Template for Library Repositories

Manual

IO-Aero Team

Dec 18, 2024

Table of Contents

Chapter 1

General Documentation

This section contains the core documentation for setting up and starting with IO-TEMPLATE-LIB. It covers everything from installation to basic and advanced configurations.

1.1 Introduction

TODO

1.2 Requirements

The required software is listed below. Regarding the corresponding software versions, you will find the detailed information in the Release Notes.

1.2.1 Operating System

The supported operating system is Ubuntu with the Bash shell.

1.2.2 Python

This project utilizes Python from version 3.10, which introduced significant enhancements in type hinting and type annotations. These improvements provide a more robust and clear definition of function parameters, return types, and variable types, contributing to improved code readability and maintainability. The use of Python 3.12 ensures compatibility with these advanced typing features, offering a more structured and error-resistant development environment.

1.2.3 Docker Desktop

The project employs PostgreSQL for data storage and leverages Docker images provided by PostgreSQL to simplify the installation process. Docker Desktop is used for its ease of managing and running containerized applications, allowing for a consistent and isolated environment for PostgreSQL. This approach streamlines the setup, ensuring that the database environment is quickly replicable and maintainable across different development setups.

1.2.4 Miniconda

Some of the Python libraries required by the project are exclusively available through Conda. To maintain a minimal installation footprint, it is recommended to install Miniconda, a smaller, more lightweight version of Anaconda that includes only Conda, its dependencies, and Python.

By using Miniconda, users can access the extensive repositories of Conda packages while keeping their environment lean and manageable. To install Miniconda, follow the instructions provided in the `scripts` directory of the project, where the operating system-specific installation script named

`run_install_miniconda` is available for Ubuntu (Bash shell).

Utilizing Miniconda ensures that you have the necessary Conda environment with the minimal set of dependencies required to run and develop the project efficiently.

1.2.5 DBeaver Community - optional

DBeaver is recommended as the user interface for interacting with the PostgreSQL database due to its comprehensive and user-friendly features. It provides a flexible and intuitive platform for database management, supporting a wide range of database functionalities including SQL scripting, data visualization, and import/export capabilities. Additionally, the project includes predefined connection configurations for DBeaver, facilitating a hassle-free and streamlined setup process for users.

1.3 Installation

1.3.1 Python

The project repository contains a `scripts` directory that includes operating system-specific installation scripts for Python, ensuring a smooth setup across various environments.

- **Ubuntu**: For users on Ubuntu, the `run_install_python.sh` script is provided. This Bash script is created to operate within the default shell environment of Ubuntu, facilitating the Python installation process.

1.3.2 AWS Command Line Interface

Within the project's `scripts` directory, you will find a set of scripts specifically designed for the installation of the AWS Command Line Interface (AWS CLI). These scripts facilitate the installation process on different operating systems, ensuring a consistent and reliable setup.

- **Ubuntu**: Ubuntu users should utilize the `run_install_aws_cli.sh` script. This script is a Bash script that simplifies the AWS CLI installation on Ubuntu systems by setting up the necessary repositories and installing the CLI via apt-get.

1.3.3 Miniconda

The `scripts` directory includes a collection of operating system-specific scripts named `run_install_miniconda` to streamline the installation of Miniconda. These scripts are designed to cater to the needs of different environments, making the setup process efficient and user-friendly.

- **Ubuntu Bash Shell**: Ubuntu users can take advantage of the `run_install_miniconda.sh` script. This Bash script is intended for use within the Ubuntu terminal, encapsulating the necessary commands to install Miniconda seamlessly on Ubuntu systems.

1.3.4 Docker Desktop

The `scripts` directory contains scripts that assist with installing Docker Desktop on macOS and Ubuntu, facilitating an automated and streamlined setup.

- **Ubuntu**: The `run_install_docker.sh` script is available for Ubuntu users. This Bash script sets up Docker Desktop on Ubuntu systems by configuring the necessary repositories and managing the installation steps through the system's package manager.

