Pacific Northwest NATIONAL LABORATORY

PNNL-32485

VOLTTRON Modular Framework

Enabling flexible and scalable deployment solutions

January 2022

Craig H. Allwardt
Chandrika Sivaramakrishanan
Shwetha Niddodi
Jereme Haack

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

PNNL-32485

VOLTTRON Modular Framework

Enabling flexible and scalable deployment solutions

January 2022

Craig H. Allwardt
Chandrika Sivaramakrishanan
Shwetha Niddodi
Jereme Haack

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99354

Acknowledgments

The authors would like to thank Andrew Rodgers of ACE IoT Solutions, Ben Bartling of Slipstream, David Raker of University of Toledo, Nancy Min at Ecolong and the rest of the Eclipse VOLTTRON Community for their engagement with the project and this effort in particular.

Version History

Version	Date	Description
1.0	12/30/2021	Initial Draft Release
1.1	01/13/2022	Added section using copier for agent generation

Contents

Figures

1. Introduction

VOLTTRON™ is an open-source platform for distributed sensing and control. The platform provides services for collecting and storing data from buildings and devices and provides an environment for developing applications which interact with that data. The platform allows developers to build out their use cases by utilizing these frameworks and integrating new capabilities. To simplify the deployment of systems built on VOLTTRON, a new way of organizing the codebase is being explored. This document details these efforts through a new code repository layout for the VOLTTRON platform and services, and how this new layout provides targeted deployments using standard python deployment packages (wheels). In addition, this paper will discuss the development of third-party agents and how they can be integrated within the VOLTTRON ecosystem. Finally, we will discuss core platform development and direction for the modularized version of VOLTTRON.

2. Motivation

The current VOLTTRON source code is in a single repository – https://github.com/VOLTTRON/volttron - and contains the code for core platform, essential service agents, commonly used agents, documents, and test cases. This document will refer this version of VOLTTRON source as the monolithic version.

Although the current structure makes it easy to search for VOLTTRON features and agents, it requires users to clone the entire VOLTTRON code repository into their local environment to create a running VOLTTRON instance. End users cannot pick and choose only the required agents to download/setup in their environment. Updates done to individual agents cannot be released to the community as soon as they are available but must be held back to be a part of the bigger VOLTTRON release. The monolithic structure also imposes license and Python version requirements across all the entire code base as libraries must be compatible with all the parts of the system. All these drawbacks severely limit ease of deployment and flexibility.

We want to update VOLTTRON's code structure such that VOLTTRON server and agents can be developed, deployed, and maintained independently, each even in its own environment or container. Figure 1 shows our vision for this new VOLTTRON modular architecture.

