Getting DevOps off the Ground

An Ubuntu + SaltStack + gitfs + PXE Infrastructure Build

Chris Malone
Nuvation

Table Of Contents

Foreword

Content Summary

This is a technical document which details the process of building a computer infrastructure capable of rapid and repeatable deployment to both virtualized and bare metal machines. It is written in an additive manner such that each step can be understood both individually and as part of a greater whole. All content starts as a baseline of detailed, manual steps which are subsequently automated using open-source software. This culminates in an automated infrastructure built using well documented code.

The companion code for this document is maintained here:
https://github.com/love2scoot/pxesaltbase-formula

Intended Audience

Although this document details the full process of building an integrated deployment infrastructure, it is written in such a way that all individual components are fully detailed in and of themselves. As such, this content may be applicable to many different parties.

For example, the overall infrastructure build is probably most relevant to IT professionals or software developers. However, an end user looking for a good guide on automating Ubuntu installations may find the information on preseed files handy.

While this content is relatively technical in nature, it is presented in such a way to maximize the benefit to the largest audience possible.

tl;dr

Want to cut straight to the chase? Go to the final Build Walkthrough.

License

Introduction

The Promise of DevOps

Over the last 10 years the concept of “IT” has seemed to bifurcate into two distinct fields: infrastructure support and DevOps. While the former remains a fundamental component to a successful IT department, implementing DevOps has become an essential approach in order to keep pace with the speed of modern deployments.

> **Wikipedia:**
> **DevOps** is a term used to refer to a set of practices that emphasizes the collaboration and communication of both software developers and other information-technology (IT) professionals while automating the process of software delivery and infrastructure changes. It aims at establishing a culture and environment where building, testing, and releasing software can happen rapidly, frequently, and more reliably.

While the definition of DevOps is easy to understand, embodying this concept in a set of integrated tools is a different matter. There are many ways to approach this, and a multitude of possible solutions. For the purpose of this document we have selected one path and attempted to detail the process of designing, implementing, releasing, and testing a basic DevOps infrastructure.

Goals

This document seeks to meet a basic set of goals, which are structured around providing the highest level of benefit both discretely and, more importantly, as a comprehensive solution to a basic DevOps infrastructure.

Unified, piecewise documentation

When using internet forums to researching answers to IT challenges, it is easy to find solutions to specific queries, but seldom are comprehensive approaches available. One of the goals of this document is to unify all documentation for the given approach such that each discrete component can stand on its own, while still informing the greater whole. In this way, users looking for a subset of this solution can come away with as much benefit as those who are seeking a more comprehensive approach.

Repeatable deployment

In some cases, excellent documentation is available, but when an attempt is made to replicate an approach, the user runs into errors or unexpected results. The second goal of this document is to detail processes *as well as all major version data* for the components involved. In this way the end user is not left stranded due to version creep, and will be able to deploy each component in a predictable and repeatable manner.

Host agnostic

While virtualization is now a fundamental part of most DevOps approaches, it can be equally important to support implementation on bare metal. Within software development, the use of virtualization can help a team rapidly iterate on source. In contrast, hardware development work will oftentimes require the direct connection of "hardware in the loop" to a build system. The third goal of this document is to design a solution such that deployment can be agnostic to either a VM or bare metal.

Rapid development

Finally, this document should help illustrate the benefits inherent in the DevOps process by dogfooding the very processes detailed within. The end product will not only be an outline for how to deploy this infrastructure, but will allow for leveraging this infrastructure to build itself. This will allow the end user to rapidly iterate on this content and deploy a fully integrated DevOps infrastructure.

Building Blocks

While the goals of implementing DevOps seem clear, there are several methods that can be used to deliver on this promise. In order to provide a comprehensive platform, we will need to integrate several different software elements. In this section we provide a summary of each piece of software and how it will integrate into this platform.

VMWare Workstation Player

In order to kick off the development cycle, we need an easy way to install our base OS. This process could alternately use bare metal, but the better choice here is to use a hypervisor, specifically VMWare Workstation Player. This hypervisor will run atop Windows or Linux and help keep the development cycle simple. Note that the use of this hypervisor is only necessary during the building of our PXE solution, after which the PXE server itself will be used for provisioning new machines.

Ubuntu

As a very widely supported Linux variant with extensive online documentation, Ubuntu is an optimal choice for an open source OS. Not only will Ubuntu be used as the basis for configured machines, but it will additionally be used as the base on which our DevOps infrastructure is built.

SaltStack

Most approaches to DevOps utilize a piece of software for Configuration Management, essentially automating the build, deployment, and continuous configuration of machines across an enterprise. This area is relatively well established with several different software implementations. Examples here include Ansible, Puppet, Fabric, Chef, and SaltStack (among others). The are numerous articles written comparing these options, but for the purposes of this document we will be using SaltStack as our Configuration Management software of choice.

gitfs

While several version control systems exists, git is well established and has excellent community support and documentation. Additionally, SaltStack has integrated support for git through use of gitfs. gitfs allows SaltStack to directly access source stored on a git server. This approach allows for a very rapid development cycle that scales well from singular developers up to teams. The use of gitfs within SaltStack also simplifies the process of configuring and deploying systems at scale.

