VOLTTRON-ANSIBLE Documentation

Craig Allwardt, Ben LaRoque

Sep 22, 2021

CONTENTS:

1 Getting started with recipes **3**

2 Available recipes **5**
- 2.1 Ensure host key entries . . . 5
- 2.2 Host configuration . . . 5
- 2.3 Install platform . . . 5
- 2.4 Configure agents . . . 6
- 2.5 Run platform . . . 6
- 2.6 Backup deployment . . . 6

3 Recipe examples **7**
- 3.1 Step 0: Provision the VMs . . . 7
- 3.2 Step 1: Prepare configuration files . . . 8
- 3.3 Step 2: Configured the systems . . . 10
- 3.4 Step 3: Install the platform . . . 10
- 3.5 Step 4: Start the platform . . . 10
- 3.6 Step 5: Configure agents . . . 11
- 3.7 Extra steps . . . 11

4 Feature planning **13**

5 Recipes configuration **15**
- 5.1 Available inventory configuration . . . 15
- 5.2 Available platform configuration . . . 15

6 Indices and tables **17**

Index **19**

Begining with version 7, VOLTTRON introduces the concept of recipes. This system leverages ansible to orchestrate the deployment and configuration process for VOLTTRON. These recipes can be used for any deployment, but are especially useful for larger scale or distributed systems, where it is necessary to manage many platforms in an organized way. Some of the key features are:

1. Platform management logic is implemented using custom ansible modules, which each perform narrow units of work.
2. The roles and modules are composed in playbooks which implement more complete workflows or procedures. These can be used as they are, or taken as starting point for building playbooks specific to a particular use case.
3. Implementation details specific to host-system differences (such as hardware architecture or linux distribution) are abstracted so that the user experience is consistent across supported systems.
4. Ansible's inventory system is leveraged so that the marginal burden of managing additional VOLTTRON deployments is low, and confidence of uniformity among those deployments is high.

CHAPTER

ONE

GETTING STARTED WITH RECIPES

The recipes system is designed to be executed from a user workstation or other server with ssh access to the hosts which will be running the VOLTTRON platforms being configured. In order to do so, you require a python environment with ansible installed. You can do this using pip, your system's package manager, or in whatever environment you like. If installing volttron locally, it is included as an optional feature when bootstrapping the VOLTTRON environment, to enable that see the Bootstrap-Options section in the main documentation and include the `--deployment` flag.

The VOLTTRON recipes are maintained in a dedicated github repository as an ansible-galaxy package. To just use the latest version of the collection, you can use install directly from github with the command:

```
ansible-galaxy install git+https://github.com/volttron/volttron-ansible.git
```

Note that the above requires that you have the `git` package installed). You can also clone or download the repo and install the collection from a local directory with the following comands:

```
ansible-galaxy collection build <path/to/volttron-ansible>
ansible-galaxy install volttron-deployment-<version>.tar.gz
```

Note:

- here the first command expects a path to the root of the volttron-ansible repo and produces a local .tar.gz archive containing the packaged galaxy collection
- the first command will print the version number of the collection, which is included in the name of the output tar.gz archive and is used as an argument in the second command
- if executing either of the above commands with the output already existing (for example, if you're making local changes which you want to test), you may need to add the `--force` flag to overwrite the existing files.

Additionally, to use the recipes you will need to create a set of recipe configuration files specific to your use case (discussed in more detail in the *Recipes configuration* section). These include:

host inventory file A file which uses a valid ansible inventory (official docs) to configure the details of each remote server to be managed. This file contains details of how the recipe deploys the system, including paths to where files are to be found on the user's system and where files should be created and stored on the managed systems.

platform configuration file A file for each remote VOLTTRON platform containing the runtime configuration details for the platform itself. This file is used to generate the platform's configuration file and to define the agents to be installed in the remote platform (remote agent management is not yet supported, see *Feature planning*).

Examples of these files can be found in the `examples` directory of the volttron-ansible repo and are discussed in the *Recipe examples* section below.

When working with recipes, a user will generally use the `ansible-playbook` command (see the full official documentation). This command is used to execute a playbook, which applies a set of ansible "roles" and "tasks" to

remote systems based on their respective definitions and the user provided inventory. In the *Recipe examples* section there are several working examples, which demonstrate common flags such as `-i` for specifying the inventory file, or `-K` when administrative privileges are required. The official docs provide details for more advanced options, such as using `-l` to apply a playbook to only a subset of the inventory, or other flags for executing only part of a playbook.

CHAPTER

TWO

AVAILABLE RECIPES

All provided recipes are ansible playbooks and can be executed directly using the the `ansible-playook` command line tool. Each of the available playbooks are discussed in the following subsections, they can all be found in the top-level directory of the volttron-ansible repo.

