YARN Application Catalog

Design Document

Version 1.4
March 27, 2019

YARN Application Catalog

1 Overview

1.1 Requirement Description

Hadoop cluster has been designed to run analytic workload. YARN resource manager is limited to run Hadoop optimized applications. New technology such as machine learning and cloud data warehouse can benefit from Hadoop resilience without invest additional hardware and administration skillsets. This design document describes the approach to improve Hadoop eco-system for enabling docker containers to run on Hadoop cluster with minimum changes to daily administration of Hadoop clusters.

1.2 Solution Summary

YARN Application Catalog will be a new sub-project devoted to catalog and index Yarnfiles created by Yarn services. From YARN Application Catalog, user can launch multi-tiered docker containers on Hadoop cluster. The resource utilized by docker containers will be computed by YARN resource manager to ensure organization enforced limit is respected in the available resource pools.

1.3 Solution Differentiation

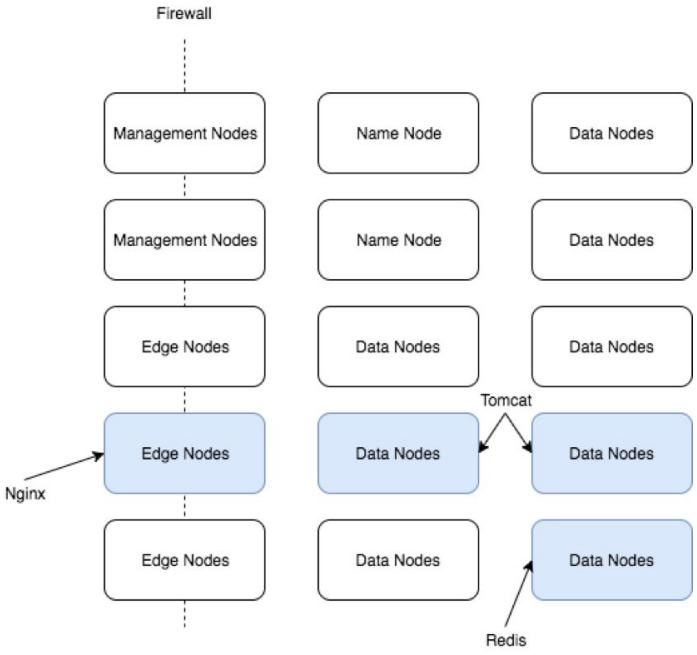
YARN Application Catalog may appear to look like existing technology such as Kubernetes or Kitematic. The common problem with existing technology is lack of persistence storage which is offered by HDFS for YARN services. When combined YARN docker support, HDFS and YARN Application Catalog, Hadoop can enable highly available infrastructure to deploy micro services at scale that is difficult to achieve with Kubernetes or Kitematic.

In the initial implementation, YARN Application Catalog will only work on Linux platform. YARN Application Catalog is target as platform as a service instead of infrastructure as a service. The advantage of YARN Appstore is to create isolation between system framework and application framework with efficient elastic parallelization in Hadoop environment. YARN Application Catalog reduces development cost for applications and provide light-weight approach to manage large scale applications.

2 External

2.1 Container based cloud provision system

Docker container can be spawn on any node manager and use nodes label to decide if the node should be an edge node or private node. Preconfigured firewall helps to isolate private network traffic from the public network, and reduce the repetitive system and network configuration required in traditional system management.



2.2 Docker Image repository on HDFS

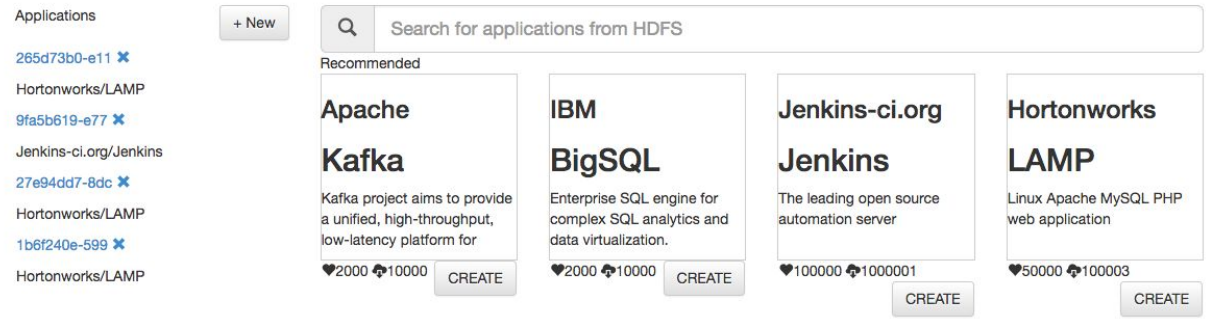
Docker Trusted Registry supports drivers to store docker images on various storage system. Docker images are immutable objects and this makes docker image storage on HDFS as an attractive alternative to local file system to improve resilience of docker images.