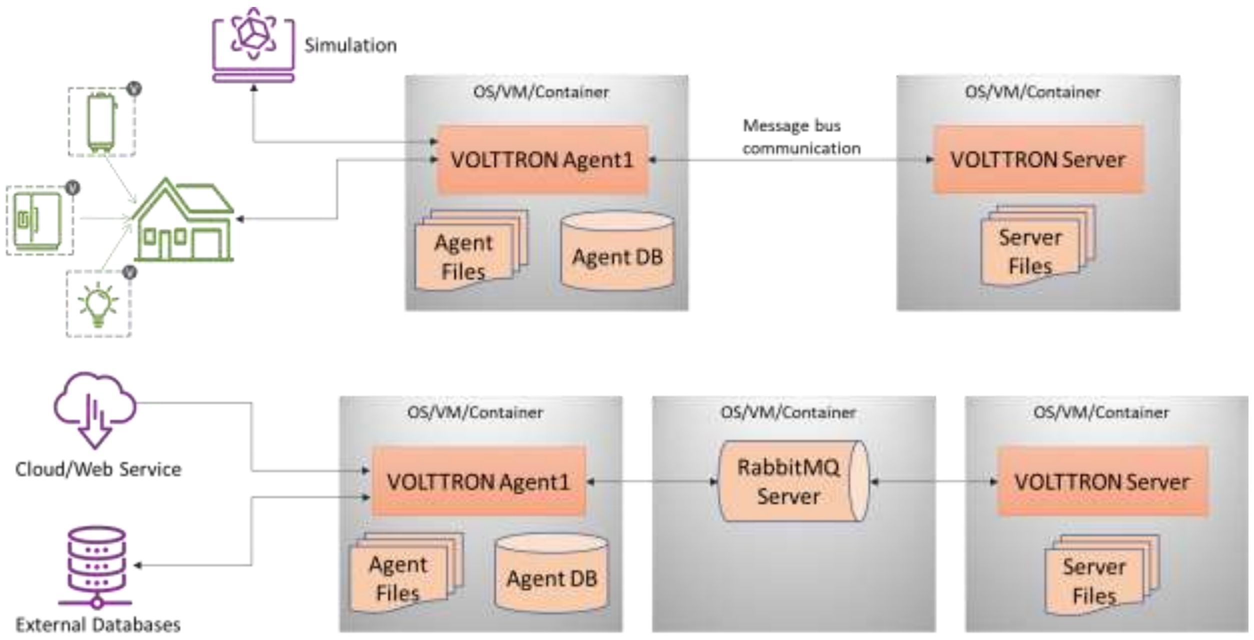


Figure 1: Architecture goal

As a first step towards this modular architecture, we have split the VOLTTRON code into multiple logical or modular units. This modular version of VOLTTRON, helps us overcome many of the limitations of monolithic code and provide additional benefits as well

- End users can install volttron-server (the base platform) and only the agents they are interested in using Python's standard package installer: pip

- Each of the modular component can be updated and released for community use independently

- Agent developers need not clone core VOLTTRON code and can pick a package management tool of their choice. For example: setuptools, poetry, pipenv

- Agent developers can pin specific version of volttron client as requirement for their agent. This also simplifies testing, as changes in agent code need not trigger a regression testing of VOLTTRON core

- Agents can be published to PyPI and deployed from PyPI

- Allows community members to maintain their code in their own repositories making code ownership clear for agents and packages

- Licenses need to be compatible only within individual repository/wheel

- Multiple instances of an installed agent share the same source code

While this modular design simplifies aspects of deployment, it creates more complexity for core VOLTTRON development. Contributors will need to clone multiple repositories and manage version compatibility across full VOLTTRON releases. However, we believe the benefits to the VOLTTRON community outweighs the added burden on core VOLTTRON development. The goal here is to improve the long-term flexibility of the platform and enable deployers to better tailor the platform to their needs. We welcome feedback from the community on this effort and hope to address as many use cases as possible.

3. Repositories

In the current modular version of VOLTTRON, VOLTTRON's core code is split between three main components, each in its own github repository:

1. volttron-server (https://github.com/VOLTTRON/volttron-server)
2. volttron-client (https://github.com/VOLTTRON/volttron-client)
3. volttron-utils (https://github.com/VOLTTRON/volttron-utils)

Each of these code repositories is responsible for building a python wheel, an installable component for use in a python environment. The volttron-server wheel depends on volttron-client, and volttron-client depends on volttron-utils. In addition to the core components, VOLTTRON agents are broken out into their own repositories. In the following sections we describe these three core components in detail. Figure 2 shows the components and its relations.

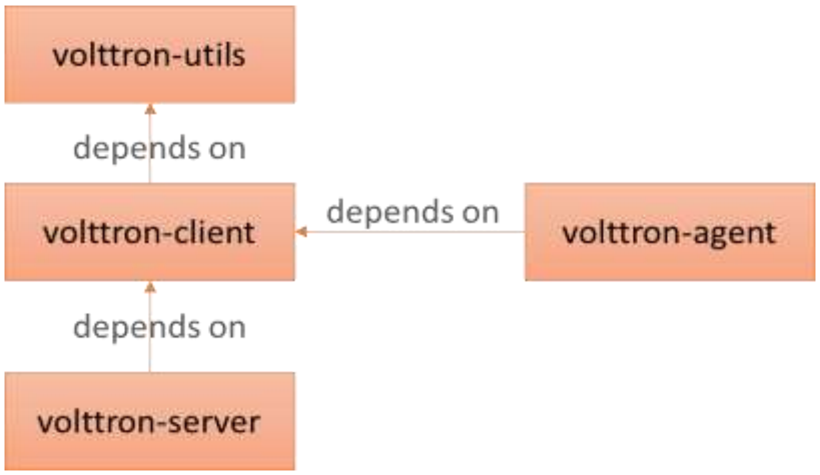


Figure 2: Modular VOLTTRON components

3.1. volttron-server

This contains all core server-side functionalities of VOLTTRON. This includes essential functions such as handling connections to ZMQ and RabbitMQ message bus, routing, tracking, monitoring, scheduling, and necessary code for managing installed agents.

