Credits: https://events.linuxfoundation.org/open-source-summit-india/


@newt_sch
*Welcome Everyone

Why Am here?

Why Am here?

Operators Live in the Past:
Designing Reliable Kubernetes Controllers

Who Am I?

Someshwaran Mohan Kumar

@som23x
blog.someshwaran.me

Senior Developer Advocate

elastic

Elasticsearch Logstash Kibana

LinkedIn

Let’s Make this Interactive?

Let’s understand Who is Here?

Let’s understand Who is Here?

- SREs / Platform Engineers / Cloud / Infra Devs?
- OSS contrib / Operator authors
- Students? Hearing K8s/Operators for the first time?

Let’s Make this Interactive?

Make a promise( )

Agenda

- Elasticsearch And ECK: The Foundation
- What Kubernetes Operators Actually Do
- Inside An Operator: The Backstory & The Internals
- Open Source Projects That Made Our Lives Easier
- Lessons Learned: Best Practices From Production

Worked on Elasticsearch?
ECK? ECE? Heard of it?

Elasticsearch

- Distributed, H/A, search & analytics engine
  - Ranked results (e.g., BM25), fuzzy matching
  - Spelling, Synonyms, Phrases, Stemming
  - Timeseries, geospatial
  - Vector search, (G)AI/ML, RAG search apps
- Scales with index **shards** (primary & replica)
- A shard consists of Lucene segments
- Auto roll-over of time-series (e.g., logs) data **streams** (chain of indices) through data **tiers**
- Shards recover from disk/remote, and relocate

github.com/elastic/elasticsearch

elasticsearch Public

Free and Open Source, Distributed, RESTful Search Engine

Java 75.4k 25.6k

Deployment options:

Public CSP

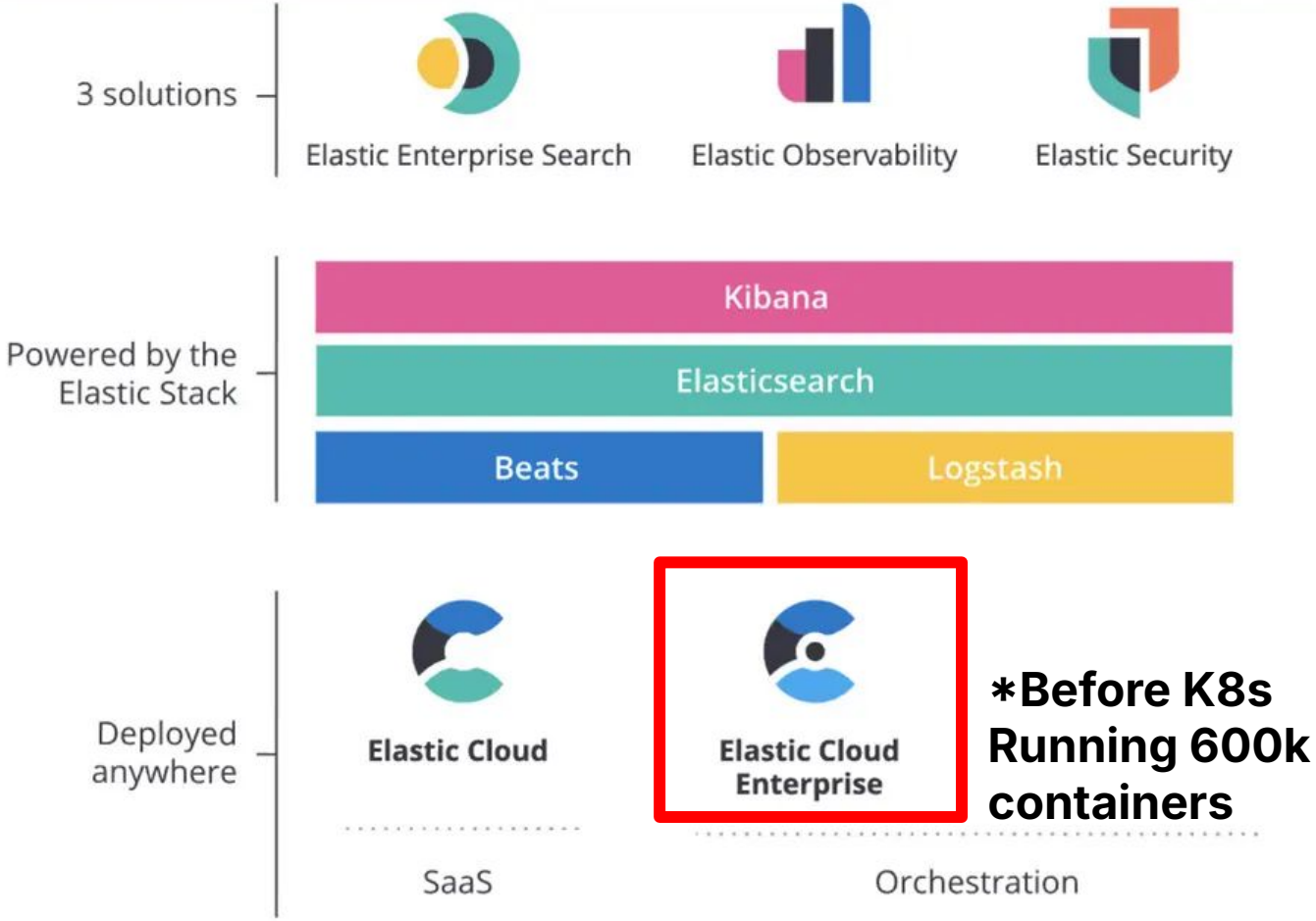
aws Google Cloud Microsoft Azure

Hybrid

On-Premises

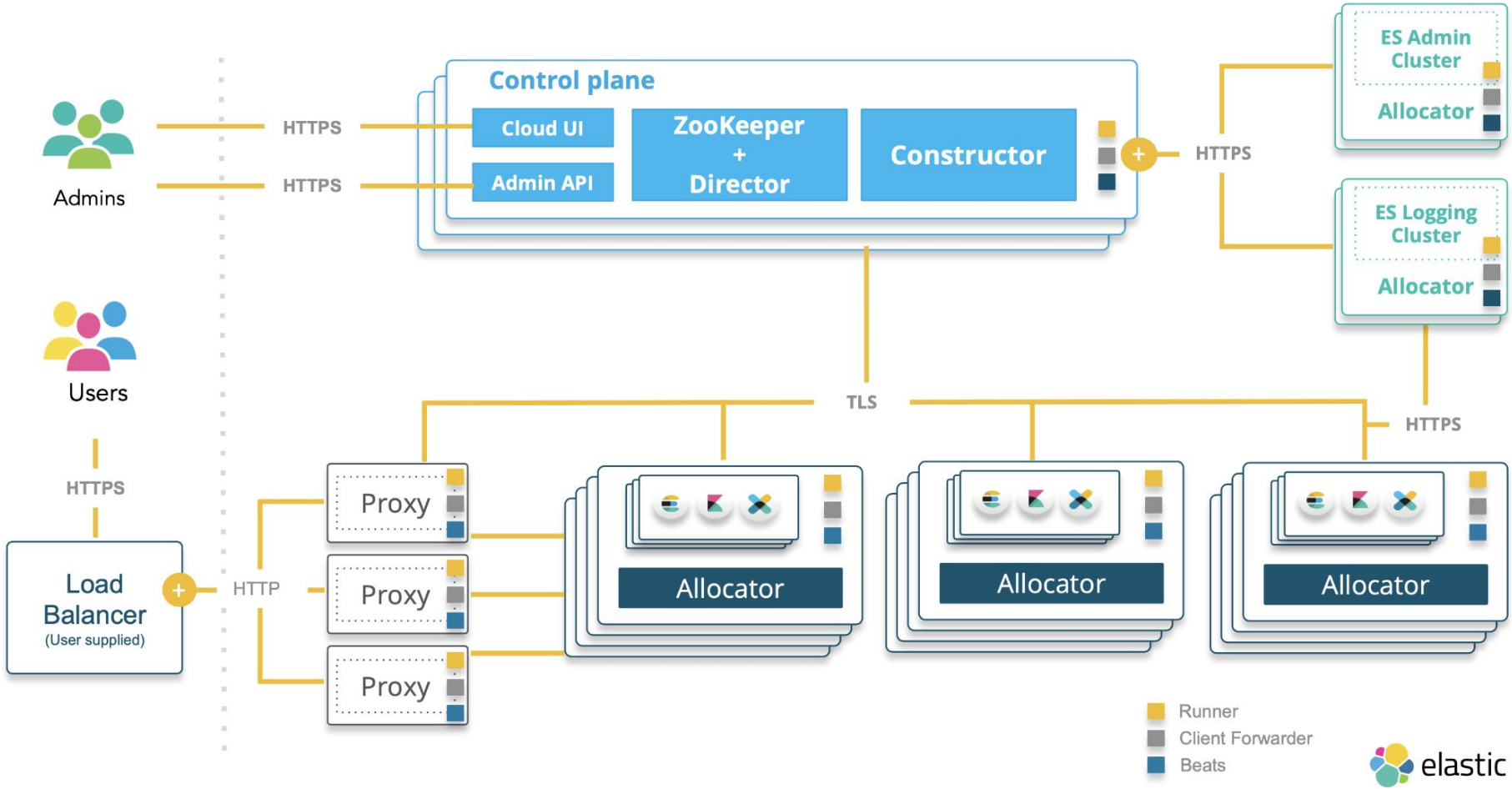
Users:

