All-in-one Kubeflow Training Operator

Authors: Jiaxin Shan(@Jeffwan) Bytedance,
Wang Zhang(@zw0610) Tencent
Create Date: April, 12th, 2021
Last Update: May, 18th, 2021

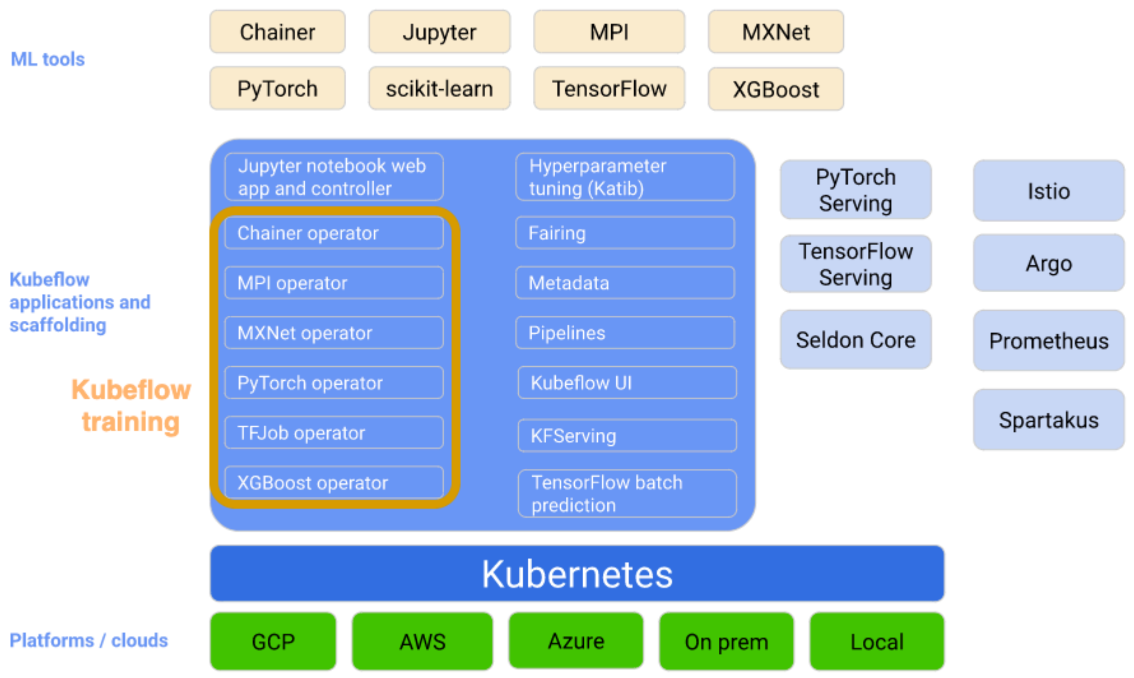
Background

The Kubeflow project is a complex project that aims at simplifying the provisioning of a Machine Learning infrastructure. It is built on top of Kubernetes and adapts lots of kubernetes core components for ML use cases.

Kubeflow training is a group of Kubernetes operators that add to Kubeflow support for distributed training of Machine Learning models using different frameworks.

There are two major distributed training strategies: one based on parameter servers (tf-operator, etc) and the other based on collective communication primitives such as allreduce (mpi-operator).

The latest Kubeflow release supports eight different operators. See **Appendix A** for details.



(credits Bachir@ https://dzlab.github.io/ml/2020/07/18/kubeflow-training/)

As the community continued to grow and support other distributed training frameworks, the community did some refactoring and made common types available in a standalone repo (kubeflow/common) to reduce code duplication and provide a clear pattern for long-term API governance. See **appendix B** for more details.

Kubeflow/common in the past year helps a lot in sharing common types and implementations. At the same time, Job API becomes fairly stable and most operator implementations look pretty close. However, project maintenance is still an outstanding problem these days. Some challenges are

- **Hard to keep framework independent features (sdks, metrics) and bug fixes consistent between operators.**
- **Hard to reuse shared testing and release infrastructure.**
- **Duplicate efforts to interact with Kubeflow components like KFP LauncherOp, Katib as well external orchestra schemes like knowledge distilling, job queue and actor-learner rl training**
- **Maintainers efforts increases linearly with the number of supported frameworks**
- **High cost for new developers to learn difference between operators**

There're also some user facing challenges

- **Users who use multiple frameworks have to install different operators. Duplicate efforts on logs, metrics, and api interactions.**
- **JobSpec looks slightly different and it doesn't give user consistency experience**

Proposal 1 - Monolithic

Build a unified operator that supports multiple DL/ML frameworks. (TF/PyTorch/MPI/MXNet/XGBoost)

High Level Design

Kubebuilder can be used to generate skeletons. It creates a single binary that contains all the controllers.

