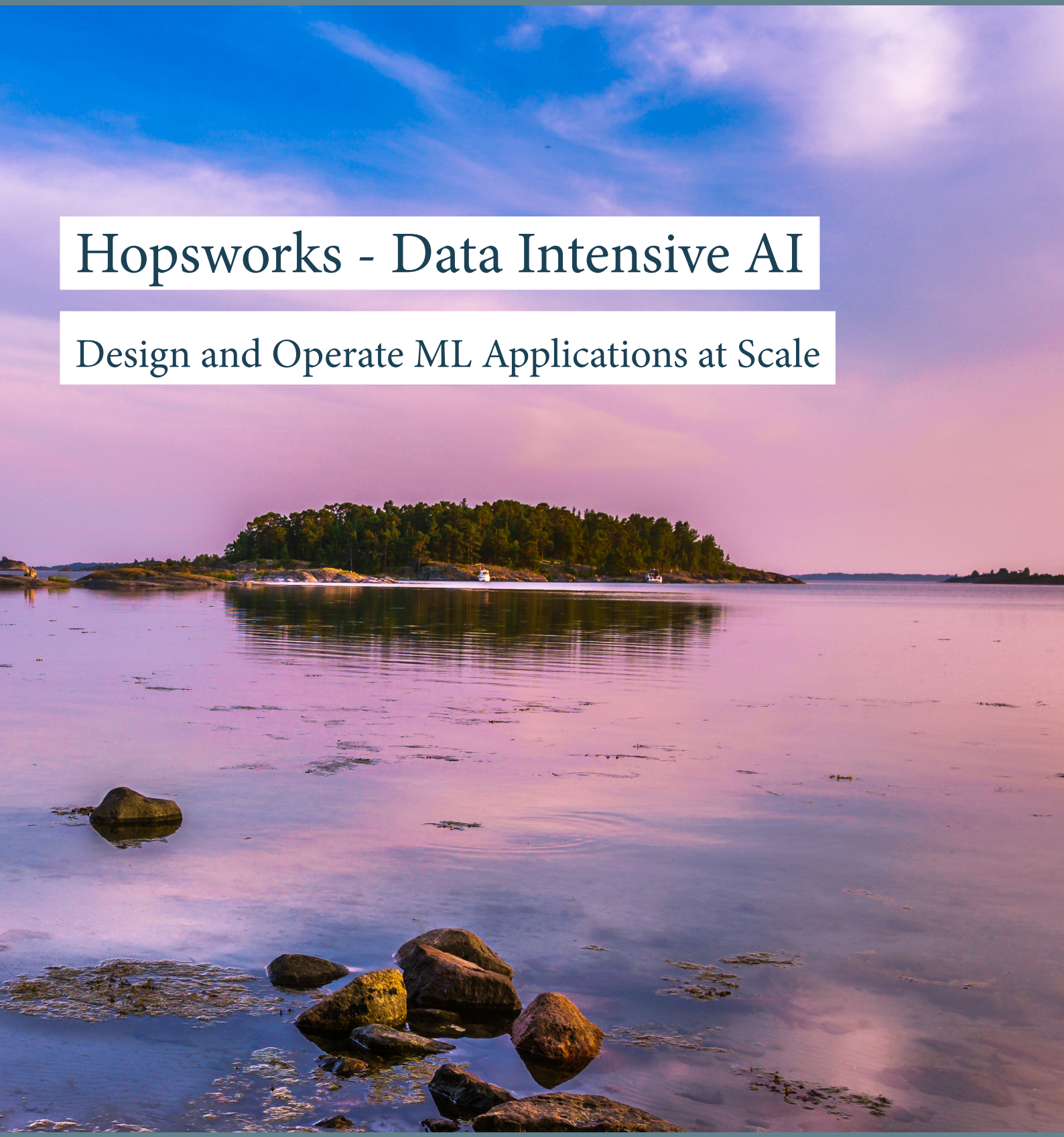
LOGICAL CLOCKS

WHITE PAPER

Hopsworks - Data Intensive AI

Design and Operate ML Applications at Scale



Hopsworks

Data-Intensive AI

Hopsworks is an open-source Enterprise platform for designing and operating machine learning (ML) pipelines at scale. You can write end-to-end ML pipelines entirely in Python and all pipeline stages can be easily scaled out to handle more data and progress faster. Hopsworks supports popular open-source frameworks for data engineering and data science, including ScikitLearn, Spark, Beam/Flink, TensorFlow, PyTorch. Hopsworks makes it easier for Data Scientists to write production-ready code, by supporting a Feature Store to ensure data quality and clean training data for ML models, and also by making Jupyter notebooks first-class citizens in the platform. Notebooks can be used to write production code that is run directly in ML pipelines. Airflow can be used to orchestrate and operate the different stages in ML pipelines, while Hopsworks also provides support for HopsFS, the world's most scalable HDFS-compatible filesystem, with unique support for small files and high throughput.

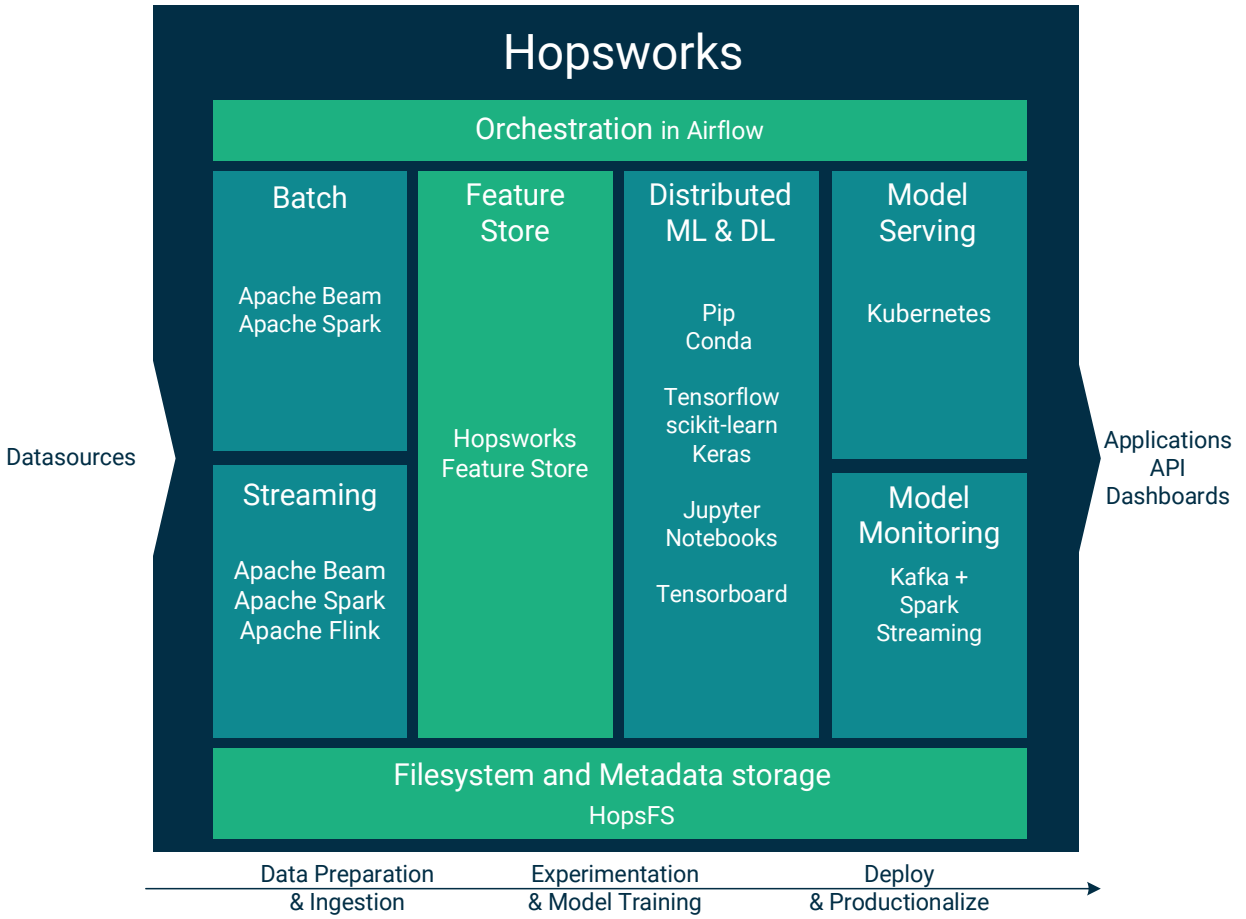


Figure 1: Hopsworks is an Enterprise Platform for designing and operating ML applications at scale.

Dynamic Role-based Access Control

Manage Projects like Github Repositories and share Datasets like Dropbox

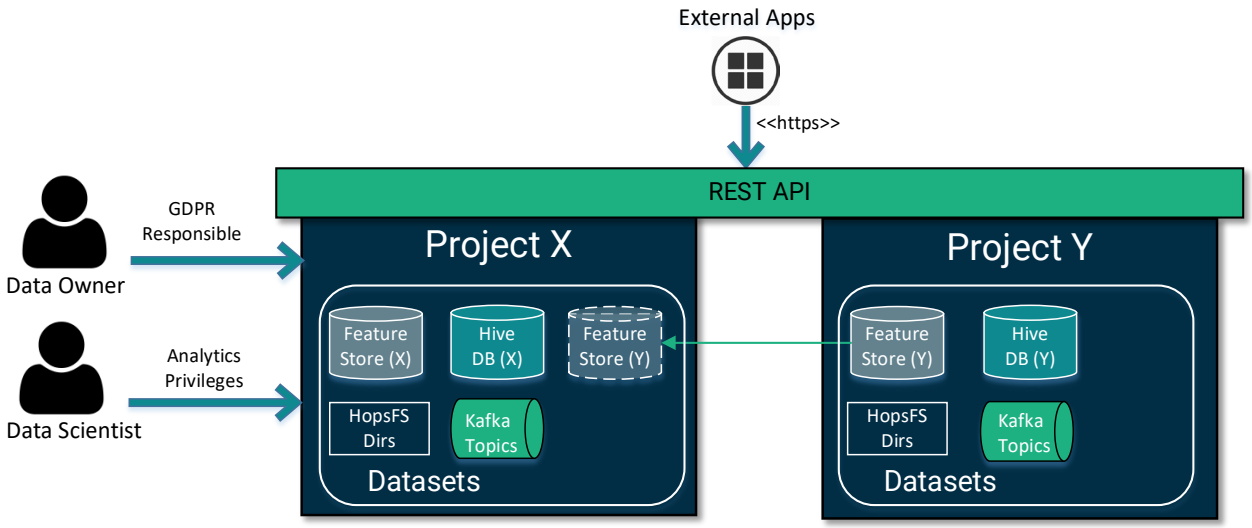


Figure 2: Projects and Datasets are first-class entities. Files, databases, feature-stores can be shared between projects.

Hopsworks provides a new 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, and users can be assigned roles that prevent them from exporting data from the project. In Hopsworks, sharing data does not involve copying data. Datasets can still be securely shared between projects, without the need for duplicating the dataset. In Hopsworks, thanks to a unified metadata layer, a Dataset is not just a directory in HopsFS, but also a Feature Store, a Hive databases, or a Kafka topic. That is, databases, feature stores, and Kafka topics are all multi-tenant - they are private to a project, but can also be securely shared between projects. Hopsworks implements its project-based multi-tenancy security model by internally using X.509 certificates for user authentication, with a new certificate created for every user in a project. Hopsworks also provides role-based access control within projects, with pre-defined "Data Owner" and "Data Scientist" 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).

