Noronha

Release 1.6.7

Noronha Development Team

Jan 23, 2023

CONTENTS:

DataOps framework for Machine Learning projects.

CHAPTER

ONE

INTRODUCTION

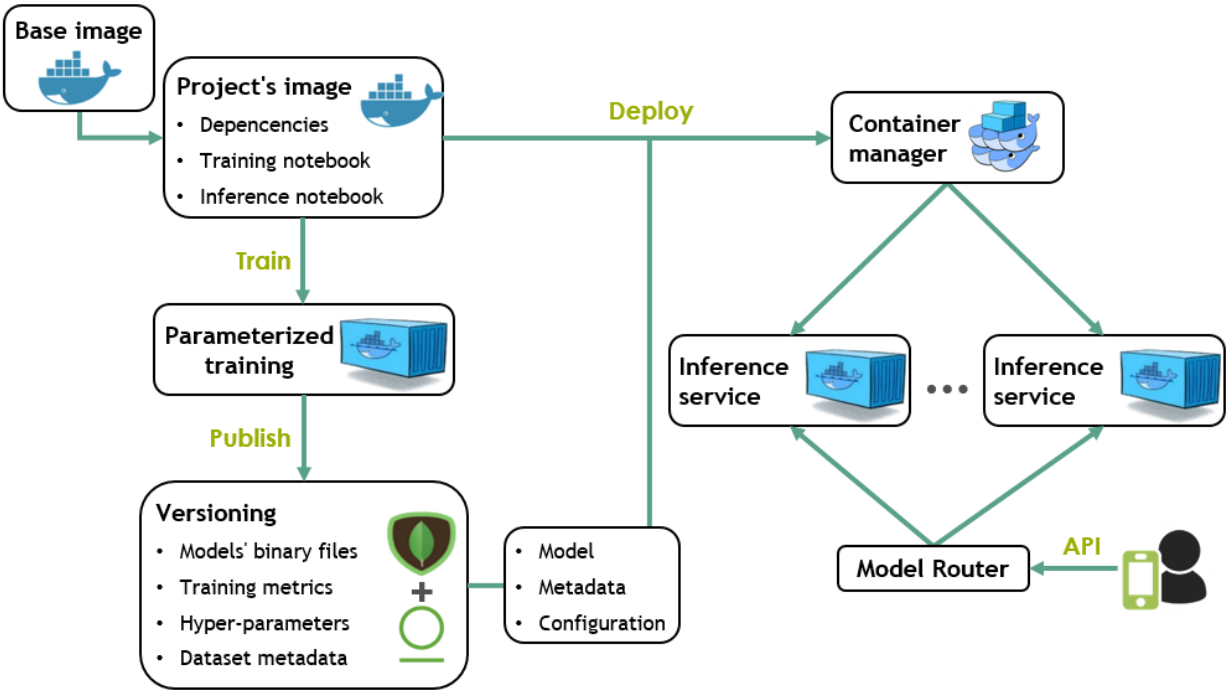
1.1 What's this?

Noronha is a framework that hosts Machine Learning projects inside a portable, ready-to-use DataOps architecture. The goal here is to help Data Scientists benefit from DataOps practices without having to change much of their usual work behavior.

1.2 Overview

The following steps and the diagram bellow describe together the basic training and deploying workflow of a Machine Learning project inside Noronha:

1. Noronha's base image is used as a starting point to provide the tools a project needs to run inside the framework.
2. The project is packed in a Docker image with its dependencies, a training notebook and a notebook for inference.
3. Every training is a parameterized execution of the training notebook, inside a container of the project's image.
4. Every model version produced is published to a versioning system in which MongoDB stores metadata and a file manager like Artifactory stores raw files and binaries.
5. When deploying a model, containers of the project's image are created for running the inference notebook as a service. Every asset necessary is injected into the containers.



1.3 Pre-requisites

To use Noronha in its most basic configuration all you need is:

- Any recent, stable Unix OS.
- Docker v17+ with Swarm mode enabled and configured to be used without sudo.
- A Conda v4.5+ environment with Python v3.5+.
- Git v2+.

For a more advanced usage of the framework, see the *configuration guide*.

1.4 Installation

You can easily install Noronha by activating your Conda environment and running the following commands:

```
pip install noronha-dataops

nha get-me-started
```

This assumes you're going to use the default plugins (MongoDB and Artifactory) in native mode (auto-generated instances). To use plugins differently, see the *configuration guide*.

1.5 Basic usage

Let's start with the simplest project structure:

```
project_home:
+-- Dockerfile
+-- requirements.txt
```

This is what the Dockerfile may look like:

```
# default public base image for working inside Noronha
FROM noronhadataops/noronha:latest

# project dependencies installation
ADD requirements.txt .
RUN bash -c "source ${CONDA_HOME}/bin/activate ${CONDA_VENV}  && conda install --file␣
↪requirements.txt"

# deploying the project's code
ADD . ${APP_HOME}
```

Now record your project's metadata and build it:

```
nha proj new --name my-proj
nha proj build --tag develop
```

Then, run the Jupyter Notebook interface inside your project's image for editing and testing code:

```
nha note --edit --tag develop
```

For fully-working project templates and end-to-end tutorials, see the examples directory.

CHAPTER

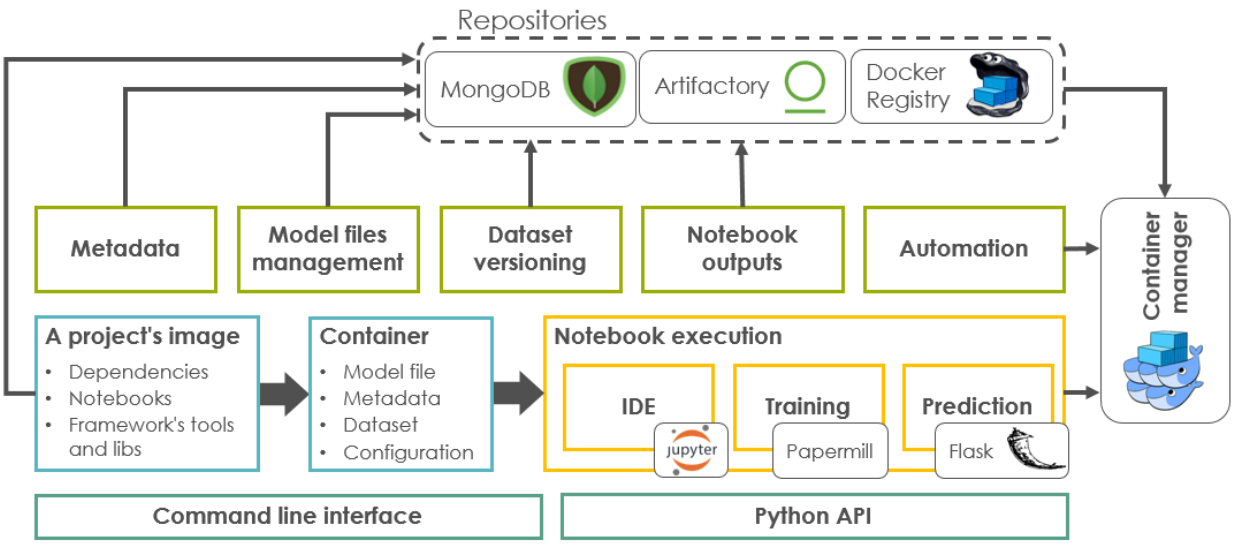
TWO

USER GUIDE

2.1 Key Concepts

This section describes some important concepts that usually come up when working with Noronha. It's recommended that you read and understand them in order to use the framework correctly.

The image bellow summarizes the components that compose the framework:



2.1.1 Project Repositories

When recording a project in Noronha, three kinds of repositories are supported. None of them are mandatory, but at least one is recommended.

- **Home:** Local directory where the project is hosted (recommended when working in a sandbox).
- **Git:** The project's remote Git repository (recommended for production ready projects).
- **Docker:** The project's remote Docker repository (recommended for mock-ups, prototyping and third party image testing).

For the first two kinds (local, git) the framework assumes that your project's root contains a Dockerfile that uses noronhadataops/noronha:latest as its base image. Everytime you build your project with Noronha from one of these two repositories, it's going create a Docker image for the project and record some metadata related to it.

As for the third kind of repository (Docker), the framework assumes your repository contains a pre-built image, managed by the user or a third-party. When building from this repository, Noronha is just going to pull and tag the image for usage in the project's tasks. Besides, if you try to run a project task with a Docker tag that hasn't been recorded by Noronha yet, its default behaviour is to try to find an image with that tag in the project's Docker repository.

