nomad

Release 1.20

Mar 01, 2018

Contents

Contents:

CHAPTER 1

Quick Start

To use nomad, you'll have to supply some basic configuration, in `ini` format:

```
[nomad]
engine = sqla
url = sqlite:///test.db
```

Note: See *Acquiring URLs* about ways you can use to specify url of database.

Save this in a file named *nomad.ini*. The directory, containing this file, will be your nomad environment. You can store file under any other name, but then you'll have to supply it as an option to nomad calls (like `nomad -c myconf.ini`).

Then initialize your database to be used with nomad:

```
$ nomad init
```

This will create a table in your database with 2 fields - `name` and `date`. This table is used then for tracking which migrations have been applied already to this database.

And then you can create a migration:

```
$ nomad create 2012-09-21-first
```

This will create directory with name `2012-09-21-first` and two files inside: `migration.ini` and `up.sql`. Name of first file matters - it contains information about dependencies of a migration (which can be passed as `-d` option to `create` command). Name of second file doesn't matter - any `*.sql`, `*.j2`, or executable files (file with executable bit set) will be run. `*.sql` files are applied to database, `*.j2` files are applied to the database after being passed through the `jinja2` template system, and executable files (which can be your script to do something before or after migration, or even migration itself) are just executed.

You can then list or apply migrations - just read help about them (`nomad help ls` or `nomad help apply`). Also, reading *Basic nomad tests* can be helpful as well.

CHAPTER 2

Acquiring URLs

Nomad supports few different ways of specification how it should connect to database.

There are two parameters it has to acquire: `engine` and `url`.

Engine is specified as a string. Supported engines right now consist of `dbapi` and `sqla`. Both of them support SQLite, MySQL and PostgreSQL databases, the first one requiring only db api modules and second one requiring SQLAlchemy library.

URL can be specified in few different ways:

- `url` - just a string, path like `sqlite:///test.db` or `mysql://user:pass@host/db`
- `url-python` - taking variable from Python module, has two approaches to fetching python module:
  - From `sys.path`, when it looks like one: `yourapp.settings:dburl`
  - From filesystem, when it looks like path to file: `../settings.py:dburl`
- `url-file` - taking contents of a file: `../dburl.txt`
- `url-command` - taking output of a command: `grep mysql ../settings.txt`

Note: Look at *URL acquiring test* to see how various options are used.

CHAPTER 3

Basic nomad tests

First, set up environment:

```
$ NOMAD=${NOMAD:-nomad}
$ cat > nomad.ini <<EOF
> [nomad]
> engine = sqla
> url = sqlite:///test.db
> [foo]
> bar = zeta
> EOF
```

First, initialize migrations repository:

```
$ $NOMAD init
Versioning table initialized successfully
$ sqlite3 test.db '.schema'
CREATE TABLE nomad (
            name varchar(255) NOT NULL,
            date datetime NOT NULL
        );
```

First migration:

```
$ $NOMAD create 0-first
$ echo "create table test (value varchar(10));" > 0-first/up.sql
$ $NOMAD ls
\x1b[32m0-first\x1b[0m (esc)
```

Upgrading:

```
$ $NOMAD apply -a
applying migration 0-first:
  sql migration applied: up.sql
$ sqlite3 test.db '.schema test'
CREATE TABLE test (value varchar(10));
```

```
$ sqlite3 test.db 'select name from nomad'
0-first
```

Dependencies:

```
$ $NOMAD create 1-second
$ $NOMAD create 2-third -d 1-second
$ $NOMAD ls
\x1b[32m1-second\x1b[0m (esc)
\x1b[32m2-third\x1b[0m (1-second) (esc)
$ $NOMAD apply 2-third
applying migration 1-second:
  sql migration applied: up.sql
applying migration 2-third:
  sql migration applied: up.sql
```

Natural sorting works:

```
$ $NOMAD create 3-fourth
$ $NOMAD create 10-eleventh
$ $NOMAD ls
\x1b[32m3-fourth\x1b[0m (esc)
\x1b[32m10-eleventh\x1b[0m (esc)
```

No problems with trailing slash that can easily occur from autocomplete:

```
$ $NOMAD apply 3-fourth/
applying migration 3-fourth:
  sql migration applied: up.sql
$ $NOMAD apply 3-fourth
\x1b[31mError: migration 3-fourth is already applied\x1b[0m (esc)
[1]
```

