KubeCon CloudNativeCon India 2026

#KubeCon #CloudNativeCon

Beyond the Primary CR:

The Design Choice of What and What Not to Watch in Kubernetes Operators

KubeCon CloudNativeCon India 2026

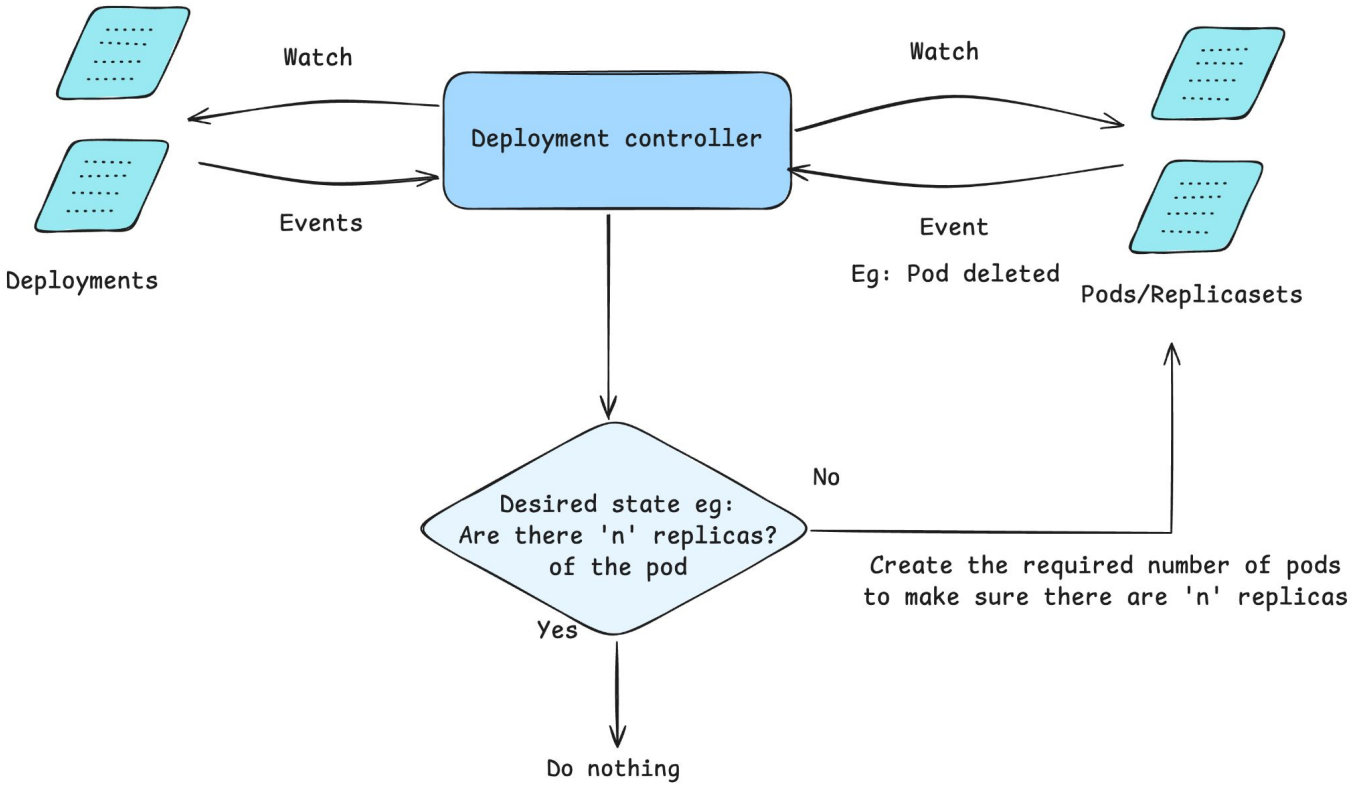
Presenters

Guna K Kambalimath
IBM

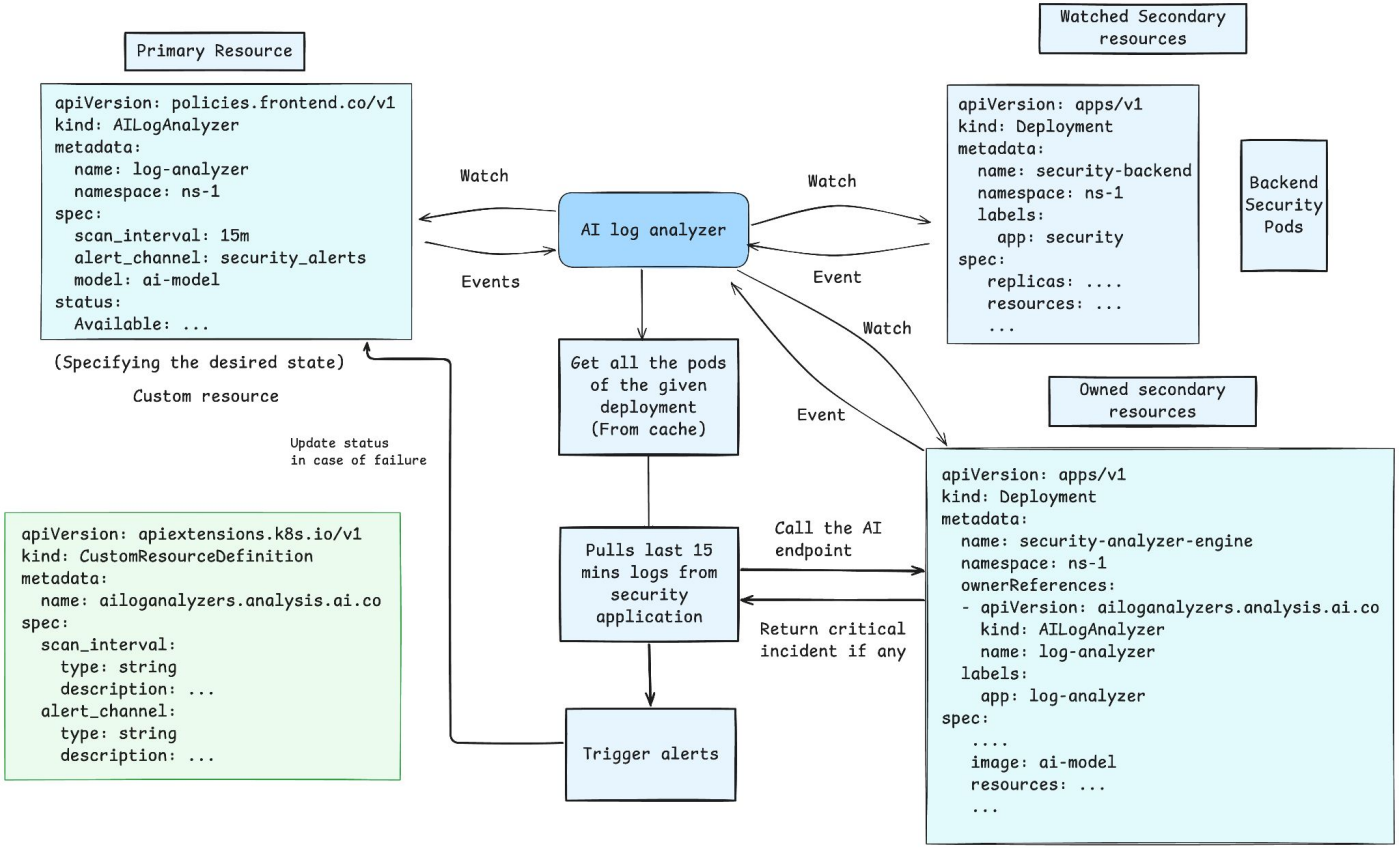
Kishen V
IBM

Introduction

The Analogy!


Watch
Deployment controller
Watch
Events
Event
Deployments
Eg: Pod deleted
Pods/Replicasets
Desired state eg:
Are there 'n' replicas?
of the pod
No
Create the required number of pods
to make sure there are 'n' replicas
Yes
Do nothing