PXE

Preboot e**X**ecution **E**nvironment allows machines (physical or virtual) to boot using a configuration provided over a network. In our case we will be using PXE as the most fundamental service on which new machines will be built. PXE essentially allows us to boot to a menu within which we can provide a variety of specific configuration options. These options will allow a machine to boot in a specific and repeatable way.

Development

Approach to Development

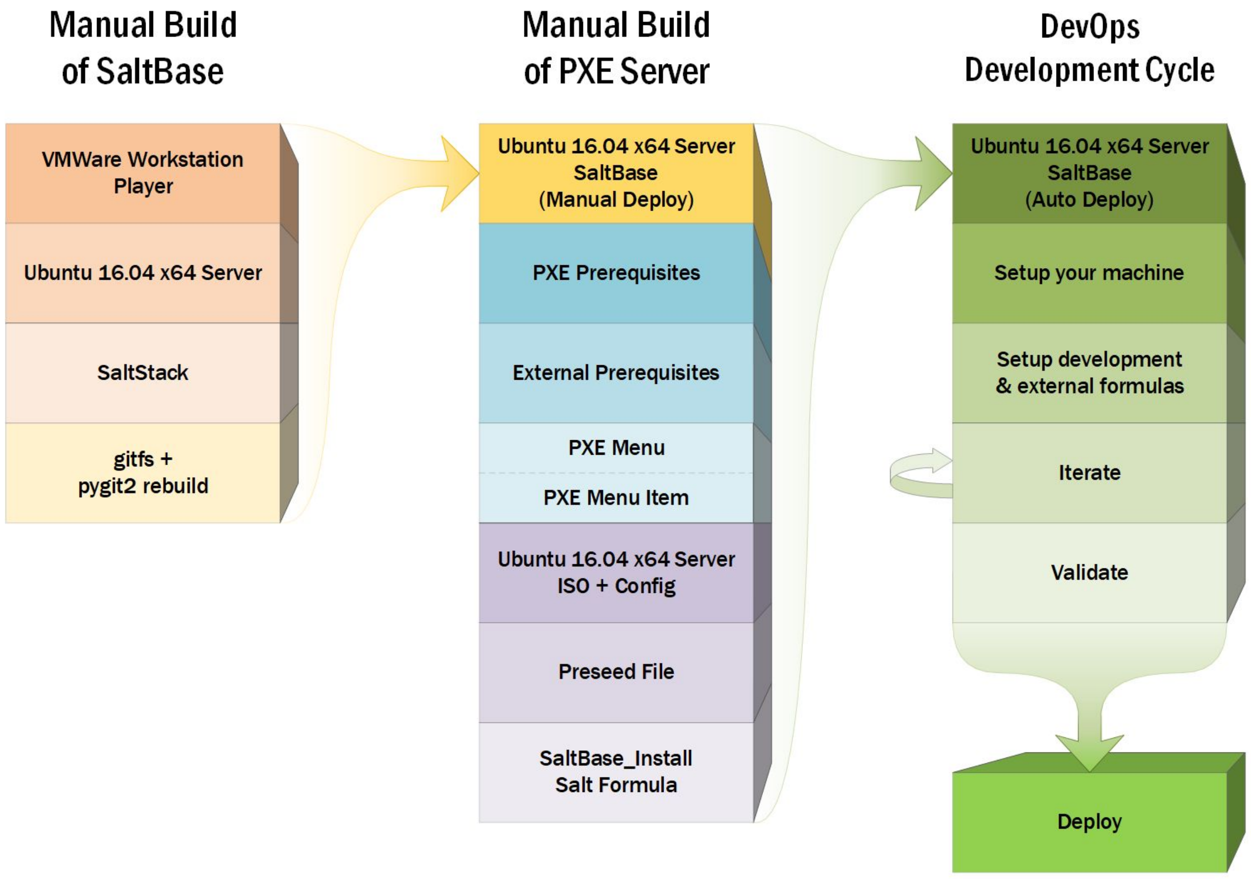
A DevOps environment allows the developer to leverage standardized base configurations, integrated version control, rapid iteration, and verification cycles to their maximum advantage. Once this environment is brought to bear on development and configuration, developers will immediately be able to leverage the efficiency gains it promises. We will explore each of these topics, and provide some suggested best practices.

Deployment Methods

While the *development* cycle can be completed on virtual hosts, *deploying* a system configuration should be host agnostic. When considering deployment methods, the advantages and limitations of deploying to both virtualized hosts and bare metal hardware should be taken into consideration. This deployment process needs to be as optimized as possible while ensuing that the resulting machine configuration performs identically on both virtualized and bare metal hosts.

The Approach: Layering

In order to best emulate the process of developing a DevOps infrastructure from scratch, our solution will be built one layer at a time. After each layer is built manually, we can then work toward automating that process. We then add another layer (and so on) until we arrive at a comprehensive automated solution. The figure below provides a high level view of the building blocks that integrate into our solution.