Scalability at Every Stage in a ML Pipeline

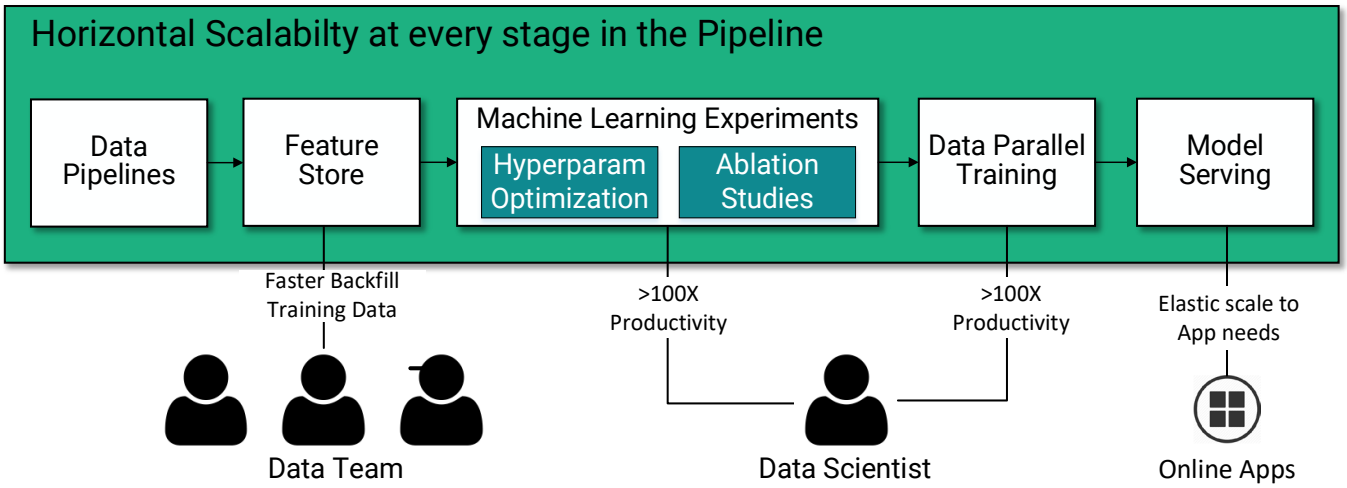


Figure 3: Horizontal Scalability of ML Pipelines in Hopsworks can both increase Data Scientist productivity and reduce the time required to put models in production. as well as enabling the training of models on larger datasets.

Hopsworks enables Data Scientists and Data Engineers to write ML pipeline code that can, without changes, scale from a single virtual machine on a laptop to a whole cluster. Every stage in a Hopsworks ML pipelines is horizontally scalable. ML pipelines will not bottleneck on Feature Engineering pipeline stages can be written with data parallel frameworks, including Spark, PySpark, and Beam/Flink. For backfilling training datasets, the Feature Store can be scaled to store PBs of data and run parallel jobs to quickly create training datasets in the file format of chioce for Data Scientists (.tfrecords, .numpy, .csv, .petastorm, .hdf5, etc).

- **Larger Datasets**. Hopsworks' distributed filesystem, HopsFS, also enables the efficient storage and processing of large datasets - from MBs to PBs in size. HopsFS is important in on-premises deployments, where no object store is available. HopsFS has a HDFS API with native support in Spark, Beam/Flink, TensorFlow, Pandas, and PyTorch (through Petastorm).
- **Parallel Experimentation**. GPUs are an expensive resource, but Data Scientists are even more expensive Hopsworks enables the parallel execution of hyperparameter optimization experiments and ablation studies. The hops Python library uses PySpark to parallelize the hyperparameter trials in TensorFlow/Keras, and PySpark, and ScikitLearnpik.
- **Distributed Training**. Hopsworks supports distributed training of models in TensorFlow and PyTorch, using PySpark to hide the complexity of setting up and managing the distributed ring of workers in CollectiveAllReduce.
- **Elastic Model Serving**. Hopsworks uses Kubernetes to support the dynamic scaling up or down of the number of model serving servers used for a given model. This allows the amount of compute used for online models to be dynamically sized to the needs of the online applications that use those models..

End-to-End ML Pipelines

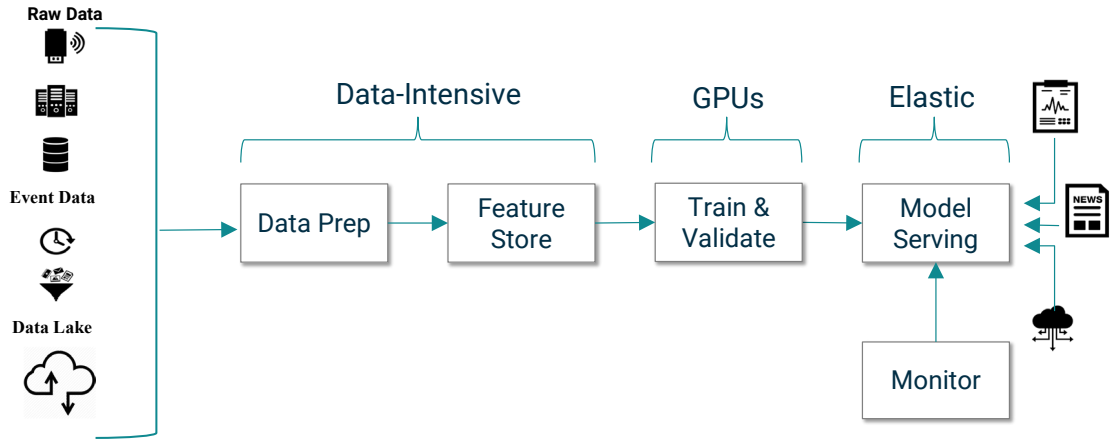


Figure 4: Productionize ML Applications with a Data-Intensive End-to-End Machine Learning Pipeline. Hopsworks supports frameworks to ensure scalable end-to-end pipelines: PySpark, TensorFlow, PyTorch, Scikit-learn, and Apache Beam/Flink.

While much attention has been heaped on TensorFlow/Kerass,PyTorch, H20, and Scikit-Learn as the most popular open-source frameworks for training machine learning models, there is less clarity in industry about the open-source frameworks that should be used to build Machine Learning pipelines. ML pipelines are the fundamental building block for productionizing ML models, as they are responsible for reliably taking new data, cleansing and featurizing it, training the new model, validating the model, and finally (if all tests pass) deploying the model to production. If a model is running in production as a real-time model, infrastructure is also needed to monitor the model and notify if it is not performing as expected.

As supervised ML models benefit from increasing amounts of training data, most ML pipelines are designed from the beginning to be horizontally scalable, that is, they can be scaled to the correct size of input data (from a single container for small data to a cluster of hundreds of containers for Big Data). Apache Spark and Apache Beam/Flink are dominant data-parallel programming frameworks for building the data pipelines needed to feed ML models. The same code written for Spark or Beam can process from MBs to TBs of data using from one to thousands of CPU cores. Hopsworks' is an open platform for ML pipelines and supports the two dominant paradigms for building ML pipelines: Apache Spark and Apache Beam (in combination with TensorFlow Extended and Apache Flink), along with TensorFlow/Keras, PyTorch, Scikit-Learn, and H20 for training ML models.

Hopsworks also includes an orchestration framework, Airflow, to coorrdinate the execution of ML pipelines. The orchestration logic can be written in Python, enabling entire End-to-End ML pipelines to be written in Python. Java/Scala are also supported and often used for the data preparation stages of ML pipelines.

Hopsworks' Feature Store

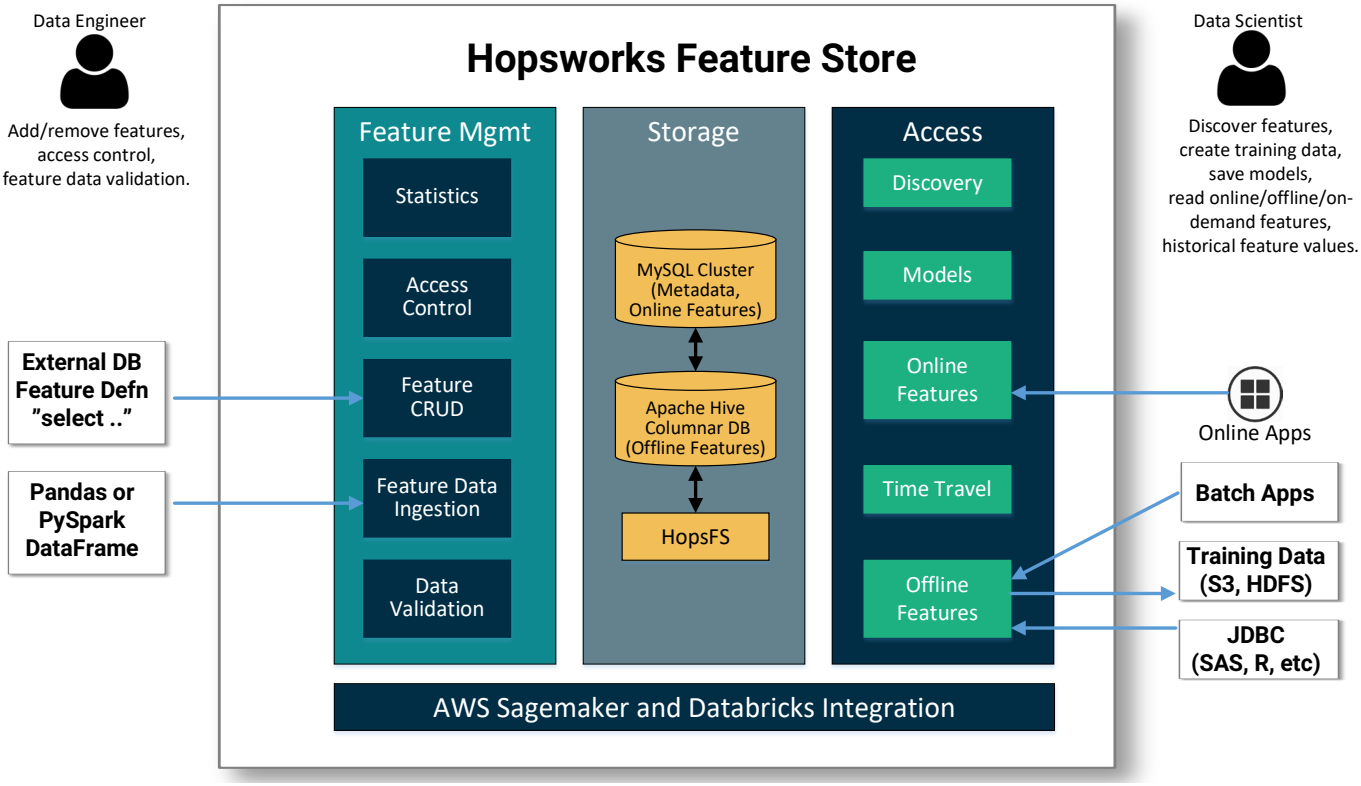


Figure 5: Hopsworks' Feature Store

Hopsworks' Feature Store [SysML'19] is a new data layer in horizontally scalable machine learning pipelines that:

- enables features to be discovered, analyzed, and reused across applications,
- ensures consistency of feature engineering between training and model servicing,
- enables time-travel queries to read historical values for feature values (important to generate new training data),
- and helps ensure high quality feature data through integration with data validation tooling.

In the Feature Store, Data Engineers typically have main responsibility for adding new features to the feature store. New features are added to meet new requirements from Data Scientists. However, if the feature is a simple SQL string for an external datastore, then Data Scientists can often handle such features themselves. Hopsworks supports the concept of projects. A project is a secure repository of data and code and members, where ach member has either a data owner role or a more restricted data scientist role. Each project can have its own FeatureStore. This way organizations can have a global feature store for less sensitive features (in a global project that all employees are a member of), while sensitive features can reside in a closed project with control over which users have access to the features. Features can be defined either in applications (Python/Scala/Java) or in the Hopsworks UI (for example, for simple features that are SQL queries on external databases). Feature data can be ingested using either a Python or Scala/Java API that takes a Pandas or Spark dataframe, and registers it as a FeatureGroup, along with user-supplied metadata for the features (name, description, etc). The data for a FeatureGroup needs to be validated using the Data Validation API before it is used by Data Scientists to create training data for models. Users can specify in the Hopsworks UI or in a data engineering application data validation rules to enforce expected values and ranges for features. Feature statistics can also be access via the Hopsworks UI or from Hopsworks' REST API.

