SaltStack from Scratch

Mark Bools

December 10, 2020

Document ID: A000000002
Last Modified: 2022-08-03

Contents

Chapter 1

How to. . .

This chapter offers some guidance on getting the most from this book.

1.1 . . . read this book

If I was being flippant I might say, "with your eyes" but I'm bigger than that so instead I will suggest some ways you might use this material.

This book is organised as a single narrative course centred around Salt. If you start on page one, read through each page and follow along with all the material, you should end up being proficient with Salt by the end.

This book contains mistakes. Deliberate mistakes (and no doubt some mistakes that I did not intend, that's life). Sometimes we will do things more than once. WTF? In most educational material we are presented with 'perfect' solutions, this is not realistic. The real world sucks. It changes constantly and today's ideal solution is okay but tomorrow the boss (or customer) decides new technology is desirable how do we handle this? This is were some soft skills may be required to persuade them to change their mind. Assuming we cannot persuade them to change we need to plan and execute a migration from the older solution to the new solution. Rather than avoid this sort of complexity this course takes it head on.

Don't have time to follow along from the start? Perhaps you already know enough about Salt's state system and feel confident skipping that material. No problem. At the start of each section you will find details of how to create an appropriate environment for that section (see §1.3). These 'checkpoints' also mean that if you mess up you can simply throw your environment away, recreate it using the closest checkpoint and continue with the course. In fact I encourage you to mess up your environment. You will learn much more by 'playing'. So set up and environment, mess around with your own ideas and then, when you are ready to do the next part of the course, either restore your own saved snapshots or tear down the environment and build a new one using the provided checkpoint code.

Having difficulty? No problem. Ask for help. Someone in the community may help and since I am in the community I can also help clarify things. If enough people are confused by the material then obviously I messed up and need to rewrite that material to be more clear; it shall be done.

1.2 ...get the most from this book

Do the examples! You'll get much more from the material by following along and investigating for yourself.

1.3 ...manage your workspace

We are going to be doing a lot of practical work throughout this book so it is worthwhile considering how we will manage our workspace.

A lot of this section will only make sense once you have read about the tools Git, VirtualBox, and Vagrant but I'm assembling basic advice here to make it easier to refer to later.

We need to control two working environments throughout this course:

1. The *host* workspace—this is the workspace on your computer, and the one we will set up shortly.
2. The *guest* workspace—this is the workspace on the VirtualBox virtual machines we will be creating. Generally you will find these already created on each virtual machine.

If you follow this book from page one to the end you should find your workspaces are always in sync with whatever the book is dealing with but practically many of you will jump to sections of particular interest, skipping many sections. Anticipating this I've put in plenty of checkpoints, particularly for the guest workspace.

1.3.1 Initial setup of your workspace

Assuming you have installed Git (see §2.4) you can create your project's host workspace.

```bash
mkdir sfs
cd sfs
git clone --depth 1 https://gitlab.com/saltyvagrant.classes/sfs-material.git course-material
mkdir classroom
mkdir archive
```

Your `sfs` workspace now contains three directories; `course-material` that holds all this books accompanying material, `classroom` is where you will follow along with the course, and `archive` is where we will store various backup files.

Throughout this book I use the `WSR` directory[1] to refer to the root of your workspace. Any path that does not explicitly start from the `WSR` root assumes you are following instructions from the last checkpoint and are relative to whichever directory you should be in at the time.

[1]If you are using Microsoft Windows as your host you will need to convert '/' to '\' in paths whenever working in the host workspace. (It is precisely because of this sort of "conversion" nonsense that we use the guest workspace most of the time.)

bash

```bash
1  cd WSR
2  cd WSR/classroom
3  cd ../course-material
4  cd WSR
5  cd classroom
```

Lines 1, 2, and 4 each start with `WSR` and are therefore not really relative to your current working directory. You should take these to be absolute directories rooted at your workspace root `WSR`. If your workspace is at `/home/fred/xyz` then `WSR/classroom` should be read as `/home/fred/xyz/classroom`.

Line 3 is relative to your current working directory (in this example `WSR/classroom`) and is referring to `WSR/course-material` (since the parent of `WSR/classroom` it `WSR` and `course-material` is to be found directly under this directory).

Line 5 is again relative to your current working directory. As you just moved to `WSR` (line 4) this refers to your `classroom` directory under that root.