Manual Build of SaltBase
VMWare Workstation Player
Ubuntu 16.04 x64 Server
SaltStack
gitfs + pygit2 rebuild
Manual Build of PXE Server
Ubuntu 16.04 x64 Server SaltBase (Manual Deploy)
PXE Prerequisites
External Prerequisites
PXE Menu
PXE Menu Item
Ubuntu 16.04 x64 Server ISO + Config
Preseed File
SaltBase_Install Salt Formula
DevOps Development Cycle
Ubuntu 16.04 x64 Server SaltBase (Auto Deploy)
Setup your machine
Setup development & external formulas
Iterate
Validate
Deploy

Part I: Manual Build of SaltBase

The first major process is the manual configuration of a **SaltBase** Virtual Machine. This VM is built using open tools which, when correctly configured, will result in a basic template (or SaltBase) on which the next major step of the DevOps infrastructure can be built.

Part II: Manual Build of PXE Server

Using the SaltBase output from Part I, a manually configured **PXE server** will be built. This PXE server will include a boot configuration which converts the *manual* build process detailed within Part I to an *automated* deployment of a SaltBase machine. This auto deployed SaltBase machine will, in turn, be the base on which the next part of our DevOps solution will be built.

Part III: DevOps Development Cycle

Using the auto deployed SaltBase, the detailed **DevOps Development Cycle** will be documented. This development cycle will suggest best practices and articulate the benefits afforded by the Ubuntu + Salt + gitfs + PXE approach.

PXE Feature Add

Now that the basic functionality of the DevOps infrastructure is fulfilled, we can look into adding additional functionality to the PXE server itself with additional boot menu options.

Dogfooding: SaltStack build of PXE

Now that the entire build process has been outlined manually (including the additional features), we will dogfood the process by building the very infrastructure documented herein. This section will show how to build a DevOps infrastructure using SaltStack by automating the manual steps detailed in Part II and in the Feature Add. The output will be the fully automated build of a new PXE server.

Looking Forward

Taking stock of all the content within this document, we'll look at areas of enhancement and avenues of further research which could potentially be layered on top of this infrastructure.

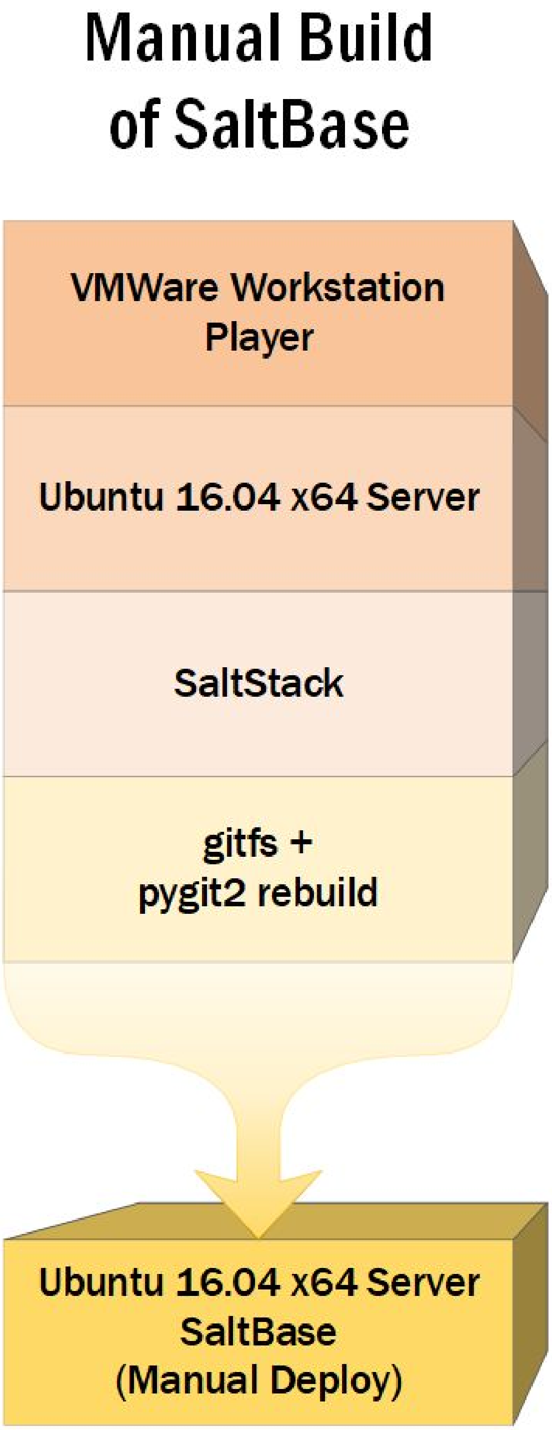
Appendix A: Detailed Breakdown of Helper Scripts

SaltBase machines have (3) helper scripts loaded into /root by default. This section will look into each script, how it is written, and how it streamlines the setup of a new SaltBase machine.