ML Pipelines with PySpark

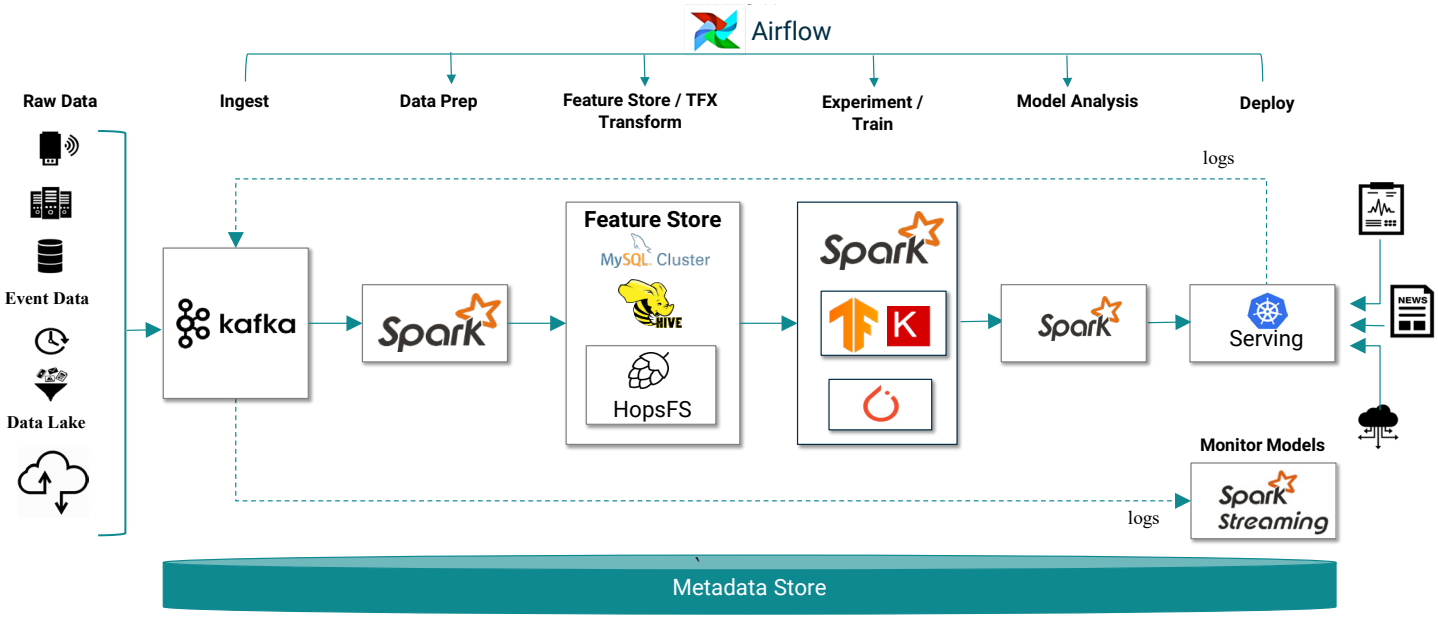


Figure 6: End-to-End ML Pipelines in Python with PySpark, TensorFlow/Keras/PyTorch.

Data pipelines are challenging to develop as they are expected to be completely reliable but they have no control over their input data – it is hard to test a data pipeline against all known data inputs, when you don't know all known data inputs. ML applications differ from traditional data processing pipelines by placing additional requirements on the underlying infrastructure. Hopsworks provides the following features to support end-to-end ML pipelines using Apache Spark::

- data quality validation using the Deequ library (similar to TFX data validation),
- integration with the Hopsworks' Feature Store, where Spark (or Pandas) dataframes can be materialized to the (online and/or offline) Feature Store,
- unique support for paralleized trials and training of ML models.

Hopsworks also includes unique support for parallelizing both hyperparameter optimization trials and ablation study experiments with PySpark. With the Maggy framework, developed by Logical Clocks, Hopsworks now supports the industry's most advanced support for both reducing the time required to execute hyperparameter optimization trials and optimize GPU utilization. Maggy provides a novel architecture to enable the asynchronous execution of trials in Apache Spark, early stopping of trials, and custom optimizers to support directed search. Maggy includes Asynchronous Successive Halving, random search, and grid search out-of-the-box, and customizeed optimizers can easily be included.

ML Pipelines with TensorFlow Extended (TFX)

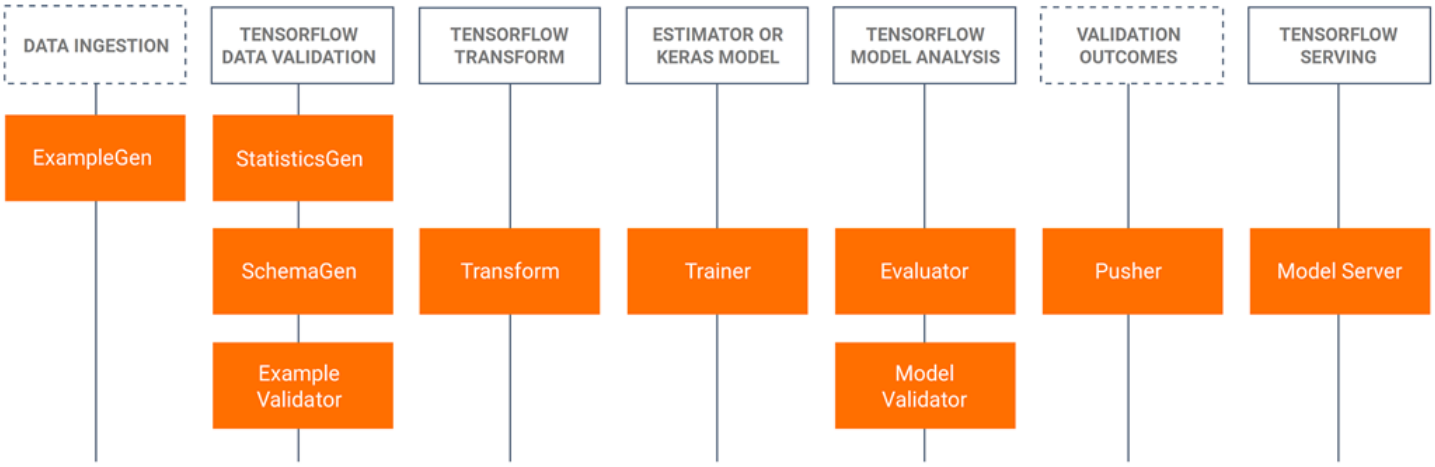


Figure 7: TensorFlow Extended supports components that help ensure data quality, model quality, and consistent feature engineering. (Image from: https://www.tensorflow.org/tfx)

Hopsworks supports the use of TensorFlow Extended (TFX) components in ML pipelines. In Hopsworks, ML pipelines are executed as Airflow DAGs (Python programs that define a workflow as a directed acyclic graph). Hopsworks supports TFX components as stages in ML pipelines, so you can use TensorFlow Data Validation, TensorFlow Transform, TensorFlow Model Analysis in your ML pipelines. In Hopsworks, there is no need to write a TFX pipeline to gain the benefits of TFX, as TFX components can be written and tested in Jupyter notebooks or as Python programs and included directly in a Airflow ML pipeline.

Apache Beam/Flink for TFX

Hopsworks supports Apache Beam for the execution of TFX components, such as Data Validation, TFX Transform, and Model Analysis . TFX components require Apache Beam to be able to scale to handle large volumes of data, and Beam requires an execution engine (runner) to parallelize the execution of Apache Beam jobs. Apache Flink is the most complete open-source runner for Apache Beam, and Hopsworks supports the execution of both Apache Flink and Apache Beam jobs (using the Flink runner). Apache Beam jobs in Hopsworks can be written in either Python or Java, and Apache Beam Python programs can also be written in Jupyter notebooks. Even the Jupyter notebooks can be included in Airflow ML pipelines as Hopsworks jobs.

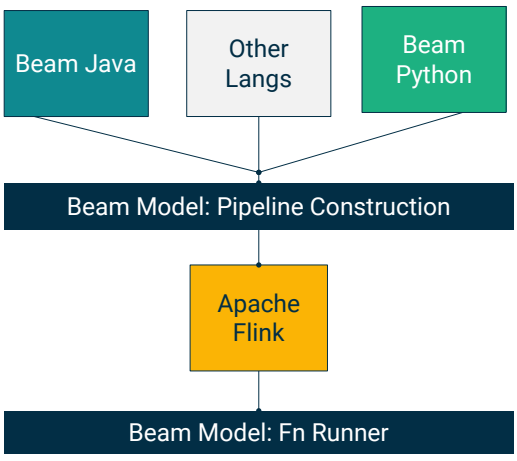


Figure 8: Apache Beam on Hopsworks using Flink

Streaming Analytics in Hopsworks

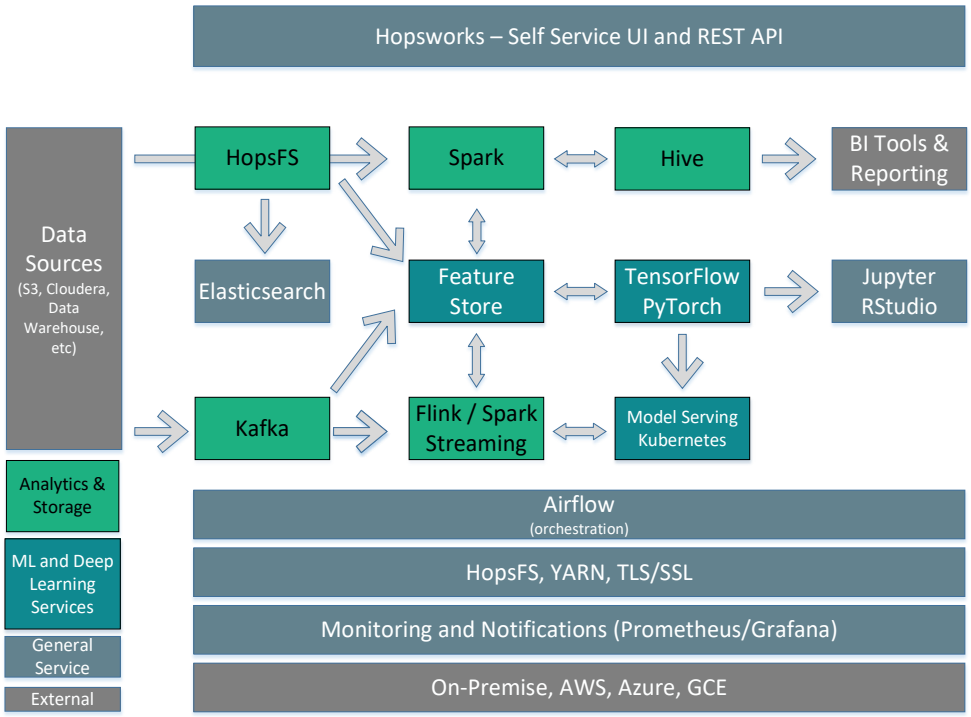


Figure 9: Hopsworks supports both stream processing and data-parallel processing with Apache Spark, Apache Flink, and Apache Beam. Kafka is a fully project-based multi-tenant service in Hopsworks - Kafka topics are private to a project, but can be explicity shared between projects. Kafka access-control support is built using certificates within projects (see Security Architecture).

A Hopsworks installation comes with Apache Kafka customized with unique project-based multi-tenancy support - each project can have its own project-specific Kafka topics that are private to that project. Just like Datasets in Hopsworks, Kafka topics can also be securely shared between projects. Hopsworks' Kafka multi-tenancy is built on a unique TLS-based access control layer for Kafka that integrates with Hopsworks' project membership lists.

Hopsworks also provides library support in Java/Scala/Python to make TLS-enabled Kafka easier to use, see examples here. A fully-configured Kafka consumer or producer can be instantiated in a single line of code:Hopsworks supports Spark Streaming, Beam, and Flink as frameworks for building streaming analytics applications. HopsFS also provides support for checkpointing streaming applications, with its HDFS compatability.