AI log analyzer - Custom controller

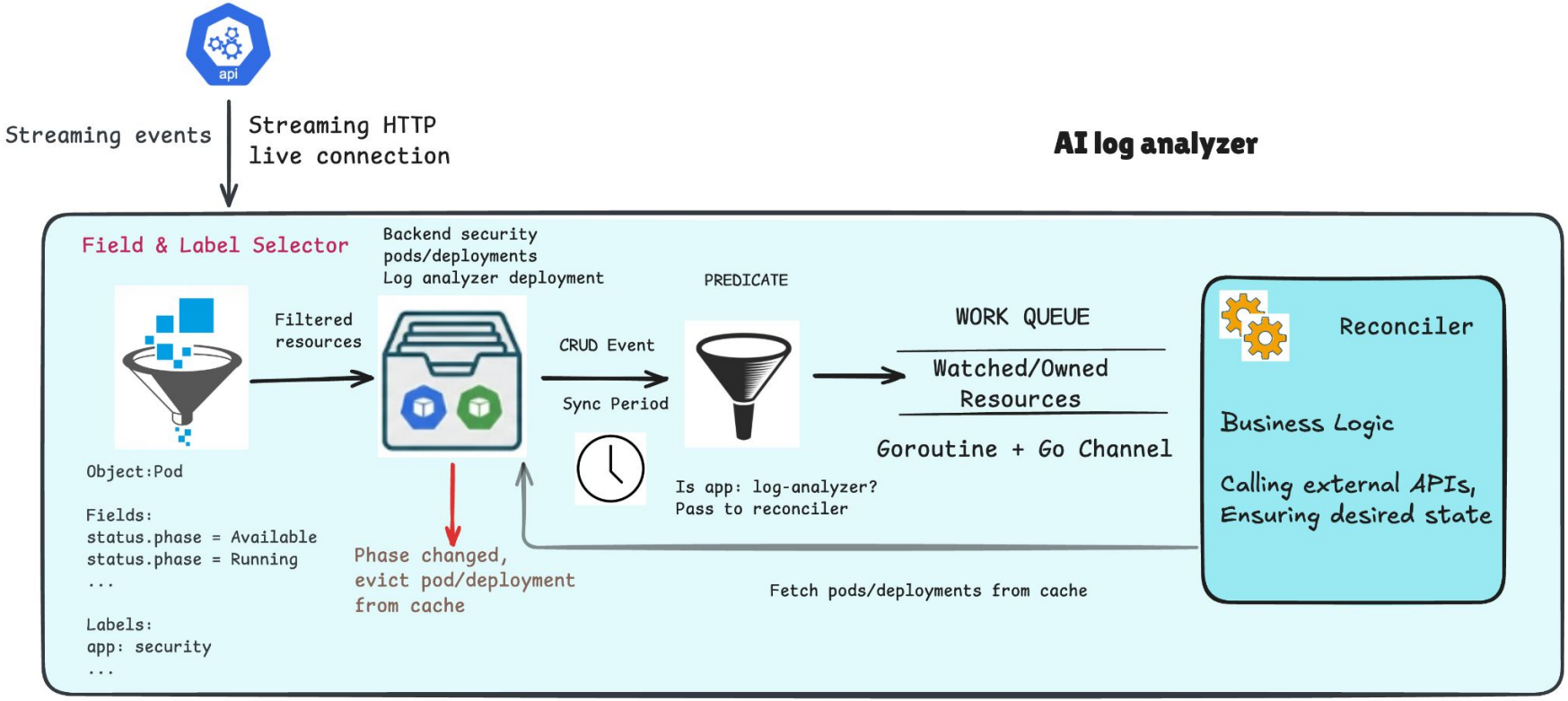


Zooming into the controller cache and its configurable options

Narrowing the Watch

- Field and Label Selectors
- Namespace scoping
- Sync Period
- Predicates

Inside the controller!



Don't Cache the Noise


```
requirement, _ := labels.NewRequirement("app", selection.In,
                    []string{"IAM", "security-backend", "log-analyzer"})

labelSelector := labels.NewSelector()
labelSelector = labelSelector.Add(*requirement)

mgr, err := ctrl.NewManager(ctrl.GetConfigOrDie(), ctrl.Options{
    Scheme:                 scheme,
    Metrics:                metricsServerOptions,
    Cache: cache.Options{
        // Label/Field selector applicable for all watched resources
        DefaultLabelSelector: labelSelector,
        // Generic field selectors in k8s - metadata.name, metadata.namespace
        DefaultFieldSelector: fields.SelectorFromSet(fields.Set{
            "metadata.namespace": "iam-security",
        }),
        // Selectors applicable for specific resources