Appendix B: Quick SaltStack Primer

Although this document makes extensive use of SaltStack, explaining this configuration management tool in-line would be burdensome. This appendix can be used as reference for basic structure, nomenclature, formatting, topology, and development practices centered around SaltStack.

Part I: Manual Build of Salt Base



The first major process is the manual configuration of a **SaltBase** Virtual Machine. This VM is built using open source tools which, when correctly configured, will result in a basic template (or SaltBase) on which the next major step of the DevOps infrastructure can be built. This process can be broken down into:

- Configuration of a VM using VMWare Workstation Player
- Installation of the Ubuntu 16.04 x64 Server OS
- Configuration of SaltStack packages
- The configuration of gitfs and the building of supporting packages
- The manual deployment of an Ubuntu 16.04 x64 Server SaltBase machine

VMWare Workstation Player

Why Player

Using a hypervisor for the installation of our development environment keeps our approach simple and flexible. VMWare Workstation Player will run atop Windows or Linux, allowing these first steps to be taken on an existing computer. Note that the use of a hypervisor is only necessary in the building of our PXE solution, after which the PXE server itself will be used for provisioning.

Install VMWare Workstation Player

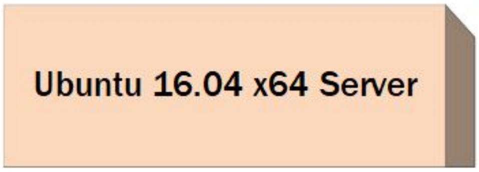
You can download VMWare Workstation Player from the [VMWare site](). Follow the [simple instructions]() for installation. At the time of this writing, VMWare Workstation Player was on version 12.5.2.

Building the Development VM

1. Start VMWare Workstation Player
2. Click the **Create a New Virtual Machine** button on the right.
3. Leave the radio button on **I will install the operating system later** and click **Next**.
4. On the **Guest Operating System** window, change the radio button to **Linux** and change the **Version dropdown** to **Ubuntu 64-bit**. Click **Next**.
5. On the **Name the Virtual Machine** window, change the values in the **Virtual Machine Name** field and the **Location** field to fit your needs. Click **Next**.
6. On the **Specify Disk Capacity** window, change the **Maximum Disk Size (GB)** field to at least 50GB. If you prefer to have the disk stored as a single file, make this change. Click **Next**.
7. On the **Name the Virtual Machine** window, click on **Customize Hardware**.
    a. In the Hardware list, select **Network Adapter**.
    b. On the right hand side, under **Network Connection** section, change the radio button to **Bridged**.
    c. Click the **Close** button to complete the modification of the Virtual Hardware.
    d. Click the **Finish** button to complete the creation of the Virtual Machine.

Ubuntu

Why Ubuntu



Ubuntu is a well supported and well documented Linux distribution. While it lacks an association with the default enterprise approach taken by RHEL / CentOS, it works well for development and can be deployed in production using LTS releases. Ubuntu provides a good common ground between development teams and IT teams, helping move toward the goal of a full DevOps solution.

Installation media

Download Ubuntu 16.04 x64 Server ISO from the Ubuntu site. While the instructions in this document *may* work for newer versions of Ubuntu Server, this content was originally built for v16.04.2.

Mounting the ISO

1. If not already open, start VMWare Workstation Player.
2. From the list of virtual machines, select the VM created in the section above, and select **Edit virtual machine settings** from the menu on the right.
3. In the Hardware list, select **CD/DVD (SATA)**.
4. On the right hand side, under the **Connection** section, change the radio button to **Use ISO image file**.
5. Click the **Browse** button and find the Ubuntu ISO you downloaded above and select this file.
6. Click the **Close** button to complete the modification of the Virtual Hardware.

Manual Install vs preseed file

Although we could use a preseed file to quickly and easily build the VM, we're going to step through the setup of the development VM manually as this is in line with the layering approach espoused above. We will leverage a preseed file for automated installs later in this document.

Manual Install Process

1. If not already open, start VMWare Workstation Player.
2. Select your Development VM from the list of virtual machines and start the VM
3. Select your keyboard type from the list
4. Select **Install Ubuntu Server**
5. Select your Language from the list

6. Select your location from the list
7. Do NOT detect keyboard layout by selecting **No**
8. Select **English (US)** when asked for keyboard
9. Select **English (US)** when asked for keyboard layout
10. Specify a hostname when prompted
11. Specify the Full name for the new user as **temp**
12. Specify the Username for the new user as **temp**
13. Specify the password. Re-enter to confirm
14. When prompted to configure your home folder for encryption, select **No**
15. The installer will attempt to find your timezone. Confirm if correct, change it if it guessed incorrectly.
16. Select **Guided - use entire disk** when prompted for the partition scheme
17. Select the Virtual Disk for use with the OS (there should only be one option)
18. If prompted, select **Finish partitioning and write changes to disk**
19. Confirm the changes should be written to the disk by selecting **Yes**
20. Leave the HTTP proxy field blank and **Continue**
21. Select **No automatic updates** when prompted
22. On the **Software Selection** screen, leave this as the default (standard system utilities only) and select **Continue**.
23. Confirm with **Yes** that you wish to install GRUB to the master boot record
24. Select **Continue** to reboot the Virtual Machine

Basic OS Setup

After reboot, a few basic steps should get the VM ready for configuration.