volttron-server includes dynamically loaded services as well as a standard set of core services. The core services that are executed by default are as follows

1. Authentication service
2. Control service
3. Configuration store
4. Health service

5. Peer service

6. Pubsub service

7. External discovery and routing

3.2. volttron-client

This component contains all the base classes required to easily create a VOLTTRON agent and enable agents' connection and communication with the VOLTTRON server. It provides the base *Agent* class that all VOLTTRON agents inherit. It also contains all classes and functionalities related to the VOLTTRON INTERCONNECT PROTOCOL (VIP) enables communications between agents, controllers, services, and the volttron. VIP enables subsystems - a point to point protocol that defines a function or set of related functions and dictates the message format that follows the subsystem name in a VIP message. Each subsystem provides a wrapper API (Application Program Interface) around VIP message protocol allowing developers a higher level interface for their code. volttron-client provides the following subsystems:

- Hello
- Ping
- RPC
- Pubsub
- Channel
- Auth
- Query
- Web
- Peerlist
- Heartbeat
- Health
- Config store

Except for RPC, all the other subsystems communicate with the corresponding service on the server side to execute the operation.

The volttron-client repository also contains code for command line tools – volttron-ctl/vctl, volttron-cfg/vcfg - and other helper class such as topic name parsers, and health status objects.

3.3. volttron-utils

Volttron-utils contains useful utility methods that are used by both client and server code. For example, certs.py module in volttron-utils contains methods to create Certificate Signing Request (CSR) and CA signed certificates; logging.py contains functions to setup logging and logging to file; time.py contains functions to parse timestamp string to python timestamp and vice versa.

3.4. Agents

Keeping with the modular design paradigm, all agents that work with this new modular version of VOLTTRON will be maintained in its own repository. All agents depend on the volttron-client package and are installed into a volttron-server's environment/virtual environment using vctl install command. Agents can also be pip installed on an isolated environment and run from command line to connect to a local/remote VOLTTRON server.

4. VOLTTRON installation

For users who want to use the VOLTTRON environment with currently available agents, install process would be greatly simplified with the modular version of VOLTTRON. Instead of cloning source code, users can install packages directly from PyPI. Steps for VOLTTRON installation include:

- Install system dependencies
- If needed, setup and activate a python virtual environment
- pip install volttron-server (this will automatically pull down volttron-client and volttron-utils)
- Start volttron server
- vctl install <agent-name> - for each agent that you want to install in your environment. Agents can be installed using pip install <package-name>. Please refer to the section - Installing Agents - for all available options

5. New agent development

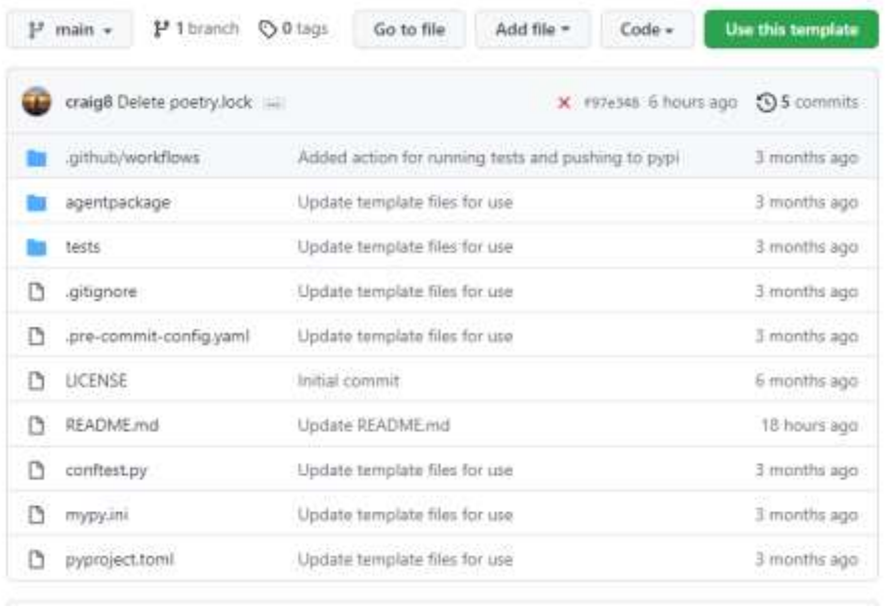
To develop a new agent that works with the modular VOLTTRON code, a developer need not clone VOLTTRON server, client or utils code. The developer only needs a VOLTTRON setup as described in Section 4. Section 5.1 and 5.2 provides two ways to begin projects with example code that can be modified for your use case. The first uses github templates and the second uses a program called copier.

