Motivation:

Kubernetes itself does not provide any templating mechanism. There are multiple benefits of a template design for the Kubeflow community. The first is to expedite the development of a AI modeling process where the switch of an intermedium model parameters are frequent. One example is the switch of parameters of hidden layers in the transfer learning process. The second is a fast synthesis of a new AI training flow and exploration of the optimization process. We propose a template construct for hidden layers replacement during transfer learning process. The deployment medium is a static yaml file ready for kubeflow pipeline execution.

Methods:

We define the template workflow as follows: execute and store the state of an AI flow, treats the workflow as an active object. The template contains a list entry points for a functions that define the instructions to execute. An entry point specifies the main function of the workflow or the first template to be executed.

To fulfill the definition of Kubernetes CRD custom resources, we use Kubernetes container for the construct. We wrap the input layer of the model through an empty container, and provide it to other models for operation, and finally use the Yaml file to provide parameter replacement, artifacts, fixtures , loop and recursion workflows. The following are the steps for our transfer learning template construct.

1. trainOp serves as the main entry point, configure the target model and save the path to the model files. When faced with multiple Kubernetes clusters, we provide resources that can be applied to all environments with minimal modifications. When the application is running in the staging environment, it should connect to the staging database and use environment variables to inject the correct database URL.



1. The following is the terminology explanation of the above preparation operation.

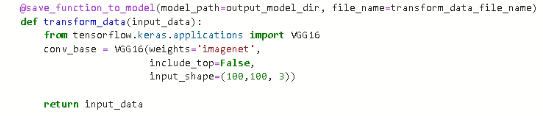
(a)model\_relative\_path - to save the model, save a model with tensorflow.keras to a folder under the current directory of model\_path.

(b).model\_name - to use the generated model name as the file holder for all generated model files.

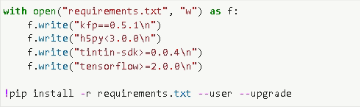
(c).model\_dir = os.path.join(output\_model\_dir, model\_name) save the model to the path.

(d).pathlib is an object-oriented file system path, which encapsulates the operations related to files and folders in categories such as Path, making the operations of files/folders more object-oriented. Classes are provided that represent filesystem paths, have semantics applicable to different operating systems, and also provide I/O operations.

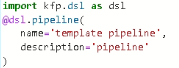
3. pathlib.Path(model\_dir).mkdir(parents=True, exist\_ok=True)—Write the transfer learning input layer into the empty folder:



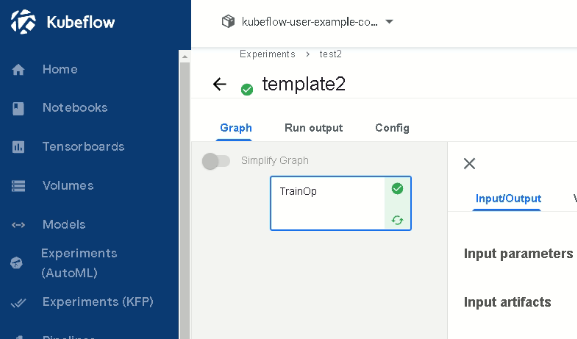
1. Write the required environment and parameters into requirements.txt:



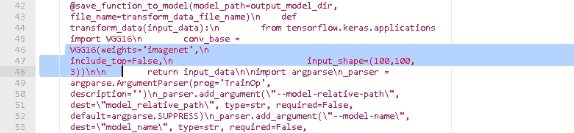
1. Build a pipeline components, and a pod to store the model:



Add a comment to the defined templated\_pipeline\_func category to configure the model\_relative\_path and model\_name into the workflow, as well as the default resources for cpu and memory. Finally, kfp.compiler encodes the model into a yaml file and outputs it as a compressed file, and runs it in kubeflow as follows:



1. The packaged model input layers can be seen in the pod:



1. Use kubectl in the Minikube to check for the log of the pod from 6.



1. The template workflow is summarized as below:

