Plone Ansible Playbook Documentation

Release 1.2.0

Steve McMahon / Plone Installer Team

April 10, 2016

Contents

Description

Use Ansible to provision a full-stack Plone server

CHAPTER 1

Introduction

Plone's Ansible Playbook can completely provision a remote server to run a full-stack, production-ready Plone server, including:

- Plone in a cluster configuration;
- Automatic starting and process control of the Plone cluster with supervisor;
- Load balancing of the cluster with HAProxy;
- Caching with Varnish;
- Nginx as a world-facing reverse proxy and URL rewrite engine;
- An outgoing-mail-only mail server using Postfix;
- Monitoring and log analysis with munin-node, logwatch and fail2ban.
- Use of a local VirtualBox provisioned via Vagrant to test and model your remote server.

An Ansible playbook and roles describe the desired condition of the server. The playbook is used both for initial provisioning and for updating.

Note: If you want to take more control of your playbook, the Plone server role is available by itself, and is listed on Ansible Galaxy.

1.1 TL;DR

1. Install a current version of Ansible;
2. If you wish to test locally, install Vagrant and VirtualBox;
3. Check out or download a copy of this package;
4. Run `ansible-galaxy -p roles -r requirements.yml install` to install required roles;
5. Copy one of the `sample*.yml` files to `local-configure.yml` and edit as needed.
6. To test in a local virtual machine, run `vagrant up` or `vagrant provision`;
7. To deploy, create an Ansible inventory file for the remote host (look at vbox_host.cfg for an example) and run `ansible-playbook --ask-sudo-pass -i myhost.cfg playbook.yml`;
8. Set a real password for your Plone instance on the target server;
9. Set up appropriate firewalls.

1.2 Automated-server provisioning

The goal of an automated-server provisioning system like Ansible is a completely reproducible server configuration. If you wish to achieve this goal, discipline yourself to never changing configuration on your target machines via login.

That doesn't mean you never log in to your provisioned server. It just means that when you do, you resist changing configuration options directly. Instead, change your playbook, test your changes against a test server, then use your playbook to update the target server.

We chose Ansible for our provisioning tool because:

1. It requires no client component on the remote machine. Everything is done via ssh.
2. It's YAML configuration files use structure and syntax that will be familiar to Python programmers. YAML basically represents a Python data structure in an outline. Conditional and loop expressions are in Python. Templating via Jinja2 is simple and clean.
3. Ansible's documentation is excellent and complete.
4. Ansible is easily extended by roles. Many basic roles are available on Ansible Galaxy.

1.3 If you need to log in

You should not need to. But if you do, you should know:

1. The Plone zeoserver and zeoclient processes should be run under the plone_daemon login; they will normally be controlled via supervisor;
2. Run buildout as plone_buildout.

CHAPTER 2

Provisioning a Plone server

2.1 The stack

It's easy to install Plone on a laptop or desktop for testing, development, theming and evaluation. Installing Plone for production, particularly for a busy or complex site is harder, and requires you learn about a variety of moving parts:

- ZEO server
- ZEO clients
- Process-control
- Load balancing
- Reverse-proxy caching
- URL rewriting and HTTPS support including certificate management

If any of this is new to you, spend some time with the Guide to deploying and installing Plone in production before continuing.

2.2 What about other apps?

This playbook assumes that your target server will be pretty much devoted to Plone's stack. If that doesn't match your plans, then feel free to pick and choose among the roles that have been created and gathered to make up this playbook. Then use them and others to create your own.

CHAPTER 3

Major choices

Your production-server requirements may vary widely. Perhaps the biggest variable is the number of logged-in users you wish to support. You may serve thousands of complex pages per second – if they are not customized per user – on the lightest of servers. On the other hand, if you expect to serve 100 pages per second of content that is customized per user, you'll need one or more powerful servers, and will spend serious analysis time optimizing them.

This playbook is trying to help you out at both extremes – and in-between. To meet these varied needs requires that you make some important configuration choices. Fortunately, you're not stuck with them! If a server configuration doesn't meet your needs, scale up your server power and edit your playbook configuration.

Take a look at the `sample*.yml` files for configuration examples. These present the most commonly changed configuration options.

CHAPTER 4

Requirements

4.1 Target server

4.1.1 Supported platforms

At the moment, while the environment with the fullest support for the target server is Debian/Ubuntu, some initial support is available for CentOS. This is simply because the expertise of the initial authors is with the .deb world. Adding RPM environments should not be difficult, but we need help. Your pull requests are welcome.

