Kubeflow Operator(s) for Kubernetes and OpenShift

Authors: Animesh Singh, Weiqiang Zhuang, Tommy Li
Contributors: Jeremy Lewi, Vaclav Pavlin

Date: Dec 10th, 2019
Short Link:
http://bit.ly/kubeflow-operator

Current Design: TLDR

- Operators help deploy, monitor and manage the lifecycle of applications deployed on Kubernetes. And OpenShift.
- Build a Go based Operator, built on KFDef CRM, KFCtl as the nucleus for Controller
- Handle customization primarily through Kustomize Overlays, and use plugins if advanced customizations are desired
- For Kubeflow, the reconcile function is the "apply" function. kfctl and the KF operator would share this function so that we only have to implement changes once, for example if we want to add the "control-plane" label to the KF namespace to prevent webhooks from being applied to the KF namespace it would be nice if we only had to update the apply function.
- Going with this design might also negate the need for individual Operators for Kubeflow components, as we can have specialized kfdef for individual components if needed, for e.g. KFServing, Kubeflow Pipelines etc.

Code structure:

1. /deploy: Contains all the k8s resources for deploying the operator image and crd.
2. /build: Operator image build script
3. /pkg/controller: main package for operator controller logic
4. /cmd/manager: main.go file for the operator go program.

- Pros:
    - CLI and plugin builds are still separated from the operator build since they are based on different main.go files under /cmd
    - The main package changes for the operator are under /pkg/controller.
    - Other KubeFlow Project such as Katib and KFServing have all the package code under /pkg and separate the builds for the operator/controller and different microservices/sidecars based on the different main.go under /cmd
- Cons
    - Size of the CLI may increase 0.4 - 0.5 MB due to the extra operator source code.
    - No pkg isolation between the different go programs.

Reconciler Design Flow:

1. Limit reconcile concurrent run to 1 to have singleton and avoid race condition. (we can increase the number of concurrent runs once the framework is more mature)
2. Add watcher for every possible resource deletion on the cluster. (done)
    a. Once deletion event is detected, check is the manage-by has the kfctl label (done)
    b. If true, proceed with reconcile and run kfapp apply. (done)
3. If kfdef updated, check deployment status for completion. If the new kfdef generation is not yet starting to deploy, run the kfapp apply. Also add logic for kfdef deletion. (done)

List of issues:

1. Kfctl delete is only deleting the namespace. We need to enhance the delete function to be aware of what is deployed and do a further clean up with all the cluster-wise resources and Knative/Istio.
2. Currently we are limiting the number of kfdef on each namespace using ResourceQuota on Kubernetes 1.15+ since KubeFlow doesn't support multiple instances yet.
3. Kfdef on master branch is missing the kustomize secretGenerator due to issue #156. Therefore please use the kfdef in branch-v0.7 for end to end testing. https://github.com/kubeflow/kfctl/pull/187

Overview:

Operators help deploy, monitor and manage the lifecycle of applications deployed on Kubernetes. And OpenShift. Especially from OpenShift Container Platform 4.1 and onward, with the Operator Lifecycle Manager being installed by default, it just takes several clicks to install an operator and managed applications. The Operator Framework offers an open source toolkit to build, test, package operators and

manage the lifecycle of operators. It includes the following tools:

