Deploy

Feb 27, 2020

Contents

This documentation is split into:

Development Guides Instructions on getting set-up and making changes.

Deployment Guides Instructions for specific deployment tasks.

User Guides Documentation for our deployments of services.

Reference Infrequently accessed reference material.

CHAPTER 1

Development Guides

Follow the *Get started* guide before following any of the *Deployment Guides*. If you're new to Salt, *Learn Salt*.

To make changes to the deploy repository, read the *Update server configurations* guide. If you run into trouble, read the *Troubleshoot* guide.

1.1 Development Guides

1.1.1 Get started

1. Install Salt

On macOS, using Homebrew, install Salt and Salt SSH with:

```
brew install salt
```

If you encounter issues, try installing with pip:

```
pip install salt salt-ssh
```

For other operating systems and package managers, see this page (or this page) to install a recent version (2019 or later).

You must use Salt with Python 3. If your system package uses Python 2, install salt-ssh with pip into a Python 3 virtual environment.

2. Clone repositories

You must first have access to two private repositories. Contact an owner of the open-contracting organization on GitHub for access. Then:

```
git clone git@github.com:open-contracting/deploy.git
git clone git@github.com:open-contracting/deploy-salt-private.git deploy/salt/private
git clone git@github.com:open-contracting/deploy-pillar-private.git deploy/pillar/
↪private
```

Note: This documentation is for working with OCP servers. If you want to work with other servers, then instead of cloning the private repositories, copy and edit the template directories:

```
git clone git@github.com:open-contracting/deploy.git
cp -r deploy/pillar/private-templates deploy/pillar/private
cp -r deploy/salt/private-templates deploy/salt/private
```

3. Add public key to remote servers

Add your public key to `salt/private/authorized_keys/root_to_add`, e.g.:

```
cat ~/.ssh/id_rsa.pub >> salt/private/authorized_keys/root_to_add
git commit salt/private/authorized_keys/root_to_add -m "Add public key"
git push origin master
```

Then, add this public key to all servers:

```
salt-ssh -i '*' state.sls_id root_authorized_keys_add core
```

4. Configure Salt for non-root user

Unless your local user is the root user, run:

```
./script/setup
```

This script assumes your SSH keys are `~/.ssh/id_rsa` and `~/.ssh/id_rsa.pub`.

You're now ready to *Deploy a service*.

1.1.2 Learn Salt

We use Salt (a.k.a. SaltStack) to deploy apps to servers, and to otherwise manage servers.

We use Agentless Salt (i.e. using the `salt-ssh` command). This avoids having to run Salt minions on servers, and requires only SSH to connect to the server and Python to run operations on it.

To orient you to the repository: When you run the `salt-ssh` command, it reads `Saltfile`, which directs it to read the `salt-config` directory. `salt-config/master` directs it to read the `salt` and `pillar` directories. The `top.sls` file in each directory serves as an index to the other SLS files, which in turn refer to the files in sub-directories.

Read Salt Best Practices and Salt Formulas Style before editing this repository.

1.1.3 Update server configurations

1. Update private templates

If you add, remove or rename a file or variable in `pillar/private` or `salt/private`, replicate the changes in `pillar/private-templates` or `salt/private-templates`.

This allows others to use this repository to, for example, deploy Kingfisher to their own servers.

2. Test changes

To preview what is going to change, use test=True, for example:

```
salt-ssh 'ocds-docs-live' state.apply test=True
```

To preview changes to a Pillar file, run, for example:

```
salt-ssh 'ocds-docs-live' pillar.items
```

To compare Jinja2 output after refactoring but before committing, use `script/diff` to compare a full state or one SLS file, for example:

```
./script/diff ocds-docs-staging
./script/diff ocds-docs-staging ocds-docs-common
```

If you get the error, `An Exception occurred while executing state.show_highstate: 'list' object has no attribute 'values'`, run `state.apply test=True` as above. You might have conflicting IDs.

Using a testing virtual host

To test changes to the Apache files for the *OCDS Documentation* (for example, to test new redirects or proxy settings):

1. Make changes inside `{% if testing %}` blocks in the config files
2. *Deploy* the OCDS Documentation
3. To test manually, visit the testing version of the live website or staging website
4. To test automatically, run (using the fish shell):

```
pip install -r requirements.txt
env FQDN=testing.live.standard.open-contracting.org pytest
```

Update the tests if you changed the behavior of the Apache files.

Once satisfied, move the changes outside `{% if testing %}` blocks. After deployment, the tests should pass if `FQDN` is omitted or set to standard.open-contracting.org.

Using a virtual machine

1. Create a virtual machine
2. Get the virtual machine's IP address
   - If using VirtualBox, run (replacing `VMNAME`):

     ```
     VBoxManage guestproperty get VMNAME "/VirtualBox/GuestInfo/Net/0/V4/IP"
     ```

3. Update the relevant target in `salt-config/roster` to point to the virtual machine's IP address
4. In the relevant Pillar file, change `https` to `no`, if certbot is used to enable HTTPS
5. Edit `/etc/hosts` to map the virtual machine's IP address to the service's hostname
6. Deploy to the virtual machine and test

Note that Python errors that occur on the virtual machine might still be reported to Sentry. The `server_name` tag in any error reports is expected to be different, but the error reports might still confuse other developers who don't know to check that tag.

