Imperial College London

Kubernetes thoughts

DEEP UNDERGROUND NEUTRINO EXPERIMENT

- The control plane has, broadly:
  - kube-apiserver
    - The way to discuss with the cluster, get its status, user inputs, run control sends stuff here
    - ETCD cluster
      - Smart DB that records the actions that have been taken on the cluster
  - Controller Manager
    - Node-controller: monitors the nodes
    - Replication-controller: makes sure the pods are running
    - much more details probably needed
  - kube-scheduler
    - What assigns Pod to Node, depending on defined policies
      - Filter the nodes one which the pod can be run
      - Scores which node is the best to place a pod on it (configurable)

Imperial College London

Kubernetes thoughts

DEEP UNDERGROUND NEUTRINO EXPERIMENT

- NodeSelector can be used in the POD configuration to run on the correct node for frontend at least.
- Node affinity is a more nuanced way to assign pod to the correct host, but also allows other types of pods to run on the host
- Inter-pod affinity is way to get pods running on the same node, and vice versa (pod anti-affinity: these 2 pods can't run on the same node)
- All affinity can be either soft or hard
- https://www.devopsschool.com/blog/understanding-node-selector-and-node-affinity-in-kubernetes/

Kubernetes thoughts

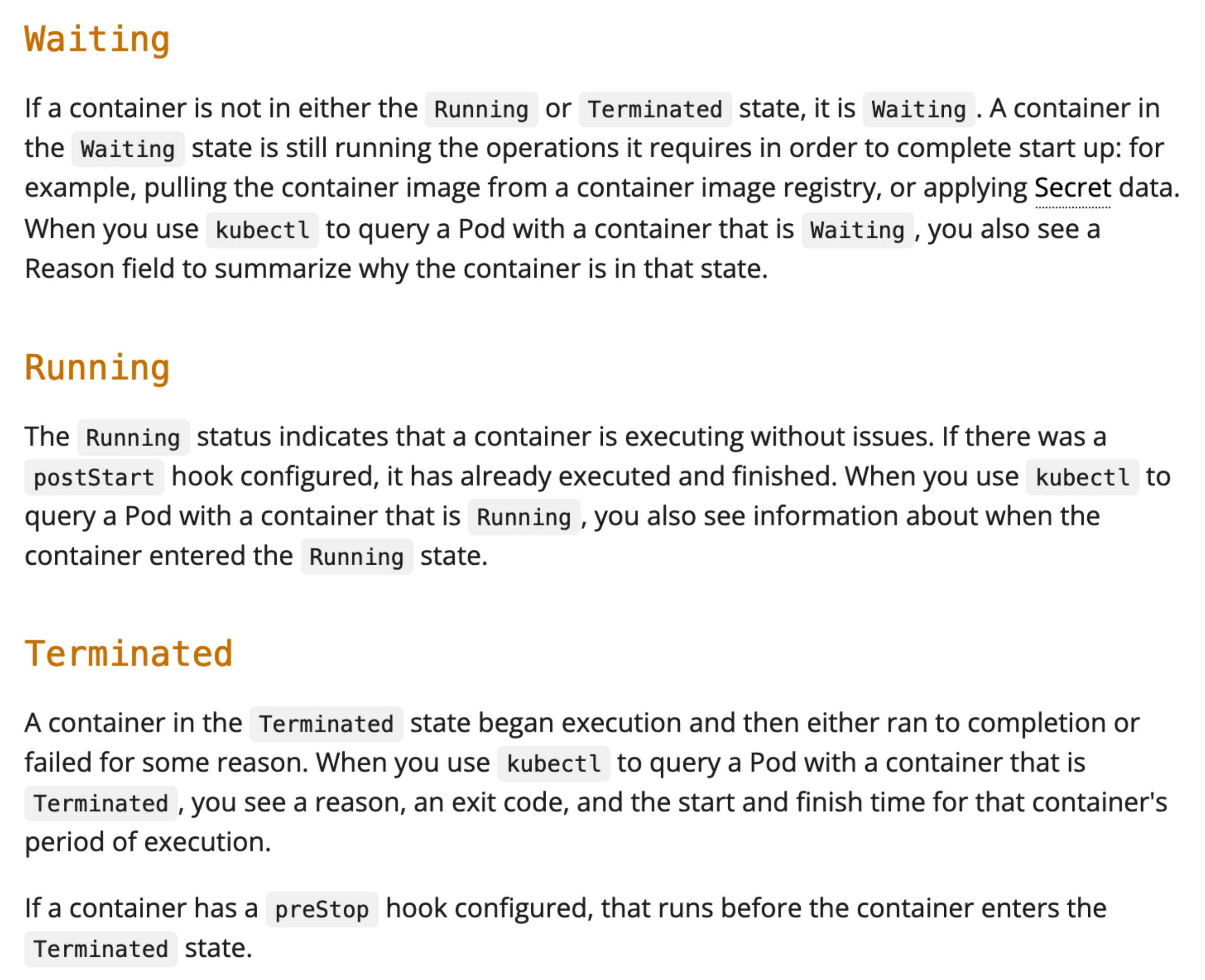
- If I understood things correctly, before deciding if we need our own controllers, we also need to understand if we need custom resources (CRD - custom resource definition).
    - Once we have this, they are accessible from the ETCD cluster, and become "natively" available in Kubernetes control plane
    - Custom *resource*, it means this cannot be a state of the application (I found this example for dummies https://www.tutorialworks.com/kubernetes-custom-resources/ very helpful) it's a more or less a *physical* quantity, for example, an APA (please correct me if I'm wrong)
    - One can then use these custom resources to make our own custom controller
        - "Ensure that we are running with 5 adjacent APAs"… and the controller should have the logic to do that
    - It may also be that we don't need CRDs to make our controllers

