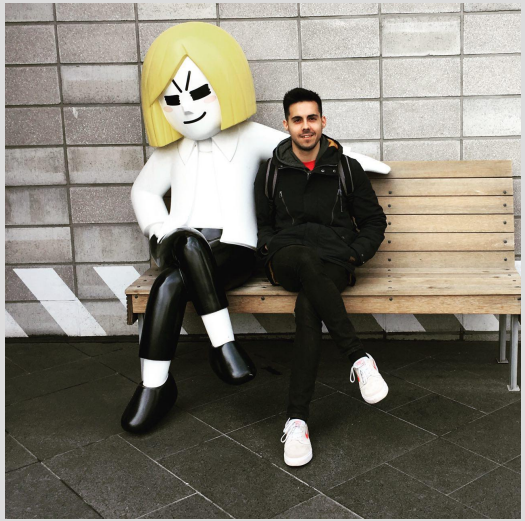
BARCELONA JUG - THE RISING OF KUBERNETES OPERATORS

WHO WE ARE?

```
apiVersion: user.openshift.io/v1
groups: Solutions Engineering
kind: User
metadata:
  name: Mario Vázquez
  email: mavazque@redhat.com
  background:
    - Systems Administrator
    - Consultant
  social:
    - twitter.com/mvazce
    - github.com/mvazquezc/
    - linkedin.com/in/mariovazquezcebrian/
```



AGENDA

1. Why are we here?
2. What is an Operator
3. An Operator in Action
4. Deep Down on K8s Controllers
5. Creating a simple Operator using the Operator Framework SDK
6. Integrating the Operator using the Operator Lifecycle Manager

WHY ARE WE HERE?


OPERATORS
OPERATORS EVERYWHERE
imgflip.com

WHAT IS AN OPERATOR?

- **An Operator is a method of packaging, deploying and managing a Kubernetes application.** A Kubernetes application is an application that is both deployed on Kubernetes and managed using the Kubernetes APIs and kubectl tooling.

- To be able to make the most of Kubernetes, you need a set of cohesives APIs to extend in order to service and manage your applications that run on Kubernetes. **You can think of Operators as the runtime that manages a specific type of application on Kubernetes.**

- **Operators automate actions usually performed manually, reducing the chances for errors and simplifying complexity.** A simple operator could be one that defines how to deploy an application, and an advanced one, will also provide day2 ops automation like backup, recovery, upgrades, etc.

HOLD ON! WHAT ABOUT CONTROLLERS?

- **Controllers take care of routine tasks** to ensure the observed state matches the desired state of the cluster.

- Each controller is **responsible for a particular resource** in the Kubernetes world.

- Operators use the controller pattern, but **not all controllers are Operators**. It's only an Operator if it's got:
  - Controller Pattern
  - API Extension
  - Single-App Focus

Sources: https://bit.ly/2mkuY7q https://bit.ly/2zCRHp6

TO PUT IT SIMPLE...

- Recipe for an operator:

    - Custom Resource Definition (CRD)

      +

    - Custom Controller

      +

    - Domain Specific Knowledge

      =

      OPERATOR

AN OPERATOR IN ACTION

- Before going into the details, let's see how an Operator works.
- We are going to use the ETCD Operator.
- We will deploy an ETCD instance using the Operator.
- We will interact with the ETCD instance.

DEMO TIME

QUESTIONS?

CONTROLLER COMPONENTS

- There are **two main components** of a controller: **Informer/SharedInformer** and **Workerqueue**.

- **Informer**
  - In order to retrieve an object's information, the controller sends a request to Kubernetes API server. However, **querying the API repeatedly can become expensive**.
  - Additionally, the controller doesn't really want to send requests continuously. **It only cares about events** when the object has been created, modified or deleted.
  - Not much used in the current Kubernetes (instead SharedInformer is used)

Source: https://bit.ly/2mkuY7q

CONTROLLER COMPONENTS

- **SharedInformer**
  - The informer creates a **local cache of a set of resources only used by itself**. But, in Kubernetes, there is a bundle of controllers running and caring about multiple kinds of resources.
  - In this case, the SharedInformer helps to create a **single shared cache among controllers**.
- **Workqueue**
  - The SharedInformer can't track what each controller is up to, **the controller must provide its own queuing and retrying mechanism** (if required).
  - Whenever a resource changes, the Resource Event Handler (part of the Lister) puts a key into the Workqueue.

Source: https://bit.ly/2mkuY7q

CONTROLLER PATTERN

- In Kubernetes, a controller is a **control loop that watches the shared state of the cluster** through the API server and makes changes attempting to **move the current state towards the desired state**.

- Examples of controllers:
  - Replication Controller
  - Namespace Controller
  - ServiceAccount Controller
  - Many more

Source: https://bit.ly/2mkuY7q

HOW A CONTROLLER WORKS

- **Control Loop**
  - Every controller has a Control Loop which basically does:
    - Processes every single item from the Queue
    - Picks the item and do whatever it needs to do with that item
    - Pushes the item back to the Queue / Ignores it / Deletes it
    - Updates the status to reflect the new changes
    - Starts over
- **Queue**
  - Stuff is put into the queue
  - Stuff is taken from the queue in the Sync Loop
  - **Queue doesn't store objects**, it stores MetaNamespaceKey
    - Key-Value with namespace for the resource and name for the resource

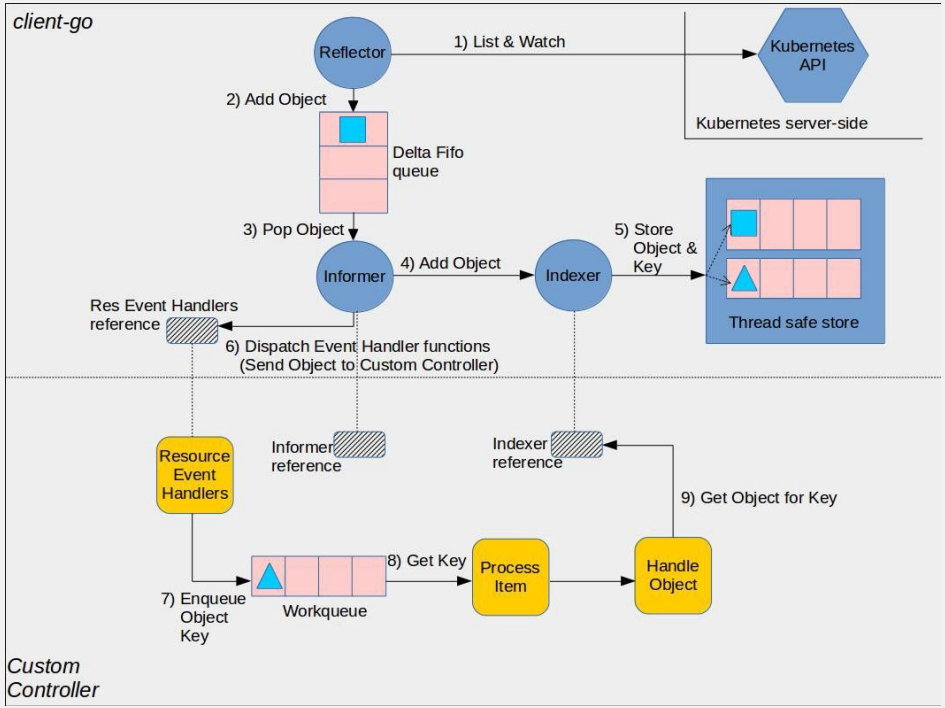
HOW A CONTROLLER WORKS

- **SharedInformer**
  - **Is a shared data cache** and distributes the data to all the Listers that are interested in knowing about changes happening to those data
  - The most important part of the SharedInformer are the **EventHandlers**
    - This is how you register your interest in specific object events (Addition, Updation, Deletion)
  - The Controller will look at what was sent by the EventHandler as it will put objects into the Queue
    - When dealing with updates sometimes the SyncLoop will actually verify if it's needed to process the data
  - **Listers**
    - Important part of the SharedInformers, you want to use them
      - Listers vs ClientGo: Listers are **designed specifically to be used within controllers**, they **have access to the cache** while ClientGo will hit the API Server which is expensive when dealing with thousands of objects