Only one manager is needed and it will contain all the controllers defined in that project. Each framework will have a corresponding job controller. This enables performance enhancements such as only having one shared informer cache for all the controllers, reducing the number of requests to k8s API. At the same time, user specified controllers can register themselves to the manager in a configurable way. This helps users scope down frameworks they like to use instead of enabling all controllers.

Each job controller setup with unified manager

```
mgr, err := ctrl.NewManager(ctrl.GetConfigOrDie(), ctrl.Options{})

if IsJobRegistered(tfjob_controller.GetSchema()) {
    tfjob_controller.SetupWithManager(mgr)
}

if IsJobRegistered(pytorchjob_controller.GetSchema()) {
   pytorch_controller.SetupWithManager(mgr)
}

...
```

Advantages

1. Share the same testing and release infrastructure.
2. Unlock more production grade features like manifests/metadata supports.
3. Save huge efforts in each kubeflow release.
4. One source of truth for other kubeflow components to interact with.

Remaining Challenges

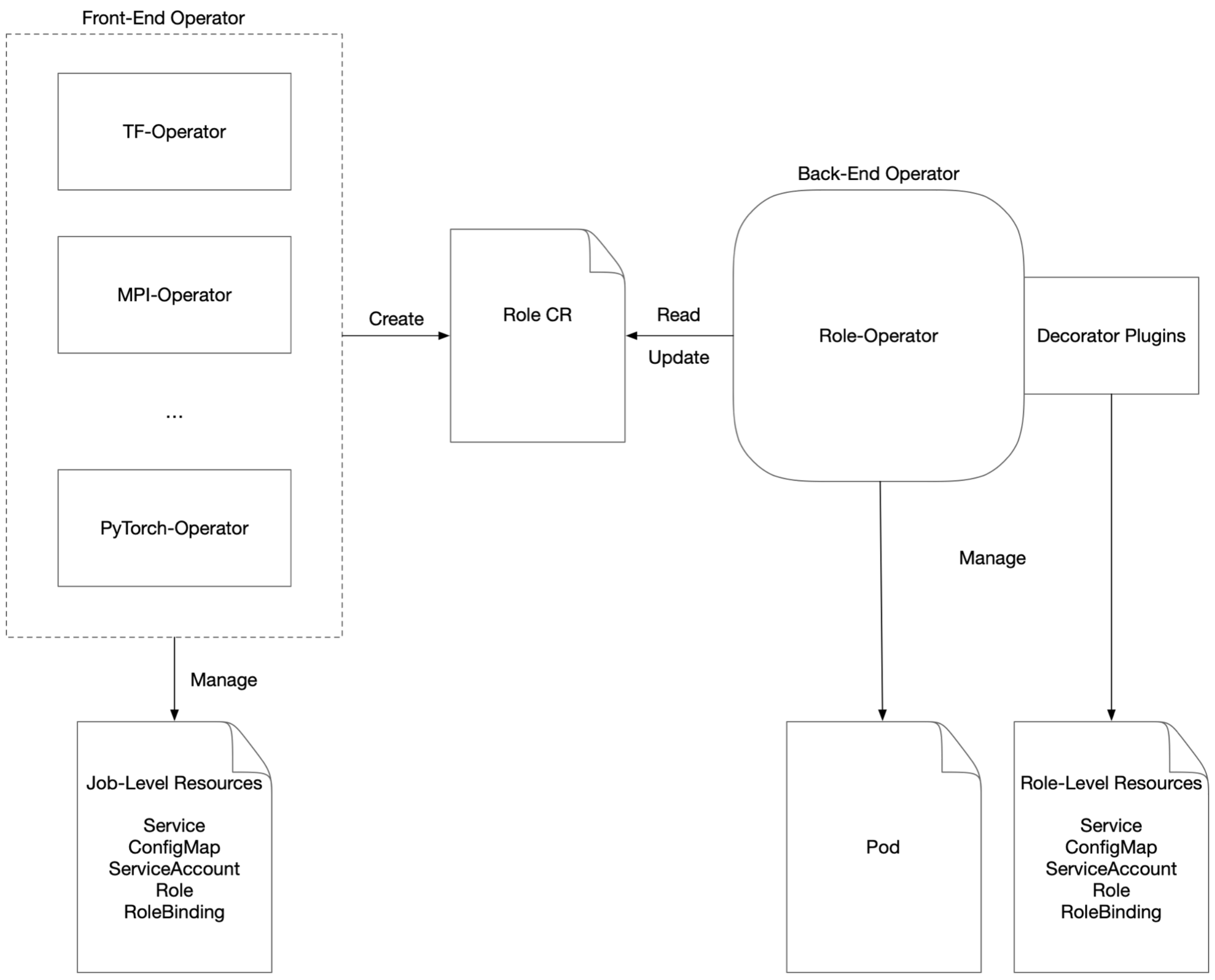
1. What's the process to add a new framework?
2. Community loses granular control to graduate/release a single operator.
3. Can we guarantee all operators' implementations following the same pattern? Mpi-operator is a little bit different from those built on Parameter Servers.
4. The future plan for kubeflow/common? Kubernetes sync staging components to separate repo, we should probably learn that pattern.