This is a common Dockerfile to be used with Noronha:

```
FROM noronhadataops/noronha:latest

ADD requirements.txt .
RUN bash -c "source ${CONDA_HOME}/bin/activate ${CONDA_VENV} \
 && pip install -r requirements.txt \
 && rm -f requirements.txt"

ADD . ${APP_HOME}
```

You can also find other common Dockerfile structures in the examples section.

For the third kind of repository (docker) the framework assumes that the image is already managed by the user or a third-party, thus it's ready to use and no builds will be performed by the framework itself.

2.1.2 Containers and Notebooks

Why do we build projects into Docker images? That's because in Noronha every project task is basically a notebook execution that happens inside a container, and the project's image is used to create those containers:

- **IDE:** Jupyter Notebook inside a container, for editing and testing your code in an interactive manner.
- **Train:** Model training inside a container. It's basically an execution of the training notebook. Usually, by the end of the execution a model version is produced.
- **Deploy:** Prediction service inside a group of containers with one or more replicas. Here, the prediction notebook is used to sustain an HTTP endpoint for serving prediction requests.

You can find templates for structuring your training and prediction notebooks in the examples section.

2.1.3 Islands (Plugins)

There is also a fourth kind of container that we call an "island". This is like an embedded plugin, a container which is created and managed by the framework itself and that is responsible for performing some useful task. In this case, the image used to create the container depends on the task its meant to perform.

These are the default islands:

- **MongoDB:** Database for storing metadata related to everything in this framework (projects, models, build versions, etc). See the *data model section* for further info.
- **Artifactory:** File manager for storing model files, small datasets and notebook execution outputs.

These are the optional islands:

- **Router:** simple Node.js proxy that routes prediction requests to the respective model deployments.
- **Nexus:** alternative file manager that you can use instead of Artifactory.

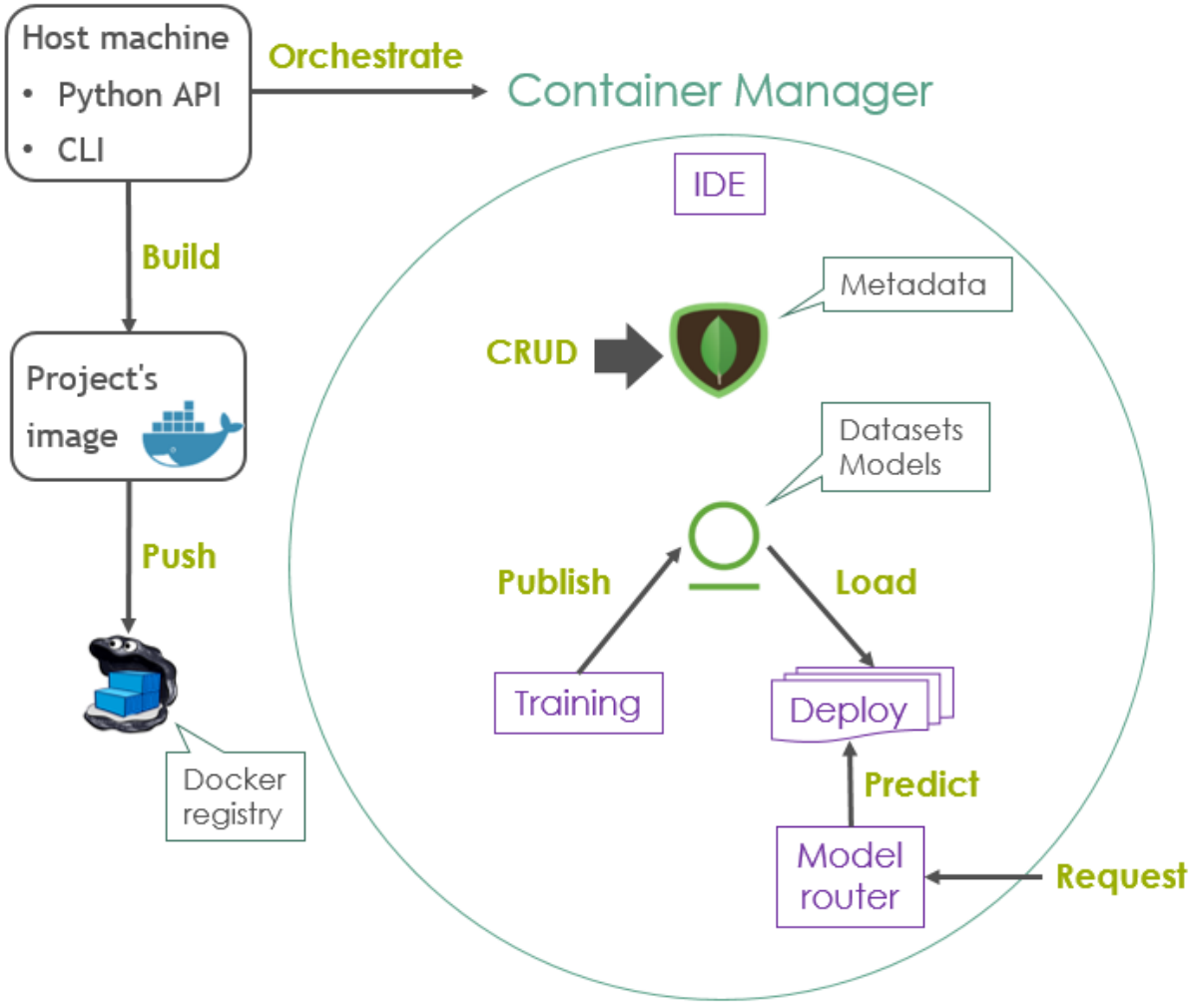
Instead of using a native (embedded) island, the file management and metadata storage tasks can also be performed by foreign (external, dedicated) instances, managed by the user or a third party (e.g.: a huge Nexus server in you company's cloud or even a MongoDB cluster). To setup Noronha in this kind of environment see the *configuration guide*.

2.1.4 Orchestration

Noronha relies on a container manager to manipulate its containers. This can be either of:

- **Docker Swarm:** This is the default container manager, and it's meant for experimenting with the framework in a local, non-distributed environment.
- **Kubernetes:** This is the recommended container manager for working with multiple projects and users in a real productive cluster. To configure Noronha to use Kubernetes as its container manager, see the *configuration guide*.

The image below illustrates how the framework components interact inside and outside the container manager. However, this is a simplified representation, since in reality multiple trainings and/or deployments belonging to one or more projects may coexist in the same environment.

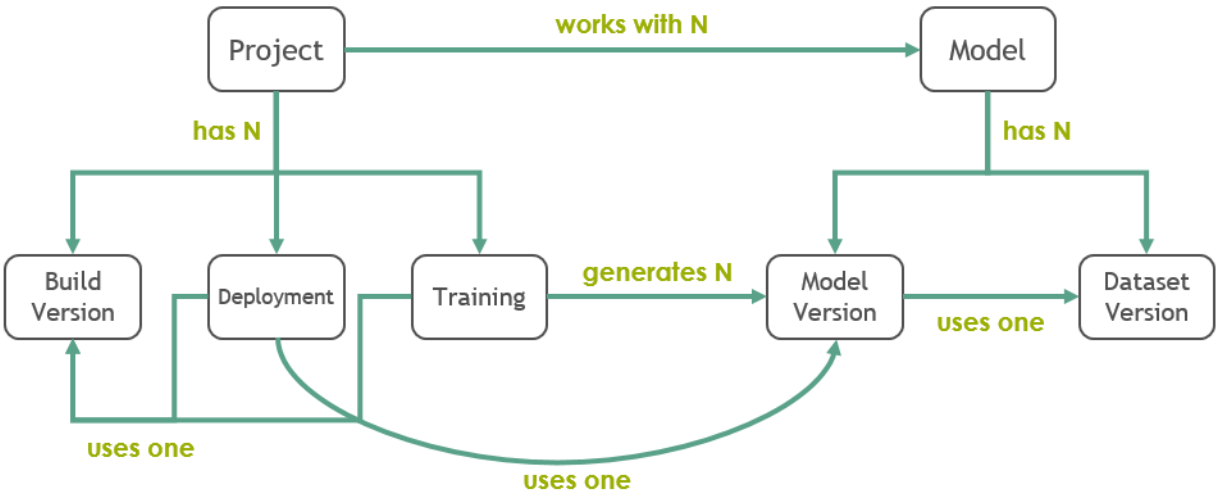


Note that the *host machine* may be a local computer or even one of the servers that compose the container manager's node pool. The framework's libraries also need to be present in this host so that it can use Noronha's API to interact with the container manager. Usually, a host like this is referred to as *off board*, since it's not running inside a container, whereas IDE, training and deployments are referred to as *on board*.