1.3.5 DBeaver - optional

DBeaver is an optional but highly recommended tool for this software as it offers a user-friendly interface to gain insights into the database internals. The project provides convenient scripts for installing

DBeaver on macOS and Ubuntu.

- **Ubuntu**: For Ubuntu users, the `run_install_dbeaver.sh` script facilitates the installation of DBeaver. This Bash script automates the setup process, adding necessary repositories and handling the installation seamlessly.

1.3.6 Python Libraries

The project's Python dependencies are managed partly through Conda and partly through pip. To facilitate a straightforward installation process, a Makefile is provided at the root of the project.

- **Development Environment**: Run the command `make conda-dev` from the terminal to set up a development environment. This will install the necessary Python libraries using Conda and pip as specified for development purposes.
- **Production Environment**: Execute the command `make conda-prod` for preparing a production environment. It ensures that all the required dependencies are installed following the configurations optimized for production deployment.

The Makefile targets abstract away the complexity of managing multiple package managers and streamline the environment setup. It is crucial to have both Conda and the appropriate pip tool available in your system's PATH to utilize the Makefile commands successfully.

1.4 Configuration IO-TEMPLATE-LIB

1.4.1 .settings.io_aero.toml

This file controls the secrets of the **IO-TEMPLATE-LIB** library. This file is not included in the repository. The file `.settings.io_aero_template.toml` can be used as a template.

The customisable entries are:

Parameter	Description
postgres_password	Password of the database user
postgres_password_admin	Password of the database administrator

The secrets can be set differently for the individual environments (`default` and `test`).

Examples:

```
[default]
postgres_password = "..."
postgres_password_admin = "..."

[test]
postgres_password = "postgres_password"
postgres_password_admin = "postgres_password_admin"
```

1.4.2 settings.io_aero.toml

This file controls the behaviour of the **IO-TEMPLATE-LIB** library.

The customisable entries are:

Parameter	Description
check_-value	`default` for productive operation, `test` for test operation
is_verbose	Display progress messages for processing

The configuration parameters can be set differently for the individual environments (`default` and `test`).

Examples:

```
[default]
check_value = "default"
is_verbose = true

[test]
check_value = "test"
```

1.5 Configuration Logging

In **IO-TEMPLATE-LIB** the Python standard module for logging is used - details can be found here.

The file `logging_cfg.yaml` controls the logging behaviour of the library.

Default content:

```
version: 1

disable_existing_loggers: False

formatters:
  simple:
    format: "%(asctime)s [%(name)s] [%(module)s.py  ] %(levelname)-5s
 %(funcName)s:%(lineno)d %(message)s"
  extended:
    format: "%(asctime)s [%(name)s] [%(module)s.py  ] %(levelname)-5s
 %(funcName)s:%(lineno)d \n%(message)s"

handlers:
  console:
    class: logging.StreamHandler
    level: INFO
    formatter: simple
  file_handler:
    class: logging.FileHandler
    level: INFO
    filename: logging_io_aero.log
    formatter: extended

root:
  level: DEBUG
  handlers: [ console, file_handler ]
```

1.6 First Steps

To get started, you'll first need to clone the repository, which contains essential scripts for various operating systems. After cloning, you will use these scripts to install the necessary foundational software. Finally, you will complete the repository-specific installation to set up your environment correctly. Detailed instructions for each of these steps are provided below.

1.6.1 Cloning the Repository

Start by cloning the *io-template-lib* repository. This repository contains essential scripts and configurations needed for the project.

```
git clone https://github.com/io-aero/io-template-lib
```

1.6.2 Install Foundational Software

Once you have successfully cloned the repository, navigate to the cloned directory.