At the moment, we are testing with Ubuntu 14 (Trusty) LTS, Ubuntu 15 (Vivid) and with Debian wheezy, Debian jessy, and CentOS 7.

The following components are currently not supported for the CentOS environment:

- `jnv.unattended-upgrades`
- `tersmitten.fail2ban`
- The `firewall.yml` playbook.

4.1.2 SSH access; sudo

Beyond the basic platform, the only requirements are that you have `ssh` access to the remote server with full `sudo` rights.

For local testing via virtual machine, any machine that supports VirtualBox/Vagrant should be adequate.

4.2 Local setup

On your local machine (the one from which you're controlling the remote server), you will need a recent copy of Ansible. docs.ansible.com has thorough installation instructions. We will be testing with release versions of Ansible, so don't feel a need to track Ansible development. (Note: don't us your OS package manager to install Ansible; you may get an unusably out-of-date version.)

Ansible's only dependency is a recent version of Python 2.6 or later.

You will also nearly certainly want git, both for cloning the playbook and for version-controlling your own work.

To clone the master branch of the playbook, use the command:

```
git clone https://github.com/plone/ansible-playbook.git
```

4.3 Ansible role requirements

We have a few Ansible role dependencies which you may fulfill via Ansible Galaxy with the command:

```
ansible-galaxy -r requirements.yml -p roles install
```

This should be executed in your playbook directory. Downloaded requirements will be dropped into the `roles` directory there.

CHAPTER 5

Setting up the Playbook

5.1 Clone or branch-and-clone

Take a few moments to think about how you're going to customize the Plone Playbook. Are you likely to make substantial changes? Or simply change the option settings?

If you expect to make substantial changes, you'll want to create your own git branch of the Plone Playbook. Then, clone your branch. That way you'll be able to push changes back to your branch. We assume that you either know how to use git, or will learn, so we won't try to document this usage.

If you expect to change only option settings, then just clone the Plone Playbook to your local computer (not the target server):

```
git clone https://github.com/plone/ansible-playbook.git
```

5.2 Picking up required roles

Roles are packages of Ansible settings and tasks. The Plone Playbook has separate roles for each of the major components it works with. These roles are not included with the playbook itself, but they are easy to install.

To install the required roles, issue the command `ansible-galaxy -p roles -r requirements.yml install` from the playbook directory. This will create a roles subdirectory and fill it with the required roles.

If you want to store your roles elsewhere, edit the `ansible.cfg` file in the playbook directory.

CHAPTER 6

Customizing the deployment

There are three major strategies for customization: branching, a local configuration file and Ansible inventory variables.

If you are working on your own branch, it's yours. You may set variables inside the playbook.

If you cloned or downloaded the master distribution, you will probably want to avoid changing the files from the distribution. That would make it hard to update. Instead, create a new file `local_configure.yml` and put your custom option specifications in it. This file will not be overridden when you pull an update from the master.

For a quick start, copy one of the `sample*.yml` files to `local_configure.yml`, then customize.

Using the local configuration strategy, add only the options you wish to change to `local_configure.yml`. Edit them there.

6.1 Ansible inventory variables

Ansible allows you to set variables for particular hosts or groups of hosts. Check the Ansible documentation on Inventory variables for details. This is a particularly good approach if you are hoping to support multiple hosts, as different variables may be set for different hosts.

If you use inventory variables, note that any variable you set in `local_configure.yml` will override your inventory variables.

Inventory variables are not as practical for use with Vagrant if you're using `vagrant up` to provision. Instead, use `vagrant up --no-provision` to bring up the box, then use `ansible-playbook` to provision.

6.2 Customizing buildout configuration

Plone is typically installed using buildout to manage Python dependencies. Plone's Ansible Playbook uses operating-system package managers to manage system-level dependencies and uses buildout to manage Python-package dependencies.

Buildout cofiguration files are nearly always customized to meet the need of the particular Plone installation. At a minimum, the buildout configuration details Plone add ons for the install. It is nearly always additionally customized to meet performance and integration requirements.

You have two available mechanisms for doing this customization in conjunction with Ansible:

- You may rely on the buildout skeleton supplied by this playbook. It will allow you to set values for commonly changed options like the egg (Python package) list, ports and cluster client count.

- You may supply a git repository specification, including branch or tag, for a buildout directory skeleton. The Plone Ansible Playbook will clone this or pull updates as necessary.