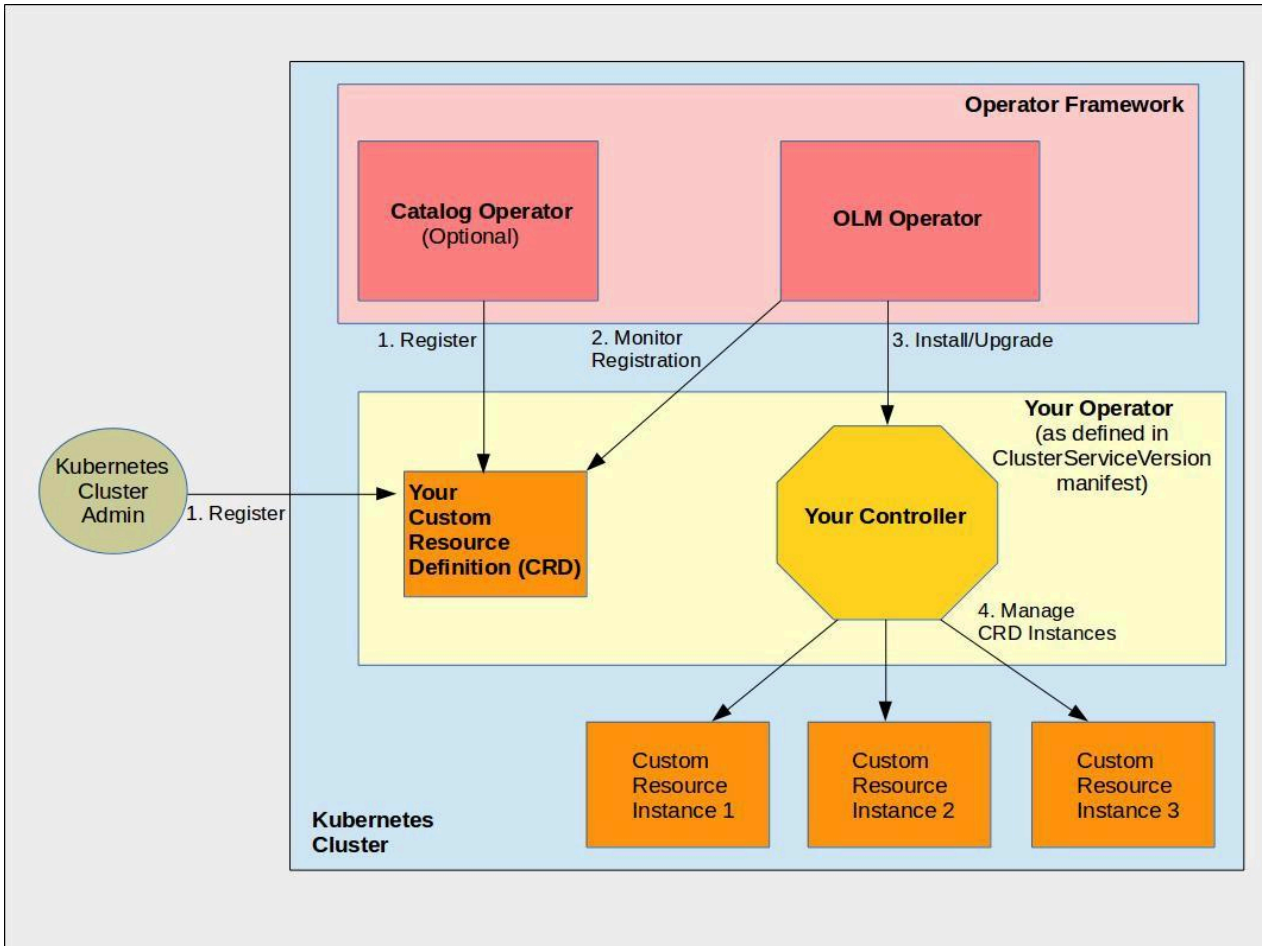
- operator-sdk — write, test, and package operators
- operator-courier — build, verify, and push operator manifests (CustomResourceDefinitions and ClusterServiceVersions)
- operator-registry — store the manifest data in database and provide operator catalog data to Operator Lifecycle Manager
- operator-lifecycle-manager — use installation, upgrade, and role-based access control control operators (the “operator of operators”)
- operator-metering — collect operational metrics of operators for “day 2” management
- operator-mark etplace — register off-cluster operators
- community-operators — host community-created operators and publish to operatorhub.io

To leverage the benefits of frameworks and tooling around Operators, we have created a Kubeflow operator to deploy the Kubeflow framework on OpenShift Container Platform and manage the lifecycle of deployed Kubeflow components. The Kubeflow op erator will target a certain set of components within a `kfdef` manifest. The operator is being designed to achieve the following goals:

- The operator will be flexible enough to allow the customization of which components to be installed.
- The operator can limit the RBAC permissions to maintain the security on OpenShift Container Platform, or work with specific container runtime (e.g. containerd on IBM Cloud)
- The operator will provide multi-tenancy support.
- The operator will provide easy upgrade and update of Kubeflow.

Further, the same Kubeflow operator is expected to work on Kubernetes clusters. Kubeflow operator will detect the platform (OpenShift or Kubernetes..) and perform the necessary operations.

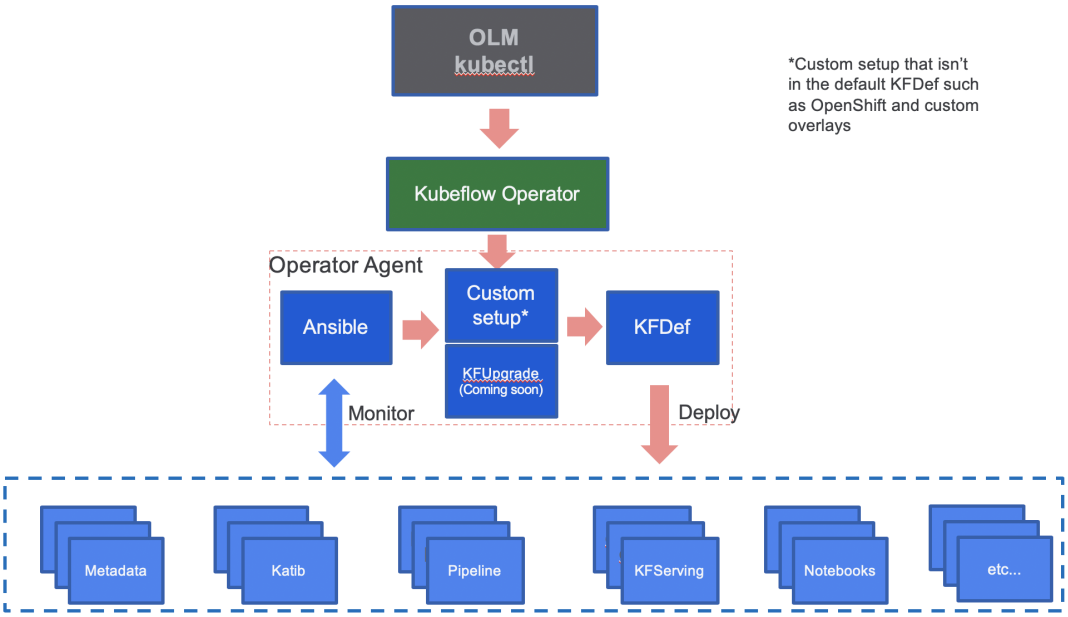
The Kubeflow operator will be installed either manually with `kubectl` or `oc` CLI commands, or through Operator Lifecycle Manager (OLM). OLM helps install, manage and upgrade operators.