HOW A CONTROLLER WORKS

- **SyncHandler AKA Reconciliation Loop**
  - The first invocation of the SyncHandler will be always **getting the MetaNamespaceKey** to get the Namespace/Resource you want to work with
  - Once the MetaNamespaceKey is ready, **the object is gathered from the cache**
    - Because we are using a shared cache, the resource we're getting is not an object but a pointer to the cached object
    - If you only want to read the object, you're good to go
    - If you want to modify the object then you have to call **DeepCopy** on the object
    - **DeepCopy is an expensive operation**, make sure you will be modifying the object before DeepCopying it
  - Now you will be coding your business logic

HOW A CONTROLLER WORKS

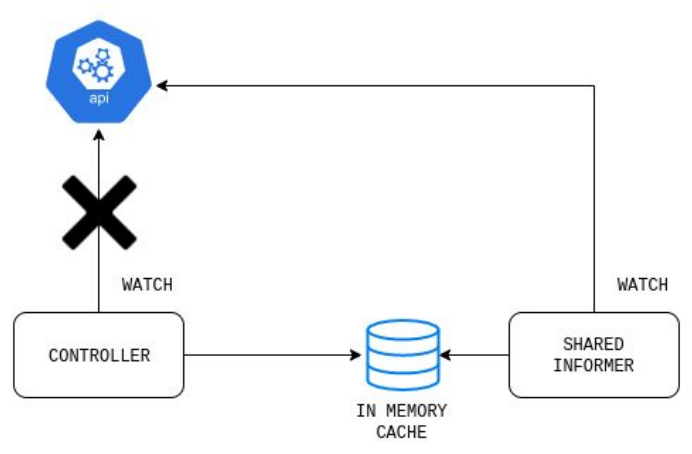


Source: https://bit.ly/2vOiVUT


Kubernetes
Controllers are hard
CHANGE MY MIND
imgflip.com

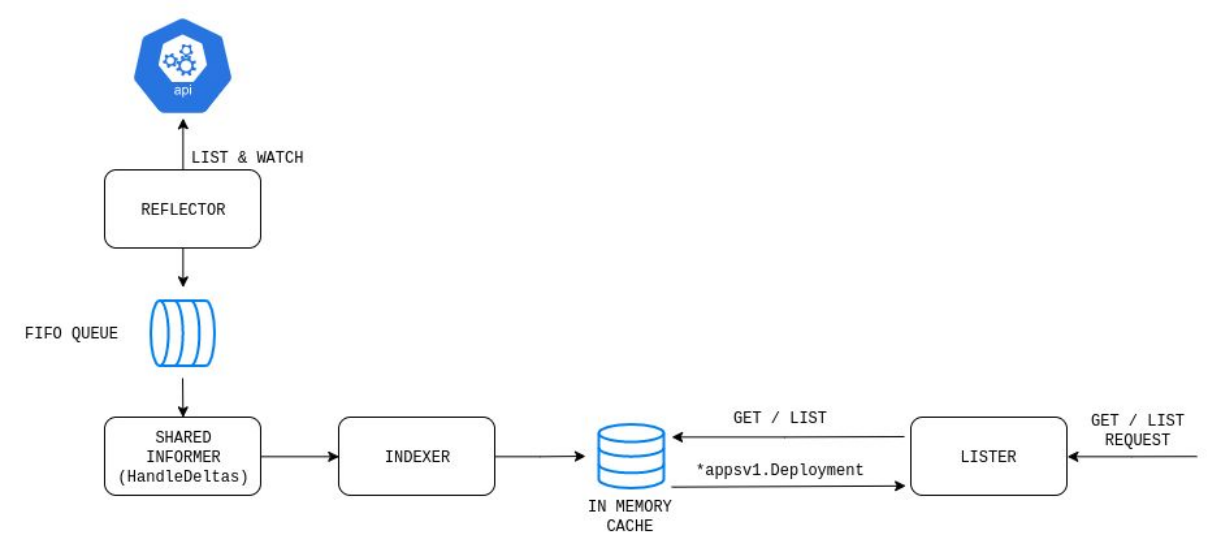
CONTROLLERS UNDER THE HOOD

API vs SharedInformer



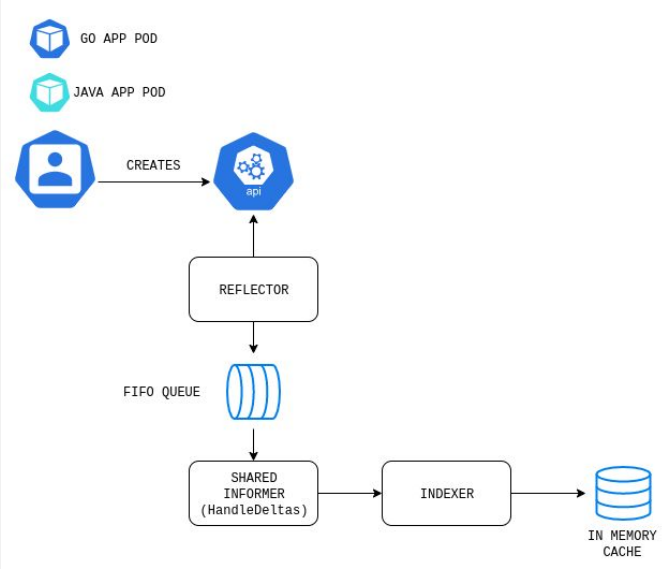
CONTROLLERS UNDER THE HOOD

SharedInformer Detail



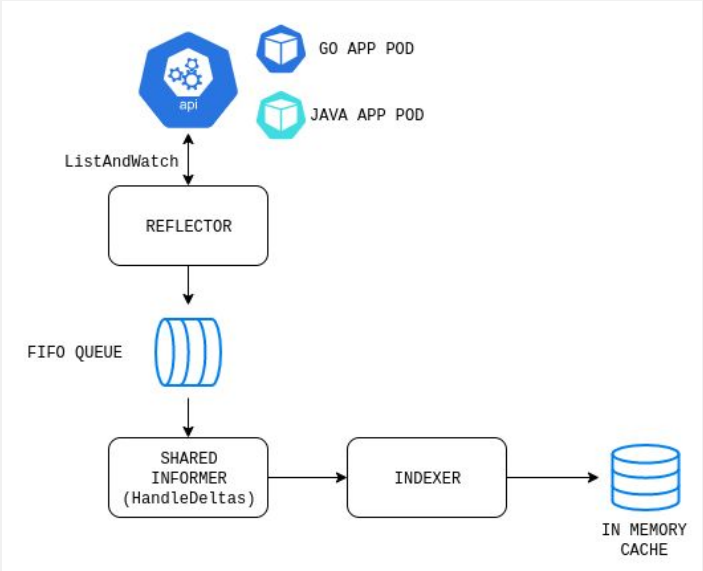
CONTROLLERS UNDER THE HOOD

SharedInformer Cache Flow



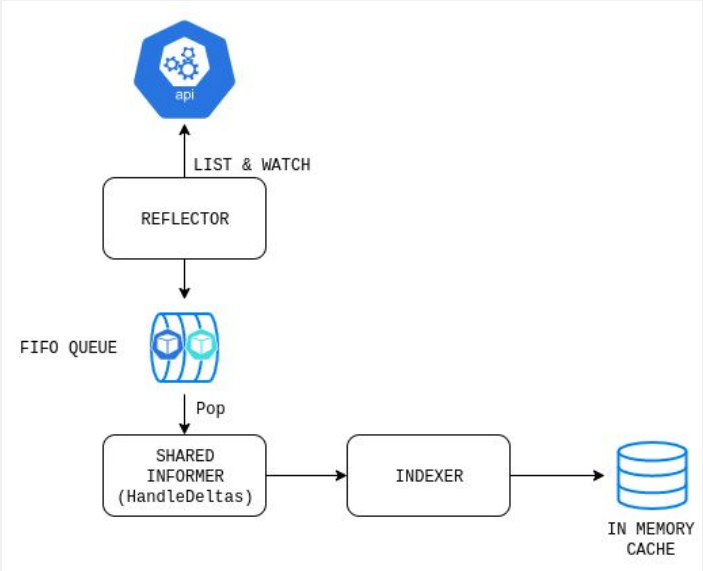
CONTROLLERS UNDER THE HOOD

SharedInformer Cache Flow



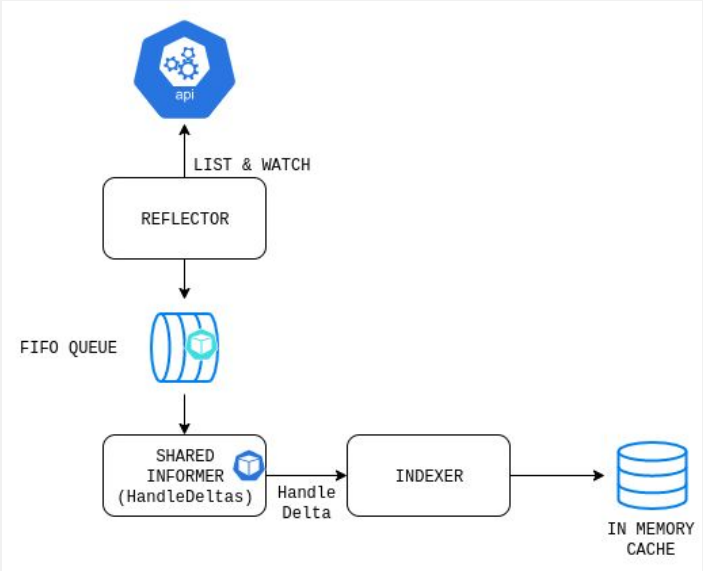
CONTROLLERS UNDER THE HOOD

SharedInformer Cache Flow



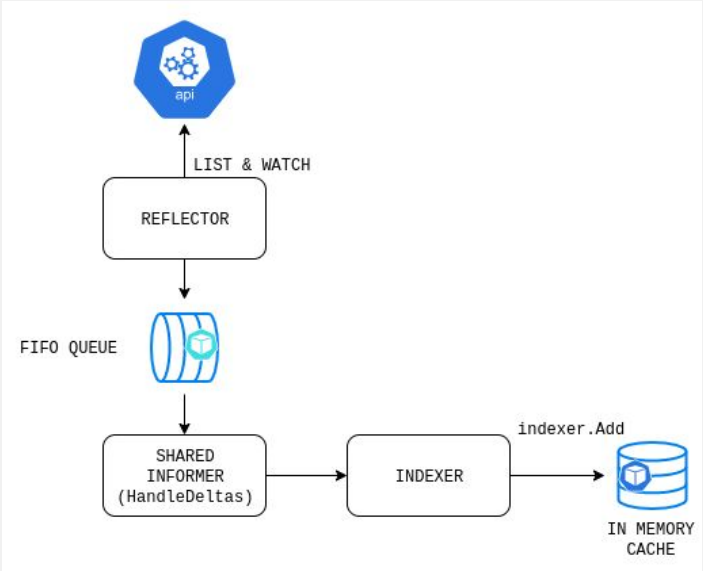
CONTROLLERS UNDER THE HOOD

SharedInformer Cache Flow