If you choose the git repository strategy, your buildout skeleton must, at a minimum, include `bootstrap.py` and `buildout.cfg` files. It will also commonly contain a `src/` subdirectory and extra configuration files. It will probably **not** contain `bin/`, `var/` or `parts/` directories. Those will typically be excluded in your `.gitignore` file.

If you use a buildout directory checkout, you must still specify in your Playbook variables the names and listening port numbers of any client parts you wish included in the load balancer configuration. Also specify the name of your ZEO server part if it is not `zeoserver`.

CHAPTER 7

The Configuration File

The configuration file format is YAML with Jinja2 templating. It's well-documented at docs.ansible.com.

CHAPTER 8

Testing with Vagrant

This is really easy. Vagrant includes an Ansible provisioner and will run the playbook when you first run `vagrant up` and again when you run `vagrant provision`.

While Vagrant knows about Ansible, and the playbook specification is in your VagrantFile, you still must have Ansible itself available.

If you've installed Ansible globally, no other steps are necessary. If you wish to use a virtualenv to contain your Ansible installation, it's a little more work to get going:

```
cd ansible.playbook
virtualenv ./
bin/pip install ansible
bin/pip install ansible-vagrant
vagrant up

bin/ansible-playbook-vagrant playbook.yml
```

CHAPTER 9

Common errors

ssh stores host keys and checks them every time you try to reconnect to the same address. Since your Vagrant installs are always at the same host and port (127.0.0.1:2222), you will receive *SSH Error: Host key verification failed while connecting to 127.0.0.1:2222* error messages each time you install and connect with a new virtual box.

To resolve these errors, use the command:

```
ssh-keygen -f "~/.ssh/known_hosts" -R [127.0.0.1]:2222
```

to remove the old host key, then try again.

CHAPTER 10

Testing

Vagrant maps host ports into the guest VirtualBox OS. The standard mapping takes host port 2222 to the guest's SSH port, 22.

The Vagrantfile included with this kit maps several more ports. The general rule is to map each guest port to a host port 1000 higher:

```
config.vm.network "forwarded_port", guest: 80, host: 1080
config.vm.network "forwarded_port", guest: 1080, host: 2080
config.vm.network "forwarded_port", guest: 6081, host: 7081
config.vm.network "forwarded_port", guest: 8080, host: 9080
config.vm.network "forwarded_port", guest: 4949, host: 5949
```

Note that when you use host port 1080 to connect to guest host 80, the virtual hosting will not work correctly. You'll get the homepage, but links – including those to stylesheets and JS resources, will be wrong. So, you can't really test virtual host rewriting via Vagrant.

CHAPTER 11

Live host deployment

11.1 Creating a host file

You'll need to tell Ansible how to connect to your host. There are multiple ways to do this. The easiest for our purposes is to create a *manifest* file.

Create a file with a name like `myhost.cfg` that follows the pattern:

```
plone.com ansible_ssh_user=stevem ansible_ssh_host=192.168.1.50 ansible_ssh_port=5555
```

You may leave off the `ansible_ssh_host` setting if the hostname is real. However, when doing early provisioning, it's often not available. `ansible_ssh_port` is only required if you want to use a non-standard ssh port. `ansible_ssh_user` should be the login id on the remote machine. That user must have sudo rights.

11.2 Running your playbook

```
ansible-playbook --ask-sudo-pass -i myhost.cfg playbook.yml
```

The `--ask-sudo-pass` option instructs Ansible to ask for your user password when it uses sudo for provisioning. It's not required if the remote user has password-less sudo rights.

11.3 Updating

Using tags for quick, partial updates.

The following tags are set up in playbook.yml.

- plone
- haproxy
- varnish
- postfix
- logwatch
- munin
- motd
- nginx

When you use one of these tags while running your playbook, only the bare minimum setup and the module named will be updated.

Apply a tag using the *–tags* option. Example: *–tags="nginx"*

11.4 Firewall

The main playbook, `playbook.yml`, does **not** configure your firewall.

A separate playbook, `firewall.yml` sets up a basic firewall that closes all ports except ssh, http and https. The munin-node port is also opened to your monitoring server(s).

Note: To reach other ports, use SSH tunnelling. In the default setup, you will have to use a tunnel and connect to the load-balancer port in order to get access to the Zope root. (The default proxy-cache setup blocks http basic authentication.)

11.5 Passwords

You must set the `plone_initial_password` variable to the desired password for the Zope `admin` user. Use this id only for initial Plone login, then create users within Plone.