(https://miro.medium.com/max/2116/1*GYLAUB7KGCysjPgwek-pPA.jpeg)

Prior Design Discussions

Initial Implementation: xxx Operator



The current release of Kubeflow is 0.7.0. Deploying Kubeflow on existing Kubernetes clusters, including OpenShift Container Platform, can be done by `kfctl` CLI tool on a predefined `kfdef` manifest. There are two example manifests currently provided by community and vendor. The k8s_istio contains all Kubeflow core components, without authentication. The existing_arrikto uses Dex for authentication and also contains all Kubeflow core components.

Each component of Kubeflow includes a base directory with manifests for custom resource definitions, deployments, services etc. Overlays can be added to each component to customize further with `kustomize` tool. Deploying Kubeflow on Kubernetes clusters is simply running one line command `kfctl apply -f ${CONFIG_FILE} -V`, where `kfctl` takes care of creating all K8S resources.

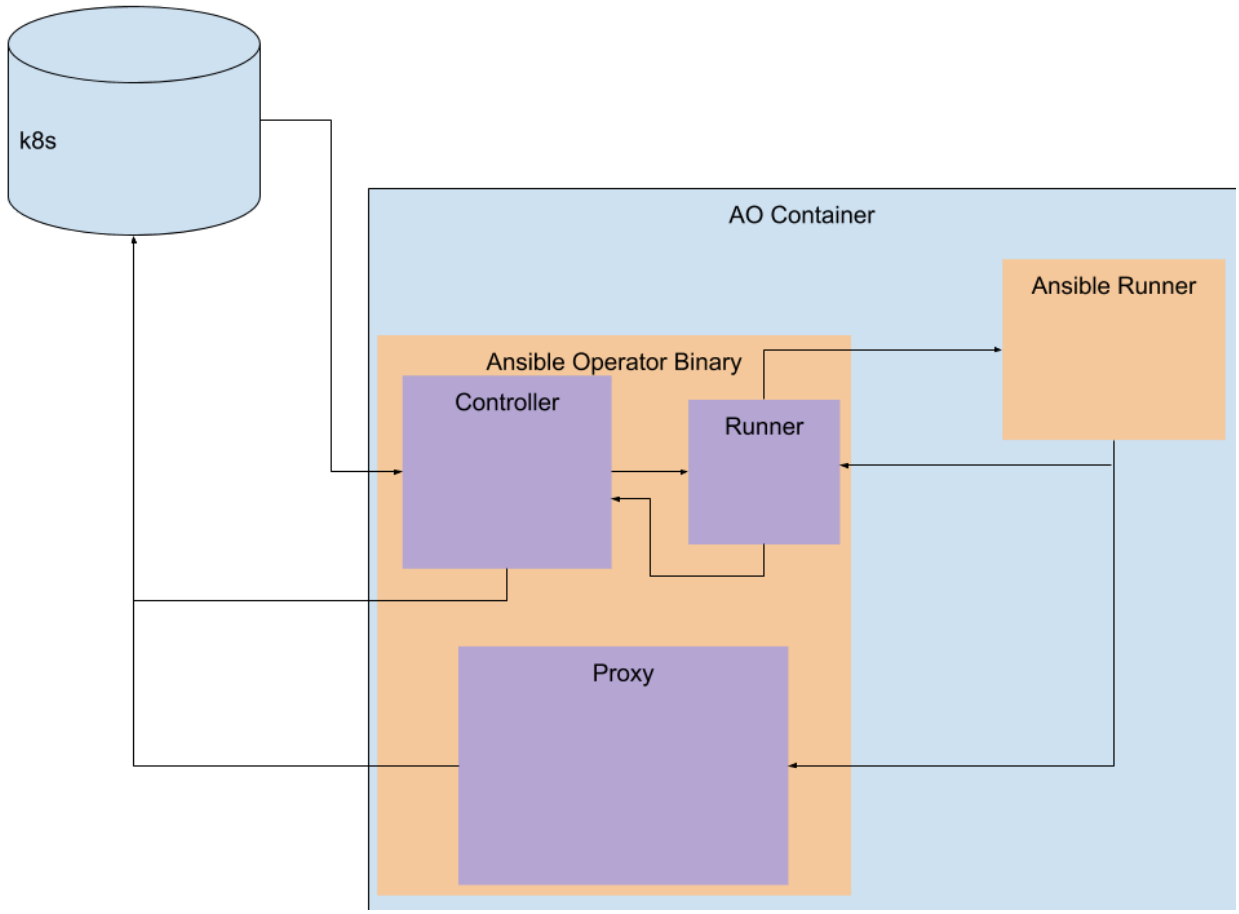
On specific clouds, for e.g IBM Cloud, Additionally, there are specific customizations needed because of non Docker container runtime, which is containerd. Similarly, on OpenShift Container Platform, or its open source version OKD, due to the strict user privileges and RBAC control, more preparation work is needed before running the `kfctl` command. Some of these 'extra' tasks may be common to other K8S flavors and can be contributed back to Kubeflow. But packaging those are specific to OpenShift and making them transparent to end users would be greatly desired.