Imperial College London

Kubernetes thoughts

DEEP UNDERGROUND NEUTRINO EXPERIMENT

- I'm very confused about FSM, I don't see how we can leverage the k8s' controller reconciliation logic to pass commands all the way down to the processes that run in the pod
  - I'm not even sure if this is what we wanted in the first place (see last slide about controller loop, I still haven't understood it)
  - At first sight, it doesn't really look like this is possible, pod states listed here: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#container-states and I couldn't find if/how we could extend them.
    - Stateful in k8s != FSM
      - Stateful in k8s (I think):
        - Access discs whose data has to be shared across multiple pods (DB with a lot of users as case example)
        - Replicas need to be started in particular order
        - Replicas need to have "persistent" address
    - https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/ needs more reading anyway, and is going to be useful
  - To me, run control commands looks more like a client-server interaction than an admin-server interaction, so this would be *outside* the k8s realm.



`Waiting`

If a container is not in either the `Running` or `Terminated` state, it is `Waiting`. A container in the `Waiting` state is still running the operations it requires in order to complete start up: for example, pulling the container image from a container image registry, or applying Secret data. When you use `kubectl` to query a Pod with a container that is `Waiting`, you also see a Reason field to summarize why the container is in that state.

`Running`

The `Running` status indicates that a container is executing without issues. If there was a `postStart` hook configured, it has already executed and finished. When you use `kubectl` to query a Pod with a container that is `Running`, you also see information about when the container entered the `Running` state.

`Terminated`

A container in the `Terminated` state began execution and then either ran to completion or failed for some reason. When you use `kubectl` to query a Pod with a container that is `Terminated`, you see a reason, an exit code, and the start and finish time for that container's period of execution.

If a container has a `preStop` hook configured, that runs before the container enters the `Terminated` state.