Dependencies should not break when applying all migrations:

```
$ $NOMAD create 11-twelfth -d 10-eleventh
$ $NOMAD apply -a
applying migration 10-eleventh:
  sql migration applied: up.sql
applying migration 11-twelfth:
  sql migration applied: up.sql
```

It's possible to insert % into a db:: $ $NOMAD create 12-thirteen $ echo "insert into test values ('test%');" > 12-thirteen/up.sql $ $NOMAD apply -a applying migration 12-thirteen:

sql migration applied: up.sql

$ sqlite3 test.db 'select * from test' test%

Using configuration templates $ $NOMAD create 13-fourteen $ mv 13-fourteen/up.sql 13-fourteen/up.sql.j2 $ echo "create table {{ foo.bar }} (value varchar(10));" > 13-fourteen/up.sql.j2 $ $NOMAD create 14-fifteen $ mv 14-fifteen/up.sql 14-fifteen/up.sql.j2 $ echo "insert into {{ foo.bar }} values ('test');" >> 14-fifteen/up.sql.j2 $ $NOMAD apply -a applying migration 13-fourteen:

sql template migration applied: up.sql.j2

applying migration 14-fifteen: sql template migration applied: up.sql.j2

$ sqlite3 test.db 'select value from zeta' test

CHAPTER 4

URL acquiring test

Test different methods to acquire URLs.

Directly specified URL:

```
$ NOMAD=${NOMAD:-nomad}
$ cat > nomad.ini <<EOF
> [nomad]
> engine = sqla
> url = sqlite:///test.db
> EOF
$ $NOMAD info
<Repository: .>:
  <SAEngine: sqlite:///test.db>
  Uninitialized repository
```

Setup for Python object tests:

```
$ cat > somemod.py <<EOF
> dburl = 'sqlite:///test-py.db'
> EOF
```

URL from Python object from `sys.path`:

```
$ cat > nomad.ini <<EOF
> [nomad]
> engine = sqla
> url = python:somemod:dburl
> EOF
$ PYTHONPATH=.:$PYTHONPATH $NOMAD info
<Repository: .>:
  <SAEngine: sqlite:///test-py.db>
  Uninitialized repository
```

URL from Python object using path:

```
$ cat > nomad.ini <<EOF
> [nomad]
> engine = sqla
> url = python:\${confdir}/somemod.py:dburl
> EOF
$ $NOMAD info
<Repository: .>:
  <SAEngine: sqlite:///test-py.db>
  Uninitialized repository
```

URL from Python package:

```
$ mkdir package
$ mv somemod.py package/__init__.py
$ cat > nomad.ini <<EOF
> [nomad]
> engine = sqla
> url = python:\${confdir}/package:dburl
> EOF
$ $NOMAD info
<Repository: .>:
  <SAEngine: sqlite:///test-py.db>
  Uninitialized repository
```

URL from a file:

```
$ echo 'sqlite:///test-file.db' > url
$ cat > nomad.ini <<EOF
> [nomad]
> engine = sqla
> url = file:url
> EOF
$ $NOMAD info
<Repository: .>:
  <SAEngine: sqlite:///test-file.db>
  Uninitialized repository
```

URL from a command:

```
$ echo 'sqlite:///test-cmd.db' > url
$ cat > nomad.ini <<EOF
> [nomad]
> engine = sqla
> url = command:"cat url"
> EOF
$ $NOMAD info
<Repository: .>:
  <SAEngine: sqlite:///test-cmd.db>
  Uninitialized repository
```

URL from JSON file:

```
$ echo '{"db": [{"url": "sqlite:///test-json.db"}]}' > url.json
$ cat > nomad.ini <<EOF
> [nomad]
> engine = sqla
> url = json:url.json:db.0.url
> EOF
```

```
$ $NOMAD info
<Repository: .>:
  <SAEngine: sqlite:///test-json.db>
  Uninitialized repository
```

URL from INI file:

```
$ echo '[db]\nurl = sqlite:///test-ini.db' > url.ini
$ cat > nomad.ini <<EOF
> [nomad]
> engine = sqla
> url = ini:url.ini:db.url
> EOF
$ $NOMAD info
<Repository: .>:
  <SAEngine: sqlite:///test-ini.db>
  Uninitialized repository
```