2.2 Noronha's Data Model

This section describes how Noronha stores its metadata in MongoDB and how these metadata documents relate to each other. Reading and understanding this section is going to help you when creating and manipulating projects, models and other objects in Noronha.

The diagram bellow gives a hint on the document relationships described in detail here:



This guide adopts the following conventions for representing document fields that link to other documents:

- **Referenced document:** fields in **bold** are like pointers. Their content is always consistent with the original document they refer to. Fields like these are meant to answer questions like: which model is my project using? Which model files are expected?
- *Embedded document:* fields in *italic* are like snapshots. Their content is a report of how the referred document was when the field was updated. Fields like these are meant to answer questions like: which version of my project's code was used in that training?

2.2.1 Project

Represents a project that is managed by the framework. Also referred to as *proj*.

```
{
    name: name-of-the-project  # only alphanumerical and dashes
    desc: free text description
    model: list of models used by this project
    home_dir: local directory where the project is hosted
    git_repo: the project's remote Git repository
    docker_repo: the project's remote Docker repository
    # see project repositories
}
```

2.2.2 Build Version

Represents the Docker image that was created when the project was built by Noronha. Also referred to as *bvers* or *bv* (not to mistake for *beavers* :D).

```
{
    tag: Docker tag
    proj: the project which was built
    docker_id: the Docker hash associated to the image that was created
    git_version: the Git hash associated to the last commit before the project was built
    built_at: date and time when it was built
    built_from: either 'local', 'git' or 'pre-built' (determined by the build command)
}
```

2.2.3 Model

Represents a model that is managed by the framework.

```
{
    name: name-of-the-model  # only alphanumerical and dashes
    desc: free text description
    model_files: list of file docs. These files compose the model's persistence
    data_files:  list of file docs. These files compose a dataset for training the model
}

# btw, this is how a file doc is defined:
{
    name: file.extension
    desc: free text description
    required: if true, this file can never be left out
    max_mb: maximum file size in MB. Not necessary, but good to know
}
```

Note that this is **not** a model *version*, but a model *definition*: it's like a template that describes how a model is going to be persisted. Of course, when starting project we usually have no clue of how the model is going to be, but don't worry: all properties except the model's name can be edited later.

2.2.4 Dataset

Represents a dataset that is managed by the framework. Also referred to as *ds* (not a data scientist though :D).

```
{
    name: name-of-the-dataset  # only alphanumerical and dashes
    model: the model to which this dataset belongs
    stored: if true, the dataset files are stored in Noronha's file manager
    details: dictionary with arbitrary details about the dataset
    compressed: if true, all dataset files are compressed into a single tar.gz file
    lightweight: if true, the dataset files are stored in a lightweight file storage

}
```

2.2.5 Training

Represents the execution of a training. Also referred to as *train* (not the one that runs on *rails* :D).

```
{
    name: name-of-the-training  # only alphanumerical and dashes
    proj: the project responsible for this training
    bvers: the build version that was used for running this training
    notebook: relative path inside the project's repository to the training notebook␣
↪that was executed
    task: task doc. Represents the training's progress and state
    details: dictionary with arbitrary details about the training
}

# btw, this is how a task doc is defined:
{
    state: either one of WAITING, RUNNING, FINISHED, FAILED, CANCELLED
    progress: number between 0 and 1
    start_time: when the task started
    update_time: when the task's state and/or progress was updated
}
```

2.2.6 Model Version

Represents a persistent model that was generated during a training. Also referred to as *movers* or *mv*.

```
{
    name: name-of-the-version  # only alphanumerical and dashes
    model: the parent model definition (template) that shapes this version
    train: the training execution that generated this version
    ds: the dataset that was used for training the model
    details: dictionary with arbitrary details about the version
    pretrained: reference to another model version that was used as a pre-trained asset␣
↪in order to train this one
    compressed: if true, all model files are compressed into a single tar.gz file
    lightweight: if true, the model files are stored in a lightweight file storage
}
```

2.2.7 Deployment

Represents a group of one or more identical containers providing a prediction service. Also referred to as *depl*.

```
{
    name: name-of-the-deployment  # only alphanumerical and dashes
    proj: the project to which this deployment belongs
    movers: the model version used in this deployment
    bvers: the build version (docker image) used for creating this deployment's␣
↪containers
    notebook: relative path inside the project's repository to the prediction notebook␣
↪that is executed
    details: dictionary with arbitrary details about the deployment
}
```

2.2.8 Treasure Chest

Represents a pair of credentials recorded and stored securely in the framework. Also referred to as *tchest*.

```
{
    name: name-of-the-tchest  # only alphanumerical and dashes
    owner: os-user-to-whom-it-belongs
    desc: free text description
    details: dictionary with arbitrary details about the tchest
}
```

2.3 Configuring Noronha

2.3.1 Configuration Files

Noronha's default configuration file is packaged together with its Python libraries, under the resources directory. It's a YAML file in which the top keys organize properties according to the subjects they refer to.

```
---
project:
  working_project: null

logger:
  level: INFO
  pretty: false
  join_root: true
  max_bytes: 1048576  # 1mb
  bkp_count: 1

mongo:
  native: true
  port: 30017
  database: nha_db
  write_concern:
    w: 1
    j: true
    wtimeout: 5

router:
  native: true
  port: 30080

file_store:
  native: true
  port: 30023
  type: artif  # (artif, nexus)

lightweight_store:
  enabled: false
  native: false
  type: cass
```

(continues on next page)

(continued from previous page)

```
  hosts: ['cassandra']
  port: 30042
  keyspace: nha_db
  replication_factor: 3

docker:
  daemon_address: unix:/var/run/docker.sock
  target_registry: null
  registry_secret: null

container_manager:
  type: swarm
  resource_profiles:
    nha-gpu:
      enable_gpu: false
      requests:
        memory: 512
        cpu: 1
      limits:
        memory: 2048
        cpu: 1

web_server:
  type: simple
  enable_debug: false
```

This configuration can be extended by placing a *nha.yaml* file with the desired keys in the current working directory or in the user's home directory at *~/.nha/*. The file resolution is as follows:

- **./nha.yaml**: if present, this file will be used to extend the default configuration. No other files will be looked for.
- **~/.nha/nha.yaml**: if the previous alternative wasn't available, this file will be used instead.
- If none of the alternatives above was available, only the default configuration is going to be used.

2.3.2 Conventions for Islands

The following properties are common for all *plugins*.

- **native:** (boolean) If true, this plugin runs inside a container manager by Noronha. Otherwise, this plugin runs in a dedicated server, managed by the user or a third-party. The later option is referred to as *foreign mode*, in opposition to the *native mode* (default: true).
- **host:** This property is only used in *foreign mode*. It refers to the hostname or IP of the server in which Noronha is going to find the service (e.g.: MongoDB's hostname or IP, as it appears in its connection string).
- **port:** In *foreign mode*, this refers to the port in which the plugin is exposed (e.g.: MongoDB's port, as it appears in its connection string). In *native mode*, this refers to the server port in which Noronha is going to expose the plugin. Note that if your container manager is Kubernetes only the ports between 30000 and 31000 are available.
- **user:** Username for authenticating in the plugin (*foreign mode* only).
- **pswd:** Password for authenticating in the plugin (*foreign mode* only).
- **tchest:** Instead of specifying credentials explicitly, you may set this property with the name of a *Treasure Chest* that holds your pre-recorded credentials.

- **disk_allocation_mb:** This property is only used in *native mode*. When Noronha creates a volume to store the plugin's data, it's going to ask the container manager for this amount of storage, in megabytes.

The following topics describe the properties under each configuration subject (top keys in the YAML file).

2.3.3 Router

The following properties are found under the key *router* and they refer to how Noronha uses its model router.

- **port:** As explained in the *island conventions* (default: 30080).

2.3.4 MongoDB

The following properties are found under the key *mongo* and they refer to how Noronha uses its database.

- **port:** As explained in the *island conventions* (default: 30017).
- **database:** Name of the database that Noronha is going to access in MongoDB. Created in runtime if not existing (default: nha_db).
- **write_concern:** Dictionary with the concern options that Noronha should use when writing to the database, as in MongoDB's manual. The following example represents the default values for this property:

```
write_concern:
  w: 1
  j: true
  wtimeout: 5
```

2.3.5 File Manager

The following properties are found under the key *file_store* and they refer to how Noronha uses its file manager.