        ByObject: map[client.Object]cache.ByObject{
            &corev1.Pod{}: {
                // Specific field/label selector
                Field: fields.SelectorFromSet(fields.Set{
                    "status.phase": "Running",
                }),
            },
        },
        SyncPeriod: ptr.To(20 * time.Second),
    },
})
```

Don't Cache the Noise!

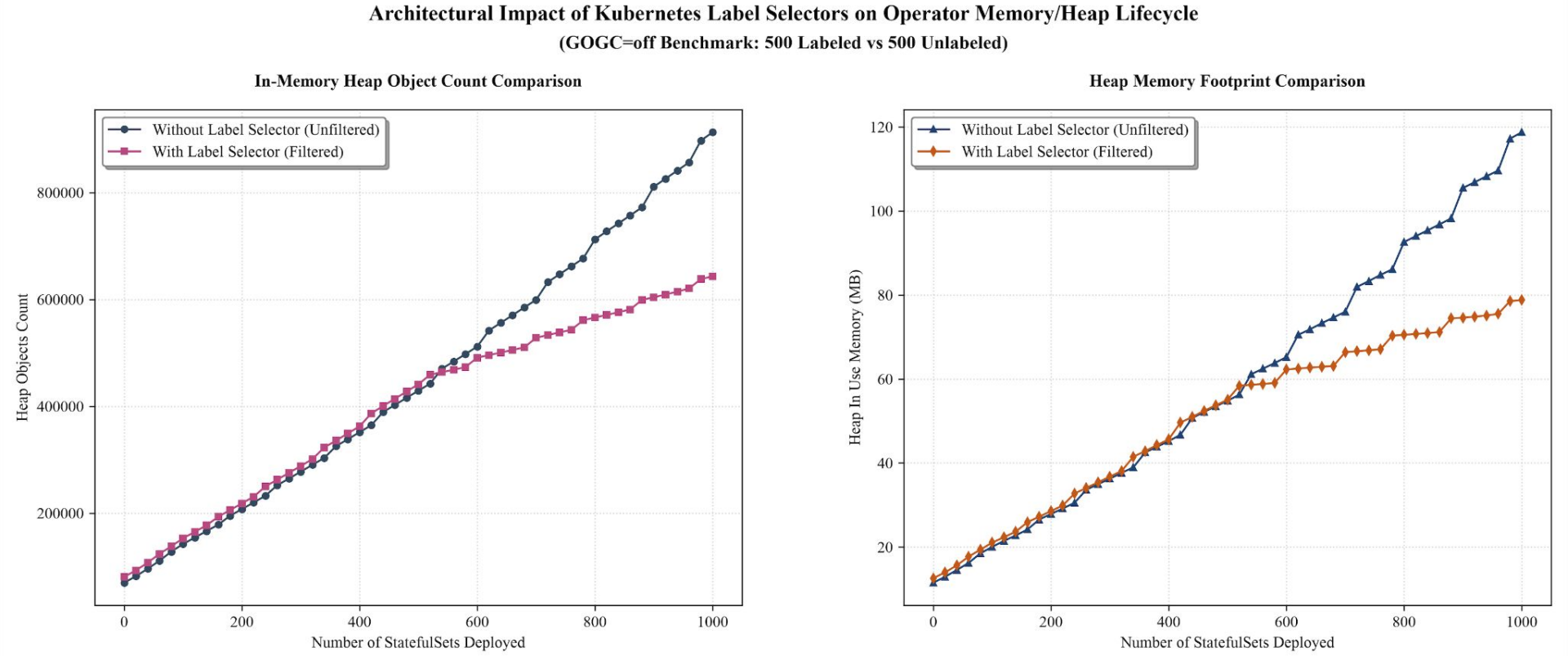

```go
// Add this predicate function before SetupWithManager
func ignoreSpecificLabels() predicate.Predicate {
  return predicate.NewPredicateFuncs(
    func(object client.Object) bool {
      labels := object.GetLabels()
      if labels == nil {
        // No labels, don't allow reconciliation
        return false
      }

      // Check for app label
      if appLabel, exists := labels["app"]; exists {
          // Block reconciliation for IAM and security-backend
          if appLabel == "IAM"
                || appLabel == "security-backend" {
                  return false
          }
      }
      return true // Allow reconciliation for all other cases
    })
}
```

