Pacific Northwest
NATIONAL LABORATORY

PNNL-31991

Application of Configuration Management Approaches to Deployment of the VOLTTRON™ Platform

September 2021

Benjamin H LaRoque
Craig H Allwardt
Shwetha Niddodi
Jereme N Haack

U.S. DEPARTMENT OF ENERGY

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

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor Battelle Memorial Institute, nor any of their employees, makes **any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights**. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or Battelle Memorial Institute. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

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

Printed in the United States of America

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062;
ph: (865) 576-8401
fax: (865) 576-5728
email: reports@adonis.osti.gov

Available to the public from the National Technical Information Service
5301 Shawnee Rd., Alexandria, VA 22312
ph: (800) 553-NTIS (6847)
email: orders@ntis.gov <https://www.ntis.gov/about>
Online ordering: http://www.ntis.gov

PNNL-31991

Application of Configuration Management Approaches to Deployment of the VOLTTRON™ Platform

September 2021

Benjamin H LaRoque
Craig H Allwardt
Shwetha Niddodi
Jereme N Haack

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

Pacific Northwest National Laboratory
Richland, Washington 99354

Summary

This report reviews approaches for the deployment of the VOLTTRON™ building control platform. It focuses on the competing priorities of scalability, repeatability, and simplicity by describing why each is important and how different real-world deployments may strike a unique balance between them. When making a new deployment of VOLTTRON™, the users should evaluate the needs of their particular case against these priorities to identify the best-fit management strategy.

Contents

Figures

1.0 Introduction

In most cases, software deployment falls into one of two categories: (1) ad hoc deployments that involve a person installing, configuring, and running the application according to some manual or interactive procedure; or (2) managed deployments that leverage some form of scripting or automation to execute an equivalent sequence of tasks. An ad hoc pattern is often much faster to complete initially and can be much easier to work with in cases where the desired details are not known in advance. For environments with a small number of deployed instances, the ad hoc solution may be preferrable because the time saved on repetition is reduced and the overhead cost of implementing the solution within the management system is non-zero. On the other hand, as a deployment matures it often becomes increasingly important for it to be robust and reliable, with a clear path for recovery or recreation in the event of a hardware failure in the host system. It is also often the case that a demonstrated deployment pattern will need to be repeated as new systems are added, as an upgrade path after host system upgrades to hardware or operating system, etc.

In this report, we discuss the common features of a managed deployment pattern for the VOLTTRON™ use-case. VOLTTRON™ is an open source software platform supporting agent-based control of remotely operable devices, with a focus on the devices used for building monitoring and control. We review the features such a pattern needs to provide and the various requirements that would drive one to use such a pattern. We present a description of the Pacific Northwest National Laboratory (PNNL) campus deployment of VOLTTRON as a case study, explore some of the deployment challenges exposed by that experience, and discuss some of the upcoming features and plans related to providing better support for these patterns.

2.0 Overview of Managed Deployments

Managed software deployments require an explicit statement of the desired system state, often in the form of configuration files. Configuration management tools provide increased reliability and repeatability of the deployment process by combining that description of the desired system state, with scripted logic for measuring the existing system state and moving it to the desired state as needed. This is in contrast to interactive deployment processes, which require less advance preparation, but which depend on consistent user input. In most cases, adoption of such a process is driven either by the need for high levels of system reliability or by the need to repeat the deployment consistently (either simultaneously across many systems, or sequentially on a single system).

When moving toward a managed deployment of an application like VOLTTRON, the following characteristics are key to a successful design:

1. The desired state of the system should be clearly defined in a format that can be version-controlled. In addition to providing a means of reviewing the history of changes and the ability to restore a previous state, version control allows the relevant state to be stored separately from the deployment and available in the event of hardware loss.
2. The deployment process should by fully automated, taking the desired system state definition as input. This is critical for providing robustness and scalability because otherwise a system is limited by the rate and reliability of a person taking actions. It also decouples the expression of the desired state details from the implementation of how to achieve them, allowing each to evolve independently and to be reused.

3. The process should provide an idempotent implementation. That is to say, if a system is not in the desired state, the automated process should move it to the desired state, but a system that is already in the desired state should be allowed to run without interruption.