To set up the project on an Ubuntu system, the following steps should be performed in a terminal window within the repository directory:

a. Grant Execute Permission to Installation Scripts

Provide execute permissions to the installation scripts:

```
chmod +x scripts/*.sh
```

b. Install Python and pip

Run the script to install Python and pip:

```
./scripts/run_install_python.sh
```

c. Install AWS Command Line Interface

Execute the script to install the AWS CLI:

```
./scripts/run_install_aws_cli.sh
```

d. Install Miniconda and the Correct Python Version

Use the following script to install Miniconda and set the right Python version:

```
./scripts/run_install_miniconda.sh
```

e. Install Docker Desktop

This step is not required for WSL (Windows Subsystem for Linux) if Docker Desktop is installed in Windows and is configured for WSL 2 based engine.

To install Docker Desktop, run:

```
./scripts/run_install_docker.sh
```

f. Install Terraform

To install Terraform, run:

```
./scripts/run_install_terraform.sh
```

g. Optionally Install DBeaver

If needed, install DBeaver using the following script:

```
./scripts/run_install_dbeaver.sh
```

h. Close the Terminal Window

Once all installations are complete, close the terminal window.

1.6.3 Repository-Specific Installation

After installing the basic software, you need to perform installation steps specific to the *io-template-lib* repository. This involves setting up project-specific dependencies and environment configurations. To perform the repository-specific installation, the following steps should be performed in a command

prompt or a terminal window (depending on the operating system) in the repository directory.

1.6.4 Setting Up the Python Environment

To begin, you'll need to set up the Python environment using Miniconda, which is already pre-installed. You can use the provided Makefile for managing the environment.

a. For production use, run the following command:

```
make conda-prod
```

b. For software development, use the following command:

```
make conda-dev
```

These commands will create and configure a virtual environment for your Python project, ensuring a clean and reproducible development or production environment. The virtual environment is automatically activated by the Makefile, so you don't need to activate it manually.

Minor Adjustments for GDAL

The installation of the GDAL library requires the following minor operating system-specific adjustments:

In Ubuntu, the GDAL library must be installed as follows:

```
sudo apt-get install gdal-bin libgdal-dev
```

1.6.5 System Testing with Unit Tests

If you have previously executed *make conda-dev*, you can now perform a system test to verify the installation using *make tests*. Follow these steps:

a. Run the System Test:

Execute the system test using the following command:

```
make tests
```

This command will initiate the system tests using the previously installed components to verify the correctness of your installation.

b. Review the Test Results:

After the tests are completed, review the test results in the terminal. Ensure that all tests pass without errors.

If any tests fail, review the error messages to identify and resolve any issues with your installation.

1.7 Advanced Usage

TODO

Chapter 2

Development

2.1 Makefile Documentation

This document provides a detailed overview of the Makefile used in developing Python libraries, explaining each target, its purpose, and the tools involved. This Makefile supports tasks from code formatting and static analysis to test automation and documentation generation, tailored for libraries rather than standalone applications.

2.1.1 Makefile Functionalities

General Configuration

This Makefile is configured to streamline various development tasks:

- **MODULE**: Sets the primary module to `iotemplatelib`, representing the library name.
- **PYTHONPATH**: Defines the paths where Python modules are located (`${MODULE}`, `docs`, `scripts`, `tests`).
- **ENV_FOR_DYNACONF**: Sets the Dynaconf environment to `test` for library-specific configurations.
- **LANG**: Configures the environment language encoding to `en_US.UTF-8`.

Available Targets

Each target in the Makefile corresponds to a specific step or tool in the development and deployment process.

Development Targets

- **`help`**: Lists all available commands and their descriptions, which is helpful for new contributors.
- **`dev`**: Combines formatting, linting, and testing into a single target for quick development checks.
- **`docs`**: Generates and validates documentation using Sphinx.
- **`everything`**: Runs all development-related checks, formatting, and tests, then compiles the code with Nuitka.
- **`final`**: Runs a complete workflow, including code formatting, linting, testing, and generating documentation.
- **`pre-push`**: Prepares the code for committing, including running tests, linting, and incrementing the version.