1. Login using the **temp** user created at installation
2. Enable the root account (specify the root password here)

```
sudo passwd root
```

3. Logout of the temp user

```
exit
```

4. Login as **root**
5. Remove the temp user

```
userdel -r temp
```

6. Enable SSH for root by changing the config file using a simple sed replace

```
sed -i "s/^PermitRootLogin.*/PermitRootLogin yes/g" /etc/ssh/sshd_config
```

7. Restart the ssh service

```
service ssh restart
```

From this point, the Virtual Machine can be either accessed directly or via SSH using root.

SaltStack

Why SaltStack

Most approaches to DevOps utilize a piece of software for Configuration Management, essentially automating the build, deployment, and continuous configuration of machines across an enterprise. We have chosen SaltStack as our Configuration Management software of choice for the following reasons:

- SaltStack is very actively developed, with extensive documentation and active forums.
- The YAML markup is easy to understand and visually parse, which simplifies development.
- SaltStack is powerful and flexible, minimizing the reliance on external scripts or bash commands.
- When configured correctly, SaltStack can seamlessly leverage formulas directly from network connected sources. This enables the configuration of salted machines using any source available, delivering on code reuse at scale.
- The capacity to continuously configure machines allows SaltStack to scale all the way from local development to enterprise, delivering on many facets of DevOps.

From the above characteristics, we have seen the following benefits:

- Repeatable, programmatic builds of engineering virtual machines / toolchains.
- Easy maintenance and improvement of existing configurations through use of simple markup and scripting committed to source control.
- Elimination of copying entire virtual machines to engineering teams (especially important when considering projects that span geographically distant design centers)
- Rapid development of machine configuration through the easy, iterative cycle enabled through use of SaltStack.
- Easy reuse of configuration code, allowing for increased momentum and a shortened development cycle.

Side Note: Included in Appendix B is a Quick SaltStack Primer which briefly covers the concepts, nomenclature, and standards associated with this Configuration Management tool. If you need an introduction to (or review of) Saltstack, this section should come in handy.

Installing SaltStack

SaltStack is actively developed, and as such, the SaltStack packages included within the regular OS package repositories can fall out of date rapidly. The preferred method for installing SaltStack, therefore, is to update the package sources list to include a repository hosted by the SaltStack developers.

Apt source changes

The first step here is to add a new apt source to the list on the OS, allowing for the installation of the latest salt packages..

1. Login to the VM either directly or via SSH using root.
2. Add the latest key for the SaltStack repo (16.04)

   ```
   wget -O -
   https://repo.saltstack.com/apt/ubuntu/16.04/amd64/latest/SALTST
   ACK-GPG-KEY.pub | apt-key add -
   ```

3. Add the repo to the sources list (16.04)

   ```
   echo deb
   http://repo.saltstack.com/apt/ubuntu/16.04/amd64/latest xenial
   main >> /etc/apt/sources.list.d/new1604saltstack.list
   ```

Installation of packages

Next we install the SaltStack packages.

1. Update the local package listing with the new packages

   ```
   apt update
   ```

2. Install the salt packages

   ```
   apt install -y salt-common salt-master salt-minion
   ```

Configure VM as Self-Mastered

After installation, we must perform a few short configuration changes to setup the VM as a self-mastered minion.

1. By default, a Salt Minion will try to connect to the DNS name "salt". We are using a self mastered approach for development, and as such we only need to add the DNS name to the hosts file. This will essentially allow the local salt minion to point to the local salt master.

```
nano /etc/hosts
```

2. Add the entry for salt (new content in light green)

```
127.0.0.1    localhost salt
```

3. Start the services

```
service salt-master start
service salt-minion start
```

4. Next we'll need to add the key of the minion to the master. The easiest way to do this is simply have it auto-accept all keys without confirmation.

```
salt-key -y -A
```

Build basic Salt Master Config

Next we need to setup SaltStack with a basic configuration. Note, that all files within /etc/salt/master.d are concatenated and interpreted along with the standard configuration file located at /etc/salt/master. This gives us the advantage of being able to build a very small config file for salt which is easy to maintain *without* touching the default config file.

1. Create a new salt-master config file:

```
nano /etc/salt/master.d/devops_standard.conf
```

2. Add the following lines:

```
#The root location for local salt source
file_roots:
  base:
    - /srv/salt

#The hash to use when discovering the hash of a file on the
master server
hash_type: sha256

#The root location for local pillar source
pillar_roots:
  base:
    - /srv/pillar

#Specify the search order of the backend file systems
```

```
fileserver_backend:
  - roots
```

3. Restart the Salt master in order to pick up these changes:

```
service salt-master restart
```

4. Create the folders that contain the salt formulas and pillars (in the default configuration)

```
mkdir /srv/salt
mkdir /srv/pillar
```

5. Build the default top.sls file for the master and a placeholder for the pillar

```
echo "base:" > /srv/salt/top.sls
echo "  '*':" >> /srv/salt/top.sls
cp /srv/salt/top.sls /srv/pillar/top.sls.example
```

Testing the SaltBase Machine

We run through a few basic tests to verify function of the newly installed and configured salt.