Key challenges when assembling a managed deployment (or migrating an ad hoc deployment to a managed state) include the following:

1. It is critical that the deployment system accurately discover the difference between the desired state and the existing state. In particular, it is important to deal with cases in which prior automation steps may have started but not been completed (for example, the presence of updated configuration files on a system does not necessarily mean that they have been loaded).
2. There must be a way for an administrator to investigate and resolve problems with the system without breaking the management system's knowledge of the desired system state. This is generally done either by making state changes exclusively via the management system, or by including tooling that allows the discovered state to be retrieved and updated as the new desired state. Of these two approaches, the former is more robust, but the latter can be a reasonable compromise.
3. For systems like VOLTTRON, where applications deployed on different servers need to be securely connected to each other, the management system needs to properly establish those connections in a secure fashion. That is, systems should not be configured to accept anonymous authentication, but instead the credentials and/or certificates should be obtained from the source of truth and placed in the correct location for use by the application. The management system should also support a scheme for updating and/or rotating the credentials in accordance with best practices.

3.0 A Deployment Case Study: the PNNL Campus Deployment

The VOLTTRON development team at PNNL maintains a collection of deployed VOLTTRON platforms on the Lab's campus in Richland, Washington. The system serves several functions including providing a more realistic testbed for new versions of the software, providing a research and development platform for building controls research, and collecting a long-duration timeseries data set for use by artificial intelligence/machine learning (AI/ML) research (Kim et al. 2020; Huang et al. 2019; Katipamula et al. 2017a, 2017b; and Katipamula et al. 2016).

The system topology, shown in Figure 1, consists of a physical server connected to a facility-specific network for each building being monitored. Each building's VOLTTRON platform includes a Platform Driver agent for data collection and a Forward Historian that sends data to a central platform; the buildings with support for VOLTTRON-based control include those agents in the same platform. The central collection platform runs on a managed virtual machine and runs historians, which connect to various databases for longer-term data storage.

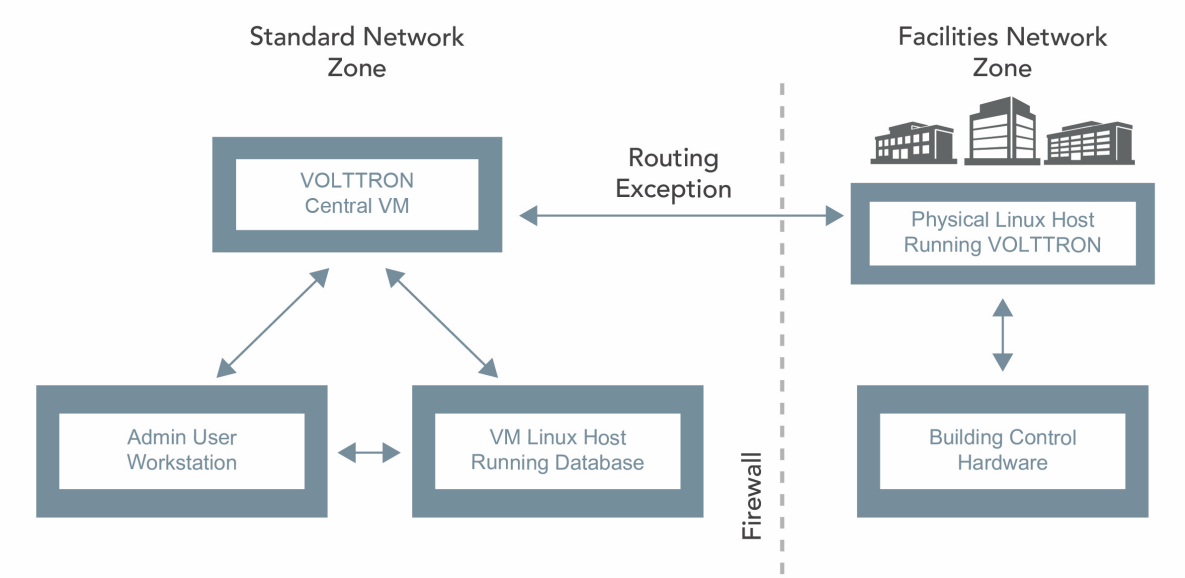

Figure 1. Network topology for two VOLTTRON platforms, one on a restricted facility network zone with connectivity to building control systems and one in an accessible zone. A networking exception allows the platform in the restricted zone to connect to the platform in the standard zone. The platform in the standard network zone is able to then connect to a database for data storage. This platform also provides a network route for administrative users who need to reach the platform in the restricted zone.

The campus deployment has been running since 2016 (https://www.pnnl.gov/main/publications/external/technical_reports/PNNL-26866.pdf) and has expanded and evolved over time based on the projects leveraging it. The initial deployment and configuration was all done manually, as were the addition of new buildings and the upgrade to new versions of the platform. With release of VOLTTRON™'s deployment recipes (https://volttron.readthedocs.io/en/main/deploying-volttron/recipe-deployment.html, https://github.com/volttron/volttron-ansible), all aspects of managing the system have been migrating into an ansible-based system. Ansible is an open source configuration management tool, designed to support distributed and scalable environments. An ansible playbook is a collection of composable units, called roles and modules, which each ensure an aspect of configuration. A playbook is run in combination with an inventory which defines a set of hosts to be configured along with any unique configuration values which determine the exact behavior of the roles and modules when run on each host. The following sections describe the workflows that have been fully or partially implemented.