Code Formatting and Linting

- **`format`**: Formats code using:

    - **black**: Formats code according to PEP 8.
    - **docformatter**: Formats docstrings per PEP 257.
- **lint**: Runs several static analysis and linting tools:
    - **ruff**: A high-performance Python linter.
    - **bandit**: Analyzes code for common security issues.
    - **vulture**: Identifies and flags unused code.
    - **mypy**: Enforces type checking.

Testing

- **tests**: Runs all tests using `pytest`.
- **pytest-ci**: Installs `pytest` and related dependencies, then executes all tests.
- **pytest-first-issue**: Stops running tests at the first failure.
- **pytest-ignore-mark**: Skips tests marked `no_ci`.
- **pytest-issue**: Runs only tests marked with `issue`.
- **pytest-module**: Allows for testing a specific module by setting `TEST-MODULE`.

Static Analysis and Security

- **bandit**: Identifies common security vulnerabilities.
- **vulture**: Detects unused and dead code.
- **mypy**: Enforces type-checking rules across the codebase.
- **mypy-stubgen**: Generates stub files to improve type hinting support for external tools and users.

Environment Management and Versioning

- **conda-dev**: Creates a Conda environment for development using a `config/environment_dev.yml` configuration file.
- **conda-prod**: Creates a production-ready Conda environment using `config/environment.yml`.
- **next-version**: Increments the version number using a custom `next_version.py` script.
- **version**: Displays the current versions of Python and pip.

Documentation

- **sphinx**: Generates both HTML and PDF documentation with Sphinx, using a `docs/source` directory for the source files.

Code Compilation

- **nuitka**: Compiles the code with Nuitka to create a dynamic link library for faster execution and distribution.

2.1.2 How to Use the Makefile

1. **Setup**: Ensure all tools mentioned in the Makefile are installed and accessible in your development environment.
2. **Common Commands**:

- **`make dev`**: Runs development checks, including formatting, linting, and testing.
- **`make docs`**: Builds the project documentation in HTML and PDF formats.
- **`make everything`**: Runs a comprehensive pre-check workflow.
- **`make pre-push`**: Prepares code for committing, including version increments.
- **`make conda-dev`** / **`make conda-prod`**: Creates Conda environments based on project requirements.

3. **Testing**:
   - **`make tests`**: Runs all tests.
   - **`make pytest-module TEST-MODULE=<module>`**: Runs tests for a specific module.
4. **Compilation**:
   - **`make nuitka`**: Compiles the project code using Nuitka for improved performance.

2.1.3 Tool Analysis and Necessity

Detailed Tool Analysis

This section reviews each tool in the Makefile and its purpose in the development workflow.

1. **Act (`nektos/act`)**: Used to simulate GitHub Actions locally. Essential for testing CI workflows before pushing to remote repositories, but not required if CI is not configured in GitHub Actions.
2. **Bandit**: Security vulnerability scanner that identifies common issues in Python code. This tool is crucial for codebases where security is a priority.
3. **Black**: Enforces code consistency through standardized formatting, improving readability and reducing merge conflicts.
4. **Compileall**: Compiles Python scripts into bytecode, which is beneficial in performance optimization and distribution.
5. **Conda**: Creates isolated environments for development and production with specific dependencies, ensuring reproducibility across setups.
6. **Coveralls**: Tracks code coverage and uploads results to `coveralls.io`. Useful if coverage metrics are essential to the project's CI/CD goals.
7. **Docformatter**: Ensures docstrings adhere to PEP 257 standards, increasing documentation quality and readability.
8. **Mypy**: Enforces type hints across the codebase, reducing runtime errors and improving code clarity.
9. **Nuitka**: Compiles the codebase to C, creating optimized binaries. While helpful, it may be optional depending on the library's distribution needs.
10. **Pylint**: A linter for enforcing coding standards and detecting potential issues in the code. It complements `ruff` but may be redundant if `ruff` alone meets project needs.
11. **Pytest**: The primary testing tool, with flexible configurations for unit and integration testing.
12. **Ruff**: A high-performance linter that can serve as a faster alternative to `Pylint` for common linting tasks.
13. **Sphinx**: Generates high-quality documentation from code and docstrings, ideal for library documentation.
14. **Vulture**: Detects unused code, keeping the codebase lean by identifying and flagging dead code.