- **port:** As explained in the *island conventions* (default: 30023).
- **use_ssl:** (boolean) Set to true if your file manager server uses https (*foreign mode* only) (default: false).
- **check_certificate:** (boolean) When using SSL encryption, you may set this to false in order to skip the verification of your server's certificate, although this is not recommended (*foreign mode* only) (default: true).
- **type:** Reference to the file manager that Noronha should use (either *artif*, for Artifactory, or *nexus*, for Nexus) (default: artif).
- **repository:** Name of an existing repository that Noronha should use to store its model files, datasets and output notebooks. For Artifactory, the default is *example-repo-local*. For Nexus there is no default value, since the first repository needs to be created manually through the plugin's user interface.

2.3.6 Lightweight Store

The following properties are found under the key *lightweight_store* and they refer to how Noronha uses its lightweight file storage.

This is a storage alternative to be used along with the standard file manager, so that small datasets and model versions can be persisted and restored faster. This feature is specially useful when your prediction notebook uses a *LazyModelServer*.

Note that in order to use this feature you should have already configured an external Cassandra Database. An easy way to experiment with it in a sandbox environment is to use a dockerized Cassandra.

- **enabled:** (boolean) Set to *true* if you're going to use this feature (default: false).
- **native:** As explained in the *island conventions*. Currently, the only supported value is *false*.
- **type:** The type of database. Currently, the only supported value is *cass*.
- **port:** The database's communication port (default: 9042).
- **hosts:** List of hostnames or IP's to connect with your database.

2.3.7 Project

The following properties are found under the key *project* and they refer to how Noronha handle's your project.

- **working_project:** this tells the framework which project you are working on right now. This is important because many features such as training or deploying models can only be performed inside the scope of a project. However, before looking into this property the framework checks two other alternatives: was a project name provided as argument to the function? Is the current working directory a local repository for a project?

2.3.8 Logger

The following properties are found under the key *logger* and they refer to how Noronha logs messages.

- **level:** Log verbosity level, as in Python's logging (one of: ERROR, WARN, INFO, DEBUG) (default: INFO).
- **pretty:** (boolean) If true, all dictionaries and exception objects are pretty-printed (default: false).
- **directory:** Path to the directory where log files hould be kept (default: ~/.nha/logs/)
- **file_name:** Log file name (default: noronha.log)
- **max_bytes:** Max log file size, in bytes (default: 1mb).
- **bkp_count:** Number of log file backups to be kept (default: 1).
- **join_root:** (boolean) If true, log messages by other frameworks such as Flask and Conu are also dumped to Noronha's log file.

2.3.9 Docker

The following properties are found under the key *docker* and they refer to how Noronha uses the Docker engine.

- **daemon_address:** Path or address to Docker daemon's socket (default: unix:/var/run/docker.sock).
- **target_registry:** Address of the Docker registry to which the images built by the framework will be uploaded (default is null, so images are kept locally).

The following parameters are only used if the chosen container manager is Kubernetes:

- **registry_secret:** Name of the Kubernetes secret that your cluster uses to access the registry configured in the previously. This property is recommended if your containers fail with the message *ImagePullBackOff*.

2.3.10 Container Manager

The following properties are found under the key *container_manager* and they refer to how Noronha uses the container manager.

- **type:** Reference to the container manager that Noronha should use as its backend (either *swarm*, for Docker Swarm, or *kube*, for Kubernetes) (default: swarm).
- **api_timeout:** The maximum time, in seconds, to wait before the container manager completes a requested action (default: 20 for Docker Swarm, 60 for Kubernetes).
- **resource_profiles:** A mapping in which the keys are resource profile names and the values are resource specifications. Example:

```
light_training:
  requests:
    memory: 256
    cpu: 1
  limits:
    memory: 512
    cpu: 2

heavy_training:
  requests:
    memory: 2048
    cpu: 2
  limits:
    memory: 4096
    cpu: 4

# cpu values are expressed in vCores and are expected to be integer, float or string.␣
↪Precion lower than 0.001 is not allowed
# memory values are expressed in MB and are expected to be integer.
```

Such resource profile names may be specified when starting an IDE, training or deployment (note that when deploying with multiple replicas, the resources specification will be applied to each replica).

Another interesting strategy is to specify default resource profiles according to the *work section*. The available work sections are *nha-ide*, *nha-train* and *nha-depl*. Those refer to the default resource specifications applied when using the IDE, training or running a deploy, respectively. Example:

```
nha-ide:
  requests:
    memory: 256
    cpu: 1
  limits:
    memory: 512
    cpu: 2

nha-train:
  requests:
    memory: 2048
```

(continues on next page)

(continued from previous page)

```
    cpu: 2
  limits:
    memory: 4096
    cpu: 4
```

Additional configuration may be added to these profiles in order to further customize containers. Here is an example of the possible configuration that a resource profile accepts and implements:

```
nha_secure:
  service_type: ClusterIP

gpu_training:
  enable_gpu: true
  requests:
    memory: 256
    cpu: 1
  limits:
    memory: 512
    cpu: 2

elastic_deploy:
  auto_scale: true  # does not affect Docker Swarm deploy
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
  requests:
    memory: 256
    cpu: 1
  limits:
    memory: 512
    cpu: 2
```

You can change the type of Kubernetes service that Noronha creates using the *service_type* keyword. Accepted values are: ClusterIP, NodePort, LoadBalancer

GPU support is added through the *enable_gpu* keyword. Currently, Noronha does not support ID-specific GPU assignment or multiple GPUs per Pod.

Kubernetes Horizontal Pod Autoscaling (HPA) support is added through the *auto_scale* keyword. If no additional keys are specified, the default values from the code above are used. Currently there is only support for CPU-based autoscaling.

- **healthcheck:** A mapping that describes how the container manager is going to probe each container's liveness and readiness in a deployment. The values in the following example are the default ones:

```
healthcheck:
  enabled: false  # whether to apply healthchecks or not
  start_period: 60  # seconds before healthchecks start
  interval: 30  # seconds between each healtchcheck
  timeout: 3  # seconds for the container to respond to a healthcheck
  retries: 3  # number of consecutive healthcheck failures for a container to be force-
↪restarted
```

The following parameters are only used if the chosen container manager is Kubernetes:

- **namespace:** An existing Kubernetes namespace in which Noronha will create its resources (default: default).
- **storage_class:** An existing storage class that Noronha will use to create persistent volume claims for storing its plugins' data (default: standard).
- **nfs:** A mapping with the keys *path* and *server*. The key *server* should point to your NFS server's hostname or IP, whereas *path* refers to an existing directory inside your NFS server. Noronha will create volumes under the specified directory for sharing files with its training, deployment and IDE containers.

2.3.11 WebServer

The following properties are found under the key *web_server* and they refer to how Noronha configures your inference service. These can be overriden when you instanciate a ModelServer in your predict notebook.

- **type:** defines which server you want to use. Current supported options are: *simple* and *gunicorn*.
- **enable_debug:** this option is used to set a debug mode for your server.
- **threads:** dictionary with keys: *enabled* to enable multi-thread, *high_cpu* to set a higher thread count and *number* to set a specific thread count, which overrides *high_cpu*.

```
threads:
  enabled: false
  high_cpu: false
  number: 1
```

- **extra_conf:** dictionary with keys that may vary depending on your server. For Gunicorn configuration options, please refer to: Gunicorn manual

Below is a complete example of *web_server* configuration:

```
web_server:
  type: gunicorn
  enable_debug: false
  threads:
    enabled: true
    high_cpu: true
  extra_conf:
    workers: 1
```

CHAPTER

THREE

REFERENCE

3.1 CLI Reference

This section describes the usage of Noronha's command line interface. Each topic in this section refers to a different API subject such as projects, models and so on.

3.1.1 General

The entrypoint for Noronha's CLI is either the keyword *noronha*, for being explicit, or the alias *nha*, for shortness and cuteness. You can always check which commands are available with the *help* option:

```
nha --help  # overview of all CLI subjects
nha proj --help  # describe commands under the subject *proj*
nha proj new --help  # details about the command *new* under the subject *proj*
```

Note that the Conda environment in which you *installed* Noronha needs to be activated so that this entrypoint is accessible. Besides, we assume these commands are executed from the *host machine*.

The entrypoint also accepts the following flags and options for customizing a command's output:

```
-l, --log-level TEXT  Level of log verbosity (DEBUG, INFO, WARN, ERROR)
-d, --debug           Set log level to DEBUG
-p, --pretty          Less compact, more readable output
-s, --skip-questions  Skip questions
-b, --background      Run in background, only log to files
```

Usage example for skipping questions in background and keeping only pretty warning messages in the log files:

```
nha --background --skip-questions --log-level WARN --pretty proj list
nha -b -s -l WARN -p proj list  # same command, shorter version with aliases
```

The default directory for log files is **~/.nha/logs**. For further log configuration options see the *log configuration section*.