2.1 Ensure host key entries

The `ensure-host-keys.yml` playbook provides a recipe which updates your local user's `known_hosts` file with the remote host keys for each remote listed in your inventory file. This is most commonly useful in cases where VOLTTRON is being deployed to virtual machines which are being provisioned automatically and therefore may have chaning host keys and automatically included ssh keys. It makes changes in your user's local `~/.ssh/known_hosts` file. This playbook has no VOLTTRON-specific content but is provided as a convenience.

2.2 Host configuration

The `host-config.yml` playbook conducts system-level package installation and configuration changes required for installing and running VOLTTRON. The playbook uses the inventory and associated host configuration files to determine which optional dependencies are are required for the particular deployment (for example, dependencies for rabbitMQ when using that message bus). The playbook also allows the user to specify extra system-level dependencies to be included, this can be used as a convenience, to avoid needing to write an additional playbook for installing packages, or to cover the case where custom agents may have additional requirements that the recipes is otherwise unaware of.

Note that because this playbook installs system packages, it must be passed a sudo password when run (this is done with the standard ansible "become" system).

2.3 Install platform

The `install-platform.yml` playbook installs the VOLTTRON source in the configured location, creates the virtual environment with both dependencies and VOLTTRON installed, and configures the platform itself. The recipe detects optional dependencies required by the platform (for example, support for rabbitMQ message bus or web support), as well as supporting extra bootstrap options and PyPI packages to be included. It also creates an activation script which will set VOLTTRON-related environmental variables as well as activating the virtual environment, making it easy to interact with the platform locally if required.

2.4 Configure agents

The `configure-agents.yml` playbook copies the local directory of configuration files to the remote system, and installs and configures all agents listed in the platform's configuration file. The agent installation is done using a loop over calls to the `volttron_agent` custom module, which itself makes calls to the `install_agents.py` script packaged with the volttron platform. Note that currently this script does not support making modifications to an existing agent.

2.5 Run platform

The `run-platform.yml` playbook ensures that the remote VOLTTRON platform instances are in the desired running state. The default state is "running", but this is configurable in the inventory (and since variables can be set from the CLI, both starting and stopping are achievable without changing the playbook or inventory).

2.6 Backup deployment

The `backup.yml` playbook will create a gzipped tar archive of the configured volttron root and home directories on the remote. The default behavior places the archive in the `/tmp` directory on the remote system, but setting the `retrieve_archive` variable to true will pull the archive back to the system from which the playbook is being run. See the `vars` section at the top of the playbook file for comments with more details.

CHAPTER

THREE

RECIPE EXAMPLES

In this section we will go through the process of creating recipe configuration and inventory from scratch and then executing the available recipes. We also show some useful patterns for how you one might leverage the system in case-specific ways going forward (note the summary of upcoming features in the *Feature planning* section).

In this example we will use the VOLTTRON recipes system to prepare multiple virtual machines and to install the VOLTTRON libraries, configure a platform, and start the platform on each machine. We will use the opportunity to demonstrate a few different inventory configuration patterns enabled by ansible, as well as demonstrating how custom interactions.

3.1 Step 0: Provision the VMs

Before you can run anthing on a remote machine, you'll need to have the machine running and have access to it. This could be physical or virtual machines; in this example we just need three available systems. One option for doing this is with vagrant; if you follow their documentation to install vagrant and virtualbox as your hypervisor, then you can use the the `Vagrantfile` included with the VOLTTRON source ($VOLTTRON_ROOT/examples/deployment/Vagrantfile). To start the VMs you run `vagrant up`. You may then use `vagrant ssh-config >> ~/.ssh/config` to add store the ssh access configuration for the VMs for your user. All of the input/configuration files used here can be found in the `examples/deployment` subdirectory of the VOLTTRON repository.

Note:

1. The above instructions assume you are working on a MacOS or native linux system, for Windows users you shoudl reference the configuration documentation for your ssh client.
2. The above command appends to your ssh config, but ssh uses the first relevant configuration it finds. You may need to remove the generated configuration entries before attempting to update them.

Also note that you can configure the (local) behavior of the ansible CLI tools in an `ansible.cfg` file. The search path for those is documented by ansible and includes your home directory and the working directory. In order to use python3 whenever possible it is useful to set the interpreter setting to `auto` as shown in the included file:

Listing 1: ansible.cfg

```
[defaults]
interpreter_python = auto
```

3.2 Step 1: Prepare configuration files

VOLTTRON's recipes require there to be two levels of configuration. The ansible inventory is used to configure the behavior of the roles and playbooks which constitute the recipe, and the platform configurations contain more detailed configuration details of each platform and its components. The actual tasks executed consider both sources so that for any particular configuration choice there should only need to be a single source of truth. We construct each in the following subsections.

