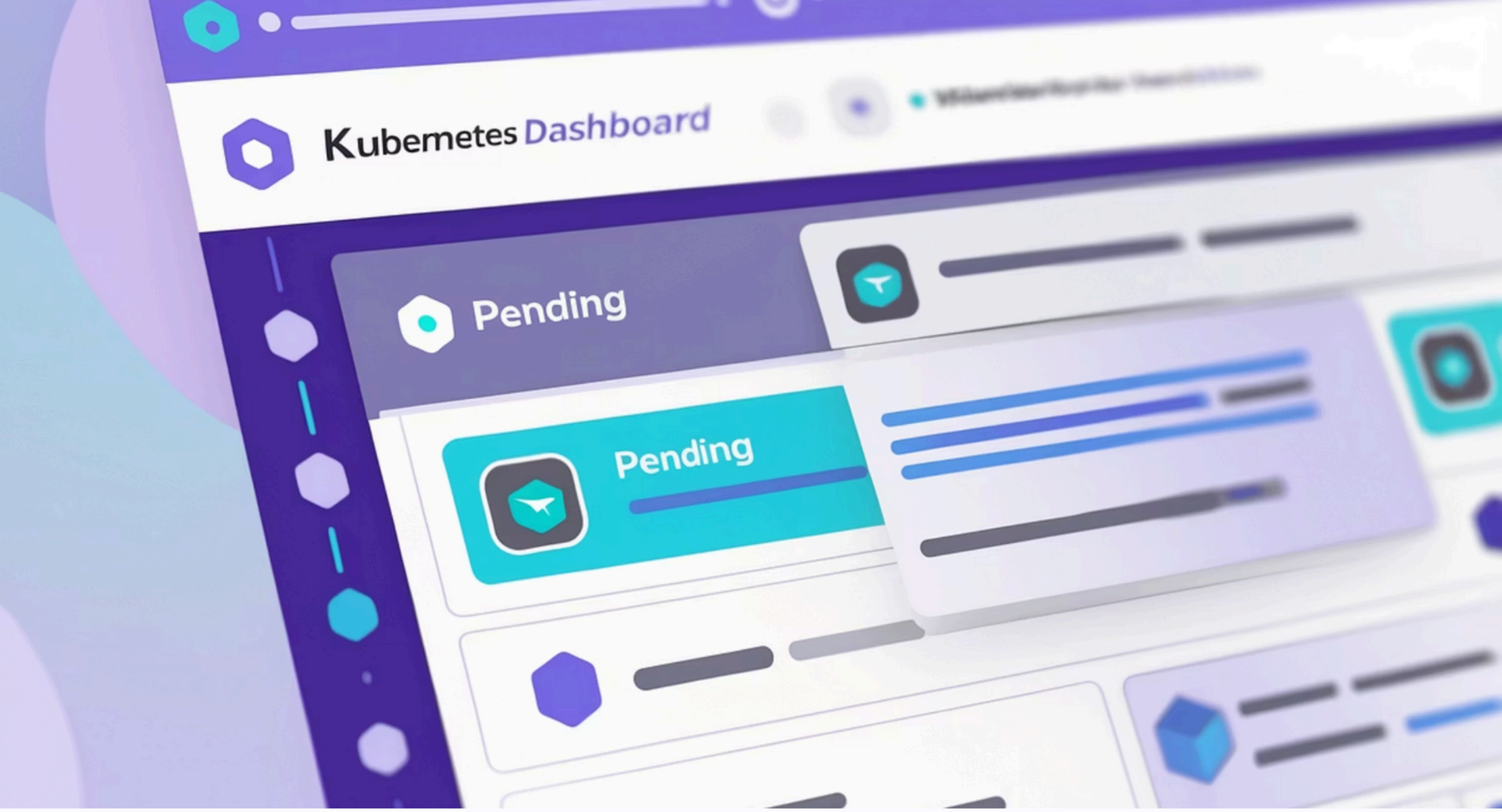
Debug Command

Check command and args in pod spec

Update Config

Fix the root cause and redeploy with corrected YAML

CareerByteCode
Learning Made simple



Scheduling

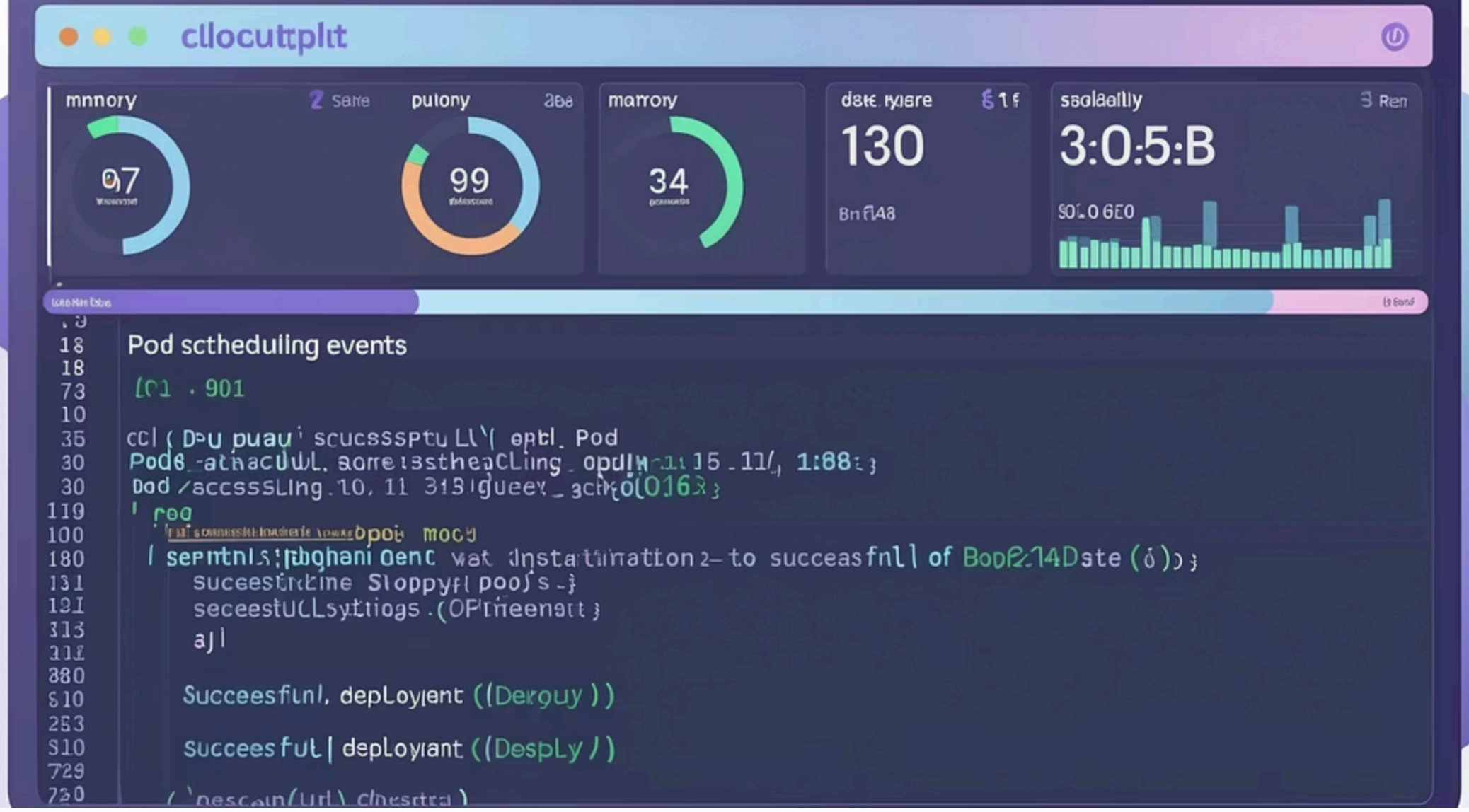
Pods Stuck in Pending

Never get assigned to a node