Don't Cache the Noise!

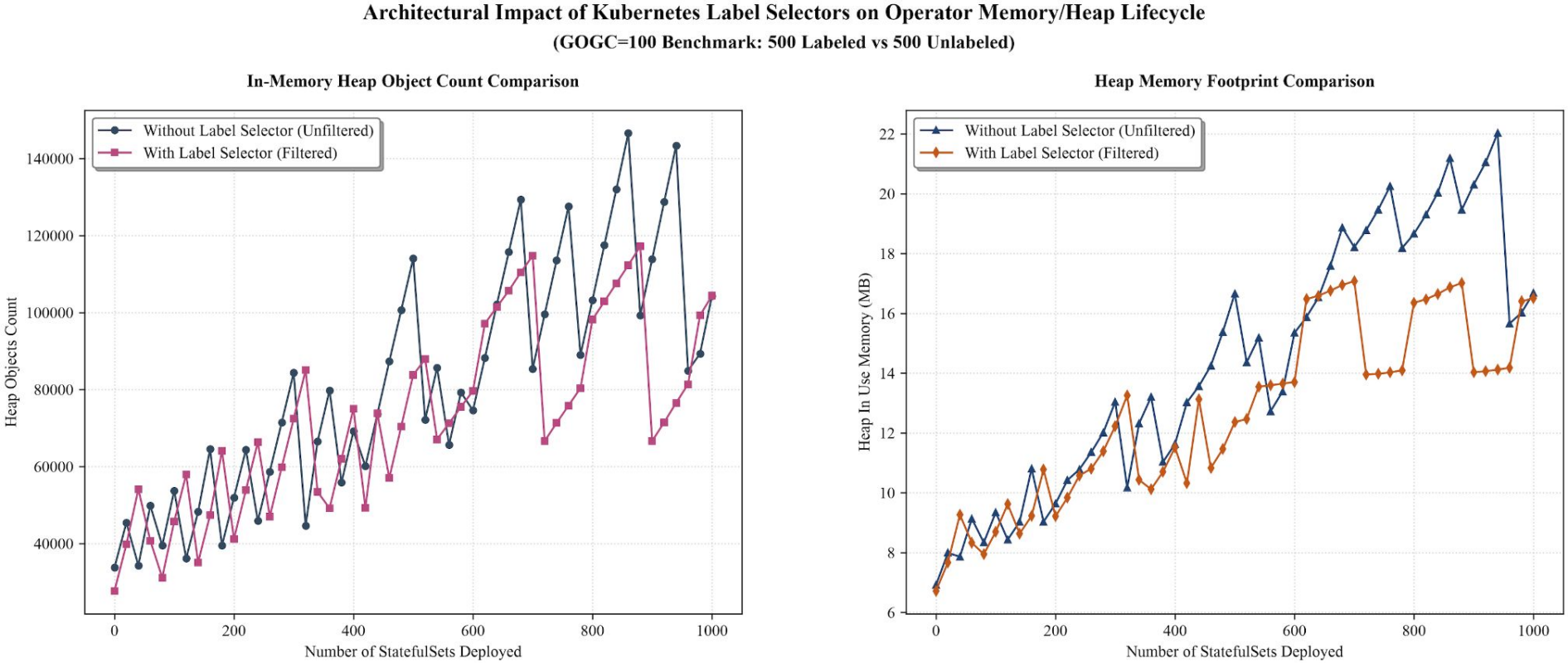
- **Zero Waste Memory**
  - Eliminates the unnecessary cache usage by keeping unready or pending metadata completely out of the operator's cache
- **Reduction in API Noise**
  - Mutes intermediate lifecycle chatter so the operator only reconciles when necessary.
- **Protection from Log Storming**
  - Insulates the operator's workqueue from application churn and the endless retry loops of crashing/unavailable workloads.

What happens when we ignore the ballooning memory?

