# Contents

1 Hopsworks Overview ............................................. 5  
   1.1 Audience .................................................. 5  
   1.2 Revision History ......................................... 6  
   1.3 What is Hopsworks? ....................................... 7  
      1.3.1 Concepts: Projects, Datasets, Users ............... 8  
      1.3.2 Unified Scale-Out Metadata ....................... 8  
      1.3.3 HopsML - Machine Learning Lifecycle .......... 8  
      1.3.4 Security ........................................... 9  
   1.4 What is Hops? ............................................. 11  

2 Getting Started ................................................. 12  
   2.1 Single Machine Installation ............................. 12  
      2.1.1 Importing an ova image ............................ 12  
      2.1.2 Deploying a Hopsworks vm with Karamel .......... 13  
   2.2 Hopsworks Amazon Machine Image (AMI) ............... 13  
      2.2.1 Going further ..................................... 16  

3 Installation ...................................................... 19  
   3.1 Hops Manual Installation Guide .......................... 19  
      3.1.1 Purpose and Overview ............................. 19  
      3.1.2 Required Softwares ............................... 19  
      3.1.3 Download and Compile Sources .................. 19  
      3.1.4 Building the DAL Driver .......................... 20  
      3.1.5 Building the GPU Driver .......................... 20  
      3.1.6 Building Hops ...................................... 20  
      3.1.7 Installing the Backend Database ................. 21  
      3.1.8 Hops Cluster Setup ................................ 21  
      3.1.9 Configuring NameNodes ............................. 21  
      3.1.10 Configuring DataNodes ............................ 22  
      3.1.11 Configuring HDFS Clients ........................ 22  
   3.2 Cloud Platforms (AWS, GCE, OpenStack) ............... 23
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.1</td>
<td>Karamel/Chef</td>
<td>23</td>
</tr>
<tr>
<td>3.3</td>
<td>On-Premises (baremetal) Installation</td>
<td>25</td>
</tr>
<tr>
<td>3.4</td>
<td>Vagrant (Virtualbox)</td>
<td>27</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Default Setup</td>
<td>28</td>
</tr>
<tr>
<td>3.5</td>
<td>Windows</td>
<td>29</td>
</tr>
<tr>
<td>3.6</td>
<td>Apple OSX/Mac</td>
<td>29</td>
</tr>
<tr>
<td>3.7</td>
<td>Hopsworks Chef Cookbooks</td>
<td>30</td>
</tr>
<tr>
<td>3.8</td>
<td>System Requirements</td>
<td>31</td>
</tr>
<tr>
<td>3.8.1</td>
<td>Recommended Setup</td>
<td>31</td>
</tr>
<tr>
<td>3.8.2</td>
<td>Entire Hops platform on a single baremetal machine</td>
<td>32</td>
</tr>
<tr>
<td>3.8.3</td>
<td>Entire Hops platform on a single virtualbox instance (vagrant)</td>
<td>32</td>
</tr>
<tr>
<td>3.8.4</td>
<td>DataNode and NodeManager</td>
<td>32</td>
</tr>
<tr>
<td>3.8.5</td>
<td>NameNode, ResourceManager, NDB Data Nodes, Hopsworks, and ElasticSearch</td>
<td>33</td>
</tr>
<tr>
<td>3.9</td>
<td>Version upgrades instructions</td>
<td>33</td>
</tr>
<tr>
<td>3.9.1</td>
<td>Version 0.7.0</td>
<td>34</td>
</tr>
<tr>
<td>3.9.2</td>
<td>Version 0.10</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>User Guide</td>
<td>38</td>
</tr>
<tr>
<td>4.1</td>
<td>Hopsworks User Guide</td>
<td>38</td>
</tr>
<tr>
<td>4.1.1</td>
<td>First Login (without 2-Factor Authentication)</td>
<td>38</td>
</tr>
<tr>
<td>4.1.2</td>
<td>First Login with 2-Factor Authentication</td>
<td>40</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Register a New Account on Hopsworks</td>
<td>40</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Forgotten Password / Lost Smartphone</td>
<td>42</td>
</tr>
<tr>
<td>4.1.5</td>
<td>Update your Profile/Password</td>
<td>42</td>
</tr>
<tr>
<td>4.1.6</td>
<td>If it goes wrong</td>
<td>42</td>
</tr>
<tr>
<td>4.1.7</td>
<td>Create a New Project</td>
<td>43</td>
</tr>
<tr>
<td>4.1.8</td>
<td>Delete a Project</td>
<td>44</td>
</tr>
<tr>
<td>4.1.9</td>
<td>Datasets</td>
<td>44</td>
</tr>
<tr>
<td>4.1.10</td>
<td>Data Set Browser</td>
<td>44</td>
</tr>
<tr>
<td>4.1.11</td>
<td>Upload Data</td>
<td>45</td>
</tr>
<tr>
<td>4.1.12</td>
<td>Compress Files</td>
<td>45</td>
</tr>
<tr>
<td>4.1.13</td>
<td>Share a Data Set</td>
<td>45</td>
</tr>
<tr>
<td>4.1.14</td>
<td>Free-text Search</td>
<td>45</td>
</tr>
<tr>
<td>4.1.15</td>
<td>Jobs</td>
<td>46</td>
</tr>
<tr>
<td>4.1.16</td>
<td>Workflows</td>
<td>46</td>
</tr>
<tr>
<td>4.1.17</td>
<td>Logs visualization</td>
<td>47</td>
</tr>
<tr>
<td>4.1.18</td>
<td>Job metrics</td>
<td>48</td>
</tr>
<tr>
<td>4.1.19</td>
<td>Jupyter Notebooks</td>
<td>48</td>
</tr>
<tr>
<td>4.1.20</td>
<td>Apache Kafka</td>
<td>51</td>
</tr>
<tr>
<td>4.1.21</td>
<td>Apache Hive</td>
<td>54</td>
</tr>
<tr>
<td>4.1.22</td>
<td>Metadata Management</td>
<td>58</td>
</tr>
<tr>
<td>4.1.23</td>
<td>MetaData Designer</td>
<td>58</td>
</tr>
<tr>
<td>4.1.24</td>
<td>MetaData Attachment and Entry</td>
<td>58</td>
</tr>
<tr>
<td>4.1.25</td>
<td>Feature Store</td>
<td>59</td>
</tr>
<tr>
<td>4.1.26</td>
<td>Apache Airflow</td>
<td>67</td>
</tr>
<tr>
<td>4.2</td>
<td>HopsFS User Guide</td>
<td>72</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Unsupported HDFS Features</td>
<td>72</td>
</tr>
<tr>
<td>4.2.2</td>
<td>NameNodes</td>
<td>73</td>
</tr>
</tbody>
</table>
CHAPTER 1

Hopsworks Overview

1.1 Audience

This document contains different guides for installation, users, administration, and developers. For the following different types of readers, we recommend reading the guides:

- Data Scientists and Data Engineers
  - HopsML (Machine Learning Guide)
  - User Guide
- Administrators
  - Installation Guide
  - Administration Guide
- Hopsworks Developers
  - Installation Guide
  - User Guide
  - Developer Guide
## 1.2 Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Release</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nov 2015</td>
<td>0.1</td>
<td>First release of Hops Documentation.</td>
</tr>
<tr>
<td>Jan 2017</td>
<td>0.3.0</td>
<td>Updated version of Hops and Hopsworks.</td>
</tr>
<tr>
<td>Jan 2018</td>
<td>0.4.0</td>
<td>Update installation with vagrant and add documentation for new services</td>
</tr>
<tr>
<td>Sep 2018</td>
<td>0.5.0</td>
<td>Update Hops APIs and installation with AWS</td>
</tr>
<tr>
<td>Nov 2018</td>
<td>0.6.0</td>
<td>Updated background Hopsworks, APIs and services</td>
</tr>
</tbody>
</table>
1.3 What is Hopsworks?

Fig. 1: Hopsworks unifies a number of open-source analytics and ML frameworks behind a unified REST API.

Hopsworks is a managed platform for scale-out data science, with support for both GPUs and Big Data, in a familiar development environment. Hopsworks can be used either through its User-Interface or via a REST API. Hopsworks unique features are:

• a user-friendly UI for development with the latest open-source platforms for Data Science (Jupyter, Conda, etc),

• Github-like Projects to manage teams/products/workflows/data,

• managed GPUs as a Resources - scale out Deep Learning training and hyperparameter optimization,

• the world’s fastest, most-scalable distributed hierarchical filesystem,

• a REST API for the whole Hopsworks Platform,

• a TLS Certificate based security model with extensive auditing and data provenance capabilities,

• end-to-end support for Python-based Deep Learning workflows with: a Feature Store, Data and Model Validation, Model Serving on Kubernetes, workflow orchestration in Airflow.

Hopsworks supports the following open-source platforms for Data Science:

• development: Jupyter, plugin to IDEs (vi the REST API), Conda/Pip;

• machine learning frameworks: TensorFlow, Keras, PyTorch, ScikitLearn;
• data analytics and BI: SparkSQL, Hive;
• stream processing: Spark streaming, Flink, Kafka;
• model serving: Kubernetes/Docker.

1.3.1 Concepts: Projects, Datasets, Users

Hopsworks provides a new stronger, GDPR-compliant security model for managing sensitive data in a shared data platform. Hopsworks’ security model is built around Projects, which are analogous to Github repositories. A project contains datasets, users, and programs (code). Sensitive datasets can be sandboxed inside a project, so that users are prevented from exporting that data from the project or cross-linking that data with data in other projects. Note, this capability is provided in competitor data platforms by creating a whole new cluster for the sensitive dataset. In Hopsworks, sharing data does not involve copying data. Datasets can still be securely shared between projects, without the need for duplicating the dataset. Supported datasets in Hopsworks include Hive databases, Kafka topics, and subtrees in HopsFS (HDFS). Hopsworks implements its project-based multi-tenancy security model by supporting TLS certificates (instead of Kerberos) for user authentication, with a new certificate created for every use in every project. Hopsworks also provides role-based access control within projects, with pre-defined DataOwner and Data-Scientists roles provided for GDPR compliance (Data owners are responsible for the data and access to the data, while Data Scientists are processors of the data).

![Diagram](https://via.placeholder.com/150)

Fig. 2: Just like Github is made up of repositories, Hopsworks is made up of lots of Projects. A Project is, in turn, a collection of users, data assets, and programs (code).

1.3.2 Unified Scale-Out Metadata

Hopsworks is enabled by a unified, scale-out metadata layer - a strongly consistent in-memory data layer that stores metadata for everything from Projects/Users/Datasets in Hopsworks, Filesystem metadata in HopsFS, Kafka ACLs, and YARN quota information. Hopsworks’ metadata layer is kept consistent by mutating it using transactions and its integrity is ensured using foreign keys.

1.3.3 HopsML - Machine Learning Lifecycle

Hopsworks provides HopsML as a set of services and platforms to support the full machine learning lifecycle, including:
Fig. 3: Projects also have quotas associated with them - CPU/GPU and how much data they can store.

Fig. 4: Hopsworks includes open-source frameworks for scalable data science in a single, secure platform.

- data management with HopsFS, Hive, Kafka, and Elasticsearch;
- training machine learning models on both GPUs and CPUs, including distributed training on GPUs;
- serving of models in production using Kubernetes, with Hopsworks providing authorized, audited access to scale-out models on TensorFlowServing, SparkML, or ScikitLearn;
- model management and monitoring with a Spark Streaming application analyzing model usage in near-realtime.

1.3.4 Security

Hopsworks’ security model is designed to support the processing of sensitive Datasets in a shared (multi-tenant) cluster. The solution is based on Projects. Within a Project, a user may have one of two different roles, a **Data Owner** - who is like a superuser, and a **Data Scientist** - who is allowed to run programs (do analysis), but not allowed to:

- copy data either in or out of the Project,

1.3. What is Hopsworks?
• cross-link the data in the Project with data in other Projects (even if she is a member of the other projects).

That is, the Project acts like a sandbox for the data within it.

To realize this security model, Hopsworks implements dynamic role-based access control for projects. That is, users do not have static global roles. A user’s privileges depend on what the user’s active project is. For example, the user may be a Data Owner in one project, but only a Data Scientist in another project. Depending on which project is active, the user may be a Data Owner or a Data Scientist. The Data Owner role is strictly a superset of the Data Scientist role - everything a Data Scientist can do, a Data Owner can do.

A Data Scientist can

• run applications (Jobs, Jupyter)
• upload programs to a restricted number of DataSets (Resources, Jupyter)

A Data Owner can

• upload/download data to the project,
• add and remove members of the project
• change the role of project members
• create and delete DataSets
• import and export data from DataSets
• design and update metadata for files/directories/DataSets

1.4 What is Hops?

Hops is a next-generation distribution of Apache Hadoop, with a heavily adapted implementation of HDFS, called HopsFS. HopsFS is a new implementation of the Hadoop Filesystem (HDFS) based on Apache Hadoop\(^1\) 2.8, that supports multiple stateless NameNodes, where the metadata is stored in an in-memory distributed database (NDB). HopsFS enables NameNode metadata to be both customized and analyzed, because it can be easily accessed via SQL or the native API (NDB API).

HopsFS replaces HDFS’s Active-Standby Replication architecture with a set of stateless (redundant) NameNodes backed by an in-memory, shared nothing NewSQL database. HopsFS provides the DAL-API as an abstraction layer over the database, and implements a leader election protocol using the database. This means HopsFS no longer needs the following services required by Apache HDFS: quorum journal nodes, Zookeeper, and the Snapshot server.

You can read more about HopsFS in this HopsFS paper\(^2\).

---

\(^1\) http://hadoop.apache.org/releases.html
\(^2\) https://www.usenix.org/conference/fast17/technical-sessions/presentation/niazi
CHAPTER 2

Getting Started

2.1 Single Machine Installation

The easiest way to start using Hopworks locally is to import a VirtualBox appliance (ova) or let the Karamel installation software install Hopworks in a vm by running the simplesetup.sh script.

Both ways are described in detail below.

2.1.1 Importing an ova image

To successfully import and start the image, it is necessary to have VirtualBox installed on your workstation and at least 20GB of storage and 16GB of memory available. You can then follow these steps to import Hopworks:

1. Download the image\(^3\).
2. Import the image from the VirtualBox GUI or from the command line with `vboxmanage import hopsworks-0.10.0.ova`
3. Start the vm in **headless** mode from the VirtualBox GUI or from the command line with `vboxmanage startvm hopsworks0 --type headless`
4. Wait for approximately 3 minutes until all services have started and then access Hopworks at https://localhost:64724/hopsworks/ and login with username: admin@hopsworks.ai and password: admin.

For further details on how to use Hopworks, see **User Guide** (page 38).

Advanced users might want to ssh in the vm. To do this, you need to:

1. Download the ssh key\(^4\).

---

\(^3\) [http://snurran.sics.se/hops/ova/hopsworks-0.10.0.ova](http://snurran.sics.se/hops/ova/hopsworks-0.10.0.ova)

\(^4\) [http://snurran.sics.se/hops/ova/insecure_private_key](http://snurran.sics.se/hops/ova/insecure_private_key)
2. Then do `ssh -p 31989 -i insecure_private_key vagrant@localhost`

3. To view the status of all services, you can do `sudo /srv/hops/kagent/kagent/bin/status-all-local-services.sh`

4. To find a particular forwarded port, you can do `vboxmanage showvminfo hopsworks0 | grep <port>`

### 2.1.2 Deploying a Hopsworks vm with Karamel

If you are using a debian based distribution, you can run the `simplesetup.sh` script. (Tested on Ubuntu 18.04 LTS)

```
./simplesetup.sh
```

It is required that vagrant, virtualbox and chefdk are already installed on the host machine. If you run

```
./simplesetup.sh --install-deps
```

the script will automatically download and install the required versions of these dependencies.

After installing the dependencies, it will create and deploy the VM (needs ~16GB of RAM). This operation might take up to 1 hour, depending on your host machine specs.

To trace execution progress, tail the `nohup` file.

```
tail -f karamel-chef/nohup
```

The script will output the port on which you can access Hopsworks when installation is completed and the demo user credentials.

If you want to destroy your VM, run the kill script

```
./karamel-chef/kill.sh
```

### Going further

For detailed instructions on how to perform production deployments in-house or in the cloud, see *Installation* (page 19).

### 2.2 Hopsworks Amazon Machine Image (AMI)

We provide an Amazon Machine Image (AMI) of the latest version of Hopsworks. To start using the AMI log into your AWS account⁵ (or create a new one). The AMI is currently available in the London and Ohio regions.

Once you are logged in, click on *Launch Instance* to launch a machine instance based on the AMI.

---

⁵ https://eu-west-2.console.aws.amazon.com/ec2/v2/home?region=eu-west-2#Home:
Step 1 requires you to select the AMI. Select Community AMIs (1) and search for hopsworks (2). Click select on the desired version (3) to progress with the launch.

In step 2 you have to select which instance type you want to use. Currently the AMI is not configured to use GPUs. So you can run the AMI on instances of type $T$ (general purpose) or $C$ (compute optimized). Make sure the instance you launch has at least 16 GB of memory and 4 vCPUs available. We suggest using either the $t2.xlarge$ or the $t3.xlarge$. Select the type and proceed with the instance configuration.

Click on Next: Configure Instance Details to proceed to step 3.

Step 3 requires you to configure the instance details. You can leave the default values and proceed with configuring storage.

In step 4 you will be asked to configure the disk space. You can change the size of the Root volume as you wish, however we recommend you configure the volume with at least 40 GB of space.

In step 5 you can add tags to your instance. You can skip this step.

In step 6 you need to configure the security group. Make sure you allow incoming traffic from outside on both port 22 (for SSH connections) and on port 443 (for HTTPS connections). Click Add Rule and select HTTPS from the drop-down menu.

Once you are done with the configuration, you can launch you instance by clicking Review and Launch and finally on Launch.

In this final step AWS will ask you which key you want to use to SSH into the machine. Create a new pair (1), give it a name (2) and download the key (3). Finally click Launch Instances (4)

As the popup says, you won’t be able to re-download the same key in the future.

Please refer to the AWS documentation for more information on how to launch an EC2 instance:


2.2. Hopsworks Amazon Machine Image (AMI)
The instance is now starting. Go back to the instance panel (Instances entry in the left panel) to check the launch progress.

Once the status is *running* you can SSH into the machine using the Public IP you find on the instances page. It’s likely that the key you just downloaded has the wrong permission. Set the permission of the key file to 400 with the following command:

```
chmod 400 <demo.pem>  #Replace demo.pem with your key name
```

After that you can ssh into the instance:

```
ssh -i <key path> ubuntu@<public ip>  #Replace demo.pem with your key name
```

If you are on Windows you can use PuTTY\(^7\).

Once you log into the instance, you will find in the home directory of the ubuntu user (/home/ubuntu) a script called `start-services.sh`. Run the script.

```
./start-services.sh
```

The script will ask you to type in the email to register the instance and will start all the services (this step might take few minutes).

Once all the services are up and running, open the browser and visit `https://<public_ip>/hopsworks` where `<public_ip>` is the public ip of your instance.

You can log into Hopsworks using the following credentials:

- username : admin@hopsworks.ai
- password : admin

---

2.2.1 Going further

You can now start using the platform! For more information on the different services the platform provides see Hopworks User Guide (page 38)

For more complex installations (i.e. GPU support, multi-machine setup) refer to the Installation (page 19) page.
2.2. Hopsworks Amazon Machine Image (AMI)
2.2. Hopsworks Amazon Machine Image (AMI)

**Fig. 9: Create and download the SSH key**

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

**Note:** The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

1. **Create a new key pair**
2. **Key pair name:** `demo`
3. **Download Key Pair**

You have to download the **private key file** (*.pem file) before you can continue. **Store it in a secure and accessible location.** You will not be able to download the file again after it's created.

4. **Launch Instances**

**Fig. 10: Instances panel**
3.1 Hops Manual Installation Guide

3.1.1 Purpose and Overview

All applications running on HDFS and YARN can easily migrate to HopsFS and HopsYARN, as both HopsFS and HopsYARN supports same client facing APIs as HDFS and YARN. Setting up HopsFS is similar to HDFS except HopsFS allows multiple NameNodes that store the metadata in an external database. Similarly, HopsYARN supports a multiple ResourceManagers, although internally there will be a leader that acts as the scheduler while other ResourceManagers will act as ResourceTrackers that handle communications with NodeManagers.

Hops can be installed using Karamel\(^8\), an orchestration engine for Chef Solo, that enables the deployment of arbitrarily large distributed systems on both virtualized platforms (AWS, Vagrant) and bare-metal hosts (see Hops Auto Installer (page 19) for more details). This document serves as starting point for manually installing and configuring Hops.

3.1.2 Required Softwares

Ensure that Java 1.7.X and Protocol Buffers 2.5 are installed.

3.1.3 Download and Compile Sources

Hops consists of two modules:

- the Data Access Layer (DAL) and its implementation (for a target database such as MySQL Cluster).

\(^8\) http://www.karamel.io/
• the GPU abstraction layer and its implementation (for NVIDIA GPUs).

• Hops Hadoop;

Separating the data access layer permits different (distributed) transactional database storage engines with different licensing models. More… (page 149)

### 3.1.4 Building the DAL Driver

Download the source code for Data Access Layer Interface:

```bash
> git clone https://github.com/hopshadoop/hops-metadata-dal
> cd hops-metadata-dal
> mvn install
```

Download the source code for Data Access Layer Implementation for MySQL Cluster Network Database (NDB):

```bash
> git clone https://github.com/hopshadoop/hops-metadata-dal-impl-ndb
> cd hops-metadata-dal-impl
> mvn install
```

This generates a driver jar file ./target/hops-metadata-dal-impl-ndb-1.0-SNAPSHOT.jar which is used by the HopsFS to communicate with the database.

### 3.1.5 Building the GPU Driver

Download the source code for the GPU Layer Interface:

```bash
> git clone https://github.com/hopshadoop/hops-gpu-management.git
> cd hops-gpu-management
> mvn install -DskipTests
```

Download the source code for the GPU layer Implementation for NVIDIA:

```bash
> git clone https://github.com/hopshadoop/hops-gpu-management-impl-nvidia.git
> cd hops-gpu-management-impl-nvidia
> mvn install -DskipTests
```

### 3.1.6 Building Hops

Download the source code for Hops:

```bash
> git clone https://github.com/hopshadoop/hops
> cd hops
> mvn clean install generate-sources package -Pdist,ndb,native -Dtar
```

This generates a hadoop distribution folder ./hadoop-dist that uses Hops instead of Hadoop.
3.1.7 Installing the Backend Database

Hops uses NDB to store the filesystem metadata. NDB can be installed using Karamel\(^9\). Karamel comes with many sample installation recopies for NDB that can be found in the examples folder of the Karamel installation. Instructions for manually installing NDB is out of the scope of this documentation. We refer you to the official NDB Installation Manual\(^10\) for installing NDB.

3.1.8 Hops Cluster Setup

Installation involves copying the hadoop-dist folder on all the machines in the cluster. Ensure that all the machines have Java 1.7.X or higher installed.

Configuring Hops in Non-Secure Mode

Hops consist of the following types of nodes: NameNodes, DataNodes, ResourceManagers, NodeManagers, and Clients. All the configurations parameters are defined in core-site.xml (common for HopsFS and HopsYARN), hdfs-site.xml (HopsFS), erasure-coding-site.xml (for erasure code) and yarn-site.xml (HopsYARN) files.

Currently Hops only supports non-secure mode of operations. In the following sections we will discuss how to configure the different types of nodes. As Hops is a fork of the Hadoop code base, most of the Hadoop configuration parameters\(^11\) are supported in Hops. In this section we highlight only the new configuration parameters and the parameters that are not supported due to different metadata management scheme.

3.1.9 Configuring NameNodes

HopsFS supports multiple NameNodes. A NameNode is configured as if it is the only NameNode in the system. Using the database a NameNode discovers all the existing NameNodes in the system. One of the NameNodes is declared the leader for housekeeping and maintenance operations.

All the NameNodes in HopsFS are active. Secondary NameNode and Checkpoint Node configurations are not supported. See section (page 72) for detail list of configuration parameters and features that are no longer supported in HopsFS.

For each NameNode define fs.defaultFS configuration parameter in the core-site.xml file. In order to load NDB driver set the dfs.storage.driver.* parameters in the hdfs-site.xml file. These parameter are defined in detail here (page 118).

A detailed description of all the new configuration parameters for leader election, NameNode caches, distributed transaction handling, quota management, id generation and client configurations are defined here (page 112).

The NameNodes are started/stopped using the following commands:

\(^9\) http://www.karamel.io/
\(^11\) http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml
> $HADOOP_HOME/sbin/hadoop-daemon.sh --script hdfs start namenode
> $HADOOP_HOME/sbin/hadoop-daemon.sh --script hdfs stop namenode

### Formating the Cluster

Running the format command on any NameNode truncates all the tables in the database and inserts default values in the tables. NDB atomically performs the truncate operation which can fail or take very long time to complete for very large tables. In such cases run the `/hdfs namenode -dropAndCreateDB` command first to drop and recreate the database schema followed by the format command to insert default values in the database tables. In NDB dropping and recreating a database is much quicker than truncating all the tables in the database.

See section (page 22) for instructions for formatting the filesystem.

### 3.1.10 Configuring DataNodes

HopsFS DataNodes configuration is identical to HDFS DataNodes. In HopsFS a DataNode connects to all the NameNodes. Make sure that the `fs.defaultFS` parameter points to valid NameNode in the system. The DataNode will connect to the NameNode and obtain a list of all the active NameNodes in the system, and then connects/registers with all the NameNodes in the system.

The datanodes are started/stopped using the following commands:

> $HADOOP_HOME/sbin/hadoop-daemon.sh --script hdfs start datanode
> $HADOOP_HOME/sbin/hadoop-daemon.sh --script hdfs stop datanode

### 3.1.11 Configuring HDFS Clients

In HDFS the client connects to the `fs.defaultFS` NameNode. In HopsFS the client obtains the list of active NameNodes from the NameNode defined using `fs.defaultFS` parameter. The client then uniformly distributes the subsequent filesystem operations among the list of NameNodes.

In `core-site.xml` we have introduced a new parameter `dfs.namenodes.rpc.addresses` that holds the rpc address of all the NameNodes in the system. If the NameNode pointed by `fs.defaultFS` is dead then the client tries to connect to a NameNode defined by the `dfs.namenodes.rpc.addresses`. As long as the NameNode addresses defined by the two parameters contain at least one valid address the client is able to communicate with the HopsFS. A detailed description of all the new client configuration parameters are here (page 117).