URL from YAML file:

```
$ echo 'db:\n    - url: sqlite:///test-yaml.db' > url.yaml
$ cat > nomad.ini <<EOF
> [nomad]
> engine = sqla
> url = yaml:url.yaml:db.0.url
> EOF
$ $NOMAD info
<Repository: .>:
  <SAEngine: sqlite:///test-yaml.db>
  Uninitialized repository
```

Nothing defined:

```
$ echo '[nomad]\nengine=sqla' > nomad.ini
$ $NOMAD info
\x1b[31mError: database url was not found in the nomad Configuration\x1b[0m (esc)
[1]
```

MultiURL:

```
$ rm url.ini
$ cat > nomad.ini <<EOF
> [nomad]
> engine = sqla
> url = ini:url.ini:db.url sqlite:///test.db
> EOF
$ $NOMAD info
<Repository: .>:
  <SAEngine: sqlite:///test.db>
  Uninitialized repository
$ echo '[db]\nurl = sqlite:///test-multi.db' > url.ini
$ $NOMAD info
<Repository: .>:
  <SAEngine: sqlite:///test-multi.db>
  Uninitialized repository
```

Environment variable:

```
$ cat > nomad.ini <<EOF
> [nomad]
> engine = sqla
> url = env:DATABASE_URL
> EOF
$ DATABASE_URL=sqlite:///test.db $NOMAD info
<Repository: .>:
  <SAEngine: sqlite:///test.db>
  Uninitialized repository
```

Note: Right now only SQL databases are supported (SQLite/MySQL/PgSQL through DB API or anything what SQLAlchemy supports), but whole architecture is structured so that it is easy to add support for NoSQL dbs.

CHAPTER 5

Layout

Nomad's migration store is a directory with `nomad.ini` and directories with migrations inside. Each such directory must contain `migration.ini` to be recognized as a migration and this directory name is an unique identifier of a migration.

Your directory tree thus will look like this:

```
migrations/
  nomad.ini
  2011-11-11-first-migration/
    migration.ini
    up.sql
  2011-11-12-second-migration/
    migration.ini
    1-pre.py
    2-up.sql
    3-post.py
  2011-11-13-third-migration/
    migration.ini
    1-pre.py
    2-up.sql
    3-post.sql.j2
```

Nomad uses table called `nomad` to track what was applied already. It's just a list of applied migrations and dates when they were applied.

CHAPTER 6

Interface

To start working, create `nomad.ini`, and initialize your database (I assume it already exists):

```
$ nomad init
```

Then you can start creating your first migration:

```
$ nomad create 0-initial
```

Put your ALTERs and CREATEs in `0-initial/up.sql` and apply a migration:

```
$ nomad apply -a # or nomad apply 0-initial
```

Nomad should report which migrations it applied successfully, but you can check status of that with `nomad ls -a` (or `nomad ls` to see only unapplied migrations).

I guess it's time to create new migration:

```
$ nomad create 1-next -d 0-initial
```

`-d 0-initial` means you want your `1-next` to depend on `0-initial`. This means Nomad will never apply `1-next` without applying `0-initial` first. You usually want to depend on migrations which created tables you're going to alter, or just to make it easier - on the latest available migration.

CHAPTER 7

Usage

There are three types of migration files that `nomad` supports:

1. Plain SQL files with the extension `.sql`. Just put SQL commands you need to execute in the migration folder and they will be executed.
2. Executable files. All file extensions are supported as long as the file is executable. These files must contain everything necessary to migrate your data, including setting up a database connection. `nomad` will pass all of the *Configuration* variables as environmental variables, prefixed with their section.
3. Template files with the extension `.j2`. These templates will be passed through the Jinja2 templating library. You must install the `jinja2` library for this functionality. The entire *Configuration* is available to the template files as a single dictionary. These could be useful if you are distributing an application where the end user needs to control some aspects of the migrations (ie. additional database users and passwords, additonal database names, etc.).

```
# nomad.ini
[db]
another_user = reader
another_pass = pass
```

```
# migrations/0001-initial/up.sql.j2
CREATE ROLE {{ db.another_user }};
ALTER ROLE {{ db.another_user }} WITH NOSUPERUSER LOGIN PASSWORD '{{ db.another_
↪pass }}' VALID UNTIL 'infinity';
```

Files inside of each migration folder are executed in lexographical order.