KubeCon CloudNativeCon India 2026

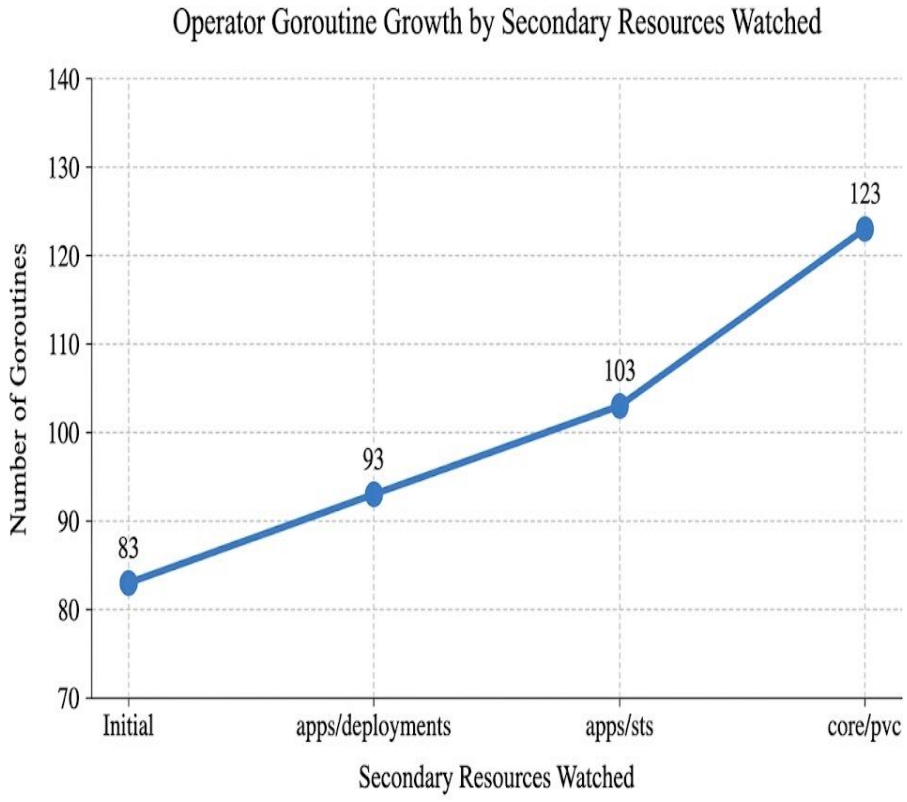


What happens when we ignore the ballooning memory?