Common Causes

Insufficient resources, node selectors, or taints/tolerations

CareerByteCode
Learning Made simple



Fix: Address Scheduling Issues

Check Node Status

kubectl get nodes -o wide to see available resources

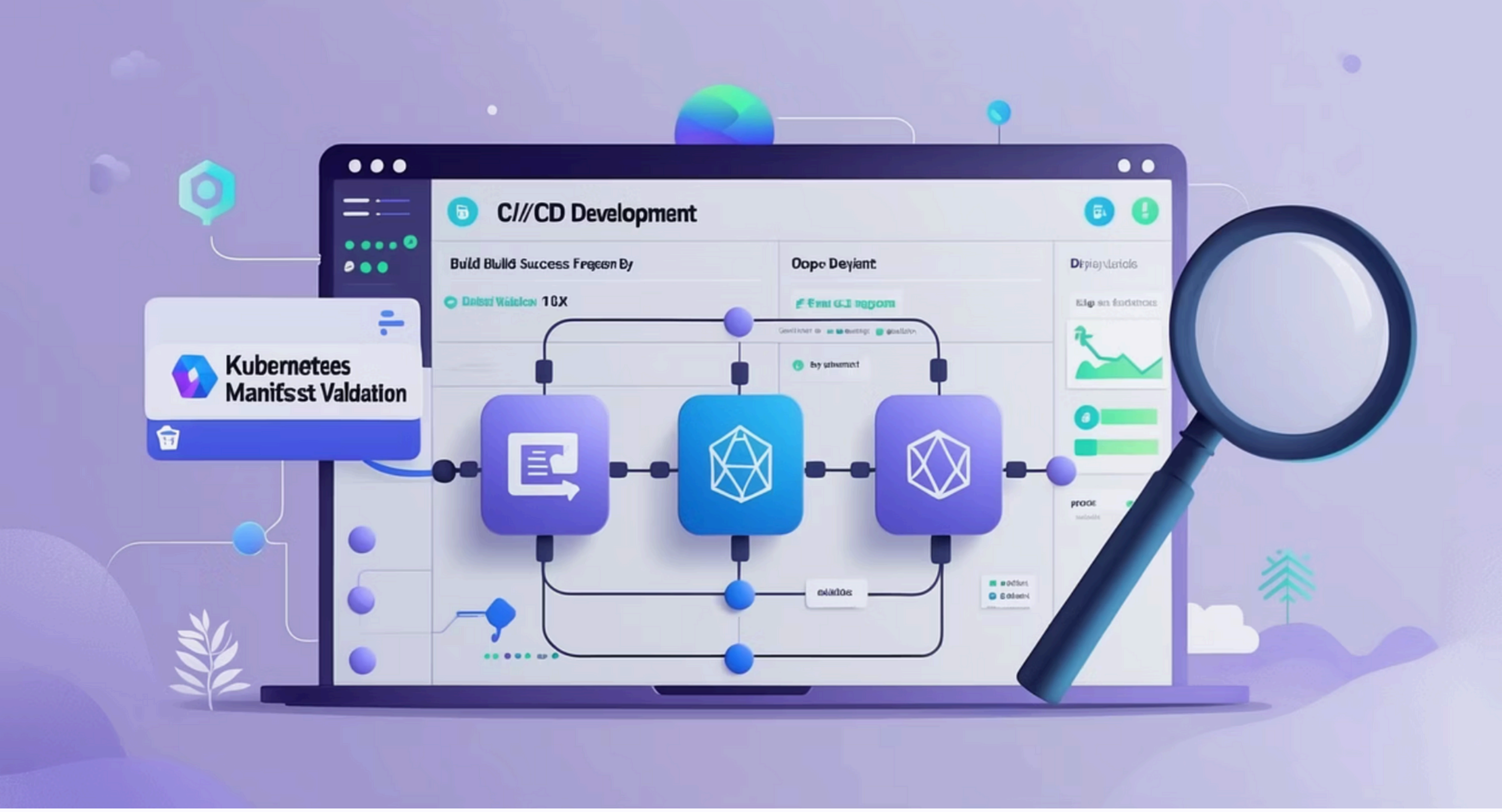
Review Pod Spec