HopsFS clients are invoked in an identical manner to HDFS:

> $HADOOP_HOME/bin/hdfs {parameters}
> $HADOOP_HOME/bin/hadoop dfs {parameters}
3.2 Cloud Platforms (AWS, GCE, OpenStack)

Hops can be installed on a cloud platform using Karamel/Chef.

3.2.1 Karamel/Chef

2. Run Karamel.
3. Click on the “Load Cluster Definition” menu item in Karamel. You are now prompted to select a cluster definition YAML file. Go to the examples/stable directory, and select a cluster definition file for your target cloud platform for one of the following cluster types:
   1. Amazon Web Services EC2 (AWS)
   2. Google Compute Engine (GCE)
   3. OpenStack
   4. On-premises (bare metal)

For more information on how to configure cloud-based installations, go to help documentation at http://www.karamel.io. For on-premises installations, we provide some additional installation details and tips later in this section.

Choosing which services to run on which nodes

You now need to decide which services you will install on which nodes. In Karamel, we design a set of Node Groups, where each Node Group defines a stack of services to be installed on a machine. Each machine will only have one Node Group set of services. We now provide two recommended setup:

• a single node cluster that includes all services on a single node.
• a tiny cluster set of heavy stacks that includes a lot of services on each node.
• a small cluster set of heavy stacks that includes lots of services on each node.
• a large cluster set of light stacks that includes fewer services on each node.

Single Node Setup You can run the entire Hopsworks application platform on a single node. You will have a NodeGroup with the following services on the single node:

1. Hopsworks, Elasticsearch, MySQL Server, NDB Mgmt Server, HDFS NameNode, YARN ResourceManager, NDB Data Node(s), HDFS DataNode, YARN NodeManager

Tiny Cluster Setup

We recommend the following setup that includes the following NodeGroups, and requires at least 2 nodes to be deployed:
1. Hopsworks, Elasticsearch, MySQL Server, NDB Mgmt Server, HDFS NameNode, YARN Resource-
   Manager, NDB Data Node

2. HDFS DataNode, YARN NodeManager

This is really only a test setup, but you will have one node dedicated to YARN applications and file storage,
while the other node handles the metadata layer services.

Small Cluster Setup

We recommend the following setup that includes four NodeGroups, and requires at least 4 nodes to be
deployed:

1. Hopsworks, Elasticsearch, MySQL Server, NDB Mgmt Server,
2. HDFS NameNode, YARN ResourceManager, MySQL Server
3. NDB Data Node
4. HDFS DataNode, YARN NodeManager

A highly available small cluster would require at least two instances of the last three NodeGroups.
Hopsworks can also be deployed on multiple instances, but Elasticsearch needs to be specially configured if
it is to be sharded across many instances.

Large Cluster Setup

We recommend the following setup that includes six NodeGroups, and requires at least 4 nodes to be de-
ployed:

1. Elasticsearch
2. Hopsworks, MySQL Server, NDB Mgmt Server
3. HDFS NameNode, MySQL Server
4. YARN ResourceManager, MySQL Server
5. NDB Data Node
6. HDFS DataNode, YARN NodeManager

A highly available large cluster would require at least two instances of every NodeGroup. Hopsworks can
also be deployed on multiple instances, while Elasticsearch needs to be specially configured if it is to be
sharded across many instances. Otherwise, the other services can be easily scaled out by simply adding
instances in Karamel. For improved performance, the metadata layer could be deployed on a better network
(10 GbE at the time of writing), and the last NodeGroup (DataNode/NodeManager) instances could be
deployed on cheaper network infrastructure (bonded 1 GbE or 10 GbE, at the time of writing).

Hopsworks Configuration in Karamel

Karamel Chef recipes support a large number of parameters that can be set while installing Hops. These
parameters include, but are not limited to,:

- usernames to install and run services as,
- usernames and passwords for services, and
- sizing and tuning configuration parameters for services (resources used, timeouts, etc).
Here are some of the most important security parameters to set when installing services:

- Superuser username and password for the MySQL Server(s)
  - Default: ‘kthfs’ and ‘kthfs’
- Administration username and password for the Glassfish administration account(s)
  - Default: ‘adminuser’ and ‘adminpw’
- Administration username and password for Hopsworks
  - Default: ‘admin@hopsworks.ai’ and ‘admin’

Here are some of the most important sizing configuration parameters to set when installing services:

- DataMemory for NDB Data Nodes
- YARN NodeManager amount of memory and number of CPUs
- Heap size and Direct Memory for the NameNode
- Heap size for Glassfish
- Heap size for Elasticsearch

### 3.3 On-Premises (baremetal) Installation

For on-premises (bare-metal) installations, you will need to prepare for installation by:

1. identifying a master host, from which you will run Karamel;
   1. the master must have a display for Karamel’s user interface;
   2. the master must be able to ping (and connect using ssh) to all of the target hosts.
2. identifying a set of target hosts, on which the Hops software and 3rd party services will be installed.
   1. the target nodes should have http access to the open Internet to be able to download software during the installation process. (Cookbooks can be configured to download software from within the private network, by changing the ‘download_url’ chef attribute to a URL to a local http server IP address).
   
   The master must be able to connect using SSH to all the target nodes, on which the software will be installed.

   If you have not already copied the master’s public key to the .ssh/authorized_keys file of all target hosts, you can do so by preparing the machines as follows:

   1. Create an openssh public/private key pair on the master host for your user account. On Linux, you can use the ssh-keygen utility program to generate the keys, which will by default be stored in the $HOME/.ssh/id_rsa and $HOME/.ssh/id_rsa.pub files. If you decided to enter a password for the ssh keypair, you will need to enter it again in Karamel when you reach the ssh dialog, part of Karamel’s Launch step.
   2. Create a user account USER on the all the target machines with full sudo privileges (root privileges) and the same password on all target machines.

12 ‘admin@hopworks.ai’
3. Copy the $HOME/.ssh/id_rsa.pub file on the master to the /tmp folder of all the target hosts. A good way to do this is to use pscp utility along with a file (hosts.txt) containing the line-separated hostnames (or IP addresses) for all the target machines. You may need to install the pssh utility programs (pssh), first.

```
$sudo apt-get install pssh
or
$yum install pssh

$vim hosts.txt
  # Enter the row-separated IP addresses of all target nodes in hosts.txt
  128.112.152.122
  18.31.0.190
  128.232.103.201
  ..... 

$parallel-scp -h hosts.txt -P PASSWORD -i USER ~/.ssh/id_rsa.pub /tmp
$parallel-ssh -h hosts.txt -i USER -P PASSWORD mkdir -p /home/USER/.ssh
$parallel-ssh -h hosts.txt -i USER -P PASSWORD "cat /tmp/id_rsa.pub >> /home/USER/.ssh/authorized_keys && rm /tmp/id_rsa.pub"
```

Update your Karamel cluster definition file to include the IP addresses of the target machines and the USER account name. After you have clicked on the launch menu item, you will come to a Ssh dialog. On the ssh dialog, you need to open the advanced section. Here, you will need to enter the password for the USER account on the target machines (sudo password text input box). If your ssh keypair is password protected, you will also need to enter it again here in the keypair password text input box.

1. Find out the network interface(s) on all servers:

These should be the same on all hosts. If they are not, you will need to add a new group in the Karamel cluster definition file for each different network interface. On each target machine, run: >ifconfig

1. Set the correct hostname

Hops uses TLS/SSL certificates for security and we generate certificates for services such as Kafka. It is important that the common name (CN) in the certificate is the same as the hostname returned by the command line operation:

```
hostname
```

In ubuntu, the hostname returned by the ‘hostname’ command is in the file /etc/hostname

```
cat /etc/hostname
```

On ubuntu versions with support for systemd, you can set the hostname with:

```
hostnamectl set-hostname hadoop1
```

You also need to update the entries in /etc/hosts. For example, if you have an ubuntu machine with an old hostname in /etc/hosts:

```
grep '127.0.1.1' /etc/hosts
127.0.1.1  vagrant1
```
You should remove this entry from /etc/hosts

If you decide to call your hosts ‘hadoop1..hadoopN’, then the hostname ‘hadoop1’ can be resolved to an IP address in /etc/hosts. This is our recommended way of resolving hostnames to IP addresses, rather than DNS which can be a source of problems if the DNS service is not working or slow or flakey.

```
grep hadoop1 /etc/hosts
>10.0.104.161 hadoop1
```

`. Hosting installation files on the local network`

Install an nginx server to the host deployment files for installation. Edit nginx’ port (do not use the default port 80, as it will clash with Hopsworks). Copy the installation files to ‘/var/www/html/software’. On Ubuntu 16.04, run the following:

```
sudo apt-get install nginx
vi /etc/nginx/sites-available/default   # Set the port to '1111'
sudo mkdir /var/www/html/software
# Copy all the software to the folder: /var/www/html/software
```

When you install using Karamel, you should set the download url to be the URL of the nginx server and its software folder. Here is an example of part of a cluster.yml:

```
name: HopsworksBaremetal
baremetal:
  username: hops
...
attrs:
  download_url: "http://192.168.0.2:1111/software"
...
```

`. Gmail Account setup`

You will need to create a new gmail account which will be used for email notifications, such as validating new Hopsworks accounts (when a user registers a new account, he/she receives an email and needs to click on a link in the email to validate his/her email address).

You need to enable gmail to send emails using a SMTP server. By default, a gmail account used by Hopsworks will not be allowed to send emails. You need to enable the following: go into the gmail account settings and ‘allow less secure apps’:

```
Allow less secure apps: ON
```

3.4 Vagrant (Virtualbox)

You can install Hopsworks and Hops on your laptop/desktop with Vagrant. You will need to have the following software packages installed:
The first step to setup the whole Hopsworks stack is to clone karamel-chef cookbook which will orchestrate the running of the other cookbooks.

```bash
git clone https://github.com/hopshadoop/karamel-chef.git
```

Then `cd karamel-chef/`. In the directory `cluster-defns/` there are some predefined cluster definitions. The number at the beginning of the file defines the number of VMs that will be created. You can use either a single VM or 3 VMs where the services will be distributed in different machines. In the directory `vagrantfiles/` are the virtual machines specifications. The name of the files contains the OS of the VM - Ubuntu or CentOS followed by the number of instances.

To start the installation you should run the `./run.sh` Bash script. If you run the script without any parameters you will get a small help message. The usage of the script is the following:

```bash
./run.sh OS NUMBER_OF_VMS NAME_OF_CLUSTER_DEFINITION [no-random-ports]
./run.sh ports
```

Where:
- `OS` is the desired operating system - ubuntu or centos.
- `NUMBER_OF_VMS` could be either 1 or 3 for the existing configurations.
- `NAME_OF_CLUSTER_DEFINITION` is the name of the cluster definition.
- By default Vagrant will forward the VMs ports to random ports in the host machine to avoid collisions. If you don’t want it provide the `no-random-ports` option.

If you want to destroy your VMs execute the `./kill.sh` script. This will wipe out the installation and the virtual machines.

### 3.4.1 Default Setup

For instance if you want to create a Hopsworks setup with 3 VMs with Ubuntu OS you would run:

```bash
./run.sh ubuntu 3 hopsworks
```

The output of the installation process is logged to `nohup.out` file. At any point you can run `./run.sh ports` to print the mapping of guest ports to host ports.

When the installation finishes you can access Hopsworks from your browser at `http://127.0.0.1:8080/hopsworks`. The default credentials are:
username: admin@hopsworks.ai
password: admin

The Glassfish web application server is also available from your browser at http://127.0.0.1:4848. The default credentials are:

username: adminuser
password: adminpw

The MySQL Server is also available from the command-line, if you ssh into the vagrant host (vagrant ssh). The default credentials are:

username: kthfs
password: kthfs

It goes without saying, but for production deployments, we recommend changing all of these default credentials. The credentials can all be changed in Karamel during the installation phase from the cluster definitions. Check the available attributes in their respective cookbooks.

3.5 Windows

You can also install Hopsworks on vagrant and Windows. You will need to follow the vagrant instructions as above (installing the same software packages) aswell as installing:

- Powershell

After cloning the github repo, from the powershell, you can run:

```
$ cd hopsworks-chef
$ berks vendor cookbooks
$ vagrant up
```

3.6 Apple OSX/Mac

First, install protobuf and cmake:

```
$ brew install protobuf250
$ brew install cmake
```

Note: you need protobuf version 2.5 (not 2.6). You can check if it’s already installed with

```
$ protoc --version
libprotoc 2.5.0
```

Hops runs on JDK 1.8, so you should have that installed (download it here). However, due to some inconsistencies between Java version you’ll have to create a symlink like this:

You should now be able to clone the repositories and build the code.

### 3.7 Hopsworks Chef Cookbooks

Hopsworks’ automated installation is orchestrated by Karamel and the installation/configuration logic is written as ruby programs in Chef. Chef supports the modularization of related programs in a unit of software, called a Chef *cookbook*. A Chef cookbook can be seen as a collection of programs, where each program contains instructions for how to install and configure software services. A cookbook may consist one or more programs that are known as *recipes*. These Chef recipes are executed by either a Chef client (that can talk to a Chef server) or chef-solo, a standalone program that has no dependencies on a Chef Server. Karamel uses chef-solo to execute Chef recipes on nodes. The benefit of this approach is that it is agentless. That is, Karamel only needs ssh to be installed on the target machine to be able to install and setup Hopsworks. Karamel also provides dependency injection for Chef recipes, supplying the parameters (*Chef attributes*) used to execute recipes. Some stages/recipes return results (such as the IP address of the NameNode) that are used in subsequent recipes (for example, to generate configuration files containing the IP address of the NameNode, such as core-site.xml).

The following is a brief description of the Chef cookbooks that we have developed to support the installation of Hopsworks. The recipes have the naming convention: <cookbook>/<recipe>. You can determine the URL for each cookbook by prefixing the name with [http://github.com/](http://github.com/). All of the recipes have been *karamelized*, that is a Karamelfile containing orchestration rules has been added to all cookbooks.

- **logicalclocks/hops-hadoop-chef**
  - This cookbook contains recipes for installing the Hops Hadoop services: HopsFS NameNode (hops::nn), HopsFS DataNode (hops::dn), YARN ResourceManager (hops::rm), YARN NodeManager (hops::nm), Hadoop Job HistoryServer for MapReduce (hops::jhs), Hadoop ProxyServer (hops::ps).

- **logicalclocks/ndb-chef**
  - This cookbook contains recipes for installing MySQL Cluster services: NDB Management Server (ndb::mgmd), NDB Data Node (ndb::ndbd), MySQL Server (ndb::mysqld), Memcached for MySQL Cluster (ndb::memcached).

- **logicalclocks/zeppelin-chef**
  - This cookbook contains a default recipe for installing Apache Zeppelin.

- **logicalclocks/hopsworks-chef**
  - This cookbook contains a default recipe for installing Hopsworks.

- **logicalclocks/spark-chef**
  - This cookbook contains recipes for installing the Apache Spark Master, Worker, and a YARN client.
3.8 System Requirements

3.8.1 Recommended Setup

We recommend either Ubuntu/Debian or CentOS/Redhat as operating system (OS), with the same OS on all machines. A typical deployment of Hops Hadoop uses

- DataNodes/NodeManagers: a set of commodity servers in a 12-24 SATA hard-disk JBOD setup;
- NameNodes/ResourceManagers/NDB-database-nodes/Hopsworks-app-server: a homogeneous set of commodity (blade) servers with good CPUs, a reasonable amount of RAM, and one or two hard-disks;
- MySQL Cluster Data nodes: a homogeneous set of commodity (blade) servers with a good amount of RAM (up to 1 TB) and good CPU(s). A good quality SATA disk is needed to store database logs. We also recommend at least 1 NVMe disk to store small files in HopsFS. These can be added later when more capacity for small files is needed.

- Hopsworks: a single commodity (blade) server with a good amount of RAM (up to 512 GB) and good CPU(s). A good quality disk is needed to store logs. Either SATA or a large SSD can be used.

For cloud platforms, such as AWS, we recommend using enhanced networking (25 Gb+) for the MySQL Cluster Data Nodes and the NameNodes/ResourceManagers. High latency connections between these machines will negatively affect system throughput.

### 3.8.2 Entire Hops platform on a single baremetal machine

You can run Hopsworks and the entire Hops stack on a bare-metal single machine for development or testing purposes, but you will need at least:

<table>
<thead>
<tr>
<th>Component</th>
<th>Minimum Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Linux, Mac</td>
</tr>
<tr>
<td>RAM</td>
<td>16 GB of RAM</td>
</tr>
<tr>
<td>CPU</td>
<td>2 GHz dual-core minimum. 64-bit.</td>
</tr>
<tr>
<td>Hard disk space</td>
<td>50 GB free space</td>
</tr>
</tbody>
</table>

### 3.8.3 Entire Hops platform on a single virtualbox instance (vagrant)

You can run Hopsworks and the entire Hops stack on a single virtualbox instance for development or testing purposes, but you will need at least:

<table>
<thead>
<tr>
<th>Component</th>
<th>Minimum Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Linux, Mac, Windows (using Virtualbox)</td>
</tr>
<tr>
<td>RAM</td>
<td>16 GB of RAM</td>
</tr>
<tr>
<td>CPU</td>
<td>2 GHz dual-core minimum. 64-bit.</td>
</tr>
<tr>
<td>Hard disk space</td>
<td>50 GB free space</td>
</tr>
</tbody>
</table>

### 3.8.4 DataNode and NodeManager

A typical deployment of Hops Hadoop installs both the Hops DataNode and NodeManager on a set of commodity servers, running without RAID (replication is done in software) in a 12-24 hard-disk JBOD setup. Depending on your expected workloads, you can put as much RAM and CPU in the nodes as needed. Configurations can have up to (and probably more) than 1 TB RAM and 48 cores.

The recommended setup for these machines in production (on a cost-performance basis) is:
### 3.8.5 NameNode, ResourceManager, NDB Data Nodes, Hopsworks, and Elastic-Search

NameNodes, ResourceManagers, NDB database nodes, ElasticSearch, and the Hopsworks application server require relatively more memory and not as much hard-disk space as DataNodes. The machines can be blade servers with only a disk or two. SSDs will not give significant performance improvements to any of these services, except the Hopsworks application server if you copy a lot of data in and out of the cluster via Hopsworks. The NDB database nodes will require free disk space that is at least 20 times the size of the RAM they use. Depending on how large your cluster is, the ElasticSearch server can be colocated with the Hopsworks application server or moved to its own machine with lower RAM and CPU requirements than the other services.

1 GbE gives great performance, but 10 GbE really makes it rock! You can deploy 10 GbE incrementally: first between the NameNodes/ResourceManagers <-> NDB database nodes to improve metadata processing performance, and then on the wider cluster.

The recommended setup for these machines in production (on a cost-performance basis) is:

<table>
<thead>
<tr>
<th>Component</th>
<th>Recommended (2018)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Linux, Mac, Windows (using Virtualbox)</td>
</tr>
<tr>
<td>RAM</td>
<td>256 GB RAM</td>
</tr>
<tr>
<td>CPU</td>
<td>Two CPUs with at least 12 cores. 64-bit.</td>
</tr>
<tr>
<td>Hard disk</td>
<td>12 x 10 TB SATA disks</td>
</tr>
<tr>
<td>Network</td>
<td>10/25 Gb/s Ethernet</td>
</tr>
</tbody>
</table>

The Hops stack can be installed on both cloud platforms and on-premises (baremetal). The recommended machine specifications given below do not take into account whether local storage is used or a cloud storage platform is used. For best performance due to improved data locality, we recommend local storage (instance storage in Amazon Web Services (AWS)/EC2).

### 3.9 Version upgrades instructions

This section contains the actions to perform manually (if any), required updates to the cluster definition and heads up of potential problems (and solutions) you might encounter when upgrading to a specific version.

Upgrades operate also on the database schemas, it is highly advised to take a backup of the database before starting the upgrade.
Hopsworks does not support online upgrades yet, so you need to shutdown all the services before starting the upgrade. You can do that from the Admin UI or by running as root user /srv/hops/kagent/kagent/bin/shutdown-all-local-services.sh on all the machines.

During an update, use the same cluster definition you used for the installation. You can modify the attributes, change location of the services, but do not remove a service from the cluster definition just because it doesn’t need to be updated. Karamel/Chef uses the location of the services in the cluster definition to template the configuration files. Removing services from the cluster definition might result in incomplete configuration files.

### 3.9.1 Version 0.7.0

Version 0.7.0 depends on Hops 2.8.2.6. This Hops version requires a change of the type of the Inode ID. This requires modifications to the cluster definition and some manual intervention.

#### Changes to the cluster definition

1. In the global `attrs` section add the current version attribute. The attribute should be set to your current installed version. Example:

```yaml
attrs:
  install:
    current_version: "0.6.1"
```

2. The upgrade requires rebuilding the Elasticsearch index for projects and datasets. Force the rebuilding by adding the following attribute to the cluster definition:

```yaml
elastic:
  projects:
    reindex: "true"
```

3. If your installation includes Hive, and you are installing it using the `hive2::default` or `hive2` entries in your recipes list, you should replace it with:

```yaml
recipes:
- hive2::db
- hive2::tez
- hive2::server2
- hive2::metastore
```

4. In version 0.7.0 we have promoted `conda` to a first-level Chef cookbook. For every nodes group in the `groups` section, add the following block in the `recipes` section. Order does not matter.

```yaml
- conda
```
Manual actions required

The Inode ID type change requires several modifications of the database schema. Before starting the upgrade, make sure you have a backup of your database. Before starting the upgrade you should disable the foreign key checks in MySQL.

```sql
set global FOREIGN_KEY_CHECKS=0;
```

Once the `hive2::db` recipe has run successfully you can proceed to re-enable the foreign key checks.

```sql
set global FOREIGN_KEY_CHECKS=1;
```

As a heads up, the setting is restored to 1 if you restart your MySQL server.

3.9.2 Version 0.10

Version 0.10 requires some changes to the cluster definition.

Changes to the cluster definition

1. In the global `attrs` section add the current version attribute. The attribute should be set to your current installed version. Example:

```json
attrs:
  install:
    current_version: "0.9.1"
```

2. Hopsworks 0.10 drops support for Zeppelin. As such, the Zeppelin cookbook is not maintained anymore. Please remove the `-zeppelin` entry from your cluster definition

3. Hopsworks 0.10 adds support for PyTorch. PyTorch is installed in the base environments and it will be available in all the newly created environments after the upgrade. PyTorch is installed from the Conda repository. If you are using an internal mirror you need to make the PyTorch channel available. Alternatively you can rely on the public mirror, but be careful about rate limiting! The attributes to specify the Conda internal mirrors have changed to:

```json
attrs:
  conda:
    channels:
      default_mirrors: "[main/free mirror location]"
      pytorch: "[pytorch mirror location]"
    use_defaults: "false"
```

4. Hopsworks 0.10 makes available to the users more logging information related to operations to the python environments. This is achieved using filebeat. In the cluster definition, on all the node groups, you should add `hopslog::filebeat-kagent` in the recipe list.

---

14 https://anaconda.org/pytorch/repo
5. Remember to add the `hopsworks::migrate` recipe in the same node group which the Hopsworks recipe belongs to. This will run `Expat` which will migrate old job configurations to the new format and create the Kibana index pattern for the Python environment logs. Migrate will also change the storage type of the `/Project` directory on HopsFS to use, whenever possible, NVME disks as storage. Migrate will not manage your old Jupyter configuration, which means already existing Jupyter configuration will be lost. Users need to manually backup this configuration.

6. To be more consistent, the attribute to specify the HTTPS port has changed to:

```yaml
attrs:
  hopsworks:
    https:
      port: 443
```

7. Add the following as an additional protection against accidental formatting of the FileSystem:

```yaml
attrs:
  hops:
    format: false
```

8. Hopsworks 0.10.0 add support to Sqoop. To install it, you can add the Sqoop recipe `-hops_airflow::sqoop` in the same group as the Hopsworks recipe.

9. In Hopsworks 0.10.0 we simplifyed the configuration for Hops TLS. Now you can just specify:

```yaml
attrs:
  hops:
    tls:
      enabled: true
      prod: true [If you want to enable certificate validation/hostname verification]
```

**FAQ**

- **Kafka brokers don’t start. How can I solve it?**

This might happen when you are running with multiple Kafka brokers. We have a bug in the Chef recipe which makes that the broker ids get mixed. You can fix the `server.properties` manually and put back the old ids.

- **I can’t reach Hopsworks on port HTTP anymore**

To improve security, we decided to disable HTTP in Hopsworks by default. If, for any reason, you want to keep using HTTP you can re-enabled it from the Glassfish admin UI which is available on port `4848` on the Hopsworks server. From there you can also setup automatic redirect to HTTPS.

The Hopsworks stack includes a number of services, but also requires the installation of a number of third-party distributed services:

- Java (OpenJDK or Oracle JRE/JDK)
- Apache Spark
Due to the complexity of installing and configuring all Hopsworks’ services, we recommend installing Hopsworks using the automated installer Karamel/Chef, http://www.karamel.io. We do not provide detailed documentation on the steps for installing and configuring all services in Hopsworks. Instead, Chef cookbooks contain all the installation and configuration steps needed to install and configure Hopsworks. The Chef cookbooks are available at https://github.com/logicalclocks.
4.1 Hopsworks User Guide

4.1.1 First Login (without 2-Factor Authentication)

On initial installation, you can login with the default username and password.

```
username: admin@hopsworks.ai
password: admin
```

Upon successful login, you will arrive on the landing page:
In the landing page, on the left you can see the Guided tours which will guide you through launching a SparkPi application. On the top center part is the search bar where you can search for public projects and datasets. On the right side, your Projects are listed where you can click and select them. Finally, on the upper right corner is your personal menu and administration panel, and the notifications menu.

**If it goes wrong**

If login does not succeed, something has gone wrong during installation. The possible sources of error and the Web Application Server (Glassfish) and the database (MySQL Clusters).

**Actions:**

- Double-check that system meets the minimum system requirements for Hopsworks. Is there enough available disk space and memory?
- Log in to Glassfish and make sure both hopsworks-ear and hopsworks-web are deployed.
- Default Glassfish credentials are:
  
  ```
  username: adminuser
  password: adminpw
  ```
- Investigate MySQL Cluster misconfiguration problems. Are the mgm server, data nodes, and MySQL server running? Do the hops and hopsworks databases exist and are they populated with tables and rows? If not, something went wrong during installation.
- Re-run the installation, as something may have gone wrong during installation.
4.1.2 First Login with 2-Factor Authentication

Hopsworks supports two factor authentication for enhanced protection. If your administrator has enabled 2 factor authentication, then during registration opt-in for it.