The inventory configuration structure available from ansible is quite sophisticated and anything described in the ansible inventory documentation is available for use with recipes. For this example we start with a truely minimal configuration, which consists only of the list of our three machines (where we've made the conventional choice that the inventory host names will match the VM names in the generated ssh configuration.

Listing 2: recipe-inventory.minimal.yml

```yaml
---
all:
  hosts:
    web:
      ansible_host: web
    collector1:
      ansible_host: collector1
    collector2:
      ansible_host: collector2
```

The minimal platform configuration file is a yaml file with a `config` key equal to an empty dictionary. The default location is a file named the same as the inventory host name with extension `.yml`, in a directory next to the inventory file with name also matching the inventory host name. (This location is configurable via inventory variables). In this case the instance name is taken to be the inventory host name and all other configs are left blank. That configuration looks like:

Listing 3: web/web.yml

```yaml
---
config: {}
```

By way of a demonstration, we'll expand the minimal inventory to achieve the following:

- set the volttron_home to be `~/volttron_home` on each systems by using a group variable (line 15)
- override the above to be `~/vhome` on only the web host by using a host variable (line 6)
- configure collector1 to use the local VOLTTRON source tree rather than downloading from github (line 9)
- install the "drivers" optional feature when bootstrapping the volttron virtualenvironment on collector1 (lines 10-11)

These are all achived with an inventory that looks like:

Listing 4: recipe-inventory.yml

```yaml
---
all:
  hosts:
    web:
      ansible_host: web
      volttron_home: "~/vhome"
      volttron_use_local_source: yes
    collector1:
      ansible_host: collector1
      volttron_use_local_source: yes
      volttron_features:
      - drivers
      #extra_requirements:
      #- ipython
      #- jwt #TODO seems like this may need to be a core dependency?
    collector2:
      ansible_host: collector2
  vars:
    volttron_home: "~/volttron_home"
```

You could expand the inventory to assign values to any of the variables documented in the *Recipes configuration* section. These can be applied to specifc hosts, or to groups per the ansible inventory system.

Similarly, we can modify the platform configuration by updating the platform configuration file. For example, we'll set collector1 to use the RabbitMQ message bus by replacing the empty dictionary with that configuration as seen here:

Listing 5: collector1/collector1.yml

```yaml
---
config: {}
  #message-bus: rmq

#TODO this isn't used yet
agents:
  listener:
    agent_state: present
    agent_source: '$VOLTTRON_ROOT/examples/ListenerAgent'
    agent_running: True
    agent_enabled: True
    agent_tag: listen
    agent_config_store:
    - path: listener_store
      name: ""
      absolute_path: False
      present: True
    - path: "asdf"
      name: "foo.json"
      present: False
    - path: top_store.json
      name: pnnl/richland/isb/magaix.json

  historian.sqlite:
    agent_source: '$VOLTTRON_ROOT/services/core/SQLHistorian'
    agent_config: 'config.sqlite'
```

(continues on next page)

(continued from previous page)

```
    agent_running: True
    agent_enabled: True
```

3.3 Step 2: Configured the systems

With the configuration files written, we can start running some actual recipes. The first needed is host-config. Since this needs admin access on the remote systems, we add the `-K` flag, so the execution command looks like:

```
ansible-playbook -K \
                 -i <path/to/your/inventory>.yml \
                 <path/to/>volttron/deployment/recipes/host-config.yml
```

Take note of the output while running, each step reports on the action taken on every remote, which may differ based on configuration choices made in the prior step.

3.4 Step 3: Install the platform

Having configured the system in the last step, we now are able to install and configure the VOLTTRON platform. Because this is a user-space process, the `-K` flag is not used. This command looks like:

```
ansible-playbook -i <path/to/your/inventory>.yml \
                 <path/to/>volttron/deployment/recipes/install-platform.yml
```

Again, if you review the console output your will see that platform configuration differences are reflected.

Having completed this step, many components have been added to the user's home directory on the remote systems (or whatever alternate location configured in the inventory file). If you ssh to those systems you can inspect those directories and files which have been created. In the example, the web node will have the following extra content in the user's home directory:

```
.
├── activate-web-env/
├── ansible_venv/
├── vhome/
├── volttron/
├── volttron-source.tar.gz
└── volttron.venv/
```

3.5 Step 4: Start the platform

Starting the platform follows the same pattern as the prior two steps, the command is:

```
ansible-playbook -i <path/to/your/inventory>.yml \
                 <path/to/>volttron/deployment/recipes/run-platforms.yml
```

This particular playbook simply starts the platform (assuming it is not already running).