3. Review code

For context, for other repositories, work is done on a branch and tested on a local machine before a pull request is made, which is then tested on Travis CI, reviewed and approved before merging.

However, for this repository, in some cases, it's impossible to test changes to server configurations, for example: if SSL certificates are involved (because certbot can't verify a virtual machine), or if external services like Travis are involved. In other cases, it's too much effort to setup a test environment in which to test changes.

In such cases, the same process is followed as in other repositories, but without the benefit of tests.

In entirely uncontroversial or time-sensitive cases, work is done on the `master` branch, deployed to servers, and committed to the `master` branch once successful. In cases where the changes require trial and error, the general approach is discussed in a GitHub issue, and then work is done on the `master` branch as above. Developers can always request informal reviews from colleagues.

Take extra care when making larger changes or when making changes to higher-priority apps.

Change server name

If the virtual host uses HTTPS, you will need to acquire SSL certificates for the new server name and remove the SSL certificates for the old server name.

1. Change the `ServerName`
2. In the relevant Pillar file, change `https` to `certonly`
3. *Deploy the service*
4. In the relevant Pillar file, change `https` to `force` or `both`
5. Remove the old SSL certificates, for example:

```
salt-ssh 'ocds-docs-staging' file.remove /etc/letsencrypt/live/dev.standard.open-
↪contracting.org
```

To check for old SSL certificates that were previously not removed, run:

```
salt-ssh '*' cmd.run 'ls /etc/letsencrypt/live'
```

Remove content

If you delete a file, service, package, user, authorized key, Apache module, or virtual host from a file, it will not be removed from the server. To remove it, after you *deploy*:

Delete an authorized key

1. Cut it from `salt/private/authorized_keys/root_to_add`
2. Paste it into `salt/private/authorized_keys/root_to_remove`
3. Run:

```
salt-ssh '*' state.sls_id root_authorized_keys_add core
salt-ssh '*' state.sls_id root_authorized_keys_remove core
```

4. Delete it from `salt/private/authorized_keys/root_to_remove`

Delete a file

Run, for example:

```
salt-ssh 'ocds-docs-staging' file.remove /path/to/file_to_remove
```

Delete a service

Stop and disable the service.

To stop and disable the `icinga2` service on the `ocds-docs-staging` target, for example:

```
salt-ssh 'ocds-docs-staging' service.stop icinga2
salt-ssh 'ocds-docs-staging' service.disable icinga2
```

If you deleted the `uwsgi` service, also run, for example:

```
salt-ssh 'cove-live-ocds-3' file.remove /etc/uwsgi/apps-available/cove.ini
salt-ssh 'cove-live-ocds-3' file.remove /etc/uwsgi/apps-enabled/cove.ini
```

Delete a package

Remove a package and its configuration files, and remove any of its dependencies that are no longer needed.

To scrub Icinga-related packages from the `ocds-docs-staging` target, for example:

```
salt-ssh 'ocds-docs-staging' pkg.purge icinga2,nagios-plugins,nagios-plugins-contrib
salt-ssh 'ocds-docs-staging' pkg.autoremove list_only=True
salt-ssh 'ocds-docs-staging' pkg.autoremove purge=True
```

Then, login to the server and check for and delete any remaining packages, files or directories relating to the package, for example:

```
dpkg -l | grep icinga
dpkg -l | grep nagios
ls /etc/icinga2
ls /usr/lib/nagios
```

Delete an Apache module

1. Add a temporary Salt ID, for example:

```
headers:
    apache_module.disabled
```

2. Deploy the relevant service, for example:

```
salt-ssh 'toucan' state.apply
```

3. Remove the temporary salt ID

Delete a virtual host

Run, for example:

```
salt-ssh 'cove-ocds-live-2' file.remove /etc/apache2/sites-enabled/cove.conf
salt-ssh 'cove-ocds-live-2' file.remove /etc/apache2/sites-available/cove.conf
salt-ssh 'cove-ocds-live-2' file.remove /etc/apache2/sites-available/cove.conf.include
```

You might also delete the SSL certificates as when *changing server name*.

Track upstream

The files in this repository were originally in the opendataservices-deploy repository. Some common files might have improvements in the original repository. To check for updates, run:

```
git clone git@github.com:OpenDataServices/opendataservices-deploy.git
cd opendataservices-deploy
git log --name-status setup_for_non_root.sh updateToMaster.sh Saltfile pillar/common_
↪pillar.sls salt-config/master salt/apache.sls salt/apache/000-default.conf salt/
↪apache/000-default.conf.include salt/apache/_common.conf salt/apache/cove.conf salt/
↪apache/cove.conf.include salt/apache/prometheus-client.conf salt/apache/prometheus-
↪client.conf.include salt/apache/robots_dev.txt salt/apt/10periodic salt/apt/
↪50unattended-upgrades salt/core.sls salt/cove.sls salt/letsencrypt.sls salt/lib.sls␣
↪salt/nginx/redash salt/prometheus-client-apache.sls salt/prometheus-client/
↪prometheus-node-exporter.service salt/system/ocdskingfisher_motd salt/uwsgi.sls␣
↪salt/uwsgi/cove.ini
```

- `setup_for_non_root.sh` corresponds to `script/setup`
- `updateToMaster.sh` corresponds to `script/update`
- `salt-config/roster`, `pillar/top.sls` and `salt/top.sls` are common files, but are unlikely to contain improvements

This repository has all improvements up to September 30, 2019.