3.1 Use of Ansible for Host Administration

For the campus deployment, the first aspect of managed deployment is in no way specific to VOLTTRON. The Linux hosts on which the VOLTTRON platforms are deployed need to be maintained with regular security patches and we expect certain system daemon configuration customizations. We have developed several ansible playbooks using the standard features of ansible itself for applying these configurations and for installing updates. This ensures that all hosts have the same configuration, and any needed changes are applied everywhere uniformly. It also automates the process of configuring any new hosts if a new building is added or if the control of a building is migrated onto a new machine.

Because the configuration details are specific to the PNNL campus and the features used are all provided as part of ansible, further details are not included here.

3.2 Use of Ansible for Backup and to Extract Pre-existing Configurations

To take an already existing deployment of VOLTTRON™ and migrate it into the management system, the platform and agent's configuration data must be imported into the format expected by VOLTTRON™'s recipe system. For agent installation and configuration, this is relatively straightforward, because the agent configuration file required by recipe is precisely the same file that is used to install an agent under ad hoc deployment. The configuration store is more complex because the platform retains a single file per agent, but the recipe system expects a file for each entry in the store. A shell script was developed to parse an existing configuration store file into the expected format (https://github.com/VOLTTRON/volttron-ansible/blob/v1.3/examples/process_configuration_store.sh).

3.3 Use of VOLTTRON™'s Recipes for Platform Management

For the campus deployment, we maintain a private git repository with a complete set of input configuration files for the VOLTTRON™ recipe system. It consists of an ansible inventory file with all the platforms listed, as well as a directory for each building's configuration. The files follow the pattern described in the recipe tutorial (https://volttron.readthedocs.io/projects/volttron-ansible/en/main/#step-1-prepare-configuration-files), but the pattern is expanded to include all of the agents deployed on each building, including the full configuration store for the platform driver.

It is therefore possible to use any of the standard recipe playbooks described in VOLTTRON™ documentation (https://volttron.readthedocs.io/projects/volttron-ansible/en/main/#available-recipes) for tasks such as installing a particular version of VOLTTRON™, starting or stopping the platform, or creating a backup archive of the VOLTTRON_HOME and VOLTTRON_ROOT directories.

3.4 Use of VOLTTRON™'s Recipe Components for Custom Tasks

In addition to the workflows provided in the public recipes, we have developed several custom playbooks. The playbooks leverage the variables expected to be part of the configuration used by the recipes, as well as the ansible modules that are included in the ansible-galaxy package provided as part of the recipe system, but they implement additional procedures or workflows. The playbooks are maintained in a private repository next to the recipes' configuration files because their implementations make assumptions that are specific to the deployment or because they are not sufficiently idempotent to be suitable for general release.

For example, we have a playbook that updates the configuration of a single agent, rather than all of the agents on a particular platform (https://github.com/VOLTTRON/volttron-ansible/blob/v1.3/examples/update-one-agent.playbook.yml). This could be problematic because the recipe component that updates the agent configuration files will update files for all agents, but the playbook will only actually run the volttron_agent module to apply those changes to a single agent. Running this playbook is much faster than the full configuration recipe on systems on which many agents are installed, and doing so is very useful when testing configuration variations or a new agent. Because we are able to leverage the existing ansible

modules, this playbook only needs to implement the customized logic for only updating a single VOLTTRON™ agent, rather than needing to reproduce the logic for updating the remote agent.

While not currently implemented, a similar playbook is planned for use with the installation of custom agents related to building controls research. Those playbooks would potentially require custom steps to clone agent source code, which is not part of the main VOLTTRON™ repository, but would then be able to leverage the existing recipe modules for actually installing those agents. Our current plan also involves having a separate inventory that manages only the agents that are specific to the experiment. Because the agent installation does not remove agents that already exist, this having a separate inventory will allow the experiment-related agents to be deployed and managed independently from the permanent agents.

4.0 Known Challenges and Possible Future Patterns

During the process of migrating the PNNL campus deployment of VOLTTRON™ from ad hoc deployments to a managed system, several specific areas of concern were identified. We discuss them here along with our current methods of mitigation. For each concern, we also discuss some changes in or enhancements to the VOLTTRON™ platform and/or the volttron-ansible recipe system, which would enable better solutions in the future. Not all of these are currently on the VOLTTRON™ roadmap and input from the user community is desired as the core team determines what pattern or patterns to support.