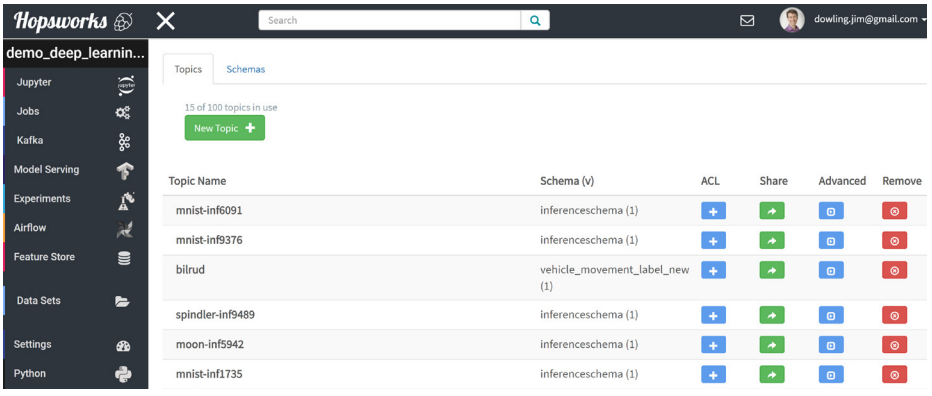


Figure 10: Hopsworks provides UI and API support for managing Kafka topics and Avro Schemas.

Jobs and Airflow (ML Pipelines)

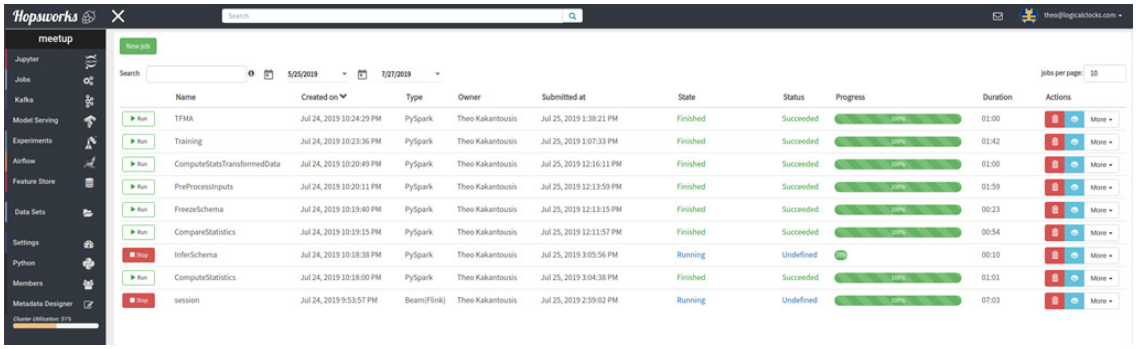


Figure 11: Jobs in Hopsworks can be Spark, PySpark, Beam, Flink, or Kubernetes tasks. Jobs can be run from the UI, the REST API, or from Airflow (that can launch orchestrate their execution, run timed jobs, notify on Job errors, etc).

Hopsworks provides a Jobs service (REST API and UI) to execute programs (Jupyter notebooks, PySpark, Java/Scala Spark, Beam/Flink, Kubernetes tasks). ML applications (TensorFlow/PyTorch/Scikit-learn/etc) are productionized by running them as a Job. Jobs provide UIs for debugging: Spark/Flink UI, logs in Kibana, Grafana for performance debugging, YARN UI for YARN JOBS. Logs for jobs are stored on HopsFS in the Logs dataset, private to the project. Jobs are orchestrated as ML pipelines using Airflow in Hopsworks.

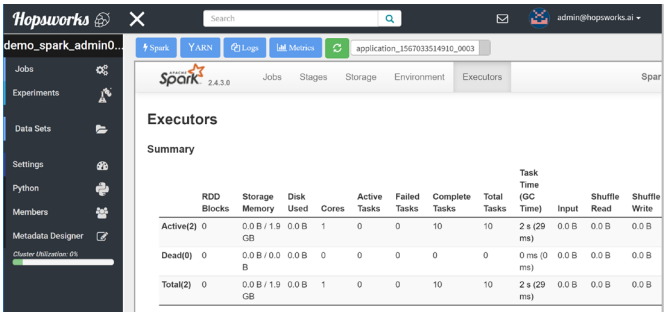


Figure 12: Jobs provide UIs for debugging: Spark/Flink/YARN UI, Kibana for logs, Grafana for resource utilization.

Airflow in Hopsworks

Hopsworks provides project-based multi-tenancy support for Airflow, where Airflow DAGs are private to projects Hopsworks' Airflow includes a Hopsworks JobOperator to run Jobs in Hopsworks. ML pipelines are typically chains of Hopsworks' Jobs, with additional error-handling logic and support for notifications (Slack, email, etc).

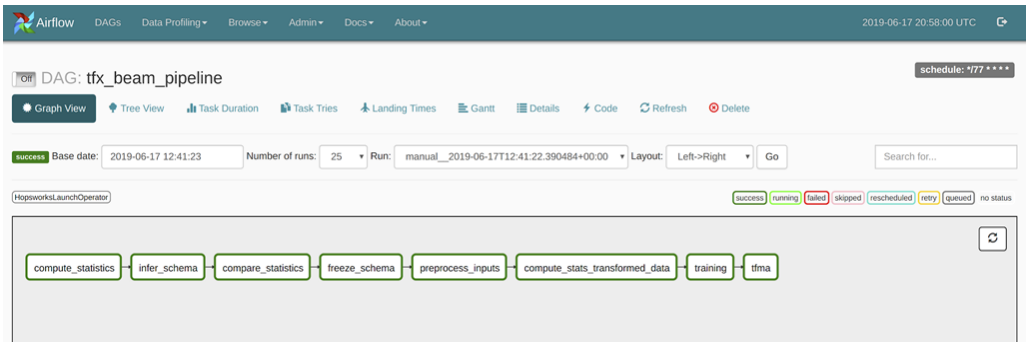


Figure 13: Airflow can run Jobs in Hopsworks using the HopsworksOperator. In this example, Airflow is running a pipeline of Hopsworks TFX Jobs.

Model Management, Serving and Monitoring

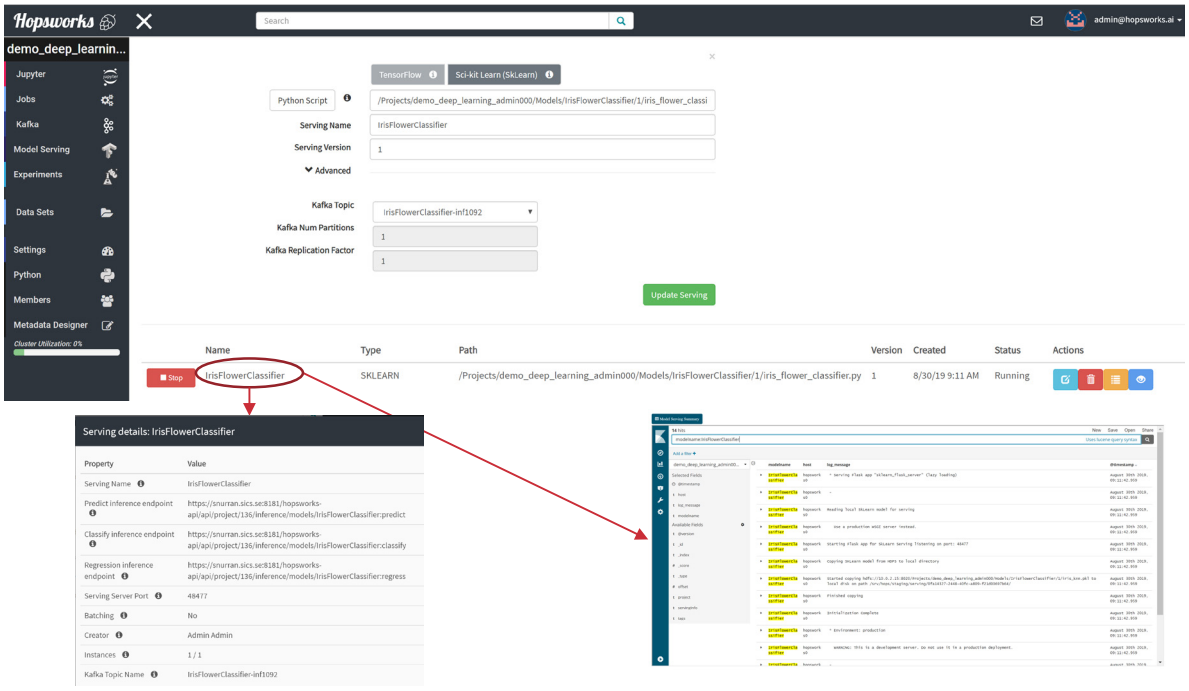


Figure 14: Model Management, Serving and Monitoring in Hopsworks

Models are the valuable output of a machine learning training run. Hopsworks manages models and annotates them extensively with metadata, so that developers can easily perform root cause analysis when a model behaves in an unexpected manner. In Hopsworks, models can be managed either from the UI, see image above, or from the REST API. Models can be stored, listed, downloaded, as well as run as online model serving servers. Hopsworks support online model serving for TensorFlow/Keras (using TensorFlow model serving server) and using the hops python library for Scikit-learn and H2O applications (the model server is a flask server managed by Hopsworks). Model servers are run on Kubernetes and their logs can be viewed in the Hopsworks UI in realtime, and prediction requests/results can be automatically logged to a Kafka topic, from where they can be processed in real-time for monitoring/archiving. When the hops python API is used to save models after training, see example notebook, the saved model is linked to the input training dataset, the jupyter notebook or python file used to train it and the conda.yml environment used to execute that Python program. When you debug a model, you can easily navigate from the model to the program and dataset used to train the model, helping you to reproduce the training run and finding the root-cause of the model's problem. Hopsworks also collects statistics on models that are visualized in the UI- how long it took to train them, who trained them, the application/notebook/python-program used to train them, the training dataset used (its version and its versioned features from the feature store), the hyperparameters used to train the model, and the output performance of the model on its evaluation dataset;

Notebooks as First-Class Citizens

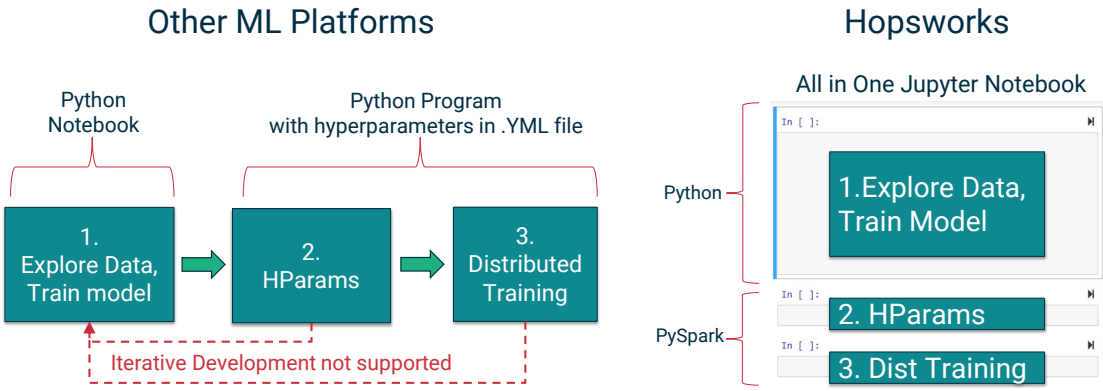


Figure 15: Other ML Plaforms throw away Notebook code.

Notebooks are the future of Data tooling and are at the heart of Neflix's data platform - Netflix run >100k Jupyter notebooks/day. Hopsworks has first-class support for Jupyter notebooks, enabling them to be used for more than just explorative development and visualization. Notebooks can be parameterized jobs in production ML pipelines. Hopsworks supports a development process where ML developers can start by writing a Python notebook that can then be easily extended to run as a PySpark job - parallelizing hyperparamter optimization tasks and massively reducing training time for deep neural networks by training on up to 100s of GPUs. The hops python library makes writing such distributed programs as easy as writing single-threaded Python programs.