Examine nodeSelector, affinity rules, and tolerations

Investigate Events

kubectl describe pod shows scheduler decisions

CareerByteCode
Learning Made simple



Prevention: Automate YAML Validation

Pre-deployment Checks

Catch configuration issues before they hit the cluster

Tools

Use kubeval, datree, or conftest in CI/CD pipelines

CareerByteCode
Learning Made simple

Prevention: Secure Default Configurations

RBAC

Apply least-privilege role-based access control to all resources

Pod Security

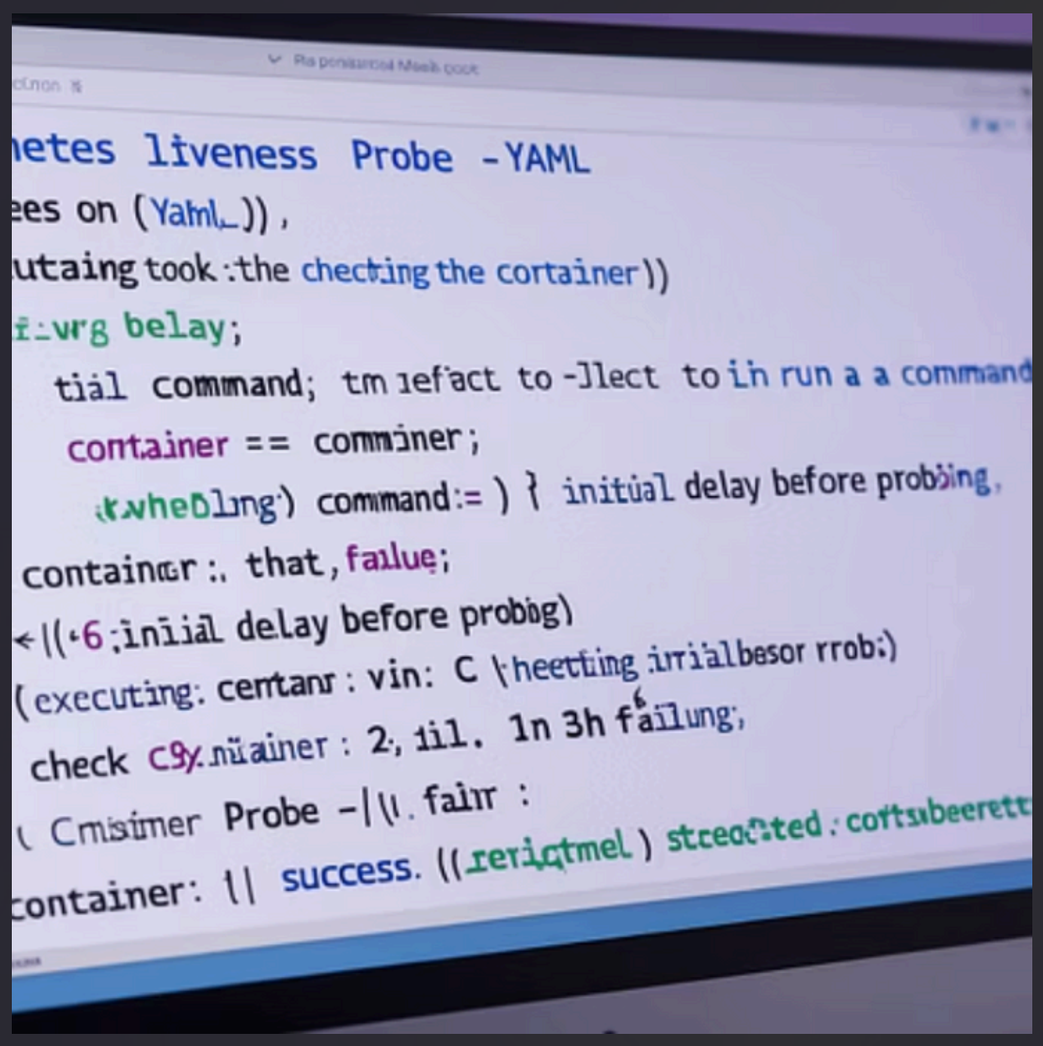
Disable privileged containers and root access by default