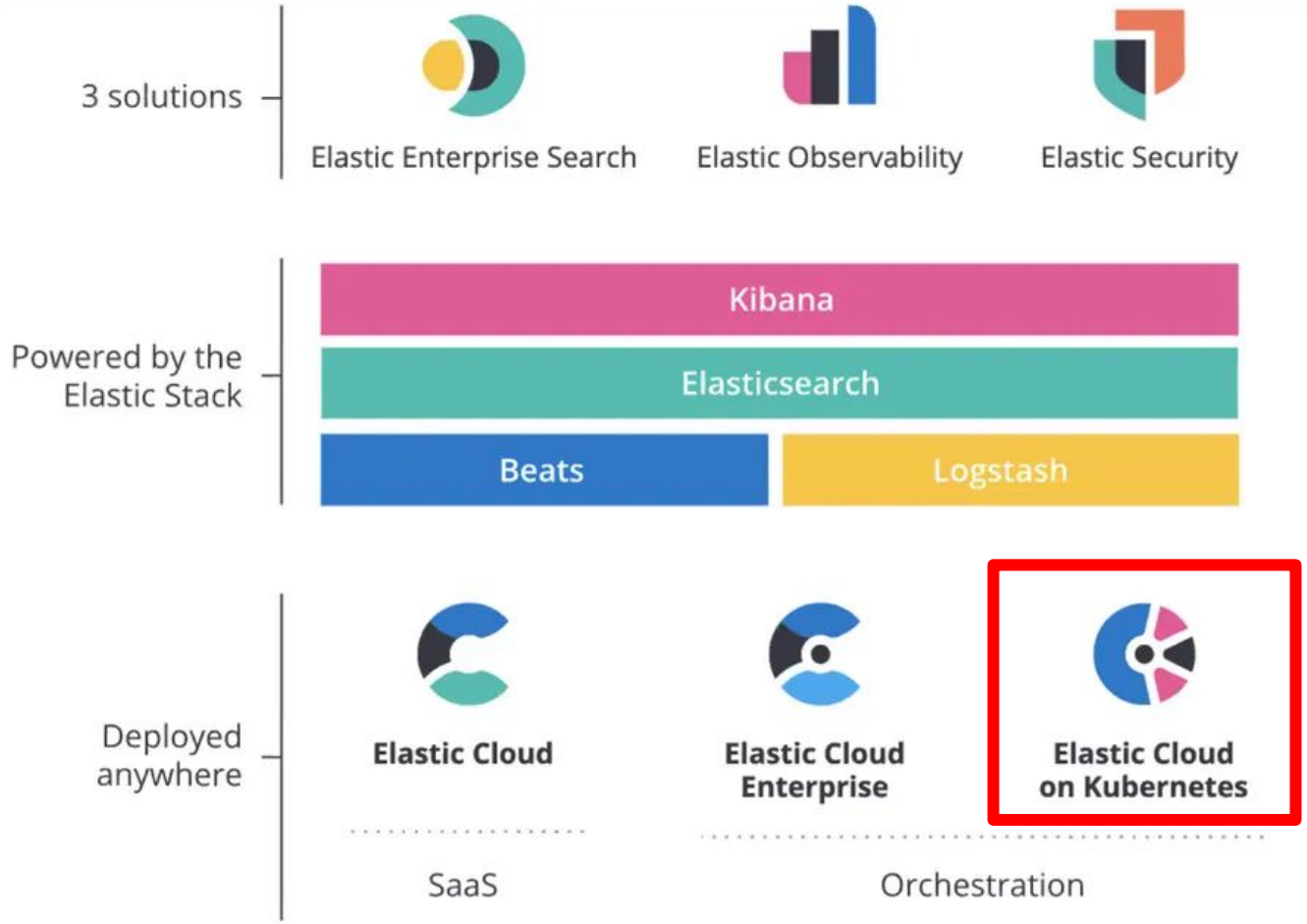
GitHub PhonePe



***Before K8s Running 600k containers**

Elastic Cloud Enterprise high level architecture




3 solutions
Elastic Enterprise Search
Elastic Observability
Elastic Security
Powered by the Elastic Stack
Kibana
Elasticsearch
Beats
Logstash
Deployed anywhere
Elastic Cloud
Elastic Cloud Enterprise
Elastic Cloud on Kubernetes
SaaS
Orchestration

README | Contributing | License | Security

Elastic Cloud on Kubernetes (ECK)

Elastic Cloud on Kubernetes automates the deployment, provisioning, management, and orchestration of Elasticsearch, Kibana, APM Server, Enterprise Search, Beats, Elastic Agent, Elastic Maps Server, Logstash, Elastic AutoOps Agent, and Elastic Package Registry on Kubernetes based on the operator pattern.

Current features:

- Elasticsearch, Kibana, APM Server, Enterprise Search, and Beats deployments
- TLS Certificates management
- Safe Elasticsearch cluster configuration & topology changes
- Persistent volumes usage
- Custom node configuration and attributes
- Secure settings keystore updates

Supported versions:

- Kubernetes 1.31-1.35
- OpenShift 4.16-4.21
- Elasticsearch, Kibana, APM Server: 8+, 9+
- Enterprise Search: 8+
- Beats: 8+, 9+
- Elastic Agent: 8+, 9+ (Fleet, Standalone)
- Elastic Maps Server: 8+, 9+
- Logstash: 8.12+, 9+
- Elastic AutoOps Agent: 9.2.1+ (Enterprise), 9.2.4+ (Basic)
- Elastic Package Registry: 8+

Source: https://github.com/elastic/cloud-on-k8s

Elastic Cloud on Kubernetes

ECK

Built on the Kubernetes Operator pattern, Elastic Cloud on Kubernetes (ECK) extends the basic Kubernetes orchestration capabilities to support the setup and management of Elasticsearch, Kibana, APM Server, Beats, Elastic Agent, Elastic Maps Server, Logstash, AutoOps Agent Policy and Package Registry on Kubernetes.

Source: https://www.elastic.co/docs/deploy-manage/deploy/cloud-on-k8s

Elasticsearch is a Huge StatefulSet!
It’s complex in design

Elastic Cloud on Kubernetes

ECK

Built on the Kubernetes Operator pattern, Elastic Cloud on Kubernetes (ECK) extends the basic Kubernetes orchestration capabilities to support the setup and management of Elasticsearch, Kibana, APM Server, Beats, Elastic Agent, Elastic Maps Server, Logstash, AutoOps Agent Policy and Package Registry on Kubernetes.

Source: https://www.elastic.co/docs/deploy-manage/deploy/cloud-on-k8s

To deploy the ECK operator:

1. Install Elastic's custom resource definitions with `create`:

```
kubectl create -f https://download.elastic.co/downloads/eck/3.4.0/crds.yaml
```

CRDs

You'll see output similar to the following as resources are created:

```
customresourcedefinition.apiextensions.k8s.io/agents.agent.k8s.elastic.co created
customresourcedefinition.apiextensions.k8s.io/apmservers.apm.k8s.elastic.co created
customresourcedefinition.apiextensions.k8s.io/beats.beat.k8s.elastic.co created
customresourcedefinition.apiextensions.k8s.io/elasticmapsservers.maps.k8s.elastic.co created
customresourcedefinition.apiextensions.k8s.io/elasticsearches.elasticsearch.k8s.elastic.co created
customresourcedefinition.apiextensions.k8s.io/kibanas.kibana.k8s.elastic.co created
customresourcedefinition.apiextensions.k8s.io/logstashes.logstash.k8s.elastic.co created
```

2. Using `kubectl apply`, install the operator with its RBAC rules:

```
kubectl apply -f https://download.elastic.co/downloads/eck/3.4.0/operator.yaml
```

Operator

> **Note**
>
> The ECK operator runs by default in the `elastic-system` namespace. While this namespace is used for the operator itself, it is recommended that you deploy your application workloads in a separate, dedicated namespace instead of `elastic-system` or `default`. You will need to consider this when setting up your applications.

3. Using `kubectl logs`, monitor the operator's setup by watching the logs:

```
kubectl -n elastic-system logs -f statefulset.apps/elastic-operator
```

4. Use `kubectl get pods` to check the operator status, passing in the namespace using the `-n` flag:

```
kubectl get -n elastic-system pods
```

When the operator is ready to use, it will report as `Running`

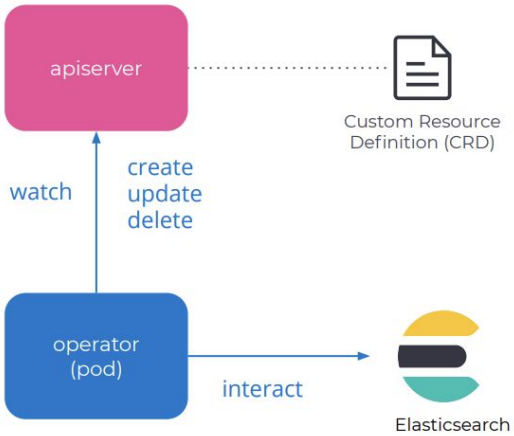
```
$ kubectl get -n elastic-system pods
NAME                 READY   STATUS    RESTARTS   AGE
elastic-operator-0   1/1     Running   0          1m
```

Deploy Elasticsearch / How It Looks!

```
cat <<EOF | kubectl apply -f -
apiVersion: elasticsearch.k8s.elastic.co/v1
kind: Elasticsearch
metadata:
  name: quickstart
spec:
  version: 9.4.2
  nodeSets:
  - name: default
    count: 1
    config:
      node.store.allow_mmap: false
EOF
```

```
NAME          HEALTH   NODES   VERSION   PHASE   AGE
quickstart             1       9.4.2             1s
```

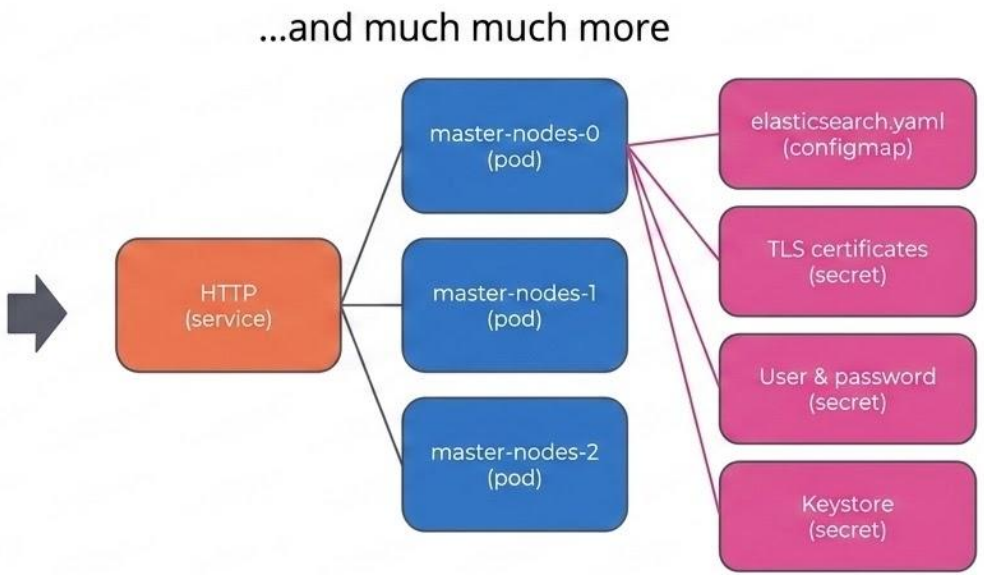
```
NAME                        READY   STATUS    RESTARTS   AGE
quickstart-es-default-0     0/1     Pending   0          9s
```



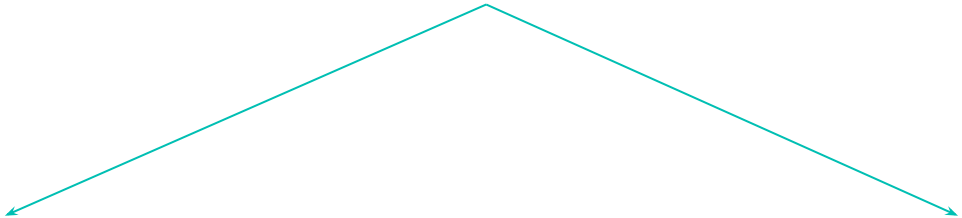
But, Why Does It Use An Operator? Is That Required?