Necessity and Potential Redundancies

- **Essential Tools**: The primary tools for formatting, testing, linting, and documentation (`black`, `pytest`, `sphinx`) are essential for maintaining code quality and documentation standards.
- **Redundancies**:
  - **`ruff` vs. `pylint`**: Both tools serve similar purposes for linting, with `ruff` being faster but less feature-rich. Using one might be sufficient.
  - **Compileall**: May be optional if bytecode optimization is not a priority.
  - **Coveralls**: Necessary only if tracking code coverage is a project goal.
  - **Nuitka**: Suitable for projects where performance is critical or if distributing compiled binaries is necessary. If not, it could be removed to simplify the build process.

Recommendations

1. **Optimize Linting**: Consider using only `ruff` if it meets your needs for linting and code formatting, as it covers a broad range of checks.
2. **Assess Compilation Needs**: `Nuitka` can be excluded if the library does not require compilation into binary format for distribution.
3. **Review Testing and Coverage**: Use `coveralls` and `pytest` if code coverage is an essential metric; otherwise, they can be excluded to streamline the process.

Chapter 3

API Documentation

Here, you will find detailed API documentation, which includes information about all modules within the IO-TEMPLATE-LIB, allowing developers to understand the functionalities available.

3.1 iotemplatelib

3.1.1 iotemplatelib package

Submodules

iotemplatelib.glob_local module

Global constants and variables for IO-Aero systems.

This module defines a set of constants and variables that are globally used throughout the IO-Aero software projects. These include configuration parameters, error messages, and default settings that are essential for the operation and error handling within various components of the system.

iotemplatelib.glob_local.**ARG_TASK** (*str*)
in function calls and command line arguments throughout the software.

Type A constant key used to reference the 'task' argument

iotemplatelib.glob_local.**ARG_TASK_CHOICE** (*str*)
is intended to hold the user's choice of task once determined at runtime.

Type Initially set to an empty string, this variable

iotemplatelib.glob_local.**ARG_TASK_VERSION** (*str*)
argument for tasks, indicating the version of the task being used.

Type A constant key used to reference the 'version'

iotemplatelib.glob_local.**FATAL_00_908** (*str*)
This message is formatted with the name of the OS when raised.

Type Error message template for unsupported operating systems.

iotemplatelib.glob_local.**IO_TEMPLATE_LIB_VERSION** (*str*)
template library, indicating the version of the global constants and variables.

Type The current version number of the IO-Aero

iotemplatelib.glob_local.**LOCALE** (*str*)
ensuring consistent language and regional format settings.

Type Default locale setting for the system to 'en_US.UTF-8',

iotemplatelib.glob_local.**INFO_00_007 = "INFO.00.007 Section: '{section}' - Parameter: '{name}'='{value}'"**

Information message indicating the value of a specific configuration parameter.

Type str

iotemplatelib.io_settings module

Managing the application configuration parameters.

This module initializes and configures the settings for the IO-Aero application using the Dynaconf library. It allows for a flexible, environment-specific configuration that supports multiple file formats and environment variables.

iotemplatelib.io_settings.**settings** (*Dynaconf*)

settings. It is set to read configuration from TOML files specific to the IO-Aero project and environment variables with a specific prefix.

Type A configuration object that handles the application

iotemplatelib.io_settings.**Usage**

This module should be imported to access the `settings` object which provides

the configuration parameters across the application. For example

from config_module import settings print(settings.SOME_CONFIGURATION_KEY)

Module contents

IO-TEMPLATE-LIB.