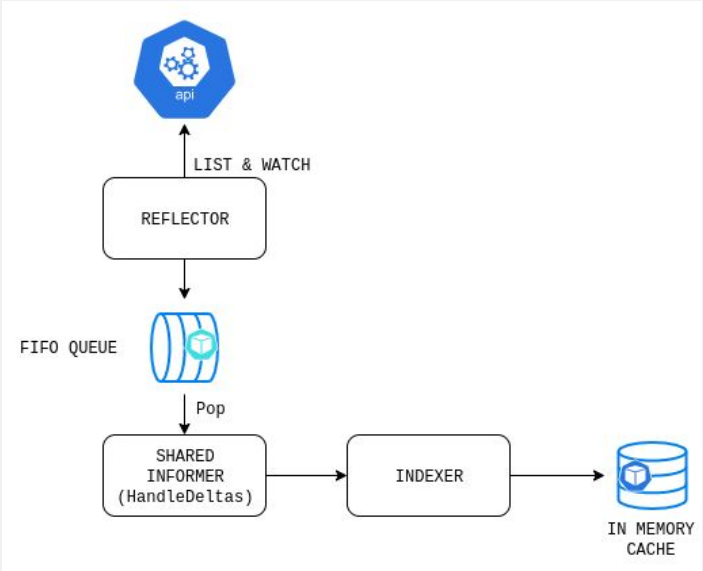
CONTROLLERS UNDER THE HOOD

SharedInformer Cache Flow



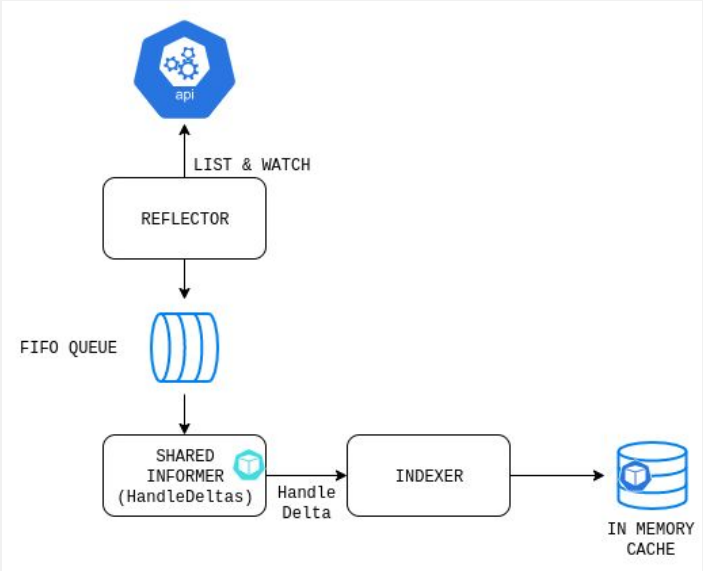
CONTROLLERS UNDER THE HOOD

SharedInformer Cache Flow



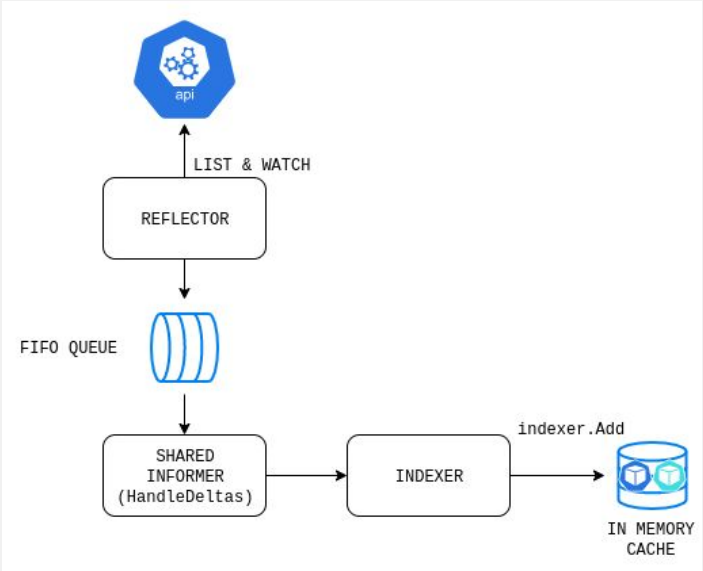
CONTROLLERS UNDER THE HOOD

SharedInformer Cache Flow



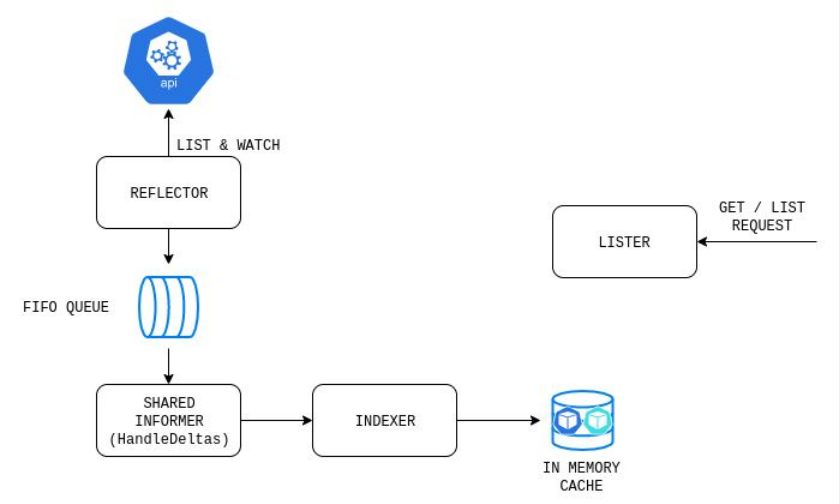
CONTROLLERS UNDER THE HOOD

SharedInformer Cache Flow



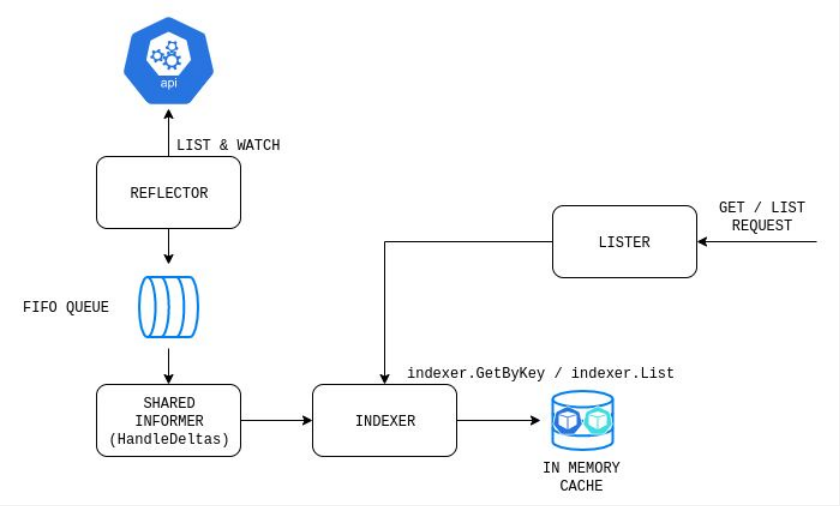
CONTROLLERS UNDER THE HOOD

SharedInformer Cache Flow



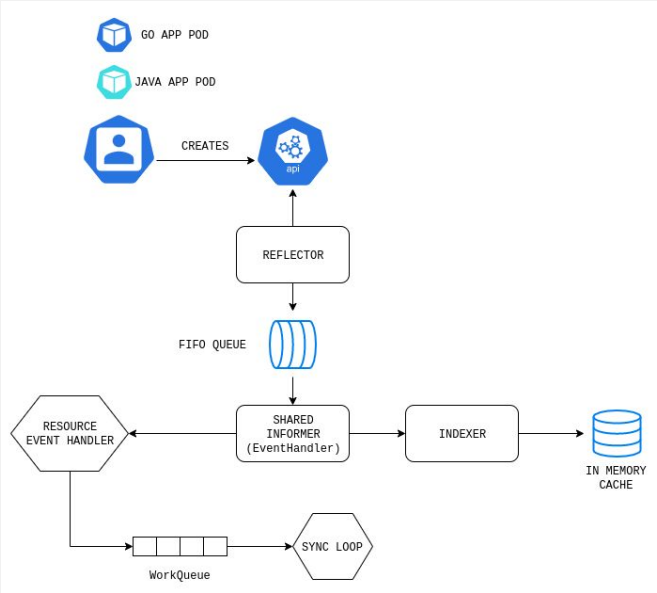
CONTROLLERS UNDER THE HOOD

SharedInformer Cache Flow



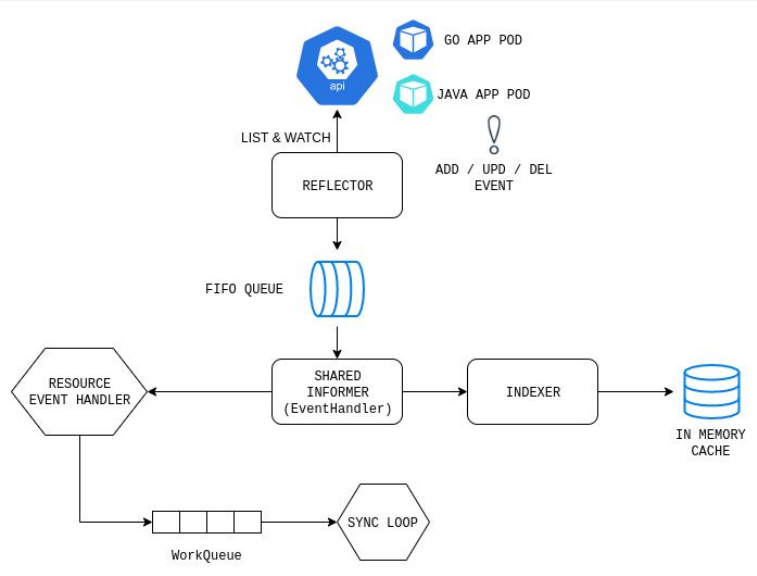
CONTROLLERS UNDER THE HOOD

WorkQueue Flow



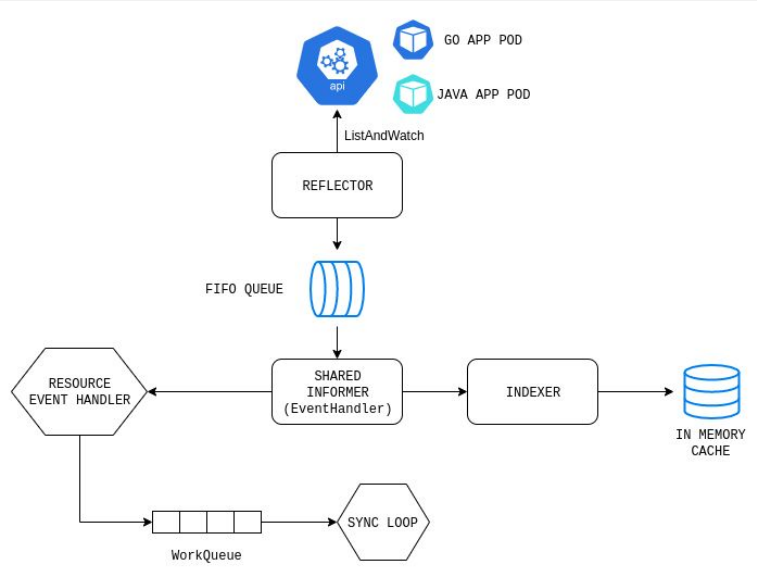
