KUBERNETES

RESOURCE MANAGEMENT

Christian Dussol

THE RESOURCE MANAGEMENT CHALLENGE

Without governance

- ✗ Pods can consume unlimited resources
- ✗ One misconfigured deployment affects entire cluster
- ✗ Teams exceed budgets without visibility
- ✗ In-place resizing becomes dangerous

THE RESOURCE MANAGEMENT TRIFECTA

Pod Resources

What each container needs/uses
→ Defined in YAML specs

LimitRange

Rules for individual containers
→ Namespace-level constraints

ResourceQuota

Budget for entire namespace
→ Total resource limits

POD RESOURCES

REQUESTS & LIMITS

Requests **What I need**

Scheduler allocation

Guaranteed resources

QoS baseline

Limits **Maximum I can use**

Protection boundary

Throttling trigger

OOM threshold

POD RESOURCES IN ACTION

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
  - name: app
    image: nginx
    resources:
      requests:
        cpu: "500m"
        memory: "512Mi"
      limits:
        cpu: "1"
        memory: "1Gi"
```

What I need to start

Maximum I can consume

2 LIMIT RANGE

INDIVIDUAL CONTAINER GUARDRAILS

What it controls

Max size per container

Min size per container

Default values (if not specified)

Why it matters

Prevents oversized containers from destabilizing nodes

Key insight

Validates EVERY pod creation and resize operation

LIMIT RANGE CONFIGURATION

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: container-limits
  namespace: my-namespace
spec:
  limits:
  - type: Container
    max:
      cpu: "4"
      memory: "8Gi"
    min:
      cpu: "100m"
      memory: "128Mi"
  default:
    cpu: "1"
    memory: "1Gi"
```