Proposal 2 - Duo-Deck

The principle idea of this proposal is to ***force*** training operators to use the kubeflow/common, which no longer presents itself as a golang package (sdk), instead as a service accepting requests via (declarative) API.

1. Convert kubeflow/common into a (common-)role-operator that supports a new CRD: `` `*Role*` ``. The conceptual struct of `` `*Role*` `` is listed below.
2. Convert existing training operators to 'front-end' operators, generating corresponding role CRs and auxiliary resources like service, configMap or serviceAccount.

High Level Design



```
type Role struct {
      // Standard Kubernetes type metadata.
      metav1.TypeMeta `json:",inline"`

      // Standard Kubernetes object's metadata.
      // +optional
      metav1.ObjectMeta `json:"metadata,omitempty"`

      // Specification of the desired state of the Role.
      // +optional
      Spec *commonv1.ReplicaSpec `json:"spec,omitempty"`
```

```
	// Most recently observed status of the Role.
	// Populated by the system.
	// Read-only.
	// +optional
	Status commonv1.ReplicaStatus `json:"status,omitempty"`
}
```

1. End-users still submit existing CRs like TFJob, MPIJob
2. ‘Frontend’ operators creates auxiliary job-level Kubernetes resources like Service, ConfigMap
3. ‘Frontend’ operators decompose Job CR into different Role CRs, defining PodTemplate as well as features like Fault-Tolerant, Elastic Training, etc.
4. ‘Backend’ operator watches all Role CRs and manages Pods performance
5. ‘Backend’ operator has decorator plugins processing decorators defined in each Role CRs and use the corresponding plugins to maintain role-level Kubernetes resources like Service, ConfigMap

Advantages

1. Force all operators to use the unified meta API (Role CRD)
2. Sharing advantages from proposal 1

Remaining Challenges

1. Operator developers switch from programming with sdk to programming with API
2. Community loses granular control to graduate/release a single operator
3. The design is not compatible with all or guaranteed compatible with all operators design

Appendixes

Appendix A: Existing Kubeflow Training Operators

Framework	**Project Repo**	**Status**
Tensorflow	https://github.com/kubeflow/tf-operator	
PyTorch	https://github.com/kubeflow/pytorch-operator	
MPI	https://github.com/kubeflow/mpi-operator	

MXNet	https://github.com/kubeflow/mxnet-operator	
XGBoost	https://github.com/kubeflow/xgboost-operator	
Fate	https://github.com/kubeflow/fate-operator	
Chainer	https://github.com/kubeflow/chainer-operator	Plan to archive
Caffe2	https://github.com/kubeflow/caffe2-operator	Plan to archive
PaddlePaddle	https://github.com/kubeflow/community/pull/502	Reviewing
DGL	https://github.com/kubeflow/community/pull/512	Reviewing

Appendix B: Kubeflow/common History

Community kicks off kubeflow/common project in early 2019 and finishes first version around mid 2019. (see issue Proposal for a Common Operator). The first draft doesn't provide enough flexibility and maturity and most operators at that time still import tf-operator as a dependency to leverage common tools.

After some enhancements and refactoring in mid 2020 (see issue Enhance maintainability of operator common module), it becomes stable and can be easily adapted into operators written by kubebuilder/operator-sdk or low level client sets. Currently, tf-operator, mxnet-operator, xgboost-operator are all built with kubeflow/common library.