CONTROLLERS UNDER THE HOOD

WorkQueue Flow



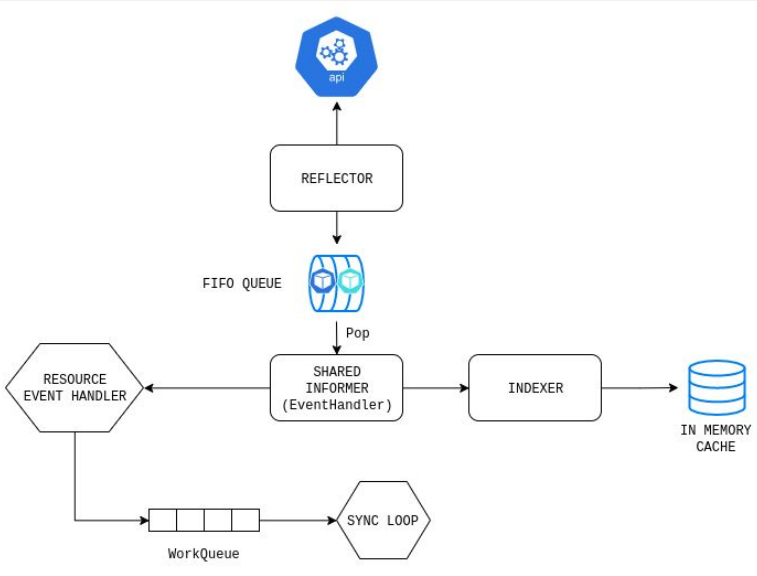
CONTROLLERS UNDER THE HOOD

WorkQueue Flow



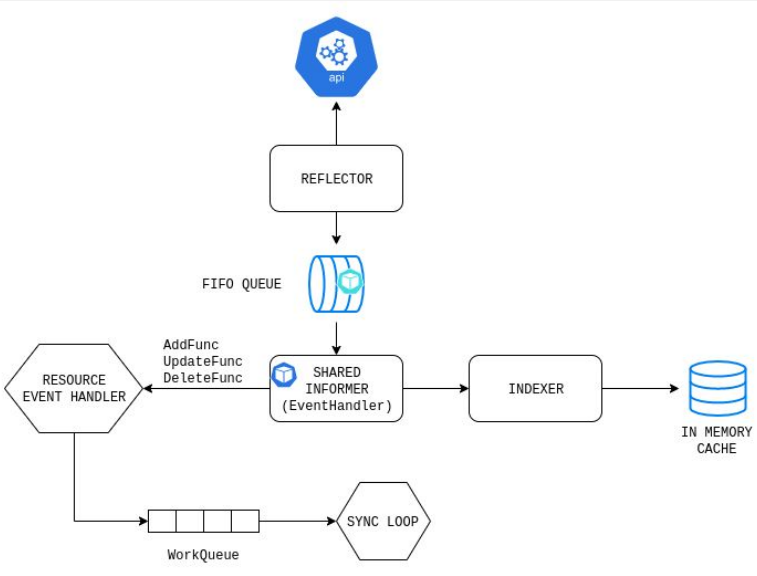
CONTROLLERS UNDER THE HOOD

WorkQueue Flow



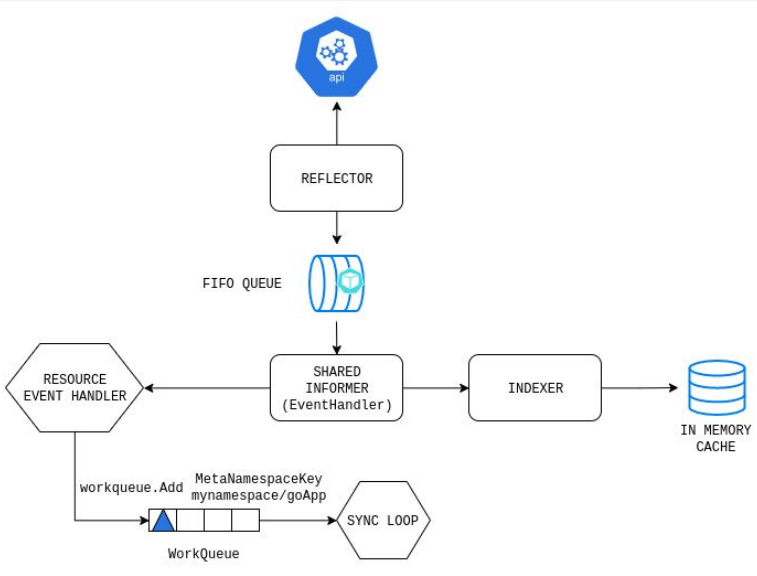
CONTROLLERS UNDER THE HOOD

WorkQueue Flow



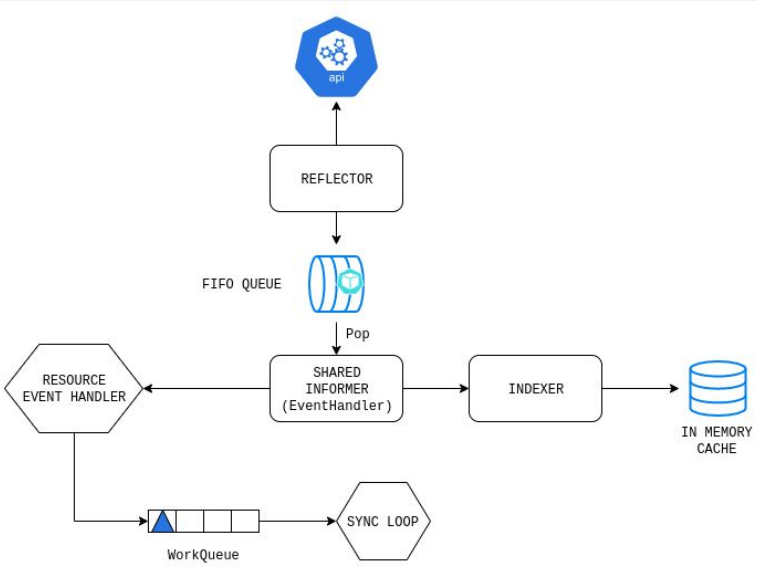
CONTROLLERS UNDER THE HOOD

WorkQueue Flow



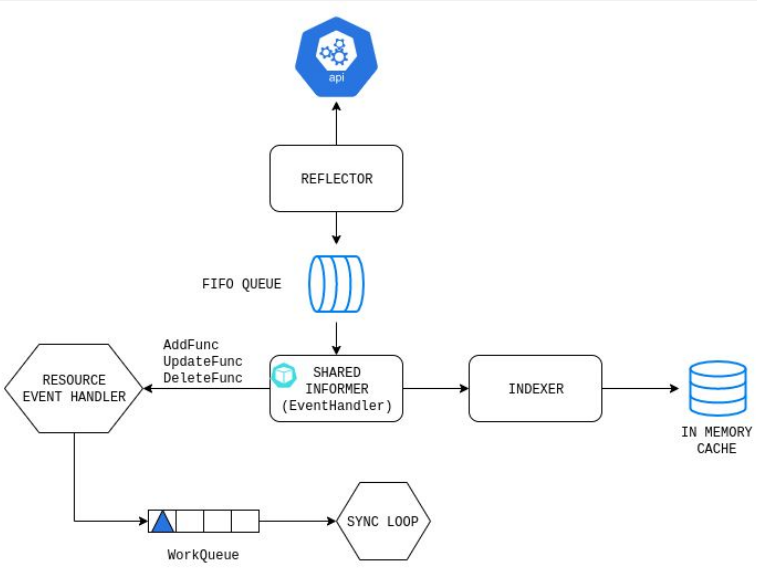
CONTROLLERS UNDER THE HOOD

WorkQueue Flow



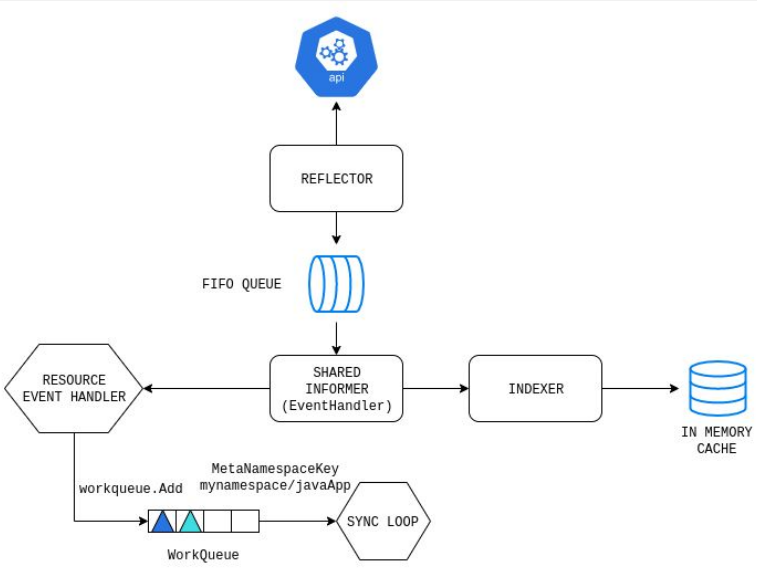
CONTROLLERS UNDER THE HOOD

WorkQueue Flow



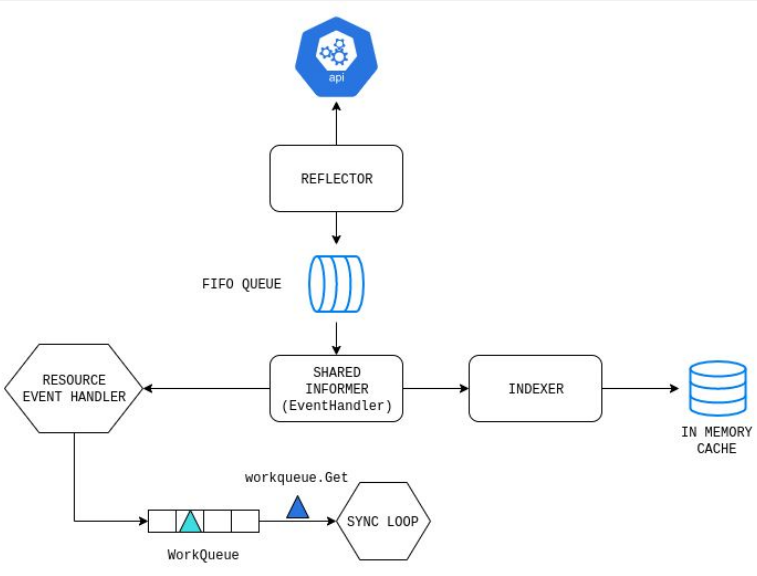
CONTROLLERS UNDER THE HOOD

WorkQueue Flow



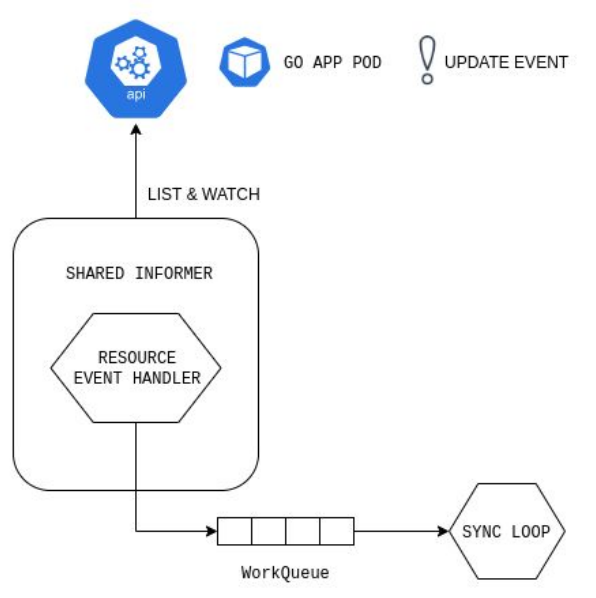
CONTROLLERS UNDER THE HOOD