2.3 Docker configuration to pull image from private registry

Docker pulls image from private docker registry. The image only needs to be tagged with private registry location to enable uploading of private docker image. The syntax looks like:

```
docker tag [hostname]:[port]/[image_name]:[version]
docker push [hostname]:[port]/[image_name]:[version]
```

2.4 User interface for searchable YARN applications



A list of glossary items are:

Application Store - A central hub of applications categorized in YARN Appstore. Information about YARN applications are stored in Solr for fast searching of available applications.

Application Store Entry - Individual application information that describes the application, the docker components comprised the application.

Application List - List of deployed application.

Application Detail - Information about the per deployed instance of the application. This contains both the configuration of the application, and running status of the application. Configuration includes resource, and environment information. Status includes container ID, physical hostname and state of the deployed application.

Application Store lists recommended applications, and provide a SOLR based text box for finding applications from YARN Application Catalog. Application can be deployed to Hadoop cluster with a few clicks of buttons.

2.6 Application data are written to HDFS through NFS Client -> NFS Gateway

Docker can mount additional volume using –v flag. The data directories mounted through NFS Client can be exposed to container to write sequential data directly to HDFS.

2.7 Application Status to be reported on YARN UI2

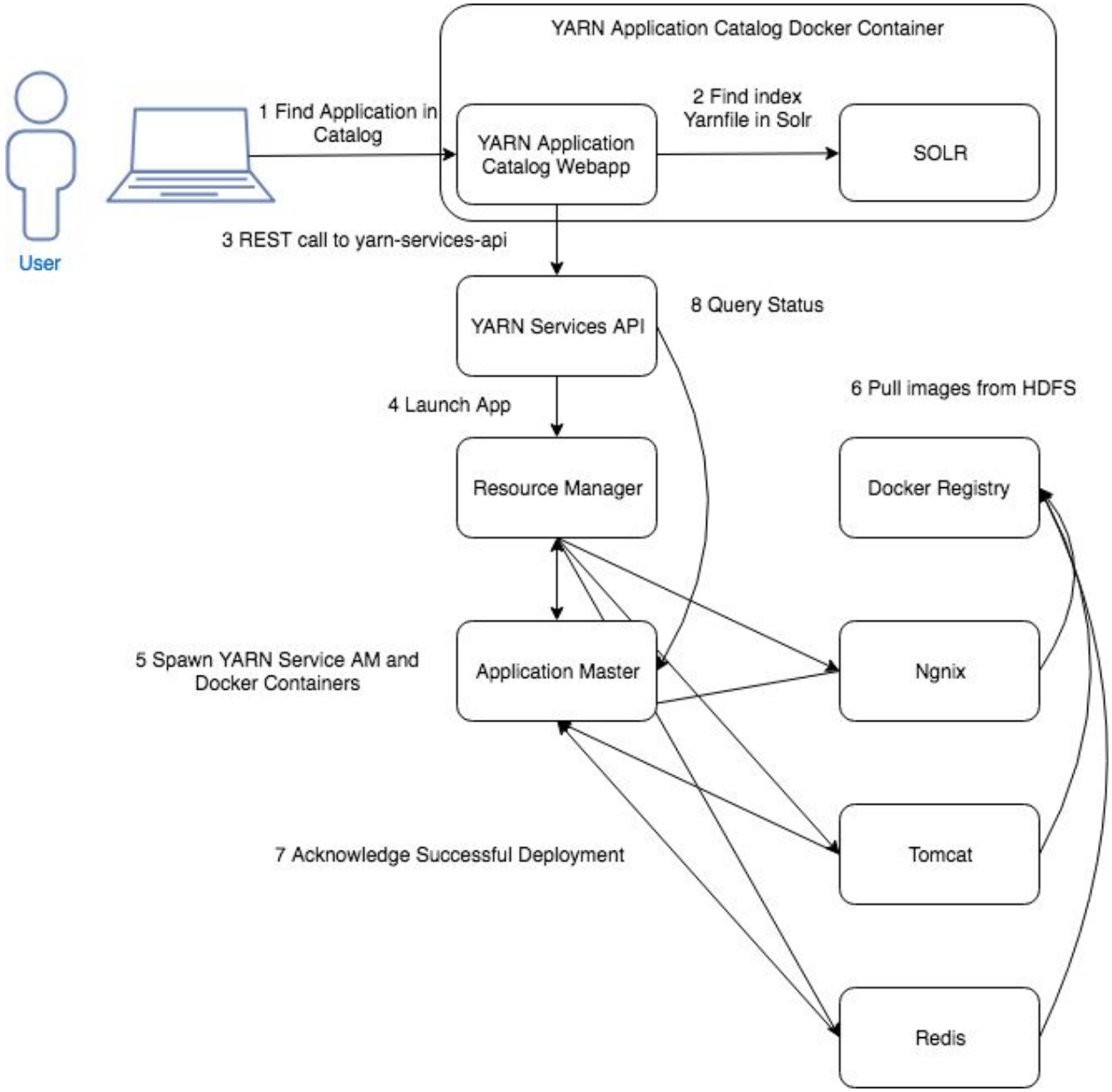
Status	Running
Started At	2017-07-11T17:23:47Z
Volumes	/hdfs/path1
Block Device IO	
Read Bps	1000MB/s
Write Bps	48MB/s
Read IOps	1024
Write IOps	23
Memory	16GB
Swap	10MB
CPU count	2
CPU Percent	50%

Docker inspect provides details of container metrics. Information such as CPU count, block IO device throughput and Memory Swappiness can be reported to Resource Manager.