1.1.4 Troubleshoot

Pillar file gotchas

- If unquoted, `yes`, `no`, `true` and `false` are parsed as booleans. Use quotes to parse as strings.

- A blank value is parsed as `None`. Use the empty string `''` to parse as a string.
- Below, if `a` is equal to an empty string, then `b` will be `None`:

```
{% set extracontext %}
b: {{ a }}
{% endset %}
```

Instead, surround it in quotes:

```
{% set extracontext %}
b: "{{ a }}"
{% endset %}
```

Check history

If you don't understand why a configuration exists, it's useful to check its history.

The files in this repository were originally in the opendataservices-deploy repository. You can browse that repository from before the switchover (August 5, 2019). That repository was itself re-organized at different times. You can browse before moving content from *.conf to *.conf.include (June 5, 2019).

CHAPTER 2

Deployment Guides

If you need to perform a specific deployment task, follow the relevant how-to guide.

All changes to servers should be made using Salt to ensure that changes are documented and reproducible; changes should not be made manually, which is undocumented and error-prone.

2.1 Deployment Guides

2.1.1 Get deploy token

Before performing any deployment task, run the *Setup* tasks. Once done, run the *Cleanup* tasks.

Setup

1. Update deploy repositories

Ensure the `deploy`, `salt/private`, and `pillar/private` repositories are on the `master` branch and are up-to-date. You can run this convenience script to run the appropriate `git` commands:

```
./script/update
```

Check the output in case there are any issues switching to the `master` branch or any conflicts pulling from GitHub.

2. Check if Kingfisher is busy

Note: Skip unless you're working on Kingfisher.

1. *Access Scrapyd's web interface*, click "Jobs" and look under "Running". If any spiders are running, don't deploy without the consent of helpdesk analysts.

2. Connect to the Kingfisher server as the `root` user:

```
ssh root@scrape.kingfisher.open-contracting.org
```

3. Check if any *long-running tasks* are running. If any would be interrupted by the deployment, don't deploy without the consent of helpdesk analysts.

```
for i in root ocdskfs ocdskfp; do echo $i; su $i -c "tmux ls"; done
```

3. Get deploy token

Only one person should be making changes to a server at once. To implement this rule, the Deploy token wiki page indicates who holds the 'deploy token'. Whoever holds the deploy token is the only person who can make changes to *any* server, until the deploy token is released. If the wiki page has "Held by: <NAME>", that person holds the token; if it has "Held by: nobody", then the token is released. To hold the token:

1. Go to the Deploy token wiki page
   - If "Held by" is followed by a person's name, wait until the deploy token is released
2. Click the "Edit" link, replace "nobody" with your name and click the "Save" button
   - If this results in an edit conflict, wait until the deploy token is released

Cleanup

1. Release deploy token

1. Go to the Deploy token wiki page
2. Click "Edit", replace your name with "nobody", add an entry under History, and click "Save". If any servers were rebooted, add a note to the entry.

2.1.2 Deploy a service

1. Watch Salt activity

Note: This step is optional.

1. Find the server's IP or fully-qualified domain name in the roster:

```
cat salt-config/roster
```

2. Open a secondary terminal to connect to the server as root, for example:

```
ssh root@live.docs.opencontracting.uk0.bigv.io
```

3. Watch the processes on the server:

```
watch -n 1 pstree
```

4. Access your primary terminal

2. Run Salt function

To deploy a service, indicate the desired target and the `state.apply` function, for example:

```
salt-ssh -i 'ocds-docs-staging' state.apply
```

The `-i` flag disables StrictHostKeyChecking, which avoids an extra prompt the first time you connect to a host.

If the output has an error of `Unable to detect Python-2 version`, you don't have Python 2.7 in your PATH. To fix this, if you use `pyenv`, run, for example:

```
pyenv shell system
```

The `state.apply` function often completes within one minute. You can ignore this warning: `DeprecationWarning: encoding is deprecated, Use raw=False instead`.

In the secondary terminal, to monitor what Salt is doing, look at the lines below these:

```
|-sshd-+-sshd---bash---watch
|      |-sshd---bash---watch---watch---sh---pstree
```

3. Check Salt output

Look for these lines at the end of the output in the primary terminal:

```
Summary for ocds-docs-staging
-------------
Succeeded: ## (changed=#)
Failed:     0
```

Then:

1. Check that the app is still responding in your web browser.
2. If there are any failed states, look for each in the output (red text) (or search for `Result: False`) and debug.
3. If there are any changed states, look for each in the output (blue text) (or grep for `Changes: \n[^\n-]`) and verify the changes.

Common changed states are:

Function: service.running, ID: apache2 Apache was reloaded

Function: cmd.run, ID: prometheus-client-apache-password This change is a false positive

For a Django app, common changed states are:

Function: git.latest A new commit was deployed

Function: virtualenv.managed This change is a false positive

Function: cmd.run, Name: . .ve/bin/activate; python manage.py migrate –noinput Django migrations were applied

Function: cmd.run, Name: . .ve/bin/activate; python manage.py collectstatic –noinput Static files were copied

Function: service.running, ID: uwsgi uWSGI was reloaded

4. Manual cleanup

If you *changed the server name* or *deleted a service, package, user, file, or authorized key*, follow the linked steps to cleanup manually.