1.3.2 Regular host workspace activities

There are a number of actions you may want to repeat throughout this course. Rather than repeat them in full each time I present them here and simply refer to these entries as necessary.

1.3.2.1 Checkpoint Classroom

You will need to checkpoint the classroom at least once, when you start the course. If you get lost in the material you can reset your classroom to one of the checkpoints in the book. This will clean up you `classroom` directory ensuring you are ready to proceed with the book's follow-along lessons.

1. Shutdown any running classroom Virtual Machine (VM)[2] (if one is currently set up).

   bash

   ```bash
   cd WSR/classroom
   vagrant halt
   ```

2. Backup your current classroom

   bash

   ```bash
   cd WSR
   mv classroom archive/classroom_<date>
   ```

   Replace `<date>` with the date of the backup (I recommend using a `YYYMMDD` format as this sorts properly). For example, if today where December 3rd 2020 and I wanted to backup my classroom I would us the following[3].

[2]A segmented presentation or emulation of a physical computer allowing multiple 'guest' machines to share the physical resources of the 'host' computer.

[3]Windows users should use `move` rather than `mv`

```bash
cd WSR
mv classroom archive/classroom_20201203
```

3. Copy the relevant material from `course-material`

   ```bash
   cd WSR
   cp course-material/<checkpoint>/ classroom
   ```

   Replacing `<checkpoint>` with the name if the checkpoint from which you want to proceed.

4. Start up the classroom

   ```bash
   cd WSR/classroom
   vagrant up
   ```

   This will start up any VM required for the classes.

1.3.2.2 Snapshot Classroom

If you are following the advice given above and 'playing' with your classrooms then I suggest your take a snapshot of your environment just before you start to play. This way you can quickly reset your classroom back you a point where it is ready for you to continue following along with this course.

1. Follow along with this course.

2. Decide to 'play' for a while so take a snapshot.

   ```bash
   cd WSR/classroom
   vagrant snapshot save class_<date>
   ```

   Replace `<date>` with the date of the snapshot (I recommend using a `YYMMDD` format as this sorts properly). For example, if today where December 3rd 2020 and I wanted to backup my classroom I would use the following.

   ```bash
   cd WSR/classroom
   vagrant snapshot save class_201203
   ```

3. Play with your classroom environment.

4. Decide to resume the course as described in this book.

5. Restore your classroom to the saved snapshot.

   ```bash
   cd WSR/classroom
   vagrant snapshot restore class_<date>
   ```

   Where `<date>` is the date of the snapshot to be restored. For example, to restore the snapshot from December 3rd 2020 we created earlier I would use the following.

   ```bash
   cd WSR/classroom
   vagrant snapshot restore class_201203
   ```

6. Resume course.

One other useful snapshot command is `list`, this can be used to show your previously saved snapshots (useful if, like me, your forget this sort of thing).

```bash
cd WSR/classroom
vagrant snapshot list
```

1.3.2.3 Update material

This book, and consequently the accompanying material, is continually being updated[4]. Most updates will be to the *guest* workspace consequently the host workspace will rarely need updating, the following procedure will update your host workspace and bring your guest systems up to date.

```bash
cd WSR/course-material
git pull
```

This will update the course material in the host workspace. If you have created a `classroom` then you may need to re-copy the relevant course material and re-provision the virtual machine.

```bash
cd WSR
cp -rf course-material/<checkpoint>/* classroom
cd classroom
vagrant up --provision
```

[4]If you have downloaded the PDF version of this book then you should download the latest version at the same time you update the course material, otherwise they will get out of sync.

Line 2 copies the relevant checkpoint files (obviously replacing `<checkpoint>` with the actual checkpoint directory you want to use). Line 4 will update any guest virtual machines (even if they already exist or are running).

Chapter 2

Setting Up Your Environment

In order that we are all seeing the same environment as we progress through the following material you will need to install three applications onto your computer (the host computer):

- `VirtualBox`
- `vagrant`
- `git`

Let's take a look at each and discuss why they are required.

2.1 VirtualBox

VirtualBox is Oracle's virtual machine application. This allows us to run a virtual (guest) machine on our host computer. This in turn means that even if you are running, for example, a Windows PC you will be able to run the Linux servers required to follow along with this material.