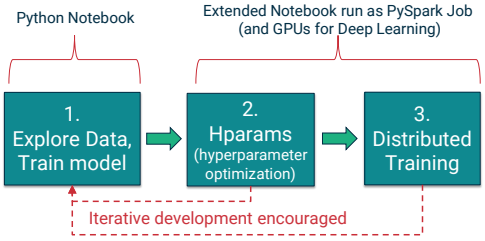


Figure 16: In Hopsworks, the inner training loop for your ML program is written in Python. Other cells in your nodebook can be later added to run the same notebook as a PySpark jobs with parallel hyperparameter trials or to support distributed training to speed up training if you have a large dataset.

Notebooks in ML Pipelines

In Hopsworks, Data Scientists can easily build production ML pipelines from their Jupyter notebooks. There is no need to hand over their code to a ML engineer to productionize their work. ML pipelines can be created added to a Python program in Airflow by creating a PySpark Job from a notebook in the UI. When Hopsworks launches the notebook-as-a-job, it converts the .ipynb file to a .py file and then runs it as a PySpark job. Airflow can then orcestrate ML pipelines consisting of mixed notebooks and jobs.

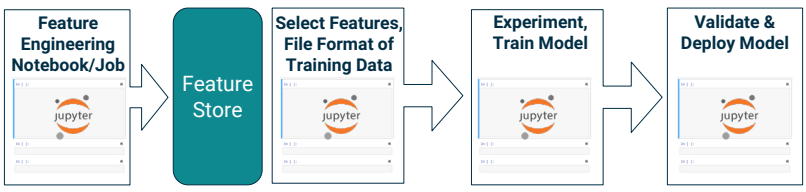


Figure 17: Notebooks can be run directly in ML Pipelines that are orchestrated by Airflow

TensorFlow/Keras or PyTorch

Hopsworks supports the development and training of deep learning models in the latest versions of TensorFlow/Keras and PyTorch. Hopsworks also provides the hops python library with documentation as a way to help make hard things easier in Hopsworks. For example, a single API call is all you need to create a training dataset using the Feature Store, install a python library in your project, create a consumer/producer for Kafka, or run a set of parallel trials for hyperparameter optimization or ablation studies.

CollectiveAllReduceStrategy made Easy

Distributed deep learning offers the promise of reduced training times and the ability to train larger models on larger volumes of data, producing more accurate models (than your competitors). However, vanilla TensorFlow and PyTorch leave the infrastructural complexity of configuring and operating a distributed training program to the developer. Hopsworks, however, makes distributed training as easy as single-threaded training by transparently configuring and managing the lifecycle of the TensorFlow/PyTorch processes, and providing a distributed filesystem to store training data and manage checkpoints, logging, and TensorBoard data. For more details, see this Spark Summit talk.

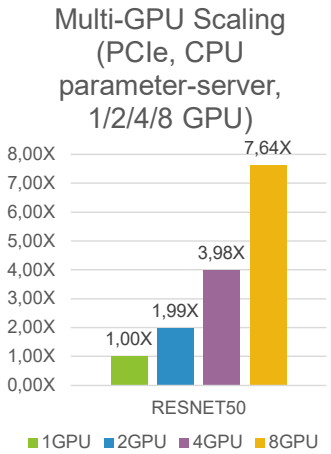


Figure 18: Reduce training time by adding GPUs

Nvidia Cuda™ or AMD ROCm™

Hopsworks supports both Nvidia Cuda™ and AMD ROCm™ for deep learning. Applications written in TensorFlow can be run without any changes required on Nvidia graphics cards (Tesla™ or GeForce™) or on ROCm-enabled AMD graphics cards (such as MI25™, MI50™, and Vega™ R7). Hopsworks support is based on customer GPU support in HopsYARN as well as official Docker™ containers (if applications are trained on Kubernetes™.

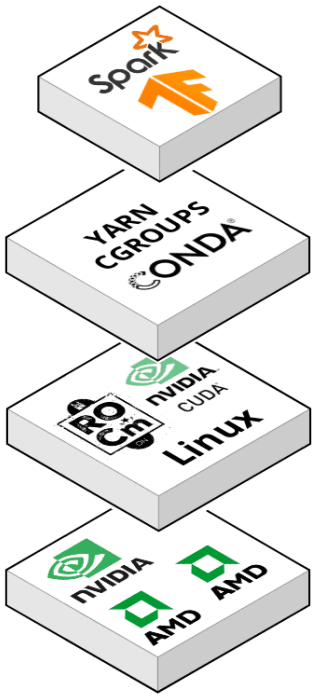


Figure 19: Hopsworks manages GPUs from the driver level up, supporting both Nvidia Cuda and AMD ROCm.

Python, like on your Laptop

Hopsworks is both a development and operational platform for ML applications. ML applications are primarily written in Python, and Hopsworks provides first-in-class support for making Python libraries easy to include when developing (clustered) Python applications and also when running applications in production.Developers can search for Python libraries using either Pip or Conda (private conda repository servers can also be installed alongside Hopsworks) and install them by clicking a button on the UI. Python libraries can even be installed directly in Python applications using Hops API calls.

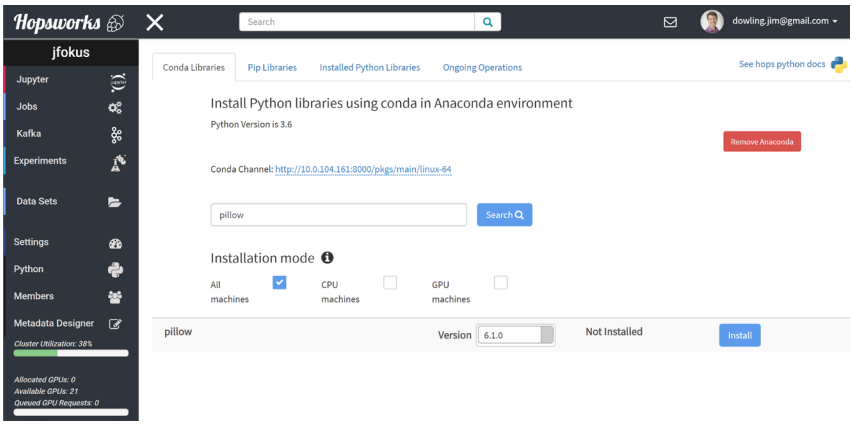


Figure 20: Search for and Install Python libraries using Pip/Conda in the UI.

Immutable Infrastructure with Conda

Instead of requiring developers to write and maintain cumbersome Dockerfiles, Hopsworks uses a conda environment per Hopsworks' project to manage Python dependencies. When users install Python libraries in a project, the base conda environment is forked and a conda environment will be managed at all hosts in the cluster for that Hopsworks project. Instead of requiring developers to write and maintain cumbersome Dockerfiles, Hopsworks uses a conda environment per Hopsworks' project to manage Python dependencies.

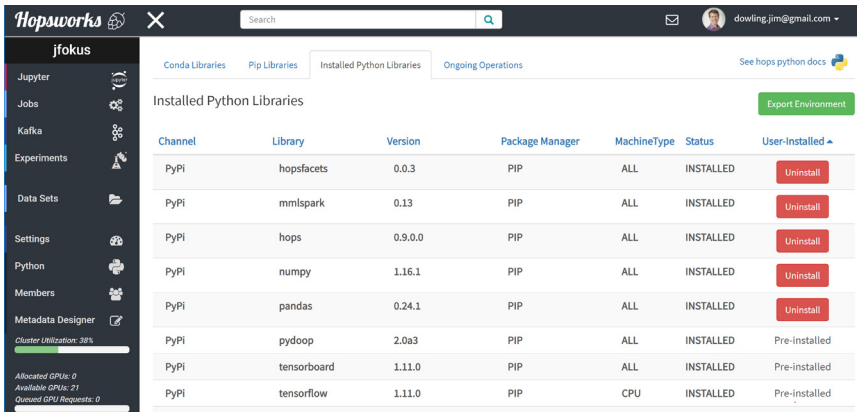


Figure 21: Manage your Conda environment in the UI.

When users install Python libraries in a project, the base conda environment is forked and a conda environment will be managed at all hosts in the cluster for that Hopsworks project. The developer experience is close to the laptop experience - users search for Python libraries and install them and then they can imported in applications. For the ML infrastructure engineer, a project and its Python environment can be cloned and versioned to give an immutable infrastructure for running Python in production.

Hops library: Python and Java/Scala

The hops util library (both Python and Java) provide functions that enable Python, Java, Scala applications to easily use services in the Hopsworks platform: hopsworks, FeatureStore, Kafka, HopsFS, Hive, TLS certificates. Hopsworks also provides standard Hadoop libraries for using object stores, such as S3, and any Python library that can be installed with Pip or Conda can be used.. Python programs can be run on Jupyter notebooks, Kubernetes or as PySpark applications or Beam applications.

Parallel ML with Python functions and PySpark

Supervised machine learning frameworks are typically built around the core abstraction of an inner training loop, inside which the model parameters are updated during training. The hops library makes it easy to write a Python function to run the inner training loop (see below) that is executed in a cluster using PySpark. For hyperparameter optimization, results are collected by the Driver (main scope), while for distributed training, the Driver connects the workers and the distributed filesystem.

```
In [ ]:
def innerloop_fn(args):
    print("Execute Python function")
```

```
In [ ]:
from hops import experiment
args = { "param1": "v1", "param2":"v2"}
experiment.launch(innerloop_fn, args)
```

Figure 22: ML tasks can be parallelized writing the inner training loop in a Python function, and an external Driver (in PySark) creates and configures workers to execute the function in parallel.

Hops Library Examples - Pandas, Numpy, Kafka

The hops python library includes additional support for reading/writing with Pandas and NumPy to HopsFS, see example notebooks here, and sample code shown below. Hops also supports accessing services like Kafka with simple API calls (see below) - the client does not need to configure TLS certificates or the Kafka broker endpoints or the Avro schema for the topic as these are resolved by the hops library for you.

```
In [ ]:
from hops import kafka
consumer = Consumer(kafka.get_kafka_default_config())
consumer.subscribe(["topic1"])
```

```
In [1]:
from hops import hdfs
from hops import numpy_helper as numpy
data = numpy.load("TourData/numpy/C_test.npy")
numpy.save(hdfs.project_path() + "/Resources/full-path.npy", data)
```

```
In [3]:
from hops import hdfs
from hops import pandas_helper as pandas
import pandas as pd
features = ["Age", "Workclass", "fnlwgt", "Education", "Education-Num", "Marital Status",
            "Occupation", "Relationship", "Race", "Sex", "Capital Gain", "Capital Loss",
            "Hours per week", "Country", "Target"]
train_data = pandas.read_csv(hdfs.project_path() + "/TourData/census/adult.data", names=features,
                             sep=r'\s*,\s*', engine='python', na_values="?")
pandas.write_csv(hdfs.project_path() + "/Resources/full-path.csv", train_data)
```

Figure 23: Sample Code snippets using the hops Python library to read (1) consume from a Kafka topic, (2) read a Numpy array from HopsFS, and (2) read a Pandas dataframe from HopsFS.

Citizen Data Scientists

Just as BI tools extended their reach to incorporate easier accessibility to both data and analytics, ML tools also need to be usable by a wider group of users in an organization. Some of the tasks needed as part of ML application development do not require knowledge of programming or statistics, but require domain knowledge of the business, knowledge of the data and its terms of use, and an ability to see relationships in the data that can have predictive insight.

Hopsworks' provides a UI with support for:

- managing access to data through assigning roles to users within the context of a project,
- managing globally accessible datasets and sharing datasets between projects;
- writing feature data validation rules in the Feature Store using UI support - to ensure valid, clean feature data;
- studying models for proper governance and (GDPR) compliance - what features were used to train which models by what users, are the correct tags applied to datasets (anonymized, sensitive, etc);
- managing access to feature stores and sharing of feature stores between different projects;
- feature usage to provide insights into which features are less widely used and may no longer be needed in the Feature Store;
- feature data visualization, to see data distributions, relationships between features, aggregate statistics on features.