It Makes Our Lives A Bit Easier?

```
apiVersion:
elasticsearch.k8s.elastic.co/v1beta1
kind: Elasticsearch
metadata:
  name: elasticsearch-sample
spec:
  version: 7.5.0
  nodeSets:
  - name: master-nodes
    count: 3
    config:
       node.master: true
  - name: data-nodes
    count: 2
    config:
       node.data: true
```



Let’s Understand Two Things



“What is an Operator”

“Why was it Introduced?”

What is an Operator?

- Operators turn an SRE's “BRAIN” into software (basically a conditional loop inside the K8s cluster)
- You declare what you want, the operator handles the rest, **FOREVER, literally FOREVER**.

Where It All Started?

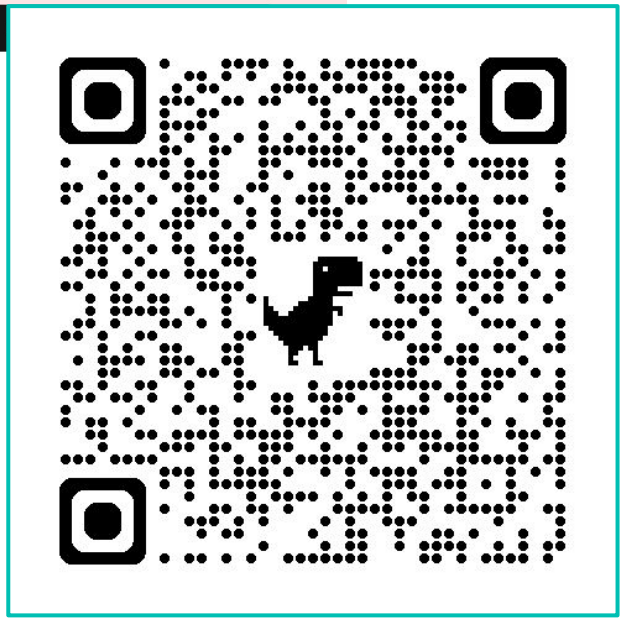
Introducing Operators: Putting Operational Knowledge into Software

Brandon Philips

November 3, 2016 | 6-minute read

Hybrid cloud | Containers

Share



< Back to all posts

A Site Reliability Engineer (SRE) is a person that operates an application by writing software. They are an engineer, a developer, who knows how to develop software specifically for a particular application domain. The resulting piece of software has an application's operational domain knowledge programmed into it.

Our team has been busy in the Kubernetes community designing and implementing this concept to reliably create, configure, and manage complex application instances atop Kubernetes.

We call this new class of software Operators. An Operator is an application-specific controller that extends the Kubernetes API to create, configure, and manage instances of complex stateful applications on behalf of a Kubernetes user. It builds upon the basic Kubernetes resource and controller concepts but includes domain or application-specific knowledge to automate common tasks.

Stateless is Easy, Stateful is Hard

With Kubernetes, it is relatively easy to manage and scale web apps, mobile backends, and API services right out of the box. Why? Because these applications are generally stateless, so the basic Kubernetes APIs, like Deployments, can scale and recover from failures without additional knowledge.

A larger challenge is managing stateful applications, like databases, caches, and monitoring systems. These systems require application domain knowledge to correctly scale, upgrade, and reconfigure while protecting against data loss or unavailability. We want this application-specific operational knowledge encoded into software that leverages the powerful Kubernetes abstractions to run and manage the application correctly.

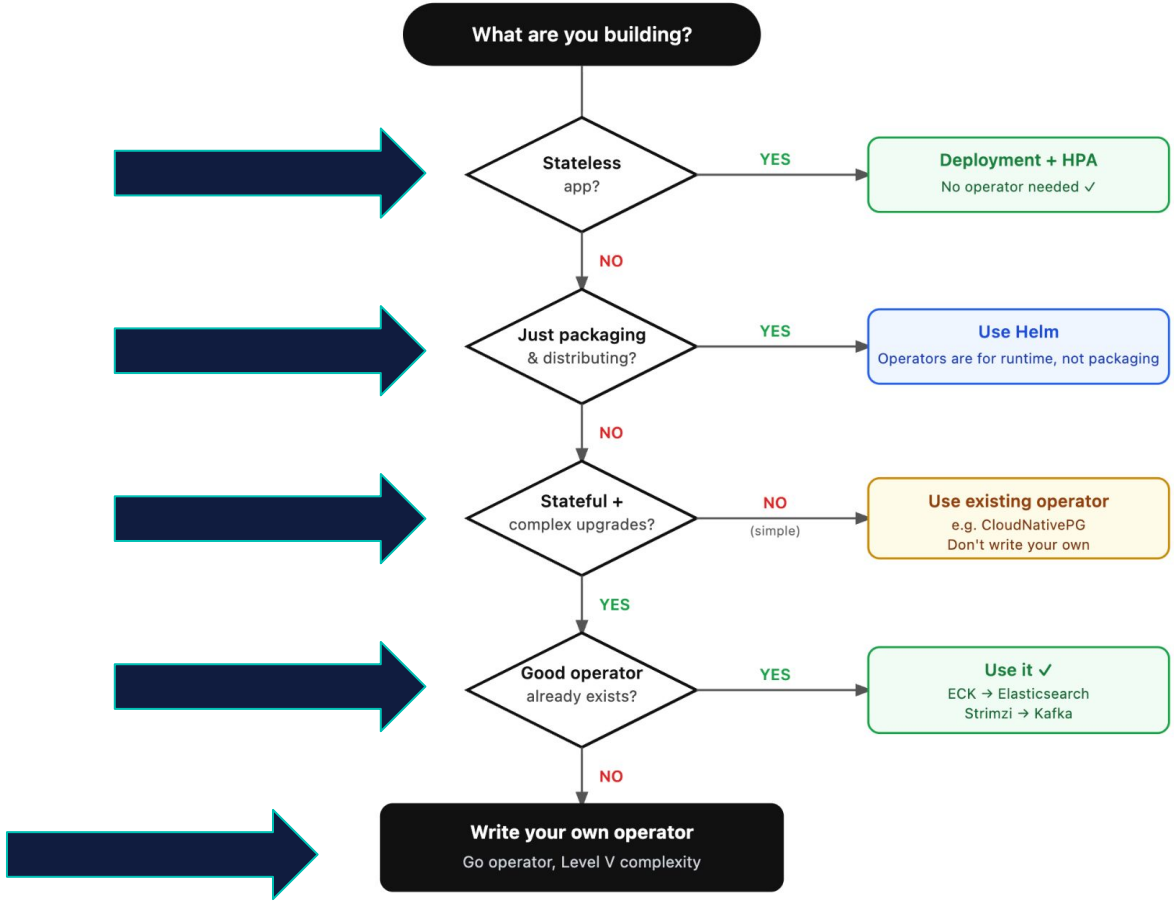
https://www.redhat.com/en/blog/introducing-operators-putting-operational-knowledge-into-software

How Are Operators Created?

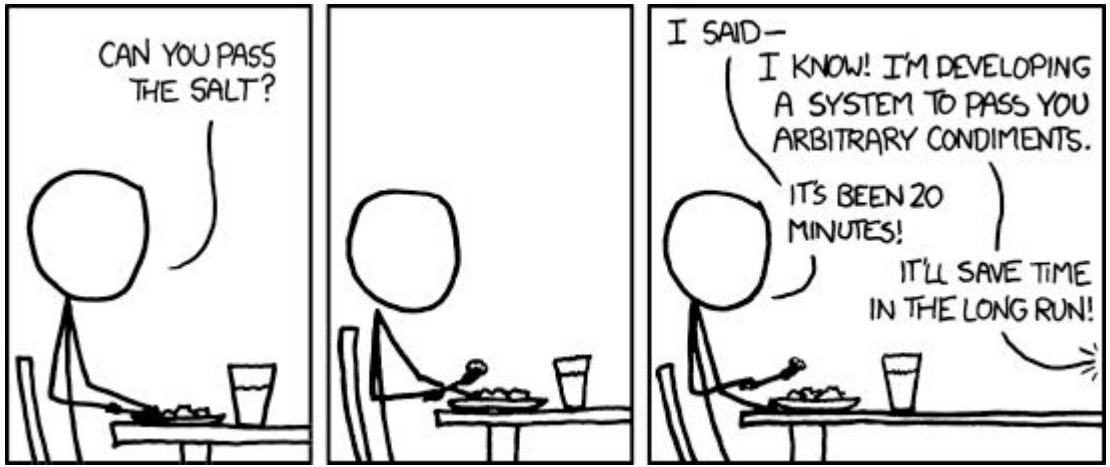
- Controllers
- CustomResource
- CustomResourceDefinitions

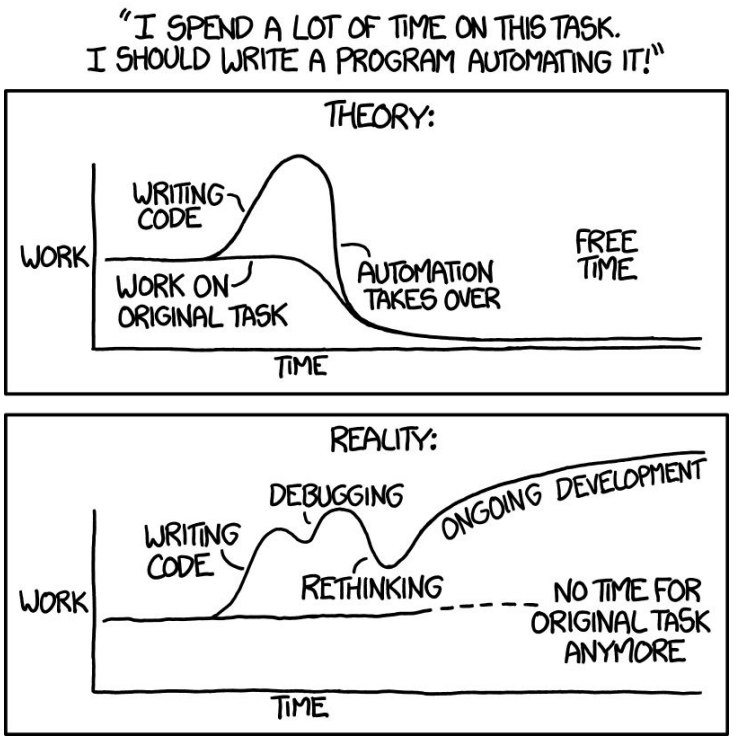
Do You Need An Operator In The First Place?