5. Close the secondary terminal

Note: Skip this step if you didn't watch Salt activity on the remote server.

1. Stop watching the processes, e.g. with `Ctrl-C`
2. Disconnect from the server, e.g. with `Ctrl-D`

2.1.3 Create a server

A server is created either when a service is moving to a new server, or when a service is being introduced.

As with other deployment tasks, do the *setup tasks* before (and the cleanup tasks after) the steps below.

1. Create the server

Create the server via the *host*'s interface.

Bytemark

1. Login to Bytemark
2. Click "Servers" and "Add a cloud server"
    1. Enter a *Name*
    2. Select a *Group*
    3. Set *Location* to "York"
    4. Set *Server Resources*
    5. Set *Operating system* to "Ubuntu 18.04 (LTS)"
    6. Check *Enable backups*
    7. Set *Take a backup every* to 7 days
    8. Set *Starting on* to the following Thursday at a random time before 10:00 UTC
    9. Set *Root user has* to "SSH key (+ Password)" and enter your public key
    10. Click "Add this server"
3. Wait for the server to boot (a few minutes)
4. Click "Info" and copy the "Hostname/SSH"

2. Deploy the service

1. Connect to the server over SSH
    1. Change the password of the root user
    2. Install packages for Agentless Salt:

    ```
    apt-get install python-concurrent.futures python-msgpack
    ```

2. Update this repository
    1. Add the server to `salt-config/roster`, using the hostname from above
    2. Add a target to `salt/top.sls`, if necessary, and include the `prometheus-client-apache` state
    3. Add a target to `pillar/top.sls`, if necessary
    4. Add any states, if necessary
    5. If a service is moving to the new server, update occurrences of the old server's hostname and IP address, as needed
3. Upgrade packages (can be slow):

    ```
    salt-ssh TARGET pkg.upgrade refresh=True dist_upgrade=True
    ```

4. Reboot the server:

    ```
    salt-ssh TARGET system.reboot
    ```

5. *Deploy the service* (can be slow)

3. Update external services

1. *Add the server to Prometheus*
2. Add (or update) the service's DNS entries in GoDaddy
3. Add (or update) the service's row in the Health of software products and services spreadsheet
4. Add (or update) managed passwords, if appropriate

If the service is being introduced:

1. Add its downtime monitor to UptimeRobot
2. Add its error monitor to Sentry
3. Add the analytics tag for Google Analytics, if appropriate

2.1.4 Manage server packages

All servers use the unattended-upgrades package for security updates. On development/staging servers, the package is configured to automatically reboot the server, according to the `automatic_reboot` variable in Pillar. For other servers, administrators perform the steps below on a weekly basis.

The commands below will run on all servers. To run on specific servers, replace `'*'` with either a glob pattern like `'ocds-docs-*'` or with a comma-separated list using the `-L` flag, like `-L ocds-docs-live,ocds-docs-staging`.

As with other deployment tasks, do the *setup tasks* before (and the cleanup tasks after) the steps below.

1. List upgrades

```
salt-ssh '*' pkg.list_upgrades
```

Consider whether any upgrades are backwards-incompatible or have post-installation steps.

2. Upgrade packages

```
salt-ssh '*' pkg.upgrade
```

Monitor the output for relevant messages.

3. Reboot

1. Find the servers that need to be rebooted:

   ```
   salt-ssh '*' file.file_exists /var/run/reboot-required
   ```

2. Reboot the servers that need to be rebooted. For example:

   ```
   salt-ssh -L server-one,server-two system.reboot
   ```

   Sometimes, this command hangs, waiting for a response from a server that is already shutting down. Simply wait 30 seconds and stop the command.

3. Check the servers have rebooted without issue:

   ```
   salt-ssh '*' file.file_exists /var/run/reboot-required
   ```

   All servers should respond with `False`.

2.1.5 Delete a server

A server is deleted either when a service is moving to a new server (*create the new server*, first), or when a service is being retired.

As with other deployment tasks, do the *setup tasks* before (and the cleanup tasks after) the steps below.

1. If appropriate, notify relevant users of the change
2. Remove the server from Prometheus
3. Shutdown the server via the *host*'s interface
4. Remove all occurrences of the server's FQDN and IP address from this repository

If the service is being retired:

1. Remove its configuration from this repository
2. Remove its DNS entries from GoDaddy
3. Remove its downtime monitor from UptimeRobot
4. Remove its error monitor from Sentry
5. Remove its row from the Health of software products and services spreadsheet
6. Remove its managed passwords, if appropriate

Finally:

1. Cancel the server via the host's interface

2.1.6 Maintain a server

For tasks related to upgrading packages, see *Manage server packages*.

Perform a long-running operation

If an operation will take a long time to run, run it in a terminal multiplexer (`tmux`), in case you lose your connection to the server. To open a session in `tmux`, use this command, replacing `initials-task-description` with your initials and a short description of your task. By including your initials, it is easy for others to determine to whom the session belongs – especially if you forget to close it.

```
tmux new -s initials-task-description
```

If you lose your connection to the server, re-attach to your session with:

```
tmux attach-session -t initials-task-description
```

To manually detach from a session, press `Ctrl-b`, release both keys, then press `d`.

If you forget the name of your session, list all sessions with:

```
tmux ls
```

Clean root user directory

1. Run:

   ```
   salt-ssh '*' cmd.run 'ls'
   ```

2. Leave any `post.install.log` files
3. Delete any `index.html*` files
   - These are created when a developer runs `wget` commands to e.g. test proxy settings.