2.8 Security

Docker images will be running in non-privileged mode only. The file system access credential maps directly to the user who spawn container applications. Kerberos credential is presented by NFS gateway to namenode to verify the service user. OS will verify the container user using standard Linux file system access.

3 Internal



3.1 YARN Application Catalog

A Searchable catalog can be developed to find available YARN applications hosted in HDFS. YARN applications are descriptor files that describe Docker images, and list of parameters to set for docker images. This REST API server provides detailed information about the image

name, version number, description, number of pulls, favors, and last updated status. Any YARN application uploaded to Application Catalog can be deployed by cluster users.

3.1.1 Docker Registry

Docker Inc provides Docker Registry available for third party to modify and contribute. Docker registry can be used to deposit private docker images on HDFS, and use NFS gateway to proxy docker images to Docker Registry. This enables to serve docker images from HDFS without code modification to docker registry.

Run docker registry:

```
docker run -d -p 5000:5000 --name registry -v
/mnt/hdfs/apps/docker:/var/lib/registry registry:2
```

The registry storage can be mounted from NFS mount point. See section 3.4 for mounting /mnt/hdfs on all nodes.

3.2 Application Catalog User Interface

Application Catalog runs as a YARN service with web application server and Solr in a Docker container. The standard features are:

1. Search for YARN applications
2. Deploy YARN applications
3. Delete YARN applications
4. Monitor running YARN application instances
5. Upgrade YARN applications

3.3 YARN Service Integration

In Hadoop 3, YARN descriptor files called Yarnfile describes the relationship between docker containers to deploy in Hadoop environment. YARN Application can register Yarnfile with YARN Application Catalog to provide catalog for indexing YARN applications. YARN Application Catalog utilize Yarn services API REST API to deploy, check the status of application deployed in YARN.

3.3.1 Nodemanager UI Integration

YARN UI provide the real time information about the deployment of the docker containers. YARN Application Catalog leverages YARN internal API to obtain terminal access (YARN-8762) to docker containers.

3.3.2 YARN Service Application Master Integration

YARN Service Application Master has the real time state of the deployed service. Application Catalog retrieves service state by proxy application status request from Resource Manager API Service REST API to YARN Service Application Master.

3.4 File System Support

When HDFS is exposed with NFS like mount points, each datanode can be configured to use NFS Gateway, and use FUSE client library to expose HDFS as a local mount point. Docker container does not require to include HDFS client libraries. This improves the ability to integrate with other non-Java based applications. Docker container can be started with -v flag to map local mount point into container. This provides container to have ability to write data directly to HDFS. Ambari integration with NFS Gateway can modify /etc/fstab to include NFS mount point to include local NFS Gateway.

Local mount NFS proxy on all nodes:

```
mount -t nfs -o vers=3,proto=tcp,nolock,noacl,sync `hostname`:/
/mnt/hdfs
```

3.5 Security

3.5.1 YARN Application Registry Access Control

YARN services is designed to have user own namespace. Other users can not access application deployed by other users. Application catalog inherits the same working model because Application Catalog runs as a YARN services docker container. In single user mode, Application registration and deployment are both working as the user who started the Application Catalog. In multi-users mode, Application Catalog can register and deploy applications on behalf of the end user who submitted the request. This will involve in deeper

level of integration for YARN service REST to have capability to run impersonation at web protocol level (HADOOP-16095).

3.5.2 Docker Container UID/GID

Docker daemon control is managed through UNIX socket. When securing Docker daemon, only YARN will interact with Docker daemon control interface. Docker container is restricted to non-root user by passing -u UID:GID parameter to docker daemon control interface. This will ensure the program running in docker environment is running as designated user. This change has been proposed in YARN-4266. There should be no expose of docker command line internals to the outside world to prevent security mistakes. User defined volumes will be checked for input verification before mount points are mapped to docker images.
User process inside docker container maps to UID/GID of the host operating system. Docker container launch command includes -u=$(id -u $(whoami)):$(id -g $(whoami)), this enables the process maps directly to the same UID/GID of host operating system. This is required to ensure HDFS volume receives the same user/group membership for file system access.

3.5.3 NFS Gateway Locality Support

NFS Kerberos configuration can expose NFS export to read/writeable by local host only. NFS Gateway will be exported to all nodes, and each datanode can read/write to its own NFS Gateway. Ambari NFS Gateway configuration can be improved to reduce security exposure to unknown hosts. In hdfs-site.xml:

```
<property>
  <name>dfs.nfs.exports.allowed.hosts</name>
  <value>127.0.0.1 rw</value>
</property>
```