Do I need a Kubernetes Operator?



Writing Operator Experience! And, I Surely Wanted To Automate



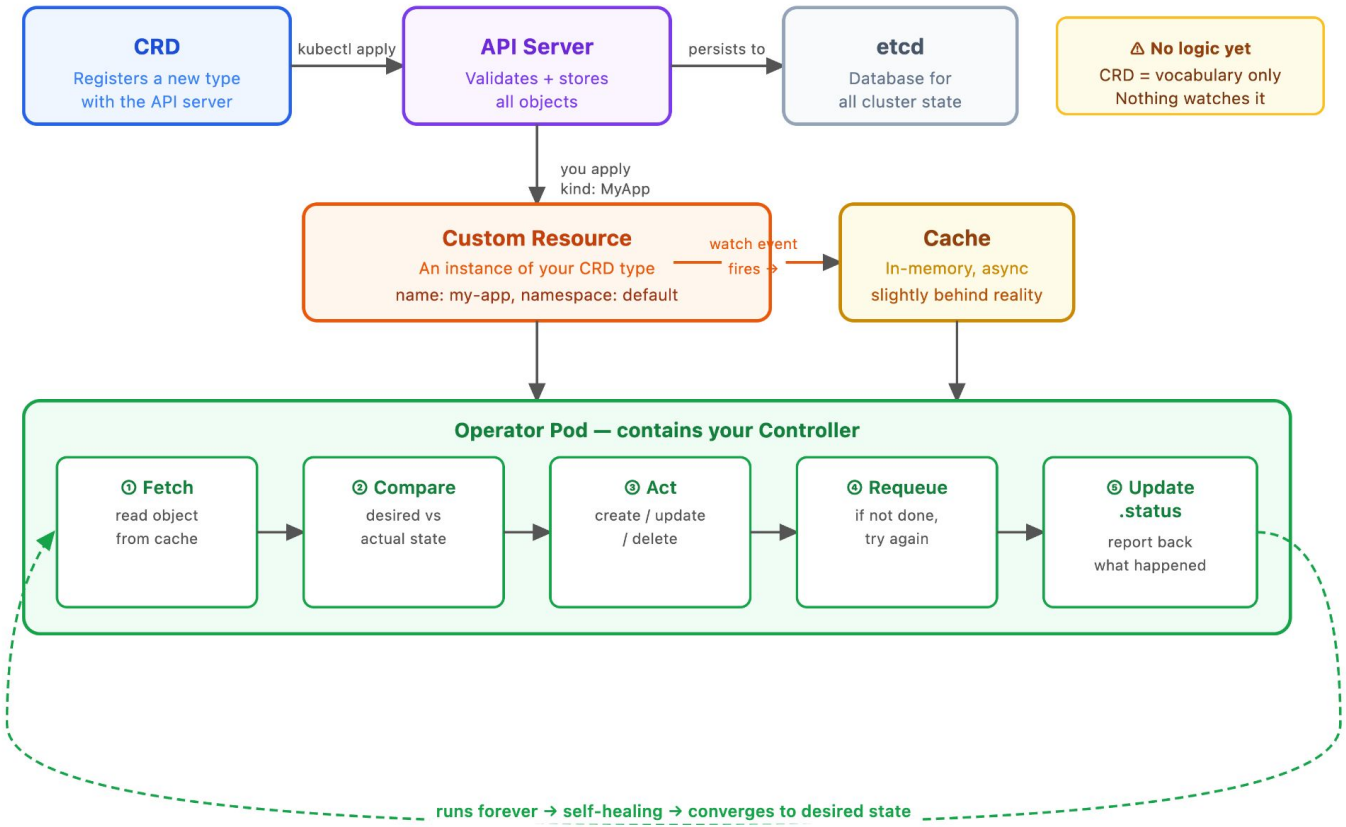


Source: https://xkcd.com/974/ https://xkcd.com/1319/

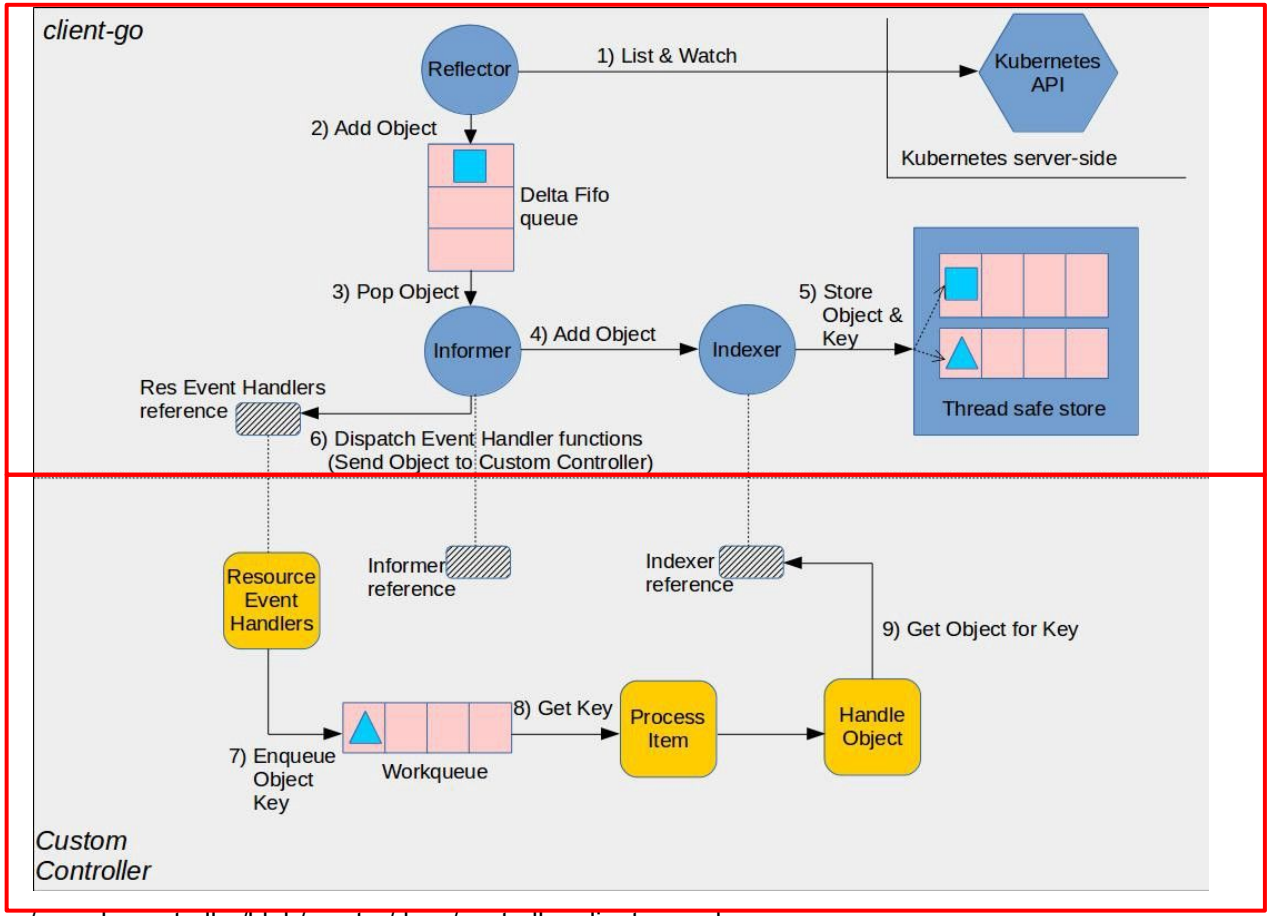
So You Really Wanted to Automate. And, Want an Operator For IT.

Let’s D erators!

