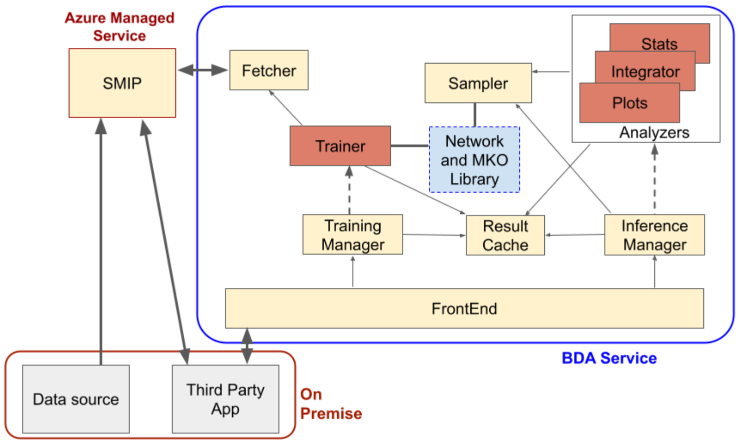


Figure 1: Overview of the BDA Service Architecture

1 Distribution overview

The code distribution contains two top level directories. `docs` and `src`. Additionally, there may be miscellaneous files such as `readme.md`, `license.txt`, and `.gitignore` at the top level.

`docs` contains documentation, both source .tex and output .pdf files.

`src` contains the source and ancillary files for all of the microservices that make up the BDA Service. These services are show graphically in Figure 1. Additionally, there are directories within `src` that contain scripts to deploy the BDA Service.

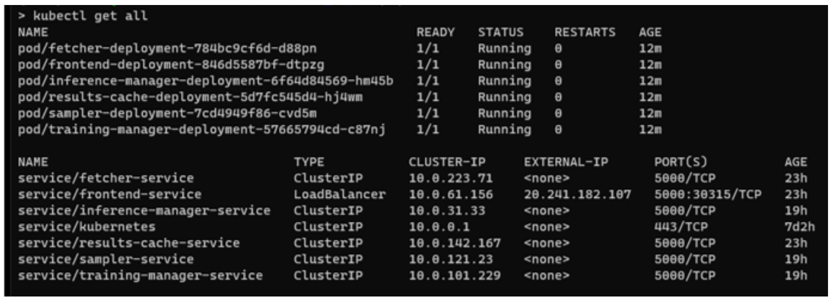
2 Deployment

2.1 Dockerizing BDA Service

The distribution includes scripts for creating all the microservice docker images to instantiate the BDA Service either locally or in the cloud. The intent is to deploy the Service to a kubernetes cluster.

The main script to create the needed docker images is `src/docker/build_images.sh`. This script can be from the `src` directory. Before running this script, it is important to do two things: First, edit the script variable `roottag` to point to a docker registry that will be used for deployment to the cloud. This may be an AWS, Azure, or other registry. A local registry can be used if you plan to run the Service locally or to manually push images to the cloud registry. Second, make sure the current user has authenticated to the registry. Useful flags during building are include the following:

`-p` build and then push to the registry



```
> kubectl get all
NAME                                              READY   STATUS    RESTARTS   AGE
pod/fetcher-deployment-784bc9cf6d-d88pn           1/1     Running   0          12m
pod/frontend-deployment-846d5587bf-dtpzg          1/1     Running   0          12m
pod/inference-manager-deployment-6f64d84569-hm45b 1/1     Running   0          12m
pod/results-cache-deployment-5d7fc545d4-hj4wm     1/1     Running   0          12m
pod/sampler-deployment-7cd4949f86-cvd5m           1/1     Running   0          12m
pod/training-manager-deployment-57665794cd-c87nj  1/1     Running   0          12m

NAME                                TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)          AGE
service/fetcher-service             ClusterIP      10.0.223.71    <none>           5000/TCP         23h
service/frontend-service            LoadBalancer   10.0.61.156    20.241.182.107   5000:30315/TCP   23h
service/inference-manager-service   ClusterIP      10.0.31.33     <none>           5000/TCP         19h
service/kubernetes                  ClusterIP      10.0.0.1       <none>           443/TCP          7d2h
service/results-cache-service       ClusterIP      10.0.142.167   <none>           5000/TCP         23h
service/sampler-service             ClusterIP      10.0.121.23    <none>           5000/TCP         19h
service/training-manager-service    ClusterIP      10.0.101.229   <none>           5000/TCP         19h
```

Figure 2: kubectl view of deployed BDA service, including external access IP address (here 20.141.182.107:5000)

`-R [registry name]` update default registry name before building.

Running this script should incorporate all the code into docker images and optionally push this image to where a cluster will pick them up, create containers from them, and run them as the BDA Service.

2.2 Kubernetes

The intent of this code is to send it into the kubernetes (k8s) cluster in the cloud for efficient use of powerful resources. Basic scripts needed to do this are in the `src/k8s` directory, although a knowledgeable user of k8s will likely wish to make changes to this deployments.

The `Cluster_start.sh` script creates and uses a set of k8s yaml files in a directory structure based on the image registry in which the docker images reside. For instance, if the images were pushed to `reg.aws.io/myReg`, then `Cluster_start.sh` will create a series of k8s yaml files in `k8s/reg.aws.io/myReg/` with image specifications inside those files pointing to `reg.aws.io/myReg`. This behavior is directed by the `-R [registry name]` flag to `Cluster_start.sh`.

The `Cluster_start.sh` script also does a `kubectl apply -f` to these files, one at a time (or a `kubectl delete -f` if the `-d` flag is specified) to start up (or shut down) the various containers and services.

The `frontend-service` service acts as an ingress point into the k8s cluster on port 5000. The external IP address for this ingress point can be seen (as in Figure 2 by using `kubectl get service` or `kubectl get all`.