11.6 Hotfixes, Updates, Upgrades

Warning: If you are administering an Internet-accessible Plone install, you **must** subscribe to the Plone-Announce mailing list to receive vital security and version update announcements. Expect to apply periodic hotfixes to maintain your site.

This is the **minimum** responsibility of a site administrator. Ideally you should also participate in the Plone community and read other Plone news.

CHAPTER 12

Configuration options

12.1 System options

12.1.1 admin_email

```
admin_email: sysadmin@yourdomain.com
```

It is important that you update this setting. The admin_email address will receive system mail, some of which is vitally important.

If you don't set this variable, the playbook won't run.

12.1.2 motd

```
motd: |
    Message of the day
    for your server
```

Sets the server's message of the day, which is displayed on login.

Defaults to:

```
motd: |
    This server is configured via Ansible.
    Do not change configuration settings directly.
```

12.1.3 auto_upgrades

```
auto_upgrades: (yes|no)
```

Should the operating system's auto-update feature be turned on. You will still need to monitor for updates that cannot be automatically applied and for cases where a system restart is required after an update.

Defaults to *yes*

> **Warning:** Turning on automatic updates does not relieve you of the duty of actively administering the server. Many updates, including vital security updates, will not happen or take effect without direct action.

12.1.4 additional_packages

```
additional_packages:
    - sockstat
```

List any additional operating system packages you wish to install. Default is empty.

Note: The operating system packages necessary for the components in this kit are automatically handled when a part is installed.

12.1.5 timezone

```
timezone: "America/Los_Angeles\n"
```

Specify the timezone that should be set on the server. Default is "UTCn".

Note: The timezone string must be terminated with a newline character (n).

CHAPTER 13

Plone options

13.1 plone_initial_password

```
plone_initial_password: alnv%r(ybs83nt
```

Initial password of the Zope `admin` user. The initial password is used when the database is first created.

Defaults to `""` – which will fail.

13.2 plone_target_path

```
plone_target_path: /opt/plone
```

Sets the Plone installation directory.

Defaults to `/usr/local/plone-{{ plone_major_version }}`

13.3 plone_var_path

```
plone_var_path: /var/plone_var
```

Sets the Plone installation directory.

Defaults to `/var/local/plone-{{ plone_major_version }}`

13.4 plone_buildout_git_repo

```
buildout_git_repo: https://github.com/plone/plone.com.ansible.git
buildout_git_version: master
```

`buildout_git_repo` defaults to none (uses built-in buildout).

`buildout_git_version` is the tag or branch. Defaults to `master`.

Note: If you use your own buildout from a repository, you still need to specify your client count so that the playbook can 1) set up the supervisor specifications to start/stop and monitor clients, and 2) set up the load balancer.

Client part names must follow the pattern *client#* where # is a number (1,2,3 ...). Client ports must be numbered sequentially beginning with 8081 or the value you set for plone_client_base_port. The zeoserver part must be named *zeoserver* and be at 8100 or the value you set for plone_zeo_port.

If you use your own buildout, all Plone settings except `plone_client_count`, `plone_client_base_port`, and `plone_client_max_memory` are ignored.

13.5 plone_major_version

```
plone_version: '5.0'
```

13.6 plone_version

```
plone_version: '5.0'
```

Which Plone version do you wish to install? This defaults to the current stable version at the time you copy or clone the playbook. Both plone_major_version and plone_version should be quoted so that they will be interpreted as strings.

13.7 plone_client_count

```
plone_client_count: 5
```

How many ZEO clients do you want to run?

Defaults to `2`

Note: The provided buildout always creates an extra client `client_reserve` that is not hooked into supervisor or the load balancer. Use it for debugging, running scripts and quick testing. If you need to remotely connect to the reserve client, you'll typically do that via an ssh tunnel.

13.8 plone_zodb_cache_size

```
plone_zodb_cache_size: 30000
```

How many objects do you wish to keep in the ZODB cache.

Defaults to `30000`

Note: The default configuration is *very* conservative to allow Plone to run in a minimal memory server. You will want to increase this if you have more than minimal memory.

13.9 plone_zserver_threads

```
plone_zserver_threads: 2
```

How many threads should run per server?

Defaults to `1`

13.10 plone_client_max_memory

```
plone_client_max_memory: 800MB
```

A size (suffix-multiplied using "KB", "MB" or "GB") that should be considered "too much". If any Zope/Plone process exceeds this maximum, it will be restarted. Set to `0` for no memory monitoring.