1. Verify the versions of the installed packages

```
salt-master --version; salt-minion --version
```

2. Verify the status of the salt services

```
service salt-master status
service salt-minion status
```

3. Verify communication between the master and the minion using the command

```
salt '*' test.ping
```

Build and apply a basic formula

In order to fully test the functionality of the self-mastered minion we need to build a basic formula and apply it to the machine using a highstate. For this test we'll make use of the file.touch state.

1. Create the formula.

```
nano /srv/salt/testformula.sls
```

2. Drop the following source into this file

```
# First formula
first_state:
  file.touch:
    - name: /tmp/testfile
```

3. Save the file and exit nano
4. Edit the top.sls salt file to include the new formula

```
nano /srv/salt/top.sls
```

5. Add a single line to add testformula to the top file (removing all other formulas). It should now read:

```
base:
  '*':
    - testformula
```

6. Save the file and exit nano
7. Apply the state

```
salt '*' state.highstate
```

Read through the results. Note the reported success of the state and creating the file, this should be displayed in green. Also note that the state is reported using the name assigned to the state within the formula (`first_state`). Had there been two states within the formula each would have been listed in the results.

8. Apply the state again

```
salt '*' state.highstate
```

Note the reported success of the state but also note that the no changes were made since the file already exists. This success without change should be displayed in blue.

9. Clean up the test by removing the "`- testformula`" line from top.sls.

```
nano /srv/salt/top.sls
```

It should now read:

```
base:
  '*':
```

This now completes the setup of salt on our machine. Next, we'll add gitfs support to add additional flexibility to our machine.

gitfs

Why gitfs

Building on our self mastered salt machine, we next layer gitfs. gitfs allows a master to directly use salt formulas stored on a git server. There are several advantages to this approach:

- Salt source can be edited and pushed to a central git server, allowing the developer to choose their environment of choice
- Using a git server allows developers to verify salt formulas across several different machines by deploying across different hardware / hypervisors
- gitfs allows developers to leverage formulas across teams by using salt stored both locally (on the master) and remotely (on the git server) simultaneously
- gitfs allows developers to leverage formulas stored on public repositories, like the SaltStack formulas on GitHub

The pygit2 problem

Adding gitfs functionality to our machine should be a simple matter, but there is a challenge within one of the dependent packages, pygit2. pygit2 allows python (fundamental to salt) to interact directly with git servers. The pygit2 package is dependent on a library called libgit2 which is compiled to only support the SSH protocol by default. Ideally, libgit2 would support *both* SSH and HTTP/S, but no package repository or ppa makes this available. We are left with only one option: downloading and compiling libgit2 from source (including the dependencies for HTTP/S functionality) before installing pygit2. This section will cover how to compile the libgit2 package, install pygit2, and verify the capabilities of pygit2 after installation.

Solution: Compile from source

1. Install the build tools and dependent packages

```
apt install -y build-essential cmake libssh2-1-dev
python-dev python-pip python-cffi libssl-dev libffi-dev
pkg-config libcurl4-openssl-dev libhttp-parser-dev
```

2. Download, uncompress, and build libgit2 then install pygit2

```
cd /tmp
wget
https://github.com/libgit2/libgit2/archive/v0.25.0.tar.gz
tar -zxf v0.25.0.tar.gz
cd ./libgit2-0.25.0
cmake .
make
make install
ldconfig
pip install pygit2==0.25.0
```

Side Note: the **pip install pygit2** line must be locked to the same version number as libgit2 which is being built from source.

Since we are downloading libgit2 using the line:

```
wget https://github.com/libgit2/libgit2/archive/v0.25.0.tar.gz
```

The corresponding pip install line reads:

```
pip install pygit2==0.25.0
```

Test and verify

1. Verify that the newly installed pygit2 allows for use of both https and ssh protocols. (both bool commands should return TRUE)

```
python
```

```
import pygit2
bool(pygit2.features & pygit2.GIT_FEATURE_HTTPS)
bool(pygit2.features & pygit2.GIT_FEATURE_SSH)
exit()
```

Add config changes for gitfs

Now that pygit2 is correctly installed, we need to make the configuration changes to the salt master in order to leverage remote gitfs sources.

1. Edit the config file:

```
nano /etc/salt/master.d/devops_standard.conf
```

2. Add the lines for the gitfs configuration (new lines in light green). It should now read:

```
#The root location for local salt source
file_roots:
  base:
    - /srv/salt

#The hash to use when discovering the hash of a file on the
master server
hash_type: sha256

#The root location for local pillar source
pillar_roots:
  base:
    - /srv/pillar

#Specify the search order of the backend file systems
fileserver_backend:
  - roots
  - git

#Specify which method provides gitfs access to salt
gitfs_provider: pygit2
```

3. Create a placeholder file to hold the list of remote gitfs repositories.

   ```
   nano /etc/salt/master.d/gitfs_remotes.conf
   ```

   While technically gitfs remotes can be stored in the `devops_standard.conf` file, it's better practice to keep these in their own file.