WorkQueue Flow



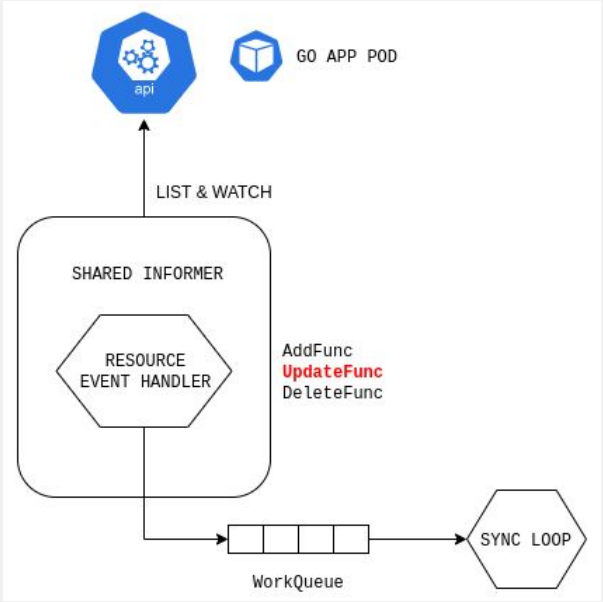
CONTROLLERS UNDER THE HOOD

Controller Cycle



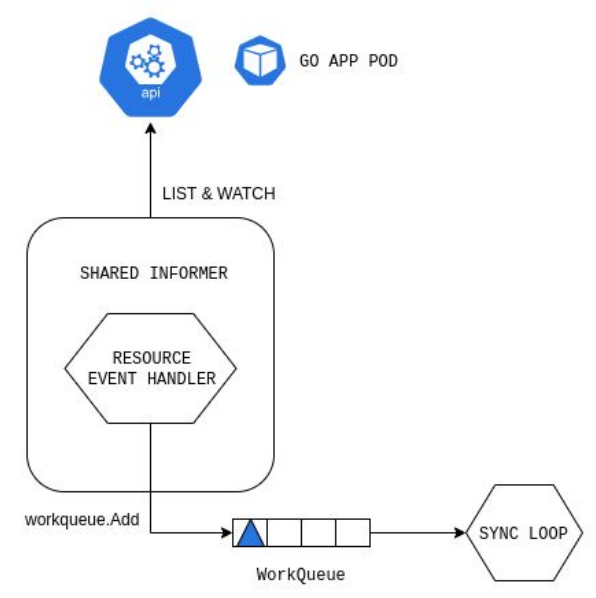
CONTROLLERS UNDER THE HOOD

Controller Cycle



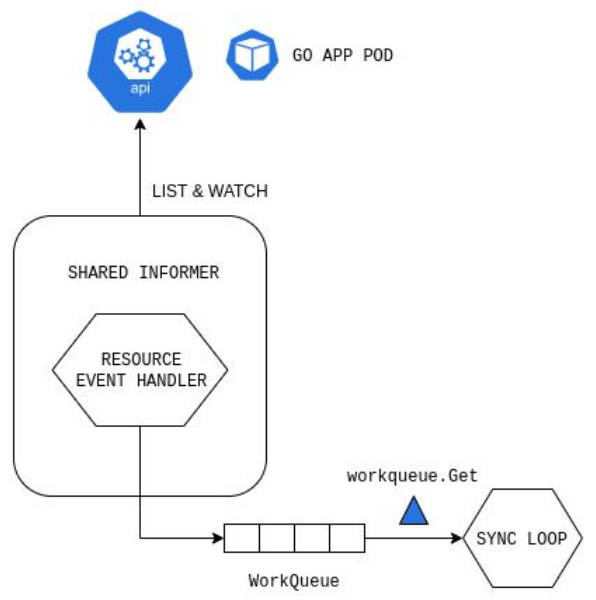
CONTROLLERS UNDER THE HOOD

Controller Cycle



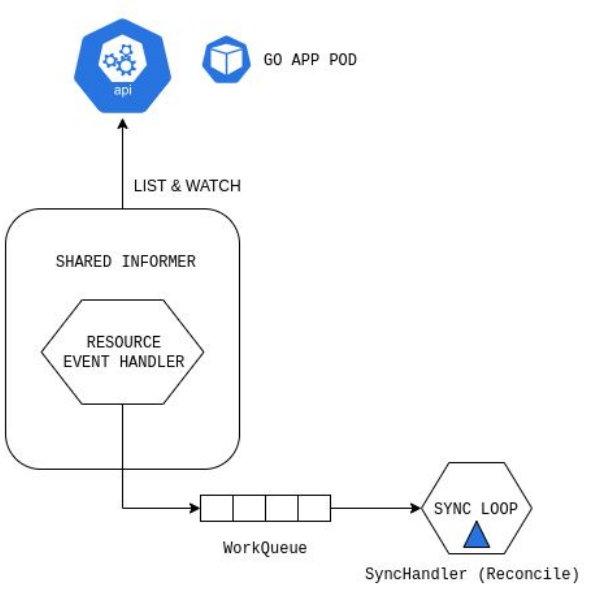
CONTROLLERS UNDER THE HOOD

Controller Cycle



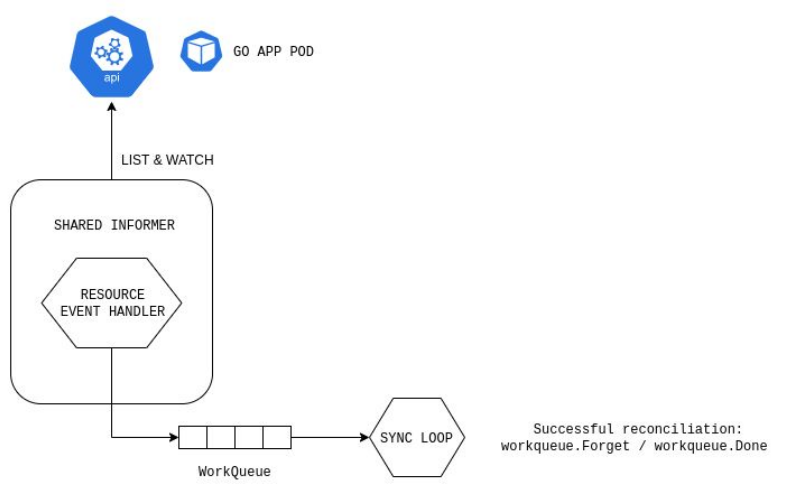
CONTROLLERS UNDER THE HOOD

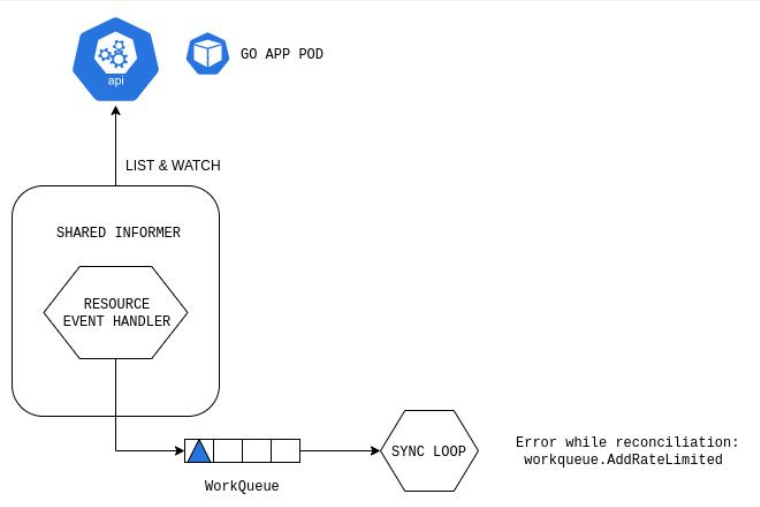
Controller Cycle

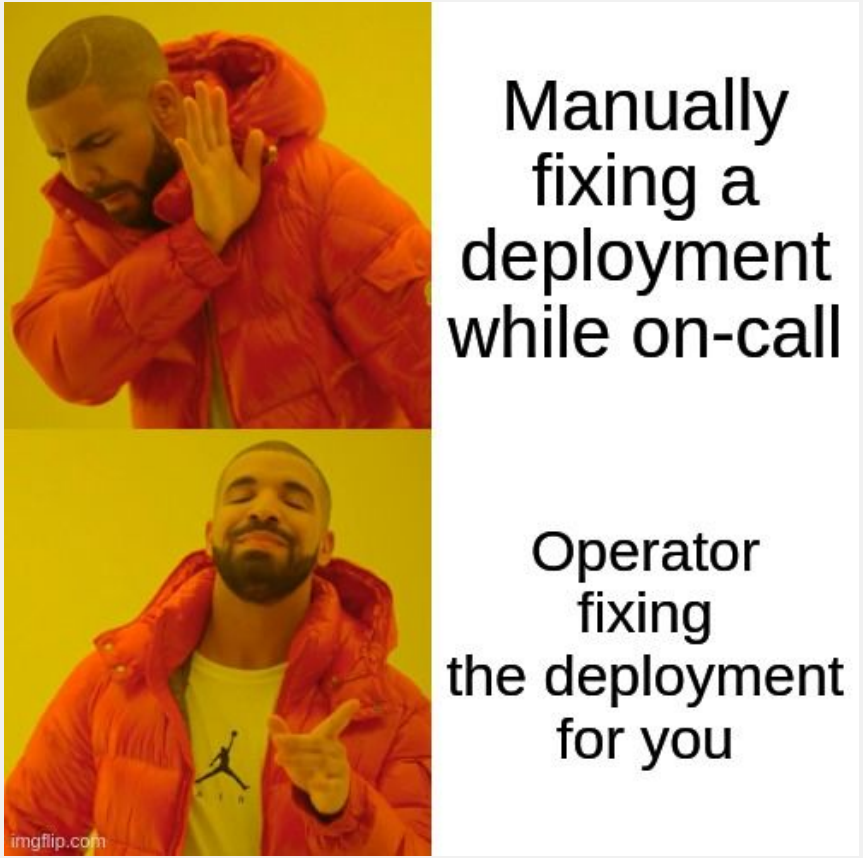


CONTROLLERS UNDER THE HOOD

Controller Cycle



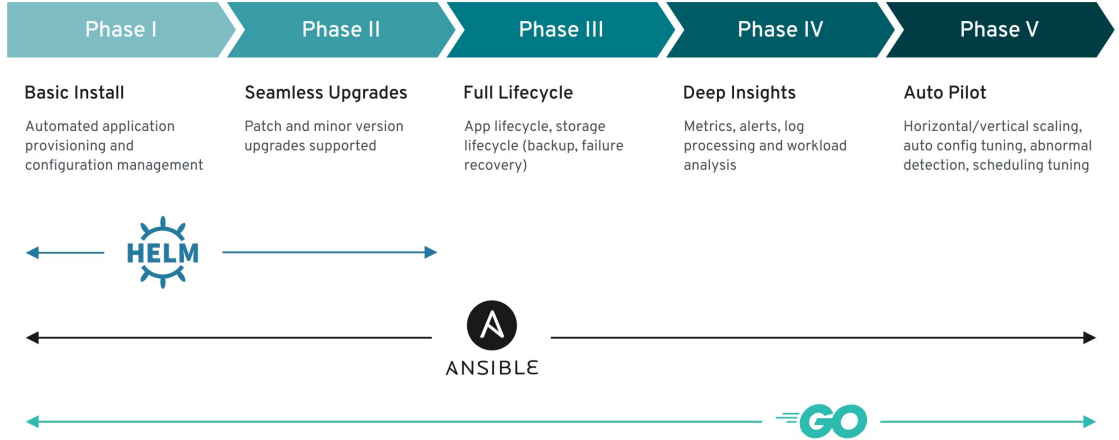



Manually fixing a deployment while on-call
Operator fixing the deployment for you
imgflip.com