Auto-remove packages

To show the packages that were automatically installed and are no longer required:

```
salt-ssh 'ocds-docs-staging' pkg.autoremove list_only=True
```

To remove these, run:

```
salt-ssh 'ocds-docs-staging' pkg.autoremove purge=True
```

To show the packages that were removed but not purged, run:

```
salt-ssh '*' pkg.list_pkgs removed=True
```

Check mail

Find mailboxes with mail across servers:

```
salt-ssh '*' cmd.run 'find /var/mail -type f -not -size 0'
```

Connect to a server, for example:

```
ssh root@process.kingfisher.open-contracting.org
```

Open the mailbox:

```
mail -f /var/mail/root
```

You might see a lot of repeat messages.

Here are common commands:

- number: open that message
- `h`: show a screen of messages
- `z`: go to the next screen
- `d 5-10`: delete the messages 5 through 10
- `d *`: delete all messages
- `q`: save changes and exit
- `x`: exit without saving changes

In most cases, all messages can be ignored and deleted. Relevant messages might include:

Failed cron jobs Try to correct the failure

Failed attempts to use sudo If the attempt is not attributable to a team member, discuss security measures

Check that no messages were saved:

```
ls ~/mbox
```

Upgrade Ubuntu

To determine the current releases, run:

```
salt-ssh '*' cmd.run 'lsb_release -a'
```

To check the long term support of the releases, consult the Ubuntu documentation.

2.1.7 Prometheus tasks

Monitor a service

If the `prometheus-client-apache` state applies to the target, Node Exporter is served from port 80 (see `pillar/private/prometheus_pillar.sls`), or a port set by the `prometheus.client_port` variable in the target's Pillar file. It is served from the domain configured in the server's `/etc/apache2/sites-enabled/prometheus-client.conf` file (see `salt/prometheus-client-apache.sls`), and/or from the server's FQDN or IP address if the port isn't 80.

If the `prometheus-client-nginx` state applies to the target, Node Exporter is served from port 9158 (see `salt/nginx/prometheus-client`). It is served from the server's FQDN.

1. For a Hetzner server, set the `prometheus.client_port` variable in the target's Pillar file to `7231`, and *re-deploy* the service.
2. Check that Node Exporter is accessible and that its "Metrics" page displays metrics.

   For Apache, open, for example, http://prom-client.live.docs.opencontracting.uk0.bigv.io for a Bytemark server or http://95.217.76.74:7231 for a Hetzner server.

   For Nginx, open, for example, http://live.redash.opencontracting.uk0.bigv.io:9158.

   The username is `prom`. The password is set by the `prometheus.client_password` variable in the `pillar/private/prometheus_pillar.sls` file.
3. If Node Exporter isn't accessible, edit the `prometheus.client_port` and/or `prometheus.client_fqdn` variables in the target's Pillar file as needed, and *re-deploy* the service.
4. Add a job to `salt/private/prometheus-server-monitor/conf-prometheus.yml`, following the pattern of others.
5. *Deploy* the Prometheus service.
6. Check that the job is "UP" on Prometheus' Targets page.

Note: Bytemark assigns hostnames like `<server>.<group>.opencontracting.uk0.bigv.io` to its servers, and implements wildcard DNS for any subdomains. By default, Node Exporter is served from `prom-client.<hostname>` on port 80, which works for Bytemark servers without additional configuration. For Hetzner servers, additional configuration is needed. Instead of adding a DNS entry and setting the `prometheus.client_fqdn` variable, we simply set the `prometheus.client_port` variable and access Node Exporter by the server's IP address.

2.1.8 OCDS documentation tasks

Add a new language

1. In `salt/apache/ocds-docs-live.conf.include` and `salt/apache/ocds-docs-staging.conf.include`, add the new language in the `options` variable.
2. In `tests/test_docs.py`, update the `languages` variable.

Add a new profile

Below, substitute `{root}`, `{latest-branch}`, `{minor-branch}` and `{dev-branch}`. For example: `ppp`, `latest` `1.0` and `1.0-dev`.

1. Edit `salt/ocds-docs/robots_live.txt`
2. For Googlebot, add:

```
Allow: /profiles/{root}/{latest-branch}
```

3. If the profile publishes schema files, also add:

```
Allow: /profiles/{root}/schema
Allow: /profiles/{root}/extension
```

4. If the profile has a single branch, skip these steps. Otherwise, for all user agents, add:

```
Disallow: /profiles/{root}/{minor-branch}
Disallow: /profiles/{root}/{dev-branch}
```

5. If the profile has older versions, also add, for each `{old-version}`:

```
Disallow: /profiles/{root}/{old-branch}
```

Publish draft documentation

To configure a documentation repository to push builds to the *staging server*:

1. Access the repository's Travis page
2. Click "More options" and "Settings"
3. Set the private key:
    1. Enter "PRIVATE_KEY" in the first input under "Environment Variables"
    2. Get the `ocds-docs` user's private key (deploy-docs.sh will restore the newlines and spaces):

```
cat salt/private/ocds-docs/ssh_authorized_keys_from_travis_private | tr '\n' '
↪#' | tr ' ' '_'
```