You should have a one-time password generator installed at your smartphone, such as Google Authenticator. After you have clicked the Register button, scan with your smartphone the QR code presented to you. Next time you login again, it will prompt you both for your password and for the six-digits one-time password generated by the application at your smartphone.

4.1.3 Register a New Account on Hopsworks

The process for registering a new account is as follows:

1. Register your email address and details and use the camera from within Google Authenticator to store your 2nd factor credential;

2. Validate your email address by clicking on the link in the validation email you received;

---

15 https://support.google.com/accounts/answer/1066447?hl=en
3. Wait until an administrator has approved your account (you will receive a confirmation email).
Register a new account with a valid email account. If you have two-factor authentication enabled, you will then need to scan the QR code to save it on your phone. If you miss this step, you will have to recover your smartphone credentials at a later stage.

In both cases, you should receive an email asking you to validate your account. The sender of the email will be either the default hopsworks@gmail.com or a gmail address that was supplied while installing Hopsworks. If you do not receive an email, wait a minute. If you still haven’t received it, you should contact the administrator.

**Validate the email address used in registration**

If you click on the link supplied in the registration email, it will validate your account. **You will not be able to login until an administrator has approved your account.**

After your account has been approved, you can now go to HopsWork’s login page and start your Google Authenticator application on your smartphone. On Hopsworks login page, you will need to enter

- the email address your registered with
- the password you registered with
- on Google Authenticator find the 6-digit number shown for the email address your registered with and enter it into Hopsworks.

### 4.1.4 Forgotten Password / Lost Smartphone

If you forget your password or lose your 2nd factor device (smartphone or yubikey), you will need to recover your credentials. On the login screen, click on **Need Help?** to recover your password or replace the QR code for your smartphone.

### 4.1.5 Update your Profile/Password

After you have logged in, in the upper right-hand corner of the screen, you will see your **email address with a caret icon**. Click on the caret icon, then click on the menu item **Account**. A dialog will pop-up, from where you can change your password and other parts of your profile. You cannot change your email address and will need to create a new account if you wish to change your email address. You can also logout by clicking on the **sign out** menu item.

### 4.1.6 If it goes wrong

Contact an administrator or go to the Administration Guide section of this document. If you are an administrator:

- Does your organization have a firewall that blocks outbound SMTP access? Hopsworks needs SMTP outbound access over TLS using SSL (port 587 or 465).
- Is the Glassfish server up and running? Can you login to the Glassfish Administration console (on port 4848)?

---

[16] If you are an administrator, you can jump now to the Hopsworks Administration Guide to see how to validate account registrations, if you have administrator privileges.
• Inside Glassfish, check the JavaMail settings. Is the gmail username/password correct? Are the SMTP server settings correct (hostname/ip, port, protocol (SSL, TLS))? 

User fails to receive an email to validate her account

• This may be a misconfigured gmail address/password or a network connectivity issue.
• Does your organization have a firewall that blocks outbound SMTP access?
• For administrators: was the correct gmail username/password supplied when installing?
• If you are not using a Gmail address, are the smtp server settings correct (ip-address or hostname, port, protocol (SSL, TLS))? 

User receives the validate-your-email message, but is not able to validate the account

• Can you successfully access the Hopsworks homepage? If not, there may be a problem with the network or the webserver may be down.
• Is the Glassfish webserver running and hopsworks-war, hopsworks-ear application installed, but you still can’t logon? It may be that MySQL Cluster is not running.
• Check the Glassfish logs for problems and the Browser logs.

User successfully validates the account, but still can’t login

The user account status may not be in the correct state, see next section for how to update user account status.

User account has been disabled due to too many unsuccessful login attempts

From the Hopsworks administration application, the administrator can re-enable the account by going to “User Administration” and taking the action “Approve account”.

User account has been disabled due to too many unsuccessful login attempts

Contact your system administrator who will re-enable your account.

4.1.7 Create a New Project

You can create a project by clicking on the New button in the Projects box. This will pop-up a dialog, in which you enter the project name, an optional description, and select an optional set of services to be used in the project. You can also select an initial set of members for the project, who will be given the role of Data Scientist in the project. Member roles can later be updated in the Project settings by the project owner or a member with the data owner role. A valid project name can not contain spaces or special characters such as __, /, \, å, ä, etc.

As soon as you have created a new project and click on it on the Projects box, you will see the project main page as illustrated in the picture below.

On the left-hand side of the project main page is the Project Menu. On the top section are the currently active services for your project such as the Job launcher UI and History service, Kafka etc In the middle section is the Data Sets browser menu where you can explore your project’s datasets. Finally, on the bottom section is various settings for the project. From the Settings menu you can modify the description of the project, the data retention period and see some statistics. From the Members menu
you can add new members to your project and share your datasets or remove existing ones. Using the Metadata Designer you can attach more intuitive metadata to your project. Also, in the project’s menu you can always see the current cluster utilization.

4.1.8 Delete a Project

Right click on the project to be deleted in the projects box. You have the options to:
- Remove and delete data sets;
  - If the user deletes the project, the files are moved to trash in HopsFS;
- Remove and keep data sets.

4.1.9 Datasets

A Dataset is a subtree of files and directories in HopsFS. Every Dataset has a home project, and by default can only be accessed by members of that project. A Data Owner for a project may choose to share a dataset with another project or make it public within the organization.

When you create a new Project, by default, a number of Datasets are created:
- **Resources**: should be used to store programs and resources needed by programs. Data Scientists are allowed upload files to this dataset.
- **Logs**: contains outputs (stdout, stderr) for applications run in Hopsworks.
- **Jupyter**: contains Jupyter notebook files. Data Scientists are allowed upload files to this dataset.
- **Experiments**: contains runs for experiments launched using the HopsML API in PySpark/TensorFlow/Keras/PyTorch.
- **Models**: contains trained machine learning model files ready for deployment in production.

  **Hive** databases are also Datasets in Hopsworks - the Hive database’s datafiles are stored in a HopsFS subtree. As such, Hive Databases can be shared between Projects, just like Datasets. **Feature Stores** are also stored in a HopsFS subtree, and can also be shared between Projects.

4.1.10 Data Set Browser

The Data Set tab enables you to browse Data Sets, files and directories in this project. It is mostly used as a file browser for the project’s HDFS subtree. You cannot navigate to directories outside of this project’s subtree. For a quick preview of a file, go to the Datasets menu, navigate to a file, right click on that file and choose the Preview option. A pop-up window will appear with a small preview of the file. The picture below illustrates the Dataset Browser with a new sample dataset. You can add new datasets pressing the Create New Dataset button. You can edit the datasets by right clicking on them. A README file is auto-generated for every dataset.
4.1.11 Upload Data

Files can be uploaded using Hopsworks’ web interface. Go to the project you want to upload the file(s) to. You must have the Data Owner role for that project to be able to upload files. In the Data Sets tab, on the top left corner there is the Upload button.

After pressing on the Upload button, the following window will appear which will let you select the files or folders from your local hard drive by clicking on Upload File or Upload Folder. Next step is to click Upload all which will upload your datasets. At any time you can pause the uploading and resume it later. There is no limit at the size of the files.

4.1.12 Compress Files

HopFS supports erasure-coding of files, which reduces storage requirements for large files by roughly 50%. If a file consists of 6 file blocks or more (that is, if the file is larger than 384 MB in size, for a default block size of 64 MB), then it can be compressed. Smaller files cannot be compressed.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compress file</td>
<td>You have to have the Data Owner role to be able to compress files. Select a file from your project. Right-click and select Compress to reduce the size of the file by changing its replication policy from triple replication to Reed-Solomon erasure coding.</td>
</tr>
</tbody>
</table>

4.1.13 Share a Data Set

Only a data owner or the project owner has privileges to share Data Sets. To share a Data Set, go to the Data Sets Browser in your project, and right-click on the Data Set to be shared and then select the Share Dataset option. A popup dialog will then prompt you to select (1) a target project with which the Data Set is to be Shared and whether the Data Set will be shared as read-only (Can View) or as read-write (Can edit). To complete the sharing process, a Data Owner in the target project has to click on the shared Data Set, and then click on Accept to complete the process.

4.1.14 Free-text Search

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search from Landing Page</td>
<td>On landing page, enter the search term in the search bar and press return. Returns project names and Data Set names that match the entered term.</td>
</tr>
<tr>
<td>Search from Project Page</td>
<td>From within the context of a project, enter the search term in the search bar and press return. The search returns any files or directories whose name or extended metadata matches the search term.</td>
</tr>
</tbody>
</table>
4.1.15 Jobs

The Jobs tabs is the way to create and run YARN applications. Hopsworks supports the following YARN applications:

- Apache Spark
- Apache Flink
- Adam (a bioinformatics data parallel framework)

If you are a beginner it is **highly** advisable to click on the Tours button at landing page. It will guide you through launching your first Spark application!

To run a job upload the required jar files and libraries to your dataset using the Dataset Browser. Click on the Jobs tab from the Project Menu and follow the steps below:

- Step 1: Press the New Job button on the top left corner
- Step 2: Give a name for your job
- Step 3: Select one of the available job types
- Step 4: Select the jar file with your job that you have uploaded earlier
- Step 5: Give the main class and any possible arguments
- (Optional) Step 6: In the Pre-configuration you can choose existing configurations according to existing jobs history and our heuristics
- Step 6: In the Configure and create tab you can manually specify the configuration you desire for your job and dependencies for the jar
- Step 7: Click on the Create job button
- Step 8: Click on the Run button to launch your job

On the right-hand side you can view some information about your job such as the Spark/Flink dashboard, YARN application UI, logs with Kibana and metrics with Grafana.

Job logs are available in Kibana during runtime and are also aggregated once the job is finished. They are available at the bottom of the Jobs page by clicking on them.

4.1.16 Workflows

Hopsworks provides a way to programmatically build Spark Job Workflows by using HopsUtil\(^\text{17}\). To build a workflow, users need to do the following:

- Create the jobs that need to be part of the project’s workflow.
- Create and run the workflow job, that is the one that implements and initiates the workflow.

\(^\text{17}\) https://github.com/hopshadoop/hops-util
API

HopsUtil\(^{18}\) provides a simple API (please consult HopsUtil javadoc\(^{19}\) for detailed information) to manage Hopsworks jobs. The API consists of the following two methods that accept a number of different parameters for more fine grained workflow management.

- **startJobs**: Gets a number of job IDs as input parameter and starts the respective jobs of the project for which the user invoking the jobs is also their creator. It can be used like `HopsUtil.startJobs(1);`

- **waitJobs**: Waits for jobs (supplied as comma-separated job IDs) to transition to a *running* (default) state or *not running*, depending whether an optional boolean parameter is true or not. It can be used like `waitJobs(1,5,11);`, which means the method will return when all three jobs with IDs 1,5,11 have finished running, or `waitJobs(false, 1,5,11);` which means the method will return when all jobs have entered the running state.

Example

Suppose there are two jobs in a project that prepare and analyze data, `PrepareData` and `DataAnalyzer` respectively. A simple workflow would be to run the `PrepareData` first in order to clean up and transform some data to a format required by the analyzer. Once the `PrepareData` job is finished, we want the `DataAnalyzer` to start automatically. The `WorkflowManager` code to build the workflow would look like:

```java
//Start job with ID: 6145 that prepares data for job with id 6146
HopsUtil.startJobs(6145);

//Wait for job with ID: 6145 to complete
HopsUtil.waitJobs(6145);

//Start job with ID: 6146
HopsUtil.startJobs(6146);
```

Source code for `WorkflowManager` is available [here]\(^{20}\).

The Job Service after running the workflow would look like the following image:

*Note that the API accepts job IDs instead of names, since different jobs can have the same name. To find the ID of a job, users need to click on the Job Details page shown in the following image:*

4.1.17 Logs visualization

Hopsworks aggregates the logs of the applications launched and uses Kibana for visualization and discovery. To get access to your logs, upon a job completion, click on the Job UI button and then on Logs tab. A sample output for the SparkPi job looks like the following.

---

\(^{18}\) https://github.com/hopshadoop/hops-util

\(^{19}\) http://snurran.sics.se/hops/hops-util-javadoc/0.1.0/

At the top section of the left-hand side of the main screen are the selected fields of the log that are shown on the main page. At the bottom section are the rest of the available indexed fields. On the top is the search bar where you can search for specific fields. For example if you want to preview the warning messages type priority=WARN. On the right side of the search bar you can save your search query and load it later.

Also you can visualize certain fields of your logs by clicking on the *Visualize* button. For example, assume we want to make a pie chart of the severity of the log messages.

- Step 1: Click on the *Visualize* button
- Step 2: Select the *Pie chart*
- Step 3: Click *From a new search* and select your project’s name from the drop-down menu
- Step 4: On the *buckets* section click *Split Slices*
- Step 5: The severity level is text so in the *Aggregation* drop-down select *Terms* and the *Field* we would like to visualize is the *priority* field
- Step 6: Click the green *Play* button on the top

The pie chart should look like the following. On the top right-hand side is the legend. Hopefully most of your job’s messages will be INFO and a few WARN! On the top right corner is the visualization menu where you can save, load or share the current chart.

### 4.1.18 Job metrics

We provide live monitoring of your application with Grafana where you can view metrics such as JVM memory consumption, Garbage Collection activity, HDFS activity, completed tasks etc. After you have launched your job, click on the *Job UI* button on the right and select the *Metrics* tab. A sample dashboard is illustrated below.

You can zoom in at any panel by left clicking, hold and drag across the X axis. If you want to share the dashboard, click on the *Share dashboard* button on the top which gives the options of sharing a link, an interactive snapshot of your dashboard or exporting it in a JSON format.

### 4.1.19 Jupyter Notebooks

This tutorial will go over the basics of using Jupyter notebooks on Hopsworks.

**Jupyter Notebooks Basics**

**Open the Jupyter Service**

Jupyter is provided as a micro-service on Hopsworks and can be found in the main UI inside a project.

**Start a Jupyter notebook server**

When you start a Jupyter notebook server you have the possibility to specify Spark properties of the notebooks you will create on the server. Hopsworks provides *Machine Learning-as-a-Service* with Tensorflow, Spark, supporting distributed training, parallel experiments, hyperparameter tuning, and model serving (HopsML). If you are doing machine learning on hops you probably want to select the notebook servers
“Experiment”, “Parallel Experiment” or “Distributed training” as shown in the figure below. See HopsML for more information on the Machine Learning pipeline. For general purpose notebooks, select the type “Spark (Static)” or “Spark (Dynamic)”.

**Jupyter + Spark on Hopsworks**

As a user, you will just interact with the Jupyter notebooks, but below you can find a detailed explanation of the technology behind the scenes.

When using Jupyter on Hopsworks, a library called sparkmagic\(^1\) is used to interact with the Hops cluster. When you create a Jupyter notebook on Hopsworks, you first select a **kernel**. A kernel is simply a program that executes the code that you have in the Jupyter cells, you can think of it as a REPL-backend to your jupyter notebook that acts as a frontend.

Sparkmagic works with a remote REST server for Spark, called livy\(^2\), running inside the Hops cluster. Livy is an interface that Jupyter-on-Hopsworks uses to interact with the Hops cluster. When you run Jupyter cells using the pyspark kernel, the kernel will automatically send commands to livy in the background for executing the commands on the cluster. Thus, the work that happens in the background when you run a Jupyter cell is as follows:

- The code in the cell will first go to the kernel.
- Next, the kernel kernel sends the code as a HTTP REST request to livy.
- When receiving the REST request, livy executes the code on the Spark driver in the cluster.
- If the code is regular python/scala/R code, it will run inside a python/scala/R interpreter on the Spark driver.
- If the code includes a spark command, using the spark session, a spark job will be launched on the cluster from the Spark driver.
- When the python/scala/R or spark execution is finished, the results are sent back from livy to the pyspark kernel/sparkmagic.
- Finally, the pyspark kernel displays the result in the Jupyter notebook.

The three Jupyter kernels we support on Hopsworks are:

- Spark, a kernel for executing scala code and interacting with the cluster through spark-scala
- PySpark, a kernel for executing python code and interacting with the cluster through pyspark
- SparkR, a kernel for executing R code and interacting with the cluster through spark-R

All notebooks make use of Spark, since that is the standard way to allocate resources and run jobs in the cluster.

In the rest of this tutorial we will focus on the pyspark kernel.

**Pyspark notebooks**

Create a pyspark notebook

---

\(^1\) [https://github.com/jupyter-incubator/sparkmagic](https://github.com/jupyter-incubator/sparkmagic)

\(^2\) [https://github.com/apache/incubator-livy](https://github.com/apache/incubator-livy)
After you have started the Jupyter notebook server, you can create a pyspark notebook from the Jupyter dashboard:

When you execute the first cell in a pyspark notebook, the spark session is automatically created, referring to the Hops cluster.

The notebook will look just like any python notebook, with the difference that the python interpreter is actually running on a Spark driver in the cluster. You can execute regular python code:

Since you are executing on the spark driver, you can also launch jobs on spark executors in the cluster, the spark session is available as the variable spark in the notebook:

When you execute a cell in Jupyter that starts a Spark job, you can go back to the Hopsworks-Jupyter-UI and you will see that a link to the SparkUI for the job that has been created.

In addition to having access to a regular python interpreter as well as the spark cluster, you also have access to magic commands provided by sparkmagic. You can view a list of all commands by executing a cell with %%help:

### Plotting with PySpark Kernel

So far throughout this tutorial, the Jupyter notebook have behaved more or less identical to how it does if you start the notebook server locally on your machine using a python kernel, without access to a Hadoop cluster. However, there is one main difference from a user-standpoint when using pyspark notebooks instead of regular python notebooks, this is related to plotting.

Since the code in a pyspark notebook is being executed remotely, in the spark cluster, regular python plotting will not work. What you can do however, is to use sparkmagic to download your remote spark dataframe as a local pandas dataframe and plot it using matplotlib, seaborn, or sparkmagics built in visualization. To do this we use the magics: %%sql, %%spark, and %%local. The steps to do plotting using a pyspark notebook are illustrated below. Using this approach, you can have large scale cluster computation and plotting in the same notebook.

**Step 1 : Create a remote Spark Dataframe:**

**Step 2 : Download the Spark Dataframe to a local Pandas Dataframe using %%sql or %%spark:**

**Note:** you should not try to download large spark dataframes for plotting. When you plot a dataframe, the entire dataframe must fit into memory, so add the flag 

-\texttt{maxrows x}

 to limit the dataframe size when you download it to the local Jupyter server for plotting.

Using %%sql:

Using %%spark:

**Step 3 : Plot the pandas dataframe using Python plotting libraries:**

When you download a dataframe from spark to pandas with sparkmagic, it gives you a default visualization of the data using autovizwidget, as you saw in the screenshots above. However, sometimes you want custom plots, using matplotlib or seaborn. To do this, use the sparkmagic %%local to access the local pandas dataframe and then you can plot like usual. Just make sure that you have your plotting libraries (e.g matplotlib or seaborn) installed on the Jupyter machine, contact a system administrator if this is not already installed.
Want to Learn More?

We have provided a large number of example notebooks, available here. Go to Hopsworks and try them out! You can do this either by taking one of the built-in tours on Hopsworks, or by uploading one of the example notebooks to your project and run it through the Jupyter service. You can also have a look at HopsML, which enables large-scale distributed deep learning on Hops.

4.1.20 Apache Kafka

Hopsworks provides Kafka-as-a-Service for streaming applications. Hopsworks provides by default the HopsUtil and hops-util-py libraries which make programming easier by abstracting away all the configuration boilerplate code such as Kafka endpoints, topics etc. Using these libraries, you can be up and running a simple Kafka on Hopsworks in minutes.

The following sections demonstrate different ways for writing Kafka applications on Hopsworks:

- Using the **Kafka service** on Hopsworks to setup Kafka topics in the cluster.
- Using the **Jobs service** on Hopsworks to submit jobs that produce/consume to/from Kafka.
- Using **Jupyter notebooks** on Hopsworks for producing/consuming to/from Kafka.

Our service is tightly coupled with our project-based model so only members of a project can use a specific Kafka topic, unless specified otherwise. The Kafka service on Hops is multi-tenant, allowing users to share topics between projects as desired.

Kafka Tour

If users prefer to be guided through the rest of the guide in Hopsworks, they can follow the **Kafka Tour** by selecting it from the available tours in the landing page.

Example Spark Streaming Jobs with Kafka on Hopsworks

Download and compile the example application

You can download and compile a sample Spark streaming by following these steps:

- Step 1: run `git clone https://github.com/logicalclocks/hops-examples` to clone the example project
- Step 2: run `cd hops-examples/spark/ && mvn package` to build the example project

Create a Kafka topic and schema

The next step is to create a Kafka topic that the sample spark streaming application will produce to and consume from. To create a topic, we use the Kafka service available in Hopsworks.

- Step 1: From the project box on the landing page, select a project
- Step 2: Click on the **Kafka** tab and the topics page will appear
• Step 3: First we need to create a schema for our topic, so click on the Schemas tab and New Avro Schema. Copy the sample schema from below and paste it into the Content box. Then click on the Validate button to validate the schema you provided and then Create.

```
{
  "fields": [
    {
      "name": "timestamp",
      "type": "string"
    },
    {
      "name": "priority",
      "type": "string"
    },
    {
      "name": "logger",
      "type": "string"
    },
    {
      "name": "message",
      "type": "string"
    }
  ],
  "name": "myrecord",
  "type": "record"
}
```

• Step 4: Click on New Topic, give a topic name, select the schema you created at Step 3 and press Create.

Advanced Kafka Topic Creation

A Kafka topic by default will be accessible only to members of a specific project. In order to share the topic with another project click on the Kafka service from the menu on the left. This will bring you to Kafka main page as illustrated below. Then press the the Share topic button on the appropriate topic and select the name of the project you would like to share with.

You can also fine grain access to Kafka topics by adding ACLs easily through Hopsworks. Once you have created a Kafka topic, click on the Kafka service and then on the Add new ACL button.

When creating a new ACL you are given the following options:

• **Permission Type** - Whether you will allow or deny access according to the ACL you are about to create

• **Operation Type** - The operation this ACL will affect:
  - `read` : Read from the topic
  - `write` : Write to the topic
  - `detail` : Get information about this topic

---

23 https://github.com/logicalclocks/hops-examples
24 https://github.com/hopshadoop/hops-util
25 https://github.com/logicalclocks/hops-util-py
When you are done with the ACL parameters click on the Create button. 

As an example assume that we have already created a Kafka topic for our project and we have shared this topic with another project named another_sample_project. We would like members of the other project NOT to be able to produce on this topic. Then the ACL would look like the following.

If you would like to see more details about your Kafka topic click on the Advanced view button. In the picture below we can see that there are three ACLs. The first is the default ACL which is applied when a topic is created. The second was created when we shared the topic with another project, allowing full access and finally the third is the custom ACL we created before.

Upload the compiled sample application and use it to create Spark jobs on Hopsworks

- Step 1: Upload the jar file from hops-examples/spark/target/ to a dataset. The jar is named: hops-examples-spark-X.Y.Z-SNAPSHOT.jar.

- Step 2: Click on the Jobs tabs at project menu and follow the instructions from the Jobs section. Create a new job for the Producer. Select Spark as job type and specify the jar file that you just uploaded. The name of the main class is io.hops.examples.spark.kafka.StructuredStreamingKafka and argument is producer. At the Configure and create tab, click on Kafka Services and select the Kafka topic you created at Step 4. Your job page should look like the following:

- Step 3: We repeat the instructions on Step 6 for the Consumer job. Type a different job name and as argument to the main class pass consumer /Projects/YOUR_PROJECT_NAME/Resources/Data. The rest remain the same as the Producer job.

Run the created producer/consumer jobs Run both jobs. While the consumer is running you can check its execution log. Use the Dataset browser to navigate to the directory /Resources/Data-APPLICATION_ID/. Right click on the file part-00000 and Preview the content.

A sample output would look like the following:

Example Python Notebook with Kafka Producer and Consumer

You can find several example notebooks using kafka at hopsexamples.

In this section we will demonstrate how you can use a jupyter notebook and python to produce/consume kafka messages. In this section it is assumed that you have already created a Kafka topic named “test” to produce/consume from and that you have enabled anaconda (which comes with some pre-installed packages, including the python package kafka-confluent) in your project.

Start Jupyter

https://github.com/logicalclocks/hops-examples

4.1. Hopsworks User Guide
Start Jupyter by going to the Jupyter tab, selecting Spark (static or dynamic), filling in the system properties and pressing “Start”.

**Create the new notebook**

Create a new notebook and paste the following

```python
from hops import kafka
from hops import tls
from confluent_kafka import Producer, Consumer

TOPIC_NAME = "test"
config = kafka.get_kafka_default_config()
producer = Producer(config)
consumer = Consumer(config)
consumer.subscribe(["test"])

# wait a little while before executing the rest of the code (put it in a different Jupyter cell)
# so that the consumer get chance to subscribe (asynchronous call)
for i in range(0, 10):
    producer.produce(TOPIC_NAME, "message {}".format(i), "key", callback=delivery_callback)

# Trigger the sending of all messages to the brokers, 10sec timeout
producer.flush(10)

for i in range(0, 10):
    msg = consumer.poll(timeout=5.0)
    if msg is not None:
        print('Consumed Message: {} from topic: {}'.format(msg.value(), msg.topic()))
    else:
        print("Topic empty, timeout when trying to consume message")
```

### 4.1.21 Apache Hive

**Introduction**

Hopsworks uses a fork\(^27\) of Apache Hive that enables users to keep the metadata storage consistent with the filesystem when a they delete their data, as the metadata describing databases, tables and partitions are deleted as well.

This page serves as a guide on how to use Hive from within Hopsworks. For information on how to write HiveQL (the language used to query the data) and configuration parameters available for tweaking, please refer to the Apache Hive wiki\(^28\).

**Using Hive with Hopsworks**

In order to use Hive from Hopsworks, users first need to create a database. Each project that is created with the Hive service enabled gets a Hive database. Users can enable the Hive service either when creating a new project or via the Settings section.