How Operators Work



client-go under the hood



https://github.com/kubernetes/sample-controller/blob/master/docs/controller-client-go.md

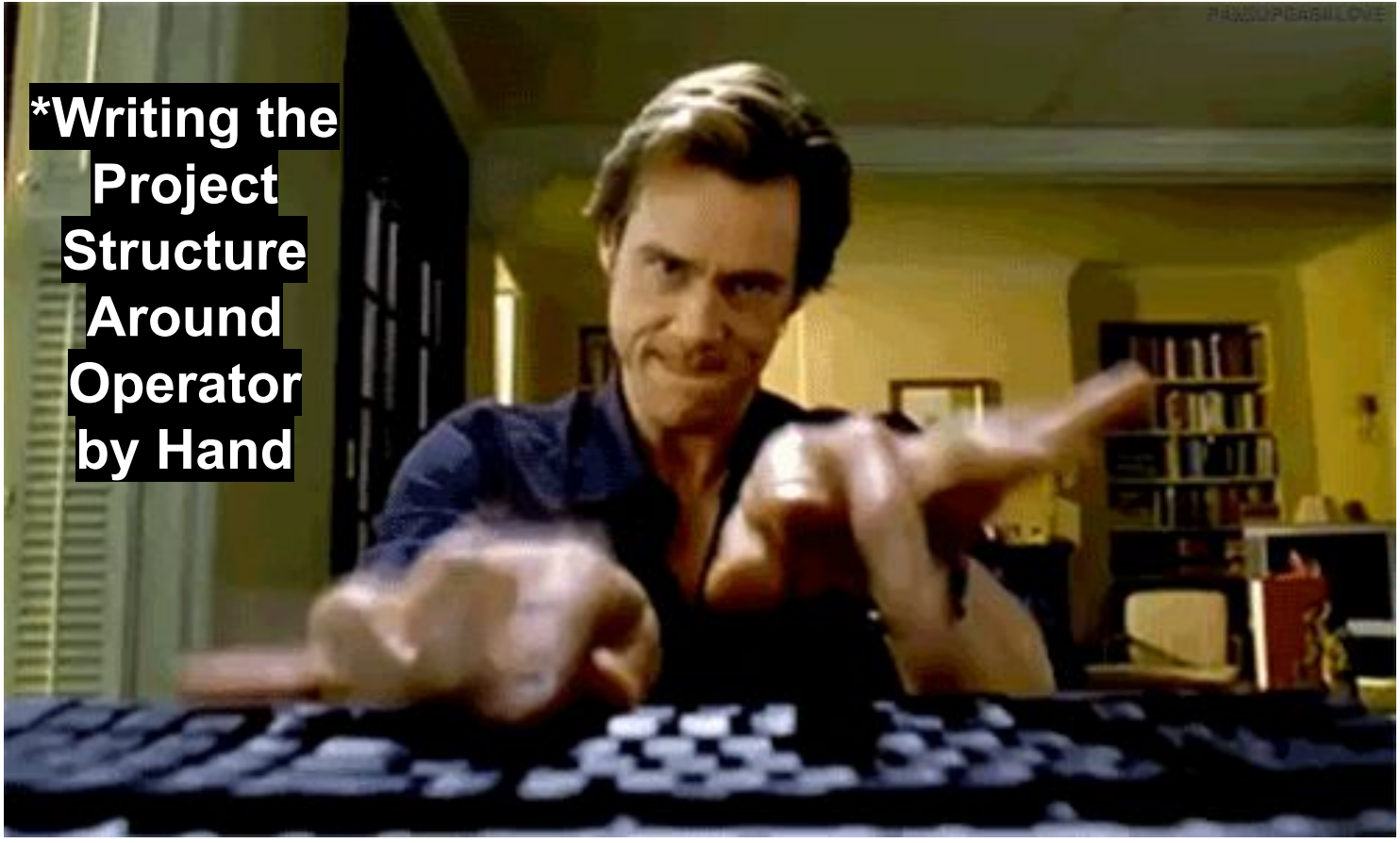
master client-go / examples / workqueue / main.go

Code Blame 224 lines (188 loc) · 7.05 KB Raw

```
149  func main() {
167      }
168
169      ctx := context.Background()
170
171      // create the pod watcher
172      podListWatcher := cache.NewListWatchFromClient(clientset.CoreV1().RESTClient(), "pods", v1.NamespaceDefault, fields.Everything())
173
174      // create the workqueue
175      queue := workqueue.NewTypedRateLimitingQueue(workqueue.DefaultTypedControllerRateLimiter[string]())
176
177      // Bind the workqueue to a cache with the help of an informer. This way we make sure that
178      // whenever the cache is updated, the pod key is added to the workqueue.
179      // Note that when we finally process the item from the workqueue, we might see a newer version
180      // of the Pod than the version which was responsible for triggering the update.
181      indexer, informer := cache.NewIndexerInformer(podListWatcher, &v1.Pod{}, 0, cache.ResourceEventHandlerFuncs{
182          AddFunc: func(obj interface{}) {
183              key, err := cache.MetaNamespaceKeyFunc(obj)
184              if err == nil {
185                  queue.Add(key)
186              }
187          },
188          UpdateFunc: func(old interface{}, new interface{}) {
189              key, err := cache.MetaNamespaceKeyFunc(new)
190              if err == nil {
191                  queue.Add(key)
192              }
193          },
194          DeleteFunc: func(obj interface{}) {
195              // IndexerInformer uses a delta queue, therefore for deletes we have to use this
196              // key function.
197              key, err := cache.DeletionHandlingMetaNamespaceKeyFunc(obj)
198              if err == nil {
199                  queue.Add(key)
200              }
201          },
202      }, cache.Indexers{})
```

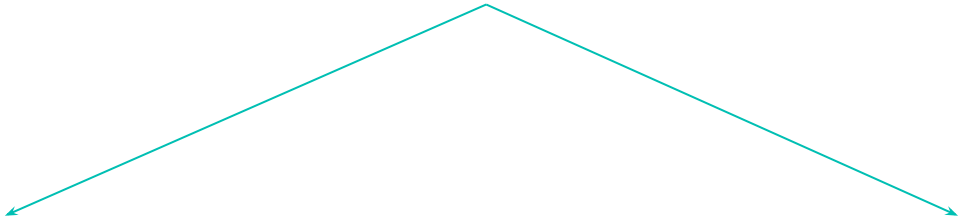
***Enqueue the key, not the object**

To Start Building an Operator!
You Need Hardwork?



Hardwork is NOT Required!
Tools That Do That For You!

To Start Building an Operator!



KubeBuilder

Operator-SDK

What KubeBuilder Generates For You!

- project layout, main.go
- manager + client wiring
- CRD manifests
- RBAC rules
- deepcopy methods
- Dockerfile, Makefile

Called as Scaffolding

What You Write?

- The Reconcile( ) logic
- Your custom resource's spec fields
- What "desired state" means
- Status reporting, ordering, SSA

If You’re Using Go!

Under the Hood: How Kubebuilder Powers the Operator SDK

Kubebuilder is a framework for building Kubernetes APIs using custom resource definitions (CRDs).

Similar to web development frameworks such as *Ruby on Rails* and *SpringBoot*, Kubebuilder increases velocity and reduces the complexity managed by developers for rapidly building and publishing Kubernetes APIs in Go. It builds on top of the canonical techniques used to build the core Kubernetes APIs to provide simple abstractions that reduce boilerplate and toil.

Kubebuilder does **not** exist as an example to *copy-paste*, but instead provides powerful libraries and tools to simplify building and publishing Kubernetes APIs from scratch. It provides a plugin architecture allowing users to take advantage of optional helpers and features. To learn more about this see the Plugin section.

Kubebuilder is developed on top of the controller-runtime and controller-tools libraries.

Kubebuilder is also a library

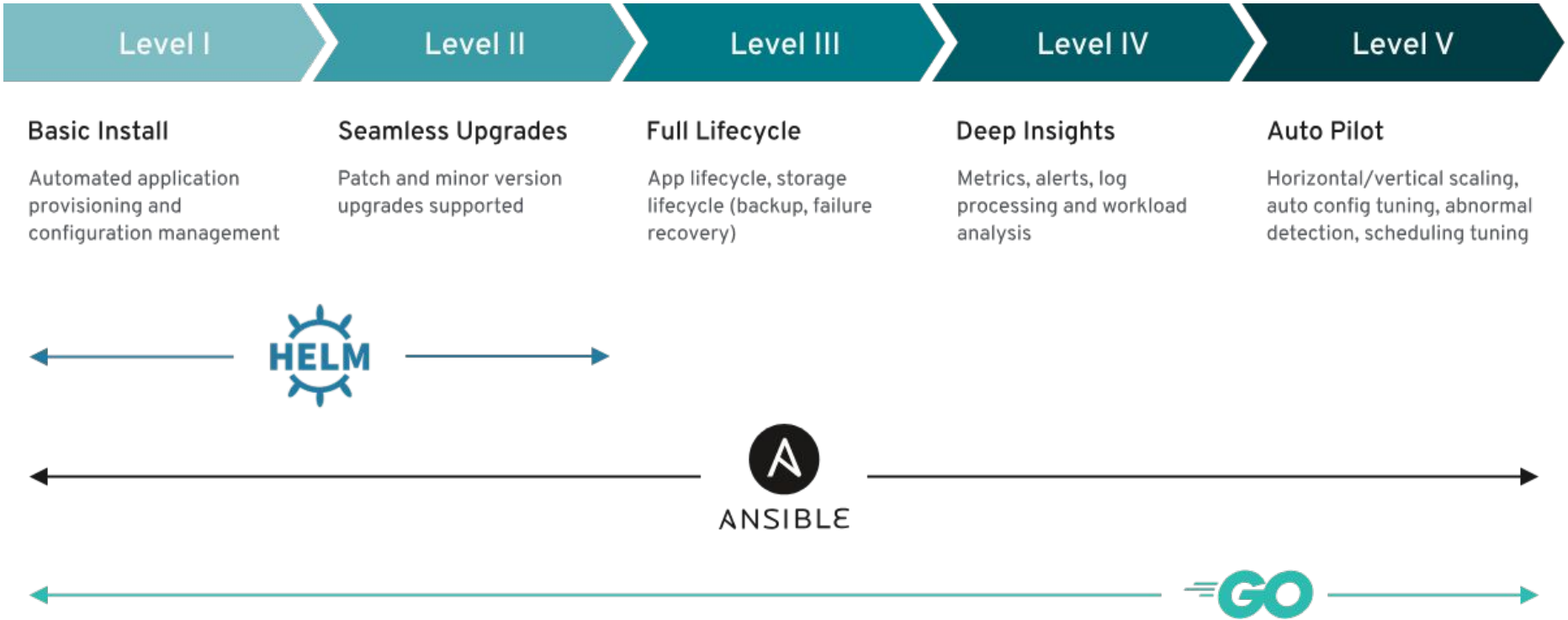
Kubebuilder is extensible and can be used as a library in other projects. Operator-SDK is a good example of a project that uses Kubebuilder as a library. Operator-SDK uses the plugin feature to include non-Go operators *e.g. operator-sdk's Ansible and Helm-based language Operators*.

To learn more see how to create your own plugins.

https://github.com/kubernetes-sigs/kubebuilder

operator-sdk Public

Operator-SDK uses the plugin feature to include non-Go operators *e.g. operator-sdk's Ansible and Helm-based language Operators*.



https://operatorhub.io/getting-started

If NOT Using Go!
Choose One That Aligns

Writing your own operator

If there isn't an operator in the ecosystem that implements the behavior you want, you can code your own.

You also implement an operator (that is, a Controller) using any language / runtime that can act as a client for the Kubernetes API.