Defaults to `0` (turned off)

Note: This setting is used in configuration of the `memmon` monitor in supervisor: superlance plugin.

13.11 plone_additional_eggs

```
plone_additional_eggs:
    - Products.PloneFormGen
    - collective.cover
    - webcouturier.dropdownmenu
```

List additional Python packages (beyond Plone and the Python Imaging Library) that you want available in the Python package environment.

The default list is empty.

Note: Plone hotfixes are typically added as additional eggs.

13.12 plone_sources

```
plone_sources:
  -  "my.package = svn http://example.com/svn/my.package/trunk update=true"
  -  "some.other.package = git git://example.com/git/some.other.package.git rev=1.1.5"
```

This setting allows you to check out and include repository-based sources in your buildout.

Source specifications, a list of strings in mr.developer sources format. If you specify plone_sources, the mr.developer extension will be used with auto-checkout set to "*" and git_clone_depth set to "1".

Private repository source present a special challenge. The typical solution will be to set up a repository user with the ssh public key for the plone_buildout user.

13.13 plone_zcml_slugs

```
plone_zcml_slugs:
    - plone.reload
```

List additional ZCML slugs that may be required by older packages that don't implement auto-discovery. The default list is empty. This is rarely needed.

13.14 plone_additional_versions

```
plone_additional_versions:
  - "Products.PloneFormGen = 1.7.16"
  - "Products.PythonField = 1.1.3"
  - "Products.TALESField = 1.1.3"
```

The version pins you specify here will be added to the `[versions]` section of your buildout. The default list is empty.

13.15 plone_install_zeoserver

```
plone_install_zeoserver: no
```

Allows you to turn on and off the creation of a zeoserver. Defaults to *yes*. Useful if the zeoserver is not on the same machine as the clients.

13.16 plone_zeo_ip

```
plone_zeo_ip: 192.168.1.100
```

The ip address for the Zope database server. Defaults to *127.0.0.1*. Useful if the zeoserver is not on the same machine as the clients.

13.17 plone_zeo_port

```
plone_zeo_port: 6100
```

The port number for the Zope database server. Defaults to `8100`.

13.18 plone_client_base_port

```
plone_client_base_port: 6080
```

The port number for your first Zope client. Subsequent client ports will be added in increments of 1. Defaults to `8081`.

13.19 plone_environment_vars

```
plone_environment_vars:
    - "TZ US/Eastern"
    - "zope_i18n_allowed_languages en"
```

A list of environment variables you wish to set for running Plone instances.

Defaults to:

```
- "PYTHON_EGG_CACHE ${buildout:directory}/var/.python-eggs"
```

13.20 plone_client_extras

```
plone_client_extras: |
    z2-log-level = error
```

Extra text to add to all the client buildout parts. Defaults to "".

13.21 plone_client1_extras

```
plone_client1_extras: |
    webdav-address = 9080
    ftp-address = 8021
```

Extra text to add to only the first client buildout part. Defaults to "".

13.22 plone_extra_parts

```
plone_extra_parts:
  zopepy: |
    recipe = zc.recipe.egg
    eggs = ${buildout:eggs}
    interpreter = zopepy
    scripts = zopepy
  diazotools: |
    recipe = zc.recipe.egg
    eggs = diazo
```

Extra parts to add to the automatically generated buildout. These should be in a key/value format with the key being the part name and the value being the text of the part. Defaults to `{}`.

13.23 plone_buildout_extra

```
plone_buildout_extra: |
  allow-picked-versions = false
  socket-timeout = 5
```

Allows you to add settings to the automatically generated buildout. Any text specified this way is inserted at the end of the `[buildout]` part and before any of the other parts. Defaults to empty.

Use this variable to add or override controlling settings to buildout. If you need to add parts, use `plone_extra_parts` for better maintainability.

13.24 plone_buildout_extra_dir

```
plone_buildout_extra_dir: local_path
```

Copies a local directory or the *contents* of a directory into the buildout directory on the remote server.

Use this variable to drop extra files (or even subdirectories) into the buildout directory. Local path may be absolute or relative to the playbook directory. Put a "/" on the end of the local path if you wish to copy the contents of the directory. Leave of the trailing "/" to copy the directory itself.

If the copied files change, buildout will be run if plone_autorun_buildout is true (the default). However, the autorun mechanism is not able to detect any other kind of change. For example, if you've used this setting, then remove it, the autorun will not be triggered.