Virtualisation also isolates our host computer from the machines that we use. This has the advantage that no matter how badly we mess up the virtual environment it will have no effect on our host computer and any change to our host computer will have no effect on the virtual machines[1].

2.2 Vagrant

Vagrant is HashiCorp's command line tool for managing virtual machines. Vagrant provided a simple consistent method for defining virtual machines as code. This means we can all easily set up the same virtual machine environment without the need to rely on following complex set up instructions.

As with many topics covered in this course, there is a more detailed book covering Vagrant *Vagrant from Scratch*[Boo20c].

[1]This is not 100% true, but close enough for our purposes here.

2.3 git

Git has become the *de facto* standard in version control tools. Git is a powerful tool, unfortunately its history means it has a bloated command line interface that is often daunting and confusing to newcomers. Fear not! We will initially use `git` commands to obtain some files and nothing more (so you can just type the commands with no need to understand them) but as we progress we will explain the `git` command line and if you are interested in learning Git in detail there is a complete book on the topic *Git from Scratch*[Boo20b].

2.4 Installing the host tools

I have prepared some brief installation videos but to get the most up-to-date instructions for installing these host tools follow the instructions on their web sites.

Chapter 3

Core Concepts

There are several concepts that crop up across the design and management of IT that are so universally applicable that it is worth learning them regardless of your specific interests.

This chapter introduces these core concepts.

3.1 Cohesion

Cohesion refers to how closely elements are related to one another. In generally it is desirable to keep related things together and unrelated things separate.

3.2 Coupling

Coupling refers to how tightly to elements depend upon one another. In general we want elements to be as independent as practicable.

3.3 Abstraction

Abstraction is the process of extracting the essential from the incidental.

3.4 Separation of Concerns

Separation of concerns is a general principal that employs abstraction to increased cohesion and reduce coupling. The general idea is for elements to 'mind their own business', performing a well bounded function (or set of factions).

3.5 Scope

Scope is the 'range of applicability' of an element.

3.6 Context

Everything operates in a context. Most of the time in IT the context is well defined.

3.7 Contingency

Probably best summarised as ‘it depends’, contingency is the idea that we often must account for things changing.

3.8 Entropy

‘Things degrade over time’, or perhaps more accurately ‘without concerted effort to prevent it, thing get worse over time’. In software circles this is colloquially known as ‘bit rot’, the idea that without specific work to avoid it a software system’s structure will, over time, become more complex, more difficult to maintain, and more prone to error.

3.9 Parsimony

The ‘KISS’ (Keep It Simple, Stupid) principle. Do not make things more complex than required. The more parts a system has the more opportunity the more things there are to go wrong, so it makes sense to use as few things as possible.

Part I

Roadmap

Author Note

Overview of SaltStack.
Quick demo.
Plan of attack.

Chapter 4

Quick Overview of SaltStack

SaltStack is an infrastructure management toolkit. Written primarily in Python et provides tools for building, monitoring, and 'healing' infrastructure.

4.1 The Flexibility Problem

One major difficulties people face when learning Salt is there are often several ways to achieve the same thing. Worse, the Salt documentation and many other courses use these different forms interchangeably. In this course we will use one consistent form and refer the interested reader to alternate forms which I document in the appendices.

One common issue is the multiple ways to write common state stanzas. The following all achieve the same basic outcome (the tool `git` is installed).

```
git:
  pkg.installed: []
```

```
install-git:
  pkg.installed:
    pkgs:
      - git
```

```
install-git:
  pkg.installed:
    - name: git
```

4.2 The Terminology Problem

Salt has some 'odd' terminology. This takes some getting used to but once mastered it becomes quite natural.

We will investigate each of these as we encounter them in the course, but for now just be aware of these terms.

master A computer designated as a controller.

minion A device to be controlled by SaltStack.

grain Theses are data on each minion.

state A definition of some desired setup for a minion.

state module A piece of code that assures a particular state on a minion.

execution module A piece of code that affects a change on a minion[1].

pillar Data hosted on the master.

mine A subsystem allowing minions to selectively and securely share there grain data with the master (and subsequently other minions).

beacon An agent to monitor a minion and report to the master.

reactor A subsystem running on a master, reacting to events on the SaltStack event bus.

runner Similar to `execution module` but `runner`s operate on a master rather than a minion.

orchestration runner A specialised `runner` for coordinating the configuration of several minions.