4. Add the following content to the `gitfs_remotes.conf` file and save it.

   ```
   #gitfs_remotes:
   #   - https://gitsource.bobbarker.com/happy.git:
   #     - user: for protected sources
   #     - password: for protected sources
   #     - root: salt
   ```

5. Restart the Salt master in order to pick up these changes:

   ```
   service salt-master restart
   ```

Apply a state from a gitfs source

Now that gitfs support (with both SSH and HTTP/S) is available to the machine, we can perform a full test by applying a state stored on a remote server. In this case, we'll use the ntp formula stored on the SaltStack formulas on GitHub

1. Open our gitfs_remotes.conf file for editing

   ```
   nano /etc/salt/master.d/gitfs_remotes.conf
   ```

2. Remove the # on the first line and add the following content to gitfs_remotes.conf. **Save** the file upon exit. (new content in light green)

   ```
   gitfs_remotes:
   #   - https://gitsource.bobbarker.com/happy.git:
   #     - user: for protected sources
   #     - password: for protected sources
   #     - root: salt
     - https://github.com/saltstack-formulas/ntp-formula.git
   ```

3. Edit the top file to include the new formula

   ```
   nano /srv/salt/top.sls
   ```

4. Add the following content to top.sls and **save** the file. (new content in light green)

   ```
   base:
     '*':
       - ntp
   ```

5. Restart the Salt master in order to pick up these changes:

   ```
   service salt-master restart
   ```

6. Update the local cache of remote sources (should return **TRUE**):

   ```
   salt-run fileserver.update
   ```

7. Finally, we run a highstate and ensure that the formula is applied correctly

   ```
   salt '*' state.highstate
   ```

At this point, Salt should attempt to apply the ntp formula to the salt-minion and report back results. Assuming the results are positive, this validates using formulas from remote gitfs sources. The above changes can be backed out of, if desired.

Future considerations

It is possible that future pygit2 packages will be cross compiled to integrate both SSH and HTTP/S protocols. To test this, simply install the pygit2 package (`apt-get install python-pygit2`) and run the Test and verify process above.

Save Point: Export SaltBase to .ova

Creating a universal OVA template on Windows

In order to create an OVA template that will be compatible with the maximum number of hypervisors, use the following guidelines with VMWare Workstation Player.

Would you like to save here?
Yes
No

1. Ensure the VM version is no greater than 11. If it is greater, open the .vmx file in a text editor and change the following line:

```
virtualHW.version = "11"
```

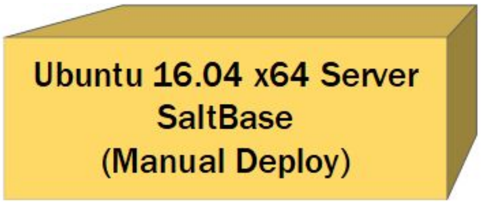
2. Open the virtual machine in VMWare Workstation Player (do not start the VM), edit the VM settings, and remove the SATA CD-ROM device (if present). Close VMWare Workstation Player.
3. Browse to the folder containing the VM
4. Open the .vmx file using a text editor
5. Find and remove all lines that include SATA or IDE, then save the .vmx file. Here are some examples:

```
sata0.present = "TRUE"
sata0:1.present = "TRUE"
sata0:1.fileName = "C:\VMWare\ubuntu-16.04-server-amd64.iso"
sata0:1.deviceType = "cdrom-image"
```

Exporting the OVA Appliance on Windows

1. Find the path to your VM
2. Open a command prompt (elevated). **NOTE:** This does not work in PowerShell.
3. Change to the OVFTool folder to execute the ovftool command

```
cd "C:\Program Files
(x86)\VMware\VMware Player\OVFTool"
```



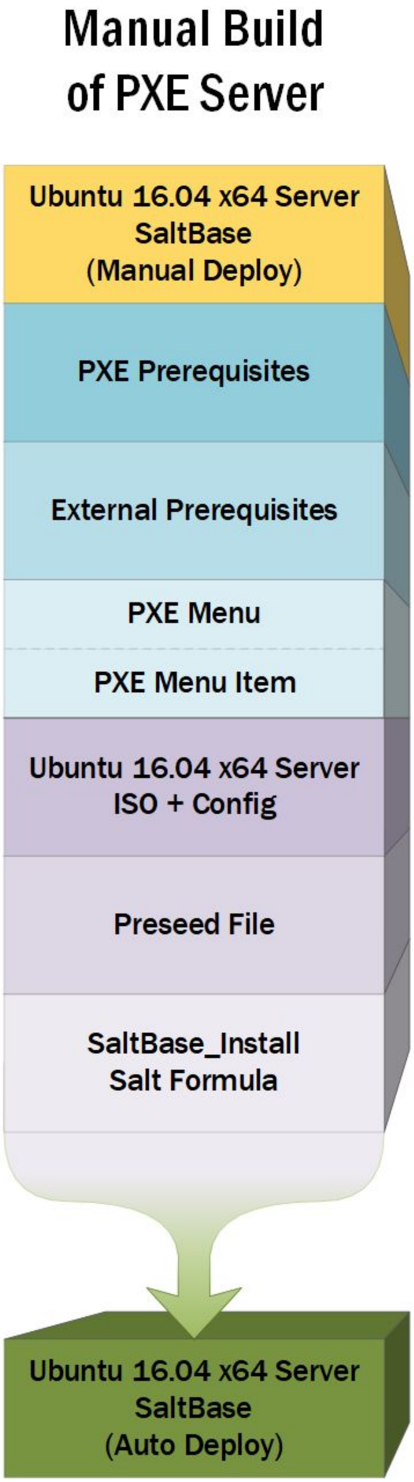
4. Convert your VM to an OVA Template

```
ovftool --compress=9 “C:\the path to your VM\your
VMname.vmx”  "c:\newdirectory\name.ova"
```