13.25 plone_autorun_buildout

```
plone_autorun_buildout: (yes|no)
```

Do you wish to automatically run buildout if any of the Plone settings change? Defaults to `yes`.

13.26 plone_buildout_cache_url

```
plone_buildout_cache_url: http://dist.plone.org/4.3.4/buildout-cache.tar.bz2
```

The URL of a buildout egg cache. Defaults to the one for the current stable version of Plone.

13.27 plone_buildout_cache_file

```
plone_buildout_cache_file: /home/steve/buildout-cache.tar.bz2
```

The full local (host) filepath of a buildout egg cache. Defaults to none. Should not be used at the same time as plone_buildout_cache_url.

13.28 supervisor_instance_discriminator

```
supervisor_instance_discriminator: customer_15
```

Optionally use this variable when you're installing multiple plone servers on the same machine. The value for supervisor_instance_discriminator will be set as a prefix to all supervisor jobs for this plone server.

You do not need to set a supervisor_instance_discriminator if the servers have different instance names.

13.29 Cron jobs

13.30 plone_pack_at

```
plone_pack_at:
  minute: 30
  hour: 1
  weekday: 7
```

When do you wish to run the ZEO pack operation? Specify minute, hour and weekday specifications for a valid *cron* time. See `CRONTAB(5)`. Defaults to 1:30 Sunday morning. Set to `no` to avoid creation of a cron job.

13.31 plone_keep_days

```
plone_keep_days: 3
```

How many days of undo information do you wish to keep when you pack the database. Defaults to `3`.

13.32 plone_backup_at

```
plone_backup_at:
  minute: 30
  hour: 2
  weekday: "*"
```

When do you wish to run the backup operation? Specify minute, hour and weekday specifications for a valid *cron* time. See `CRONTAB(5)`. Defaults to 2:30 every morning. Set to `no` to avoid creation of a cron job.

13.33 plone_keep_backups

```
plone_keep_backups: 3
```

How many generations of full backups do you wish to keep? Defaults to `2`.

Note: Daily backups are typically partial: they cover the differences between the current state and the state at the last full backup. However, backups after a pack operation are complete (full) backups – not incremental ones. Thus, keeping two full backups means that you have backups for `plone_keep_backups * days_between_packs` days. See the collective.recipe.backup documentation.

13.34 plone_keep_blob_days

```
plone_keep_blob_days: 21
```

How many days of blob backups do you wish to keep? This is typically set to *keep_backups * days_between_packs*‘ days. Default is `14`.

13.35 plone_backup_path

```
plone_backup_path: /mnt/backup/plone
```

Where do you want to put your backups? The destination must be writable by the `plone_daemon` user. Defaults to `./var` inside your buildout directory. Subdirectories are created for blob and filestorage backups.

13.36 plone_rsync_backup_options

```
plone_rsync_backup_options: --perms --chmod=ug+rx
```

Rsync options set within the backup scripts (see [collective.recipe.backup](https://pypi.python.org/pypi/collective.recipe.backup#support options)). This can be used (for example) to change permissions on backups so they can be downloaded more easily. Defaults to empty.

CHAPTER 14

Load-balancer options

14.1 install_loadbalancer

```
install_loadbalancer: (yes|no)
```

Do you want to use a load balancer? Defaults to `yes`.

Note: If you decide not to use a load balancer, you will need to make sure that the `loadbalancer_port` setting points to your main ZEO client if you are using a proxy cache. If you are not using a proxy_cache, you must make sure that `proxycache_port` points to the main ZEO client.

Defaults to `yes`.

14.2 loadbalancer_port

```
loadbalancer_port: 6080
```

The front-end port for the load balancer. Defaults to `8080`.

Note: The haproxy stats page will be at `http://localhost:1080/admin`. The administrative password is disabled on the assumption that the port will be firewalled and you will use an ssh tunnel to connect.

14.3 loadbalancer_options

```
loadbalancer_options: "maxconn 1 inter 10000 downinter 2000 rise 1 fall 2 on-error mark-down error-li
```

Use this variable to customize backend options for haproxy.

CHAPTER 15

Caching proxy options

15.1 install_proxycache

```
install_proxycache: (yes|no)
```

Do you want to install the Varnish reverse-proxy cache? Default is `yes`.

Note: If you decide not to use a proxy cache, you will need to make sure that the `proxycache_port` setting points to your load balancer front end. If you are not using a load balancer, you must make sure that `proxycache_port` points to the main ZEO client.