No container can exceed this

No container can be smaller

Applied if not specified

3 RESOURCE QUOTA

NAMESPACE BUDGET CONTROL

What it limits

Total CPU across all pods

Total Memory across namespace

Number of pods, services, PVCs

Storage capacity

Why it matters

Prevents teams from consuming cluster resources uncontrollably

Key insight

Validates cumulative usage, not individual pods

RESOURCEQUOTA CONFIGURATION

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: team-quota
  namespace: my-namespace
spec:
  hard:
    requests.cpu: "50"
    requests.memory: "100Gi"
    limits.cpu: "100"
    limits.memory: "200Gi"
    pods: "100"
```

Total baseline allocation

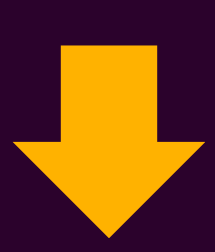
Total burst capacity

HOW THEY WORK TOGETHER

Developer creates or resizes pod

LimitRange validates
“Is this container size OK?”

ResourceQuota validates
“Does team have budget?”



Kubernetes executes
Pod created or resized safely

The Validation Flow

POLICY AS CODE

KYVERNO

Native K8s provides

LimitRange (static rules)

ResourceQuota (hard limits)

Manual enforcement

Kyverno adds

Custom validation rules

Auto-remediation

Advanced conditions

Ratio enforcement

Example

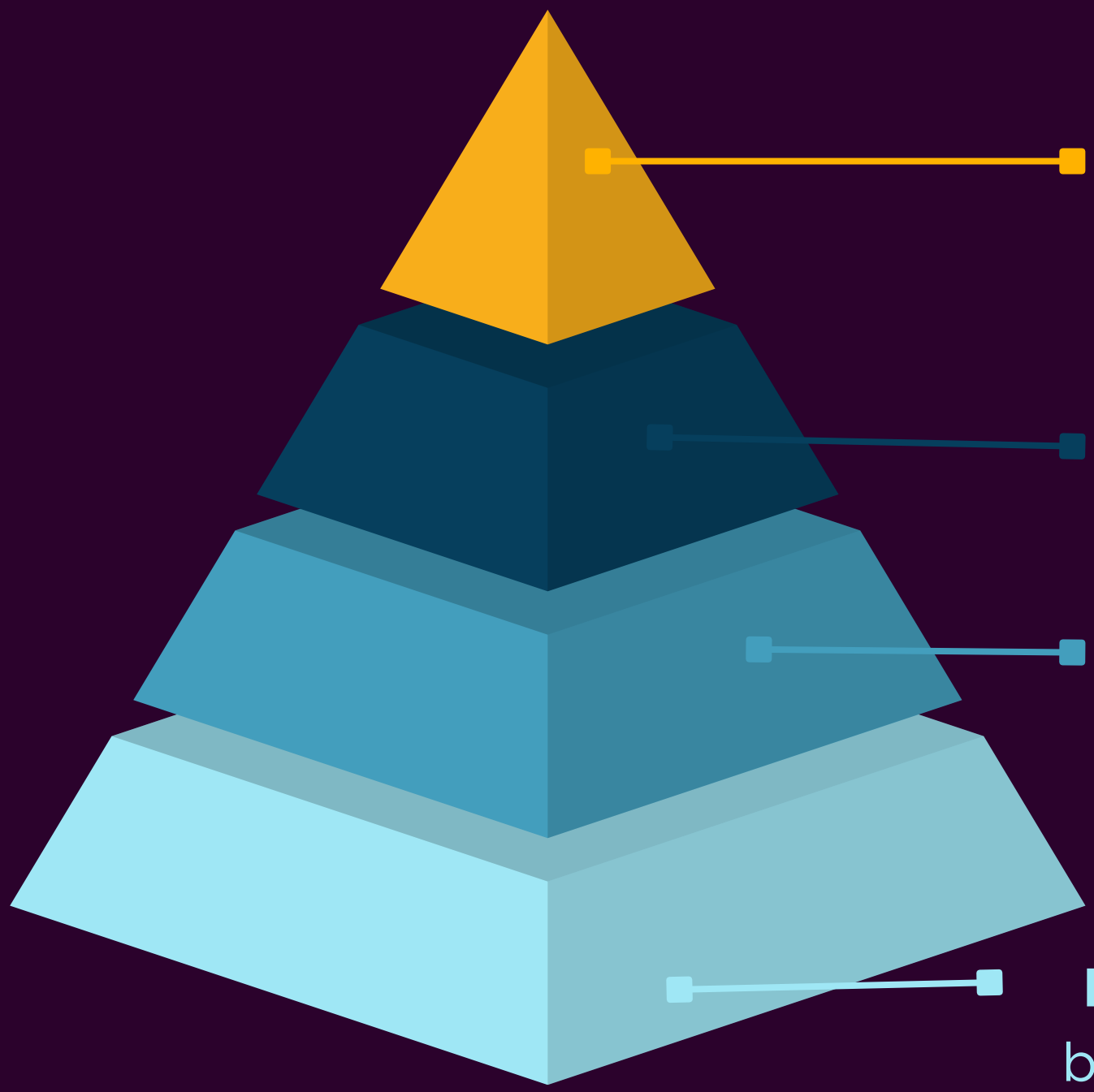
Automatically add resource specs to pods that do not have

KYVERNO AUTO-REMEDIATION

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: add-default-resources
spec:
  rules:
  - name: add-resources
    match:
      resources:
        kinds: [Pod]
    mutate:
      patchStrategicMerge:
        spec:
          containers:
          - (name): "*"
            resources:
              +(requests):
                +(cpu): "100m"
                +(memory): "128Mi"
```