    3. Enter the private key in the second input
    4. Click "Add"
4. Set the search secret:
    1. Enter "SEARCH_SECRET" in the first input under "Environment Variables"
    2. Get the value of the `ocds_secret` key in `pillar/private/standard_search_pillar.sls`
    3. Enter it in the second input
    4. Click "Add"

Publish released documentation

If this is the first numbered version of a profile, in its `docs/_templates/layout.html`, add:

```
{% block version_options %}
<!--#include virtual="/includes/version-options-profiles-{root}.html" -->
{% endblock %}
```

In any case, once the build passes on Travis for the live branch of the documentation:

1. Copy the files to the live server

Each deployment of each branch is given its own directory on the live server, named according to the format `branch-date-sequence`, for example: `1.1-2017-08-08-2`. A symlink named after each branch links to the directory to publish for that branch. In this way, you can rollback a deployment by changing the symlink.

Set environment variables, for example:

```
SUBDIR=           # include a trailing slash (leave empty for OCDS documentation)
VER=1.1           # set to the branch to deploy (not to the tag)
DATE=$(date +%F)  # assuming the build completed today; otherwise, set accordingly
SEQ=1             # increment for each deploy on the same day
```

For a profile, set `SUBDIR` to, for example, `profiles/ppp/`. For OC4IDS, set it to `infrastructure/`.

Copy files from the staging server to your local machine:

```
rsync -avP root@staging.standard.open-contracting.org:/home/ocds-docs/web/${SUBDIR}${VER}/ ${VER}-${DATE}-${SEQ}
```

Copy files from your local machine to the live server:

```
rsync -avP ${VER}-${DATE}-${SEQ} root@live.standard.open-contracting.org:/home/ocds-docs/web/${SUBDIR}/
```

Symlink the branch:

```
ssh root@live.standard.open-contracting.org "ln -nfs ${VER}-${DATE}-${SEQ} /home/ocds-docs/web/${SUBDIR}${VER}"
```

If the branch is for the latest version of the documentation, repeat this step with `VER=latest`.

2. Copy the schema and ZIP file into place

Note: You can skip this step if you are not releasing a new major, minor or patch version.

Connect to the server:

```
ssh root@live.standard.open-contracting.org
```

Set environment variables, for example:

```
SUBDIR=           # include a trailing slash (leave empty for OCDS documentation)
VER=1.1           # set to the branch as above
RELEASE=1__1__1   # set to the full release tag name
```

For a profile, set `SUBDIR` to, for example, `profiles/ppp/`. For OC4IDS, set it to `infrastructure/`.

For the **OCDS** and **OC4IDS** documentation, run:

```
# Create the directory for the release.
mkdir /home/ocds-docs/web/${SUBDIR}schema/${RELEASE}/

# Copy the schema and codelist files.
cp -r /home/ocds-docs/web/${SUBDIR}${VER}/en/*.json /home/ocds-docs/web/${SUBDIR}schema/${RELEASE}/
cp -r /home/ocds-docs/web/${SUBDIR}${VER}/en/codelists /home/ocds-docs/web/${SUBDIR}schema/${RELEASE}/

# Create a ZIP file of the above.
cd /home/ocds-docs/web/${SUBDIR}schema/
zip -r ${RELEASE}.zip ${RELEASE}
```

The files are then visible at e.g. https://standard.open-contracting.org/schema/1__1__1/.

For a **profile's** documentation, run:

```
# Create the profile and patched directories for the release.
mkdir -p /home/ocds-docs/web/${SUBDIR}extension/${RELEASE}/ /home/ocds-docs/web/$
↪{SUBDIR}schema/${RELEASE}/

# Copy the profile's schema and codelist files.
cp -r /home/ocds-docs/web/${SUBDIR}${VER}/en/*.json /home/ocds-docs/web/${SUBDIR}
↪extension/${RELEASE}/
cp -r /home/ocds-docs/web/${SUBDIR}${VER}/en/codelists /home/ocds-docs/web/${SUBDIR}
↪extension/${RELEASE}/

# Create a ZIP file of the above.
cd /home/ocds-docs/web/${SUBDIR}extension/
zip -r ${RELEASE}.zip ${RELEASE}

# Copy the patched schema and codelist files.
cp -r /home/ocds-docs/web/${SUBDIR}${VER}/en/_static/patched/* /home/ocds-docs/web/$
↪{SUBDIR}schema/${RELEASE}/
```

3. Update this repository

Note: You can skip this step if you are not releasing a new major, minor or patch version.

Below, substitute `{root}`, `{latest-branch}`, `{dev-branch}`, `{formatted-dev-branch}`, `{version}` and `{name}`. For example: `ppp`, `latest`, `1.0-dev`, `1.0 Dev`, `1.0.0.beta` and `OCDS for PPPs`.

If this is the first numbered version of a profile:

1. *Update salt/ocds-docs/robots_live.txt*.
2. In `salt/apache/ocds-docs-live.conf.include`, add the profile's latest branch, minor series and languages in the `options` variable.
3. In `tests/test_docs.py`, update the `versions`, `languages` and `banner_live` variables.
4. Add a `salt/ocds-docs/includes/version-options-profiles-{root}.html` file to this repository:

```
<option>Version</option>
<optgroup label="Live">
<option value="{latest-branch}">{version} ({latest-branch})</option>
</optgroup>
<optgroup label="Development Branches">
<option value="{dev-branch}">{formatted-dev-branch}</option>
</optgroup>
```

5. Add a `salt/ocds-docs/includes/banner_staging_profiles_{root}.html` file to this repository:

```
<div class="oc-fixed-alert-header">
    This is a development copy of the {name} docs, the <a href="/profiles/ppp/
↪{root}/en/">latest live version is here</a>.
</div>
```

Otherwise:

1. In the appropriate `salt/ocds-docs/includes/version-options*.html` file, update the version number in the text of the first `option` element.