On the other hand, although the installation instructions are simple enough, especially on Kubernetes clusters, users still have to download the desired version of `kfctl` tool and the `kfdef` manifest. These,

the common installation and configuration tasks, however, can be further automated by an operator which packages the operational knowledge together with the Kubeflow itself.

Furthermore, the operator acting as the controller of the Kubeflow can “watch” the Kubernetes resources related to Kubeflow and perform further tasks such as update, upgrade, and so on. Last but not the least, an operator can be shared among communities when it is published to operatorhub.io for end users to try and deploy in a well defined and easy way, especially on OpenShift Container Platform 4.1 and beyond.

On top of these speculations, we consider building a Kubeflow operator as a necessary and important extension to the Kubeflow project to expand the adoption from users running OpenShift Container Platform. The Kubeflow operator builds with operator SDK toolkit provided by Operator Framework project. Operator SDK can build three types of operators, Go, Helm and Ansible, as outlined in the OpenShift Container Platform document. An Ansible type of operator workflow looks as follows.



Operators written in Golang sets the bar somewhat high for anyone who wants to write an operator — someone has to know a relatively low-level programming language to get started. On top of this, you must also be familiar with Kubernetes internals, such as the API and how events are generated for resources.

The Ansible Operator was created to address this short-coming. The Ansible Operator consists of two main pieces:

1. A small chunk of Golang code, which handles the interface between Kubernetes/OpenShift and the operator.
2. A container, which receives events from the above code and runs Ansible Playbooks as required.

That's it! The Ansible and Operator SDK abstract away all of the difficult parts of writing an operator and allows you to focus on what matters — managing your applications. If you already have a large base of Ansible knowledge in your organization, you can immediately begin managing applications using Ansible Operator. A further added bonus of using Ansible for your operators is that you immediately have access to any module that Ansible can run. This allows folks to incorporate off-cluster management tasks related to your application. For example:

1. Creating DNS entries for your newly deployed applications
2. Spinning up resources external to your cluster, such as storage or networking
3. More easily do off-site backups to external cloud services
4. Manage external load balancing based on custom metrics

There are a number of possibilities that Kubernetes Operators written with Ansible can provide a potential solution for.

Below are a few of the implementation details about the work done to get this deployment going OpenShift 3.1.1 .Please note that many of these will be tackled by KFDef for OpenShift 4.X which RedHat is working on, and with PRs being driven on top of Kubeflow.

RBAC handling

In order to make KubeFlow functional without changing its source code, we need to relax some permissions on OpenShift such as giving access to the Finalizer api (Hook API for CRD controllers and it’s fixed in upstream master) and allow it to run as root.

When using Argo’s docker executor, the containerOps also needs to run with privilege because Argo’s docker executor needs to use the host machine’s docker.sock. Note: Argo’s PNS executor is not available in OpenShift 3.11 because it’s beta from Kubernetes 1.12+.

Also for Istio 1.3 and below, there are extra setup needed to be done. By default, OpenShift doesn't allow containers running with user ID 0 (root user). You must enable containers running with UID 0 for Istio's service accounts:
https://archive.istio.io/v1.3/docs/setup/platform-setup/openshift/

Below are the list of permissions we have relaxed.

```
kind: SecurityContextConstraints
metadata:
  name: anyuid
users:
- system:serviceaccount:istio-system:istio-ingress-service-account
- system:serviceaccount:istio-system:default
- system:serviceaccount:istio-system:prometheus
- system:serviceaccount:istio-system:istio-egressgateway-service-account
- system:serviceaccount:istio-system:istio-citadel-service-account
- system:serviceaccount:istio-system:istio-ingressgateway-service-account
- system:serviceaccount:istio-system:istio-cleanup-old-ca-service-account
- system:serviceaccount:istio-system:istio-mixer-post-install-account
- system:serviceaccount:istio-system:istio-mixer-service-account
- system:serviceaccount:istio-system:istio-pilot-service-account
- system:serviceaccount:istio-system:istio-sidecar-injector-service-account
- system:serviceaccount:istio-system:istio-galley-service-account
- system:serviceaccount:istio-system:istio-security-post-install-account
- system:serviceaccount:istio-system:istio-multi
- system:serviceaccount:istio-system:kiali-service-account
```