5.1. Example agent and agent template

https://github.com/VOLTTRON/volttron-developer/blob/main/AGENT_DEVELOPMENT.md provides step by step instructions to create and install a simple listener agent. Developer can use this as example to create a new agent.

Developers can also make use of the agent template at https://github.com/VOLTTRON/volttron-new-agent-template as their starting point.

1. Click the Use this template button on the top right corner.

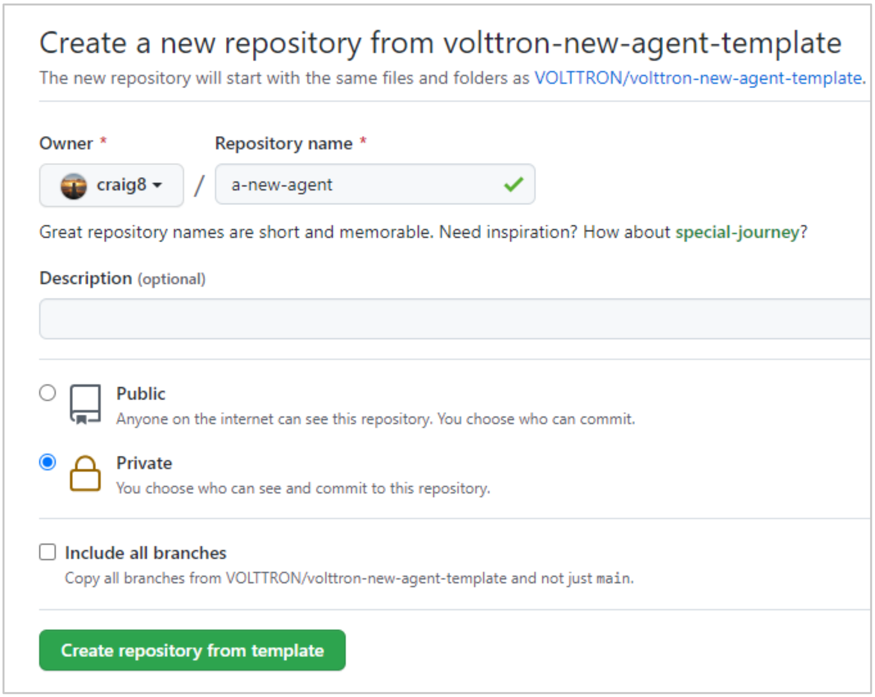


craig8 Delete poetry.lock ... #97e348 6 hours ago 5 commits

.github/workflows	Added action for running tests and pushing to pypi	3 months ago
agentpackage	Update template files for use	3 months ago
tests	Update template files for use	3 months ago
.gitignore	Update template files for use	3 months ago
.pre-commit-config.yaml	Update template files for use	3 months ago
LICENSE	Initial commit	6 months ago
README.md	Update README.md	18 hours ago
conftest.py	Update template files for use	3 months ago
mypy.ini	Update template files for use	3 months ago
pyproject.toml	Update template files for use	3 months ago

Figure 3: Agent template repository

2. Fill in the new repository name and select the github user the new repository will be available from. Finally click the “create repository from template” button



Create a new repository from volttron-new-agent-template

The new repository will start with the same files and folders as VOLTTRON/volttron-new-agent-template.

Owner * craig8 / Repository name * a-new-agent

Great repository names are short and memorable. Need inspiration? How about special-journey?

Description (optional)

( ) Public — Anyone on the internet can see this repository. You choose who can commit.

[✓ Private] — You choose who can see and commit to this repository.

[ ] Include all branches — Copy all branches from VOLTTRON/volttron-new-agent-template and not just main.