If you connect to one of the remote systems, you can source the activation script created during the platform installation step. This activates the VOLTTRON environment as well as setting appropriate environment variables. If you run `vctl`, you'll see that the platform is running (but currently has no agents installed).

3.6 Step 5: Configure agents

With a platform running, you can install agents using local configuration files using the `volttron_agent` module (which is the core of the `configure-agents` playbook. The command is:

```
ansible-playbook -i <path/to/your/inventory>.yml \
                 <path/to/>volttron/deployment/recipes/configure-agents.yml \
                 [-l hostname]
```

where the optional `-l hostname` option can be used to only make changes on one of the managed systems. This flag is a standard feature of ansible, and is stressed here because it is assumed that it will be common that the configuration of a single system will need to be updated to reflect changes in the building configuration or other similar building-specific details.

This playbook ensures that the local configuration directory is synchronized to the remote system and then installs the listed agents into the platform.

If an agent is already installed, the system will *not* update its configuration by default. You must set the "force" option to `True`, which will result in the agent being reinstalled (and therefore updated).

The per-agent configuration supports the same flags as the legacy `install-agents.py` script upon which it is built. The system will also install entries into the agent's configuration store, these can be done individually, or an entire directory structure can be used, where the directory path is used to construct the configuration store entry name. To allow for cases where the full key for one entry would be part of the name of another (and therefore would need to be both a file and a directory on the filesystem), any path element which ends in `.d` will have those two characters removed from the key in the configuration store. The system also supports explicit renaming of individual elements, which is more verbose but allows any particular special case to be covered. The system is also able to remove entries from the configuration store, but you must explicitly list them by name with a status of absent. There is currently no support for removing all configuration store entries installed but not present in the local tree.

Entries are added to the configuration store sequentially using the `vctl` tool. If an agent has a large number of entries, this can be slow and may even result in a timeout. Experience thus far shows that progress is retained and re-running the playbook will complete the installation process.

3.7 Extra steps

Having started the platform, you can leverage ansible's ad-hoc commands to interact with them. For example,:

```
ansible -i <path/to/your/inventory>.yml \
        -m shell \
        -a "volttron/env/bin/vctl status"
```

will attempt to run the `vctl status` command on each remote (assuming they are all using the default VOLTTRON_ROOT location. Similarly, you can override default inventory values from the CLI. If you'd like to shutdown the remote platforms you could either set the `platform_status` variable to `"stopped"` in the inventory, or do it on the fly with:

```
ansible-playbook -i <path/to/your/inventory>.yml \
                 <path/to/>volttron/deployment/recipes/install-platform.yml \
                 -e platform_status=stopped
```

You can also make use of the `-l` flag to limit either of the above to either a specific host or group from the inventory. You'd simply pass the name of the host or group from the inventory as the argument to that flag.

CHAPTER

FOUR

FEATURE PLANNING

The ansible-based recipes feature set has been long requested, but has some challenges when seeking to combine existing VOLTTRON usage patterns and complex state with the normal patterns in ansible (especially idempotence). This initial feature set is admittedly not very expansive, but is intended to be a starting point and an opportunity to see what usage patterns are most desired by the VOLTTRON community. Feedback in that regard is greatly appreciated. Our current planning includes the following priorities:

- Expanded support for managing agents in the installed platforms, eventually to include:
  - deploying source for an agent which is not part of the volttron repository so that custom agents can be used
  - performing code updates to an installed platform
- Support for a multi-platform patterns, including cert exchange between managed platforms

CHAPTER

FIVE

RECIPES CONFIGURATION

Configuration is available in two layers. The ansible inventory is used to configure variables which impact inventory behavior. These are used to override default fact values in the `set-defaults` role, which is included in all playbooks. This single role is used to assign fact values to all variables which are used in any of the included roles and playbooks. The deployment configuration directory contains the platform configurations for the deployed VOLTTRON instances (and is used by the various ansible plays to achieve that configuration). Where relevant, play configurations are taken from the platform configuration to ensure that there is a single source of truth.

5.1 Available inventory configuration

Building ansible inventory can be an expansive topic and is beyond the scope of this documentation, the official documentation on building inventory is a great resource for complete and current details. In addition to ansible's standard variables and facts, the following configurations are used by VOLTTRON's recipes:

5.2 Available platform configuration

For every platform, it is required that there be a configuration file at the path configured in the inventory. This file supports two top-level keys: The `config` key has content which is used to directly populate the platform config (in `$VOLTTRON_HOME/config`). It is also parsed by various recipe components to determine which message bus is in use and to detect if some optional features (such as web) are in use. The `agents` config is currently not used, but will contain details of agents to be installed into the platform. The supported format(s) of this field are still being finalized.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

H

host inventory file, 3

P

platform configuration file, 3