---

\(^27\) https://github.com/logicalclocks/hive  
\(^28\) https://cwiki.apache.org/confluence/display/Hive/Home
Once the database is created, users are able to see in the Datasets view a new dataset called `projectName.db`. This new dataset is the Hive database for the project and contains Hive’s data.

**There are two ways to run queries against Hive.** From within Hopsworks using Jupyter notebooks or by using an external client to submit queries with a Java client via JDBC:

- Using SparkSQL inside Jobs or Jupyter Notebooks
- Using an external Java/JDBC client

**Hive Workflow example**

The following is an example of a standard workflow when using Hive. Depending on where users plan to run the queries from, they can refer to the Running Queries using SparkSQL or Running Queries using Java/JDBC sections below.

The steps are: get the raw data into Hopsworks, load the data into Hive, convert the data in a more storage and computationally efficient format, such as ORC, and finally query the new table.

1. **Load the raw data into Hopsworks**: The easiest way to do it is to create a new dataset within the project and upload the data. Please remember not to generate the README.md file (or delete it after creating the dataset). This is because when creating the external table on the dataset, Hive will use all the files contained in the directory, README.me included if present. An alternative approach would be to create a directory inside the dataset and point Hive to that directory.

2. **Make Hive aware of the raw data**: To load the data into Hive, users can run the following query.

   An example of query can be the following:

   ```sql
   create external table sales(
     street string,
     city string,
     zip int,
     state string,
     beds int,
     baths int,
     sq__ft float,
     sales_type string,
     sale_date string,
     price float,
     latitude float,
     longitude float)
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ','
   LOCATION '/Projects/hivedoc/RawData'
   ```

   The above query assumes that the data is in CSV format and stored in a `RawData` dataset within the `hivedoc` project. More about the different formats supported by Hive and HiveQL can be found in the [Hive wiki](https://cwiki.apache.org/confluence/display/Hive/Home).

---

4.1. Hopworks User Guide
3. **Convert data in a more storage and computationaly efficient format**: CSV is not the best option when it comes to execute analytic queries. A better format would be ORC which compresses the data and stores it in a columnar oriented format. More about ORC [here](https://orc.apache.org/)³⁰. To convert the data users have to first create a new table:

```sql
create table orc_table (
    street string,
    city string,
    zip int,
    state string,
    beds int,
    baths int,
    sq__ft float,
    sales_type string,
    sale_date string,
    price float,
    latitude float,
    longitude float)
STORED AS ORC
```

The above table is a managed table without any specified location, this means that the table data will be managed by Hive and users will be able to access it in the `projectName.db` dataset. More complex data organization can be achieved by partitioning the table by one or multiple columns, or by using the bucketing feature. All the information about these options is available in the Hive [wiki](https://cwiki.apache.org/confluence/display/Hive/Home)³¹.

The next step is to convert the data from CSV to ORC, to do that users can run the following query:

```sql
insert overwrite table orc_table select * from sales
```

4. **Query the data**: finally the data is efficiently loaded into Hive and ready to be queried.

**Session based configuration**

Hive default configuration cannot be modified by users. What they can do though is change the values of certain configuration parameters for their sessions. Example: By default Hive is configured to not allow dynamic partitions, this means that the query shown previously at point 3 that inserts the data in the new table will fail. To enable dynamic partitioning we need to set `hive.exec.dynamic.partition.mode` to be `nostrict`. To do that users can execute the following statement, either in the PySpark notebook or in a Java/JDBC client

```sql
set hive.exec.dynamic.partition.mode=nostrict
```

This would enable dynamic partitioning for that session, other users will not be affected by this setting.

All the parameters that can be set or modified are listed in the Hive wiki under [Tez](https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties#ConfigurationProperties-Tez)³².

---

³⁰ [https://orc.apache.org/](https://orc.apache.org/)
³¹ [https://cwiki.apache.org/confluence/display/Hive/Home](https://cwiki.apache.org/confluence/display/Hive/Home)
³² [https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties#ConfigurationProperties-Tez](https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties#ConfigurationProperties-Tez)
Try it out

To try out Hive on Hopsworks, users can download a sample notebook\(^{33}\) or the Java client\(^{34}\) and a csv file\(^{35}\) containing sample data, create a dataset (without README) and upload the data.

Running Queries using SparkSQL

Users can execute queries against their Hive database using SparkSQL. The spark configuration for Hive is set up automatically when you create a Jupyter notebook. To view all tables in your project’s Hive database with PySpark inside a Jupyter notebook, run:

```python
from hops import hdfs as hopsfs
PROJECT_NAME = hopsfs.project_name()
spark.sql("use " + PROJECT_NAME)
spark.sql("show tables").show()
```

When you start Jupyter notebooks on Hopsworks, the spark session is automatically created for you with Hive enabled. If you are running Spark jobs on Hopsworks, you need to enable Hive in your spark session as follows:

```scala
val spark = SparkSession.builder().config(sparkConf).enableHiveSupport().
  .getOrCreate()
```

Here is an example notebook\(^{36}\) that shows more advanced queries.

Running Queries using Java/JDBC

Users can execute queries remotely against their Hopsworks Hive database by using an Java/JDBC client. An example with instructions on how to set up and run the remote\(^{37}\) client is available on our hops-examples\(^{38}\) github repo.

LLAP Admin

LLAP stands for *Live long and process*. It’s a cluster of long living daemons ready to be used by Hive to read data from the filesystem and to process query fragments. Hopsworks Admin users have the possibility of managing the lifecycle of the LLAP cluster. They can start and stop the LLAP cluster from the admin UI. In the admin UI they have the possibility of specifying the number of instances, the amount of memory each instance should get for the LLAP executors running inside the instance, the amount of memory for the cache and how many threads to use for the executors and for the IO.

---

\(^{33}\) https://github.com/logicalclocks/hops-examples/blob/master/notebooks/spark/PySparkWithHive.ipynb

\(^{34}\) https://github.com/logicalclocks/hops-examples/blob/master/hive/src/main/java/io/hops/examples/hive/HiveJDBCClient.java

\(^{35}\) http://snurran.sics.se/hops/hive/Sacramentorealestatetransactions.csv

\(^{36}\) https://github.com/logicalclocks/hops-examples/blob/master/notebooks/spark/PySparkWithHive.ipynb


\(^{38}\) https://github.com/logicalclocks/hops-examples#hive

---
Normal users can by default use the LLAP cluster in all the projects. By default Hive decides which fragments of the query execute on the LLAP cluster and which in a separate container. Users can change this behavior by changing the session based configuration as explained above.

### 4.1.22 Metadata Management

Metadata enables **data curation**, that is, ensuring that data is properly catalogued and accessible to appropriate users.

Metadata in Hopsworks is used primarily to discover and retrieve relevant data sets or files by users by enabling them to attach arbitrary metadata to Data Sets, directories or files in Hopsworks. Metadata is associated with an individual file or Data Set or directory. This extended metadata is stored in the same database as the metadata for HopsFS and foreign keys link the extended metadata with the target file/directory/Data Set, ensuring its integrity. Extended metadata is exported to Elastic Search, from where it can be queried and the associated Data Set/Project/file/directory can be identified (and acted upon).

### 4.1.23 MetaData Designer

Within the context of a project, click on the **Metadata Designer** button in the left-hand panel. It will bring up a metadata designer view that can be used to:

- Design a new Metadata Template
- Extend an existing Metadata Template
- Import/Export a Metadata Template

Within the Metadata Designer, you can define a Metadata template as one or more tables. Each table consists of a number of typed columns. Supported column types are:

- **string**
- single-select selection box
- multi-select selection box

Columns can also have constraints defined on them. On a column, click on cog icon (configure), where you can make the field:

- searchable: included in the Elastic Search index;
- required: when entering metadata, this column will make it is mandatory for users to enter a value for this column.

### 4.1.24 MetaData Attachment and Entry

Within the context of a project, click on the **Data Sets** tab. From here, click on a Data Set. Inside the Data Set, if you select any file or directory, the rightmost panel will display any extended metadata associated with the file or directory. If no extended metadata is associated with the file/directory, you will see “No metadata template attached” in the rightmost panel. You can attach an existing metadata template to the file.
or directory by right-clicking on it, and selecting Add metadata template. The metadata can then be selected from the set of available templates (designed or uploaded).

After one or more metadata templates have been attached to the file/directory, if the file is selected, the metadata templates are now visible in the rightmost panel. The metadata can be edited in place by clicking on the + icon beside the metadata attribute. More than one extended metadata value can be added for each attribute, if the attribute is a string attribute.

Metadata values can also be removed, and metadata templates can be removed from files/directories using the Data Set service.

4.1.25 Feature Store

In this tutorial we cover the feature store service in Hopsworks, how it should be used, how it fits into the machine learning pipeline, and the tech-stack behind it.

Feature Store: The Data Management Layer for Machine Learning in Hopsworks

The feature store is as a central place to store curated features for machine learning pipelines in Hopsworks. A feature is a measurable property of some data-sample. It could be for example an image-pixel, a word from a piece of text, the age of a person, a coordinate emitted from a sensor, or an aggregate value like the average number of purchases within the last hour. Features can come directly from tables or files or can be derived values, computed from one or more data sources.

Features are the fuel for AI systems, as we use them to train machine learning models so that we can make predictions for feature values that we have never seen before. In this tutorial we will see best practices for transforming raw/structured data into features that can be included in training datasets for training models.

From talking with many organizations that work with data and machine learning, the feature store also has a natural fit in the organizational structure, where it works as an interface between data engineers and data scientists.

Data Engineers write features into the feature store. Typically they will (1) read some raw or structured data from a data lake: (2) apply transformations on the data using som data processing framework like Spark; (3) store the transformed feature data in the feature store; (4) add documentation, versioning information, and statistics to the feature data in the feature store.

Data Scientists read features from the feature store. A data scientist tend to read features from the feature store for (1) training machine learning models and experimenting with different combination of features; and (2) serving features into machine learning models.

Motivation for the Feature Store

Machine learning systems have a tendency to assemble technical debt. Examples of technical debt in machine learning systems are:

- There is no principled way to access features during model serving.
- Features cannot easily be re-used between multiple machine learning pipelines.
• Data science projects work in isolation without collaboration and re-use.

• Features used for training and serving are inconsistent.

• When new data arrives, there is no way to pin down exactly which features need to be recomputed, rather the entire pipeline needs to be run to update features.

Using a feature store is a best practice that can reduce the technical debt of machine learning work-flows. When the feature store is built up with more features, it becomes easier and cheaper to build new models as the new models can re-use features that exist in the feature store. The Feature Store provides, among other things:

• Feature reuse

• Feature discoverability

• Feature backfilling and precomputation

• Improved documentation and analysis of features

• Software engineering principles applied to machine learning features: versioning, documentation, access-control

• Scale; The feature store needs to be able to store and manage huge feature sets (multi-terabyte at least).

• Flexibility; Data scientists must be able to read from the feature store and use the data in different machine learning frameworks, like Tensorflow, Keras, Scikit learn, and PyTorch.

• Analysis; Data scientists need an understanding of the feature data to be able to make most use of it in their models. They should be able to analyze the features, view their distributions over time, their correlations with each other etc.

• Point-in-time correctness; It can be valuable to be able to extract the value of a feature at a specific point-in-time to be able to later on change the value of the feature.

• Real-time; for client-facing models, features must be available in real-time (< 10ms) for making predictions to avoid destroying the user-experience for the user.

• Online/Offline Consistency; when a feature is used for both training and serving, and stored in two different storage layers, you want to make sure that the value and semantics of the feature is consistent.

### How to Use the Feature Store

When adopting the feature store in your machine learning work-flows, you can think of it as the interface between data engineering and data science. It has two APIs, one for writing to the feature store and one for reading features. At the end of your data engineering pipeline, instead of writing features to a custom storage location, write the features to the feature store and get benefits such as automatic documentation, versioning, feature analysis, and feature sharing. As a data scientist, when you start a new machine learning project, you can search through the feature store for available features and only add new features if they do not already exist in the feature store. We encourage to reuse features as much as possible to avoid doing unnecessary work and to make features consistent between different models.
Creating a Project on Hopworks with The Feature Store Service Enabled

To create a project with the feature store service, mark the check-box available when you create a new project in Hopworks.

Inside the project you can find the feature registry (where all the feature store data is browsable) in the feature store page that is accessible by clicking the feature store icon on the left.

Data Modeling in the Feature Store

We introduce three new concepts to our users for modeling data in the feature store.

- The **feature** is an individual versioned and documented data column in the feature store, e.g. the average rating of a customer.

- The **feature group** is a documented and versioned group of features stored as a Hive table. The feature group is linked to a specific Spark/Numpy/Pandas job that takes in raw data and outputs the computed features.

- The **training dataset** is a versioned and managed dataset of features and labels (potentially from multiple different feature groups). Training datasets are stored in HopsFS as tfrecords, parquet, csv, or tsv files.

The API

The feature store in Hopworks has a REST API that is accessible with any REST-client, or with the provided Python Scala/Java SDKs. This section gives an overview of the API and how to work with either the Python SDK or the Scala/Java SDK. We will show examples of the most common API methods. To get a full overview of the API please see the API-Docs-Python[^39], API-Docs-Scala[^40] and the feature-store_example_notebooks[^41].

Creating New Features

The feature store is agnostic to the method for computing the features. The only requirement is that the features can be grouped together in a Pandas, Numpy, or Spark dataframe. The user provides a dataframe with features and associated feature metadata (metadata can also be edited later through the feature registry UI) and the feature store library takes care of creating a new version of the feature group, computing feature statistics and linking the features to the job to compute them.

- Inserting into an existing feature group using the Python API:

```
from hops import featurestore
featurestore.insert_into_featuregroup(features_df, featuregroup_name)
```

[^39]: http://hops-py.logicalclocks.com/
[^40]: http://snurran.sics.se/hops/hops-util-javadoc/
[^41]: https://github.com/Limmen/hops-examples/tree/HOPSWORKS-721/notebooks/featurestore
• Inserting into an existing feature group using the Scala/Java API:

```java
import io.hops.util.Hops
Hops.insertIntoFeaturegroup(featuregroupName).setDataFrame(sampleDF).setMode("append").write()
```

• Creating a new feature group using the Python API:

```python
from hops import featurestore
featurestore.create_featuregroup(features_df, featuregroup_name)
```

• Creating a new feature group using the Scala/Java API:

```java
import io.hops.util.Hops
Hops.createFeaturegroup(featuregroupName).setDataFrame(featuresDf).write()
```

Reading From the Feature Store

To read features from the feature store, users can use either SQL directly or the API-functions available in Python and Scala. Based on our experience with users on our platform, data scientists can have diverse backgrounds. Although some data scientists are very comfortable with SQL, others prefer higher level APIs. This motivated us to develop a query-planner to simplify user queries. The query-planner enables users to express the bare minimum information to fetch features from the feature store. For example, a user can request 100 features that are spread across 20 different feature groups by just providing a list of feature names. The query planner uses the metadata in the feature store to infer where to fetch the features from and how to join them together.

For example, to fetch the features `average_attendance` and `average_player_age` from the feature store, all the user has to write is:

```python
from hops import featurestore
features_df = featurestore.get_features(["average_attendance", "average_player_age"])
```

and using the Scala/Java API:

```java
import io.hops.util.Hops
val features_df = Hops.getFeatures(List("average_attendance", "average_player_age")).read()
```

Creating Training Datasets

Organizations typically have many different types of raw datasets that can be used to extract features. For example, in the context of user recommendation there might be one dataset with demographic data of users and another dataset with user activities. Features from the same dataset are naturally grouped into a feature group, producing one feature group per dataset. When training a model, you want to include all features that have predictive power for the prediction task, these features can potentially span multiple feature groups. The training dataset abstraction in Hopsworks Feature Store is used for this purpose, allowing users to group a set of features with labels for training a model to do a particular prediction task.
Once a user has fetched a set of features from different feature groups in the feature store, the features can be materialized into a training dataset. By creating a training dataset using the feature store API, the dataset becomes managed by the feature store. Managed training datasets are automatically analyzed for data anomalies, versioned, documented, and shared with the rest of the organization.

To create a managed training dataset, the user supplies a Pandas, Numpy or Spark dataframe with features and labels together with metadata.

- Creating a training dataset using the Python API:

```python
from hops import featurestore
featurestore.create_training_dataset(features_df, training_dataset_name, data_format="tfrecords")
```

- Creating a training dataset using the Scala/Java API:

```scala
import io.hops.util.Hops
Hops.createTrainingDataset(training_dataset_name).setDataframe(featuresDf).setDataFormat("tfrecords").write()
```

### Reading a Training Dataset for Training a Model:

Once the training dataset has been created, the dataset is discoverable in the feature registry and users can use it to train models. Below is an example code snippet for training a model using a training dataset stored distributed in the tfrecords format on HopsFS.

- Using the Python API:

```python
from hops import featurestore
import tensorflow as tf
dataset_dir = featurestore.get_training_dataset_path(td_name)
# the tf records are written in a distributed manner using partitions
input_files = tf.gfile.Glob(dataset_dir + "part-r-*")
# tf record schemas are managed by the feature store
tf_record_schema = featurestore.get_training_dataset_tf_record_schema(td_name)
def decode(example_proto):
    return tf.parse_single_example(example_proto, tf_record_schema)
dataset = tf.data.TFRecordDataset(input_files)
    .map(decode)
    .shuffle(shuffle_buffer_size)
    .batch(batch_size)
    .repeat(num_epochs)
# three layer MLP for regression
model = tf.keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(1)
])
model.compile(optimizer=tf.train.AdamOptimizer(lr), loss="mse")
model.fit(dataset, epochs=num_epochs, steps_per_epoch=spe)
```
• Using the Scala/Java API:

```scala
import io.hops.util.Hops
val dataset_df = Hops.getTrainingDataset("team_position_prediction").read()
    setOutputCol("features").
    transform(dataset_df).
    drop("average_player_rating").
    drop("average_attendance").
    drop("sum_player_rating").
    drop("sum_player_worth").
    drop("average_player_age").
    drop("average_player_worth").
    drop("team_budget").
    drop("average_position").
    drop("sum_player_age").
    drop("sum_attendance").
    drop("sum_position")
val lr = new LinearRegression().
    setLabelCol("team_position").
    setFeaturesCol("features").
    setMaxIter(NUM_ITER).
    setRegParam(REG_LAMBDA_PARAM).
    setElasticNetParam(ELASTIC_REG_PARAM)
val lrModel = lr.fit(transformedDf)
lrModel.transform(transformedDf).select("features", "team_position", "prediction").show()
val trainingSummary = lrModel.summary
println(s"numIterations: 
    objectiveHistory: [" + trainingSummary.objectiveHistory.mkString("","") + "]")
trainingSummary.residuals.show()
println(s"RMSE: 
    r2: ")
```

The Feature Registry

The feature registry is the user interface for publishing and discovering features and training datasets. The feature registry also serves as a tool for analyzing feature evolution over time by comparing feature versions. When a new data science project is started, data scientists within the project typically begin by scanning the feature registry for available features, and only add new features for their model that do not already exist in the feature store.

The feature registry provides:

• Keyword search on feature/feature group/training dataset metadata.
• Create/Update/Delete/View operations on feature/feature group/training dataset metadata.
• Automatic feature analysis.
• Feature dependency/provenance tracking.
• Feature job tracking.

Finding Features
In the registry you can search for features, feature groups and training datasets in the search bar. Features are automatically grouped by versions in the search results.

Automatic Feature Analysis
When a feature group or training dataset is updated in the feature store, a data analysis step is performed. In particular, we look at cluster analysis, feature correlation, feature histograms and descriptive statistics. We have found that these are the most common type of statistics that our users find useful in the feature modeling phase. For example, feature correlation information can be used to identify redundant features, feature histograms can be used to monitor feature distributions between different versions of a feature to discover covariate shift, and cluster analysis can be used to spot outliers. Having such statistics accessible in the feature registry helps data scientists decide on which features to use.

Other Actions Available in the Feature Registry
A common practice using the feature store is that the data of feature groups and training datasets are inserted using the APIs in Python/Java/Scala, but the metadata is filled and edited from the feature registry UI. In addition to editing metadata about features, the registry also provides the following functionality:
• Create/Update/Delete operations on feature groups and training datasets
• Preview feature group data
• View feature group and training dataset schemas
• Create new Training Datasets by grouping features together
• Configuring storage connectors

On-Demand and Cached Features
There are two types of feature groups in the Feature Store:
• **Cached Feature Group**: This type of feature group is the most common, it will store the computed features inside the Hopsworks Feature Store.
• **On-Demand Feature Groups**: This type of feature group is not stored in Hopsworks, rather it is computed on-demand. To create an on-demand feature group you must configure a JDBC connector and a SQL query to compute the features. On-demand feature groups are typically used when an organization have feature data available in external storage systems and don’t want to duplicate the data in Hopsworks feature store.

The code-snippets below illustrates the different APIs for creating a cached vs an on-demand feature group using the Scala SDK:
//Cached Feature Group
Hops.createFeaturegroup(fgName).setDataFrame(df).write()

//On-Demand Feature Group
Hops.createFeaturegroup(fgName).setOnDemand(true).setJdbcConnector(sc).
→setSqlQuery(query).write()

External and HopsFS Training Datasets

There are two storage types for training datasets in the Feature Store:

- **HopsFS**: The default storage type for training datasets is HopsFS, a state-of-the-art scalable file system that comes bundled with the Hopsworks stack.

- **S3**: The feature store SDKs also provides the functionality to store training datasets external to a Hopsworks installation, e.g. in S3. When training datasets are stored in S3, only the metadata is managed in Hopsworks and the actual data is stored in S3. To be able to create external training datasets, you must first configure a storage connector to S3.

The code-snippets below illustrates the different APIs for creating a training dataset stored in HopsFS vs a training dataset stored in S3, using the Scala SDK:

```scala
//Training Dataset stored in HopsFS (default sink)
Hops.createTrainingDataset(tdName).setDataFrame(df).write()

//External Training Dataset
Hops.createTrainingDataset(tdName).setDataFrame(df).setSink(s3Connector).
→write()
```

Configuring Storage Connectors for the Feature Store

By default, a feature store created in Hopsworks will have three storage connectors:

- **projectname**, a JDBC connector for the project’s general-purpose Hive database

- **projectname_featurestore**, a JDBC connector for the project’s feature store database (this is where cached feature groups are stored)

- **projectname_Training_Datasets**, a HopsFS connector for storing training datasets inside the project

To configure new storage connectors, e.g. S3, HopsFS, or JDBC connectors, use the form available in the feature registry UI.

A Multi-tenant Feature Store Service

Despite the benefit of centralizing features, we have identified a need to enforce access control to features. Several organizations that we have talked to are working partially with sensitive data that requires specific access rights that is not granted to everyone in the organization. For example, it might not be feasible to publish features that are extracted from sensitive data to a feature store that is public within the organization.
To solve this problem we utilize the multi-tenant model of Hopsworks. Feature stores in Hopsworks are by default project-private and can be shared across projects, which means that an organization can combine public and private feature stores. An organization can have a central public feature store that is shared with everyone in the organization as well as private feature stores containing features of sensitive nature that are only accessible by users with the appropriate permissions.

To share a feature store with another project, share the dataset containing the feature groups and features (\texttt{projectname\_featurestore.db}) as well as the dataset that contains the training datasets (\texttt{projectname\_Training\_Datasets}). To share datasets in Hopsworks simply right-click the feature store inside of your project dataset browser:

When you have multiple feature stores shared with your project you can select which one to view in the feature registry:

**Technical Details on the Architecture**

The architecture of the feature store in hopsworks is depicted in the image below.

A feature store consists of five main components:

- The feature engineering jobs, the jobs used to compute the features and insert into the feature store.
- The storage layer for storing the feature data.
- The metadata layer used for storing code to compute features, versioning, analysis data, and documentation.
- The API, used for reading/writing features from/to the feature store.
- The feature registry, a user interface (UI) service where data scientists can share, discover, and order computation of features.

**Want to Learn More?**

We have provided a large number of example notebooks, available here\textsuperscript{42}. Go to Hopsworks and try them out! You can do this either by taking one of the built-in \textit{tours} on Hopsworks, or by uploading one of the example notebooks to your project and run it through the Jupyter service. You can also have a look at \texttt{HopsML}, which enables large-scale distributed deep learning on Hops.

**4.1.26 Apache Airflow**

Hopsworks version 0.9.0 introduces native support for Apache Airflow\textsuperscript{43}. Airflow is a platform to programatically schedule and monitor workflows. Using Hopsworks operators a user can launch and monitor jobs in Hopsworks (almost) transparently.

\textsuperscript{42} https://github.com/logicalclocks/hops-examples

\textsuperscript{43} https://airflow.apache.org/index.html
Airflow introduction

Airflow is built on top of three core concepts:

1. DAGs
2. Operators
3. Tasks

In Airflow a Directed Acyclic Graph (DAG) is a model of the tasks you wish to run defined in Python. The model is organized in such a way that clearly represents the dependencies among the tasks. For example, task B and C should both run only after task A has finished.

A DAG constructs a model of the workflow and the tasks that should run. So far a task is quite general, operators define what a task should actually execute. Operators are usually (it is recommended) atomic in a sense that they can stand on their own without sharing state with others. They can have order dependencies though, operator B should run after operator A.

Apache Airflow ships with a lot of operators for the most common (and uncommon) operations. Such operators are:

- BashOperator  
  Executes bash commands
- PythonOperator  
  Executes arbitrary Python code
- SimpleHttpOperator  
  Performs HTTP requests
- MySqlOperator, PostgresOperator, ...  
  Connectors to execute SQL commands
- More specific such as SlackOperator, HiveOperator and a lot more supplied by Airflow and/or the community

Hopworks comes with some very basic, for the moment, Airflow operators which will be discussed in detail later in this document.

Finally, tasks are instantiated operators in a specific point in time. Think of classes and objects in OOP. Since workflows can be scheduled to run repeatedly operators’ result may vary. So tasks are also defined by the time which ran. For example task A which ran on Friday 1 March 2019 16:05 succeeded but the same task which ran one day later failed.

Airflow in Hopworks

Hopworks provides seamless integration with Apache Airflow. For projects that have activated Airflow service, users can access it from the micro-services panel on the left and then click on the Open Airflow button as shown in image below. From the landing page you can upload a DAG and let Airflow scheduler pick it as we will see later.

Hopworks has native support of Airflow using our home brewed plugin which hides all the complexity from the user. Source code is available here. As the plugin matures, we will add more functionality but for the moment we have the following operators.