4.1 The Multi-platform Pattern

Under the currently documented multi-platform pattern, the process for configuring setting up a pair of platforms and establishing connections between them requires running several commands on each system (that is, configuring and running the central platform; configuring the collector platform, starting it, and installing the forwarder; getting the authentication key for the forwarder and adding it to the auth system on the central system). Other multi-platform patterns would have a similar set of challenges because connections must be initialized on one platform and then approved on another.

For the campus deployment, our current workaround for this problem is to leverage the fact that the recipe system updates the source code and configuration files, but tries to do so in a nondestructive manor with respect to the rest of the agent's state directory. This means that systems that already have existing connections should retain those across playbook executions. The actual authentication subsystem updates are completed manually as needed. An intermediate solution would be to create a new playbook that replicates the sequence of manual steps for creating a connection. However, such a playbook would either need to assume that the two platforms are in an expected running state, or put them into that state, either of which is likely to be in tension with the principles of least surprise and doing no harm (either because a platform is unexpectedly started, or because the playbook is unable to achieve the desired state).

As noted, the main challenge with this system is that establishing the connection requires actions on multiple systems and depends on all systems being in the expected state at that time. The following new workflow patterns may make it much easier to implement this process in a more robust way:

1. Use the vctl tool to make changes to the authentication file directly in circumstances when the platform is not running. Currently the auth subsystem expects that it is the only source of modifications to the files storing auth truth. In the case where the central platform is not running, a configuration must either start that platform to issue auth commands (which is problematic if there is some other reason why the platform needs to remain off), or directly edit the auth file (which is currently not a recommended/supported pattern.
2. Allow an agent's authentication keys to be externally provided to the installation process. Under this pattern, those keys would be an external secret and the recipe system could add the auth entries to the central system when it is installed/configured, even if the collection system is not yet configured.
3. Consume authentication details from some external source of truth. Under this pattern, the central platform would retrieve auth data from an external source (which would either be responsible for auth validation, or from which it would maintain a synchronized truth data). Under such a pattern, the deployment of a collection board would include updating the authentication entries in the external source of truth, to be read by the central system during its next validation or synchronization cycle. This pattern would require a substantial reworking of the auth subsystem, which is not within the current project scope. The advantage of this pattern is that the external source of truth allows the VOLTTRON™ platforms (collector and central) to each be ignorant of the state of the other until runtime, when they actually depend on having an active connection.

4.2 Secrets Management

There are currently a number of VOLTTRON™ agents that consume sensitive information that is outside of the platforms auth subsystem; the most obvious example of this is database historians, which contain connection credentials in their configuration files. With an ad hoc deployment, the main concern with this pattern is the filesystem permission management and/or access restrictions on the host system. However, in a managed deployment the configuration files are generally stored in a version control system hosted on external servers. It is generally considered to be bad practice to store credentials in readable text within a code repository. This is both because it can be hard to fully manage access permissions in a web-hosted code registry, and because when many users may be cloning a repository onto their host systems, there is no way to ensure that all of those systems protect the secret sufficiently.

For the campus deployment, we mitigate these concerns by only storing these repositories on institutionally managed systems with carefully restricted access permissions. There are a number of paths that would improve the security and reliability of this pattern:

1. The solution requiring the largest refactoring, also provides the most opportunities for convenience features. If secrets were managed by an external secret manager (such as HashiCorp's Vault), then all database credentials could be protected and issued by the vault system and would not need to be included in the version-controlled configuration files. Under this pattern, access to the vault itself would need to be handled as part of the host administration process, and the VOLTTRON™ platform would connect to the vault using credentials that are found on the host system.
2. There are a number of patterns, both within ansible and other tools, for enabling files to be encrypted and decrypted as part of a deployment process. If the consuming agents were able to consume sensitive configuration details from a file separate from the rest of their config (possibly a file specified in the main configuration file), then the sensitive data could

be encrypted before placing them under version control and only decrypted at the time of use. This pattern would also enable decoupling of the management of the credential data from the management of the agent. For example, a credential rotation process could update the credentials in the database or other remote system, and then update the corresponding credentials in the known file location on the hosts running VOLTTRON™. The platform could then be told to re-read the credentials (if already running), or would pick up the change when started (if not running), or may even be updated to try re-reading the credential file if a filesystem change is noticed and/or whenever making a new connection to the remote resource.