As you can see the SaltStack development team leaned into the 'salt' metaphor quite heavily.

Another problem is that there is a lot of ambiguity in the use of some terms. Particularly, 'state' is sometimes used to refer to a state file and at other times to refer to blocks within a state file. To avoid this particular issue I use 'state' to refer to the file and 'state stanza' to refer to blocks within a state file. When other ambiguities arise I will introduce consistent terms and refer to the appendices to discuss the relevant issue.

[1]The difference between `state module` code and `execution module` code is often confusing to new users. Briefly, a `state module` checks that some condition is true on a minion, if the condition is not met the `state module` code calls one of more `execution modules` to affect change on the minion so that the condition is met. We look at this process in detail fairly early in the course.

Part II

Configuring a Single Machine

Author Note

Introduce all the basic configuration mechanics.

To introduce the basic mechanics of Salt configuration management we use it to build out a fairly simple server.

Chapter 5

Installing Masterless Salt

One of Salt's modes is running locally. In this chapter we install Salt onto a server with the intention of using it locally.

5.1 Using bootstrap

Salt provides an installation script that simplifies installing Salt to running one command. This script is available from `https://bootstrap.saltproject.io` (you can visit this page now, assuming you're comfortable reading Bash scripts, and review its content).

Running scripts directly

It is generally a bad idea to run scripts downloaded directly from the internet as one cannot be sure they do not contain malicious code. In any environment other than scratch environments like these we're creating and destroying for these lessons you should download the script, validate its content, then store it in your own repository so that you have a 'known good copy' to use (I discuss setup and use of controlled repositories in *Devops from Scratch (Technical Support)*[Boo20a]).
In the following we will use the script direct from `https://bootstrap.saltproject.io` as risk of compromise is low and will be localised to the VM we are working on.

Setup to Follow Along

If you have not done so already, setup according to §1.3.1 and, if you need to archive any current class files §1.3.2.1.

```bash
cd sfs
```

On Mac/Linux:

```bash
cp -r course-material/sfsP2010cp010 classroom
```

On Windows:

```bash
xcopy course-material\sfsP2010cp010 classroom /E
```

All Platforms:

```bash
cd classroom
vagrant up
```

Connect to the new VM.

```bash
vagrant ssh
```

You should now be in the VM at a shell prompt. We will now download and run the Salt bootstrap script.

```bash
curl  -L https://bootstrap.saltproject.io | sudo sh -s -- 
  -X stable 3004.1
```

Let's break that line down.

`curl  -L https://bootstrap.saltproject.io`: `curl` is a utility for accessing resources form URLs, in this case `https://bootstrap.saltproject.io`. The `-L` tells `curl` to follow any redirects it receives (basically, this ensures `curl` tracks down the script even if it has moved on the server).

The pipe (`|`) tells the shell 'take everything output from the command on the left side and pass it to the input of the command on the right'.

`sudo` tells the shell to run the following command as the privileged user `root`. The commant to be run is `sh`, which runs a new shell and the `-s` tells `sh` to read commands from its input stream (in this case it will read the script `curl` is downloading). The `--` tells `sh` that characters to its right are to be parred as parameters to the bootstrap script rather than being options passed to the `sh` command.

The upshot of this is that the bootstrap script is run as `root` passing the options `-X stable 3004.1`.

The `-X` option tells the bootstrap script to not start any services (by default it assumes we want to run the minion service, we get to this later Chapter 7). The two arguements tell the bootstrap script to use the `stable` repository and to install version `3004.1` of Salt.

When run you will see a stream of log output showing the bootstrap script installing the components required to run Salt and the Salt itself.

You can confirm Salt is installed by checking that `salt-cmd` is installed.

bash
```bash
command -v salt-call
```

This will show the path to the installed `salt-call` command if everything installed.

command -v salt-call
```
/usr/bin/salt-call
```

We can perform a very somple test to ensure Salt will run.

bash
```bash
sudo salt-call --local test.ping
```

This runs `salt-call` locally[1] (the `--local` option) calling the state module `test.ping` (a simple, and for local Salt, largely pointless, test that Salt is present and responding). The return should be `True`.

sudo salt-call -local test.ping
```
local:
  True
```

[1]By default, Salt require privilege to run, hence the `sudo`. We will see later how we can allow users limited access to Salt commands without the need for them all to hold root privilege.