Chapter 4

About

This section provides additional context and legal information about IO-TEMPLATE-LIB, including release notes and licensing details.

4.1 Release Notes

4.1.1 Version 2.0.3

Release Date: 05.11.2024

Applied Software

Software	Version	Remark	Status
Miniconda	24.9.2		
Python	3.12.7		

4.2 End-User License Agreement

4.2.1 End-User License Agreement (EULA) of IO-Aero Software

This End-User License Agreement ("**EULA**") is a legal agreement between you and **IO-Aero**.

This **EULA** agreement governs your acquisition and use of our **IO-Aero Software** ("Software") directly from **IO-Aero** or indirectly through a **IO-Aero** authorized reseller or distributor (a "Reseller").

Please read this **EULA** agreement carefully before completing the installation process and using the **IO-Aero Software**. It provides a license to use the **IO-Aero Software** and contains warranty information and liability disclaimers.

If you register for a free trial of the **IO-Aero Software**, this **EULA** agreement will also govern that trial. By clicking "accept" or installing and/or using the **IO-Aero Software**, you are confirming your acceptance of the Software and agreeing to become bound by the terms of this **EULA** agreement.

If you are entering into this **EULA** agreement on behalf of a company or other legal entity, you represent that you have the authority to bind such entity and its affiliates to these terms and conditions. If you do not have such authority or if you do not agree with the terms and conditions of this **EULA** agreement, do not install or use the Software, and you must not accept this **EULA** agreement.

This **EULA** agreement shall apply only to the Software supplied by **IO-Aero** herewith regardless of whether other software is referred to or described herein. The terms also apply to any **IO-Aero** updates, supplements, Internet-based services, and support services for the Software, unless other terms accompany those items on delivery. If so, those terms apply.

License Grant

IO-Aero hereby grants you a personal, non-transferable, non-exclusive licence to use the **IO-Aero Software** on your devices in accordance with the terms of this **EULA** agreement.

You are permitted to load the **IO-Aero Software** (for example a PC, laptop, mobile or tablet) under your control. You are responsible for ensuring your device meets the minimum requirements of the **IO-Aero Software**.

You are not permitted to:

- Edit, alter, modify, adapt, translate or otherwise change the whole or any part of the Software nor permit the whole or any part of the Software to be combined with or become incorporated in any other software, nor decompile, disassemble or reverse engineer the Software or attempt to do any such things
- Reproduce, copy, distribute, resell or otherwise use the Software for any commercial purpose
- Allow any third party to use the Software on behalf of or for the benefit of any third party
- Use the Software in any way which breaches any applicable local, national or international law
- use the Software for any purpose that **IO-Aero** considers is a breach of this **EULA** agreement Intellectual Property and Ownership

IO-Aero shall at all times retain ownership of the Software as originally downloaded by you and all subsequent downloads of the Software by you. The Software (and the copyright, and other intellectual property rights of whatever nature in the Software, including any modifications made thereto) are and shall remain the property of **IO-Aero**.

IO-Aero reserves the right to grant licences to use the Software to third parties.

Termination

This **EULA** agreement is effective from the date you first use the Software and shall continue until terminated. You may terminate it at any time upon written notice to **IO-Aero**.

It will also terminate immediately if you fail to comply with any term of this **EULA** agreement. Upon such termination, the licenses granted by this **EULA** agreement will immediately terminate, and you agree to stop all access and use of the Software. The provisions that by their nature continue and survive will survive any termination of this **EULA** agreement.

Governing Law

This **EULA** agreement, and any dispute arising out of or in connection with this **EULA** agreement, shall be governed by and construed in accordance with the laws of the United States.

Indices and tables

- genindex
- modindex

5.1 Repository

Link to the repository for accessing the source code and contributing to the project:
IO-TEMPLATE-LIB GitHub Repository

5.2 Version

This documentation is for IO-TEMPLATE-LIB version 2.0.6.

i