Following are a few libraries and tools you can use to write your own cloud native operator.

> **Note:** This section links to third party projects that provide functionality required by Kubernetes. The Kubernetes project authors aren't responsible for these projects, which are listed alphabetically. To add a project to this list, read the content guide before submitting a change. More information.

- Charmed Operator Framework
- Java Operator SDK
- Kopf (Kubernetes Operator Pythonic Framework)
- kube-rs (Rust)
- kubebuilder
- KubeOps (.NET operator SDK)
- Mast
- Metacontroller along with WebHooks that you implement yourself
- Operator Framework
- shell-operator

Source: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/#writing-operator

Make Sure To Use The Right Tool!

Best Practice(s)

Best Practice 1

Develop One Operator Per Application

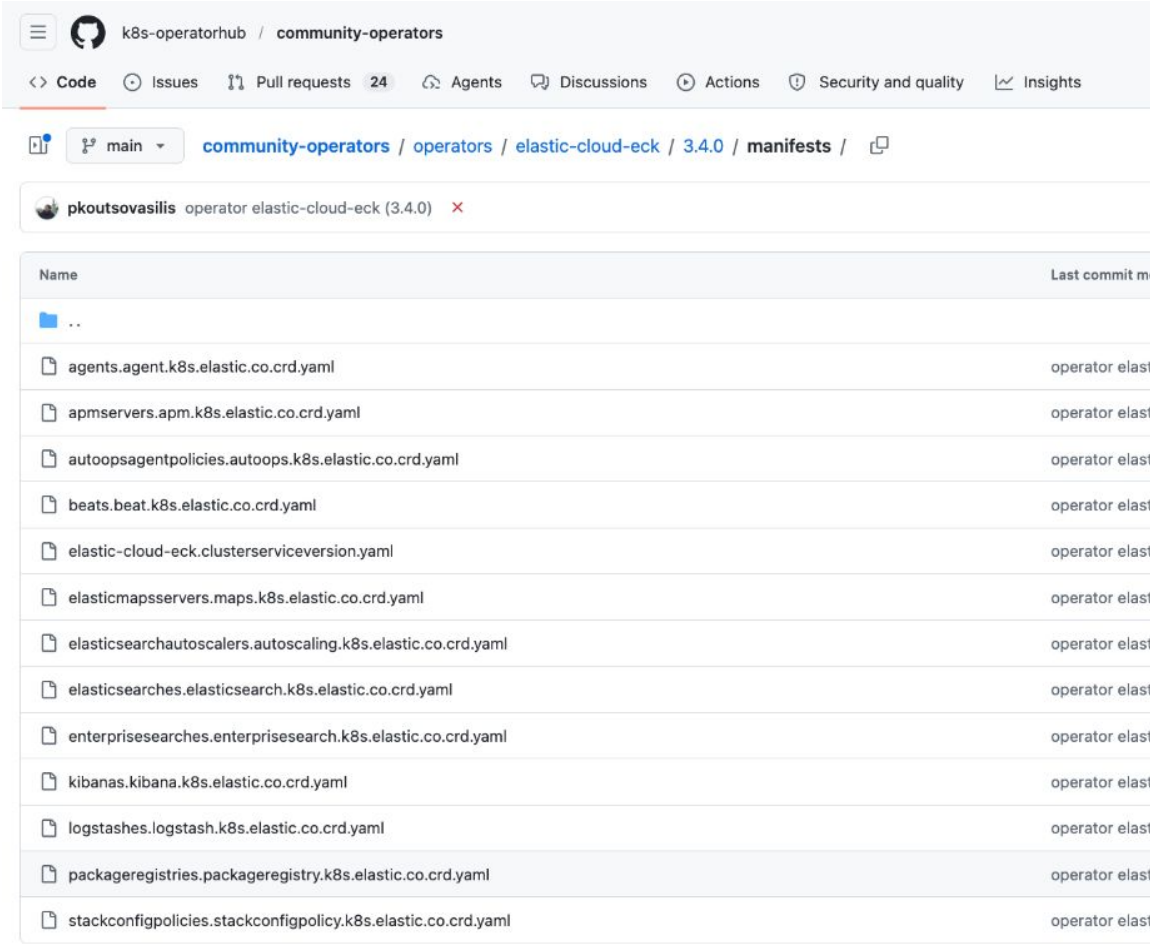
https://sdk.operatorframework.io/docs/best-practices/common-recommendation/


som@Soms-MacBook-Pro — one-operator-per-app
Best practice: one Operator per application
3 unrelated apps -> 3 independent operators (separation of concerns)
1) PostgreSQL
som@Soms-MacBook-Pro ~ % kubectl apply --server-side -f .../cloudnative-pg/.../cnpg-1.28.0.yaml
2) RabbitMQ
som@Soms-MacBook-Pro ~ % kubectl apply -f .../rabbitmq/cluster-operator/.../cluster-operator.yml
3) Redis
som@Soms-MacBook-Pro ~ % helm install redis-operator ot-helm/redis-operator -n ot-operators
som@Soms-MacBook-Pro ~ % kubectl get deploy -A | grep -E 'NAMESPACE|operator|manager'
NAMESPACE NAME READY UP-TO-DATE AVAILABLE AGE
cnpg-system cnpg-controller-manager 1/1 1 1 2m
ot-operators redis-operator 1/1 1 1 90s
rabbitmq-system rabbitmq-cluster-operator 1/1 1 1 45s
som@Soms-MacBook-Pro ~ % kubectl get crds -o custom-columns=GROUP:.spec.group | sort -u | grep -v elastic
postgresql.cnpg.io
rabbitmq.com
redis.redis.opstreelabs.in
each operator owns its CRDs, its namespace, its domain logic
=> 3 apps = 3 operators. don't build one operator to rule them all.
som@Soms-MacBook-Pro ~ %

Best Practice 2

Avoid a Design Solution Where More Than One Kind is Reconciled by the Same Controller

https://sdk.operatorframework.io/docs/best-practices/common-recommendation/



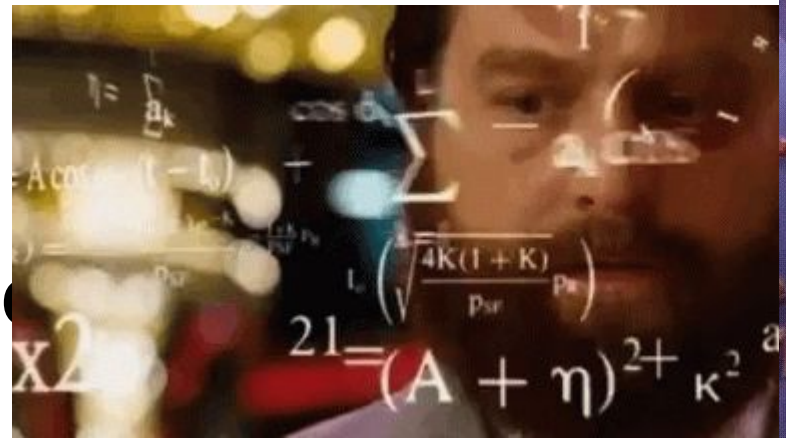
https://github.com/k8s-operatorhub/community-operators/tree/main/operators/elastic-cloud-eck/3.4.0/manifests

```
som@Soms-MacBook-Pro ~ % kubectl get crds
NAME                                                  CREATED AT
agents.agent.k8s.elastic.co                           2026-05-24T10:15:51Z
apmservers.apm.k8s.elastic.co                         2026-05-24T10:15:51Z
autoopsagentpolicies.autoops.k8s.elastic.co           2026-05-24T10:15:51Z
beats.beat.k8s.elastic.co                             2026-05-24T10:15:51Z
elasticmapsservers.maps.k8s.elastic.co                2026-05-24T10:15:51Z
elasticsearchautoscalers.autoscaling.k8s.elastic.co   2026-05-24T10:15:51Z
elasticsearches.elasticsearch.k8s.elastic.co          2026-05-24T10:15:51Z
enterprisesearches.enterprisesearch.k8s.elastic.co    2026-05-24T10:15:51Z
kibanas.kibana.k8s.elastic.co                         2026-05-24T10:15:51Z
logstashes.logstash.k8s.elastic.co                    2026-05-24T10:15:51Z
packageregistries.packageregistry.k8s.elastic.co      2026-05-24T10:15:51Z
stackconfigpolicies.stackconfigpolicy.k8s.elastic.co  2026-05-24T10:15:51Z

som@Soms-MacBook-Pro ~ % kubectl logs sts/elastic-operator -n elastic-system \
    | grep "Starting EventSource" | jq -r .controller | sort -u
agent-controller                    kb-es-association-controller
apm-es-association-controller       kibana-controller
apm-kibana-association-controller   license-controller
apmserver-controller                logstash-controller
autoops-controller                  maps-controller
beat-controller                     packageregistry-controller
elasticsearch-autoscaler            remotecluster-controller
elasticsearch-controller            stackconfigpolicy-controller
enterprisesearch-controller         trial-controller
kb-epr-association-controller       webhook-certificates-controller

som@Soms-MacBook-Pro ~ % kubectl get sts -n elastic-system
NAME               READY   AGE
elastic-operator   1/1     14d

---

# 12 CRDs, all under one API group:  k8s.elastic.co
# 20 controllers, all inside ONE operator binary
# 1 workload backing them:  elastic-operator  (ECK)
#
# => 1 operator,  12 CRDs,  20 controllers
```

Best Practice 2

Let’s Assume, You Don’t Follow Then?