Setup to Follow Along

Tear-down to Follow Along
Unusually, we no longer need this VM so you can shut it down and remove it from your host computer.
Assuming you are at the command prompt on the classroom VM, quit back to your host.

```bash
exit
```

On your host computer.

```bash
cd sfs/classroom
vagrant destroy -f
```

Chapter 6

Introducing `salt-call`

`salt-call` invokes Salt locally.

6.1 Starting at the End

Setup to Follow Along

If you have not done so already, setup according to §1.3.1 and, if you need to archive any current class files §1.3.2.1.

```bash
cd sfs
```

On Mac/Linux:

```bash
cp -r course-material/sfsP2020cp010 classroom
```

On Windows:

```bash
xcopy course-material\sfsP2020cp010 classroom /E
```

All Platforms:

```bash
cd classroom
vagrant up
```

Once your classroom VM is available connect using SSH.

```bash
vagrant ssh
```

This VM is preloaded with a Salt configuration to build this server into a simple web server consisting of `nginx` serving a simple static website form `/var/www/html`.

Run Salt locally on this server using:

```bash
sudo salt-call state.highstate
```

This command will take a while to run, during which nothing will appear to happen. On completion you will see results dumped to the screen.

On your host computer, open a web browser and visit `http://localhost:35555`. You will see a simple web site being served from the VM.

6.1.1 What did we just do?

When the VM was set up several things were preconfigured (if you're curious—and you should be—check out `sfs/classroom/provisioning/build` on your host computer[1]).

- Some tools required for the setup scripts were installed.
- `salt` was installed using the bootstrap script we used in Chapter 5.1.
- Some `salt` configuration files where put in place.
- A set of files describing the configuration we want on our machine where copied from a Git repository.
- The website files where put in place.

You then logged into this prepared machine and used `salt-call` to instruct Salt to ensure that the machine was configured according to the specification copied during VM setup.

We will now spend the next few chapters learning how to set this up from scratch and in doing this we will learn a lot of the core ideas behind Salt's configuration management functionality.

[1]If you are not confident reading Bash scripts check out *Programming from Scratch*[Boo22].

Part III

Configuring Remote Machines

Author Note

`salt-ssh`. Running arbitrary commands remotely. Running configurations remotely.

Configuring a local machine allowed us to focus on all Salt's core configuration management functionality. This can be useful but most organisations use more than one server and those servers are often configured to work together. Configuring multiple servers as individual servers is possible but problematic. Salt provides `salt-ssh` for configuring remote machines using your enterprise SSH infrastructure.

Part IV

Controlling Many Machines

Author Note

Setting up Master/Minion. We switch to ‘control’ because now we’re stepping into a ‘live’ environment.
Doing away with SSH infrastructure.
Proxy minions.
Access control.
Monitoring; beacons, reactor.
‘self-healing’ infrastructure. Increased security.
Multi-master setup.

`salt-ssh` is useful for directly distributing your multi-server configuration over your SSH infrastructure. Now now we move on to Salt’s Master/Minion facility, this allows distribution of a configuration across multiple machines without an SSH infrastructure. It also allows much more; monitoring of your infrastructure combined with reacting to event in your infrastructure allows you to create self-healing systems.

Chapter 7

Installing Master/Minion Salt

Part V

Custom Salt

Author Note

Custom plugins, modules, etc.

Sometimes you find Salt is missing some facility you need. No problem. Add the functionality. Salt's plug-in architecture provides a core bus via which events and data are passed between a Master and one or more Minions, beyond this all functionality is provided by plug-ins.

Bibliography

[Boo20a] Mark Bools. *Devops from Scratch (Technical Support)*. From Scratch. 2020. URL: `https://saltyvagrant.com/members/books/devops/devops.html`.

[Boo20b] Mark Bools. *Git from Scratch*. From Scratch. 2020. URL: `https://saltyvagrant.com/members/books/git/git.html`.

[Boo20c] Mark Bools. *Vagrant from Scratch*. From Scratch. 2020. URL: `https://saltyvagrant.com/members/books/vagrant/vagrant.html`.

[Boo22] Mark Bools. *Programming from Scratch*. From Scratch. 2022. URL: `https://saltyvagrant.com/members/books/programming/programming.html`.

Index