# TensorFlow on YARN



# Motivation

As discussed in the umbrella HADOOP-13944, we'd like to propose Tensorflow on YARN. Since TensorFlow is one of the most popular open source framework for machine learning and deep learning, it is a good start to do such an integration between TensorFlow and Hadoop. Hadoop users can use TensorFlow in a similar and easy way as other Hadoop applications. YARN is the junction for this integration. TensorFlow cluster would be good to directly run on a YARN cluster. YARN can be used as the resource manager to host the running of TensorFlow clusters, and TensorFlow clusters take charge of the execution of the deep learning jobs.

# Goals

1. One YARN cluster can run multiple TensorFlow clusters.
2. Run the TensorFlow cluster directly on YARN without Python or Tensorflow binary installation to minimize the dependencies of usability. The Tensorflow runtime should be packaged into a bundle and easy to be distributed and invoked.
3. A TensorFlow cluster usually runs in a long-term manner, and the resources are pre-allocated. This is not friendly to YARN. So in TensorFlow on YARN, the TensorFlow cluster runs in a short-term manner, it is destroyed after the session is finished.
4. Support existing TensorFlow tools and jobs.

# User Interface

At present, we focus on deep learning training first and consider inference or scoring in next phase.

**Command line**:
A command tool is provided to allow users to submit a Tensorflow job, as below. Note this is subject to change as our implementation is fast evolving.

```
yarn-tf -job mnist.py [-numworkers -numps] ...
```

This assumes a user wants to run a job “mnist.py” specifying how many workers and parameter servers.

**Client API**:
In addition, we’ll also provide client API for above computing frameworks to invoke. This is to be defined.

# Design

The following figure shows the component layout in the architecture of TensorFlow-YARN.

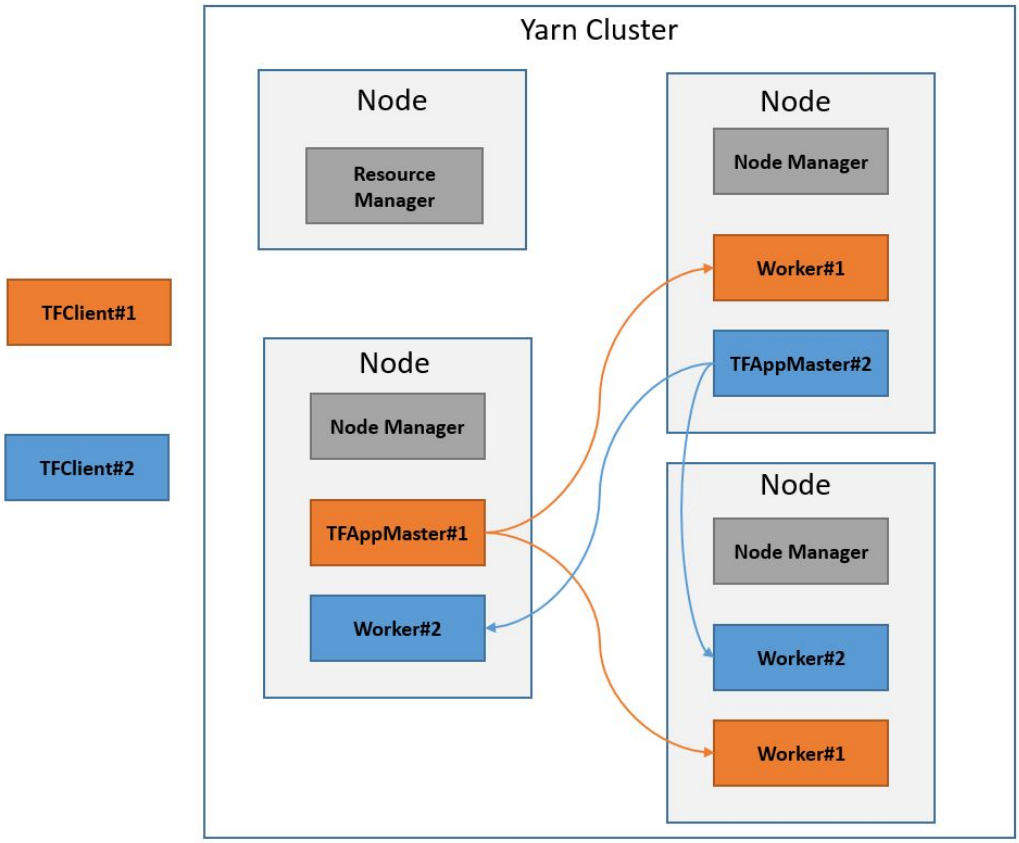


Figure 1. Architecuture of Tensorflow-YARN

One YARN cluster can run multiple TensorFlow clusters, each TensorFlow cluster runs in a short-term manner.  The tasks from the same and different sessions can run in the same node, if servers in TensorFlow clusters are located in the same node, they should have different network ports.