Hopworks operators are available by importing the module hopworks_plugin/operators.hopworks_operator
**HopsworksLaunchOperator**

With this operator you can launch jobs on Hopsworks. Jobs should have already been created from Jobs UI. For more information on how to create jobs visit our Jobs (page 46) documentation. As this operator extends the BaseOperator of Airflow it inherits all of its arguments. The extra arguments that should be passed to this operator are the following:

- **job_name**: The name of the job created from Jobs UI
- **project_id**: Numeric ID of the project job belongs to

The operator does NOT wait for job completion. It just sends the request to Hopsworks to launch the job. To wait for completion there are two special kind of operators called *sensors*.

**HopsworksJobFinishSensor**

As we mentioned earlier, HopsworksLaunchOperator does not block when it launches a job. Waiting for job completion should be another task in the DAG. This sensor waits for a specific job to reach a final state. A final state could be either **successful** or **failed**. The required arguments for this sensor are the same as in HopsworksLaunchOperator.

**HopsworksJobSuccessSensor**

This sensor is similar to the one above but it succeeds **only** if the job launched reaches a successful final state. If a job fails, then this sensor will fail too and the tasks that depend on the job success will not be executed.

In the rest of this document we will see how to write a DAG in Python, upload it to Hopsworks and monitor our tasks. It is **strongly** advised to go through this section even if you are familiar with Airflow.

**Airflow primer**

In this section we will demonstrate how to begin with Airflow in Hopsworks. First we will create our jobs, then write our DAG, upload it to Hopsworks and run it from Airflow.

**Create jobs**

The first thing we need to do is to create the jobs in Hopsworks from the Jobs UI. There we will specify the jar file to run, application arguments, dependencies, etc.

For the sake of example, we will create three jobs. All three will run a Spark job, SparkPi and the job names will be **job0**, **job1** and **job2**. From a demo_spark_project we can either copy the demo job or upload spark-examples.jar file to our datasets. After we finish creating the jobs, the Jobs UI will look like the following. It is important to note the names of the jobs.

---

44 https://github.com/logicalclocks/airflow-chef/tree/master/files/default/hopsworks_plugin
Write and upload DAG

Next step is to compose the workflow in Python. If it’s the first time you encounter Airflow DAGs you should look on Airflow documentation on writing DAGs. We open our favourite editor and start defining our tasks, some important parameters and the order of the tasks. In the end our `example_dag.py` will look like the following.

```python
import airflow
from datetime import datetime, timedelta
from airflow import DAG
from hopsworks_plugin.operators.hopsworks_operator import HopsworksLaunchOperator
from hopsworks_plugin.sensors.hopsworks_sensor import HopsworksJobSuccessSensor

delta = timedelta(minutes=-10)
now = datetime.now()
args = {
    # Username in Hopsworks
    'owner': 'meb10000',
    'depends_on_past': False,
    
    # Start date is 10 minutes ago
    'start_date': now + delta
}

dag = DAG(
    dag_id = 'windflow_dag',
    default_args = args,
    # Run every seven minutes
    schedule_interval = '*/7 * * * *'
)

# Project ID extracted from URL
PROJECT_ID = 6

task1 = HopsworksLaunchOperator(dag=dag, task_id='run_job_0', job_name='job0', project_id=PROJECT_ID)
task2 = HopsworksLaunchOperator(dag=dag, task_id='run_job_1', job_name='job1', project_id=PROJECT_ID)
task3 = HopsworksLaunchOperator(dag=dag, task_id='run_job_2', job_name='job2', project_id=PROJECT_ID)
sensor = HopsworksJobSuccessSensor(dag=dag,
    poke_interval=10,
    task_id='wait_for_success_job0',
    job_name='job0',
    project_id=PROJECT_ID)
```

(continues on next page)
In the beginning of the file, we define some arguments for the DAG. It is **very important** to set the owner to your username in Hopsworks. For security reasons, Hopsworks will fail to execute the tasks if your username is not correct. You can get your username by clicking on the `Account` button on the top right drop-down menu.

Next we define the DAG name which can be anything, it is an identifier used in Airflow. Also, we schedule the task to run every seven minutes starting 10 minutes ago. This will produce a DAG run already when we upload it since the scheduler is trying to catch up the missed runs.

Another **very important** parameter is the project ID. In the current version you must get the project ID from the URL. For example, when you navigate to Airflow service in Hopsworks, the URL in your browser would look like the following `https://localhost:8181/hopsworks/#!/project/6/airflow`. Here the project ID is **6**.

As we mentioned earlier, each `HopsworksLaunchOperator` has a `job_name` argument which is the name of the job in Hopsworks to launch. Also, `HopsworksJobSuccessSensor` takes the name of the job that it should wait for a successful final state.

Finally, we define the order of the tasks. `task1` will run first, then DAG will wait until it finishes successfully and after the `sensor` task finishes, it launches `task2` and `task3` in parallel.

Next step is to upload the DAG file written in Python to Hopsworks and let Airflow scheduler load it. From the Projects side navigation bar on the left, click on Airflow to navigate to Airflow landing page. This is a file manager that will list all the DAG files uploaded for this project. Users who are not members of the project cannot access them. On the right-hand side of the file manager click on the three vertical dots to upload your file as shown in the picture below. After you upload it, it will appear on the landing page and after some seconds it will also appear on Airflow web interface.

An important note is that by default DAGs are **paused**. They will not be executed unless you explicitly `unpause` it by clicking on the toggle left of the DAG name in Airflow UI.

### Run and monitor DAGs

So far we’ve seen how to use Hopsworks operators to launch jobs on a cluster, sensors to monitor the jobs and how to upload DAG files for Airflow to pick them. In this final section we will give a very brief Airflow introduction. For more information please visit Airflow webpage. To open Airflow web UI, from Airflow landing page on Hopsworks, click the `Open Airflow` green button on the top. This will open Airflow in a new tab.

We filter DAGs my owner, so you will only see DAGs uploaded with your username. Also, if you have administrative privileges in Hopsworks, it holds true also in Airflow. Usually, tasks run in an interval but you can always run them explicitly by clicking the play button on the right of the DAG name. From this page you can also get an overview of the tasks running, failed, queued for execution, etc. If tasks manage to run correctly, you should see in Jobs UI, jobs running and increased cluster utilization.
Clicking on a DAG name will show the monitoring page for the tasks of that DAG. In the picture below we see a visual representation of our example DAG and tasks status when we switch to the Graph View tab.

We can monitor a specific task by clicking on the task name. From the pop-up menu we can view the logs, cancel the task or the tasks that are dependent on it, etc. From the view above, regarding our example DAG, we can see that task run_job_0 ran first, then our sensor wait_for_success_job0 ran which waited for a successful exit of job_0. After the sensor has finished successfully the scheduler ran tasks run_job_1 and run_job_2 in parallel. Since HopsworksLaunchOperator does NOT block for the job to finish, a successful exit means that it managed to make a request to Hopsworks to launch the job, not that the associated job finished correctly.

In order to delete a DAG from Airflow, first pause the DAG so it won’t be scheduled for execution. Then go to the file manager in Hopsworks, switch to column view from the right corner, right click on the DAG file and delete. Finally, go to Airflow UI and delete the entry.

Conclusion

In this article we introduced (page 68) Apache Airflow as a micro-service in Hopsworks. We have seamlessly integrated it to our platform with our custom authentication module and plugin - which for the moment provides some basic operators. Future releases of Hopsworks will include more operators allowing users to build more complex workflows. Also, we demonstrated (page 69) how users can begin with Airflow by launching basic jobs on the cluster and monitor their status.

If you are using 2-Factor authentication, jump ahead to “First Login with 2-Factor Authentication”.

4.2 HopsFS User Guide

4.2.1 Unsupported HDFS Features

HopsFS is a drop-in replacement for HDFS and it supports most of the configuration parameters defined for Apache HDFS. As the architecture of HopsFS is fundamentally different from HDFS, some of the features such as journaling, secondary NameNode etc., are not required in HopsFS. Following is the list of HDFS features and configurations that are not applicable in HopsFS

- **Secondary NameNode** The secondary NameNode is no longer supported. HopsFS supports multiple active NameNodes. Thus hdfs haadmin * command; and dfs.namenode.secondary,* and dfs.ha,* configuration parameters are not supported in HopsFS.

- **Checkpoint Node and FSImage** HopsFS does not require checkpoint node as all the metadata is stored in NDB. Thus hdfs dfsadmin -{saveNamespace | metaSave | restoreFailed-Storage | rollEdits | fetchImage} command; and dfs.namenode.name.dir.*, dfs.image.*, dfs.namenode.checkpoint.* configuration parameters are not supported in HopsFS.

- **Quorum Based Journaling and EditLog** The write ahead log (EditLog) is not needed as all the metadata mutations are stored in NDB. Thus dfs.namenode.num.extra.edits.*, dfs.journald.*, and dfs.namenode.edits.* configuration parameters are not supported in HopsFS.

45 http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml
• **NameNode Federation and ViewFS** In HDFS the namespace is statically partitioned among multiple namenodes to support large namespace. In essence these are independent HDFS clusters where ViewFS provides a unified view of the namespace. HDFS Federation and ViewFS are no longer supported as the namespace in HopsFS scales to billions of files and directories. Thus `dfs.nameservices.*` configuration parameters are not supported in HopsFS.

• **ZooKeeper** ZooKeeper is no longer required as the coordination and membership service. A coordination and membership management *service* is implemented using the transactional shared memory (NDB).

As HopsFS is under heavy development some features such as rolling upgrades and snapshots are not yet supported. These features will be activated in future releases.

### 4.2.2 NameNodes

HopsFS supports multiple NameNodes. A NameNode is configured as if it is the only NameNode in the system. Using the database a NameNode discovers all the existing NameNodes in the system. One of the NameNodes is declared the leader for housekeeping and maintenance operations. All the NameNodes in HopsFS are active. Secondary NameNode and Checkpoint Node configurations are not supported. See section (page 72) for detail list of configuration parameters and features that are no longer supported in HopsFS.

For each NameNode define `fs.defaultFS` configuration parameter in the *core-site.xml* file. In order to load NDB driver set the `dfs.storage.driver.*` parameters in the `hdfs-site.xml` file. These parameter are defined in detail [here](#) (page 118).

A detailed description of all the new configuration parameters for leader election, NameNode caches, distributed transaction handling, quota management, id generation and client configurations are defined [here](#) (page 112).

The NameNodes are started/stopped using the following commands (executed as HDFS superuser):

```
> $HADOOP_HOME/sbin/start-nn.sh
> $HADOOP_HOME/sbin/stop-nn.sh
```

The Apache HDFS commands for starting/stopping NameNodes can also be used:

```
> $HADOOP_HOME/sbin/hadoop-daemon.sh --script hdfs start namenode
> $HADOOP_HOME/sbin/hadoop-daemon.sh --script hdfs stop namenode
```
Configuring HopsFS NameNode is very similar to configuring a HDFS NameNode. While configuring a single Hops NameNode, the configuration files are written as if it is the only NameNode in the system. The NameNode automatically detects other NameNodes using NDB.

### Formating the Filesystem

Running the format command on any NameNode truncates all the tables in the database and inserts default values in the tables. NDB atomically performs the truncate operation which can fail or take very long time to complete for very large tables. In such cases run the `/hdfs namenode -dropAndCreateDB` command first to drop and recreate the database schema followed by the `format` command to insert default values in the database tables. In NDB dropping and recreating a database is much quicker than truncating all the tables in the database.

### NameNode Caches

In published Hadoop workloads, metadata accesses follow a heavy-tail distribution where 3% of files account for 80% of accesses. This means that caching recently accessed metadata at NameNodes could give a significant performance boost. Each NameNode has a local cache that stores INode objects for recently accessed files and directories. Usually, the clients read/write files in the same sub-directory. Using `RANDOM_STICKY` load balancing policy to distribute filesystem operations among the NameNodes lowers the latencies for filesystem operations as most of the path components are already available in the NameNode cache. See *HopsFS Client’s* (page 75) and *Cache Configuration Parameters* (page 114) for more details.

### Adding/Removing NameNodes

As the namenodes are stateless any NameNode can be removed with out effecting the state of the system. All on going operations that fail due to stopping the NameNode are automatically forwarded by the clients to the remaining namenodes in the system.

Similarly, the clients automatically discover the newly started namenodes. See *client configuration parameters* (page 117) that determines how quickly a new NameNode starts receiving requests from the existing clients.

### 4.2.3 DataNodes

The DataNodes periodically acquire an updated list of NameNodes in the system and establish a connection (register) with the new NameNodes. Like clients, the DataNodes also uniformly distribute the filesystem operations among all the NameNodes in the system. Currently the DataNodes only support round-robin policy to distribute the filesystem operations.

HopsFS DataNodes configuration is identical to HDFS DataNodes. In HopsFS a DataNode connects to all the NameNodes. Make sure that the `fs.defaultFS` parameter points to valid NameNode in the system. The DataNode will connect to the NameNode and obtain a list of all the active NameNodes in the system, and then connects/registers with all the NameNodes in the system.

The DataNodes can started/stopped using the following commands (executed as HDFS superuser):
The Apache HDFS commands for starting/stopping Data Nodes can also be used:

```
> $HADOOP_HOME/sbin/start-dn.sh
> $HADOOP_HOME/sbin/stop-dn.sh
```

4.2.4 HopsFS Clients

For load balancing the clients uniformly distributes the filesystem operations among all the NameNodes in the system. HopsFS clients support **RANDOM**, **ROUND_ROBIN**, and **RANDOM_STICKY** policies to distribute the filesystem operations among the NameNodes. Random and round-robin policies are self explanatory. Using sticky policy the filesystem client randomly picks a NameNode and forwards all subsequent operation to the same NameNode. If the NameNode fails then the clients randomly picks another NameNode. This maximizes the NameNode cache hits.

In HDFS the client connects to the `fs.defaultFS` NameNode. In HopsFS, clients obtain the list of active NameNodes from the NameNode defined using `fs.defaultFS` parameter. The client then uniformly distributes the subsequent filesystem operations among the list of NameNodes.

In `core-site.xml` we have introduced a new parameter `dfs.namenodes.rpc.addresses` that holds the rpc address of all the NameNodes in the system. If the NameNode pointed by `fs.defaultFS` is dead then the client tries to connect to a NameNode defined by the `dfs.namenodes.rpc.addresses`. As long as the NameNode addresses defined by the two parameters contain at least one valid address the client is able to communicate with the HopsFS. A detailed description of all the new client configuration parameters are [here](#) (page 117).

4.2.5 HopsFS Client API

```java
FileSystem fs = FileSystem.get(new Configuration());
```

4.2.6 Compatibility with HDFS Clients

HopsFS is fully compatible with HDFS clients, although they do not distribute operations over NameNodes, as they assume there is a single active NameNode.

4.2.7 HopsFS Async Quota Management

In HopsFS the commands and the APIs for quota management are identical to HDFS. In HDFS all Quota management operations are performed synchronously while in HopsFS Quota management is performed asynchronously for performance reasons. In the following example maximum namespace quota for `/QDir`
is set to 10. When a new sub-directory or a file is created in this folder then the quota update information propagates up the filesystem tree until it reaches /QDir. Each quota update propagation operation is implemented as an independent transaction.

For write heavy workloads a user might be able to consume more disk space/namespace than it is allowed before the filesystem recognizes that the quota limits have been violated. After the quota updates are applied the filesystem will not allow the use to further violate the quota limits. In most existing Hadoop clusters, write operations are a small fraction of the workload. Additionally, considering the size of the filesystem we think this is a small trade off for improving throughput for read operations that typically comprise 90-95% of a typical filesystem workload.

In HopsFS asynchronous quota updates are highly optimized. We batch the quota updates wherever possible. In the linked section (page 115) there is a complete list of parameters that determines how aggressively asynchronous quota updates are applied.

### 4.2.8 Block Reporting

DataNodes periodically synchronize the set of blocks stored locally with the metadata representing those blocks using a block report. Block reports are sent from DataNodes to NameNodes to indicate the set of valid blocks at a DataNode, and the NameNode compares the sent list with its metadata. For block report load balancing the DataNodes ask the leader NameNode to which NameNode they should send the block report. The leader NameNode uses round robin policy to distribute block reports among the NameNodes. In order to avoid sudden influx of large number of block reports that can slow down the performance of other filesystem operations the leader NameNode also performs admission control for block reports. The leader NameNode only allows certain number of block reports, which is configurable, to be processed at a given time. In the linked section (page 116) there is a complete list of parameters for block report admission control.

HopsFS Block Reporting uses a mechanism for skipping redundant processing, based on the partitioning of the block ID space and maintaining a set of hashes on the NameNode. Based on your cluster’s average load, the number of partitions can be fine-tuned but a number between 1-4k partitions should be sufficient for most applications. The hashes are stored as long numerical type in the database, one set per storage volume. Metadata storage is sized in the order of #storageVolumes x numPartitions x (4+4+8)Bytes.

### 4.2.9 HopsFS ACLs

HopsFS supports extended Access Control Lists (ACLs), in similar fashion to the feature introduced in Apache Hadoop 2.4.0. Access Control Lists are a way of extending the standard unix file permissions, allowing users to extend file access based on additional group and user restrictions.

The implementation in HopsFS uses the same API as Apache Hadoop, but with slightly changed semantics.

#### API

Below you can find a summary of the most important commands.

```
hdfs dfs -getfacl /path # get acl status
hdfs dfs -setfacl -m user::rw-,user:hadoop:rw- /path #modify ACL
hdfs dfs -setfacl -k /path # apply recursively
hdfs dfs -setfacl -R ... /path # remove default ACL
hdfs dfs -setfacl --set ... /path # fully replace acl
```
Fig. 4: Google authenticator one-time password
Fig. 5: Hopsworks User Registration Page

Fig. 6: Two-factor authentication: Scan the QR Code with Google Authenticator
Fig. 7: Project main page

Fig. 8: Project Menu
Fig. 9: Datasets Browser

Fig. 10: Upload file(s)
Fig. 11: Upload screen

Fig. 12: Share dataset
Fig. 13: Guided tours

Fig. 14: Job UI

Fig. 15: Job Service - Workflow Example

Fig. 16: Job Details
Fig. 17: Job logs with Kibana

Fig. 18: Kibana visualization
Fig. 19: Grafana dashboard

Fig. 20: Open the Jupyter service on Hopsworks
Fig. 21: Start a Jupyter notebook server

Fig. 22: Create a pyspark notebook
Plotting With SparkMagic on Hops

To run large scale computations in a hops cluster from Jupyter we use sparkmagic, a Ivy REST server, and the pyspark kernel.

The fact that the default computation on a cluster is distributed over several machines makes it a little different to do things such as plotting compared to when running code locally.

This notebook illustrates how you can combine plotting and large-scale computations on a Hops cluster in a single notebook.

```
In [1]: # When pyspark kernel is started we get a Spark session automatically created for us

    Starting Spark application

    ID   YARN Application ID   Kind   State   Spark UI   Driver log   Current session?
    23   application_1598605120096_0023 pyspark  idle   Link   Link

    SparkSession available as 'spark'.
    <pyspark.sql.session.SparkSession object at 0x7f5d693074e0>
```

Fig. 23: SparkSession creation with pyspark kernel

```
In [1]: # When pyspark kernel is started we get a Spark session automatically created for us

    Starting Spark application

    ID   YARN Application ID   Kind   State   Spark UI   Driver log   Current session?
    23   application_1598605120096_0023 pyspark  idle   Link   Link

    SparkSession available as 'spark'.
    <pyspark.sql.session.SparkSession object at 0x7f62f6d80f08>

In [2]: x = [1, 2, 3]
    
    print("x: {}").format(x)
    
    x: [1, 2, 3]
```

Fig. 24: Executing python code on the spark driver in the cluster

```
In [2]: x = [1, 2, 3]
    
    print("x: {}").format(x)
    
    x: [1, 2, 3]

In [6]: x2 = spark.sparkContext.parallelize(range(0, 10)).map(lambda x: x*x).collect()

x2

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Fig. 25: Starting a spark job from Jupyter

4.3. Hops-YARN User Guide
Fig. 26: Opening the SparkUI in Hopworks

Fig. 27: The SparkUI in Hopworks
Check which "magic" functions are available from sparkmagic

```
In [2]: %help
```

<table>
<thead>
<tr>
<th>Magic</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>info</td>
<td>%ls</td>
<td>Outputs session information for the current Livy endpoint.</td>
</tr>
<tr>
<td>cleanup</td>
<td>%lc</td>
<td>Deletes all sessions for the current Livy endpoint, including this notebook's session. The force flag is mandatory.</td>
</tr>
<tr>
<td>delete</td>
<td>%ldelete</td>
<td>Deletes a session by number for the current Livy endpoint. Cannot delete this notebook's session.</td>
</tr>
<tr>
<td>logs</td>
<td>%ll</td>
<td>Outputs the current session's Livy logs.</td>
</tr>
<tr>
<td>configure</td>
<td>%lconf</td>
<td>Configure the session creation parameters. The force flag is mandatory if a session has already been created and the session will be stopped and restarted. Look at Livy's POST Sessions Request Body for a list of valid parameters. Parameters must be passed in as a JSON string.</td>
</tr>
<tr>
<td>spark</td>
<td>%ls</td>
<td>Creates spark commands. Parameters:</td>
</tr>
<tr>
<td></td>
<td>-o VAR_NAME: The Spark dataframe of name VAR_NAME will be available in the %val in the notebook context as a Spark dataframe with the same name.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-m METHOD: Sample method, either take or sample.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-n MAXROWS: The maximum number of rows of a dataframe that will be pulled from Livy to Apache. If this number is negative, then the number of rows will be unlimited.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-w FRACTION: Fraction used for sampling.</td>
<td></td>
</tr>
<tr>
<td>sql</td>
<td>%ls</td>
<td>Executes a SQL query against the variable sqlContext (Spark v2.x) or spark (Spark v1.x). Parameters:</td>
</tr>
<tr>
<td></td>
<td>-o VAR_NAME: The result of the SQL query will be available in the %val in the notebook context as a Spark dataframe.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-n, m, k are the same as the %lspar parameters above.</td>
<td></td>
</tr>
<tr>
<td>local</td>
<td>%local</td>
<td>All the code in subsequent lines will be executed locally. Code must be valid Python code.</td>
</tr>
</tbody>
</table>

Fig. 28: Printing a list of all sparkmagic commands

Load a CSV file in Spark

The pokemon csv file is available here

```
In [3]: from hops import hdfs
df = spark.sql("csv", option="header", "true").load(hdfs.project_path() + "TestJob/data/visualization/Pokemon.csv")
```

```
In [4]: df.count()

880
```

```
In [5]: df.show(5)
```

```
<table>
<thead>
<tr>
<th></th>
<th>Name</th>
<th>Type</th>
<th>Type</th>
<th>Total</th>
<th>HP</th>
<th>Attack</th>
<th>Defense</th>
<th>Sp. Atk</th>
<th>Sp. Def</th>
<th>Speed</th>
<th>Generation</th>
<th>Legendary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bulbasaur</td>
<td>Grass</td>
<td>Poison</td>
<td>318</td>
<td>49</td>
<td>49</td>
<td>65</td>
<td>65</td>
<td>45</td>
<td>1</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Ivysaur</td>
<td>Grass</td>
<td>Poison</td>
<td>495</td>
<td>66</td>
<td>62</td>
<td>83</td>
<td>80</td>
<td>66</td>
<td>1</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Venusaur</td>
<td>Grass</td>
<td>Poison</td>
<td>525</td>
<td>88</td>
<td>82</td>
<td>100</td>
<td>100</td>
<td>88</td>
<td>1</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>3V</td>
<td>Venusaur</td>
<td>Grass</td>
<td>Poison</td>
<td>525</td>
<td>88</td>
<td>82</td>
<td>100</td>
<td>100</td>
<td>88</td>
<td>1</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Charmander</td>
<td>Fire</td>
<td>null</td>
<td>399</td>
<td>39</td>
<td>52</td>
<td>43</td>
<td>69</td>
<td>50</td>
<td>1</td>
<td>False</td>
<td></td>
</tr>
</tbody>
</table>
```

only showing top 5 rows

Fig. 29: Creating a spark dataframe
Name the Spark DataFrame to Be Able to Use SQL

```
In [10]: df.createOrReplaceTempView('pokemon')
```

Use SparkMagic to Collect the Spark dataframe as a Pandas Dataframe Locally

This command will send the dataset from the cluster to the server where Jupyter is running and convert it into a pandas dataframe. This is only suitable for smaller datasets. A common practice is to run spark jobs to process a large dataset and shrink it before plotting, notice that in this case we use the \--maxrows 10 flag to limit the amount of data we download. The command below makes the result of the SQL query available as a pandas dataframe called `python_df` in `%local`.