There's also a special command for *newbies*, that's accesible directly from the entrypoint:

```
nha get-me-started
```

As stated in the *introduction*, this is going to configure the basic *plugins* in native mode automatically. This means that after running this command your *container manager* is going to be running a MongoDB service for storing Noronha's *metadata* and an Artifactory service for managing Noronha's files. This is useful if you are just experimenting with the framework and do not want to spend time customizing anything yet.

3.1.2 Project

Reference for commands under the subject *proj*.

- **info:** information about a project

```
--proj, --name    Name of the project (default: current working project)
```

- **list:** list hosted projects

```
-f, --filter    Query in MongoDB's JSON syntax
-e, --expand    Flag: expand each record's fields
-m, --model     Only projects that use this model will be listed
```

- **rm:** remove a project and everything related to it

```
--proj, --name    Name of the project (default: current working project)
```

- **new:** host a new project in the framework

```
-n, --name        Name of the project
-d, --desc        Free text description
-m, --model       Name of an existing model (further info: nha model --help)
--home-dir        Local directory where the project is hosted.
                  Example: /path/to/proj
--git-repo        The project's remote Git repository.
                  Example: https://<git_server>/<proj_repo>
--docker-repo     The project's remote Docker repository.
                  Example: <docker_registry>/<proj_image>
```

- **update:** update a projects in the database

```
-n, --name        Name of the project you want to update (default: current working␣
↪project)
-d, --desc        Free text description
-m, --model       Name of an existing model (further info: nha model --help)
--home-dir        Local directory where the project is hosted.
                  Example: /path/to/proj
--git-repo        The project's remote Git repository.
                  Example: https://<git_server>/<proj_repo>
--docker-repo     The project's remote Docker repository.
                  Example: <docker_registry>/<proj_image>
```

- **build:** encapsulate the project in a new Docker image

```
--proj          Name of the project (default: current working project)
-t, --tag       Docker tag for the image (default: latest)
--no-cache      Flag: slower build, but useful when the cached layers contain outdated␣
↪information
--from-here     Flag: build from current working directory (default option)
--from-home     Flag: build from project's home directory
--from-git      Flag: build from project's Git repository (master branch)
--pre-built     Flag: don't build, just pull and tag a pre-built image from project's␣
↪Docker repository
```

3.1.3 Build Version

Reference for commands under the subject *bvers*.

- **info:** information about a build version

```
--proj    The project to which this build version belongs (default: current working␣
↪project)
--tag     The build version's docker tag (default: latest)
```

- **list:** list build versions

```
--proj          The project whose versions you want to list (default: current working␣
↪project)
-f, --filter    Query in MongoDB's JSON syntax
-e, --expand    Flag: expand each record's fields
```

- **rm:** remove a build version

```
--proj    The project in which this version belongs (default: current working project)
--tag     The version's docker tag (default: latest)
```

3.1.4 Model

Reference for commands under the subject *model*.

- **info:** information about a model

```
--name    Name of the model
```

- **list:** list model records

```
-f, --filter    Query in MongoDB's JSON syntax
-e, --expand    Flag: expand each record's fields
```

- **rm:** remove a model along with all of it's versions and datasets

```
-n, --name    Name of the model
```

- **new:** record a new model in the database

```
-n, --name      Name of the model
-d, --desc      Free text description
--model-file    JSON describing a file that is used for saving/loading this model.
                Example:
                {"name": "categories.pkl", "desc": "Pickle with DataFrame for looking up␣
↪prediction labels", "required": true, "max_mb": 64}
--data-file     JSON describing a file that is used for training this model.
                Example:
                {"name": "intents.csv", "desc": "CSV file with examples for each user␣
↪intent", "required": true, "max_mb": 128}
```

- **update:** update a model record

```
-n, --name          Name of the model you want to update
-d, --desc          Free text description
--model-file        JSON describing a file that is used for saving/loading this model.
                    Example:
                    {"name": "categories.pkl", "desc": "Pickle with DataFrame for␣
↪looking up prediction labels", "required": true, "max_mb": 64}
--data-file         JSON describing a file that is used for training this model.
                    Example:
                    {"name": "intents.csv", "desc": "CSV file with examples for each␣
↪user intent", "required": true, "max_mb": 128}
--no-model-files    Flag: disable the tracking of model files
--no-ds-files       Flag: disable the tracking of dataset files
```

3.1.5 Dataset

Reference for commands under the subject *ds*.

- **info:** information about a dataset

```
--model    Name of the model to which this dataset belongs
--name     Name of the dataset
```

- **list:** list datasets

```
-f, --filter    Query in MongoDB's JSON syntax
-e, --expand    Flag: expand each record's fields
--model         Only datasets that belong to this model will be listed
```

- **rm:** remove a dataset and all of its files

```
--model    Name of the model to which this dataset belongs
--name     Name of the dataset
```

- **new:** add a new dataset

```
-n, --name      Name of the dataset (defaults to a random name)
-m, --model     The model to which this dataset belongs (further info: nha model --help)
-d, --details   JSON with any details related to the dataset
-p, --path      Path to the directory that contains the dataset files (default: current␣
↪working directory)
-c, --compress  Flag: compress all dataset files to a single tar.gz archive
--skip-upload   Flag: don't upload any files, just record metadata
--lightweight   Flag: use lightweight storage
```

- **update:** update a dataset's details or files

```
-n, --name       Name of the dataset you want to update
-m, --model      The model to which this dataset belongs (further info: nha model --help)
-d, --details    JSON with details related to the dataset
-p, --path       Path to the directory that contains the dataset files (default: current␣
↪working directory)
```

3.1.6 Training

Reference for commands under the subject *train*.

- **info:** information about a training execution

```
--name    Name of the training
--proj    Name of the project responsible for this training (default: current working␣
↪project)
```

- **list:** list training executions

```
-f, --filter    Query in MongoDB's JSON syntax
-e, --expand    Flag: expand each record's fields
--proj          Name of the project responsible for the trainings (default: current␣
↪working project)
```

- **rm:** remove a training's metadata

```
--name    Name of the training
--proj    Name of the project responsible for this training (default: current working␣
↪project)
```

- **new:** execute a new training

```
--name                Name of the training (defaults to a random name)
--proj                Name of the project responsible for this training (default:␣
↪current working project)
--notebook, --nb      Relative path, inside the project's directory
                      structure, to the notebook that will be executed
-p, --params          JSON with parameters to be injected in the notebook
-t, --tag             The training runs on top of a Docker image that
                      belongs to the project. You may specify the image's
                      Docker tag or let it default to "latest"
-e, --env-var         Environment variable in the form KEY=VALUE
-m, --mount           A host path or docker volume to mount on the training container.
                      Syntax: <host_path_or_volume_name>:<container_path>:<rw/ro>
                      Example: /home/user/data:/data:rw
--dataset, --ds       Reference to a dataset to be mounted on the training container.
                      Syntax: <model_name>:<dataset_name>
                      Example: iris-clf:iris-data-v0
--pretrained          Reference to a model version that will be used as a pre-trained␣
↪model during this training.
                      Syntax: <model_name>:<version_name>
                      Example: word2vec:en-us-v1
--resource-profile    Name of a resource profile to be applied for each container.
                      This profile should be configured in your nha.yaml file
```

3.1.7 Model Version

Reference for commands under the subject *movers*.

- **info:** information about a model version

```
--model    Name of the model to which this version belongs
--name     Name of the version
```

- **list:** list model versions

```
-f, --filter    Query in MongoDB's JSON syntax
-e, --expand    Flag: expand each record's fields
--model         Only versions of this model will be listed
--dataset       Only versions trained with this dataset will be listed
--train         Only model versions produced by this training will be listed
--proj          To be used along with 'train': name of the project to which this␣
↪training belongs
```

- **rm:** remove a model version and all of its files

```
--model    Name of the model to which this version belongs
--name     Name of the version
```

- **new:** record a new model version in the framework

```
-n, --name       Name of the version (defaults to a random name)
-m, --model      The model to which this version belongs (further info: nha model --help)
-d, --details    JSON with details related to the model version
-p, --path       Path to the directory that contains the model files (default: current␣
↪working directory)
--dataset        Name of the dataset that trained this model version
--train          Name of the training that produced this model version
--proj           To be used along with 'train': name of the project to
                 which this training belongs
--pretrained     Reference to another model version that was used as a pre-trained asset␣
↪for training this one.
                 Syntax: <model_name>:<model_version>
                 Example: word2vec:en-us-v1
-c, --compress   Flag: compress all model files to a single tar.gz archive
--skip-upload    Flag: don't upload any files, just record metadata
--lightweight    Flag: use lightweight storage
```

- **update:** update a model version's details or files

```
-n, --name       Name of the model version you want to update
-m, --model      The model to which this version belongs (further info: nha model --help)
-d, --details    JSON with details related to the version
-p, --path       Path to the directory that contains the model files (default: current␣
↪working directory)
--dataset        Name of the dataset that trained this model version
--train          Name of the training that produced this model version
--proj           To be used along with 'train': name of the project to which this␣
↪training belongs
```

3.1.8 Deployment

Reference for commands under the subject *depl*.