A TFAppMaster should be implemented. And the TensorFlow workers and parameter servers run either directly in YARN containers.
One TFAppMaster runs for only one session, when the session is finished, this application master should be stopped, and the containers in this TensorFlow cluster should be stopped too. The cluster specification should be generated dynamically for each TensorFlow cluster at the beginning.

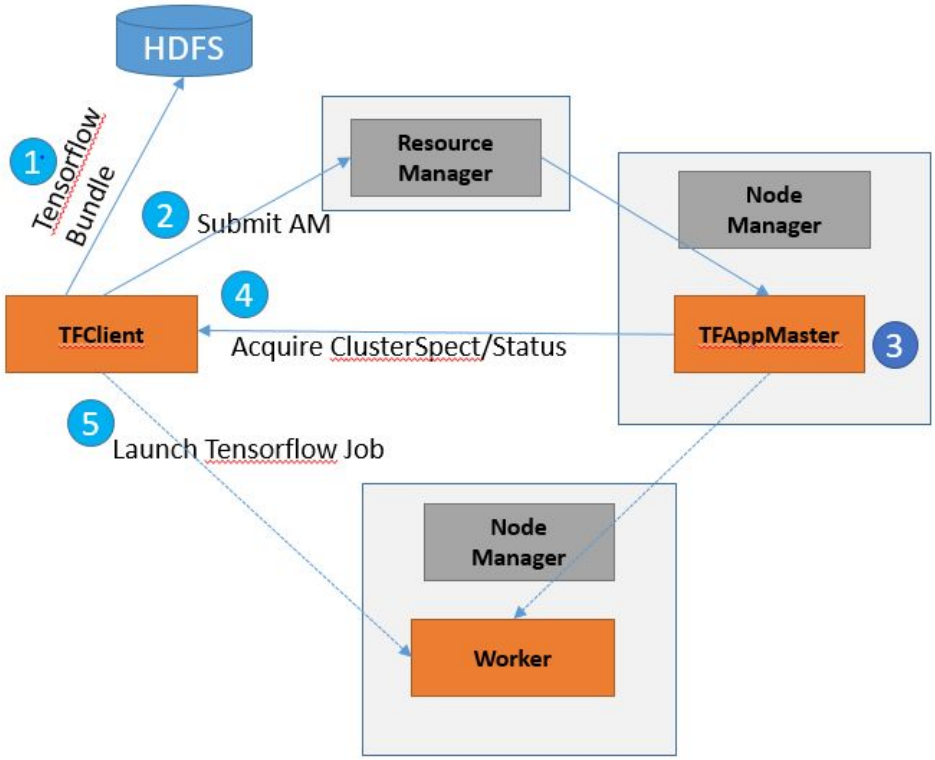


Figure 2. Tensorflow-YARN Client

Above figure illustrates how the TFClient requests a Tensorflow cluster and starts the user's job.

1. Upload the TensorFlow bundle of Tensorflow core library and JNI wrapper to HDFS for application master to leverage later
2. Submit application master launch request to RM
3. Wait for AM to finish the real launch work
4. Get back Tensorflow cluster "ClusterSpec" or status
5. Launch the user's Tensorflow application

For application master's angle, the following steps show how the it starts a TensorFlow cluster.