If this is a new major or minor version:

1. In `salt/ocds-docs/robots_live.txt`, disallow the minor branch and its dev branch, for example:

```
Disallow: /1.2
Disallow: /1.2-dev
```

2. In `salt/apache/ocds-docs-live.conf.include`, add the minor series in the `options` variable, and add a new `Location` directive like:

```
<Location /1.1/>
    SetEnv BANNER /includes/banner_old.html
</Location>
```

3. In `tests/test_docs.py`, update the `versions`, `banner_live` and `banner_old` variables.
4. In the appropriate `salt/ocds-docs/includes/banner_staging*.html` file and `salt/ocds-docs/includes/banner_old*.html>` file (if any), update the minor series.
5. In the appropriate `salt/ocds-docs/includes/version-options*.html` file, add an `option` element to the "Live" `optgroup` for the previous minor series and previous version number, for example:

```
<option value="0.9">0.9.2</option>
```

2.1.9 Kingfisher tasks

Deploy Kingfisher Process without losing Scrapy requests

This should match `salt/ocdskingfisherprocess.sls` (up-to-date as of 2019-12-19). You can `git log salt/ocdskingfisherprocess.sls` to see if there have been any relevant changes, and update this page accordingly.

This assumes that there have been no changes to `requirements.txt`. If you are adding an index, altering a column, updating many rows, or performing another operation that locks tables or rows for longer than uWSGI's `harakiri` setting, this might interfere with an ongoing collection (until queues are fully implemented).

Below, the two key operations are reloading uWSGI with the new application code, and migrating the database.

It's possible for requests to arrive after uWSGI reloads and before the database migrates. If the new application code is not backwards-compatible with the old database schema, the requests might error. If, on the other hand, your old application code is forwards-compatible with the new database schema, then reload uWSGI after migrating the database, instead of before.

`service uwsgi reload` runs `/etc/init.d/uwsgi reload`, which sends the SIGHUP signal to the master uWSGI process, which causes it to gracefully reload and not lose any requests from Scrapy.

As with other deployment tasks, do the *setup tasks* before (and the cleanup tasks after) the steps below.

1. Connect to the server as the `ocdskfp` user and change to the working directory:

```
ssh ocdskfp@process.kingfisher.open-contracting.org
cd ocdskingfisherprocess
```

2. Check that you won't deploy more commits than you intend, for example:

```
git fetch
# From https://github.com/open-contracting/kingfisher-process
#    d8736f4..173dcf2  master                                   -> origin/master
git log d8736f4..173dcf2
```

3. Update the code:

```
git pull --rebase
```

4. In a new terminal, connect to the server as the `root` user, reload uWSGI, then close your connection to the server:

```
ssh root@process.kingfisher.open-contracting.org
service uwsgi reload
```

5. In the original terminal, open a terminal multiplexer, in case you lose your connection while migrating the database. You can re-attach to the session with `tmux attach-session -t deploy`:

```
tmux new -s deploy
```

6. If workers are likely to interfere with a migration (e.g. inserting new rows that meet the criteria for an update), comment out the lines that start them in the cron table and kill them:

```
crontab -e
pkill -f ocdskingfisher-process-cli
```

7. Migrate the database (log the time, in case you need to retry). Alembic has no verbose mode for upgrades. To see the current queries, open another terminal, open a PostgreSQL shell, and run `SELECT pid, state, wait_event_type, query FROM pg_stat_activity;`. If a migration query has a `wait_event_type` of `Lock`, look for queries that block it (for example, long-running DELETE queries). To stop a query, run `SELECT pg_cancel_backend(PID)`, where `PID` is the `pid` of the query.

```
. .ve/bin/activate
date
python ocdskingfisher-process-cli upgrade-database
date
```

8. Uncomment the lines that start the workers in the cron table:

```
crontab -e
```

9. Close the session with `Ctrl-D` and close your connection to the server.

CHAPTER 3

User Guides

This section contains documentation specific to our deployment of a given service. For generic documentation of a given service that we authored, follow the *Docs* links on this page.

3.1 User Guides

3.1.1 Kingfisher Scrape

Read the Kingfisher Scrape documentation, which cover general usage.

Access Scrapyd's web interface

Open http://scrape.kingfisher.open-contracting.org

Connect to the Kingfisher Scrape server

Connect to the server as the `ocdskfs` user:

```
ssh ocdskfs@scrape.kingfisher.open-contracting.org
```

Collect data with Kingfisher Scrape

1. *Connect to the server*
2. Schedule a crawl and set its note and any other spider arguments. For example, replace `spider_name` with a spider's name and `NAME` with your name:

```
curl http://localhost:6800/schedule.json -d project=kingfisher -d spider=spider_
↪name -d note="Started by NAME."
```

Update spiders in Kingfisher Scrape

1. Merge your changes to the master branch of the kingfisher-scrape repository.
2. Connect to the server as the `ocdskfs` user and change to the working directory:

```
ssh ocdskfs@scrape.kingfisher.open-contracting.org
cd ocdskingfisherscrape
```

3. Pull your changes into the local repository:

```
git pull --rebase
```

4. Activate the virtual environment and Update the project's requirements:

```
source .ve/bin/activate
pip install -r requirements.txt
```

5. Deploy the spiders:

```
scrapyd-deploy
```

Access Scrapyd's crawl logs

From a browser, click on a "Log" link from the jobs page, or open Scrapyd's logs page for the kingfisher project.

