Substance Documentation

Release 0.10.2

Turbulent inc.

Apr 30, 2021

Contents

1 Installation **3**
- 1.1 On macOS . . . 3
- 1.2 On Windows (WSL) . . . 4
- 1.3 On Windows (Cygwin) . . . 4
- 1.4 On Linux . . . 5
- 1.5 Upgrading Substance to a new version . . . 6

2 Basic Usage **7**
- 2.1 Architecture . . . 7
- 2.2 Create your first engine . . . 7
- 2.3 Syncing local files to and from the engine . . . 8
- 2.4 Setup your first environment . . . 8
- 2.5 Consulting logs . . . 9
- 2.6 Entering a container shell . . . 9
- 2.7 Executing a command from within a running container . . . 9

3 Engine Configuration **11**
- 3.1 `box` . . . 11
- 3.2 `devroot` . . . 11
- 3.3 `docker` . . . 12
- 3.4 `driver` . . . 12
- 3.5 `network` . . . 12
- 3.6 `profile` . . . 12
- 3.7 `aliases` . . . 12

4 Environment Setup **15**
- 4.1 Environment configuration . . . 16
- 4.2 Configuring containers . . . 16

5 Substance Internals **19**
- 5.1 The `subenv` utility . . . 19
- 5.2 How to use `subenv` . . . 19
- 5.3 Creating a subenv spec (.substance) . . . 20
- 5.4 `subenv.yml` . . . 20

6 Upgrade from substance 0 to 1 **23**
- 6.1 Upgrading on macOS . . . 23

6.2 Upgrading on Windows . . . 24
6.3 Upgrading on Linux . . . 24
6.4 Post-Install steps . . . 25
6.5 Known issues . . . 25

7 FAQs and Troubleshooting Substance **27**
7.1 Known Issues And Mitigation . . . 27

8 Indices and tables **29**

SUBSTANCE

Welcome to Substance's online help!

Substance combines a virtual machine and docker into one self contained tool. Complex projects can be distributed with a .substance folder which defines all the environmental details in one so you can spend more time developing and less time setting up servers.

CHAPTER 1

Installation

Substance supports all three main desktop operating systems: macOS, Windows, and Linux. Its main prerequisite is Oracle VirtualBox, which also supports these platforms.

If an existing substance project is going to be used, make sure to check how much RAM and disk space will be required before procceding with the installation.

1.1 On macOS

Substance is not compatible with the Python distribution that ships with macOS. You must use Homebrew to install the latest Python 2.x release.

1. Download and install Oracle VirtualBox. Make sure `VBoxManage` is in your `PATH` environment variable.
2. Make sure Xcode CLI is installed:

```
$ xcode-select --install
```

3. Ensure Homebrew is installed:

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/
↪install/master/install)"
```

4. Install Homebrew's Python (Version 3 and up) distribution:

```
$ brew install python
```

5. Make sure that you are running python 3.x when installing substance. If there is 2 versions of python installed *python/pip* will probably be 2.7, and *python3/pip3* 3.x
6. The `python3` installed by homebrew should be located under `/usr/local/bin`. The `python` version under `/usr/bin` is the one installed by default in MacOS. You can verify python version with `--version` argument and locate executable with `which` command.

If homebrew's version is not found that's probably because it is not in your PATH. Use the command `brew doctor`, copy the command to add it to your PATH and restart your terminal.

7. Ensure pip3 is up to date:

```
$ sudo pip3 install -U pip
```

8. Install `substance`:

```
$ sudo pip3 install substance
```

1.2 On Windows (WSL)

Disclaimer: Substance has only been tested on Windows 10 Pro 64-bit edition.

1. Install Oracle VirtualBox. Make sure VBoxManage is in your `PATH` environment variable (System -> Advanced System Settings -> Environment Variables -> Path -> add path to your VirtualBox installation directory).
2. Open PowerShell (or cmd) as Administrator and run:

```
Copy
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-
↪Linux
```

3. Restart the computer
4. Download and install your distribution of choice. We recommend Debian/stretch from the Microsoft Store.
    1. Open the Microsoft Store and choose your favorite Linux distribution.
    2. Debian GNU/Linux (Stretch)
5. Install python3 on your linux distribution:

```
$ apt-get update
$ apt-get install -y python3 python3-dev python3-pip
```

6. Install substance with *pip*:

```
$ pip3 install substance
```

The home directory used by substance in WSL will be your Windows user's home directory (for example: C:\Users\USERNAME) This is to allow your projects and code can be edited and modified from both programs inside WSL and outside.

For simplicity, the sync process (unison) within WSL will run the linux version.

1.3 On Windows (Cygwin)

Due to poor support for command-line utilities on Windows, Cygwin is required to run Substance on Windows.

Disclaimer: Substance has only been tested on Windows 10 Pro 64-bit edition. Installation on 32-bit Windows is NOT supported.

1. Install Oracle VirtualBox. Make sure VBoxManage is in your `PATH` environment variable (System -> Advanced System Settings -> Environment Variables -> Path -> add path to your VirtualBox installation directory).

2. Install Cygwin 64-bit. Here are the steps:
    1. Create a directory `C:\cygwin64` on your main drive. Create a subdirectory named "setup" inside that directory.
    2. Download `setup-x86-64.exe` from `https://www.cygwin.com/` and place it in `C:\cygwin64\setup`
    3. Double-click on `setup-x86-64.exe`, and perform the install. Keep the default location and make sure the following packages are selected:
        - `mintty` (under "Shells")
        - `make` (under "Devel")
        - `git` (under "Devel")
        - `gcc-core` (under "Devel")
        - `python2` (under "Python")
        - `python2-devel` (under "Python")
        - `libffi-devel` (under "Libs")
        - `openssl-devel` (under "Net")
    4. Optionally, you can create a shortcut to `setup-x86-64.exe` and add it to your Start menu; you can re-run the setup whenever you want to add or remove packages to your Cygwin install.
3. Launch Cygwin Terminal (`mintty`). All the magic happens from there!
4. Run `which python && which pip` to make SURE that you are running both executables from `/usr/bin`, NOT something like `C:\Python`!
5. Execute the command `python --version`. You should see an output like `Python 2.7.12`.
6. Make sure `pip` is installed and is upgraded to the latest version by running the commands:

```
$ python -m ensurepip
$ pip install -U pip
```

7. Install `substance`:

```
$ pip install substance
```

1.4 On Linux

Make sure you are running a 64-bit Linux distribution. 32-bit is NOT supported. Substance has been tested on Mint, Ubuntu, and Arch Linux.

1. Install Oracle VirtualBox. Make sure `VBoxManage` is in your `PATH` environment variable.
2. Install the following software using your package manager. Of course, depending on the distribution, the package names may slightly vary (but you will usually find a proper equivalent):
    - `git`
    - `build-essential`
    - `libffi-devel`
    - `openssl-devel`

3. Make sure to have Python 2 available. On some distributions (like Ubuntu 14.04), this is the default Python interpreter, which means you can use `python` and `pip`. On other, more state-of-the-art distributions (like Arch), you need to install a separate `python2` package and use the commands `python2` and `pip2` for the rest of this guide. Also install `python-devel` or `python2-devel`, depending on your distribution.
4. Install `substance`:

```
$ sudo pip install substance
```

1.4.1 Ubuntu 20.04 LTS

This version of Ubuntu already comes with the development libraries required for substance.

1. Install Oracle VirtualBox.
2. Substance requires Python 3 and pip 3. Python 3 will be already installed and it is accessible with `python3`. Install pip 3 with:

```
$ sudo apt install pip3
```

3. Install `substance`:

```
$ sudo pip3 install substance
```

Make sure you use Python 3 and pip 3, and not the Python 2 counterpart.

1.5 Upgrading Substance to a new version

On all supported platforms, these commands will allow you to update the Substance on your machine without losing data or engines:

```
$ sudo pip uninstall substance
$ sudo pip install substance
```

CHAPTER 2

Basic Usage

2.1 Architecture

Substance manages local development environments contained in an *engine*. An *engine* is a virtual machine running Docker. As long as you have enough memory and processing power available on your host machine, Substance can keep multiple *engines* running in parallel. Each engine is assigned a directory on your local machine, which defaults to `$HOME/substance/[engine name]`.

Within a single engine, multiple *environments* can be setup. Each *environment* corresponds to a project with a separate codebase. Essentially, each *environment* is a separate directory within the engine's directory, which would default to `$HOME/substance/[engine name]/[env name]`.

Finally, within each environment multiple *services* can be defined. Each *service* corresponds to a long-lived Docker container with some defined filesystem volumes, environment variables, and/or exposed ports. Each *service* is labeled with an easy-to-remember name such as *web* or *database*. The *services* are defined in a templated YAML file at `$HOME/substance/[engine name]/[env name]/.substance/dockwrkr.yml.jinja`.

Fig. 1: Example of a local machine with three environments spread over two engines running in parallel.

2.2 Create your first engine

To start using a substance environment, first create an engine. For the sake of this example we will create an engine named `work`:

```
$ substance engine create work --memory 2048 --cpus 2
```

Launch your engine with the launch command:

```
$ substance engine launch work
```

After a few minutes, substance will have pulled the box and launched your engine. By default substance will create a devroot directory for this new engine in `$HOME/substance/work`.

Since this is our first and main engine, we need to tell Substance to use our new engine as the default engine for future commands. This is done with the `use` command:

```
$ substance use work
```

2.3 Syncing local files to and from the engine

Since a Substance engine is a virtual machine, there must be a way to synchronize local files to and from the virtual machine. Though engines are managed by VirtualBox, Substance does not make use of VirtualBox's *shared folder* feature due to performance reasons. Instead, Substance leverages the Unison syncing utility.

You can start the sync process like so:

```
$ substance sync
```

This will make sure all local files located under `~/substance/[engine name]` are kept in sync with the engine's local file system.

Always keep this process running while you develop!

2.3.1 Known Issues

- `Fatal error: Filesystem watcher error: cannot add a watcher: system limit reached`

  This happens when the number of files being watched on the local system exceeds the maximum amount of files permitted to be watched by a single user at the same time.

  To increase this limit, modify the value inside: `/proc/sys/fs/inotify/max_user_watches`.

- `Fatal error: Server: End_of_file exception raised in loading archive (this indicates a bug!)`

  This error can happen if some unison cache files are corrupted. to fix this, delete the contents of `~/.substance/unison/` and `/substance/.unison/` inside your substance engine.

2.4 Setup your first environment

To start working on a project, `git clone` the project in `~/substance/[engine name]/[project name]`. Make sure your files are properly sync'ed, then instruct Substance to switch to that project and initialize the development environment by issuing the following command:

```
$ substance switch [project name]
```

This will download the proper Docker images and start the Docker containers required for the project to work.

At this point, your environment is up-and-running, but the project may require more initialization steps (e.g. building Javascript and/or setuping a database). Check the project's maintainer for further setup instructions.

2.5 Consulting logs

Substance expects all services of an environment to write logs to a directory using a specific filename convention. The filename should be dash-separated lowercase segments and end with `.log`. The first segment should be the container name, other segments are optional and up to the specific Docker image. Example of valid log filenames: `web-nginx-access.log`, `database-mysql-slow.log`, etc.

To view logs of the various services of your environment:

```
$ substance logs filter1 filter2
```

This will automatically tail the logs matching the filters you provided. For example, `substance logs web php` will tail logs with the pattern `web-php-*.log` (logs of the PHP-FPM process running in the Docker container named `web`).

2.6 Entering a container shell

To open an interactive bash shell into a service container of the current environment, you can use the `substance ssh` command:

```
$ substance ssh [containername]
```

By default, this opens a shell as the root user in the root directory of the container. You can also specify a user and an initial directory with the `-u` and `-d` switches respectively:

```
$ substance ssh -u [username or uid] -d /path/to/initial/directory [containername]
```

2.7 Executing a command from within a running container

For one-off commands, rather than opening a full interactive shell, it may be easier to use the `substance exec` command:

```
$ substance exec [containername] echo "Hello, world!"
```

Just like `substance ssh`, the `-u` and `-d` switches can be used to override the default root user and directory for the command. *Make sure to specify these switches before the container name*!

You can also configure your engine to have aliases for often-used commands. For example, by default, a newly-created engine comes with an alias for the `make` command to be run within a container named `web` as the user `heap` and directory `/vol/website`, so that executing:

```
$ substance make
```

is functionally-equivalent to executing:

```
$ substance exec web -u heap -d /vol/website make
```

To learn more about aliases, consult *aliases*.

CHAPTER 3

Engine Configuration

Substance engines are defined by a simple YAML configuration file which is located at `$_HOME/.substance/engines/[enginename]/engine.yml`. This file is automatically created for you when you create a new engine using the `substance engine create` command.

You can edit this file in any text editor of your choice. As a convenience, Substance will open it in your favorite editor (specified by the `EDITOR` environment variable) using the command `substance engine edit [enginename]`.

What follows is an overview of all configuration sections in the file.

3.1 box

Specifies the box file used to create the VM in VirtualBox upon first launch. After first launch, this setting is no longer used.

3.2 devroot

This configures how the local directory for the engine will behave.

- `path` specifies the path to the directory on the local machine.
- `mode` specifies the sync mode. Possible values are `unison` (default) and `subwatch` (deprecated).
- `excludes` lists filename patterns to ignore on sync. You can use this setting to ignore additional file patterns for the projects hosted using this engine. Note that this can only match patterns based on filename, not full directory path. To ignore files based on directory path, use the `syncArgs` setting.
- `syncArgs` lists additional arguments to pass to `unison` when `mode` is set to `unison`.

Note: The `subwatch` mode is the legacy way to sync files. It does not work as expected under all conditions. You should really use `unison` instead. If `unison` really isn't working out for you, then try `subwatch`, but expect bugs.

You will also have to install the `subwatch` module separately using `pip`.

3.3 docker

This configures the Docker daemon running inside the engine. You will usually want to leave the defaults as-is.

3.4 driver

Specifies the virtualization backend to use for managing the virtual machine. `virtualbox` is the only currently-supported value for now, but more backends will be added in the future.

3.5 network

All Substance engines are networked using a Host-only network adapter serving as a DHCP server. The settings in this section are automatically filled in and updated by Substance after each launch of the engine.

3.6 profile

Specifies the capabilities of the virtual machine. Currently, two settings are supported:

- `cpus`: Specifies number of virtual CPUs
- `memory`: Specifies amount of memory to reserve for the VM (in megabytes)

Note that these settings can be changed even after the VM has been created in VirtualBox (requires a reboot of the VM).

3.7 aliases

Substance engines can define a list of commonly-used commands to execute within specific containers. This allows the user to avoid having to type a long and difficult-to-remember command for common tasks such as building a website, pulling packages, etc.

Since Substance was designed for web development first, all engines come pre-configured with a few aliases for web development. All of these aliases are configured to be executed within a container named `web` as a user named `heap` within the directory `/vol/website`:

- `substance make`: Execute `make`.
- `substance composer`: Execute the `composer` command (PHP package manager).
- `substance npm`: Execute the `npm` command (Javascript package manager)
- `substance watch`: Execute the `webpack -w` command (Web build tool in watch mode)

Note that executing an alias will pass along all arguments following the name of the alias, e.g.:

```
$ substance make clean TARGETS=all # executes 'make clean TARGETS=all' within the␣
↪container
```

You can add more aliases (or even change the configuration for the pre-configured ones) by editing your engine's definition file. Here's the example configuration for the above command:

```
aliases:
  composer:
    args:
    - composer
    container: web
    cwd: /vol/website
    user: heap
  make:
    args:
    - make
    container: web
    cwd: /vol/website
    user: heap
  npm:
    args:
    - npm
    container: web
    cwd: /vol/website
    user: heap
  watch:
    args:
    - watch
    container: web
    cwd: /vol/website
    user: heap
```

Simply add more entries to the `aliases` YAML object to define new aliases for your engine.

CHAPTER 4

Environment Setup

Warning: Some of this information may be a little out-of-date. We will soon provide an automated tool to do this work. In the mean time, please talk to the substance developers directly if you want to setup your project for substance.

At the top level of your project source code create a folder named `.substance`. In this folder. A system to initialize projects is in the works, but for now create the following base structure or copy it from a recent project:

```
.
|-- conf
|   |-- cron
|   |-- logrotate
|   `-- web
|-- data
|   |-- media
|   |-- uploads
|   `-- work
|-- database
|-- dockwrkr.yml.jinja
|-- logs
|-- spool
|-- subenv.yml
`-- var
    |-- heap
    |   `-- cache
    |-- purifier
    |   `-- cache
    `-- smarty
        `-- compile
```

Ensure the `var` directory and it's children have mode 0777.

4.1 Environment configuration

In your `.substance` folder you should create a `.env` file that will host the configuration/environment variable you will need when templating files in your project subenv.

The special variable called `SUBENV_FQDN` controls the DNS name for your project. Make sure you set it. Also, make sure this name is in the same TLD as configured in your substance config. (`~/.substance/subenv.yml`)

Here is a sample `.env`:

```
SUBENV_FQDN="myproject.local.dev"
MYCUSTOMVAR="foobar"
```

When resolving variables ; subenv will also look for a `.env` file at the root of your project for overrides.

4.2 Configuring containers

Substance uses dockwrkr to manage containers in the work engine. Your project must define the containers it requires to function and their configuration in the form of a `dockwrkr.yml` file.

Your `.substance` directory must contain a `dockwrkr.yml.jinja` that will be used by substance to resolve environment and configuration variable before writing the file in your project environment in the work engine.

Refer to the dockwrkr manual for details on this configuration as well as the subenv command for details on the variables available. All variables you defined in your `.env` file can be used in the jinja template.

Here is a sample Heap project `dockwrkr` configuration template

```
pids:
  enabled: false
  dirs: pids
containers:
  dbmaster:
    image: docker-registry.turbulent.ca:5000/heap-mysql:2.0
    hostname: dbmaster
    env:
      VAR_MYSQL_PASS: "dev"
      VAR_MYSQL_INNODB_BUFFER_POOL_SIZE: "100M"
      VAR_MYSQL_SERVER_ID: 2
      VAR_MYSQL_REPLICATION_MASTER: 1
      VAR_MYSQL_REPLICATION_USER: "replication"
      VAR_MYSQL_REPLICATION_PASSWORD: "dev"
      VAR_PROJECT_NAME: {{name}}
    publish:
      - "3306:3306"
    volume:
      - "{{SUBENV_ENVPATH}}/logs:/vol/logs"
      - "{{SUBENV_ENVPATH}}/database:/vol/database"

  cache:
    image: docker-registry.turbulent.ca:5000/heap-memcached:2.0
    hostname: cache
    publish:
      - "11211:11211"
    env:
      VAR_MEMCACHED_SIZE: "64M"
```

(continues on next page)

(continued from previous page)