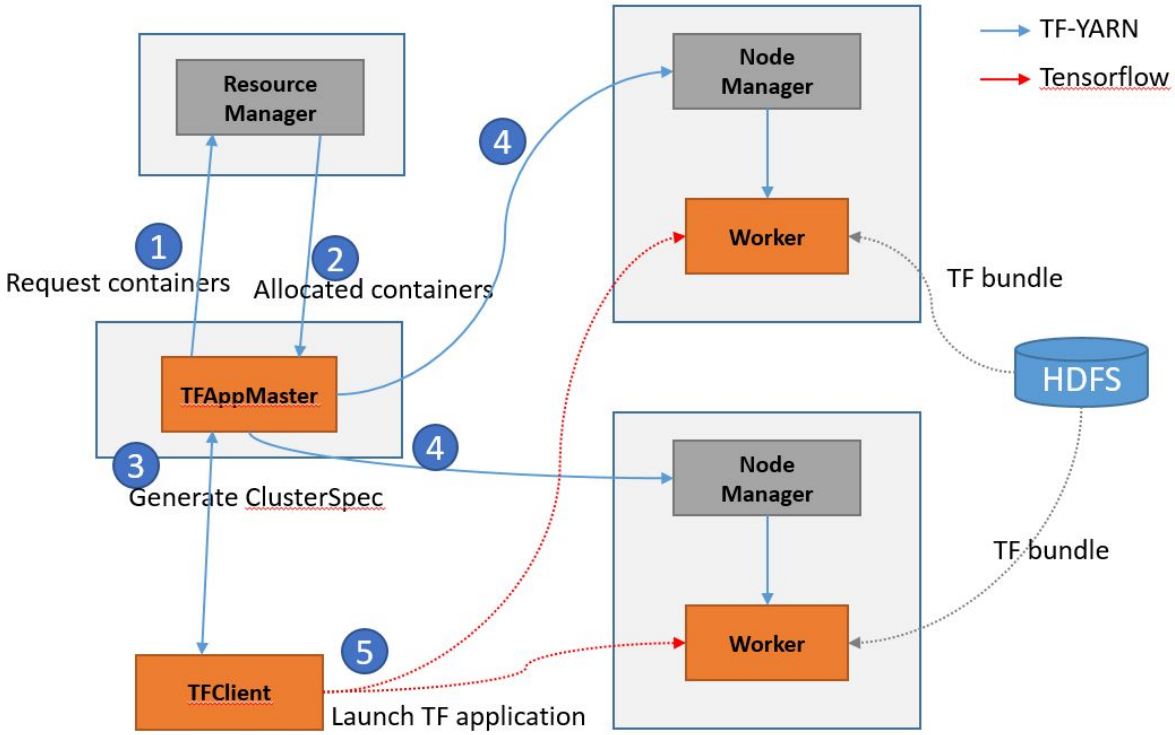


Figure 3. Tensorflow on YARN application master

**Step 1. The TFAppMaster requests the resources from Resource Manager.**
The resources include the numbers of containers, cores and memories, these can be parsed from the parameter from the client. The default number of containers is 3, the number of cores is 1, and the number of memories is 256MB (TODO need to confirm the better value).
The application master collects all the necessary information and submit the allocation request to RM.

**Step 2. Wait until all the resources are allocated**
The application master would wait until all the resources are allocated.
Here we have two approaches to do this.

1. RM can process the requested resource as a batch, which means all the resources are allocated at the same batch, or all of them fail.
2. RM processes the requests resources as usual, we can wait in the application master until all the resources are allocated.

**Step 3. Generate the cluster specification**
We should have the cluster specification before starting the TensorFlow cluster.
The allocation response from RM contains the node information, we need assign the port information to set up the cluster specification.

We could randomly generate the port between a range. For the tasks in the same job we use the same port, different ports would be used if two tasks are located in the same node. The port might be already used in the target nodes, we can re-generate ports to set up a new cluster specification. (TODO could use a zk or hdfs to record the used ports to try to avoid the conflict) Now we have the cluster specification for the TensorFlow cluster, and we can start the TensorFlow cluster now.

**Step 4. Start the containers**
The application master sends the requests to related node manages to start the containers. For YARN container, the step might fail due to the conflict of the ports. The application master should re-generate the ports as Step 3 after the failures, and send the requests to node managers again.
After receiving requests, each node manager downloads the Tensorflow bundle from HDFS for container and launches the container which accepts the cluster specification from the parameters and starts the real Tensorflow ps or worker tasks.

**Step 5. The Client starts the training**
Now all the servers in the TensorFlow cluster in the cluster are started.
Since the TFClient usually resides in a host that has end user expected environment like Python. It's better that the TFClient fetches the "ClusterSpec" from TFAppMaster and run user's job. It's also worth noting that the user's tensorflow job should parse "ClusterSpec" from input parameters.
When the Tensorflow job is finished, the TFClient informs TFAppMaster to stop all the related containers that belong to this TensorFlow cluster.

# Open Discussion

## Fault Tolerance

When a worker fails, TensorFlow will keep trying to connect the failed server. We may support fault tolerance in the similar way. In details, the TFAppMaster can try to start the server on the same node with the same configuration but if it fails anyway after several times of retries, the application master will kill all the containers in this Tensorflow cluster, generate a new cluster specification, and restart the cluster from the beginning.

## Hardware Accelerator Support

In YARN community it's going to support various accelerator technologies such as FPGA, GPU and RDMA. This work would benefit from such efforts and incorporate them when they are done and available.

## Docker support

Docker is another way to run job in YARN cluster which is relatively a  new feature. It would be good to also consider the support of the Docker way in future.