Below are the list of service accounts from KubeFlow. Since all of the core services are running with user root, we also need to give "anyuid" privilege to these service accounts in order to allow them to run with root similar to the istio setup.

```
- system:serviceaccount:{{ namespace }}:admission-webhook-bootstrap-service-account
- system:serviceaccount:{{ namespace }}:admission-webhook-service-account
- system:serviceaccount:{{ namespace }}:application-controller-service-account
- system:serviceaccount:{{ namespace }}:argo
- system:serviceaccount:{{ namespace }}:argo-ui
- system:serviceaccount:{{ namespace }}:centraldashboard
- system:serviceaccount:{{ namespace }}:default
- system:serviceaccount:{{ namespace }}:jupyter-web-app-service-account
- system:serviceaccount:{{ namespace }}:katib-controller
- system:serviceaccount:{{ namespace }}:katib-ui
- system:serviceaccount:{{ namespace }}:meta-controller-service
- system:serviceaccount:{{ namespace }}:metadata-ui
- system:serviceaccount:{{ namespace }}:metrics-collector
- system:serviceaccount:{{ namespace }}:ml-pipeline
- system:serviceaccount:{{ namespace }}:ml-pipeline-persistenceagent
```

- system:serviceaccount:{{ namespace }}:ml-pipeline-scheduledworkflow
- system:serviceaccount:{{ namespace }}:ml-pipeline-ui
- system:serviceaccount:{{ namespace }}:ml-pipeline-viewer-crd-service-account
- system:serviceaccount:{{ namespace }}:notebook-controller-service-account
- system:serviceaccount:{{ namespace }}:pipeline-runner
- system:serviceaccount:{{ namespace }}:profiles-controller-service-account
- system:serviceaccount:{{ namespace }}:profiles-default-service-account
- system:serviceaccount:{{ namespace }}:pytorch-operator
- system:serviceaccount:{{ namespace }}:seldon-manager
- system:serviceaccount:{{ namespace }}:tf-job-dashboard
- system:serviceaccount:{{ namespace }}:tf-job-operator
- system:serviceaccount:{{ namespace }}:spartakus

The kubeflow-operator needs to have privileged access because they will create all the cluster-wise resources such as KubeFlow CRD, clusterrole, etc.

The Istio-system also needs privileged access to inject sidecars based on the istio docs
https://archive.istio.io/v1.3/docs/setup/platform-setup/openshift/

---
kind: SecurityContextConstraints
metadata:
  name: privileged
users:
- system:serviceaccount:{{ namespace }}:kubeflow-operator
- system:serviceaccount:istio-system:default
```

## Authentication and multi-tenancy

For Authentication, we use Istio Dex kfdef.
https://github.com/kubeflow/manifests/blob/master/kfdef/kfctl_istio_dex.yaml

Below are the list of multi-tenancy features on the main UI

- Notebook: Namespace separation for each user.
- Katib: Users still have the permission to deploy katib jobs on any namespace from the GUI.
- Pipeline: At the moment, multi-tenancy is not yet integrated in pipeline on 0.7.
- ~~TFJob Dashboard~~: No longer being exposed on the main dashboard.
- Artifact Store: View artifacts only. Uploading artifacts can be done within Pipeline using pipeline definition/Notebook using ml-metadata sdk/Katib using container/training script.

The Istio Dex kfdef also provides OIDC auth using Istio and keeps records of authentications using the Dex services. Therefore, user requests without the correct auth token will automatically get rejected by the istio ingress-gateway with status code 403.

*Permission needed:*
The Istio dex deployment mostly depends on the type of storage it uses for storing all the user data. Some filesystem may not allow POSIX actions and/or managing files without root permission. Therefore,
```

the default storageclass used on OpenShift must allow non-root users to read/write/execute any file on the mounted path since the OIDC auth service is specified to not running as root.

For example, NFS on IBM Cloud comes with no write/execute permission for non-root users. Hence, we have to change the file system permission prior to deploying the OIDC auth service using an overlay.

Configurations

Three levels of configurations may be implemented for Kubeflow operator: the environment key-value through operator deployment, the spec key-value defined in the custom resource manifest and the config map for specific component of Kubeflow. With these mechanisms, updating Kubeflow should be easy with one `kubectl` or `oc` command to apply the updated manifest.