- **info:** information about a deployment

```
--name    Name of the deployment
--proj    Name of the project responsible for this deployment (default: current working␣
↪project)
```

- **list:** list deployments

```
-f, --filter    Query in MongoDB's JSON syntax
-e, --expand    Flag: expand each record's fields
--proj          Name of the project responsible for this deployment (default: current␣
↪working project)
```

- **rm:** remove a deployment

```
--name    Name of the deployment
--proj    Name of the project responsible for this deployment (default: current working␣
↪project)
```

- **new:** setup a deployment

```
--name                Name of the deployment (defaults to a random name)
--proj                Name of the project responsible for this deployment (default:␣
↪current working project)
--notebook, --nb      Relative path, inside the project's directory
                      structure, to the notebook that will be executed
--params              JSON with parameters to be injected in the notebook
-t, --tag             Each deployment task runs on top of a Docker image
                      that belongs to the project. You may specify the
                      image's Docker tag or let it default to "latest"
-n, --n-tasks         Number of tasks (containers) for deployment
                      replication (default: 1)
--port                Host port to be routed to each container's inference
                      service
-e, --env-var         Environment variable in the form KEY=VALUE
-m, --mount           A host path or docker volume to mount on each deployment container.
                      Syntax: <host_path_or_volume_name>:<container_path>:<rw/ro>
                      Example: /home/user/data:/data:rw
--movers, --mv        Reference to a model version to be mounted on each deployment␣
↪container.
                      Syntax: <model_name>:<version_name>
                      Example: iris-clf:experiment-v1
--resource-profile    Name of a resource profile to be applied for each container.
                      This profile should be configured in your nha.yaml file
```

3.1.9 Notebook (IDE)

You can start-up a Jupyter notebook interface for your project in order to edit and test your code inside a disposable environment that is much like the environment your code is going to find in production.

- **note:** Access to an interactive notebook (IDE)

```
--proj TEXT           Name of the project you'd like to work with.
-t, --tag             The IDE runs on top of a Docker image that belongs to the current␣
↪working project.
                      You may specify the image's Docker tag or let it default to "latest
↪"
-p, --port            Host port that will be routed to the notebook's user interface␣
↪(default: 30088)
-e, --env-var         Environment variable in the form KEY=VALUE
-m, --mount           A host path or docker volume to mount on the IDE's container.
                      Syntax: <host_path_or_volume_name>:<container_path>:<rw/ro>
                      Example: /home/user/data:/data:rw
--edit                Flag: also mount current directory into the container's /app␣
↪directory.
                      This is useful if you want to edit code, test it and save it in␣
↪the local machine
                      (WARN: in Kubernetes mode this will only work if the current␣
↪directory is part of your NFS server)
--dataset, --ds       Reference to a dataset to be mounted on the IDE's container.
                      Syntax: <model_name>:<dataset_name>
                      Example: iris-clf:iris-data-v0
--movers, --mv        Reference to a model version to be mounted on the IDE's container.
                      Syntax: <model_name>:<version_name>
                      Example: word2vec:en-us-v1:true
--resource-profile    Name of a resource profile to be applied for each container.
                      This profile should be configured in your nha.yaml file
```

3.1.10 Islands (Plugins)

Under the subject *isle* there is a branch of commands for each *plugin*. You can check a plugin's commands with the *help* option:

```
nha isle plugin --help  # overview of this plugin's commands
nha isle plugin command --help  # details about one of this plugin's commands
```

The available *plugins* are:

```
artif   File manager
mongo   Database for metadata
nexus   File manager (alternative)
router  (Optional) Routes requests to deployments
```

The commands bellow are available for all *plugins*, unless stated otherwise:

- **setup:** start and configure this plugin

```
-s, --skip-build    Flag: assume that the required Docker image for setting up
                    this plugin already exists.
```

3.1.11 Treasure Chest

Reference for commands under the subject *tchest*, which are meant to manage *Treasure Chests*.

- **info:** information about a Treasure Chest

```
--name    Name of the Treasure Chest
```

- **list:** list Treasure Chest records

```
-f, --filter    Query in MongoDB's JSON syntax
-e, --expand    Flag: expand each record's fields
```

- **rm:** remove a Treasure Chest

```
-n, --name    Name of the Treasure Chest
```

- **new:** record a new Treasure Chest in the database

```
-n, --name      Name of the Treasure Chest
--desc          Free text description
--details       JSON with any details related to the Treasure Chest
-u, --user      Username to be recorded
-p, --pswd      Password to be recorded
```

- **update:** update a Treasure Chest

```
-n, --name      Name of the Treasure Chest you want to update
--desc          Free text description
--details       JSON with any details related to the Treasure Chest
-u, --user      Username to be recorded
-p, --pswd      Password to be recorded
```

3.2 Python API Reference

3.2.1 Under Construction

3.3 Python Toolkit Reference

This section describes the usage of Noronha's toolkit, which is packed with the base image and meant to be used in any Jupyter Notebook inside your project. The goal of the toolkit is to provide a standard way of performing some common tasks when developing and testing your training and prediction notebooks. This kind of practice can make your notebooks more generic and reusable.

3.3.1 Shortcuts

Reference for functions inside the shortcuts module.

3.3.2 Publish

Reference for the model publisher, which can be found in the publish module.

3.3.3 Serving

Reference for the inference servers, which can be found in the serving module.

CHAPTER

FOUR

DEVELOPER GUIDE

4.1 Contributing to Noronha

The following sections are meant for developers that want to contribute to Noronha by developing new features, fixing bugs and/or improving it in any way.

4.1.1 Before you begin

When contributing to Noronha, it is assumed that you are comfortable with the following technologies:

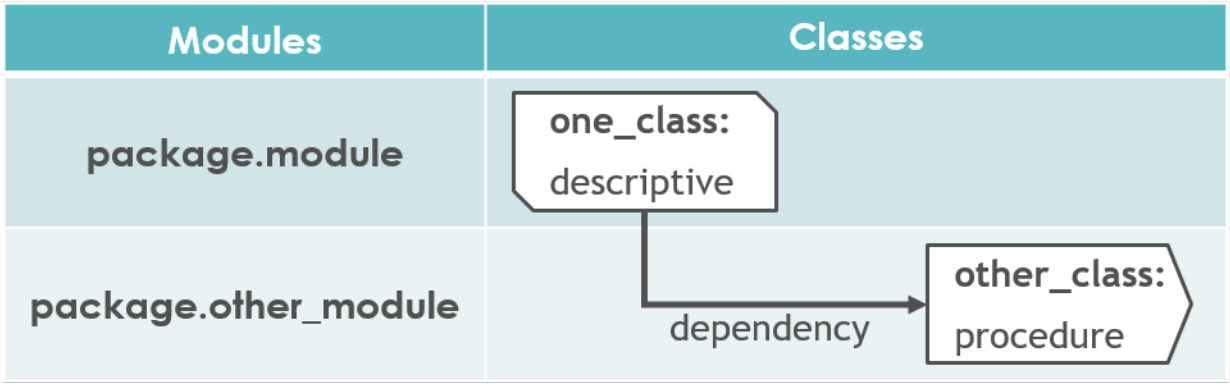
- Supervised Machine Learning
- DevOps pipelines
- Object oriented Python 3
- Docker (engine and Swarm)
- Kubernetes
- MongoDB
- REST API's
- YAML

It is also recommended that you are familiar with Noronha from a user's perspective (i.e.: have read all of the previous sections in *this manual*)