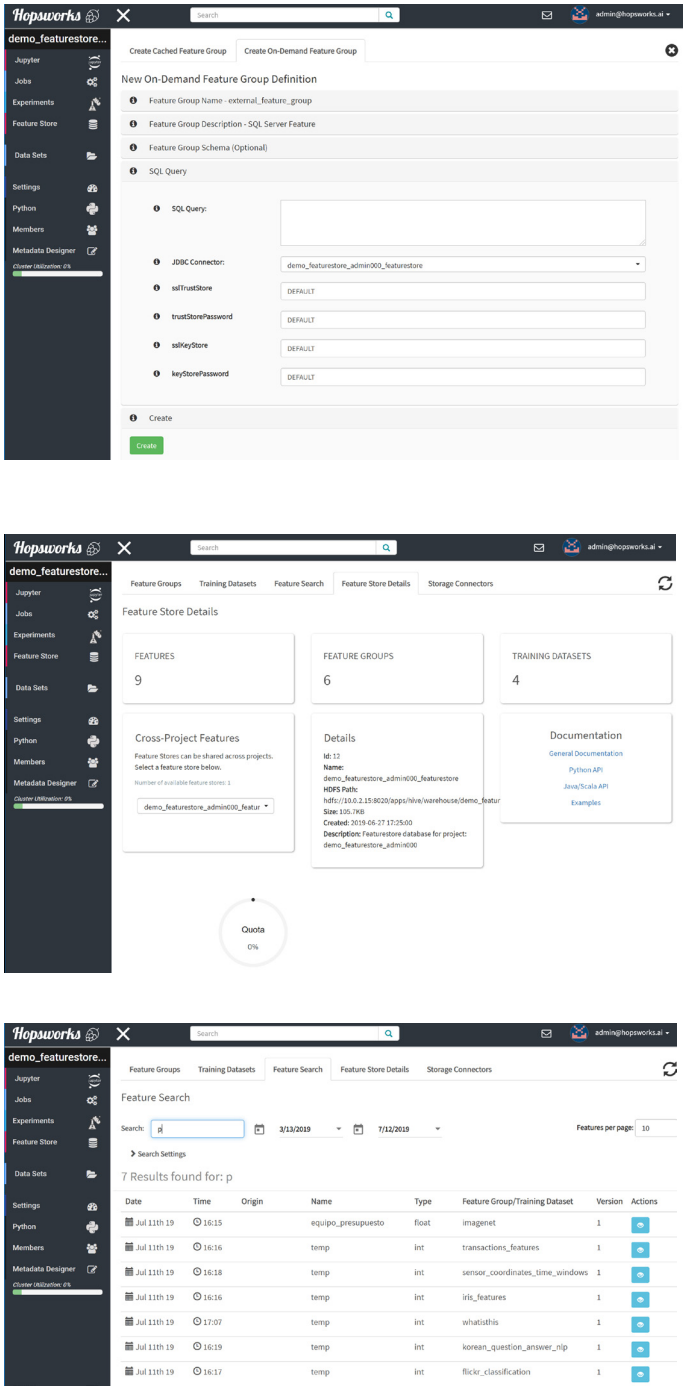


Figure 24: Many Data Scientist tasks (data validation, feature management, data sharing) can be performed in the Hopsworks UI as point-and-click operations.

AutoML with Maggy

Unified Hyperparameter Optimization and Ablation Studies

AutoML (automated machine learning) is concerned with automating, as far as possible, the human work in designing and tuning supervised ML applications for a given dataset. Distribution support is needed to make AutoML work well, and in Hopsworks, we leverage PySpark to automate the allocation of tasks to workers in a cluster. PySpark hides the complexity of distributed programming as you only need to wrap your training code in a function that will automatically be executed in parallel on different workers (with GPUs) in the cluster. With our framework Maggy and the hops python library, we provide API support for both running either synchronous or asynchronous trials for both hyperparameter optimization and ablation studies (ablation studies help you understand the behaviour of your deep neural network if you remove features/layers/regularization). Uniquely, Maggy supports asynchronous trials with early stopping on PySpark, which enables directed hyperparameter search algorithms (such as successive halving), enabling GPU utilization gains of 300% compared to Gooogle's Vizier..

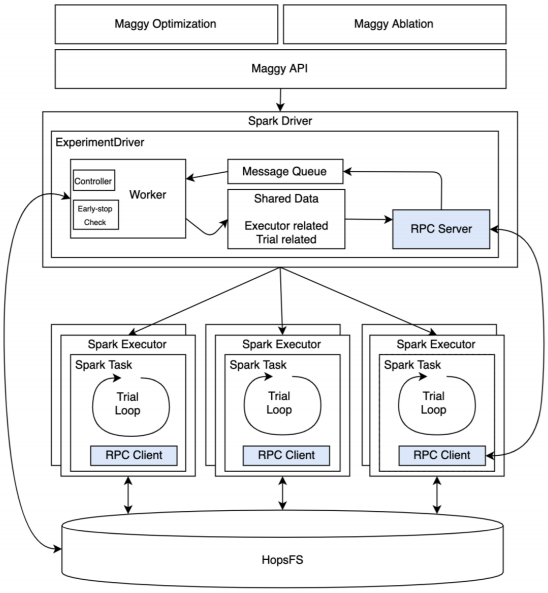


Figure 25: Maggy is a framework for the asynchronous execution of ML trials using Apache

Interactive Distributed Debbuging in Jupyter

Debbugging distributed programs is hard, but Hopsworks, however, makes it easier by enabling Data Scientists to interactively view logs from all workers directly in their Jupyter notebook. Distributed programs, such as parallel hyperparameter optimization or distributed training involve running many workers that all execute the same (inner loop) training function in parallel. In your Jupyter notebook, the main scope of yourPython program calls a function - hops.launch(training_ function) - that starts parallel workers and in the background receives logs from the workers, printing them out directly in the notebook. This way, the same code you can run on your laptop (including standard print statements) will run distributed and print out logs, only each log entry will be prefixed by the worker ID (enabling easy filtering of logs).

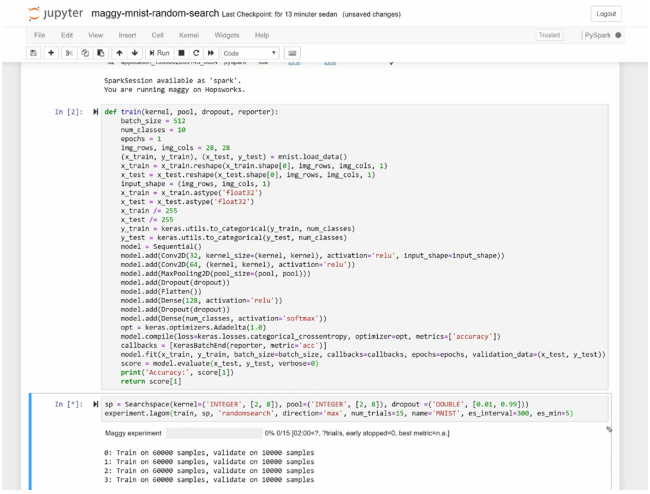


Figure 26: Maggy enables distributed debugging in Jupyter. Log messages from workers are collected and printed out in real-time.

Experiment Tracker

Hopsworks provides an experiment tracking service. similar in scope to Databrick's MLflow tracking service. In contrast to MLflow, there is no need to re-write your ML applications to wrap them inside ML programs. Instead, experiment tracking information is captured when the experiment API in the hops library is used (for example to launch hyperparameter optimization experimens or the training of models.

Hopsworks stores experiment results on HopsFS, and when you perform experiments, training models, results are stored into datasets in your project (/Experiments, / Logs, /Models). Metadata for experiments is stored in Elasticsearch, and is managed by Hopsworks' provenance service.

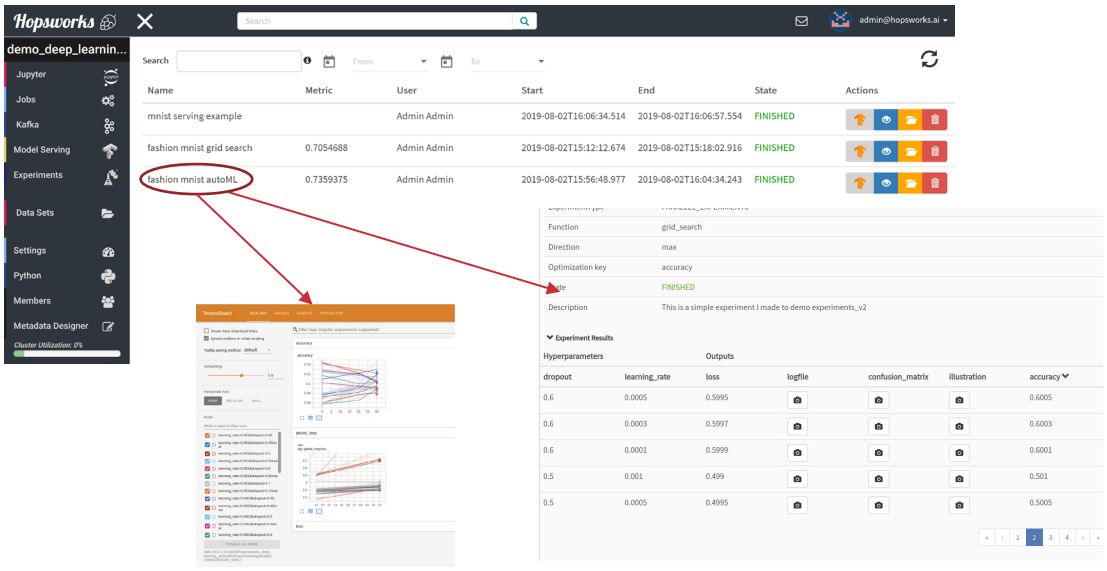


Figure 27: Hopsworks' Experiment Tracker UI.

Provenance/Lineage Tracking

Hopsworks has industry-leading ML provenance capabilities, collecting lineage information both implicitly - through the use of HopsFS filesystem and HopsYARN resource manager - and explicitly - through API calls on the hops python library. Provenance information is integrated into both experiment tracking and model management services, and enables users to:

- debug ML models by easily navigating to the python application, dataset, and Conda env used to train the model;
- manage audit trails for compliance and model interpretability - understand the origin of data, features, and behaviour of models,
- understanding usage of the Hopsworks platform, including user activity, model/feature/dataset popularity.

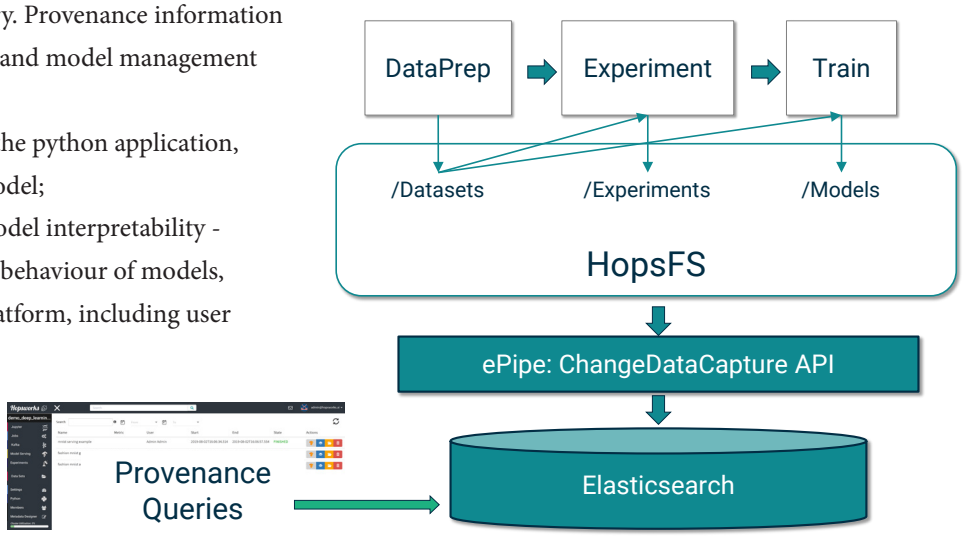


Figure 28: Hopsworks' provenance capabilities come both from observing changes to files in HopsFS (experiment results, model creation/deletion, etc) and from calls to the hops API, such as creating a training dataset or running an experiment.