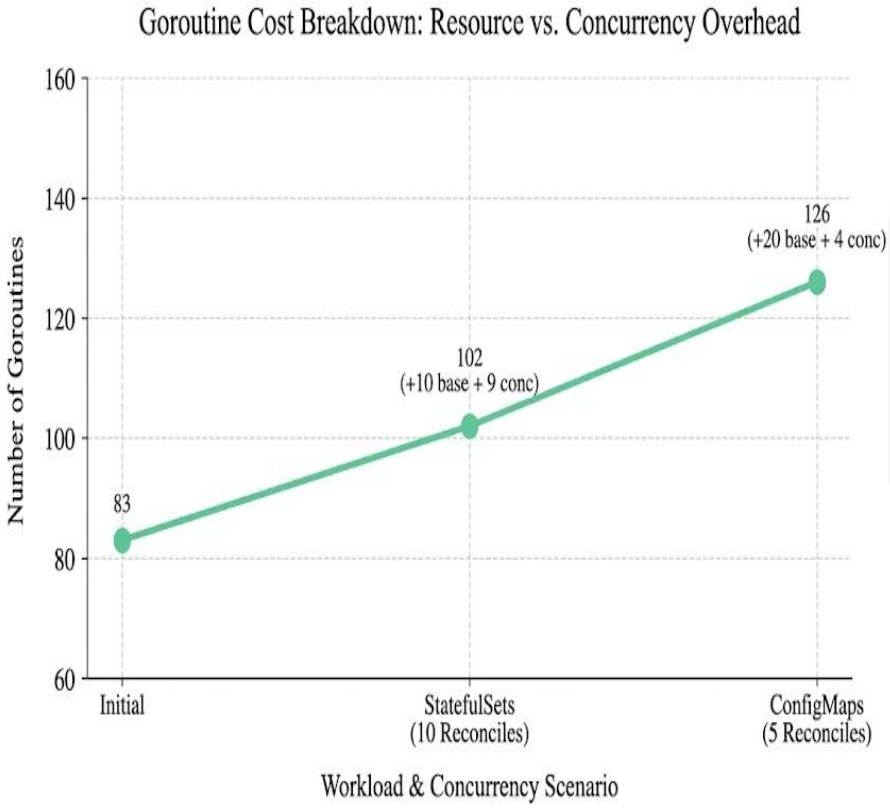
More watches - Longer queues

Goroutine Scaling Across Secondary Resources



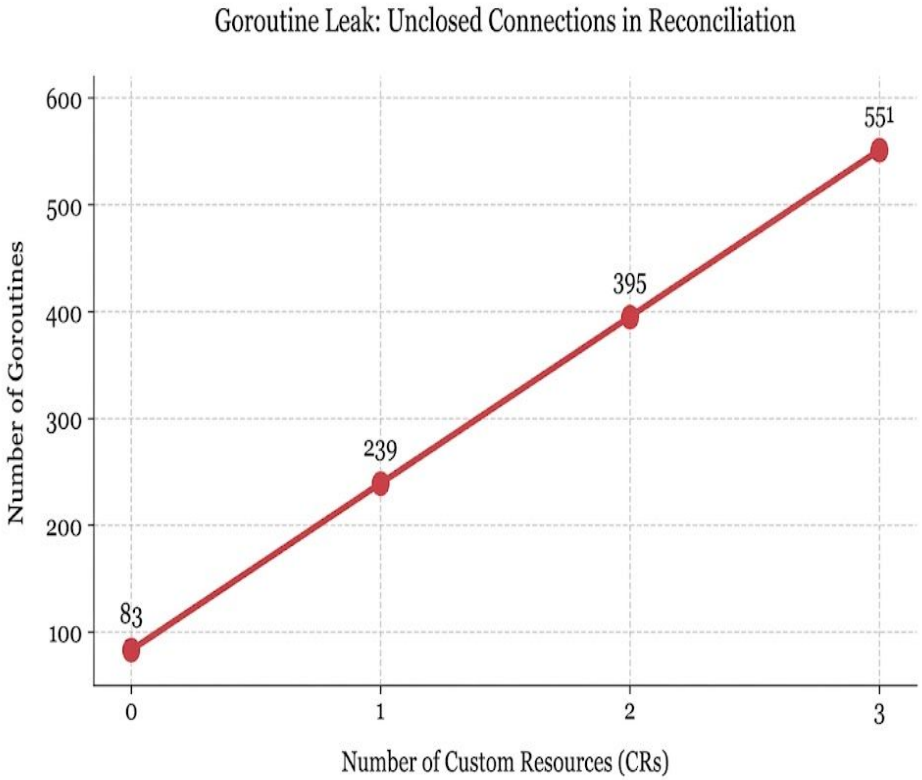

```go
func (r *CustomReconciler) SetupWithManager(mgr ctrl.Manager) error
{
    return ctrl.NewControllerManagedBy(mgr).

    For(&myapi.CustomApp{}).              // Baseline: 83 goroutines

    // Secondary Resources Watched (Using explicit Watches mapping)

    Watches(&appsv1.Deployment{}).  // +10 (Total: 93)

    Watches(&appsv1.StatefulSet{}). // +10 (Total: 103)

    Watches(&corev1.PersistentVolumeClaim{}) // +20 (Total: 123)

}
```

Secondary resource with concurrency overhead!



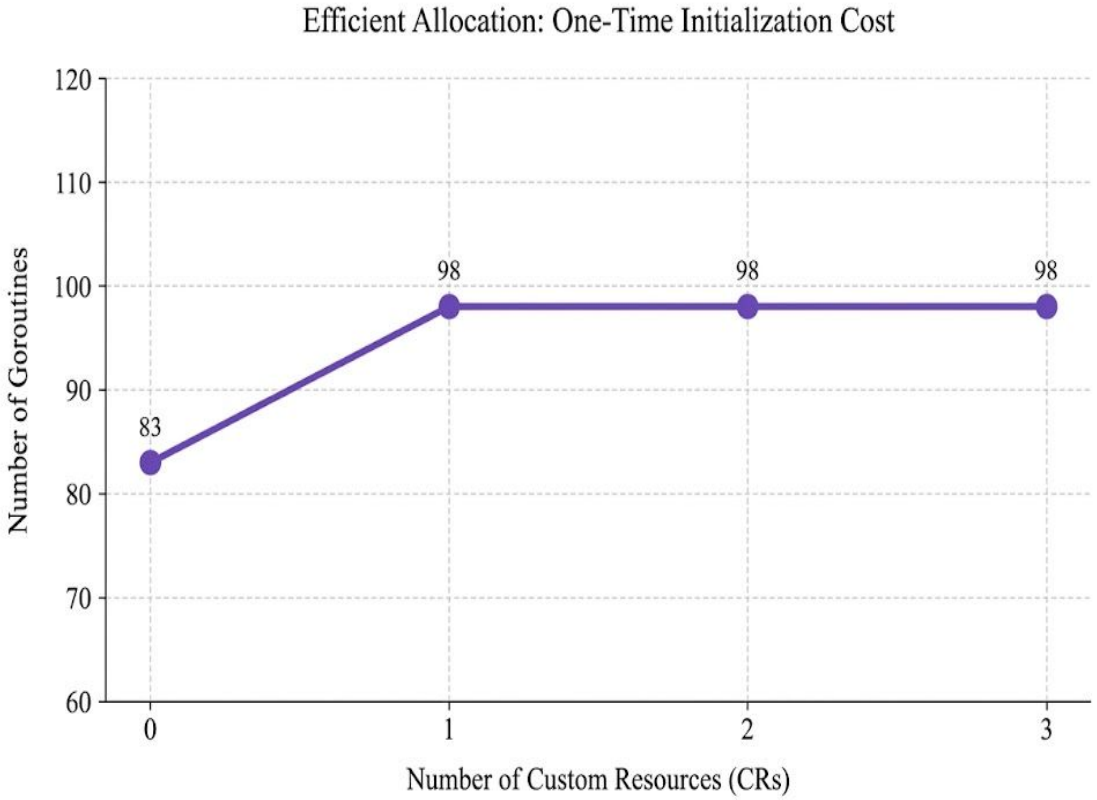

```go
func (r *CustomReconciler) SetupWithManager(mgr ctrl.Manager) error {
    return ctrl.NewControllerManagedBy(mgr).
        For(&myapi.CustomApp{}).
        Owns(&appsv1.StatefulSet{}).
        WithOptions(controller.Options{MaxConcurrentReconciles: 10}).
        Watches(&corev1.ConfigMap{}, &handler.EnqueueRequestForObject{}).
        WithOptions(controller.Options{MaxConcurrentReconciles: 5})
}
```

Unclosed Client Connections, Goroutine leaks!