15.2 proxycache_port

```
proxycache_port: 5081
```

The front-end address for the proxy cache. Defaults to `6081`.

Note: We assume the varnish cache and admin ports are firewalled and that you will administer the cache via ssh.

15.3 proxycache_size

```
proxycache_size: 512m
```

Sets the Varnish cache size. Default is `256m` – 256 megabytes.

15.4 proxycache_method

```
proxycache_method: file
```

Use this to specify Varnish's cache mechanism. Default is `malloc`.

15.5 Cache controls

These settings fine-tune the cache rules.

```
# allow compression for all except these extensions
nocompress_ext: (jpg|png|gif|gz|tgz|bz2|tbz|mp3|ogg)

# never set cookies on responses with these extensions
no_response_cookie_ext: (pdf|asc|dat|txt|doc|xls|ppt|tgz|png|gif|jpeg|jpg|ico|swf|css|js)

# To improve caching, on incoming requests remove all except these cookies
cache_sanitize_cookie_exceptions: (statusmessages|__ac|_ZopeId|__cp)

# When these cookies are not found, mark request with
# X-Anonymous header to allow split caching.
nonanonymous_cookies: __ac(|_(name|password|persistent))
```

Defaults are as indicated in the example. Don't change these without giving it some thought.

CHAPTER 16

Web-server options

16.1 install_webserver

```
install_webserver: (yes|no)
```

Do you want to install Nginx? Defaults to `yes`.

Note: If you decide not to install the webserver – which acts as a reverse proxy – you are on your own for making sure that Plone is accessible at a well-known port.

16.2 Virtual hosting setup

```
webserver_virtualhosts:
  - hostname: plone.org
    default_server: yes
    aliases:
      - www.plone.org
    zodb_path: /Plone
    port: 80
    protocol: http
    client_max_body_size: 4M
  - hostname: plone.org
    zodb_path: /Plone
    address: 92.168.1.150
    port: 443
    protocol: https
    certificate_file: /thiscomputer/path/mycert.crt
    key_file: /thiscomputer/path/mycert.key
```

Connects host names to paths in the ZODB. The `address` and `port` are used to construct the `listen` directive. If no address is specified, `*` will be used. If no port is specified, 80 will be used for http or 443 for https. If no protocol is specified, `http` will be used.

Default value:

```
webserver_virtualhosts:
  - hostname: "{{ inventory_hostname }}"
    default_server: yes
    zodb_path: /Plone
```

```
    aliases:
      - default
```

Note: If you are setting up an https server, you must supply certificate and key files. The files will be copied from your local machine (the one containing the playbook) to the target server. Your key file must not be encrypted or you will not be able to start the web server automatically.

Warning: Make sure that your source key file is not placed in a public location.

16.3 Certificates

Certificate files may be specified in one of two ways.

To copy certificate files from the machine running Ansible, use the format:

```
webserver_virtualhosts:
  - hostname: ...
    ...
    certificate_file: /thiscomputer/path/mycert.crt
    key_file: /thiscomputer/path/mycert.key
```

To use files that already exist on the controlled server, use:

```
webserver_virtualhosts:
  - hostname: ...
    ...
    certificate:
      key: /etc/ssl/private/ssl-cert-snakeoil.key
      crt: /etc/ssl/certs/ssl-cert-snakeoil.pem
```

16.4 Redirections, etc.

If you do not specify a zodb_path, the webserver role will not automatically create a location stanza with a rewrite and proxy_pass directives.

If you specify `extra`, the value will be copied into the server stanza before the `location` setions.

Let's take a look at a common use for these options:

```
- hostname: plone.com
  protocol: http
  extra: return 301 https://$server_name$request_uri;
```

This is a *redirect to https* setting.

16.5 Status and monitoring

If you want to monitor your web server, make sure you have a "localhost" hostname or "default" alias with "http" protocol. This virtual server will have the status check set up on localhost.

16.6 You should know

When you do specify a zodb_path, so that the webserver role knows that you're working with Plone, it will block URLs containing "/manage_" and will block http basic authentication. This means that it will be difficult to use the Zope Management Interface via the web server reverse proxy. Instead, use an SSH tunnel to the load balancer. Remember, this is a production installation. It *should* be hard to use the ZMI via the public interface.

CHAPTER 17

Mail-server options

17.1 install_mailserver

```
install_mailserver: (yes|no)
```

Do you want to install the Postfix mail server in a send-only configuration. Default is `yes`.