NanoRC

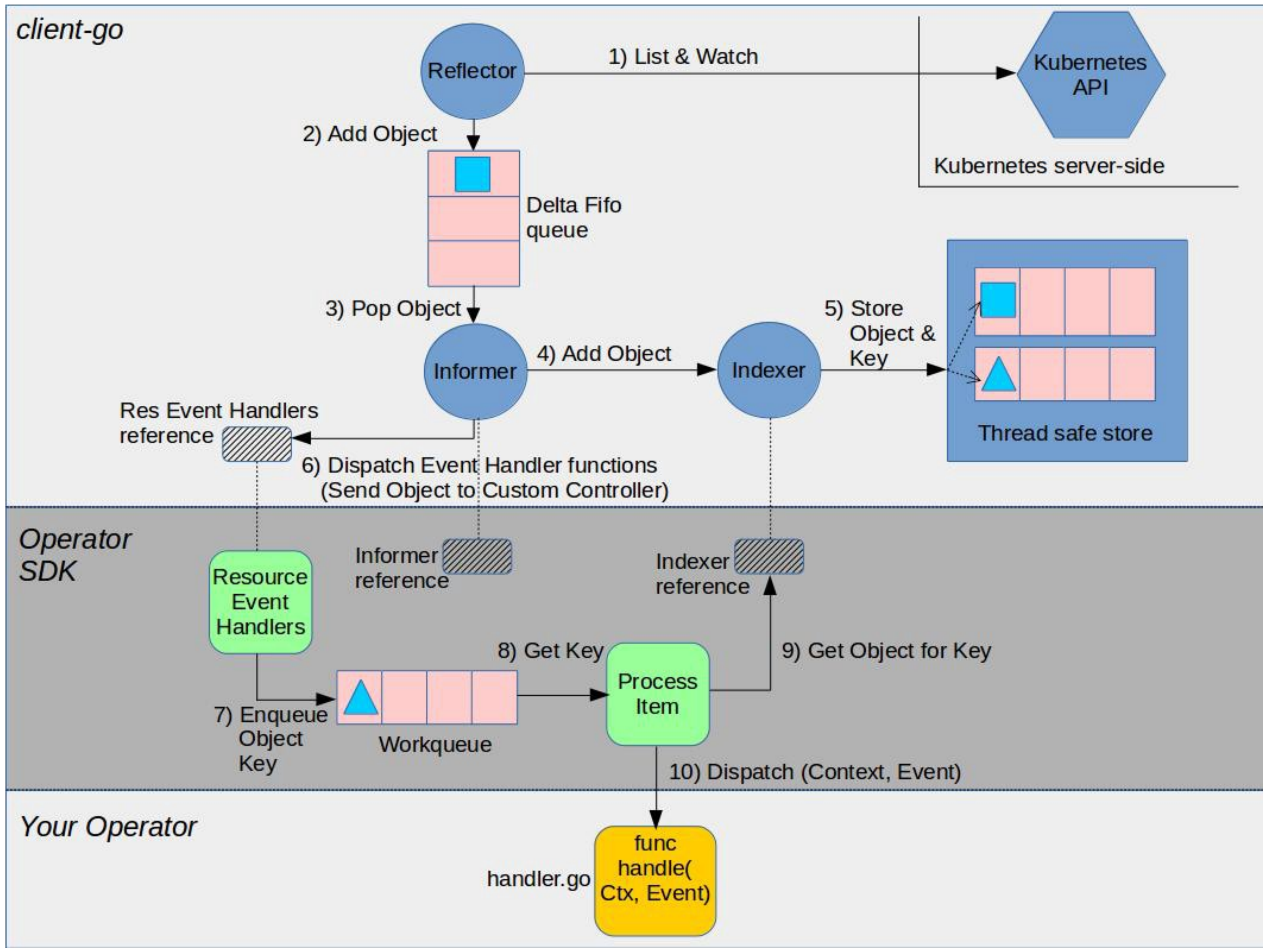
- Put it inside the cluster and make it a service?
  - Sitting on its node and waiting for commands
  - Not sure if services can talk to control plane?
    - Otherwise NanoRC would have to be outside the cluster
    - Need to understand what this means: https://kubernetes.io/docs/concepts/architecture/control-plane-node-communication/
- Right now, NanoRC uses “reverse DNS,” to know where the kind cluster is
  - Loops over all the docker containers and get the network and gateway

Next steps

- Resurrect pocket/nanorc/kubernetes

Controller loop

- Still need to “process” this correctly


client-go
Reflector
1) List & Watch
Kubernetes API
2) Add Object
Delta Fifo queue
Kubernetes server-side
3) Pop Object
Informer
4) Add Object
Indexer
5) Store Object & Key
Thread safe store
Res Event Handlers reference
6) Dispatch Event Handler functions (Send Object to Custom Controller)
Operator SDK
Resource Event Handlers
Informer reference
Indexer reference
8) Get Key
Process Item
9) Get Object for Key
7) Enqueue Object Key
Workqueue
10) Dispatch (Context, Event)
Your Operator
handler.go
func handle(Ctx, Event)