4.3 The VOLTTRON™ Central Pattern

The PNNL campus deployment uses a pattern where multiple data collection platforms all forward data to a central platform, which receives all of the data and processes them into longer-term storage (see the Appendix for more detailed discussion of our storage pattern). This design pattern is a convenient way to deal with collecting data from systems that are in restricted network zones, as is the case in our application. It also reduces the number of credentials that must be distributed, because the platforms doing the data collection do not need to be able to create their own database connections.

This VOLTTRON™ central pattern is limited by the fact that all data from all collection systems must pass through the message bus of the central platform. The point at which this becomes a bottleneck will depend on the resources available to the central platform, as well as the number of collection platforms, the number of devices on each of those platforms, and the number of points in each device.

4.4 Alternative Message Bus (RabbitMQ)

Recent versions of the VOLTTRON™ platform add support for the use of RabbitMQ as the message bus. The configuration of the RabbitMQ application is integrated with the configuration utilities packaged with VOLTTRON™, so if the application is used the configuration is tightly integrated with the rest of the system. The use of RabbitMQ offers additional patterns for linking multiple platforms as described in the project documentation (https://volttron.readthedocs.io/en/main/deploying-volttron/multi-platform/multi-platform-rabbitmq-deployment.html).

5.0 Conclusion

It is our hope that users in the VOLTTRON™ community will provide feedback that may include case studies of their own deployment patterns, major pain points experienced, or solutions being leveraged. If appropriate for public release, we would include those examples in future versions of this document and/or add the associated references. We also welcome feedback about the planning process for the design changes going forward. In addition to using direct channels of communication with which the reader may already be comfortable, please feel free to contact the VOLTTRON™ core team by email at volttron@pnnl.gov.

6.0 References

S. Katipamula, R.G. Lutes, G. Hernandez, J.N. Haack, and B.A. Akyol. 2016. "Transactional Network: Improving Efficiency and Enabling Grid Services for Building." *Science and Technology for the Built Environment* 22, no. 6:643-654. PNNL-SA-105816. doi:10.1080/23744731.2016.1171628

S. Huang, J. Lian, H. Hao, and S. Katipamula. 2019. "Transactive Control Design for Commercial Buildings to Provide Demand Response." *IFAC-PapersOnLine* 51, no. 34:151-156. PNNL-SA-134436. doi:10.1016/j.ifacol.2019.01.058

S. Katipamula, C.D. Corbin, J.N. Haack, H. Hao, W. Kim, D.J. Hostick, and B.A. Akyol, et al. 2017a. *Transactive Campus Energy Systems: Final Report*. PNNL-26866. Richland, WA: Pacific Northwest National Laboratory

S. Katipamula, K. Gowri, and G. Hernandez. 2017b. "An Open-Source Automated Continuous Condition-Based Maintenance Platform for Commercial Buildings." *Science and Technology for the Built Environment* 23, no. 4:546-556. PNNL-SA-114134. doi:10.1080/23744731.2016.1218236

W. Kim, S. Katipamula, and R.G. Lutes. 2020. "Application of Intelligent Load Control to Manage Building Loads to Support Rapid Growth of Distributed Renewable Generation." *Sustainable Cities and Society* 53. PNNL-SA-143847. doi:10.1016/j.scs.2019.101898

Appendix A – PNNL Campus Data Rotation

The campus data collection system is designed to retain data at varying levels of availability based on data age. This approach is based on the competing interests of researchers, who would like access to as much historical data as possible, and the performance considerations that prevent any storage system from being able to simply grow without bound. We currently store data in the following locations:

1. The forwarder historian on each collection system has a local SQLite cache. This is expected to be empty, but can collect data at times when network outages occur or if the central platform needs to be taken down for maintenance.
2. The platform historian on the central platform receives data from the forward historian on each collection system and can place the data in a sqlite cache. Similar to the forwarder historian, this cache is expected to remain empty except during maintenance cycles.
3. The primary storage is a timescale database that uses the SQL Historian. This system contains approximately 1 year of data. The host is on an institutionally managed virtual machine (VM), which is expected to have a high level of availability (and matches the availability of the VM for the central platform).
4. The secondary storage is also a timescale database, but it retains the prior year's worth of data. It is hosted on a VM with a potentially lower availability, but still provides a standard database interface for access.
5. The central platform also runs an archiver historian, which stores data locally in sqlite files. These files are aggregated and shipped to a tape archive for long-term retention. Access to this data source is much slower because it is a file-based system and also in a tape archive, but it does not suffer performance impacts as the data size grows (up to filesystem limits), because there is no database application that needs to traverse the data set.

Pacific Northwest National Laboratory

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99354
1-888-375-PNNL (7665)

www.pnnl.gov