[YARN-7468] [Design] Provide means for container network policy control

By Xuan Gong with inputs from Clay B, Wangda Tan, Junping Du, Vinod Kumar Vavilapalli

Motivation

Some users *"run secure clusters which are secured on the inside to keep data from leaking back out (like a glovebox, things can go in but once in, they may be tainted and can't come out except by very special decontamination)"*.

"The need is to ensure that, network-wise, the applications running on the YARN cluster can reach from/to the local HDFS'es, HBase, databases, etc. Yet, only users permissioned for data ingest jobs should reach out and pull data."

Full-scale software-defined-networking integration into YARN is not the scope of this effort, though that's an interesting direction for future.

Problem statement

We look to isolate network access for applications launched by users/groups.

Ideally, YARN should be able to isolate both of egress and ingress network for launched containers. This JIRA focuses only on egress network isolation. (Here we allow only privileged users the ability to copy sensitive data out from a cluster).

We will look at ingress network in the future.

Requirements

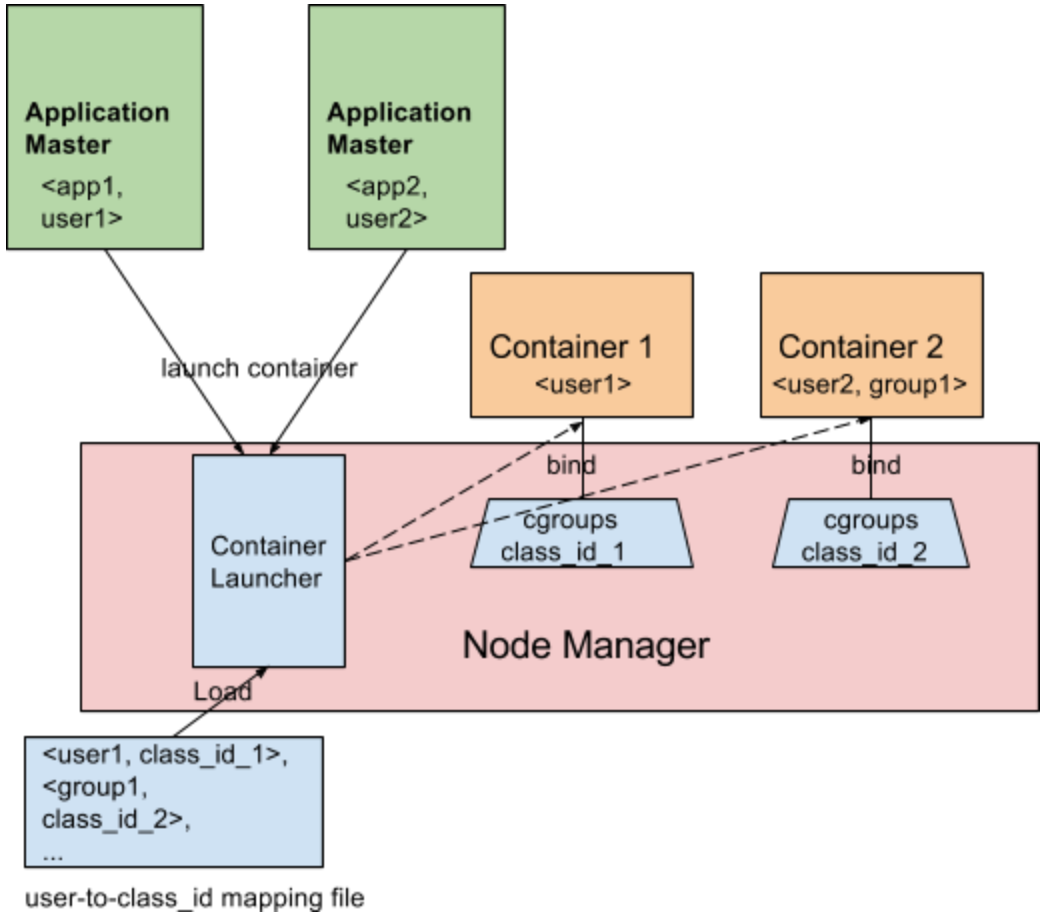
- YARN should tag the traffic going out of YARN containers to enable DMZ like use-cases
    - Data can get into YARN cluster, but bi-directional access to outside destinations has to be controlled
- YARN should not enforce isolation itself - admins should be able to use their tools like iptables
- Tags have to be stable and user configurable - and not tied to ephemeral data (container-ids for example). Also admins can pre-define tags on-which to apply policies
- Policies on the packets arising from the YARN containers
    - Confine within network
    - Accept connections to specific destination for specific users
    - A set of users and groups should be accepted to a specified set of destinations

- Definition and implementation of the policies is not part of YARN

Solution overview

- This JIRA provides container based network ingress/egress rules. We can follow in the footsteps of YARN-2140; using the same cgroups network classifier, we can filter the packets without having to use network namespaces.
- To facilitate this, it is okay to mandate applications that need access to run as specific users. For e.g, a Sqoop job accessing a particular database may have to run as a different user than a Sqoop job accessing a different set of network end-points.