CHAPTER 8

Configuration

Nomad reads database connection information from the `[nomad]` section of the `nomad.ini` file.

```
[nomad]
engine = sqla
url = pgsql://user:password@host:port/db
```

Possible configuration options:

- `engine` (required) - SQL engine to use, possible options:
  - `sqla` - use SQLAlchemy as an adapter, supports everything SQLAlchemy supports
  - `dbapi` - use regular DB API, supports `sqlite`, `mysql` and `pgsql`
- `url` (required) - URL to database, takes multiple options, see format below
- `path` - path to migrations (default: directory with `nomad.ini`)

Each migration has its own `migration.ini` file, which, by default, has a single configuration option, `nomad.dependencies`, defining which migration (or migrations) this one depends.

You may add your own configuration variables to either the `nomad.ini` or `migration.ini` files and they will be available in your jinja2 templates as a single dictionary and your executable files as environmental variables.

Note that ini-files are parsed with extended interpolation (use it like `${var}` or `${section.var}`).

A few predefined variables are provided to every migration:

- `confpath` - path to `nomad.ini`
- `confdir` - path to directory, containing `nomad.ini`
- `dir` - path to directory of migration

Example configuration:

configration	executable	template

`[nomad]`
`engine = sqla`
`url = someurl`
 	
`[foo]`
`bar = zeta`
	`NOMAD_ENGINE = sqla`
	`NOMAD_URL = someurl`
	
	`FOO_BAR = zeta`
	
	`NOMAD_CONFPATH = path`
	`NOMAD_CONFDIR = dir1`
	`NOMAD_DIR = dir2`
		`nomad.engine = sqla`
		`nomad.url = someurl`
		
		`foo.bar = zeta`
		
		`nomad.confpath = path`
		`nomad.confdir = dir1`
		`nomad.dir = dir2`

8.1 URL format

Nomad can read connection url to database in a few various ways. `nomad.url` configuration option is a space separated list of descriptions of how Nomad can obtain database connection url.

The easiest one is simply an url (like in config example). The others are:

- `file:<path-to-file>` - a path to file containing connection url
- `env:<var-name>` - an environment variable (do not prefix with *$*)
- `py:<python.mod>:<variable.name>` - a Python path to a module, containing a variable with connection url
- `cmd:<cmd-to-execute>` - command to execute to get connection url
- `json:<path-to-file>:key.0.key` - path to file with JSON and then path to a connection url within JSON object
- `yaml:<path-to-file>:key.0.key` - path to file with YAML and then path to a connection url within YAML object
- `ini:<path-to-file>:<section.key>` - path to INI file (parsed by configparser with extended interpolation) and then path to a connection url within this file

An example:

```
[nomad]
url =
  ini:${confdir}/../settings.ini:db.url
  json:${confdir}/../settings.json:db.url
  sqlite:///${confdir}/../local.db
```

Notice that in all cases in the end you have to return URL to a database in normal format, i.e. `dbtype://user:pass@host:port/dbname?options`.

`options` are supported only by pgsql right now, whatever you put there, nomad will do `set ...` before every migration. Note that if you do not supply anything there, nomad sets `statement_timeout` to 1000 ms and `lock_timeout` to 500 ms by default.

CHAPTER 9

Main ideas

- There are no downgrades - nobody ever tests them, and they are rarely necessary. Just write an upgrade if you need to cancel something.
- You can write migration in whatever language you want, Nomad only helps you track applied migrations and dependencies.
- `.sql` is treated differently and executed against database, configured in `nomad.ini`.
- Only `.sql`, `.j2`, and executable files (sorry, Windows! - though I am eager to hear ideas how to support it) are executed. You can put READMEs, pieces of documentation, whatever you want alongside your migrations.
- Name matters - everything is executed in order. Order is determined by using human sort (so that `x-1.sql` is earlier than `x-10.sql`, you can always check sorting with `ls --sort=version`).

CHAPTER 10

Indices and tables

- genindex
- modindex
- search