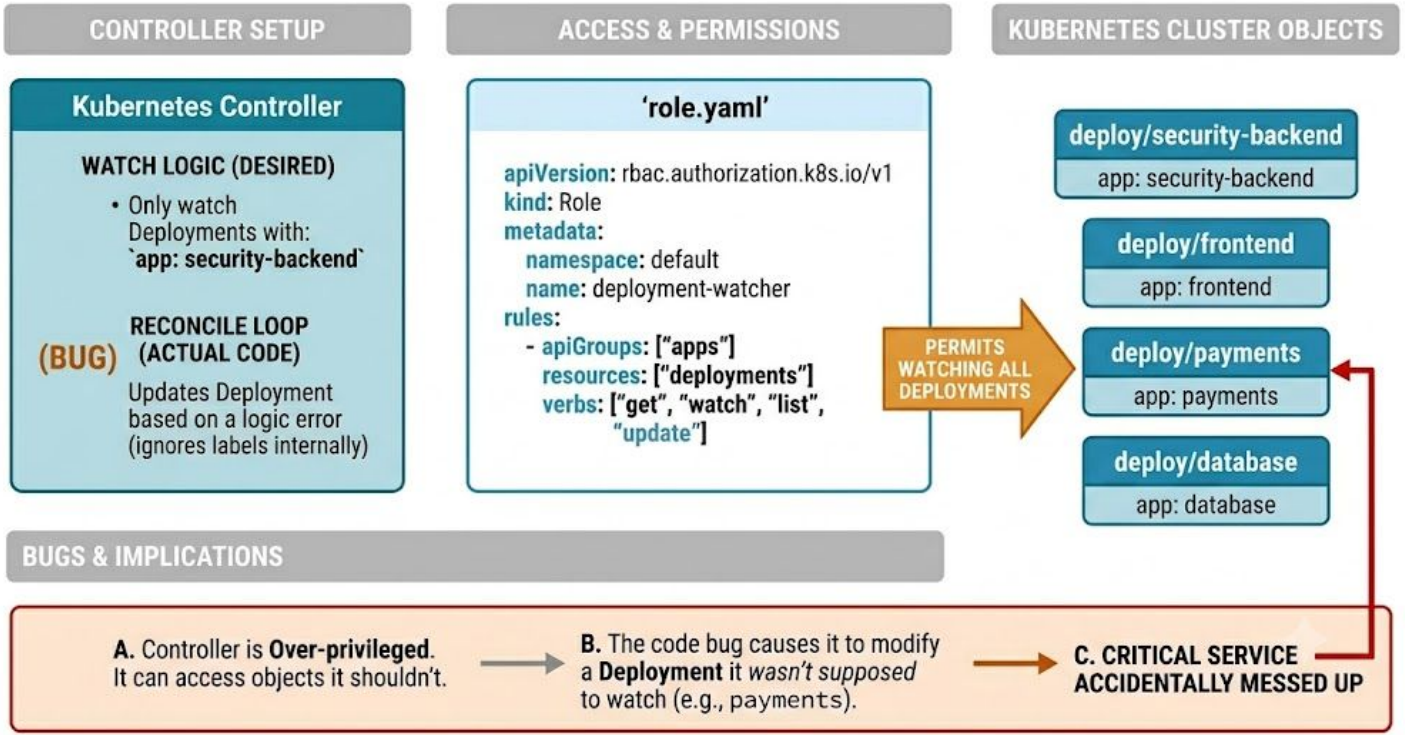

```go
func (r *CustomReconciler) Reconcile(ctx context.Context, req ctrl.Request)
(ctrl.Result, error)
    log := log.FromContext(ctx)
    // Fetch the Custom Resource
    var customResource myapi.CustomApp
    if err := r.Get(ctx, req.NamespacedName, &customResource); err != nil {
        return ctrl.Result{}, client.IgnoreNotFound(err)
    }

    // Creating a new connection client inside the Reconcile loop
    client, err := externalapi.NewClient(customResource.Spec.Endpoint)
    if err != nil {
        return ctrl.Result{}, err
    }

    // defer client.Close() ← Missing this causes the goroutines to scale up

    // Perform reconciliation
    log.Info("Successfully processed CR", "name", req.Name)
    return ctrl.Result{}, nil
}
```