```
In [11]: %sql -c sql -o python_df --maxrows 10 SELECT * FROM pokemon
```

![Fig. 30: Downloading the spark dataframe to a pandas dataframe using %%%sql](image)

Fig. 30: Downloading the spark dataframe to a pandas dataframe using %%%sql
In [15]: %spark --e df

The Pandas DataFrames are now Available in %%%local mode

In [16]: %local
df

<table>
<thead>
<tr>
<th>Type</th>
<th>Bar</th>
<th>Pie</th>
<th>Table</th>
<th>Scatter</th>
<th>Line</th>
<th>Area</th>
</tr>
</thead>
</table>

Encoding:

- **X**: Name
- **Y**: #

Fig. 31: Downloading the spark dataframe to a pandas dataframe using %%%spark

Local Plotting with Matplotlib and Seaborn

After the Data Have Been Loaded Locally as a pandas dataframe, it can get plotted on the Jupyter server. By using the magic "%%local" at the top of a cell, the code in the cell will be executed locally on the Jupyter server, rather than remotely with Livy on the Spark cluster. Once the pandas dataframe is available locally, it can be plotted with libraries such as matplotlib and seaborn. *remember to add the line: %matplotlib inline*

In [19]: %local

```python
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
```

Fig. 32: Import plotting libraries locally on the Jupyter server
Fig. 33: Plot a local pandas dataframe using seaborn and the magic `%%local`

```
In [15]: %%local
    # view the number of pokemons for Type 1 and Type 2 using one plot
    f, (ax1, ax2) = plt.subplots(2, 1, figsize=(15, 8), sharex=True)
    sns.countplot('Type 1', data=df, ax=ax1)
    sns.countplot('Type 2', data=df, ax=ax2)
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x7f535e23c850>
```

![Plot 1](image1.png)

![Plot 2](image2.png)

Fig. 33: Plot a local pandas dataframe using seaborn and the magic `%%local`

```
In [16]: %%local
    stats = df.columns[5:11]
    plt.figure(figsize=(25, 20))
    for ii, stat in enumerate(stats):
        title = "Distributions of {stat}".format(stat=stat)
        x = plt.gca().get_xlim()[0] + .6
        y = plt.gca().get_ylim()[1] - .9
        stats = np.array([np.random.normal(mu, sigma) for i in range(len(df))])
        sns.distplot(stats, x=x, y=y, *args, **kwargs)
    plt.tight_layout()
    plt.show()
```

![Distributions](image3.png)

Fig. 34: Plot a local pandas dataframe using matplotlib and the magic `%%local`
Fig. 35: Kafka topics & schemas
4.3. Hops-YARN User Guide

Fig. 36: Kafka main page

Deny access for write operations for every role originating from every host of the project another_sample_project and every member of that project

Fig. 37: Kafka ACL example

Fig. 38: Kafka topic details
### 4.3. Hops-YARN User Guide

#### Fig. 39: Kafka producer job

#### Fig. 40: Kafka output
Fig. 41: A feature store is the interface between feature engineering and model development.
Fig. 42: Create a new project with the feature store service enabled.

Fig. 43: Opening the feature store registry in Hopsworks.

Fig. 44: Concepts for modeling data in the feature store.
Fig. 45: Users query the feature store programmatically or using SQL. The output is provided as Pandas, Numpy or Spark dataframes.

Fig. 46: The life-cycle of data in HopsML. Raw data is transformed into features which are grouped together into training datasets that are used to train models.

Fig. 47: Searching for features in the feature registry.
Fig. 48: Opening that statistics for a feature group.

Fig. 49: Storage Connectors can be configured from the Feature Store UI in Hopsworks.
Fig. 50: Based on the organization need, features can be divided into several feature stores to preserve data access control.

Fig. 51: Feature stores can be shared across project boundaries.
Fig. 52: Selecting a feature store in the feature registry.
5.1 Python-First ML Pipelines

HopsML is a Python-first framework for building machine learning pipelines. HopsML is enabled by unique support for project-specific conda environments in Hopsworks. As every project in Hopsworks has its own conda environment, replicated at all data processing hosts in the cluster, Data Scientists can simply ‘pip install’ Python libraries in Hopsworks and immediately use them in their PySpark/TensorFlow/PyTorch applications, enabling interactive application development. This contrasts with an immutable infrastructure approach, where Data Scientists need to write Dockerfiles and write YML files describing cluster specifications just to install a Python library. The other unique aspect of HopsML is the use of HopsFS (a distributed filesystem) to coordinate the different steps in a pipeline. HopsFS integrates seamlessly with Estimator APIs in TensorFlow/Keras, enabling the transparent management and aggregation of logs, checkpoints, TensorBoard events, and models across many Executors in a cluster. HopsML extends these Estimator artifacts with versioned notebooks and Python environments, enabling a view of experiments that have been run and now can be easily reproduced.

A machine learning (ML) pipeline is a series of processing steps that:

- optionally ingests (raw) input data from external sources,
• wrangles the input data in an ETL job (data cleaning/validation, feature extraction, etc) to generate clean training data,
• trains a model (using GPUs) with the clean training data,
• validates and optimizes the model,
• deploys the model to production,
• monitors model performance in production.

HopsML pipelines are written as a different programs for each stage in the pipeline, and the pipeline itself is written as a Airflow DAGs (directed acyclic graph). Typically all programs in the pipeline are written in Python, although Scala/Java ca be used at the ETL stage, in particular when dealing with large volumes of input data.

For ML pipelines processing small amounts of data, developers can write a Keras/TensorFlow/PyTorch application to perform both ETL and training in a single program, although developers should be careful that the ETL stage is not so CPU intensive that GPUs cannot be fully utilized when training. For example, in an image processing pipeline, if the same Keras/TensorFlow/PyTorch application is used to both decode/scale/rotate images as well as train a deep neural network (DNN), the application will probably be CPU-bound or I/O bound, and GPUs will be underutilized.

For ML pipelines processing large amounts of data, developers can write a separate Spark or PySpark application to perform ETL and generate training data. When that application has completed, Airflow will then schedule a PySpark application with Keras/TensorFlow/PyTorch to train the DNN, on possibly many GPUs. The training data will be read from a distributed filesystem (HopsFS), and all logs, TensorBoard events, checkpoints, and the model will be written to the same distributed filesystem. When training has completed, Airflow can schedule a simple Python/Bash job to optimize the trained model (e.g., quantize model weights, remove batch norm layers, shrink models for mobile devices), using either Nvidia’s TensorRT library or TensorFlow’s transform_graph utility. The optimized model (a .pb (protocol buffers) file in TensorFlow) can then be deployed directly from HopsFS to a model serving server (TensorFlow serving Server on Kubernetes) using a REST call on Hopsworks. Finally, Airflow can start a Spark Streaming job to monitor the deployed model by consuing logs for the deployed model from Kafka.

HopsML uses HopsFS, a next-generation version of HDFS, to coordinate the different steps of an ML pipeline. Input data for pipelines can come from external sources, such as an existing Hadoop cluster or a S3 datalake, a feature store, or existing training datasets. External datasources can push data to HopsFS using either the Hopsworks REST-API or using Kafka in Hopsworks.
During a ML pipeline HopsFS acts as a central coordinator for sharing data between the different stages. Examples of such data include features from the store, existing training data, PySpark/TensorFlow application logs, TensorFlow events (aggregate from many different executors/GPUs), output models, checkpoints, partial/full results from hyperparameter optimization.

### 5.2 Hops Python Library

The Hops Python Library simply named hops is used for running Python applications and consequently a library which is used throughout the entire pipeline. It simplifies interacting with services such as Kafka, Model Serving and TensorBoard. The experiment module provides a rich API for running versioned Machine Learning experiments, whether it be a simple single-process Python application or RingAllReduce over many machines.

**Documentation:** [hops-py](http://hops-py.logicalclocks.com)

**Code examples:** [hops-examples](https://github.com/logicalclocks/hops-examples/tree/master/tensorflow/notebooks)

### 5.3 Data Collection

The datasets that you are working with will reside in your project in HopsFS. Data can be uploaded to your project in a number of ways, such as using the hops-cli client, the REST API or the uploader in the Hopsworks UI. HopsFS is the filesystem of Hops, it is essentially an optimized fork of Apache HDFS, and is compliant with any API that can read data from an HDFS path, such as TensorFlow, Spark and Pandas.

### 5.4 Data Preparation

It is important to validate the datasets used in your pipeline, for example imbalanced classes may lead to Machine Learning models being biased towards more frequently occurring labels in the dataset. Therefore it is of outmost importance for input data to be balanced and representative of the domain from which the data came. One of the big steps toward ensuring the correctness of data is through data quality and validation checks. Machine Learning models, as have been observed empirically and in papers, reduce their generalization error for larger datasets. Therefore it is also critical to have a data wrangling and validation engine which scales for ever increasing datasets. The solution for this is to go distributed in order to process every single record, but still have a rich API for perform quality checks and manipulating the data. The pipeline makes use of Spark to provide these capabilities.

Spark Dataframes can be used to transform and validate large datasets in a distributed manner. For example schemas can be used to validate the datasets. Useful insights can be calculated such as class imbalance, null values for fields and making sure values are inside certain ranges. Datasets can be transformed by dropping or filtering fields.

---

52 [http://hops-py.logicalclocks.com](http://hops-py.logicalclocks.com)
For visualizations on datasets, see spark-magic\textsuperscript{55} or facets\textsuperscript{56} examples here.

## 5.5 Feature Store

“A feature store allows teams to share, discover, and use a highly curated set of features for their machine learning problems”

Michelangelo\textsuperscript{57} by Uber

Hopsworks provides a feature store to curate, store, and document features for use in ML pipelines. The feature store serves as the interface between data engineering and data science in HopsML pipelines. The feature store requires a change in mindset for data engineering and data scientists, instead of writing custom pipelines where models have their own feature storage, it is encouraged to assemble all features in the feature store so that features can be shared between several models and projects.

The Feature Store enables the following best-practices for feature engineering:

1. Feature Reuse/Collaboration,
2. Feature Documentation,
3. Feature Backfilling,
4. Feature Versioning,
5. Automatic Feature Analysis,
6. DRY (not repeat yourself) feature engineering.

### Raw/Structured Data

![Diagram showing the process of raw data being transformed through feature engineering, stored in a feature store, and then used for training models.]

### Models

![Diagram of models with features and parameters connected, highlighting the flow from feature engineering to training.]

## 5.6 Experimentation

This section will give an overview of running Machine Learning experiments on Hops.

In HopsML we offer a rich experiment\textsuperscript{58} API for data scientists to run their Machine Learning code, whether it be TensorFlow, Keras PyTorch or another framework with a Python API. To mention some of features it provides versioning of notebooks and other resources, AutoML algorithms that will find the best hyperparameters for your model and managing TensorBoard.

\textsuperscript{55} https://github.com/logicalclocks/hops-examples/blob/master/tensorflow/notebooks/Plotting/Data_Visualizations.ipynb
\textsuperscript{56} https://github.com/logicalclocks/hops-examples/blob/master/tensorflow/notebooks/Plotting/facets-overview.ipynb
\textsuperscript{57} https://eng.uber.com/michelangelo
\textsuperscript{58} http://hops-py.logicalclocks.com/hops.html#module-hops.experiment

---

### 5.5. Feature Store
Hops uses PySpark to manage resource allocation of CPU, Memory and GPUs. PySpark is also used to transparently distribute the Python code making up the experiment to Executors which executes it. Certain hyperparameter optimization algorithms such as random search and grid search are parallelizable by nature, which means that different Executors will run different hyperparameter combinations. If a particular Executor sits idle it will be reclaimed by the cluster, which means that GPUs will be optimally used in the cluster. This is made possible by Dynamic Spark Executors.

Hops supports cluster-wide Conda for managing Python library dependencies. Hops supports the creation of projects, and each project has its own conda environment, replicated at all hosts in the cluster. When you launch a PySpark job, it uses the local conda environment for that project. This way, users can install whatever libraries they like using conda and pip package managers, and then use them directly inside Spark Executors. It makes programming PySpark one step closer to the single-host experience of programming Python.

HopsML comes with a novel Experiments service for overviewing history of Machine Learning experiments and monitoring during training.

The following is a TensorBoard visualizing Differential Evolution for hyperparameter optimization on two PySpark Executors. The X-axis being the wall-clock time.
5.7 Serving

In the pipeline we support a scalable architecture for serving of TensorFlow and Keras models. We use the TensorFlow Serving server running on Kubernetes to scale up the number of serving instances dynamically and handle load balancing. There is support for using either the gRPC client or the REST API to send inference requests. Furthermore we also support a monitoring system that logs the inference requests and allows users to implement custom functionality for retraining of models.

Model Monitoring

See `tf_model_serving`, `sklearn_model_serving` and `inferencing` for more information.
5.8 Pipeline Orchestration

HopsML pipelines are typically run as Airflow DAGs, written in Python. An Airflow pipeline is a directed acyclic graph (DAG) of tasks to be executed, orchestration rules, failure handling logic, and notifications. Airflow DAGs can be scheduled to run periodically, for example, once per hour, or Airflow can wait for an event (with sensors) before executing a task - for example, wait for _SUCCESS file in a parquet directory before understanding that the Parquet file(s) are finished being written. Typical tasks in a production Airflow ML pipeline on Hops involve Data Prep as a PySpark job, training using HopsML (PySpark + TensorFlow), model optimization using a PySpark job or a bash job, and model deployment as either a Python program or bash script.
6.1 User Administration

6.1.1 Activating users

You, the administrator, have to approve each new user account before the user is able to login to Hopsworks. When you approve the account, you have to assign a role to a user as either an:

- user
- administrator

Users that are assigned an administrator role will be granted privileges to login to the administrator application and control users and the system. Be careful in which users are assigned an administrator role. The vast majority of users will be assigned a user role.

![Mobile User Administration](image)

Fig. 1: Approve User Accounts so that Users are able to Login

6.1.2 User fails to receive an email to validate her account

- Does your organization have a firewall that blocks outbound SMTP access?
• Login to the Glassfish Webserver and check the JavaMail settings. The JNDI name should be `mail/BBCMail`. Is the gmail username/password correct? Are the smtp server settings correct (ip-address or hostname, port, protocol (SSL, TLS))? 

### 6.1.3 User receives email, but fails to validate the account

• Can you successfully access the Hopsworks homepage?
• Is the Glassfish webserver running and hopsworks.war application installed?
• Is MySQL Cluster running?

### 6.1.4 Configuring email for Hopsworks

Login to Glassfish, see *Glassfish Administration* (page 111), and update the JavaMail settings to set the email account, password, SMTP server IP and port, and whether SSL/TLS are used.

### 6.1.5 User successfully validates the account, but still can’t login

Go to the User Administration view. From here, select the user whose account will be enabled, and update the user’s account status to validated.

### 6.1.6 User account has been disabled due to too many unsuccessful login attempts

Go to the User Administration view. From here, select the user whose account will be re-enabled, and update the user’s account status to validated.

### 6.1.7 Disabling a user account

Go to the User Administration view. From here, select the user whose account will be disabled, and update the user’s account status to disabled.

### 6.1.8 Re-activating a user account

In the user administration view, you can select the action that changes the user status to activated.

### 6.1.9 Managing Project Quotas

Each project is by default allocated a number of CPU hours in HopsYARN and an amount of available disk storage space in HopsFS:

• HopsYARN Quota
• HopsFS Quota
We recommend that you override the default values for the Quota during the installation process, by overriding the Chef attributes:

- `hopsworks/yarn_default_quota`
- `hopsworks/hdfs_default_quota`

In the **Projects** view, for any given project, the administrator can update the remaining amount of HopsYARN Quota (in CPU hours) and the amount disk space allocated in HopsFS for the project.

**Fig. 2: Project Administration: update quotas, disable/enable projects.**

### 6.1.10 Disabling/Re-enabling Projects

In the **Projects** view, any given project can be disabled (and subsequently reenabled). Disabling a project will prevent members of the project from accessing data in the project, running Jobs stored in the project, or accessing the project at all.

### 6.1.11 Ubikeys in Hopsworks

Ubikeys can be used as the 2nd factor authentication device, but a Ubikey needs to be programmed before it is given to a user. We recommend programming the Ubikey using Ubuntu’s Yubikey OTP tool. From the Yubikey OTP tool program, you will have to copy the Public Identity and Secret Key fields (from Yubikey OTP) to the corresponding fields in the Hopsworks Administration tool when you validate a user. That is, you should save the Public Identity and Secret Key fields for the Yubikey sticks, and when a user registers with one of those Ubikey sticks, you should then enter the Public Identity and Secret Key fields when approving the user’s account.

```
$ sudo apt-get install yubikey-personalization-gui
$ yubikey-personalization-gui
```

Installing and starting Yubikey OTP tool in Ubuntu.

### Contents

- *Glassfish Administration* (page 111)
Glassfish Administration

If you didn’t supply your own username/password for Glassfish administration during installation, you can login with the default username and password for Glassfish:

<table>
<thead>
<tr>
<th>Glassfish URL</th>
<th><a href="https://localhost:4848">https://localhost:4848</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>adminuser</td>
</tr>
<tr>
<td>password</td>
<td>adminpw</td>
</tr>
</tbody>
</table>

Users are referred to Glassfish documentation for more information regarding configuring Glassfish.
6.2 HopsFS Configuration

6.2.1 Small Files Support

HopsFS can store small files in the database with the file system metadata to increase the performance of small files. To enable this feature use the `DB` storage type for the folder that would store small files. For example,

```bash
$ hdfs storagepolicies -setStoragePolicy -path <path> -policy DB
```

or using Java API

```java
distributedFileSystem.setStoragePolicy(path, "DB");
```

When the storage policy for a folder is set to `DB` then all the small files (Default <= 64KB) will be stored in the database. Setting the storage policy for a parent folder is sufficient, that is, if no explicit storage policy is defined for descendant folders then the descendant folders will inherit the storage policy from its parent.
Fig. 5: Copy the Public Identity and Secret Key fields from Yubikey OTP tool and enter them into the corresponding fields in the HopsWork’s Administration UI when you validate a user.

folders. The default size of small files can be changed using the following property in the `hdfs-site.xml`:

```xml
<property>
  <name>dfs.db.file.max.size</name>
  <value>65536</value>
  <description>Largest file that can be stored in the database</description>
</property>

NOTE: Changing this parameter requires restarting HopsFS Cluster and all the services that access the file system.

The small files are stored in NDB disk data tables\(^\text{59}\). The default tablespace and log file group names are `ts_1` and `lg_1` respectively. In the NDB configuration set the `FileSystemPathDD` parameter to store the disk data files on a high-performance NVMe SSD drive. The disk data files can also be set in


6.2. HopsFS Configuration
the NDB cluster definition\(^{60}\) using Chef.

```yaml
attrs:
  ndb:
    ...
  nvme:
    disks:
      - '/dev/nvme0n1'
    format: true
    undofile_size: SIZE
    logfile_size: SIZE
```

Size of undo log file is set using `undofile_size` and the size of NDB disk data files is set using `logfile_size`.

### 6.2.2 Leader Election

Leader election service is used by HopsFS and Hops-YARN. The configuration parameters for Leader Election service are defined in `core-site.xml` file.

- **dfs.leader.check.interval**: The length of the time period in milliseconds after which NameNodes run the leader election protocol. One of the active NameNodes is chosen as a leader to perform housekeeping operations. All NameNodes periodically update a counter in the database to mark that they are active. All NameNodes also periodically check for changes in the membership of the NameNodes. By default the time period is set to one second. Increasing the time interval leads to slow failure detection.

- **dfs.leader.missed hb**: This property specifies when a NameNode is declared dead. By default a NameNode is declared dead if it misses two consecutive heartbeats. Higher values of this property would lead to slower failure detection. The minimum supported value is 2.

- **dfs.leader.tp.increment**: HopsFS uses an eventual leader election algorithm where the heartbeat time period (`dfs.leader.check.interval`) is automatically incremented if it detects that the NameNodes are falsely declared dead due to missed heartbeats caused by network/database/CPU overload. By default the heartbeat time period is incremented by 100 milliseconds, however it can be overridden using this parameter.

### 6.2.3 NameNode Cache

The NameNode cache configuration parameters are defined in `hdfs-site.xml` file. The NameNode cache configuration parameters are:

- **dfs.resolvingcache.enabled**: (true/false) Enable/Disables the cache for the NameNode.

- **dfs.resolvingcache.type**: Each NameNode caches the inodes metadata in a local cache for quick path resolution. We support different implementations for the cache, i.e., INodeMemcache, PathMemcache, OptimalMemcache and InMemory.
  - **INodeMemcache**: stores individual inodes in Memcached.

---

\(^{60}\) https://github.com/logicalclocks/ndb-chef
- **PathMemcache**: is a coarse grain cache where entire file path (key) along with its associated inodes objects are stored in the Memcached.

- **OptimalMemcache**: combines INodeMemcache and PathMemcache.

- **InMemory**: Same as INodeMemcache but instead of using Memcached it uses an in-memory LRU ConcurrentLinkedHashMap. We recommend InMemory cache as it yields higher throughput.

For INodeMemcache/PathMemcache/OptimalMemcache following configurations parameters must be set.

- `dfs.resolvingcache.memcached.server.address`: Memcached server address.

- `dfs.resolvingcache.memcached.connectionpool.size`: Number of connections to the memcached server.

- `dfs.resolvingcache.memcached.key.expiry`: It determines when the memcached entries expire. The default value is 0, that is, the entries never expire. Whenever the NameNode encounters an entry that is no longer valid, it updates it.

The InMemory cache specific configurations are:

- `dfs.resolvingcache.inmemory.maxsize`: Max number of entries that could be stored in the cache before the LRU algorithm kicks in.

### 6.2.4 Distributed Transaction Hints

In HopsFS the metadata is partitioned using the inodes’ id. HopsFS tries to to enlist the transactional filesystem operation on the database node that holds the metadata for the file/directory being manipulated by the operation. Distributed transaction hints configuration parameters are defined in `hdfs-site.xml` file.

- `dfs.ndb.setpartitionkey.enabled`: (true/false) Enable/Disable transaction partition key hint.

- `dfs.ndb.setrandompartitionkey.enabled`: (true/false) Enable/Disable random partition key hint when HopsFS fails to determine appropriate partition key for the transactional filesystem operation.

### 6.2.5 Quota Management

In order to boost the performance and increase the parallelism of metadata operations the quota updates are applied asynchronously i.e. disk and namespace usage statistics are asynchronously updated in the background. Using asynchronous quota system it is possible that some users over consume namespace/disk space before the background quota system throws an exception. Following parameters controls how aggressively the quota subsystem updates the quota statistics. Quota management configuration parameters are defined in `hdfs-site.xml` file.

- `dfs.quota.enabled`: Enable/Disabled quota. By default quota is enabled.

- `dfs.namenode.quota.update.interval`: The quota update manager applies the outstanding quota updates after every `dfs.namenode.quota.update.interval` milliseconds.

- `dfs.namenode.quota.update.limit`: The maximum number of outstanding quota updates that are applied in each round.
6.2.6 Block Reporting

- **dfs.block.report.load.balancing.max.blks.per.time.window**: This is a global configuration parameter. The leader NameNode only allows certain number of blocks reports such that the maximum number of blocks that are processed by the block reporting sub-system of HopsFS does not exceed **dfs.block.report.load.balancing.max.blks.per.time.window** in a given block report processing time window.

- **dfs.block.report.load.balancing.time.window.size** This parameter determines the block report processing time window size. It is defined in milliseconds. If **dfs.block.report.load.balancing.max.blks.per.time.window** is set to one million and **dfs.block.report.load.balancing.time.window.size** is set to one minutes then the leader NameNode will ensure that at every minute at most 1 million blocks are accepted for processing by the admission control system of the filesystem.

- **dfs.blk.report.load.balancing.update.threashold.time** Using command `hdfs namenode -setBlkRptProcessSize noOfBlks` the parameter **dfs.block.report.load.balancing.max.blks.per.time.window** can be changed. The parameter is stored in the database and the NameNodes periodically read the new value from the database. This parameter determines how frequently a NameNode checks for changes in this parameter. The default is set to 60*1000 milliseconds. **dfs.blockreport.numbuckets** This parameter defines the number of buckets in the hash-based report. Reconfiguration requires a complete restart of the cluster, and datanodes and namenodes alike must share the same configuration value. ONLY USE THIS IN VERSION 2.8.2.5+.

6.2.7 Distributed Unique ID generator

ClusterJ API does not support any means to auto generate primary keys. Unique key generation is left to the application. Each NameNode has an ID generation daemon. ID generator keeps pools of pre-allocated IDs. The ID generation daemon keeps track of IDs for inodes, blocks and quota entities. Distributed unique ID generator configuration parameters are defined in **hdfs-site.xml**.

- **dfs.namenode.quota.update.id.batchsize**, **dfs.namenode.inodeid.batchsize**, **dfs.namenode.blockid.batchsize**: When the ID generator is about to run out of the IDs it prefetches a batch of new IDs. These parameters defines the prefetch batch size for Quota, inodes and blocks updates respectively.

- **dfs.namenode.quota.update.updateThreshold**, **dfs.namenode.inodeid.updateThreshold**, **dfs.namenode.blockid.updateThreshold**: These parameters define when the ID generator should pre-fetch new batch of IDs. Values for these parameter are defined as percentages i.e. 0.5 means pre-fetch new batch of IDs if 50 percent of the IDs have been consumed by the NameNode.

- **dfs.namenode.id.updateThreshold**: It defines how often the IDs Monitor should check if the ID pools are running low on pre-allocated IDs.

6.2.8 Namespace and Block Pool ID

- **dfs.block.pool.id** and **dfs.name.space.id**: Due to shared state among the NameNodes, HopsFS only supports single namespace and one block pool. The default namespace and block pool ids can be
overridden using these parameters.

### 6.2.9 Client Configurations

All the client configuration parameters are defined in `core-site.xml` file.

- **dfs.namenodes.rpc.addresses**: HopsFS support multiple active NameNodes. A client can send a RPC request to any of the active NameNodes. This parameter specifies a list of active NameNodes in the system. The list has following format `[hdfs://ip:port, hdfs://ip:port, ...]`. It is not necessary that this list contain all the active NameNodes in the system. Single valid reference to an active NameNode is sufficient. At the time of startup the client obtains an updated list of NameNodes from a NameNode mentioned in the list. If this list is empty then the client tries to connect to `fs.default.name`.

- **dfs.namenode.selector-policy**: The clients uniformly distribute the RPC calls among the all the NameNodes in the system based on the following policies. - ROUND ROBIN - RANDOM - RANDOM_STICKY By default NameNode selection policy is set to RANDOM_STICKY

- **dfs.client.max.retries.on.failure**: The client retries the RPC call if the RPC fails due to the failure of the NameNode. This configuration parameter specifies how many times the client would retry the RPC before throwing an exception. This property is directly related to number of expected simultaneous failures of NameNodes. Set this value to 1 in case of low failure rates such as one dead NameNode at any given time. It is recommended that this property must be set to value $\geq 1$.

- **dfs.client.max.random.wait.on.retry**: A RPC can fail because of many factors such as NameNode failure, network congestion etc. Changes in the membership of NameNodes can lead to contention on the remaining NameNodes. In order to avoid contention on the remaining NameNodes in the system the client would randomly wait between $[0,\text{MAX VALUE}]$ ms before retrying the RPC. This property specifies MAX VALUE; by default it is set to 1000 ms.

- **dfs.client.refresh.namenode.list**: All clients periodically refresh their view of active NameNodes in the system. By default after every minute the client checks for changes in the membership of the NameNodes. Higher values can be chosen for scenarios where the membership does not change frequently.

### 6.2.10 Data Access Layer (DAL)

Using DAL layer HopsFS’s metadata can be stored in different databases. HopsFS provides a driver to store the metadata in MySQL Cluster Network Database (NDB).
MySQL Cluster Network Database Driver Configuration

Database specific parameter are stored in a .properties file. The configuration files contains following parameters.