Proposal



We will use cgroups (Network Classifier) for the implementation. Basically, the Network classifier cgroup provides an interface to tag network packets with a class identifier (classid) - an integer. Tools like the iptables can use this tag to perform actions on such packets. Setup of iptables is out of scope of the design - it should be done by the admins. There's an example on net_cls documentation regarding to how to setup iptable to use classid.

Administrators need to provide a mapping between user/group names and classids. It is a simple plaintext file (JSON format) which contains the <user-name A, network-tag-id N> as the key value pair. What this means is that every container from any application from any queue but submitted by user A will have all outgoing network packets marked with cgroup classId = network-tag-id N. If need be, this file-input format can be made pluggable later.

The format of plaintext file could be:

```
{
   "users": [
    {
      "name": "yarn",
      "network-tag-id": "0x10000001"
    },
    {
      "name": "mapred",
      "network-tag-id": "0x10000002"
    }
    {
      "name": "hdfs,kafka", // Comma separated list of users who all map to the same ID
      "network-tag-id": "0x10000003"
    }
     ...
   ],
   "groups": [
    {
      "name": "hadoop",
      "network-tag-id": "0x20000003"
    },
    {
      "name": "group_name2"
      "network-tag-id": "0x20000004"
    }
     ...
   ],
   "default-network-tag-id":"0x99999999"
}
```

Operators can configure this file location through a configuration property. By default, we cold pick this up from YARN_CONF_DIR. We will follow these guidelines to parse the network-tag-mapping file:

- If the file contains the network-tag-id for a specific application user, we would assign the specified class_id for the net_cls subsystem. For example, in previous network-tag-mapping file, we will use 0x10000001 as the class_id for yarn user.
- If we did not set explicitly set class_id for the application user, we would check all groups returned by Hadoop's UGI, and use the class_id set for the first matched groups id.
- Otherwise, we would use default-network-tag-id.

In pseudo code it will look like the following:

```
// ...
// Map user_name -> classId
// ...
else:
// Map group_name -> classId
foreach mapping of admin_specified_group_mappings:
  if (UGI.getGroups.contains(mapping.group)):
    return mapping.classId

else:
  Use defaultClassId
```

From YARN side,

- NodeManager will read this file at startup.
- When a user submits an application and a container starts on a NodeManager, the NodeManager today already creates a cgroup for this container. To support tagging container's packets, we would modify this to also include mapping cgroup to the net_cls subsystem. Then, the users will be assigned a classid (as mentioned above, by default, we will fetch this unique classid from user-provided plaintext) that is used to tag packets originating from this cgroup.

For example:

```
mkdir /cgroup/net_cls/yarn/$container_id
echo $PID >> /cgroup/net_cls/yarn/$container_id/tasks
echo $unique-class-id > /cgroup/net_cls/yarn/$container_id/net_cls.classid
```

This whole logic will be implemented as part of the *NetworkPacketTaggingHandlerImpl*.

Other options

- We considered tag mapping in YARN to either be per container, per application or per-queue. Apps can move between queues, so queue based tagging is not feasible. Container level or application level tagging is too unstable for operators to control easily.

Configuration options

We will add a new configuration for the network isolation purpose:

- *yarn.nodemanager.resource.network.handler*. For this configuration, we can only set one of the two handlers **NetworkPacketTaggingHandlerImpl** or **TrafficControlBandwidthHandlerImpl.**
    - This is because traffic shaping is required to be per container and network packet tagging has to be per-user. And there is only one cgroup class-id that we have at our disposal.
- *Yarn.nodemanager.resource.network.tag-mapping.file-path*: Defines the location of the json file that maps users / groups to class-ids

Not Supported right now

- Changing tags of containers of a running application
- If the admin changes network-tag mapping, we need to restart NM to honor the changes. This will not affect the current running containers, it will only apply to any new containers that come to this NodeManager.