Model Analysis: What-If Tool

In deep learning parlance, a model is a ready-to-use (trained) deep neural network that can take as input some data it has never seen before and return a prediction. Although there are gaps in our theoretical understanding of deep learning, we can still examine these inscrutable artifacts as black boxes with tooll support. In Hopsworks, we support the What-If Tool (WIT) as a plugin to Jupyter that allows both novice and experienced ML practitioners to analyse trained models to help gain insights into their performance and decision making. It is crucial to understand model behavior for fairness, compliance, GDPR, and for end-to-end software processes, like Karpathy's oftware 2.0 development process. The WIT tool lets users probe inputs and outputs of trained models, to support intersectional analysis, enabling ML practitioners to to answer questions such as "How would increasing the value of age affect a model's prediction scores?" In order to help ML practitioners ask such hypothetical questions, the WIT tool allows users to change (perturb) data points and then evaluate model performance on the changed data. For this, WIT provides a datapoint editor tab. WIT can also be used to compare results across two models for the same dataset.

The WIT tool also supports counterfactual reasoning. For example, for a ML model that predicts whether a user should be given a loan or not, a ML practitioner may be interested in finding out the most similar person to person X who received a loan. We call such data points counterfactual examples and WIT calculates these datapoints using the UI. It does so by including a simple distance metric, you choose either the L1 or L2 norm, which aggregates the differences between data points' feature values across all input features Sometimes you want to know the effect of a feature across an entire range of values. For this, you can use partial dependence plots in WIT to show how model predictions change as the value of a specific feature is adjusted for a given data point. In WIT, partial dependence plots are line charts for numeric features and column charts for categorical features.

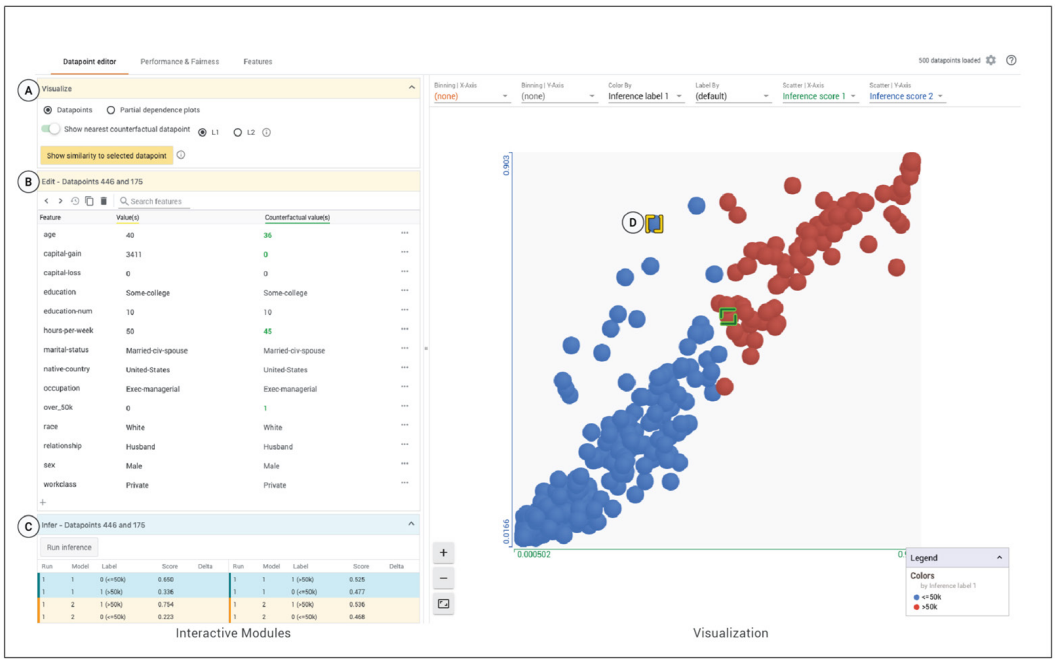


Figure 29: What-If tools from a Jupyter Notebook in Hopsworks

HopsFS - a Distributed FS for ML

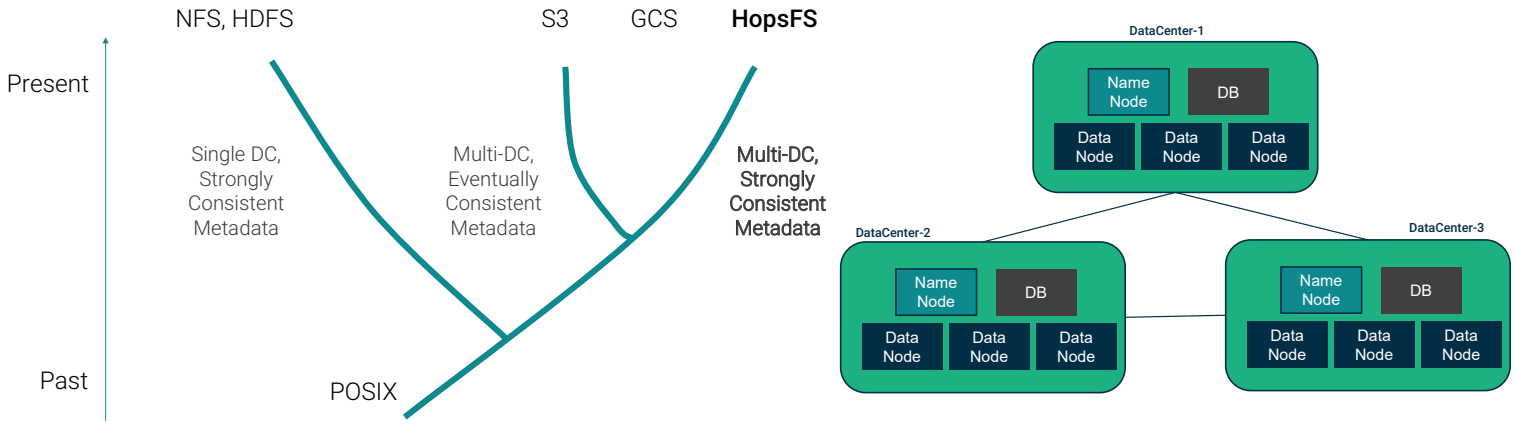
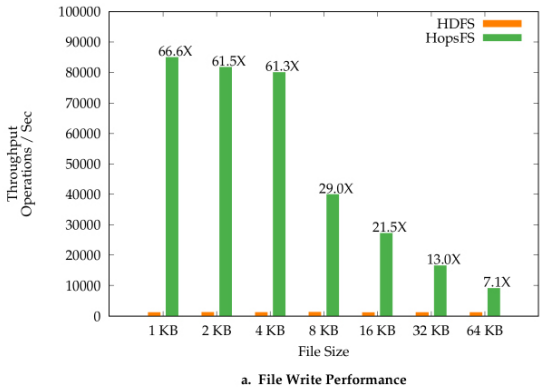
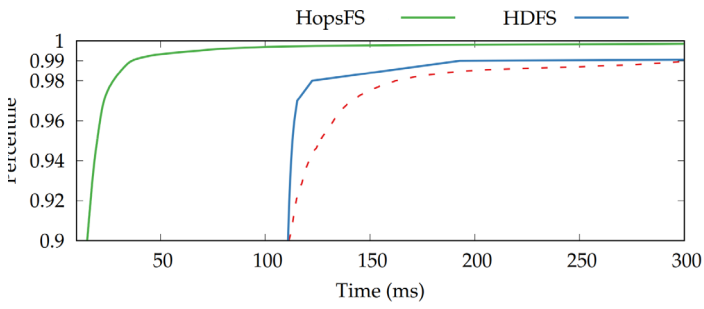


Figure 30: HopsFS is a next-generation distributed POSIX-like filesystem (left) that supports HA over Availability Zones in the Cloud (right).

HopsFS is a next-generation implementation of Apache HDFS with horizontally scalable, strongly consistent metadata. HopsFS' metadata architecture enables it to scale to over 16X the throughput of HDFS on a real-world Spotify workload [HopsFS at Usenix FAST'17]. HopsFS' metadata layer can also be used with NVMe disks to store small files. On the same Spotify workload, we showed over 66X throughput performance increases and up to 100X lower latency (millisecond latency file read/write for small files) [HopsFS at ACM Middleware'18]. HopsFS' distributed metadata layer can also be distributed across data centers for a POSIX-like filesystem with data-center level high availability. On Google Cloud, HopsFS scales to >1.6m ops/s on Spotify's workload while running over 3 different availability zones [Berlin Buzzwords'19]. If an availability zone (cluster) goes down, HopsFS willl continue to work. HopsFS' metadata is not only scalable but it is also strongly consistent, which means we can provide a change data capture API to it, as we do with ePipe [CCGrid'19], which enables free-text search of HopsFS' namespace, and extended metadata. Change data capture is key to how Hopsworks provides extensive non-intrusive provenance support for ML workflows. HopsFS provides an HDFS API and has native support in TensorFlow, Pandas, Spark, PyTorch (Petastorm), Flink/Beam,



a. File Write Performance



a. Spofity Workload: File Write Op End–to–End Latency

Figure 31: Results from: Size Matters: Improving the Performance of Small Files in Hadoop, ACM Middleware 2018.

Hive on Hops

Apache Hive is a data warehouse that runs in Hopsworks providing a SQL-like interface to query data. In Hopsworks, we run a custom Apache Hive that run on HopsFS, unifying Hive's metadata with HopsFS' metadata. The operational benefits of this approach are that it simplifies backups of metadata - you only backup a single MySQL Cluster databases and more importantly, HopsHive ensures the consistency of Hive metadata with its datafiles in HopsFS. That is, if you remove Hive datafiles for a Hive database from HopsFS, Hive's metadata will automatically be cleaned up (dropping the databases). Hopswork's Feature Store extends the same metadata, ensuring strong consistency between its own metadata, Hive's metadata, MySQL Cluster's metadata, and HopsFS.

Every project in Hopsworks can have its own Hive database that is private to that project. Hive databases can, as a dataset, be shared between projects, enabling self-service sharing of datasets between projects.

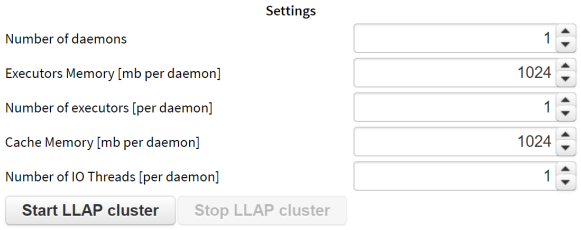


Figure 32: Launch Hive LLAP Clusters in Hopsworks

LLAP

Hopsworks supports Apache Hive 3.x, which includes a low latency engine for querying called LLAP. The Hopsworks admin UI provides an API to start or stop a LLAP cluster that is shared by all projects in Hopsworks.