Automatically adds resources to pods → No manual intervention needed

4–LAYER GOVERNANCE



Kyverno Policy-as-Code automation context-aware validation

Pod Resources Developer specifications Container requirements

LimitRange Individual container constraints Min/max boundaries

ResourceQuota Namespace budget control Total resource limits

Complete governance + automation

WHY THIS MATTERS FOR POD RESIZING

Without governance

Resize operations can impact other workloads

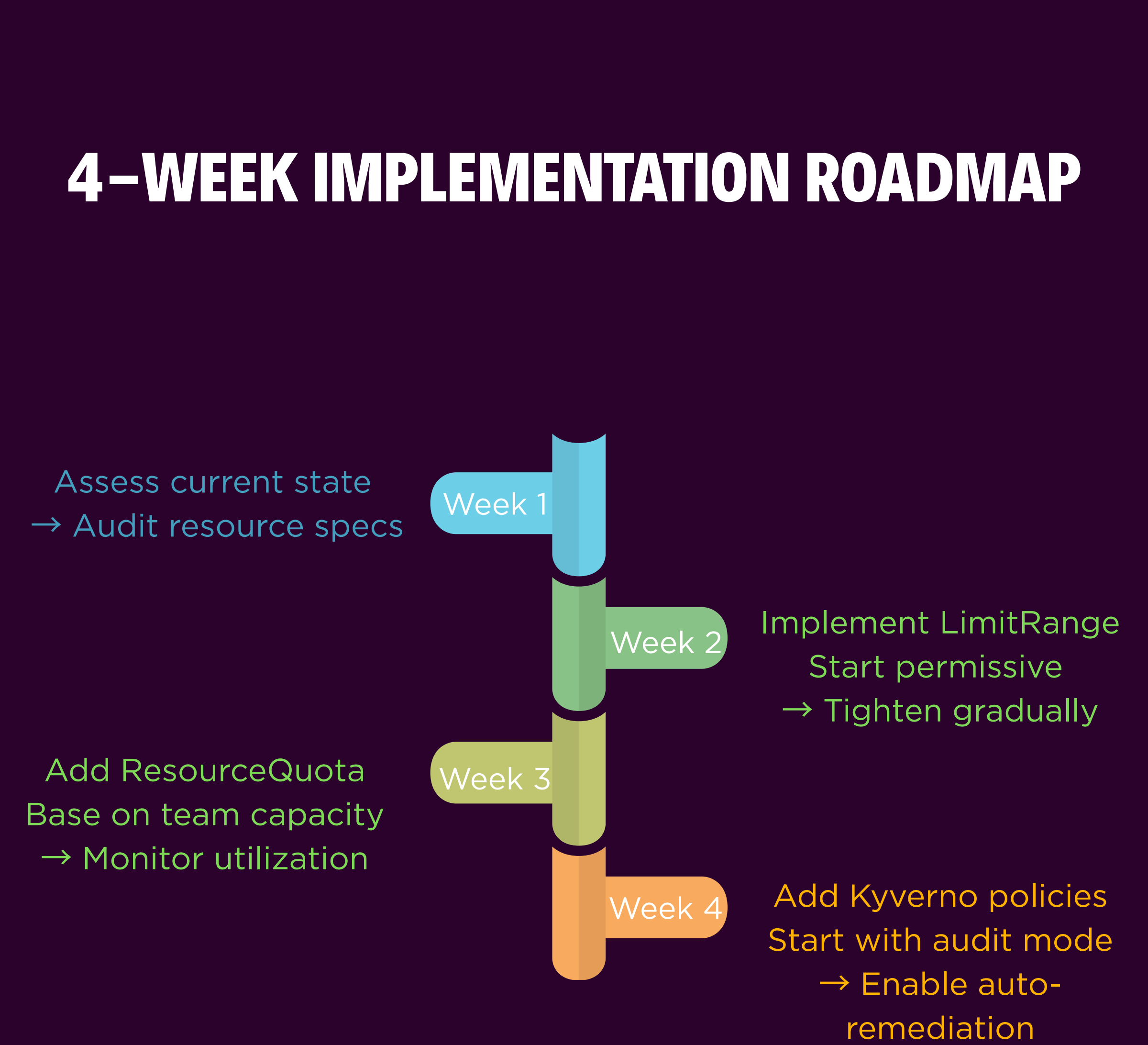
With governance

LimitRange prevents dangerous resizes

ResourceQuota prevents budget overflow

Safe boudaries for dynamic pod sizing

4–WEEK IMPLEMENTATION ROADMAP



Get the Complete Guide

This guide includes:

✅ Native K8s resource management (Pod Resources, LimitRange, ResourceQuota)

✅ 4-layer governance architecture with Kyverno

✅ Production-ready Kyverno policies

✅ Complete troubleshooting section with debug commands

✅ Best practices for enterprise environments

💻 GitHub Markdown: https://bit.ly/490bX2L