Environment key-value through operator deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubeflow-operator
spec:
...
    spec:
      containers:
        - name: ansible
 ...
          env:
            - name: WATCH_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: OPERATOR_NAME
              value: "kubeflow-operator"
```

Spec key-value defined in the custom resource manifest:

```
apiVersion: operators.ibm.com/v1alpha1
kind: Kubeflow
metadata:
  name: kubeflow-pns
annotations:
  ansible.operator-sdk/reconcile-period: "0"
spec:
  patch_minio: true
  executor: pns
```

Config map for specific component of Kubeflow:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubeflow-operator
spec:
...
    spec:
      containers:
        - name: ansible
...
            - name: KUBECTL_VERSION
              valueFrom:
                configMapKeyRef:
                  name: openaihub-install-config
                  key: KUBECTL_VERSION
```

Subpath Dependencies

On some particular deployments of OpenShift(3.11), we also allowed subpath to be used since the default minio storage is mounted with subpath. Subpath is a way to specify a sub-path inside the referenced volume instead of its root. Minio itself can be functional without subpath and subpath configuration can be changed/relaxed within OpenShift. Potential path to upstream changes to Kubeflow to eliminate dependencies on subpath

Versioning

Kubeflow operator aims to keep up with the release version of Kubeflow project. To illustrate, the directory structure will look like this in the Operator Framework's community-operators repo:

kubeflow-operator
├── kubeflow.crd.yaml
├── kubeflow.package.yaml
├── kubeflow.v0.7.0.clusterserviceversion.yaml
├── kubeflow.v0.7.1.clusterserviceversion.yaml
└── kubeflow.v1.0.0.clusterserviceversion.yaml

A ClusterServiceVersion (CSV) is a YAML manifest created from Operator metadata that assists the Operator Lifecycle Manager (OLM) in running the Operator in a cluster. The CSV is also generated with important information such as operator versioning which is useful for tracking different deployment version and upgrade to a new version using OLM.

For each operator version, an operator image will be built and maintained with tag reflecting the release version, for example, `quay.io/operators/kubeflow-operator:v0.7.0`.

Upgrade

With the version control described above, one version of KubeFlow operator will be built for each release of Kubeflow. The upgrade strategy will be inlined with the Kubeflow's upgrade strategy drafted in this document. That is, the tasks in the operator with higher version will know the existence of current Kubeflow deployment and handle the upgrade with the proposed `kfUpgrade` manifest through `kfctl` command. One more thing, Operator SDK tool makes it easy to generate a new version of CSV for Kubeflow operator by copying user-defined fields from the old CSV to the new CSV and updating the new version if any manifest data is changed.

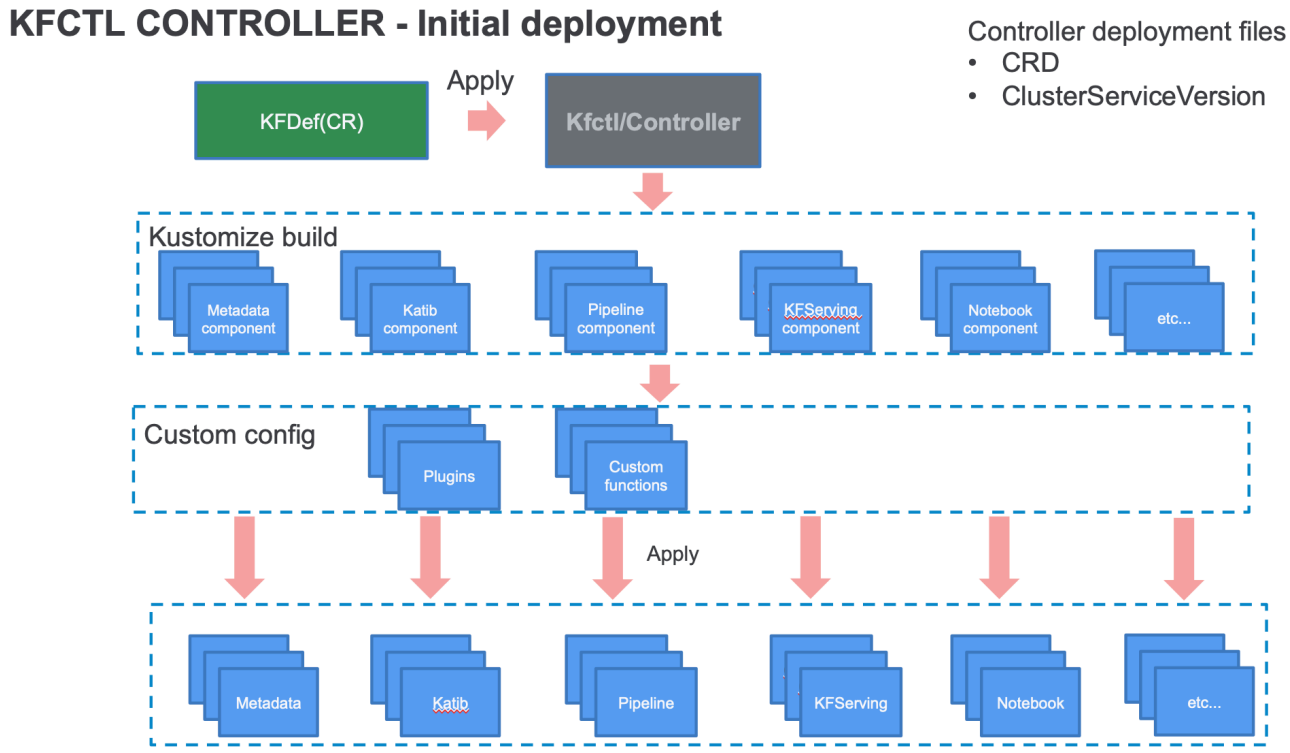
Alternate Design and Implementation - GO Operator:

1. Pros and cons
    a. Pros
        i. Golang based controller generally have better performance over bash/ansible in the long run because each bash command will run on a new process where golang program usually runs on a single process.
    b. Cons
        i. We are restricted within the Golang library and using exec for custom script on any additional implementation we want to make.
        ii. Higher barrier to entry, not very 'Operations' centric.