Note: If you choose not to install a mail server via this playbook, this becomes your responsibility.

17.2 Relaying

```
mailserver_relayhost: smtp.sendgrid.net
mailserver_relayport: 587
mailserver_relayuser: yoursendgriduser
mailserver_relaypassword: yoursendgridpassword
```

Sets up a mail relay. This may be required if you're using a service like Google Compute Engine that doesn't allow outgoing connections to external mailservers. Defaults to none.

CHAPTER 18

Monitoring options

18.1 install_muninnode

```
install_muninnode: (yes|no)
```

Do you want to install munin-node? Defaults to *yes*.

```
muninnode_query_ips:
    - ^127\.0\.0\.1$
    - ^192\.168\.10\.3$
```

What IP address are allowed to query your munin node? Specify a list of regular expressions.

Defaults to `^127\.0\.0\.1$`

Note: For this to be useful, you must set up a munin monitor machine and cause it to query your node.

18.2 install_logwatch

```
install_logwatch: (yes|no)
```

If turned on, this will cause a daily summary of log file information to be sent to the admin email address. Defaults to *yes*

18.3 install_fail2ban

```
install_fail2ban: (yes|no)
```

Fail2ban scans log files and bans IPs that show malicious signs – too many password failures, seeking for exploits, etc. Defaults to `yes`.

Note: fail2ban is only useful with an iptables-style firewall.

CHAPTER 19

Multiple Plone Servers

The easiest way to use this kit is when there is only one Plone installation for each server. You may, though, use it to install multiple Plone instances to a single server. Up to four Plone instances are supported per server. More may be added via minor customization of the playbook.

To install multiple Plone instances to a server, specify all settings that are unique per instance in a *playbook_plones* list. Settings that are not specific to a particular server may be set as usual.

At a minimum, you must set specific values for *plone_instance_name* and for the Plone and load-balancer ports. You'll usually also want to set virtual host settings.

Here's a minimal example:

```
playbook_plones:
  - plone_instance_name: primary
    plone_zeo_port: 8100
    plone_client_base_port: 8081
    loadbalancer_port: 8080
    webserver_virtualhosts:
      - hostname: "{{ inventory_hostname }}"
        aliases:
          - default
        zodb_path: /Plone
  - plone_instance_name: secondary
    plone_zeo_port: 7100
    plone_client_base_port: 7081
    loadbalancer_port: 7080
    webserver_virtualhosts:
      - hostname: www.plone.org
        zodb_path: /Plone
```

Dispatching requests to the matching Plone instance occurs in Varnish, and is done by hostname. So, in the example above, when Varnish sees www.plone.org in a request URL, it will send the request to port 7080, our secondary instance.

Remember, all the settings except the ones in *playbook_plones* are set as documented elsewhere.

Nearly all the plone_* variables, and a few others like loadbalancer_port and webserver_virtualhosts may be set in *playbook_plones*. Let's take a look at a more sophisticated instance list that handles two different versions of Plone:

```
playbook_plones:
  plone_instance_name: primary_plone
  plone_target_path: /opt/primary_plone
  plone_var_path: /var/local/primary_plone
  plone_major_version: '5.0'
```

```
  plone_version: '5.0'
  plone_initial_password: admin
  plone_zeo_port: 5100
  loadbalancer_port: 4080
  plone_client_base_port: 5081
  plone_client_count: 2
  plone_create_site: no
  webserver_virtualhosts:
    - hostname: plone.org
      zodb_path: /plone_org
      aliases:
        - www.plone.org
- plone_instance_name: secondary_plone
  plone_target_path: /opt/secondary_plone
  plone_var_path: /var/local/secondary_plone
  plone_major_version: '4.3'
  plone_version: '4.3.7'
  plone_initial_password: admin
  plone_zeo_port: 4100
  loadbalancer_port: 4080
  plone_client_base_port: 4081
  plone_client_count: 3
  plone_create_site: no
  webserver_virtualhosts:
    - hostname: plone.com
      zodb_path: /plone_com
      aliases:
        - www.plone.com
    - hostname: plone.com
      zodb_path: /plone_com
      address: 92.168.1.150
      port: 443
      protocol: https
      certificate_file: /thiscomputer/path/mycert.crt
      key_file: /thiscomputer/path/mycert.key
```

19.1 Moving beyond four

Ansible doesn't offer a way to iterate a role over a sequence, so the max count of four is hard-coded into the playbook. Read the playbook and it will be obvious how to change the limit by editing it.