From the command-line, connect to the server as the `ocdskfs` user, and change to the logs directory for the `kingfisher` project:

```
ssh ocdskfs@scrape.kingfisher.open-contracting.org
cd scrapyd/logs/kingfisher
```

Scrapy statistics are extracted from the end of each log file every hour on the hour, into a new file ending in `_report.log` in the same directory as the log file. Access as above, or, from the jobs page:

- Right-click on a "Log" link.
- Select "Copy Link" or similar.
- Paste the URL into the address bar.
- Change `.log` at the end of the URL to `_report.log` and press Enter.

3.1.2 Prometheus

Monitor

Access the monitoring service. The username is `prom`. The password is set by the `prometheus.server_password` variable in the `pillar/private/prometheus_pillar.sls` file.

The landing page lets you query the collected data. For example:

- Load averages
- Blocked processes
- Disk usage
- Disk I/O

- I/O wait
- RAM usage
- Swap usage

Other relevant pages are:

- Alerts
- Targets

Read Prometheus' documentation to learn more.

Alert manager

Access the alerting service. The username is `prom`. The password is set by the `prometheus.alertmanager_password` variable in the `pillar/private/prometheus_pillar.sls` file.

Whereas the monitoring service configures alerts, the alerting service sends alerts. Alerts are sent to the recipients set in `salt/private/prometheus-server-alertmanager/conf-alertmanager.yml`.

You can temporarily "silence" alerts, when you know your actions will trigger those alerts: for example, when shutting down a server.

CHAPTER 4

Reference

This section describes facts about our servers and deployments.

4.1 Reference

4.1.1 Communicating during downtime

For services managed by Open Data Services, please see the protocol for planned and unplanned downtime.

4.1.2 Monitoring

UptimeRobot

Website downtime is monitored by UptimeRobot, which notifies sysadmin email addresses at OCP and ODS. Keyword monitors are used where possible.

Sentry

Application errors are reported to Sentry, which notifies individual email addresses. Not all services report errors to Sentry.

Prometheus

Servers are monitored by Prometheus. Salt is used to configure Prometheus monitoring on each server, and to set up a Prometheus server to to collect metrics from these servers.

We use the following exporters:

- Node Exporter is installed on each server to export hardware and OS metrics like disk space used, memory used, etc.

- Black Box Exporter is installed on the Prometheus server to check that services are up. (Keyword monitors are more complicated to configure than on UptimeRobot, and so are not used.)

Read the *user guide* to learn how to use Prometheus.

4.1.3 Hosting

OCP uses:

- Linode for the Helpdesk CRM, managed by Dogsbody Technology
  - Contact: sysadmin@dogsbody.com
- Hetzner for Kingfisher, managed by Open Data Services
  - Contact: code@opendataservices.coop
  - The 'opencontractingpartnership' and 'opencontracting-dogsbody' users have full access. The 'opencontracting' user has limited access.
- Bytemark for all others, managed by Open Data Services
  - Contact: code@opendataservices.coop
  - The 'opendataservices' user has secondary access to the 'opencontracting' account.
- GitHub Pages for the Extension Explorer

OCDS Documentation

This page serves as an orientation to how different components of the OCDS documentation relate to each other.

Servers

- `live.docs.opencontracting.uk0.bigv.io` serves released documentation of the OCDS (e.g. 1.1) and its profiles (e.g. Public Private Partnerships). It is a reverse proxy to draft documentation (below), the OCDS Data Review Tool, and the OC4IDS Data Review Tool.
- `staging.docs.opencontracting.uk0.bigv.io` serves draft documentation of the OCDS and its profiles. If necessary, it can be browsed directly.
- `live.standard-search.opencontracting.uk0.bigv.io` serves the Search API, whose base URL is http://standard-search.open-contracting.org/v1.

Version and language switchers

The version switcher links to a `/switcher` URL path with a `branch` URL parameter. The language switcher links to a `/{version}/switcher` URL path with a `lang` URL parameter. These are redirected by Apache (you can search for `/switcher` in its config files).

Search API

The search.js file in the `standard_theme` repository sends an unauthenticated request to the API's `/search` endpoint to retrieve search results.

The Travis deploy script in this repository sends an authenticated request to the API's `/index_ocds` endpoint to index the documentation for the OCDS and its profiles.

Travis

The repositories for OCDS documentation use Travis to push builds to the staging server and to rebuild the search index for the documentation:

- Each branch of the standard repository is automatically built to:

```
https://standard.open-contracting.org/{branch}/en/
```

- Each branch of a profile's repository is automatically built to:

```
https://standard.open-contracting.org/profiles/{root}/{branch}/en/
```

In detail, the repositories use script deployment to run deploy-docs.sh in this repository. For this script to succeed, Travis must be *configured* to have access to the staging server and to the Search API's `/index_ocds` endpoint.