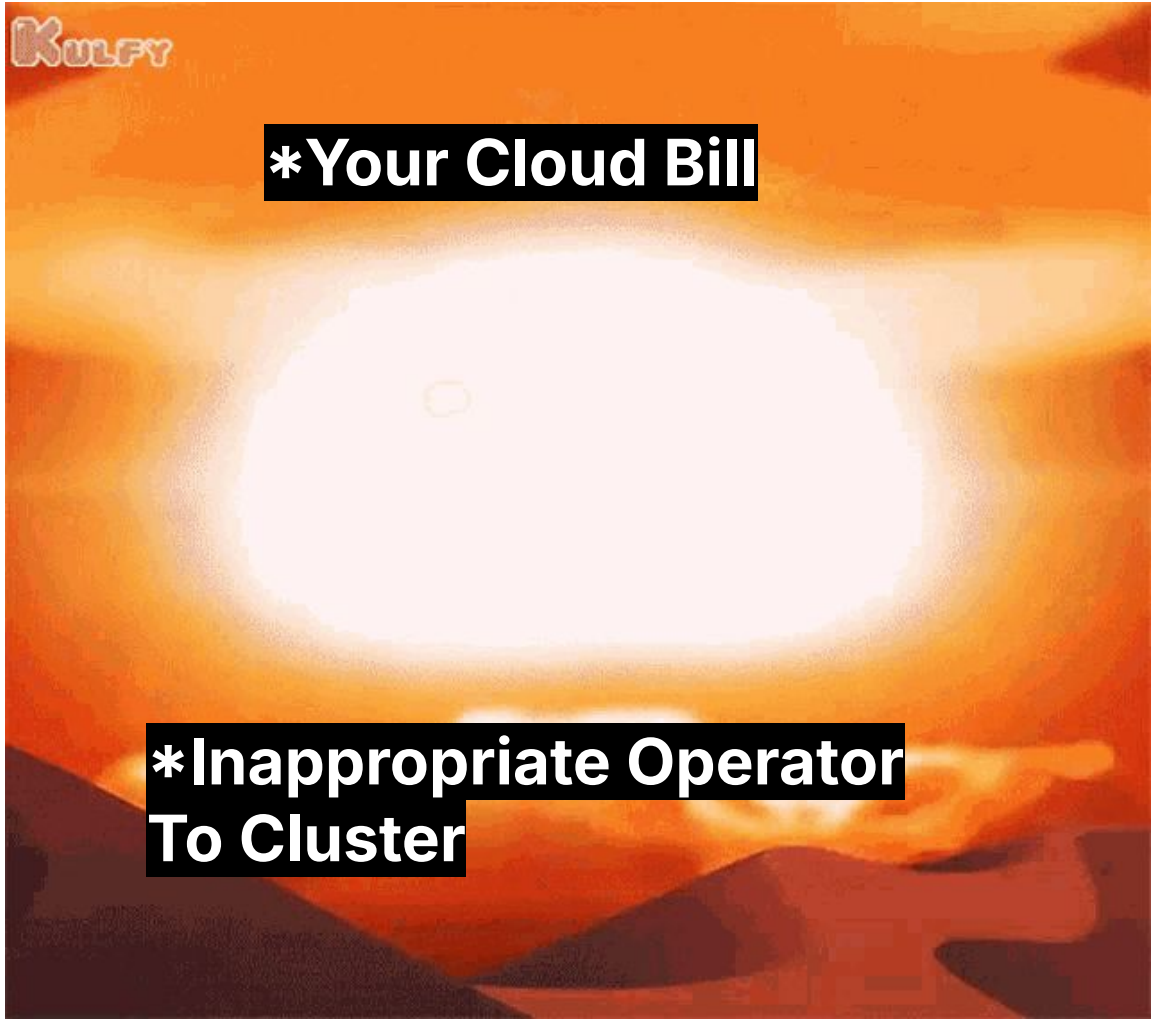
Complexity



Best Practice 3

Managing Resources

Source: https://sdk.operatorframework.io/docs/best-practices/managing-resources/


KULFY
*Your Cloud Bill
*Inappropriate Operator
To Cluster

Best Practice 3 (Managing Resources)

An operator's cost is the cache, not the code.

- **CPU = (Forever) reconcile loops + event processing**
- **Memory = informers + cache (scales with what you watch)**

main cloud-on-k8s / deploy / eck-operator / values.yaml

Code Blame

```
37
38  # priorityClassName defines the PriorityClass to be used by the operator pods.
39  priorityClassName: ""
40
41  # imagePullSecrets defines the secrets to use when pulling the operator container image.
42  imagePullSecrets: []
43
44  # resources define the container resource limits for the operator.
45  resources:
46    limits:
47      cpu: 1
48      memory: 1Gi
49    requests:
50      cpu: 100m
51      memory: 150Mi
52
53  # statefulsetAnnotations define the annotations that should be added to the operator StatefulSet.
54  statefulsetAnnotations: {}
```

*Always Set Requests & Limits

Operator crashes on startup with OOMKilled


```
[
  {
    "containerID": "containerd://...",
    "image": "docker.elastic.co/eck/eck-operator:3.4.0",
    "imageID": "docker.elastic.co/eck/eck-operator@sha256:...",
    "lastState": {
      "terminated": {
        "containerID": "containerd://...",
        "exitCode": 137,
        "finishedAt": "2022-07-04T09:47:02Z",
        "reason": "OOMKilled",
        "startedAt": "2022-07-04T09:46:43Z"
      }
    },
    "name": "manager",
    "ready": false,
    "restartCount": 2,
    "started": false,
    "state": {
      "waiting": {
        "message": "back-off 20s restarting failed container=manager pod=elastic-
          operator-0_elastic-system(57de3efd-57e0-4c1e-8151-72b0ac4d6b14)",
        "reason": "CrashLoopBackOff"
      }
    }
  }
]
```


***You Can Adjust The Requests & Limits To Avoid OOMKilled**

https://www.elastic.co/docs/troubleshoot/deployments/cloud-on-k8s/common-problems

Best Practice 4

Designing Lean Operator

Designing Lean Operators

Press Esc to move out of the editor.

```
// Old way - override the whole NewCache function
NewCache: cache.BuilderWithOptions(cache.Options{
    SelectorsByObject: cache.SelectorsByObject{
        &corev1.Secret{}: {
            Label: labels.SelectorFromSet(labels.Set{"app": "app-name"}),
        },
    },
}),

// New way - just pass cache.Options directly
Cache: cache.Options{
    ByObject: map[client.Object]cache.ByObject{
        &corev1.Secret{}: cache.ByObject{
            Label: labels.SelectorFromSet(labels.Set{"app": "app-name"}),
        },
    },
},
```

https://sdk.operatorframework.io/docs/best-practices/designing-lean-operators/

Best Practice 5

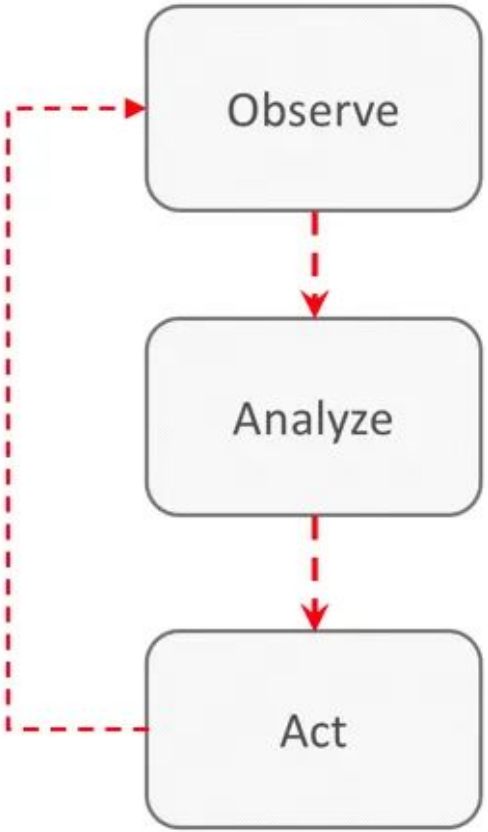
Best Practice 5

Plan Your Reconciliation
(Also, Using Predicate Filter)

Reconciliation Loop

- **Use Deterministic Naming**
- **Always Assume A Stale Cache**
- **The Entire Reconciliation Should Be Idempotent.**

Best Practice 5



Reconciliation Loop

To Avoid Forever Reconciliation Loops, Use Predicate Filter

```go
import (
    "sigs.k8s.io/controller-runtime/pkg/predicate"
    "sigs.k8s.io/controller-runtime/pkg/event"
)

// Predicate to trigger reconciliation only on size changes in the Busybox spec
updatePred := predicate.Funcs{
    // Only allow updates when the spec.size of the Busybox resource changes
    UpdateFunc: func(e event.UpdateEvent) bool {
        oldObj := e.ObjectOld.(*examplecomv1alpha1.Busybox)
        newObj := e.ObjectNew.(*examplecomv1alpha1.Busybox)

        // Trigger reconciliation only if the spec.size field has changed
        return oldObj.Spec.Size != newObj.Spec.Size
    },

    // Allow create events
    CreateFunc: func(e event.CreateEvent) bool {
        return true
    },

    // Allow delete events
    DeleteFunc: func(e event.DeleteEvent) bool {
        return true
    },

    // Allow generic events (e.g., external triggers)
    GenericFunc: func(e event.GenericEvent) bool {
        return true
    },
}
```

https://book.kubebuilder.io/reference/watching-resources/predicates-with-watch

From The Experience!

Operation Ordering

(You don't have to. Here's why you should.)

Best Practice (Operation Ordering)

Eventual Consistency

Best Practice (Operation Ordering)

Why Ordering? It’s Advantage!

Best Practice (Operation Ordering)

Create order

- CRD (wait: Established)
- Namespace / prerequisites
- Workload / Deployment
- Service (expose to cluster)
- ServiceMonitor (after stable)

Best Practice (Operation Ordering)

Tear Down in Reverse!


THIS IS FINE
*SRE
*ServiceMonitors
*Delete The Whole Stack!