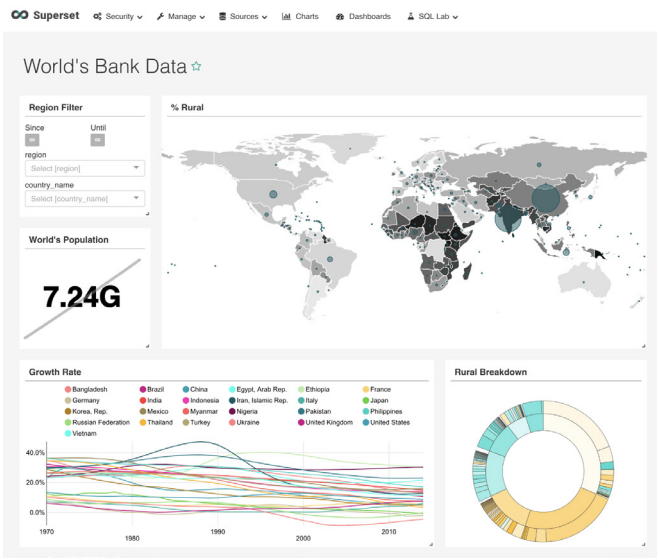


Figure 33: Apache Superset on Hive

Business Intelligence Reporting

HopsHive can be easily integrated with external BI (business intelligence) tools, such as Tableau, Qlik, Apache Superset, to provide visualizations and reporting. HopsHive provides a TLS-enabled JDBC connector and an ODBC connector to allow external tools query data in Hive.

```
In [23]: %%sql
         select * from sacramento_properties_ext limit 10
         Done.
```

Out[23]:

street	city	zip	state	beds	baths	sq__ft	sales_type	sale_date	price	latitude	longitude
3526 HIGH ST	SACRAMENTO	95838	CA	2	1	836.0	Residential	Wed May 21 00:00:00 EDT 2008	59222.0	38.631912	-121.434875
51 OMAHA CT	SACRAMENTO	95823	CA	3	1	1167.0	Residential	Wed May 21 00:00:00 EDT 2008	68212.0	38.4789	-121.43103
2796 BRANCH ST	SACRAMENTO	95815	CA	2	1	796.0	Residential	Wed May 21 00:00:00 EDT 2008	68880.0	38.618305	-121.44384
2805 JANETTE WAY	SACRAMENTO	95815	CA	2	1	852.0	Residential	Wed May 21 00:00:00 EDT 2008	69307.0	38.616837	-121.43915

Figure 34: Query Hive directly in Jupyter Notebooks

Hopsworks TLS-Based Security

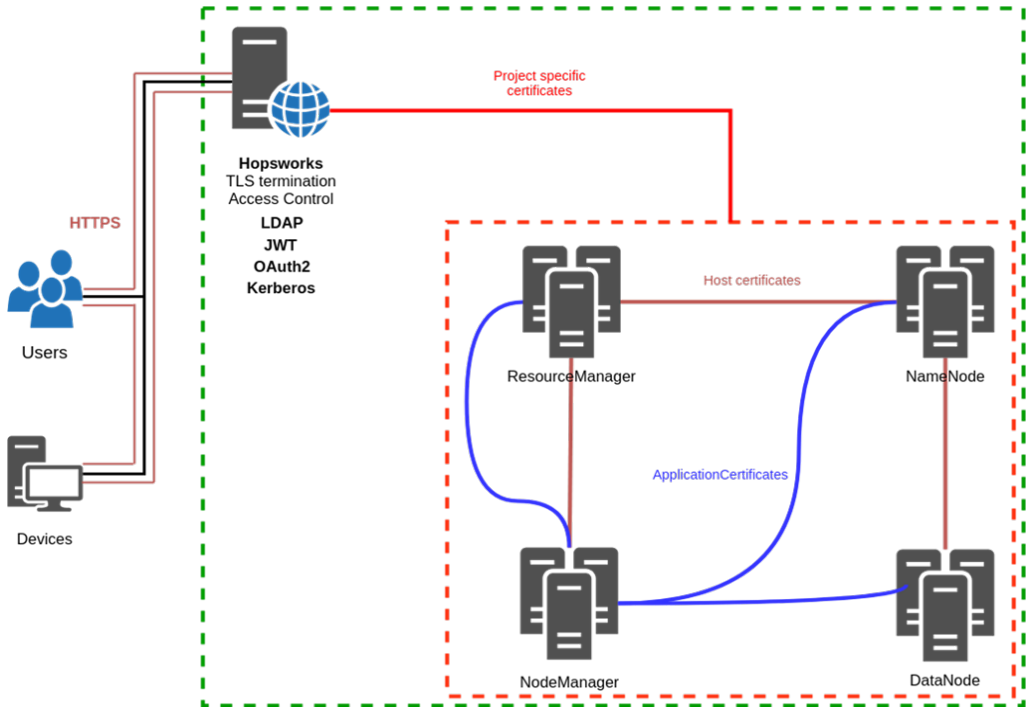


Figure 35: Hopsworks Security Architecture

At Logical Clocks we understand security is critical. We also understand that sometimes it can be cumbersome and users just ignore it. For that reason, in Hopsworks and Hops we have designed our security architecture around TLS/X.509 certificates and we make usage of certificates as transparent as possible to the end-user through API support in Java/Scala/Python. Security is a first-class citizen in Hopsworks.

Hopsworks employs HopsFS as its distributed filesystem and HopsYARN to manage resources in the cluster and execute jobs. Users interact with both the filesystem and the scheduler using Hops clients that require an X.509 certificate to authenticate themselves. Other services in Hopsworks (Kafka, Spark, etc) also communicate with each other, again using Remote Procedure Calls (RPC) that are encrypted using TLS. Hopsworks' unique project-based multi-tenancy, that is based on dynamic-role based access control, is implemented using TLS/X.509 certificates. For every project that a user is a member of, the user has a different certificate. That is, user identity when executing jobs is a combination of the project name and the user - so a user cannot just copy data between her projects, as the system sees her roles in the different projects as different identities. In Kerberos-based data platforms, such as Apache Hadoop, dynamic roles are not possible, as users have a single identity and roles in the access control system (Apache Sentry/Ranger) are static - as they must be as they are liberally cached throughout the platform. For more information on security in Hopsworks, see our website.

Hopsworks Management and Monitoring

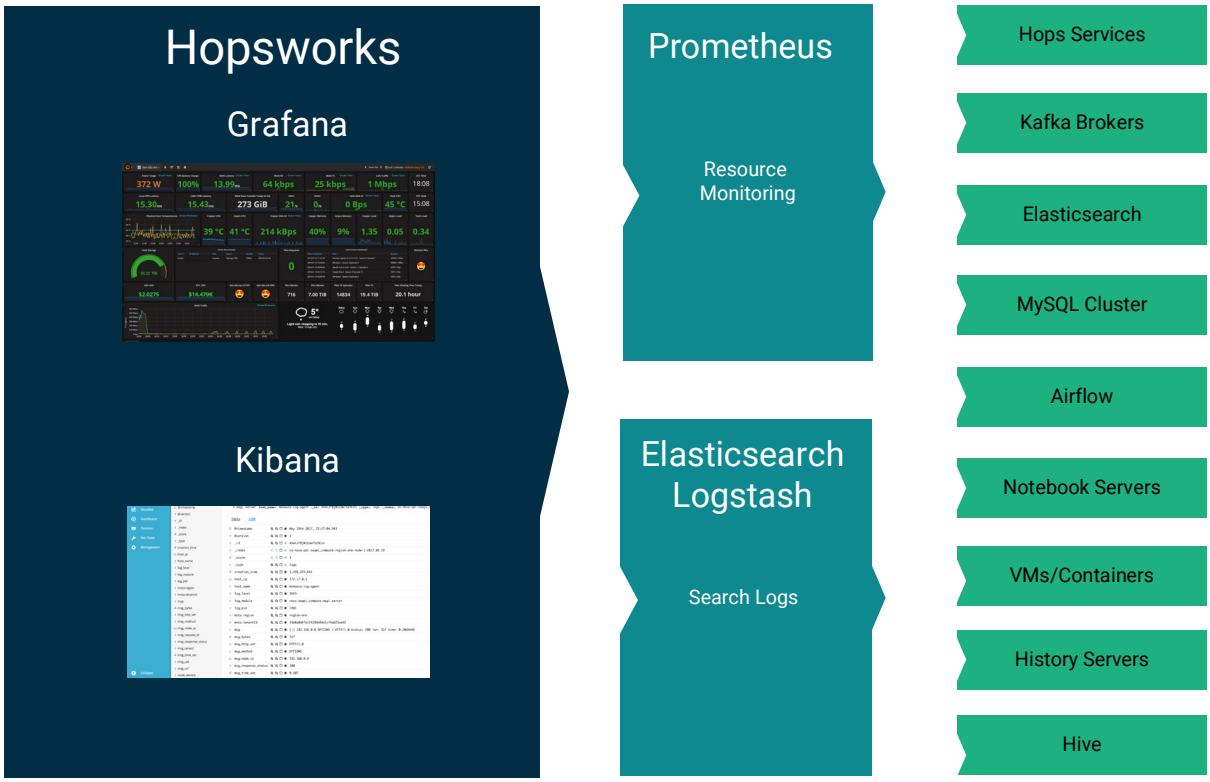


Figure 36: Hopsworks' service/hosts are monitored by Prometheus and visualized with Grafana, while logs are collected with the ELK stack and are searchable within Hopsworks' Admininstration UI.

Hopsworks provides comprehensive monitoring, logging, notification, and administration capabilities for all of its services. The Hopsworks administration UI provides a platform administrator with an overview of the status of all services, the ability to customize notifications, and actions to restart failed services. The Hopsworks administration UI also provides functionality to manage users, projects, quotas (projects have both storage and compute quotas), hosts, certificates, and backups.

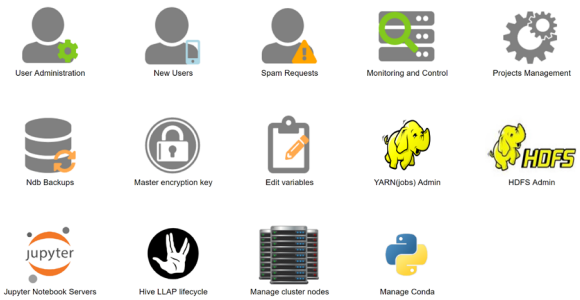


Figure 37: Hopsworks' Admininstration UI.

Hopsworks 1.0 Requirements

Supported Operating Systems: Ubuntu 16.04/18.04, Centos 7.2+, Redhat Linux 7,2+

- Single Host: 32GB RAM (minimum), 4 CPUs (x86), 20 GB+ spare disk capacity
- Supported GPUs: Nvidia Cuda (Tesla, GeForce), AMD ROCm, 2.6 (Ubuntu only)
- Networking: 10Gb/s or 25Gb/s (for distributed training of deep neural networks.
- Clustered Hopsworks:
  Hopsworks server(s): This server runs Hopsworks, and can also run services such as Elastic, Kafka brokers. They require minimum 16 GB RAM (32GB recommended), and 4 CPU cores+ is recommended, along with 20+ GB of spare disk capacity.
  Metadata server(s): These are more CPU intensive, and run the in-memory database (NDB), the NameNodes from HopsFS, and for higher performance should have a local NVMe disk. They should have 8GB+ of RAMa and 4+ CPU cores.
  GPU server(s):These servers typically have 4-10 GPUs connected over either PCI3.0/4.0 or NvLink. They typically only run a nodemanager. We recommend 2 CPU cores per GPU, and 16-32GB or RAM per GPU.
  Worker server(s): These servers run nodemanagers and HopsFS datanodes, and may have high local disk capacity (for on-premise installations).
- Cloud Platforms:
  AMIs for both AWS and GCP are available with community edition of Hopsworks. See website for details.

Hopsworks Enterprise / Community

Hopsworks community edition is a fully featured version that is available under the AGPL-v3 open-source license.

Hopsworks Enterprise Edition has some extra functionality and security, aimed at Enterprises, including:

- Single-Sign-On with ActiveDirectory (Kerberos), OAuth2, LDAP
- Kubernetes support for model-serving, Jupyter notebooks
- Github integration for Jupyter notebooks
- Online Feature Store.

the creators of

HOPSWORKS

Reach out to us!

www.logicalclocks.com

info@logicalclocks.com

.. or visit us at one of our offices:

Silicon Valley Office
470 Ramona St
Palo Alto,
94301 California
USA

Stockholm Office
Box 1263,
Isafjordsgatan 22
164 40, Kista
Sweden

UK Office
IDEALondon,
69 Wilson St
EC2A2BB, London
UK