Goroutines Independent of CR Volume

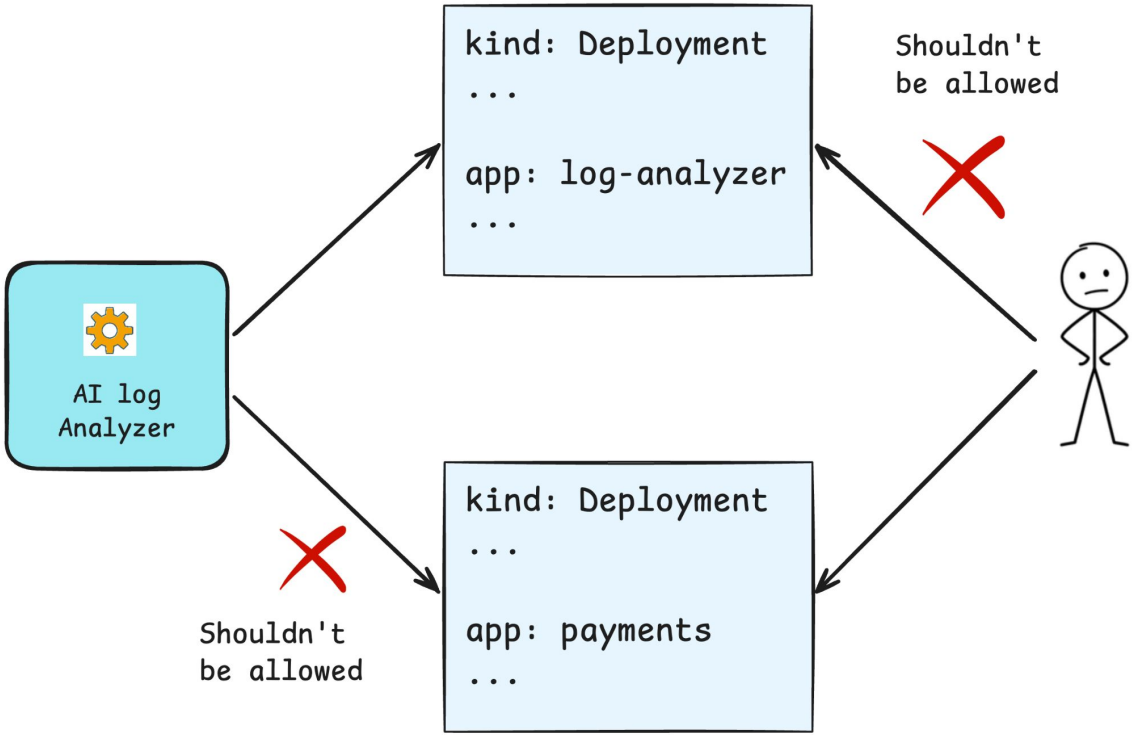


Security and Access control Separation

Over-Privileged Controllers



Over-Privileged Controllers



Validating Webhook


```go
func (v *DeploymentValidator) Handle(ctx context.Context, req admission.Request)
admission.Response {
    deploy := &appsv1.Deployment{}
    if err := v.Decoder.Decode(req, deploy); err != nil {
        return admission.Errored(http.StatusBadRequest, err)
    }

    // 1. Check for the target label
    if deploy.Labels["app"] == "log-analyzer" {

        // 2. Verify if it is owned by the Operator
        hasOperatorOwner := false
        for _, owner := range deploy.GetOwnerReferences() {
            if owner.Kind == "LogAnalyzerOperator" { // Replace with your CRD Kind
                hasOperatorOwner = true
                break
            }
        }

        // 3. Enforce that ONLY the operator's service account can make changes
        if hasOperatorOwner {
            operatorSA := "system:serviceaccount:log-analyzer-system:log-analyzer-
operator"
            if req.UserInfo.Username != operatorSA {
                return admission.Denied("Strictly managed by the Operator. Manual
updates are forbidden.")
            }
        }
    }

    return admission.Allowed("Validation passed")
}
```

Validating Admission Policy


```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: "operator-only-deployment-updates"
spec:
  failurePolicy: Fail
  matchConstraints:
    resourceRules:
    - apiGroups: ["apps"]
      apiVersions: ["v1"]
      operations: ["UPDATE"]
      resources: ["deployments"]
  validations:
    - expression: >
        !(object.metadata.labels['app'] == 'log-analyzer' &&
          object.metadata.ownerReferences.exists(owner, owner.kind ==
'LogAnalyzerOperator')) ||
        request.userInfo.username == 'system:serviceaccount:log-analyzer-system:log-
analyzer-operator'
      message: "This deployment is strictly managed by the Operator. Manual updates are
forbidden."
```

Key Takeaways - configurable options

- **Memory optimization**: Cache only resources you care about
- **Resource v/s latency balancing**: Don’t use a "one-size-fits-all" queue. Assign MaxConcurrentReconciles strategically based on resource criticality
- **Event handling**: Protect your workqueue from noise. Implement custom Predicates to intercept and drop unnecessary events *before* they ever touch your Reconciler.
- **Shift-Left Security**: Leverage native Kubernetes Validating Admission Policies (VAP) to catch and reject updates from non owners.

References

- Single Controller with Multiple CRDs and multiple deployments
- Choosing MaxConcurrentReconciles value
- Watching Secondary Resources
- Scaling to 1000s of CRs
- Reconciliation concurrency
- For v/s Owns v/s Watches in controller
- KubeCon India 2025 - One operator - Multiple controllers

KubeCon

CloudNativeCon

India 2026

Q&A