Best Practice (Operation Ordering)

Teardown order (reverse):

- Remove ServiceMonitor (Prometheus stops scraping)
- Drain / tear down workload
- Remove service
- Remove namespace / prerequisites
- Remove CRD

Best Practice (SSA)

Server-Side Apply in a Controller

```
kubectl apply --server-side [--dry-run=server]
```

ⓘ **FEATURE STATE:** Kubernetes v1.22 [stable](enabled by default)

https://kubernetes.io/docs/reference/using-api/server-side-apply/

Best Practice (SSA)

What’s The Story Before SSA?

```
14
15  const (
16      // TemplateHashLabelName is a label to annotate a Kubernetes resource
17      // with the hash of its initial template before creation.
18      TemplateHashLabelName = "common.k8s.elastic.co/template-hash"
19  )
20
21  // SetTemplateHashLabel adds a label containing the hash of the given template into the
22  // given labels. This label can then be used for template comparisons.
23  func SetTemplateHashLabel(labels map[string]string, template any) map[string]string {
24      return setHashLabel(TemplateHashLabelName, labels, template)
25  }
26
27  func setHashLabel(labelName string, labels map[string]string, template any) map[string]string {
28      if labels == nil {
29          labels = map[string]string{}
30      }
31      labels[labelName] = HashObject(template)
32      return labels
33  }
34
35  // GetTemplateHashLabel returns the template hash label value if set, or an empty string.
36  func GetTemplateHashLabel(labels map[string]string) string {
37      return labels[TemplateHashLabelName]
38  }
39
40  // HashObject returns a hash of a given object using the 32-bit FNV-1 hash function
41  // and the spew library to print the object (see WriteHashObject).
42  // This is inspired by controller revisions in StatefulSets:
43  // https://github.com/kubernetes/kubernetes/blob/8de1569ddae62e8fab559fe6bd210a5d6100a277/pkg/controller/history/controller_history.go#L89-L101
44  func HashObject(object any) string { //nolint:revive
45      objHash := fnv.New32a()
46      WriteHashObject(objHash, object)
47      return fmt.Sprint(objHash.Sum32())
48  }
49
50  // WriteHashObject writes specified object to hash using the spew library
51  // which follows pointers and prints actual values of the nested objects
52  // ensuring the hash does not change when a pointer changes.
53  // The hash can be used for object comparisons.
54  // Copy of https://github.com/kubernetes/kubernetes/blob/ea0764452222146c47ec826977f49d7001b0ea8c/pkg/util/hash/hash.go#L28
55  func WriteHashObject(hasher hash.Hash, objectToWrite any) {
56      printer := spew.ConfigState{
57          Indent:         " ",
58          SortKeys:       true,
59          DisableMethods: true,
60          SpewKeys:       true,
```

***ECK used the hash annotation for diff objects**

https://github.com/elastic/cloud-on-k8s/blob/aceb66953866a72d60758c985590e846cdfb63bf/pkg/controller/common/hash/hash.go#L18

5835232 cloud-on-k8s / pkg / controller / common / deployment / reconcile.go

Code Blame 96 lines (87 loc) · 3.21 KB · Raw

```go
func New(params Params) appsv1.Deployment {

// Reconcile creates or updates the given deployment for the specified owner.
func Reconcile(
    ctx context.Context,
    k8sClient k8s.Client,
    expected appsv1.Deployment,
    owner client.Object,
) (appsv1.Deployment, error) {
    // label the deployment with a hash of itself
    expected = WithTemplateHash(expected)

    reconciled := &appsv1.Deployment{}
    err := reconciler.ReconcileResource(reconciler.Params{
        Context:    ctx,
        Client:     k8sClient,
        Owner:      owner,
        Expected:   &expected,
        Reconciled: reconciled,
        NeedsUpdate: func() bool {
            return !maps.IsSubset(expected.Labels, reconciled.Labels) ||
                !maps.IsSubset(expected.Annotations, reconciled.Annotations) ||
                // compare hash of the deployment at the time it was built
                hash.GetTemplateHashLabel(reconciled.Labels) != hash.GetTemplateHashLabel(expected.Labels)
        },
        UpdateReconciled: func() {
            // set expected annotations and labels, but don't remove existing ones
            // that may have been defaulted or set by a user/admin on the existing resource
            reconciled.Labels = maps.Merge(reconciled.Labels, expected.Labels)
            reconciled.Annotations = maps.Merge(reconciled.Annotations, expected.Annotations)
            // overwrite the spec but leave the status intact
            reconciled.Spec = expected.Spec
        },
    })
    return *reconciled, err
}

// WithTemplateHash returns a new deployment with a hash of its template to ease comparisons.
func WithTemplateHash(d appsv1.Deployment) appsv1.Deployment {
    dCopy := *d.DeepCopy()
    dCopy.Labels = hash.SetTemplateHashLabel(dCopy.Labels, dCopy.Spec)
    return dCopy
}
```

***Don't Update Every Loop: the Spec Hash Trick**

```
160
161             // retain the resource version to avoid unconditional updates
162             resourceVersion := reconciledMeta.GetResourceVersion()
163             params.UpdateReconciled()
164             // and set the resource version back into the resource to indicate the state we are basing the update off of
165             reconciledMeta.SetResourceVersion(resourceVersion)
166             // also keep the owner references up to date
167             expectedMeta, err := meta.Accessor(params.Expected)
168             if err != nil {
169                 return err
170             }
171             expectedOwners := expectedMeta.GetOwnerReferences()
172             if expectedOwners != nil {
173                 // we can safely assume we have just one reference here given that it was created just above
174                 // but we don't want to replace wholesale in case a user has set an additional reference
175                 k8s.OverrideControllerReference(reconciledMeta, expectedOwners[0])
176             }
177
178             err = params.Client.Update(params.Context, params.Reconciled)
179             if err != nil {
180                 return err
181             }
182             if params.PostUpdate != nil {
183                 params.PostUpdate()
184             }
185             log.Info("Updated resource successfully", resourceVersionKey, params.Reconciled.GetResourceVersion())
186         }
187         return nil
188     }
```

***read-modify-write ResourceVersion**

https://github.com/elastic/cloud-on-k8s/blob/aceb66953866a72d60758c985590e846cdfb63bf/pkg/controller/common/reconciler/reconciler.go#L179

Best Practice (SSA)

Declare. Don't read-modify-write.
Send All of Them!

Best Practice (SSA)

Before: read-modify-write

```
// 1. Read (needed for resourceVersion)
cm := &corev1.ConfigMap{}
c.Get(ctx, key, cm)

// 2. Modify in memory
cm.Data["logging"] = "extensive"

// 3. Write back, may conflict on
//    resourceVersion -> retry loop
c.Update(ctx, cm)
```

Now: server-side apply

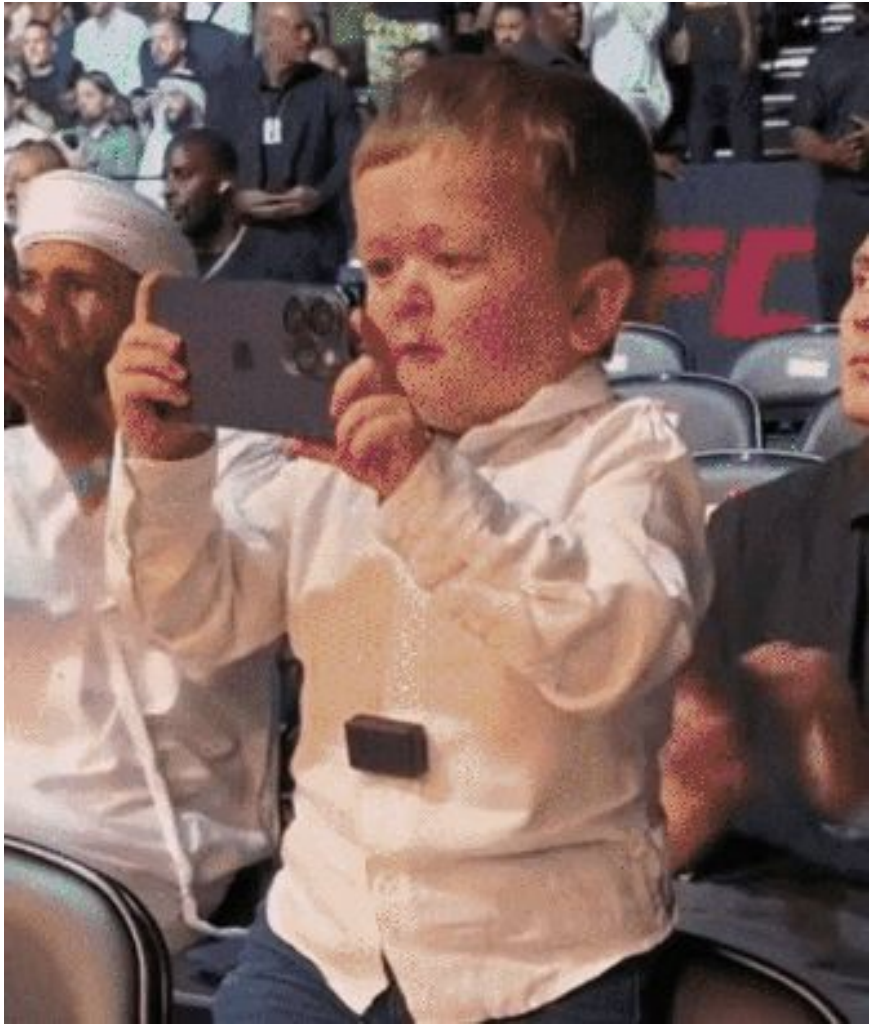
```
// Declare only your fields. No Get.
cm := applycorev1.ConfigMap("app-config", "default").
    WithData(map[string]string{"logging": "extensive"})

c.Patch(ctx, cm, client.Apply,
    client.FieldOwner("my-controller"),
    client.ForceOwnership,
)
```

Best Practice (SSA)

Force Conflicts on Fields You Own.
Only Yours.


*Relax Everyone!


In Conclusion!
*
kubernetes



Those who want to learn more about our Elastic Projects?

(Take out your phones)

Elastic Cloud Serverless

Subscribe

Elastic Cloud Serverless · Ingestion · 20 SEPTEMBER 2024

Architecting the next-generation of Managed Intake Service

APM Server has been the de facto service for ingesting data from Elastic APM agents and OTel agents. In this blog post, we will walk through our journey of redesigning the APM Server product to scale and evolve into a...

Vishal Raj

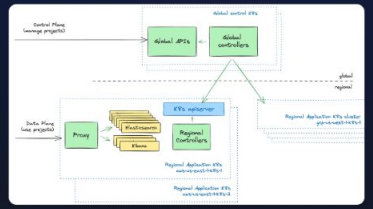
Marc Lopez Rubio

ALL ARTICLES

Elastic Cloud Serverless · 6 SEPTEMBER 2024

Data safety in a stateless world

We discuss the data durability guarantees in stateless including how we fence new writes and deletes with a safety check which prevents stale...



Elastic Cloud Serverless · 3 SEPTEMBER 2024

Leveraging Kubernetes controller patterns to orchestrate Elastic workloads globally

Understand how Kubernetes controller primitives

Elastic Cloud Serverless · 8 AUGUST 2024

Stop project sizing, start search tier autoscaling!

Serverless users no longer need to worry about cluster sizing their search tier thanks to autoscaling.


Let’s Connect Again!
jaa raha hu lekin wapas
jarur aaunga