- **com.mysql.clusterj.connectstring**: Address of management server of MySQL NDB Cluster.
- **com.mysql.clusterj.database**: Name of the database schema that contains the metadata tables.
- **com.mysql.clusterj.connection.pool.size**: This is the number of connections that are created in the ClusterJ connection pool. If it is set to 1 then all the sessions share the same connection; all requests for a SessionFactory with the same connect string and database will share a single SessionFactory. A setting of 0 disables pooling; each request for a SessionFactory will receive its own unique SessionFactory.
- **com.mysql.clusterj.max.transactions**: Maximum number transactions that can be simultaneously executed using the clusterj client. The maximum support transactions are 1024.
- **io.hops.metadata.ndb.mysqlserver.host**: Address of MySQL server. For higher performance we use MySQL Server to perform a aggregate queries on the file system metadata.
- **io.hops.metadata.ndb.mysqlserver.port**: If not specified then default value of 3306 will be used.
- **io.hops.metadata.ndb.mysqlserver.username**: A valid user name to access MySQL Server.
- **io.hops.metadata.ndb.mysqlserver.password**: MySQL Server user password
- **io.hops.metadata.ndb.mysqlserver.connection pool size**: Number of NDB connections used by the MySQL Server. The default is set to 10.
- **Database Sessions Pool**: For performance reasons the data access layer maintains a pools of pre-allocated ClusterJ session objects. Following parameters are used to control the behavior the session pool.
  - **io.hops.session.pool.size**: Defines the size of the session pool. The pool should be at least as big as the number of active transactions in the system. Number of active transactions in the system can be calculated as (dfs.datanode.handler.count + dfs.namenode.handler.count + dfs.namenode.subtree-executor-limit).
  - **io.hops.session.reuse.count**: Session is used N times and then it is garbage collected. Note: Due to improved memory management in ClusterJ >= 7.4.7, N can be set to higher values i.e. Integer.MAX_VALUE for latest ClusterJ libraries.

Loading a DAL Driver

In order to load a DAL driver following configuration parameters are added to hdfs-site.xml file.

- **dfs.storage.driver.jarFile**: path of driver jar file if the driver’s jar file is not included in the class path.
- **dfs.storage.driver.class**: main class that initializes the driver.
- **dfs.storage.driver.configfile**: path to a file that contains configuration parameters for the driver jar file. The path is supplied to the **dfs.storage.driver.class** as an argument during initialization. See hops ndb driver configuration parameters (page 117).
6.2.11 Erasure Coding

The erasure coding API is flexibly configurable and hence comes with some new configuration options that are shown here. All configuration options can be set by creating an `erasure-coding-site.xml` in the Hops configuration folder. Note that Hops comes with reasonable default values for all of these values. However, erasure coding needs to be enabled manually.

- **dfs.erasure_coding.enabled**: (true/false) Enable/Disable erasure coding.
- **dfs.erasure_coding.codecs.json**: List of available erasure coding codecs available. This value is a json field i.e.

```xml
<value>
  [
    {
      "id" : "xor",
      "parity_dir" : "/raid",
      "stripe_length" : 10,
      "parity_length" : 1,
      "priority" : 100,
      "erasure_code" : "io.hops.erasure_coding.XORCode",
      "description" : "XOR code"
    },
    {
      "id" : "rs",
      "parity_dir" : "/raidrs",
      "stripe_length" : 10,
      "parity_length" : 4,
      "priority" : 300,
      "erasure_code" : "io.hops.erasure_coding.ReedSolomonCode",
      "description" : "ReedSolomonCode code"
    },
    {
      "id" : "src",
      "parity_dir" : "/raidsrc",
      "stripe_length" : 10,
      "parity_length" : 6,
      "parity_length_src" : 2,
      "priority" : 200,
      "description" : "SimpleRegeneratingCode code"
    }
  ]
</value>
```

- **dfs.erasure_coding.parity_folder**: The HDFS folder to store parity information in. Default value is /parity
- **dfs.erasure_coding.recheck_interval**: How frequently should the system schedule encoding or re-pairs and check their state. Default value is 300000 ms.
- **dfs.erasure_coding.repair_delay**: How long should the system wait before scheduling a repair. Default is 1800000 ms.
• **dfs.erasure_coding.parity_repair_delay**: How long should the system wait before scheduling a parity repair. Default is 1800000 ms.

• **dfs.erasure_coding.active_encoding_limit**: Maximum number of active encoding jobs. Default is 10.

• **dfs.erasure_coding.active_repair_limit**: Maximum number of active repair jobs. Default is 10.

• **dfs.erasure_coding.active_parity_repair_limit**: Maximum number of active parity repair jobs. Default is 10.

• **dfs.erasure_coding.deletion_limit**: Delete operations to be handle during one round. Default is 100.

• **dfs.erasure_coding.encoding_manager**: Implementation of the EncodingManager to be used. Default is `io.hops.erasure_coding.MapReduceEncodingManager`.

• **dfs.erasure_coding.block_repair_manager**: Implementation of the repair manager to be used. Default is `io.hops.erasure_coding.MapReduceBlockRepairManager`.

This section contains new/modified configuration parameters for HopsFS. All the configuration parameters are defined in `hdfs-site.xml` and `core-site.xml` files.

### 6.3 Hops-YARN Configuration

#### 6.3.1 Configuring Hops YARN fail-over and Distribution

The main fail-over mechanism of Hops YARN is identical to the Apache Hadoop YARN fail-over mechanism and uses the same configuration as well. As a result, we invite you to have a look at the Apache Hadoop YARN configuration in order to set up your Hops YARN fail-over configuration. Our modifications of the fail-over mechanism result in few new features and a limitation in the state store that can be used to store the cluster state:

**New fail-over Features:**

- **yarn.resourcemanager.groupMembership.address**: The address of the group membership service. The group membership service is used by the clients and NodeManagers to obtain the list of alive ResourcesManagers.

- **yarn.resourcemanager.group_membership.client.thread-count**: The number of threads used by the group membership service that serve http requests. By default, it is set to 1.

**Proxy provider**

- **yarn.client.failover-proxy-provider**: Two new proxy providers have been added to the existing ConfiguredRMFailoverProxyProvider
  
  - **ConfiguredLeaderFailoverHAProxyProvider**: this proxy provider has the same goal as the ConfiguredRMFailoverProxyProvider (connecting to the leading ResourceManager) but it uses the groupMembershipService, where the ConfiguredRMFailoverProxyProvider goes through all
the ResourceManagers present in the configuration file to find the leader. This allows the Config-
uredLeaderFailoverHAProxyProvider to be faster and to find the leader even if it is not present
in the configuration file.

– **ConfiguredLeastLoadedRMFailoverHAProxyProvider**: this proxy provider establishes a
connection with the ResourceTracker that has the lowest current load (least loaded). This proxy
provider is to be used in distributed mode in order to balance the load coming from NodeMan-
agers across ResourceTrackers.

**Supported State Store:**

Since Hops YARN relies on the distributed database for the group membership management and its dis-
tributed model it was chosen not to support the existing state store to focus only on the database state store.

- **yarn.resourcemanager.store.class**: Should be set to org.apache.hadoop.yarn.server.
  resourcemanager.recovery.DBRMStateStore

**Distributed Mode**

Hops YARN distributed mode can be enabled by setting the following flags to `true`:

- **yarn.client.failover-distributed**: Set to `true` to indicate that the system should work in distributed
  mode.

In order to run efficiently, the distributed mode relies on the database streaming mechanism. The port on
which the database is listening for streaming requests can be configured with the following configuration:

- **hops.yarn.resourcemanager.event-streaming.db.port**: the port on which the database is listening
  for streaming request, by default 1186

**6.3.2 Configuring Hops YARN quota system**

In order to set a price for the different cluster resources and to charge applications in a fine-grained way, but
without causing too much load on the resource manager and the database, several configuration parameters
need to be tuned:

**Basic parameters:**

These is the minimum configuration required in order to set up the quota system:

- **yarn.resourcemanager.quota.containers.log.period**: Time, in milliseconds, between two evalua-
tions of the containers status. This is the frequency at which the quota system is evaluating application
resources utilization, we also call it a tick. The lower this value the finer the cluster utilization
evaluation (more precision on exactly when a container changed state), but the higher the load on the
resource manager and database. Default, 1000.

- **yarn.resourcemanager.quota.minTicksCharge**: The minimum number of ticks for which a con-
tainer will be charged. If a container runs for less than this number of ticks, it will still be charged for
this number of ticks. This allows the system to charge for the time to allocate and start the container, during which the container is not running, but is still using resources. This also avoids short running containers polluting the scheduling system by popping up and down unnecessarily. Default, 10.

- `yarn.resourcemanager.quota.minimum.charged.mb`: Amount of memory, in MB, used as a unit of memory in the computation of quota utilization. If this is set to 1024 and a container runs for one tick using 2048MB of memory, the container will be charged for two resource utilizations.

- `yarn.resourcemanager.quota.price.base.general`: The base price, in number of credits, for using a unit of resources during one tick for non-specialized resources (CPU, memory). Default, 1.

- `yarn.resourcemanager.quota.price.base.gpu`: The base price, in number of credits, for using a unit of resources during one tick for GPUs resources. Default, 2.

**Variable pricing:**

On top of charging for quota utilization, Hops YARN provides a variable pricing system to incentivize people to not use the cluster at pick time:

- `yarn.resourcemanager.quota.multiplicator.threshold.general`: The minimum cluster utilization, as a float between 0 and 1, after which the price of resources start to be increased proportionally to the cluster utilization, for non-specialized resources (CPU, memory). Default 0.2.

- `yarn.resourcemanager.quota.multiplicator.threshold.gpu`: The minimum cluster utilization, as a float between 0 and 1, after which the price of resources starts to increase proportionally to the cluster utilization, for GPUs. Default, 0.2.

- `yarn.resourcemanager.quota.multiplicator.increment.general`: The multiplicator by which to multiply the resource utilization in order to compute the new resource price for non-specialized resources (CPU, memory). If this is set to 2, the price for resources will increase by 2% when cluster utilization increases by 1%. Default, 1.

- `yarn.resourcemanager.quota.multiplicator.increment.gpu`: The multiplicator by which to multiply the resource utilization in order to compute the new resource price for GPUs. If this is set to 2, the price for resources will increase by 2% when the cluster utilization increases by 1%. Default, 2.

- `yarn.resourcemanager.quota.multiplicator.interval`: Time, in milliseconds between two evaluations of the resource pricing. Default, 1000.

**Charging as it runs:**

Charging application when they finish running is not compatible with long-running streaming applications. The following options allow you to charge the application for the resources they have used so far, while they are running.

- `yarn.resourcemanager.quota.containers.log.checkpoints.enabled`: Should the checkpointing system be enabled? If the checkpointing system is disabled, applications will only be charged for the resources they used when the containers finish running. This can cause problems for long running applications: impossible for the user to know how much quota they have used, resource price not adapting with time. Default, true.
• **yarn.resourcemanager.quota.containers.log.checkpoints.period**: Period, in number of ticks, at which to establish a checkpoint for a container. If the period is set to 6 and a container is running for a long time, the user will be charged for this container every 6 ticks as long as the container is running. Default, 6.

• **yarn.resourcemanager.quota.multiplicator.fixed.period**: Period, in number of checkpoints, during which the price of resources is fixed for a given application. If this value is set to 10 and an application starts while the cluster resource price is 1, then the application will pay a price of one during its first 10 checkpoints. Once the 10th checkpoint is reached, the price will be set to the current cluster resources pricing and this new price will be applied for the 10 next checkpoints. Default, 10.

**Database load:**

The following option allows you to tune the load generated by the quota system on the database:

• **yarn.resourcemanager.quota.batch.time**: Time, in milliseconds, during which quota utilization updates are batched before being committed to the database. Reducing this value allows the user to be informed faster on their quota utilization and reduce the amount of data needed to be recovered in case of ResourceManager crash. On the other hand, reducing this value increases the load on the database. Default, 500.

• **yarn.resourcemanager.quota.batch.size**: Maximum size, in number of updates, of the batch of quota utilization updates. If this number of updates is reached before the batch time is reached, the update will be persisted to the database immediately. Increasing this value decreases the load on the database but increases the computation time for each batch. Default, 100.

### 6.3.3 Configuring Hops YARN GPU support

In order for a NodeManager to offers GPUs to be scheduled by the ResourceManager certain configuration properties need to be set.

**Configuration parameters:**

• **yarn.nodemanager.resource.gpus.enabled**: Boolean `true` or `false` depending on if the NodeManager should offer GPUs from the machine to the scheduler Default, `false`.

• **yarn.nodemanager.resource.gpus**: Number of GPUs to offer to the scheduler Default, 0.

• **yarn.nodemanager.gpu.management-impl**: Implementation class to query system for GPUs. For NVIDIA GPUs use `io.hops.management.nvidia.NvidiaManagementLibrary` for AMD use `io.hops.management.amd.AMDManagementLibrary`

• **yarn.scheduler.capacity.resource-calculator**: Must be set to `org.apache.hadoop.yarn.util.resource.DominantResourceCalculatorGPU`

• **yarn.nodemanager.container-executor.class**: Must be set to `org.apache.hadoop.yarn.server.nodemanager.LinuxContainerExecutor`
• **yarn.nodemanager.linux-container-executor.resources-handler.class:** Must be set to `org.apache.hadoop.yarn.server.nodemanager.util.CgroupsLCEResourcesHandlerGPU`

When the number of GPUs is changed the Cgroup hierarchy needs to be deleted manually. Assuming the Cgroup path is `/sys/fs/cgroup` then issue the following command `sudo rmdir /sys/fs/cgroup/devices/hops-yarn`

Hops-YARN configuration is very similar to the Apache Hadoop YARN configuration. A few additional configuration parameters are needed to configure the new services provided by Hops-YARN. This section presents the new/modified configuration parameters for Hops-YARN. All the new configuration parameters should be entered in `yarn-site.xml`.

### 6.4 Hops SSL/TLS configuration

Hops is the only Hadoop distribution that supports SSL/TLS encryption at the RPC layer. This provides encryption and authentication in all intra-Hadoop communication links. In Hopsworks a registered user holds a x509 certificate for every Project he/she has created. The CN field in the certificate contains his/her project specific username and that gives us the possibility to provide application level authorization at the RPC server.

In order to enable this feature a set of configuration parameters should be set. A description of the required properties is outlined below.

In Hops the file name pattern for the **keystore** should be following:

- For the server keystore: `HOSTNAME__kstore.jks`, where HOSTNAME is the hostname of the machine
- For the client keystore: `USERNAME__kstore.jks` where USERNAME in Hopsworks is the project specific username in the form of `PROJECTNAME__USERNAME`

The file name pattern for the **truststore** should be following:

- For the server keystore: `HOSTNAME__tstore.jks`, where HOSTNAME is the hostname of the machine
- For the client keystore: `USERNAME__tstore.jks` where USERNAME in Hopsworks is the project specific username in the form of `PROJECTNAME__USERNAME`

First we will go through the properties in `$HADOOP_HOME/etc/hadoop/ssl-server.xml` configuration file. This file contains information regarding the cryptographic material needed by the entities which run an RPC server. The properties are superset of Apache Hadoop, a brief description can be found [here](https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/EncryptedShuffle.html#ssl-server.xml_Shuffle_server_Configuration:).

---

<table>
<thead>
<tr>
<th>property name</th>
<th>description</th>
<th>default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssl.server.truststore.location</td>
<td>Location of the truststore in local filesystem</td>
<td></td>
</tr>
<tr>
<td>ssl.server.truststore.password</td>
<td>Password of the truststore</td>
<td></td>
</tr>
<tr>
<td>ssl.server.truststore.type</td>
<td>Type of the truststore</td>
<td>jks</td>
</tr>
<tr>
<td>ssl.server.truststore.reload.interval</td>
<td>Reload interval of the truststore in ms</td>
<td>10000</td>
</tr>
<tr>
<td>ssl.server.keystore.location</td>
<td>Location of the keystore in local filesystem</td>
<td></td>
</tr>
<tr>
<td>ssl.server.keystore.password</td>
<td>Password of the keystore</td>
<td></td>
</tr>
<tr>
<td>ssl.server.keystore.keypassword</td>
<td>Password of the private key in the keystore</td>
<td></td>
</tr>
<tr>
<td>ssl.server.keystore.type</td>
<td>Type of the keystore</td>
<td>jks</td>
</tr>
<tr>
<td>ssl.server.keystore.reload.interval</td>
<td>Reload interval of the keystore</td>
<td>10000</td>
</tr>
<tr>
<td>ssl.server.keystore.reload.timeunit</td>
<td>Keystore reload interval TimeUnit (Java TimeUnit)</td>
<td>MILLISECONDS</td>
</tr>
</tbody>
</table>

Example of `ssl-server.xml` configuration file:

```xml
<property>
    <name>ssl.server.truststore.location</name>
    <value>/srv/hops/kagent-certs/keystores/hopsworks0__tstore.jks</value>
    <description>Truststore to be used by NN and DN. Must be specified.</description>
</property>

<property>
    <name>ssl.server.truststore.password</name>
    <value>password</value>
    <description>Optional. Default value is "".</description>
</property>

<property>
    <name>ssl.server.truststore.type</name>
    <value>jks</value>
    <description>Optional. The keystore file format, default value is "jks".</description>
</property>

<property>
    <name>ssl.server.truststore.reload.interval</name>
    <value>10000</value>
    <description>Truststore reload check interval, in milliseconds. Default value is 10000 (10 seconds).</description>
</property>

<property>
    <name>ssl.server.keystore.location</name>
    <value>/srv/hops/kagent-certs/keystores/hopsworks0__kstore.jks</value>
    <description>Keystore to be used by NN and DN. Must be specified.</description>
</property>
```

(continues on next page)
Next is a list of properties required in `$HADOOP_HOME/etc/hadoop/core-site.xml` configuration file. These properties enable SSL/TLS in the RPC server.
<table>
<thead>
<tr>
<th>property name</th>
<th>description</th>
<th>sample value</th>
<th>default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipc.server.ssl.enabled</td>
<td>Switch between SSL/TLS support for RPC server</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>ipc.server.threadpool.size</td>
<td>Number of threads utilized to read RPC requests</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>hadoop.rpc.socket.factory</td>
<td>Default Hadoop socket factory</td>
<td>org.apache.hadoop.net.HopsSSLSocketFactory</td>
<td>org.apache.hadoop.net.StandardSocketFactory</td>
</tr>
<tr>
<td>hadoop.ssl.hostname.verifier</td>
<td>Verifier used for the FQDN field at the presented x509 certificate</td>
<td>ALLOW_ALL</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>hadoop.ssl.enabled.protocols</td>
<td>Enabled SSL protocols for the SSL engine</td>
<td>TLSv1.2, TLSv1.1, TLSv1, SSLv3</td>
<td>TLSv1.2, TLSv1.1, TLSv1, SSLv3</td>
</tr>
<tr>
<td>hops.service.certificates.directory</td>
<td>Contains keystore and truststore used by the service/server</td>
<td>/srv/hops/kagent-certs/keystores</td>
<td>/srv/hops/kagent-certs/keystores</td>
</tr>
<tr>
<td>client.materialize.directory</td>
<td>Where Hopsworks has already materialized the crypto material from the database for a specific user</td>
<td>/srv/hops/certs-dir/transient</td>
<td>/srv/hops/domains/domain1/kafkacerts</td>
</tr>
<tr>
<td>client.rpc.ssl.enabled</td>
<td>SSL protocol used by the client</td>
<td>TLSv1.2</td>
<td>TLSv1</td>
</tr>
</tbody>
</table>

Example of core-site.xml configuration file:

```xml
<property>
    <name>ipc.server.read.threadpool.size</name>
    <value>3</value>
</property>

<property>
    <name>ipc.server.ssl.enabled</name>
    <value>true</value>
</property>

<property>
    <name>hadoop.ssl.hostname.verifier</name>
    <value>ALLOW_ALL</value>
</property>

<property>
    <name>hadoop.ssl.enabled.protocols</name>
    <value>TLSv1.2, TLSv1.1, TLSv1, SSLv3</value>
</property>

<property>
    <name>hops.service.certificates.directory</name>
    <value>/srv/hops/kagent-certs/keystores</value>
</property>
```

(continues on next page)
In case where the ResourceManager is deployed in High-Availability mode some extra configuration properties should be set in $HADOOP_HOME/etc/hadoop/yarn-site.xml in addition to the standard RM HA properties.

<table>
<thead>
<tr>
<th>property name</th>
<th>description</th>
<th>sample value</th>
<th>default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>yarn.resourcemanager.ha.enabled</td>
<td>Standard YARN property to enable RM HA</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>yarn.resourcemanager.ha.id</td>
<td>Standard YARN property to uniquely identify an RM</td>
<td>rm0</td>
<td></td>
</tr>
<tr>
<td>yarn.resourcemanager.ha.rm-ids</td>
<td>Standard YARN property that lists the IDs of RMs</td>
<td>rm0,rm1</td>
<td></td>
</tr>
<tr>
<td>yarn.resourcemanager.ha.cert.loc.address.rm0</td>
<td>ipaddress:port for the CertificateLocalizationService running on each of the RMs</td>
<td>10.0.2.15:8012</td>
<td></td>
</tr>
</tbody>
</table>

Follows a sample of yarn-site.xml when RM HA is enabled for two RMs:

```xml
<property>
  <name>yarn.resourcemanager.ha.enabled</name>
  <value>true</value>
</property>

<property>
  <name>yarn.resourcemanager.ha.id</name>
  <value>rm0</value>
</property>

<property>
  <name>yarn.resourcemanager.ha.rm-ids</name>
  <value>rm0,rm1</value>
</property>

<property>
  <name>yarn.resourcemanager.ha.cert.loc.address.rm0</name>
  <value>10.0.2.15:8012</value>
</property>
```

(continues on next page)
6.4.1 Certificate Revocation List validation

Since version 2.8.2.4 of Hops, Certificate Revocation List (CRL) validation is supported for the RPC servers. The CRL should be placed in publicly available location and it will be fetched periodically by Hops services - ResourceManagers, NameNodes, NodeManagers and DataNodes. Client’s certificate is validated against this CRL and if the certificate has been revoked, the connection is dropped. The configuration properties for CRL validation should be in core-site.xml. To enable CRL validation, RPC TLS should also be enabled.

<table>
<thead>
<tr>
<th>property name</th>
<th>description</th>
<th>default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>hops.crl.validation.enabled</td>
<td>Enable CRL validation</td>
<td>false</td>
</tr>
<tr>
<td>hops.crl.input.uri</td>
<td>URI where CRL will be fetched from</td>
<td></td>
</tr>
<tr>
<td>hops.crl.output.file</td>
<td>File in the local filesystem where the CRL will be written to</td>
<td></td>
</tr>
<tr>
<td>hops.crl.fetcher.interval</td>
<td>CRL fetch interval (ns,us,ms,s,m,h,d)</td>
<td>720m</td>
</tr>
<tr>
<td>hops.crl.fetcher.class</td>
<td>Fetcher class which will fetch the CRL periodically and write it to local filesystem</td>
<td>org.apache.hadoop.security.ssl.RemoteCRLFetcher</td>
</tr>
</tbody>
</table>

The CRL section in core-site.xml should look like this:

```xml
<property>
  <name>hops.crl.validation.enabled</name>
  <value>true</value>
</property>

<property>
  <name>hops.crl.input.uri</name>
  <value>http://my_host/hops.crl.pem</value>
</property>

<property>
  <name>hops.crl.output.file</name>
  <value>hadoop_home/tmp/hops.crl.pem</value>
</property>

<property>
  <name>hops.crl.fetcher.interval</name>
  <value>2d</value>
</property>

<property>
  <name>hops.crl.fetcher.class</name>
</property>
```
6.4.2 Enabling RPC/IPC TLS with Karamel/Chef

If you are using Karamel\textsuperscript{62}, then your cluster definition should look like the following in order to enable RPC TLS and CRL validation. CRL input URI would be HOPSWORKS_ENDPOINT/intermediate.crl.pem, output file $HADOOP_TMP_DIR/hops_crl.pem and fetcher interval 1d. The DevRemoteCRLFetcher class is the same as RemoteCRLFetcher but trusts any web server certificate, usually in a development environment the certificate will be self-signed:

\begin{verbatim}
hops:
  tls:
    enabled: true
    crl_enabled: true
    crl_fetcher_class: org.apache.hadoop.security.ssl.DevRemoteCRLFetcher
\end{verbatim}

Hopworks has an administrator application that allows you, the administrator, to perform management actions, monitor Hopsworks and Hops, and control Hopsworks and Hops.

\textsuperscript{62}http://www.karamel.io/
7.1 Microservices

Contents

- Adding a new Microservice (page 131)
  - Front-end (page 131)
  - Chef/Settings updates (page 132)
  - Java EE updates (page 132)
  - Swagger UI (page 133)

Hopworks makes use of the microservices model to incorporate new services. Each new microservice needs to adhere to the multi-tenant model of Hopworks based on the concepts of Projects and Users.

7.1.1 Adding a new Microservice

There are certain steps required for a new microservice to be integrated into Hopworks. The following sections describe them on a per application layer basis.

Front-end

Angular-JS

1. Add a new html file in /yo/app/views/Microservice.html
2. Add a new angularJS controller in /yo/app/scripts/controllers/MicroserviceCtrl.js

3. Add a new angularJS service in /yo/app/scripts/services/MicroserviceService.js

4. Add the new service and controller to index.html /yo/app/index.html

5. Add the service as a route to app.js to route to its URL, e.g.:
   a. when('/register', { ....} or
   b. when('/project/:projectID/datasets', { ...}

6. Add the new microservice to the array of services defined in projectCreator.js and project.js, as well as the /yo/app/navProject.html navigation bar, and add methods to project.js for showMicroservice() and goToMicroservice().

7. Add the route to the microservice to /yo/app/app.js - when('/project/:projectID/tensorflow', {...}

**Chef/Settings updates**

8. Add any new tables used by the microservice to hopsworks-chef/templates/default/tables.sql.erb

9. Add the IP for the new service to the variables table, so that it can be ‘discovered’ by Hopsworks using the Settings.java bean. In the Hopsworks Java web application, you will need to update the Settings.java bean to include the IP addr/port for the new microservice:
   a. hopsworks-chef/templates/default/rows.sql.erb
   b. hopsworks/src/main/java/.../Settings.java

**Java EE updates**

10. Create a new REST service for Microservice.java. It should be a subresource of ProjectService.java. Base it on an example, like KafkaService.java

11. Create a new facade MicroserviceFacade.java in the new package io.hops.microservice. The REST service should be a subresource of ProjectService.