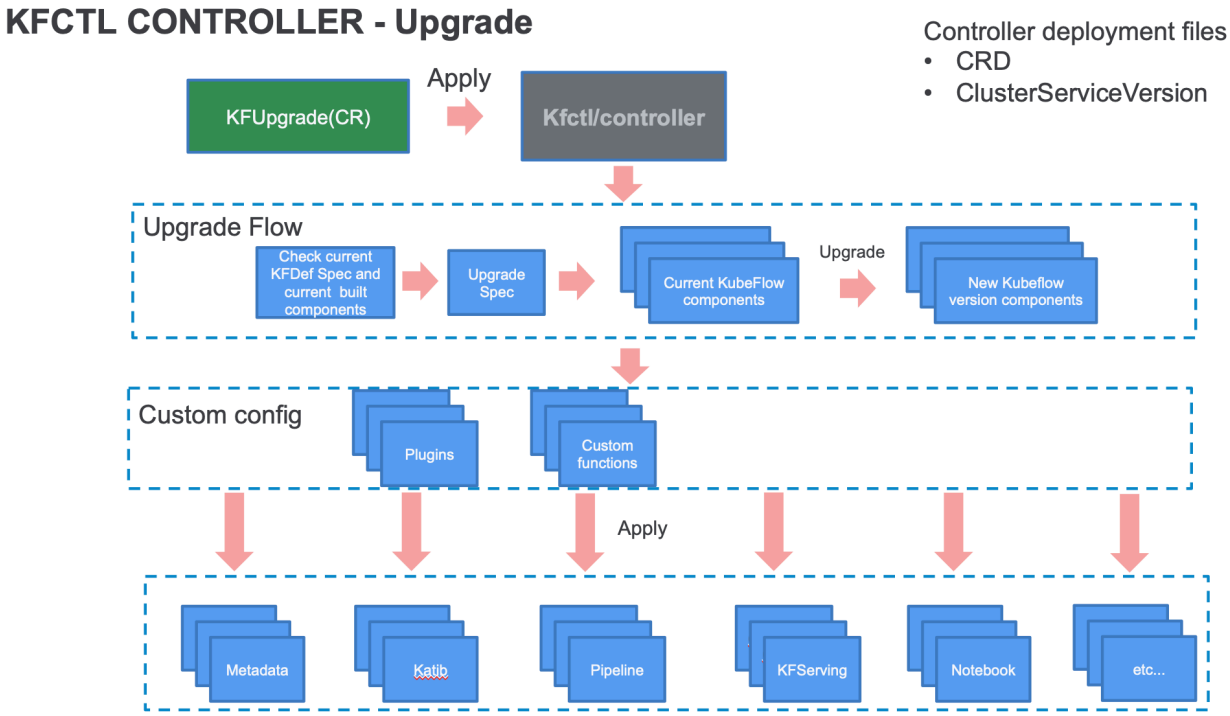
A Go based design can be followed for an Operator, to be based of from KFDef and kfctl. Kfctl TODOs for making it as an operator:

2. CRD (simple to create once we decided on the naming and version)
3. Core controller code such as API and Reconciler.
    a. Api
        i. Core functions such as apply/build
        ii. Status functions such as get deployed components
    b. Reconciler (Periodic checks)
        i. Actions to take during reconcile period
            1. apply
    c. Watcher
        i. Watch CR spec changes?
4. Ways to execute custom code using the controller.
    a. Using kfctl plugins
        i. Plugin functions are also defined as golang. Therefore we can create some functions to execute custom code and dependencies using exec command.
    b. Using pure golang within controller
        i. Using the exec library and run it as a subprocess
            1. `cmd := exec.Command("/bin/sh", script)`
5. Version control
    a. Operator versioning
        i. Using Operator Lifecycle Manager (OLM)'s ClusterServiceVersion (CSV)
    b. Application versioning

i. Keep a KubeFlow deployment versioning in the CR spec. Then we can make KFUpgrade as another CR using the same controller for doing KubeFlow version upgrade on a user specified KFDef. The KFUpgrade CR should trigger a version update action and return the update status back to the same KFUpgrade CR. Note that KFUpgrade may able to modify the deployed KFDef spec to accommodate the version upgrade