Figure 4 Create new agent repository from agent template

3. Once complete you can clone from your new repository and modify the template to fit your agent's use case. The files needing to be customized are as follows (relative to the root of https://github.com/VOLTTRON/volttron-new-agent-template or your new cloned repository)
    a. Rename 'agentpackage' directory to your package name (package names should not have any spaces or dashes, all lowercase and can't start with a number). For this example we will rename it to 'myagentpackage'.
    b. pyproject.toml file
        i. Modify the name on line 2
        ii. Packages on line 8 to { include = "myagentpackage" }
        iii. Update line 10 and 11 to the repository you created from the template
    c. myagentpackage/agent.py
        i. On line 2, 25, 27, and 100 modify "VolttronListenerAgent" to be the agent you want to be used.
    d. README.md
        i. Add your documentation on how to use your agent and its functionality. A default one will be for the templated agentpackage itself.
4. Make sure you have installed poetry (using pipx is recommended https://python-poetry.org/docs/#installing-with-pipx) or installing via command line using https://python-poetry.org/docs/#installation.
5. Open command line to the root of your new repository and execute the command

   poetry install

   Doing this will install the dependencies and requirements.

6. Run command

   poetry build

   to create artifacts – a wheel file and source distribution for your agent – to publish to PyPI. See section - Publishing to PyPI - to publish your artifact and - Installing Agents – to install the agent

5.2. Copier based example

Copier is a program that allows simple question answer format to fill in templates that are created from a github repository. To use it one can install it into an existing pip environment or in a global user pipx environment.
pipx install copier
pip install copier

In addition one should install poetry for this example.
pipx install poetry

Executing the following will start prompting the user for information and then generating a custom agent based upon the answers.
copier "gh:VOLTTRON/copier-poetry-volttron-agent" /path/to/your/new/project

Your project name

project_name? Format: str

🎤 [None]: new-listener

Your project description

project_description? Format: str

🎤 [None]: An example agent for listening to a volttron message bus and printing to standard out

Your full name

author_fullname? Format: str

🎤 [None]: Volttron User

….

The questions are formatted such that default answers are within the [ ] characters and there is a description of context the answers will be used in.

Once completed your new agent is available in /path/to/your/new/project.

cd into /path/to/your/new/project and execute *poetry install* to download dependencies and requirements for your new agent.

5.3. Packaging and dependency

In the modular version of VOLTTRON, agent code can depend on only the volttron-client package and developers are free to use any package manager such as pipenv, poetry, etc. to create a standard python wheel of their agent code. Section: Preparing your package provides more details on packaging your agent code

5.4. Agent data directory

If agents create additional files such as database files, logs, or temporary files, we recommend that agents create those in its own assigned data directory. This will be accessible using a utility method yet to be determined. Based on how your VOLTTRON host machine is setup (i.e., default system level permissions and permissions under VOLTTRON_HOME directory) your agent might still be able to write files to other directories, but this can cause issues in two use cases

1. If VOLTTRON is started in a secure mode where each agent runs as a different system user, agents will have write access only to its designated data directory inside $VOLTTRON_HOME

2. When reinstalling an agent or upgrading to a newer version of source code, only data in the data directory is automatically backed up and restored. Data outside of designated data directory should be handled manually by the user.

Because of the above two reasons we highly recommend that agents write data only to its own data directory

6. Porting agents from monolithic code base

If you have an agent that currently works with monolithic VOLTTRON code base and would like to port it to work with modular VOLTTRON code, there are four main categories of changes to do:

1. Update path of VOLTTRON core packages imported. Since core VOLTTRON code is split into client, server, and utils, the location of VOLTTRON classes and utility methods in your import statements needs to be updated. For example, all imports from *volttron.platform.vip* should now be imported from *volttron.client.vip*. You can find a package mapping between the monolithic code and modular code at https://github.com/VOLTTRON/volttron-developer/blob/main/PACKAGE_MAPPING.csv

2. Agent should now depend on volttron-client package. setup.py needs to be updated to remove existing dependency on “volttron”

3. Changes due to installed agent directory structure. In the modular version of VOLTTRON, agent’s source code is separated from the data used/created by the agent. The directory structure used for the installed agent has also been modified. The table below show the directory structure of an installed agent in monolithic code and modular code.