BUILDING CONTROLLERS/OPERATORS

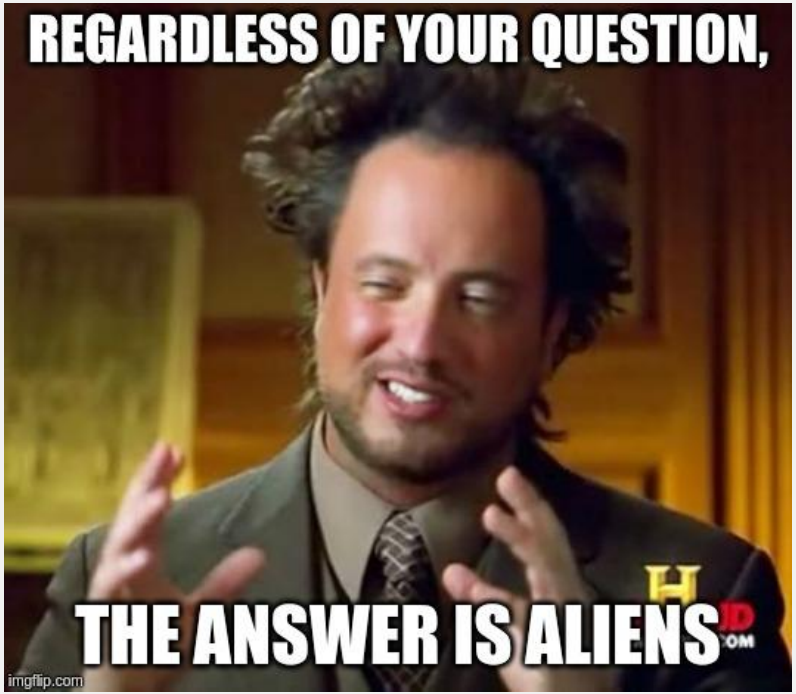
- Runtimes
  - Kubebuilder
  - OperatorSDK
  - Scratch
  - Others: Kudo, Metacontroller, etc.
- Examples
  - Cronjob controller is probably the smallest one out there
  - Sample controller will help you getting started
  - Memcached Operator
  - Writing Kube Controllers for Everyone

OPERATOR SDK MATURITY LEVELS



- You can have the same maturity levels using different frameworks. E.g: Using Java / Quarkus, etc..
- This image only shows maturity levels for the supported technologies on operator-sdk

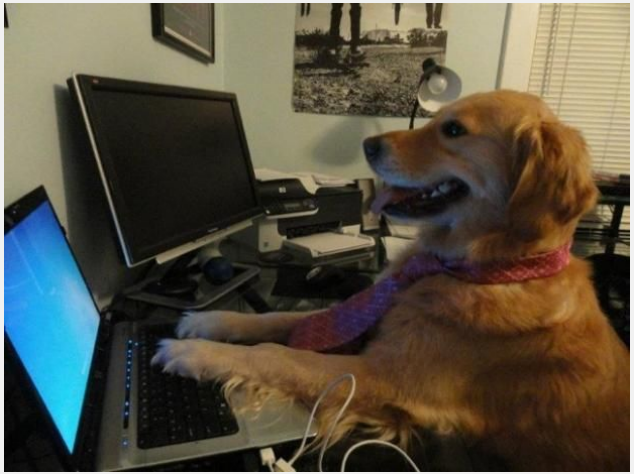
QUESTIONS?


REGARDLESS OF YOUR QUESTION,
THE ANSWER IS ALIENS
imgflip.com

WRITING YOUR VERY FIRST OPERATOR (GO)

- In this demo a very basic Go application will be deployed using our Operator
- The Operator will take advantage of the Operator Framework SDK
- The Operator will be deployed using OLM
- Demo Repository

DEMO TIME



QUESTIONS?

USEFUL RESOURCES

- Writing Operator using the Operator SDK
- Deep Dive Into Kubernetes Controllers I
- Deep Dive Into Kubernetes Controllers II
- Under the Hood of Operator Framework
- Awesome Operators (List of Operators)
- Writing a Custom Controller (CoreOS Fest)
- Kubernetes Custom Resource, Controller, and Operators Development Tools
- Client-go Under The Hood


YOU GET A THANK YOU
YOU GET A THANK YOU
YOU GET A THANK YOU
YOU GET A THANK YOU
EVERYBODY GETS A THANK YOU
imgflip.com