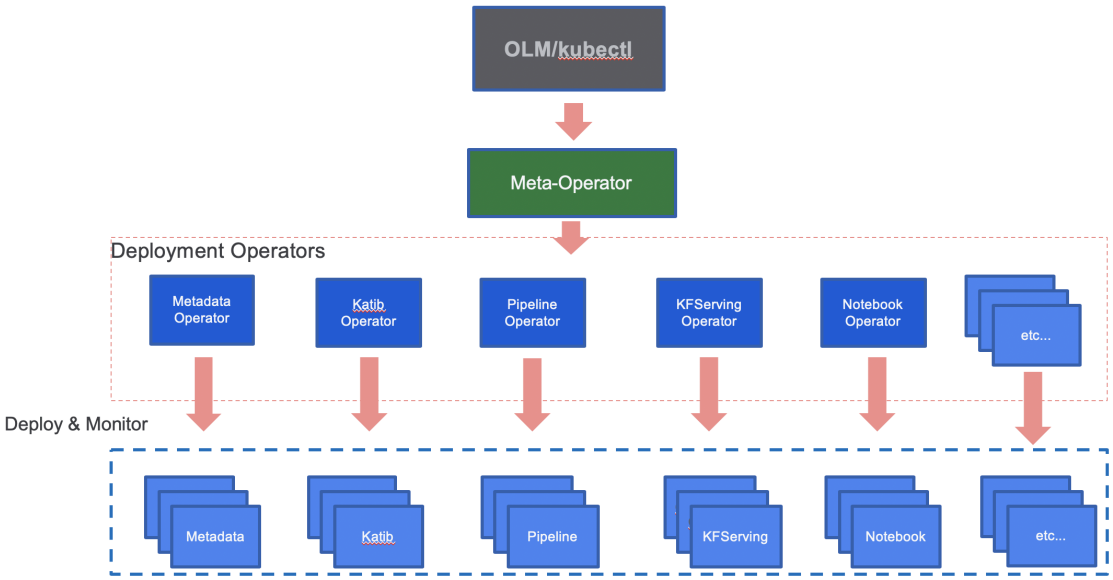


.



Future - Meta Operators

Version 2 of the Operator may follow the Meta Operator architecture pattern, where we can convert individual components of Kubeflow which are still not in Operator format, for e.g. Pipelines, KFServing etc. into individual operators, which can be deployed and managed through a meta operator.



OpenShift Issues

https://github.com/kubeflow/manifests/blob/master/kfdef/kfctl_k8s_istio.yaml

- Require cluster-admin for the operator's service account on OpenShift 3.11

This problem was hit when installing Kubeflow operator 0.7.0. Following manifest aims to create a kubeflow-istio-admin cluster role (excerpted from kustomize/istio/base/cluster-roles.yaml).

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kubeflow-istio-admin
  labels:
    rbac.authorization.kubeflow.org/aggregate-to-kubeflow-admin: "true"
aggregationRule:
  clusterRoleSelectors:
  - matchLabels:
      rbac.authorization.kubeflow.org/aggregate-to-kubeflow-istio-admin: "true"
rules: []
```

This failed on OC 3.11 cluster

Error from server (Forbidden): error when creating "role.yaml": clusterroles.rbac.authorization.k8s.io "kubeflow-istio-admin" is forbidden: must have cluster-admin privileges to use the aggregationRule
---

But the same succeeded on OC 4.1.

https://github.com/kubeflow/tf-operator/pull/1100

https://docs.google.com/document/d/12ikhUKAb3KhbO9AR6JUk_UX_D9pf7nFWfHAyLDv2BB0/edit

References

https://github.com/kubeflow/manifests/issues/686 - KFDef for openshift
https://github.com/kubernetes-sigs/kubebuilder/issues/1074 - Recipe for singletons

Appendix: Feedback

- Don't limit to a Singleton model
- Align efforts towards a Go based Operator, built on KFDef CRM, KFCtl as the nucleus for Controller

- Handle customization primarily through Kustomize Overlays, and use plugins if advanced customizations are desired
- For Kubeflow, the reconcile function is the "apply" function.
- https://github.com/kubeflow/kfctl/blob/19d118cb188dc7e89861669dd0db45157e469927/pkg/kfapp/coordinator/coordinator.go#L353
  kfctl and the KF operator would share this function so that we only have to implement changes once, for example, if we want to add the "control-plane" label to the KF namespace to prevent webhooks from being applied to the KF namespace it would be nice if we only had to update the apply function.
- Creating K8s resources to manage non K8s resources is becoming a pattern.
  See
  [1] https://github.com/GoogleCloudPlatform/k8s-config-connector
  [2]
  https://aws.amazon.com/blogs/machine-learning/introducing-amazon-sagemaker-operators-for-kubernetes/
  If we follow this pattern than the KF controller only needs to interact with K8s APIs.
- If we want to use Ansible to perform these operations an alternate design would be
  A. Create operators to manage these external systems
  B. An operator for registering DNS entries - This is what was done for GCP; we created an operator for cloud endpoints
  C. just need to create K8s resources in order to configure these external systems
  D. you can define kustomize packages for these resources
  E. Add an application entry to your KFDef.Spec.Applications to create these external resources as part of deployment.
- Custom resource just a KFDef - So people could do
  kubectl apply -f kfdef.yaml
  to create an instance of the resource that would get deployed via the operator