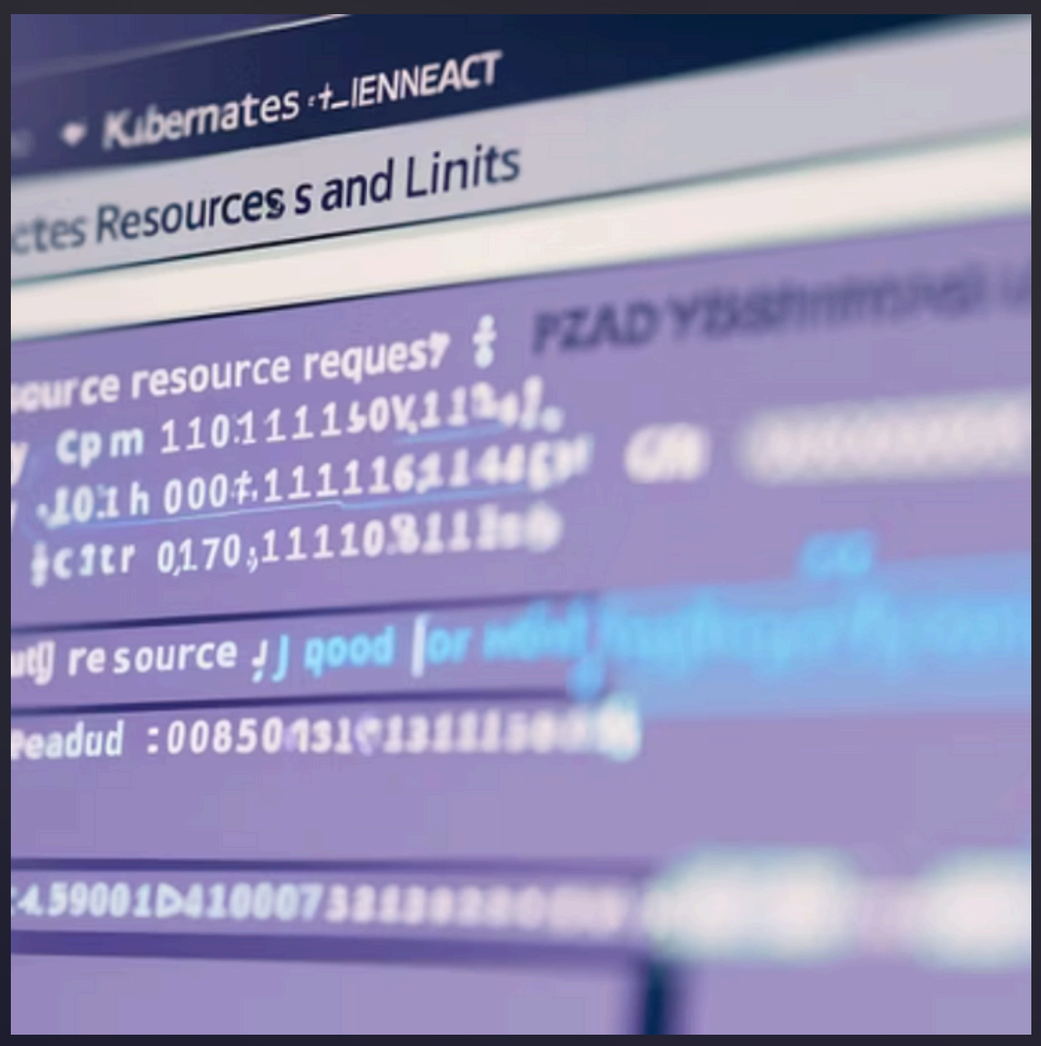
Network Policy

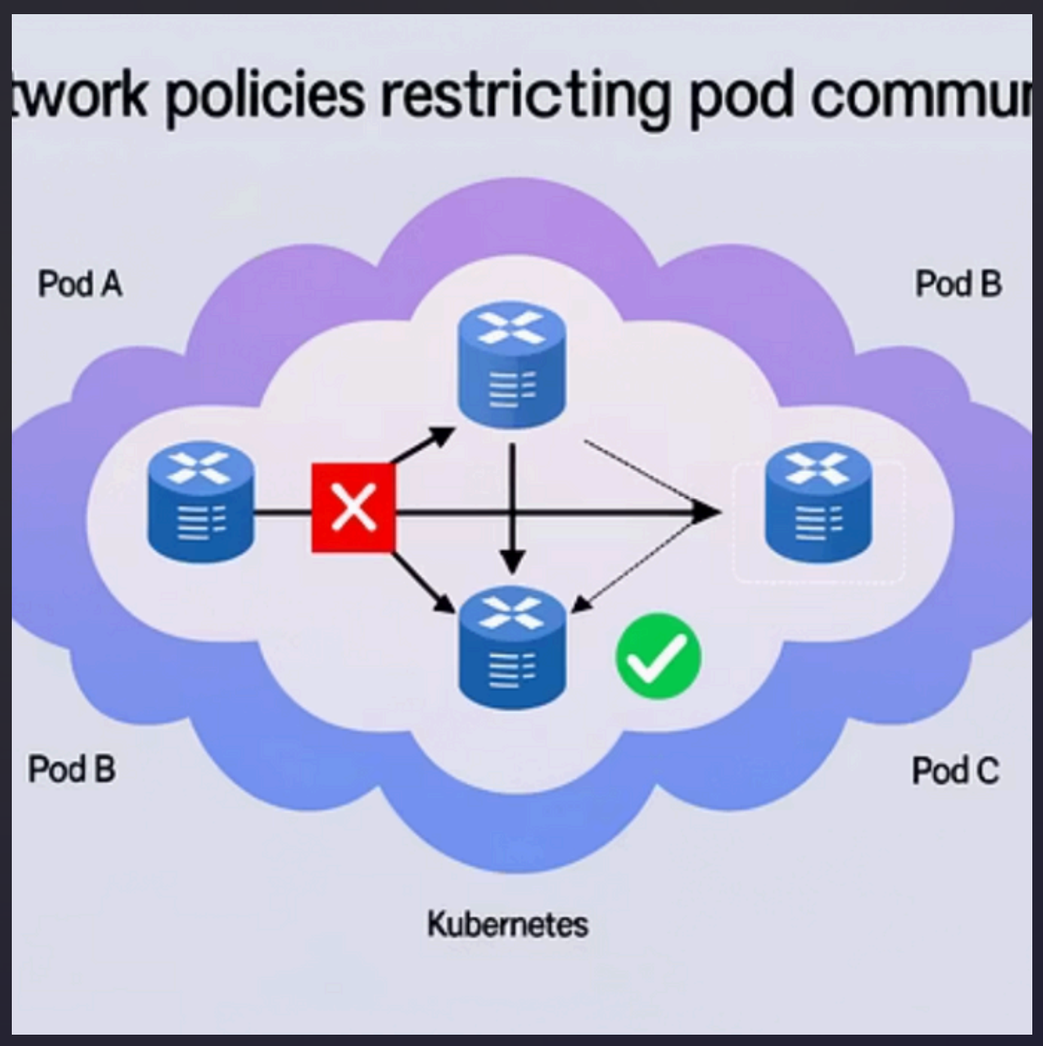
Start with default-deny and add specific allow rules

CareerByteCode
Learning Made simple

Quick Recap: Top Errors to







Always pin image versions, implement health checks, set resource boundaries, and restrict network access for a production-ready cluster.

CareerByteCode
Learning Made simple

Essential Troubleshooting

kubectl describe

Get detailed info and events for any resource

kubectl logs

View container output with --previous for crashed pods

kubectl get -o wide

See additional details including node assignments

kubectl exec -it

Run commands inside containers for live debugging

CareerByteCode
Learning Made simple

Build Reliable Kubernetes Clusters Today!

These fixes will help you avoid the most common Kubernetes design errors. Tag a fellow engineer who needs this guide, and share your own K8s troubleshooting tips in the comments!

CareerByteCode
Learning Made simple

FOLLOW

RANGARAJ BALAKRISHNAN
CLOUD ARCHITECT

@RAJCLOUDX

LinkedIn

Share Your Thoughts in the comment Section

Save the post,in case you want to see it again

careerbytecode.substack.com

+32 - 471408908