Monolithic VOLTTRON	Modular VOLTTRON

/home/volttron/.volttron/agents/
└── 914be0d1-142a-4018-a4b5-91f6af75a296
    ├── IDENTITY
    ├── listeneragent-3.3
    │   ├── listener
    │   │   ├── agent.py
    │   │   ├── __init__.py
    │   │   └── settings.py
    │   └── listeneragent-3.3.dist-info
    │   ├── config
    │   ├── DESCRIPTION.rst
    │   ├── entry_points.txt
    │   ├── keystore.json
    │   ├── METADATA
    │   ├── metadata.json
    │   ├── RECORD
    │   ├── top_level.txt
    │   └── WHEEL
    └── TAG
	/home/volttron/.volttron/agents/
	└── listeneragent-3.3_1
	    ├── AUTOSTART
	    ├── config
	    ├── data
	    ├── keystore.json
	    ├── TAG
	    └── UUID

Figure 5 Installed Agent directory comparison

- Agents top level directory is now agent’s VIP identity and not UUID. This makes it easier to identify agents’ directory inside VOLTTRON home. All VOLTTRON commands that currently work with UUID will now work with both UUID and VIP identity (code changes not yet published)

- We highly recommend that agent write data only to its own data directory (for example, <volttron_home>/agents/listeneragent_3.3_3/data). Path to this directory will be accessible using a utility method yet to be determined.

4. Add ability to push to PyPI. Refer to section - Publishing to PyPI - for more details

5. Installed agents will have source code inside your python environment and other files inside $VOLTTRON_HOME. When an agent is uninstalled the source code is uninstalled only if there are no other instance of the agent referring the same code

Figure 1 shows the architecture of the installed modular VOLTTRON instance and agents inside a virtual environment

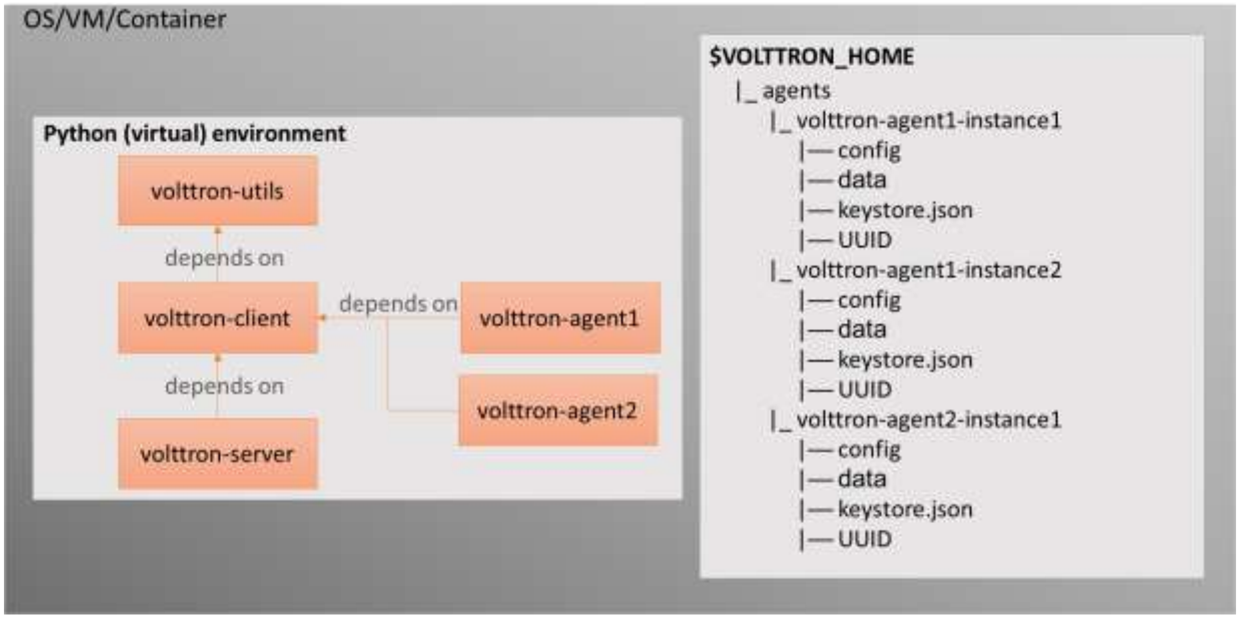


Figure 6: Separation of Agent's code from data

https://github.com/VOLTTRON/volttron-developer/blob/main/AGENT_DEVELOPMENT.md gives step by step instruction on how to update the ListenerAgent (https://github.com/VOLTTRON/volttron/tree/main/examples/ListenerAgent) to work with both the monolithic VOLTTRON code and the modular version of VOLTTRON.