```
    volume:
      - "{{SUBENV_ENVPATH}}/logs:/vol/logs"

  redis:
    image: docker-registry.turbulent.ca:5000/heap-redis:2.0
    hostname: redis
    publish:
      - "6379:6379"
    volume:
      - "{{SUBENV_ENVPATH}}/logs:/vol/logs"
      - "{{SUBENV_ENVPATH}}/database:/vol/database"

  qmgr:
    image: docker-registry.turbulent.ca:5000/heap-qmgr:2.0.1
    hostname: qmgr
    env:
      VAR_HEAP_QUEUE_WORKERS: 1
    link:
      - "dbmaster:dbmaster"
      - "cache:cache"
      - "sessions:sessions"
      - "rabbit:rabbit"
      - "redis:redis"
    volume:
      - "{{SUBENV_ENVPATH}}/logs:/vol/logs"
      - "{{SUBENV_ENVPATH}}/var:/vol/var"
      - "{{SUBENV_ENVPATH}}/spool:/vol/spool"
      - "{{SUBENV_BASEPATH}}:/vol/website"
      - "{{SUBENV_ENVPATH}}/data:/vol/data"

  web:
    image: docker-registry.turbulent.ca:5000/heap-app-dev:2.0.5
    hostname: web
    env:
      VAR_NMAILER_HOSTNAME: ""
      VAR_NMAILER_ROOT_ALIAS: ""
      VAR_NMAILER_DOMAIN: ""
      VAR_NMAILER_REMOTE_TLS: 0
      VAR_NMAILER_REMOTE_HOST: ""
      VAR_NMAILER_REMOTE_PORT: "25"
      VAR_NMAILER_REMOTE_USER: ""
      VAR_NMAILER_REMOTE_PASS: ""
      VAR_NGINX_SERVER_NAME: "{{SUBENV_FQDN}}"
      VAR_FPM_MAX_CHILDREN: 5
      VAR_FPM_MIN_CHILDREN: 5
      VAR_FPM_MAX_REQUESTS: 500
    link:
      - "dbmaster:dbmaster"
      - "cache:cache"
      - "sessions:sessions"
      - "rabbit:rabbit"
      - "redis:redis"
    publish:
      - "80:80"
      - "443:443"
      - "9001:9001"
      - "9002:9002"
```

(continues on next page)

(continued from previous page)

```
    - "9003:9003"
  volume:
    - "{{SUBENV_ENVPATH}}/logs:/vol/logs"
    - "{{SUBENV_ENVPATH}}/var:/vol/var"
    - "{{SUBENV_ENVPATH}}/spool:/vol/spool"
    - "{{SUBENV_ENVPATH}}/data:/vol/data"
    - "{{SUBENV_ENVPATH}}/conf/web:/vol/conf"
    - "{{SUBENV_BASEPATH}}:/vol/website"

cron:
  image: docker-registry.turbulent.ca:5000/heap-cron:2.0.1
  hostname: cron
  env:
    VAR_NMAILER_HOSTNAME: ""
    VAR_NMAILER_ROOT_ALIAS: ""
    VAR_NMAILER_DOMAIN: ""
    VAR_NMAILER_REMOTE_TLS: 0
    VAR_NMAILER_REMOTE_HOST: ""
    VAR_NMAILER_REMOTE_PORT: "25"
    VAR_NMAILER_REMOTE_USER: ""
    VAR_NMAILER_REMOTE_PASS: ""
  link:
    - "dbmaster:dbmaster"
    - "cache:cache"
    - "sessions:sessions"
    - "rabbit:rabbit"
    - "redis:redis"
  volume:
    - "{{SUBENV_ENVPATH}}/logs:/vol/logs"
    - "{{SUBENV_ENVPATH}}/var:/vol/var"
    - "{{SUBENV_ENVPATH}}/spool:/vol/spool"
    - "{{SUBENV_ENVPATH}}/conf/cron:/vol/conf"
    - "{{SUBENV_ENVPATH}}/data:/vol/data"
    - "{{SUBENV_BASEPATH}}:/vol/website"

logrotate:
  image: docker-registry.turbulent.ca:5000/heap-logrotate:2.0
  hostname: logrotate
  env:
    VAR_LOGROTATE_MODE: "daily"
    VAR_LOGROTATE_ROTATE: "7"
  volume:
    - "{{SUBENV_ENVPATH}}/logs:/vol/logs"
    - "{{SUBENV_ENVPATH}}/conf/logrotate:/vol/conf"
    - "/var/lib/docker/containers:/vol/docker-logs"
```

CHAPTER 5

Substance Internals

The following documentation is not required knowledge for using Substance. However, if you are curious about how Substance works under-the-hood, keep reading!

5.1 The `subenv` utility

When operating on a substance engine, the `subenv` utility is used inside the engine virtual machine to generate and switch between project environments.

`subenv` can read the spec folder `.substance`. in your project root and will template and create the runtime environment to run your project.

5.2 How to use `subenv`

You initiate/apply an environment by using the `init` command:

```
$ subenv init path/to/myprojectA
Initializing subenv from: path/to/projectA
Loading dotenv file: 'path/to/projectA/.substance/.env'
Loading dotenv file: 'path/to/devroot/projectA/.env'
Applying environment to: /substance/envs/projectA
Environment 'projectA' initialized.
```

List available environments using `ls` command:

```
$ subenv ls

NAME         CURRENT    BASEPATH                         MODIFIED
projectA     X          /substance/devroot/projectA      June 28 2016, 14:14:17
projectB     X          /substance/devroot/projectB      June 28 2016, 15:11:10
projectC     X          /substance/devroot/projectC      June 23 2016, 01:33:02
```

Switch the currently use environment with `use`:

```
$ subenv use projectC
```

View the currently in use environment with `current`:

```
$ subenv current
projectC
```

View the computed variables for the current environment:

```
$ subenv vars
name="projectA"
fqdn="projectA.local.dev"
```

5.3 Creating a subenv spec (.substance)

Each directory in your engine devroot (referred to here as a project) must have a spec directory (.substance) to define what the environment needed to run the project is.

When a specdir is applied (using `init`) subenv will go through the file structure of the specdir and take the following actions:

- Each file is copied to the environment path
- Each folder is created recursively in the environment path
- All files ending with `.jinja` are render with Jinja2 and installed in the environment path withouth the .jinja extension.

The variable context for the jinja templates is populated from the merge `.env` files in the specdir and the project directory. `subenv` will load the `.env` file in your specdir first and override the values from an optional `.env` file in your project root.

Additionally, subenv will provide the following variables for use in your jinja templates:

Variable	Value
SUBENV_NAME	Name of the subenv
SUBENV_LASTAPPLIED	Epoch time of last time this env was applied
SUBENV_SPECPATH	Full path to the specdir
SUBENV_BASEPATH	Full path to the project path
SUBENV_ENVPATH	Full path to the environment
SUBENV_VARS	Dict of all variables

5.4 subenv.yml

You can also create a `subenv.yml` file to specify additional commands to be executed once the environment is applied. The file format is YAML and only the `script` setting is available which should contain a list of shell commands to run when applying the envrionment. Each command is run as UID 1000 within the environment directory.

Sample:

```
script:
  - chmod -R 777 var
  - chmod -R 700 database
```

CHAPTER 6

Upgrade from substance 0 to 1

Upgrading from substance 0 to 1 requires moving your dependencies from Python 2 to Python 3 and VirtualBox 6.0.

6.1 Upgrading on macOS

1. Stop all your engines:

```
$ substance engine ls
$ substance engine halt <ENGINENAME> # Do this for every running engine
```

2. Uninstall substance:

```
$ sudo pip uninstall substance
```

3. Upgrade VirtualBox to 6.x:

```
- Visit https://www.virtualbox.org/wiki/Downloads
- Install the package on your computer.
- Your existing machines and engines should remain intact.
```

4. Upgrade Python to Python 3:

```
$ brew uninstall python
$ brew uninstall python@2
$ brew install python
```

5. Install `substance`:

```
$ sudo pip3 install substance
```

6. Upgrade your engine by either creating a new engine.

6.2 Upgrading on Windows

1. Stop all your engines:

```
$ substance engine ls
$ substance engine halt <ENGINENAME> # Do this for every running engine
```

2. Uninstall substance via cygwin:

```
$ sudo pip uninstall substance
```

3. Upgrade VirtualBox to 6.x:

```
- Visit https://www.virtualbox.org/wiki/Downloads
- Install the package on your computer.
- Your existing machines and engines should remain intact.
```

4. Upgrade Python to Python 3:

- Run the Cygwin setup (setup-x86_64.exe)
- Remove python, python-devel and python-pip
- Add python3, python3-devel, python3-pip

1. Install `substance` using the Cygwin terminal:

```
$ sudo pip3 install substance
```

6.3 Upgrading on Linux

1. Update your system.:

```
# Debian/Ubuntu
$ sudo apt-get update && sudo apt-get upgrade

# Arch
$ sudo pacman -Syu
```

2. Reboot:

```
- needed in case the virtualbox kernel modules were updated.
```

3. Stop all your engines:

```
$ substance engine ls
$ substance engine halt <ENGINENAME> # Do this for every running engine
```

4. Install python3:

```
# Debian/Ubuntu
$ sudo apt-get install -y python3.7 python3-pip

# Arch
$ sudo pacman -S python python-pip
```

5. Update substance:

```
$ sudo pip3 install substance
```

6. You will need to create a new engine so that it can use the newest docker image and virtualbox version:

```
$ substance engine create <ENGINENAME>
```

6.4 Post-Install steps

1. After upgrading to substance 1.x, you should take a moment to update your jinja templates to use Python 3 syntax.

```
# example:
# dockwrkr.yml.jinja

  [...]

  VAR_NMAILER_REMOTE_PORT: "587"
  VAR_NMAILER_REMOTE_USER: ""
  VAR_NMAILER_REMOTE_PASS: ""
  {%- for k, v in SUBENV_VARS.iteritems() %}

# becomes

  [...]

  VAR_NMAILER_REMOTE_PORT: "587"
  VAR_NMAILER_REMOTE_USER: ""
  VAR_NMAILER_REMOTE_PASS: ""
  {%- for k, v in SUBENV_VARS.items() %}
```

6.5 Known issues

1. If you get a `CryptographyDeprecationWarning` when running substance commands, it's because of this issue. As noted, a workaround is running `sudo pip install cryptography==2.4.2` until the problem is fixed in paramiko.

CHAPTER 7

FAQs and Troubleshooting Substance

This page answers FAQs, as well as how to diagnose, debug, and work around known issues. We are actively updating this document!

7.1 Known Issues And Mitigation

7.1.1 My engine is running, memory is up still no VPN, connection is present, but ping times out

- Stop vm
- Unload kext:

```
kextstat | grep "org.virtualbox.kext.VBoxUSB" > /dev/null 2>&1 && sudo kextunload -b␣
↪org.virtualbox.kext.VBoxUSB
kextstat | grep "org.virtualbox.kext.VBoxNetFlt" > /dev/null 2>&1 && sudo kextunload -
↪b org.virtualbox.kext.VBoxNetFlt
kextstat | grep "org.virtualbox.kext.VBoxNetAdp" > /dev/null 2>&1 && sudo kextunload -
↪b org.virtualbox.kext.VBoxNetAdp
kextstat | grep "org.virtualbox.kext.VBoxDrv" > /dev/null 2>&1 && sudo kextunload -b␣
↪org.virtualbox.kext.VBoxDrv
```

- load them back:

```
sudo kextload "/Library/Application Support/VirtualBox/VBoxDrv.kext" -r "/Library/
↪Application Support/VirtualBox/"
sudo kextload "/Library/Application Support/VirtualBox/VBoxNetFlt.kext" -r "/Library/
↪Application Support/VirtualBox/"
sudo kextload "/Library/Application Support/VirtualBox/VBoxNetAdp.kext" -r "/Library/
↪Application Support/VirtualBox/"
sudo kextload "/Library/Application Support/VirtualBox/VBoxUSB.kext" -r "/Library/
↪Application Support/VirtualBox/"
```

CHAPTER 8

Indices and tables

- genindex
- search