12. Add the new REST service to se.kth.rest.application.config.ApplicationConfig.java.

13. Create new Entity beans for the tables defined in step 6. Use the Netbeans wizard to do this - run vagrant and forward port 3306 on mysql to a port you can access. Then connect to the database ‘hopsworks’ with username/password ‘kthfs’/’kthfs’. Make sure to include the full entity name (catalog name) in the definition - ‘hopsworks’.

14. Add the beans to the persistence.xml file in /src/resources/../persistence.xml and the microservice should be added to ProjectServiceEnum.java.

**Connect up the components:**
a. Javascript - HTML -> Controller -> Service -> ............

b. Java -> REST-> Facade -> EntityBean

**Swagger UI**

To visualize and interact with the REST API's without having to implement the front-end

15. Annotate the REST service with `@Api(value = "Name of the rest endpoint", description = "description of the rest endpoint")`

16. Build hopsworks-web with `mvn clean install -Pswagger-ui` and deploy

17. Goto hopsworks/swagger-ui

### 7.2 Platform Testing

#### 7.2.1 User Guide

Every pull request to Hopsworks or the cookbooks, needs to be tested before being merged into the master branch. *hops-testing* performs end-to-end testing on the platform by executing the following steps:

- Spin up a set of new VMs with differing configuration setup (default Ubuntu/CentOS)
- Clone and build Hopsworks
- Deploy the platform
- Run the tests Hopsworks tests

A failure in building/deployment or testing will fail the entire Jenkins build.

Triggering the tests is trivial. Besides having a fork of the Hopsworks repo and of all the cookbook repos, developers need also have a fork of the *hops-testing* repo

Once developers are satisfied with their changes, they should open all the PRs necessary to merge their work to the upstream repository. In case of cookbook PRs, the links in the each Berksfile myst *always* point to master.

Developers must open also a new PR on the *hops-testing* repo. This last PR has to modify the *test_manifesto* file which contains the list of repos the change involves.

The format for this file should be:

```
organization/reponame/branch
organization/reponame/branch
```

Where *organization* is the developer’s GitHub username, *reponame* is the repository name and *branch* is the branch which contains the changes. The *test_manifesto* makes sure that all the PRs related to a single change are tested together.

63 [https://github.com/hopshadoop/hops-testing](https://github.com/hopshadoop/hops-testing)
The PR on the hops-testing repo will be picked up by Jenkins which then starts the testing process. Please add a comment to each PR in the other repos with the link of the hops-testing PR, so it will be easier for maintainers to check your PRs and merge them.

### 7.2.2 Developer Guide

This part of the guide is meant to be for those who will have to touch/maintain the testing infrastructure.

The HopsworksJenkins Github user has a fork of Hopsworks and each cookbook. All the interactions with GitHub done in the pipeline are done as this user using the git protocol and SSH key for authentication. All the steps of the pipeline are run in the same node and the SSH key for the HopsworksJenkins user should be available on that node.

Currently all the VMs are run on the same Jenkins node (same machine, same user), this is achieved by giving the VMs different names (e.g. ubuntu and centos). To avoid both machines to be killed when one of the two finishes, we need to start the VBoxSVC process as daemon:

```
/usr/lib/virtualbox/VBoxSVC --pidfile VBoxSVC.pid &
```

Building Hopsworks build and running tests run are both done inside the VM for several reasons.

1. Building Hopsworks outside the VM and copying the artifacts is slow.
2. Installing Ruby and JS dependencies can be messy and it is safer to do that in a VM environment than on the actual machine running the testing pipeline.
3. It is to expand the set of configurations being tested and requires minimal changes to the pipeline code (add new parallel stage and new template) and/or using multiple nodes for testing various configurations.

The workflow (pipeline) of a Jenkins build is defined in the Jenkinsfile and is composed of the following steps:

1. Jenkins is notified by GitHub that there is a new PR to test.
2. Jenkins runs a clean up, kills the VMs of the previous run and removes from all the repos the branch test_platform (both locally and from GitHub). The clean up is done at this stage, and not at the end of the previous testing iteration, to allow developers to look into the VMs for Karamel output in case of a failure in the deployment.
3. Jenkins updates all the local clones pulling from the master of the Hopshadoop organization.
4. Jenkins creates for each repo a test_platform branch, merges the changes in the repos specified in the test_manifesto, rewrites all the links in the Berkshelf files to point to the forks of the Hopsworksjenkins user (test_platform branches) and finally, it pushes everything to GitHub.
5. Jenkins starts the VMs using the Vagrantfiles included in the templates directory. The VMs have two shared folder: the .m2 directory, to cache maven artifacts across Jenkins builds and speed up the testing, and the test-report directory, to get the results of the test out of the VMs and make them available to Jenkins for publishing them.
6. Vagrant starts chef which as first step will clone hopworksjenkins/hopworks repo, compile it and put the artifacts in the chef_file_cache dir to be picked up during the deployments.
7. Chef starts Karamel which will use `hopworksjenkins/hopworks-chef` to deploy everything

8. Chef runs tests.

9. Tests results are published.

7.3 Extending HopsFS INode metadata

Contents

- **Example use case** (page 135)
- **Adding a table to the schema** (page 135)
- **Defining the Entity Class** (page 136)
- **Defining the DataAccess interface** (page 136)
- **Implementing the DataAccess interface** (page 137)
- **Implementing the EntityContext** (page 139)
- **Using custom locks** (page 142)

For the implementation of new features, it is often necessary to modify the `hdfs_inodes` table or add new tables in order to store extended metadata. With Hops-HDFS, this can be simply achieved by adding a new table with a foreign key that refers to `hdfs_inodes`. Adding new tables has the benefit that the original data structures do not need to be modified and old code paths not requiring the additional metadata are not burdened with additional reading costs. This guide gives a walkthrough on how to add additional INode-related metadata.

7.3.1 Example use case

Let’s assume we would like to store per user access times for each INode. To do this, we need to store the id of the inode, the name of the user and the timestamp representing the most recent access.

7.3.2 Adding a table to the schema

First, we need to add a new table storing the metadata to our schema. Therefore we’ll go to the `hops-metadata-dal-impl-ndb` project and add the following to the `schema/schema.sql` file.

```sql
CREATE TABLE `hdfs_access_time_log` (  
  `inode_id` int(11) NOT NULL,  
  `user` varchar(32) NOT NULL,  
  `access_time` bigint(20) NOT NULL,  
  PRIMARY KEY (`inode_id`, `user`)  
) ENGINE=ndbcluster DEFAULT CHARSET=latin1$$
```
Additionally we will make the table and column names available to the Java code by adding the following to the `io.hops.metadata.hdfs.TablesDef` class in `hops-metadata-dal`.

```java
class AccessTimeLogTableDef {
    public static final String TABLE_NAME = "hdfs_access_time_log";
    public static final String INODE_ID = "inode_id";
    public static final String USER = "user";
    public static final String ACCESS_TIME = "access_time";
}
```

**Note** Don’t forget to update your database with the new schema.

### 7.3.3 Defining the Entity Class

Having defined the database table, we will need to defining an entity class representing our database entries in the java code. We will do this by adding the following `AccessTimeLogEntry` class in the `hops-meta-data-dal` project.

```java
package io.hops.metadatad.hdfs.entity;

public class AccessTimeLogEntry {
    private final int inodeId;
    private final String user;
    private final long accessTime;

    public AccessTimeLogEntry(int inodeId, String user, long accessTime) {
        this.inodeId = inodeId;
        this.user = user;
        this.accessTime = accessTime;
    }

    public int getInodeId() {
        return inodeId;
    }

    public String getUser() {
        return user;
    }

    public long getAccessTime() {
        return accessTime;
    }
}
```

### 7.3.4 Defining the DataAccess interface

We will need a way for interacting with our new entity in the database. The preferred way of doing this in Hops is defining a `DataAccess` interface to be implemented by a database implementation. Let’s define
define the following interface in the \textit{hops-metadata-dal} project. For now, we will only require functionality to add and modify log entries and to read individual entries for a given INode and user.

```java
package io.hops.metadata.hdfs.dal;

public interface AccessTimeLogDataAccess<T> extends EntityDataAccess {
    void prepare(Collection<T> modified, Collection<T> removed) throws StorageException;
    T find(int inodeId, String user) throws StorageException;
}
```

### 7.3.5 Implementing the DataAccess interface

Having defined the interface, we will need to implement it using ndb to read and persist our data. Therefor, we will add a clusterj implementation of our interface to the \textit{hops-metadata-dal-impl-ndb} project.

```java
package io.hops.metadata.ndb.dalimpl.hdfs;

public class AccessTimeLogClusterj implements TablesDef.AccessTimeLogTableDef, AccessTimeLogDataAccess<AccessTimeLogEntry> {
    private ClusterjConnector connector = ClusterjConnector.getInstance();

    @PersistenceCapable(table = TABLE_NAME)
    public interface AccessTimeLogEntryDto {
        @PrimaryKey @Column(name = INODE_ID)
        int getInodeId();

        void setInodeId(int inodeId);

        @PrimaryKey @Column(name = USER)
        String getUser();

        void setUser(String user);

        @Column(name = ACCESS_TIME)
        long getAccessTime();

        void setAccessTime(long accessTime);
    }

    @Override
    public void prepare(Collection<AccessTimeLogEntry> modified, Collection<AccessTimeLogEntry> removed) throws StorageException {
        HopsSession session = connector.obtainSession();
        List<AccessTimeLogEntryDto> changes = new ArrayList<accesstimelogentrydto>();
        List<AccessTimeLogEntryDto> deletions =
    }
```

(continues on next page)
    new ArrayList<accesstimelogentrydto>();
    if (removed != null) {
        for (AccessTimeLogEntry logEntry : removed) {
            Object[] pk = new Object[2];
            pk[0] = logEntry.getInodeId();
            pk[1] = logEntry.getUser();
            InodeDTO persistable =
                session.newInstance(AccessTimeLogEntryDto.class, pk);
            deletions.add(persistable);
        }
    }
    if (modified != null) {
        for (AccessTimeLogEntry logEntry : modified) {
            AccessTimeLogEntryDto persistable =
                createPersistable(logEntry, session);
            changes.add(persistable);
        }
    }
    session.deletePersistentAll(deletions);
    session.savePersistentAll(changes);
}

@Override
public AccessTimeLogEntry find(int inodeId, String user)
    throws StorageException {
    HopsSession session = connector.obtainSession();
    Object[] key = new Object[2];
    key[0] = inodeId;
    key[1] = user;
    AccessTimeLogEntryDto dto = session.find(AccessTimeLogEntryDto.class,
        key);
    AccessTimeLogEntry logEntry = create(dto);
    return logEntry;
}

private AccessTimeLogEntryDto createPersistable(AccessTimeLogEntry logEntry,
                                            HopsSession session) throws StorageException {
    AccessTimeLogEntryDto dto = session.newInstance(AccessTimeLogEntryDto.class);
    dto.setInodeId(logEntry.getInodeId());
    dto.setUser(logEntry.getUser());
    dto.setAccessTime(logEntry.getAccessTime());
    return dto;
}

private AccessTimeLogEntry create(AccessTimeLogEntryDto dto) {
    AccessTimeLogEntry logEntry = new AccessTimeLogEntry(
        dto.getInodeId(),
        dto.getUser(),
        dto.getAccessTime());
    return logEntry;
}
Having defined a concrete implementation of the DataAccess, we need to make it available to the EntityManager by adding it to NdbStorageFactory in the hops-metadata-dal-impl-ndb project. Edit its initDataAccessMap() function by adding the newly defined DataAccess as following.

```java
private void initDataAccessMap() {
    [...] 
    dataAccessMap.put(AccessTimeLogDataAccess.class, new 
                    AccessTimeLogClusterj());
}
```

### 7.3.6 Implementing the EntityContext

Hops-HDFS uses context objects to cache the state of entities during transactions before persisting them in the database during the commit phase. We will need to implement such a context for our new entity in the hops project.

```java
package io.hops.transaction.context;

public class AccessTimeLogContext extends 
    BaseEntityContext<Object, AccessTimeLogEntry> {

    private final 
        AccessTimeLogDataAccess<AccessTimeLogEntry> dataAccess;

    /* Finder to be passed to the EntityManager */
    public enum Finder implements 
        FinderType<AccessTimeLogEntry> {
        ByInodeIdAndUser;
            @Override
        public 
            Class getType() {
            return AccessTimeLogEntry.class;
        }

            @Override
        public Annotation getAnnotated() {
            switch (this) {
            case ByInodeIdAndUser:
                return Annotation.PrimaryKey;
            default:
                throw new 
                    IllegalStateException();
            }
        }
    }

    /* Our entity uses inode id and user as a composite key. 
    * Hence, we need to implement a composite key class. 
    */
    private class Key {
        
    }
```

(continues on next page)
```java
    int inodeId;
    String user;

    public Key(int inodeId, String user) {
        this.inodeId = inodeId;
        this.user = user;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        } else if (o == null || getClass() != o.getClass()) {
            return false;
        }
        Key key = (Key) o;
        if (inodeId != key.inodeId) {
            return false;
        }
        return user.equals(key.user);
    }

    @Override
    public int hashCode() {
        int result = inodeId;
        result = 31 * result + user.hashCode();
        return result;
    }

    @Override
    public String toString() {
        return "Key{inodeId=" + inodeId + ", user=" + user + "}";
    }

    public AccessTimeLogContext(AccessTimeLogDataAccess<AccessTimeLogEntry> dataAccess) {
        this.dataAccess = dataAccess;
    }

    @Override
    Object getKey(AccessTimeLogEntry logEntry) {
        return new Key(logEntry.getInodeId(), logEntry.getUser());
    }
```

(continues on next page)
public void prepare(TransactionLocks tlm) throws TransactionContextException, StorageException {
    Collection<AccessTimeLogEntry> modified =
        new ArrayList<AccessTimeLogEntry>(getModified());
    modified.addAll(getAdded());
    dataAccess.prepare(modified, getRemoved());
}

@Override
public AccessTimeLogEntry find(FinderType<AccessTimeLogEntry> finder, Object... params) throws TransactionContextException, StorageException {
    Finder afinder = (Finder) finder;
    switch (afinder) {
        case ByInodeIdAndUser:
            return findByPrimaryKey(afinder, params);
    }
    throw new UnsupportedOperationException(UNSUPPORTED_FINDER);
}

private AccessTimeLogEntry findByPrimaryKey(Finder finder, Object[] params) throws StorageCallPreventedException, StorageException {
    final int inodeId = (Integer) params[0];
    final String user = (String) params[1];
    Key key = new Key(inodeId, user);
    AccessTimeLogEntry result;
    if (contains(key)) { // Get it from the cache
        result = get(key);
        hit(finder, result, params);
    } else { // Throw an exception
        aboutToAccessStorage(finder, params); // if reading after the reading phase
        result = dataAccess.find(inodeId, user); // Fetch the value
        gotFromDB(key, result); // Put the new value into the cache
        miss(finder, result, params);
    }
    return result;
}

Having defined an EntityContext, we need to make it available through the EntityManager by adding it to the HdfsStorageFactory in the hops project by modifying it as follows.

private static ContextInitializer getContextInitializer() {
    return new ContextInitializer() {
        @Override
        public Map<Class, EntityContext> createEntityContexts() {
            Map<Class, EntityContext> entityContexts =
                new HashMap<Class, EntityContext>();
            [...]
        }
    };
}

7.3. Extending HopsFS INode metadata
7.3.7 Using custom locks

Your metadata extension relies on the inode object to be correctly locked in order to prevent concurrent modifications. However, it might be necessary to modify attributes without locking the INode in advance. In that case, one needs to add a new lock type. A good place to get started with this is looking at the Lock, HdfsTransactionLocks, LockFactory and HdfsTransactionalLockAcquirer classes in the hops project.

7.4 Erasure Coding API Access

Contents

- Java API (page 142)
- Creation of Encoded Files (page 143)
- Encoding of Existing Files (page 143)
- Reverting To Replication Only (page 143)
- Deletion Of Encoded Files (page 144)

HopsFS provides erasure coding functionality in order to decrease storage costs without the loss of high-availability. Hops offers a powerful, on a per file basis configurable, erasure coding API. Codes can be freely configured and different configurations can be applied to different files. Given that Hops monitors your erasure-coded files directly in the NameNode, maximum control over encoded files is guaranteed. This page explains how to configure and use the erasure coding functionality of Hops. Apache HDFS stores 3 copies of your data to provide high-availability. So, 1 petabyte of data actually requires 3 petabytes of storage. For many organizations, this results in enormous storage costs. HopsFS also supports erasure coding to reduce the storage required by by 44% compared to HDFS, while still providing high-availability for your data.

7.4.1 Java API

The erasure coding API is exposed to the client through the DistributedFileSystem class. The following sections give examples on how to use its functionality. Note that the following examples rely on erasure coding being properly configured. Information about how to do this can be found in Erasure Coding (page 119).
7.4.2 Creation of Encoded Files

The erasure coding API offers the ability to request the encoding of a file while being created. Doing so has the benefit that file blocks can initially be placed in a way that meets placements constraints for erasure-coded files without needing to rewrite them during the encoding process. The actual encoding process will take place asynchronously on the cluster.

```java
Configuration conf = new Configuration();
DistributedFileSystem dfs = (DistributedFileSystem) FileSystem.get(conf);
// Use the configured "src" codec and reduce
// the replication to 1 after successful encoding
EncodingPolicy policy = new EncodingPolicy("src" /* Codec id as configured */, (short) 1);
// Create the file with the given policy and
// write it with an initial replication of 2
FSDataOutputStream out = dfs.create(path, (short) 2, policy);
// Write some data to the stream and close it as usual
out.close();
// Done. The encoding will be executed asynchronously
// as soon as resources are available.
```

Multiple versions of the create function complementing the original versions with erasure coding functionality exist. For more information please refer to the class documentation.

7.4.3 Encoding of Existing Files

The erasure coding API offers the ability to request the encoding for existing files. A replication factor to be applied after successfully encoding the file can be supplied as well as the desired codec. The actual encoding process will take place asynchronously on the cluster.

```java
Configuration conf = new Configuration();
DistributedFileSystem dfs = (DistributedFileSystem) FileSystem.get(conf);
String path = "/testFile";
// Use the configured "src" codec and reduce the replication to 1
// after successful encoding
EncodingPolicy policy = new EncodingPolicy("src" /* Codec id as configured */, (short) 1);
// Request the asynchronous encoding of the file
dfs.encodeFile(path, policy);
// Done. The encoding will be executed asynchronously
// as soon as resources are available.
```

7.4.4 Reverting To Replication Only

The erasure coding API allows to revert the encoding and to default to replication only. A replication factor can be supplied and is guaranteed to be reached before deleting any parity information.

```java
Configuration conf = new Configuration();
DistributedFileSystem dfs = (DistributedFileSystem) FileSystem.get(conf);
```

(continues on next page)
// The path to an encoded file
String path = "/testFile";
// Request the asynchronous revocation process and
// set the replication factor to be applied
dfs.revokeEncoding(path, (short) 2);
// Done. The file will be replicated asynchronously and
// its parity will be deleted subsequently.

7.4.5 Deletion Of Encoded Files

Deletion of encoded files does not require any special care. The system will automatically take care of
deletion of any additionally stored information.

7.5 Certificate Materialization

Hops and Hopsworks support transparent TLS encryption of all Remote Procedure Calls (RPC). Every user
in the system is associated with a X.509 certificate that is stored in MySQL CLuster (NDB) database. For
all RPCs originating from Hopsworks, the keystore and truststore for the user in question is materialized
from the database in the local filesystem for future use. If multiple sources request the same cryptographic
material, then a reference counter is incremented instead of materializing over and over again. The materials
from the local filesystem are deleted when there are no more references to them. Details for configuring
Hops with TLS enabled can be found in Hops SSL/TLS configuration (page 124)

7.5.1 Certificate Materializer in Hopsworks

The service in Hopsworks that is responsible for the certificates management is the CertificateMaterializer.
It provides two overloaded methods for materializing project-specific and project-generic user certificates,
CertificateMaterializer#materializeCertificates(String username, String
projectName) and CertificateMaterializer#materializeCertificates(String
projectName) respectively.

In order to remove a user’s material that is not needed any longer you can use
CertificateMaterializer#removeCertificate(String username, String
projectName) and CertificateMaterializer#removeCertificate(String
projectName). Note that these methods will not remove the keystore and truststore from the
local file-system unless there are no more references to them.
Finally if you want get access to the material itself, use `CertificateMaterializer.getUserMaterial(String username, String projectName)` and `CertificateMaterializer.getUserMaterial(String projectName)` which will return a `CertificateMaterializer.CryptoMaterial` object containing the keystore, truststore and password.

A typical use case is the following:

```java
private CertificateMaterializer certificateMaterializer;
String username = user.getUsername();
String projectName = project.getName();
try {
    certificateMaterializer.materializeCertificates(username, projectName);
    // Do something with the certificates
} finally {
    certificateMaterializer.removeCertificates(username, projectName);
}
```

Note: For DFS and Yarn client this is handled automatically as long as you use `DistributedFsService` and `YarnClientService`. So you don’t have to materialize and remove them. When a certificate has no more references it will be scheduled for deletion. So even if you have called the `remove` method, you might be able to see them in the local filesystem.

You can query the state of the `CertificateMaterializer` through a REST API, as well as force remove a stale certificate. First, you have to login to Hopsworks with an `admin (HOPS_ADMIN)` role user. To get the state of the service do a GET request to `hopsworks-api/api/admin/materializer`. This will return a map of the username associated with a certificate along with its references number. Also, a list of certificates that are scheduled for removal as depicted below.

```json
{
    "materializedState": [
        {
            "references": 2,
            "user": "projectA__userA"
        }
    ],
    "scheduledRemovals": []
}
```

In order to force remove a stale certificate do a DELETE request to `hopsworks-api/api/admin/materializer/{certificate_username}`. Use this operation with caution as it might disrupt running operations that still use this cryptographic material.

### 7.5.2 Certificate Localization Service in Hops

A similar service is running on the ResourceManager(s) and NodeManager(s) which is called `CertificateLocalizationService`. This service extracts the keystore, truststore and password from RPCs and stores them locally for internal consumption.

---

7.5. Certificate Materialization 145
On the ResourceManager, the material is piggybacked by YarnClient when submitting an application in the `ApplicationSubmissionContext`. The material can be set explicitly or read automatically if they exist in the directory set by `client.materialize.directory` in `core-site.xml`. On the NodeManager, the material is extracted by the RPC which starts a container, `ContainerManagerImpl#startContainers`. Both are transparent to the user.

To localize a certificate you should call `CertificateLocalizationService#materializeCertificates(String username, ByteBuffer keyStore, String keyStorePass, ByteBuffer trustStore, String trustStorePass)`. To remove them call `CertificateLocalizationService#removeMaterial(String username)`. To get information about a crypto material call `CertificateLocalizationService#getMaterialLocation(String username)` which returns a `CryptoMaterial` object.

If RPC TLS is enabled there will be a reference to the service in `RMContext` of RM or to `Context` of NM. An example of usage is following:

```java
try {
    // Do something with the certificates
    CryptoMaterial material = rmContext.getCertificateLocalizationService()
        .getMaterialLocation(user);
} finally {
    rmContext.getCertificateLocalizationService().removeMaterial(user);
}
```

Both in RM and NM the materials are stored in a safe directory in `/tmp` of the local file-system readable only by the user running the service.

Similar administrative operations as in CertificateMaterializer of Hopsworks are provided by the CertificateLocalizationService but through JMX. There is a JMX call to get the state of the service and to force remove a certificate. Both operations are password protected. The username and the password of the role allowed is in `$HADOOP_HOME/etc/hadoop/yarn-jmxremote.password` and `$HADOOP_HOME/etc/hadoop/rm-jmxremote.password`. The content of the file should be similar to:

```
adminRole adminPassword
```

Also, a policy file is expected in `$HADOOP_HOME/etc/hadoop/jmxremote.access` with read/write access for the admin user, such as:

```
adminRole readwrite
```

The object name of the MXBean for the ResourceManager is `Hadoop:service=RM, name=CertificateLocalizer` while for the NodeManager `Hadoop:service=NM, name=CertificateLocalizer`. You can use VisualVM to graphically interact with the beans. Open VisualVM, add a remote connection with the configured username and password. Once you connect, click on the MBeans tab (you might have to install the plugin). You will see something similar to the figure.
below. On the MBeans tab, under Hadoop > RM there is the CertificateLocalizer bean where you can get the state of the service. The return value is a JSON with the certificate name and the number of references.

![CertificateLocalization service state](image)

**Fig. 1: CertificateLocalization service state**

Next to the Attributes tabs there is the Operations tab where you can force delete a certificate by typing the username and click on the button. If there are still operations using the material, deleting them will **disrupt** the application.
Fig. 2: CertificateLocalization service force remove material
Hops License Compatibility

Hops combines Apache and GPLv2 licensed code, from Hops and MySQL Cluster, respectively, by providing a DAL API (similar to JDBC). We dynamically link our DAL implementation for MySQL Cluster with the Hops code. Both binaries are distributed separately. Hops derives from Hadoop and, as such, it is available under the Apache version 2.0 open-source licensing model. MySQL Cluster and its client connectors, on the other hand, are licensed under the GPL version 2.0 licensing model. Similar to the JDBC model, we have introduced a Data Access Layer (DAL) API to bridge our code licensed under the Apache model with the MySQL Cluster connector libraries, licensed under the GPL v2.0 model. The DAL API is licensed under the Apache v2.0 model. The DAL API is statically linked to both Hops and our client library for MySQL Cluster that implements the DAL API. Our client library that implements the DAL API for MySQL Cluster, however, is licensed under the GPL v2.0 model, but static linking of Apache v2 code to GPL V2 code is allowed, as stated in the MySQL FOSS license exception. The FOSS License Exception permits use of the GPL-licensed MySQL Client Libraries with software applications licensed under certain other FOSS licenses without causing the entire derivative work to be subject to the GPL. However, to comply with the terms of both licensing models, the DAL API needs to generic and different implementations of it for different databases are possible. Although, we only currently support MySQL Cluster, you are free to develop your own DAL API client and run Hops on a different database. The main requirements for the database are support for transactions, read/write locks and at least read-committed isolation.

8.1 Hopsworks Community License

Hopsworks is provided under the AGPLv3 open-source license. See the list of Hopsworks’ dependencies and their licenses here.64

64 https://github.com/logicalclocks/hopsworks/blob/master/LICENSE_OF_DEPENDENCIES.md