7. Installing Agents

Agents can be installed into a VOLTTRON server environment using the vctl install command. Installing an agent with the vctl command will ship the agent's wheel file to the VOLTTRON server, install the agent's source code into the server's python environment, and create a unique directory for the agent under the server's $VOLTTRON_HOME/agents directory. See Figure 5 the agent's directory structure.

Vctl install command can accept an agent source directory, or a wheel file, or a pypy project name. When a directory name is provided, the code will attempt to create a wheel using one the following three commands

1. pipenv run python3 setup.py bdist_wheel (if agent directory contains a setup.py file and a pipfile)

2. python3 setup.py bdist_wheel (if agent directory contains only a setup.py file)

3. poetry build (if agent directory contains a poetry.lock file)

If you use a different mechanism for packaging your code, generate the wheel file first and then pass the wheel file to the vctl install command.

To install and run an agent outside of VOLTTRON server's environment, build you agent's package and install the wheel file using the command

pip install <wheel file name>

The agent can then be run from the command line module name, to connect to a local/remote server

8. Publishing to PyPI

Publishing agents to PyPI (https://pypi.org) makes it easily accessible to the VOLTTRON community users and hence is highly recommended. Publishing to PyPI can be done in many ways depending on how your agent development environment is setup. For this paper, we will cover two of the most popular ways of doing this, using twine and poetry. Though it is out of scope to go through the full installation process for with all possible options, we will describe the minimum required details and reasoning behind using each of these two tools.

8.1. PyPI and PyPI Test

It is recommended to test your deployed wheels against the test instance of PyPI (https://test.pypi.org/) before deploying against the regular PyPI. TestPyPI is a separate instance of the Python Package Index (PyPI) with a different index/database of wheel files than live PyPI. This allows you to test if your deployment is setup properly before you deploy to the live index. The process is the same between the two services.

The first step to publishing your agent code is to create separate user accounts in PyPI and TestPyPI. It is important not to use the same password in both instances just to make sure you are keeping the two separate.

Once you have created and validated your account you can see the projects that you are responsible for and share ownership amongst many people. You can also create api keys to allow publishing to the project from the various deployment tools (github actions, gitlabs, etc.)

8.2. Preparing your package

1. Name your package: Packages published to PyPI should have a unique name with no underscores or spaces, not start with a number and should be easy to look up. We recommend all volttron agent package start with the prefix “volttron-” to make searching easy.

2. Configure your package: The location and format for specifying these configuration details depends on the packaging tool you use. For example, these data could be part of setup.py, setup.cfg, or pyproject.toml file.

    a. Specify volttron-client package as an agent dependency.

    b. Declare other third-party libraries that your agent depends on

    c. Define an *entry_point* for your package. VOLTTRON looks for one of the below entry points to start an agent

        i. ['console_scripts']

        ii. ['setuptools.installation']['eggsecutable']

        iii. ['volttron.agent']['launch']

    d. We also recommend adding the following metadata.

   i. Author and Author email id

   ii. License information

   iii. Description of what the agent does

   iv. Keywords

   v. PyPI Classifiers:

      1. Framework :: VOLTTRON

      2. Framework :: VOLTTRON :: Agent

      3. Framework :: VOLTTRON :: <version of volttron agent is compatible with> [Repeat this for all versions agent is compatible with]

   vi. README.md - A README file with details on what the agent does, its configurations and steps to install

e. We highly recommend semantic versioning for the package. The version number is given as three numerical components, for instance 0.1.2. The components are called MAJOR, MINOR, and PATCH, and simple rules define when to increment each component (Source/Details)

   i. Increment the MAJOR version when you make incompatible API changes.
   ii. Increment the MINOR version when you add functionality in a backwards-compatible manner.
   iii. Increment the PATCH version when you make backwards-compatible bug fixes.