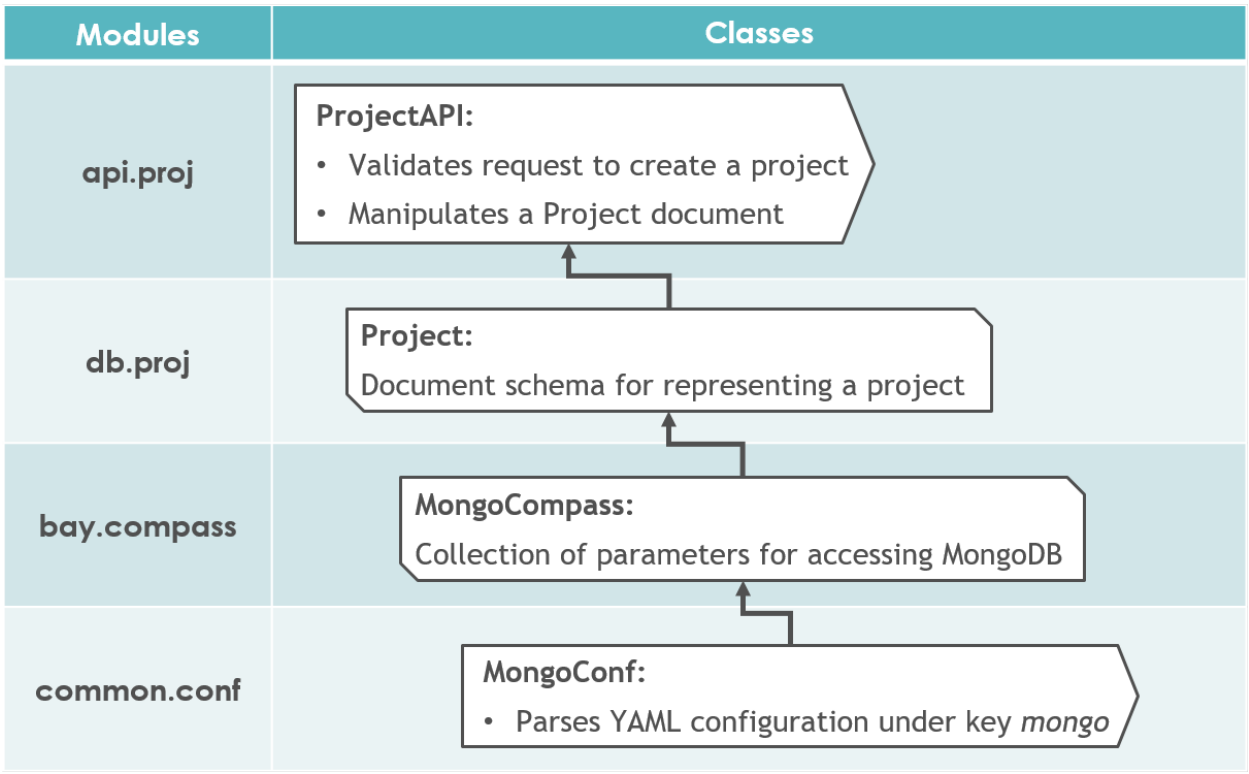
4.2 Modules Relationship

As Noronha performs a task - such as publishing a model or running a training - it relies on several modules that interact in order to produce the expected result. This section explains the relationships that occur between those modules when performing the most common tasks.

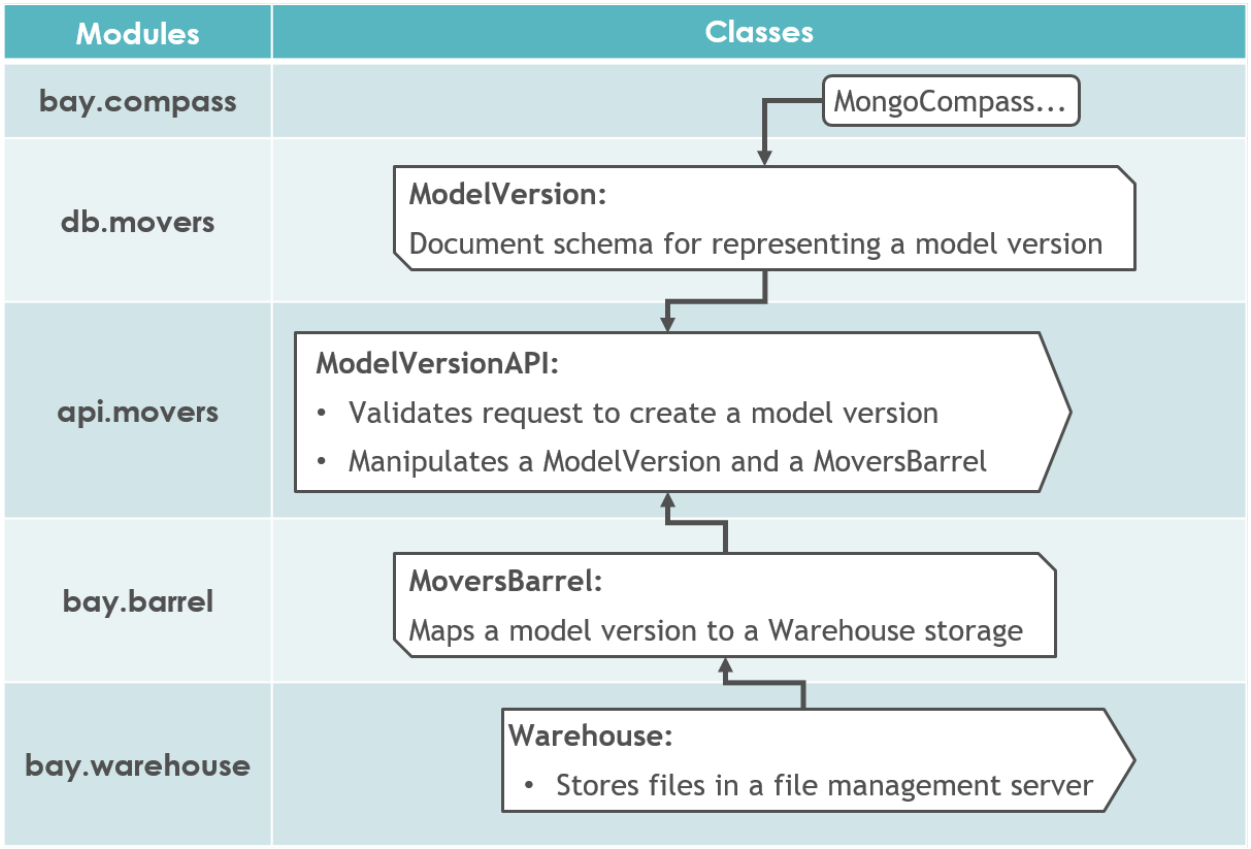
The following caption illustrates the conventions used for representing those relationshipts in the each of the topics bellow:



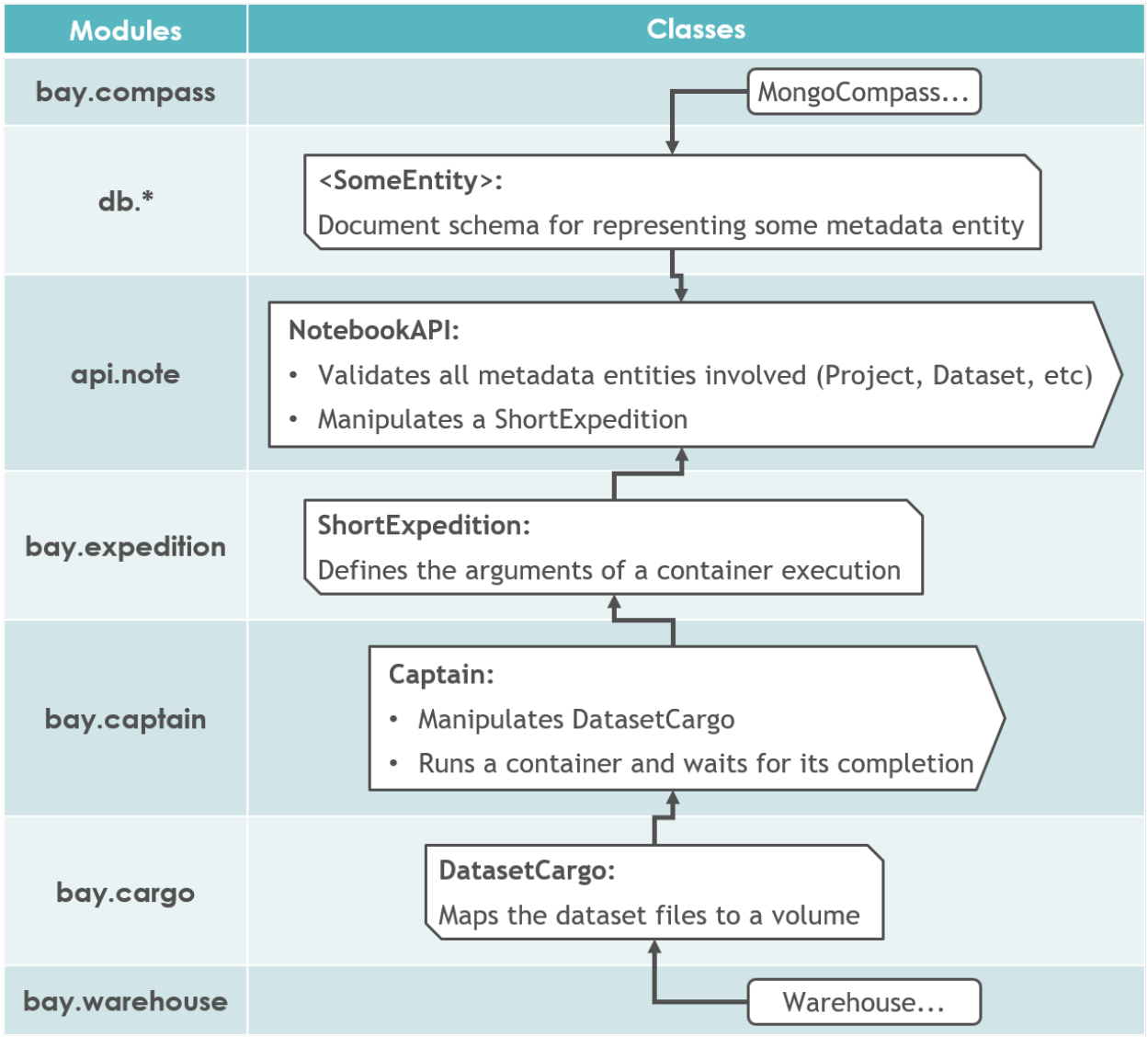
4.2.1 Creating a project



4.2.2 Publishing a model version



4.2.3 Launching the IDE



4.3 Modules Reference

This section summarizes the roles and responsibilities of the most important modules inside Noronha's software architecture.

4.3.1 db

The following topics describe the modules inside the package noronha.db, which is responsible for defining the ORM's for all metadata objects managed by Noronha, as well as utilities for handling those objects.

main.py

utils.py

proj.py

bvers.py

model.py

ds.py

train.py

movers.py

depl.py

tchest.py

4.3.2 bay

The following topics describe the modules inside the package noronha.bay, which provides interfaces that help Noronha interact with other systems such as container managers and file managers. Note that every module inside this package has a nautic/pirate-like thematic.

warehouse.py

barrel.py

cargo.py

captain.py

expedition.py

island.py

compass.py

tchest.py

anchor.py

shipyard.py

CHAPTER

FIVE

PRODUCTION GUIDE

5.1 Deploying Noronha

The following sections intent to show how to install Noronha in production. These instructions are focused on a devops team that will deploy and manage Noronha on a Kubernetes-like cluster.

5.1.1 Requirements

Minimum:

- **Kubernetes cluster (AKS, EKS, GKE, etc.)**
  - 3 nodes (2 vCPUs 8GB RAM)
  - 50 GB HDD Disk
- A container registry
- Noronha compatible machine, with kubectl installed

Recomended:

- **Kubernetes cluster (AKS, EKS, GKE, etc.)**
  - 4 nodes (8 vCPUs 30GB RAM)
  - 250 GB SSD Disk
- A container registry
- Noronha compatible machine, with kubectl installed

5.1.2 Configuring Kubernetes

You can apply all configurations in this section through kubectl:

```
kubectl -n <namespace-id> apply -f <config-file>.yaml
```

It's recomended to create a namespace for Noronha. You can do this by configuring the following script.

```
apiVersion: v1
kind: Namespace
metadata:
    name: <namespace-id>
```

Noronha will also need a service account and the permissions to access the cluster. You can create one with the following script.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: <account-id>
  namespace: <namespace-id>
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: <role-id>
  namespace: <namespace-id>
rules:
- apiGroups: ["", "extensions", "apps", "autoscaling"]
  resources: ["pods", "services", "deployments", "secrets", "pods/exec", "pods/status",
↪"pods/log", "persistentvolumeclaims", "namespaces", "horizontalpodautoscalers",
↪"endpoints"]
  verbs: ["get", "create", "delete", "list", "update", "watch", "patch"]
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: <role-id>
  namespace: <namespace-id>
subjects:
- kind: ServiceAccount
  name: <service-account-id>
  namespace: <namespace-id>
roleRef:
  kind: ClusterRole
  name: <role-id>
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: v1
kind: Secret
metadata:
  name: <service-account-id>
  namespace: <namespace-id>
  annotations:
    kubernetes.io/service-account.name: <service-account-id>
type: kubernetes.io/service-account-token
```

Noronha needs a NFS, which can be deployed in Kubernetes through the script below.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: <nfs-id>
  namespace: <namespace-id>
spec:
  accessModes:
  - ReadWriteOnce
```

(continues on next page)

(continued from previous page)

```
  resources:
    requests:
      storage: 128Gi
  storageClassName: <storage_class>  # edit the storage class for provisioning disk on␣
↪demand (Azure: default | Others: standard)
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: <nfs-id>
  namespace: <namespace-id>
spec:
  selector:
    matchLabels:
      role: <nfs-id>
  template:
    metadata:
      labels:
        role: <nfs-id>
    spec:
      containers:
      - name: <nfs-id>
        image: gcr.io/google_containers/volume-nfs:0.8
        args:
          - /nfs
        ports:
          - name: nfs
            containerPort: 2049
          - name: mountd
            containerPort: 20048
          - name: rpcbind
            containerPort: 111
        securityContext:
          privileged: true
        volumeMounts:
          - mountPath: /nfs
            name: mypvc
      volumes:
        - name: mypvc
          persistentVolumeClaim:
            claimName: <nfs-id>
---
apiVersion: v1
kind: Service
metadata:
  name: <nfs-id>
  namespace: <namespace-id>
spec:
  clusterIP: <nfs_server>  # edit the nfs internal ip (if this one is already taken)
  ports:
    - name: nfs
      port: 2049
```

(continues on next page)

(continued from previous page)

```
    - name: mountd
      port: 20048
    - name: rpcbind
      port: 111
  selector:
    role: <nfs-id>
```

5.1.3 Configuring Noronha client on the machine

After the cluster is ready, you need to configure Noronha on your machine. You may do this by configuring the .nha/nha.yaml file on your home directory.

```
logger:
  level: DEBUG
  pretty: true
  directory: /logs
  file_name: clever.log
docker:
  target_registry: <docker_registry>  # edit the docker registry used by the k8s cluster
  registry_secret: <registry_secret>  # edit the name of the k8s secret that holds your␣
↪docker registry's credentials
container_manager:
  type: kube
  namespace: clever
  api_timeout: 600
  healthcheck:
    enabled: true
    start_period: 120
    interval: 60
    retries: 12
  storage_class: <storage_class>  # edit the storage class for provisioning disk on␣
↪demand (Azure: default | Others: standard)
  nfs:
    server: <nfs_server>  # edit the nfs server ip address (same as in nfs.yaml)
    path: /nfs/nha-vols
  resource_profiles:
    nha-train:
      requests:
        memory: 5120
        cpu: 2
      limits:
        memory: 8192
        cpu: 4
```

You may share this file with other Noronha users as a template for your Noronha cluster.

5.1.4 Deploy Artifactory, MongoDB and NodeJS

Noronha may deploy Artifactory, Mongo and Node by itself:

```
nha -d -p isle artif setup
nha -d -p isle mongo setup
nha -d -p isle router setup
```

5.1.5 Ingress setup

In order to access your Kubernetes cluster from the internet, you may create an Ingress Controller with the following script:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: <ingress-id>
  namespace: <namespace-id>
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    kubernetes.io/ingress.global-static-ip-name: <ip-name>  # name of the static ip␣
↪reservation
spec:
  rules:
  - http:
      paths:
      - path: /predict  # this path is used by OnlinePredict and LazyModelServer when␣
↪serving your model
        pathType: ImplementationSpecific
        backend:
          service:
            name: nha-isle-router
            port:
              number: 80
      - path: /update  # this path is used by LazyModelServer when serving your model
        pathType: ImplementationSpecific
        backend:
          service:
            name: nha-isle-router
            port:
              number: 80
      - path: /artifactory/*  # this path is only required if you want to use␣
↪Artifactory through an ingress
        pathType: Prefix
        backend:
          service:
            name: nha-isle-artif
            port:
              number: 8081
```