8.3. Build your package

Build your package to create a wheel file and source tar file. This can be achieved by multiple ways such as setup.py or poetry. For example, if you configured your agent using an setup.py file, you could run the following command in your agent's source directory

```
python setup.py sdist bdist_wheel
```

This would generate a <agentname>-<version>-py3-none-any.*whl* file and a <agentname>-<version>*.tar.gz* file in a *dist* sub directory. These are collectively referred to as artifacts.

8.4. Publish using Twine

Twine is a utility for publishing python packages to PyPI and other python indexes. It provides build independent uploads of source and binary distribution artifacts securely. Twine allows you to pre-sign your artifacts before uploading. In addition, uploading does not require executing your setup.py file allowing testing of distribution packages before releasing.

The release process is as follows

1. Pip install twine into your current environment

2. Upload your artifacts to TestPyPI using the command:

   *twine upload -r testpypi dist/**

   where -r provides the repository and dist/* is the path to the generated distribution artifacts generated in the previous step

3. Once you verify you verify your package, upload your artifacts to PyPI using the command:

   *twine upload dist/**

For more information please see (https://twine.readthedocs.io/en/stable/ and https://packaging.python.org/en/latest/tutorials/packaging-projects/)

8.5. Publish using Poetry

Poetry is a tool for dependency management and packaging in python. By declaring your dependencies through poetry's command line interface (cli) or through its pyproject.toml file, poetry will manage (update/install) the dependencies for you. Poetry provides a single interface to build, package, and publish your project. It also makes it easy to publish your project for multiple versions of Python.

To publish using poetry:

1. Configure a repository: Poetry allows one to use multiple repositories (PyPI or PyPITest or other indexes) for deployment. To add a deployment repository, use the poetry config command as follows:

   poetry config test_pypi https://test.pypi.org/simple/

   where *test_pypi* of your repository name and https://test.pypi.org/simple/ is the actual endpoint you are publishing to.

2. To upload your wheel to the *test_pypi* repository you configured above, execute the following command

   poetry publish --repository test_pypi

   If a *--repository* is not specified poetry by default publishes to PyPI.

Please see (https://python-poetry.org/docs/repositories/) for more information about repositories and poetry.

9. Deployment Use Cases

The modular architecture simplifies the deployment process through the ability to independently develop, deploy and maintain the server and individual agent components. This is key to enabling a scalable deployment process that does not require manual actions to selectively update portions of a monolithic code base. The ability to install VOLTTRON with PyPI allows for standard strategies for deploying large numbers of instances. Additionally, users can tailor the installation according to their individual needs.

With the modular architecture, it’s easier to isolate errors in agent code and debug them. The monolithic repository does not need to be traversed to find the errors. The faulty agent alone can be updated, tested, released, and deployed without disrupting other services. This feature will be extremely useful in use cases such as the PNNL campus deployment where service availability is crucial.

Decoupling of the message bus and authentication features from rest of the VOLTTRON specific code will allow the flexibility to integrate with other third-party message libraries auth mechanisms seamlessly. Implementations that follow the VIP abstraction format will be able to easily connect, disconnect, send, and receive data in VOLTTRON ecosystem. This flexibility will benefit users who are looking to use other message bus libraries such as MQTT, and AMQP that better fit their use case needs. Users utilizing a tightly controlled VPN may not need the authentication services in VOLTTRON. Allowing a no authentication option will enable this use case to trade in-built authentication for increased throughput.

10. Next Steps

Community engagement and feedback is the next step in this process. The goal of this effort is to simplify deployment, support new use cases, and clarify the contribution process. We encourage any feedback from the community to ensure that there are no unforeseen issues proceeding in this direction.

If there are no objections to the plan outlined in this document, the next step will be to complete the modularization and make this the form of the next release. To prepare for that first release, all core agents in the platform must be ported over to the work with the modular server. The corresponding tests must also be made to work in this new environment. Likewise, documentation for the platform and agents must be updated as well.

We also plan to make new agent development easier by updating templating agent to use copier rather than github templates to make it easier to update default metadata values. (https://github.com/VOLTTRON/copier-poetry-volttron-agent). A docker based development environment could potentially make it easier for community users to test this new modular version of VOLTTRON.

The new release process will be documented to make clear the relation between version of the modular code and the numbered VOLTTRON release. A formalized release process will be developed in conjunction with the community.