The First Finish Line

We now have our first real output from our DevOps approach: a manually built VM of a SaltBase machine captured within an .ova file. This both a) lays the groundwork for the first steps into automation and b) gives us an outline on which to expand that automation. Using this .ova as the foundation, we can manually build a full PXE server allowing for the *automated* configuration and deployment of a SaltBase machine.

Part II: Manual Build of PXE Server


Manual Build of PXE Server
Ubuntu 16.04 x64 Server SaltBase (Manual Deploy)
PXE Prerequisites
External Prerequisites
PXE Menu
PXE Menu Item
Ubuntu 16.04 x64 Server ISO + Config
Preseed File
SaltBase_Install Salt Formula
Ubuntu 16.04 x64 Server SaltBase (Auto Deploy)

Using the SaltBase output from Part I, a manually configured **PXE server** will be built. This PXE server will include a boot configuration which converts the manual build process in Part I to an automated deployment of a SaltBase machine. This process can be broken down into:

- Manually deploying an Ubuntu 16.04 x64 Server SaltBase VM created as the output of Part I
- Installation and configuration of the prerequisites for a PXE server
- Configuration of external prerequisites
- Configuration of a basic PXE menu
- Expansion of the PXE menu with specific menu items
- The download and configuration of an Ubuntu ISO file on the PXE server. **This automates the OS media setup steps from Part I**
- The development of a Debian preseed file for the automated installation of the Ubuntu OS. **This automates all of the manual Ubuntu OS installation steps from Part I, in addition to all basic configuration steps for SatlStack**
- The development of the SaltBase_Install Salt Formula which will complete the configuration of a SaltBase machine. **This automates all manual configuration steps applied on top of the OS from Part I**
- The *automated* deployment of an Ubuntu 16.04 x64 Server SaltBase machine

What is PXE

As mentioned in the introduction, PXE is an abbreviation of **P**reboot e**X**ecution **E**nvironment. It allows machines (physical or virtual) to boot using a configuration provided over a network.

Our goal is to configure the PXE server with a menu which allows a user to select from multiple boot options. One of these options will configure and automatically deploy an Ubuntu 16.04 x64 Server configured as a SaltBase machine.

SaltBase Manual Deployment

Taking the output of Part I above, we can manually deploy a **SaltBase** machine as the fundamental building block for our PXE Server. Although it is possible to use a different hypervisor, we will use VMWare Workstation Player for this example. The process below will outline steps for deploying the .ova template created above to a new Virtual Machine..

1. Start VMWare Workstation Player
2. Click the **Open a Virtual Machine** button on the right.
3. On the **Open Virtual Machine** window, browse to the location of the .ova template completed above at the end of Part I. Double click on the .ova file to open it.
4. On the **Import Virtual Machine** window, you can customize both the name and path for the new VM. Click the **Import** button to complete the import process.
5. Wait while VMWare imports the VM from the .ova template.
6. Click the **Play virtual machine** button on the right to start the VM.

PXE Server Prerequisites

There are a few packages which must be installed in order to build the PXE server. Additionally, a basic configuration for these packages must be completed before the initial boot menu can be constructed.

Static IP

In order for machines to leverage PXE to boot, the PXE server should be locked to a known static IP.

1. Edit the network interfaces file

   ```
   nano /etc/network/interfaces
   ```

2. Manually change the configuration for the ethernet adapter (edits in light green)

   ```
   # This file describes the network interfaces available on your system
   # and how to activate them. For more information, see interfaces(5).

   # The loopback network interface
   auto lo
   iface lo inet loopback

   # The primary network interface
   auto ensXX # Make sure to keep this interface name the same
   iface ensXX inet static
           address 10.1.1.XXX
           netmask 255.255.255.0
           network 10.1.1.0
           broadcast 10.1.1.255
           gateway 10.1.1.1
           dns-search bobbarker.com #Enter domain name suffixes
           dns-nameservers 10.1.1.10 #Enter the IP(s) of your nameservers
   ```

   A few quick notes on the above:

   - The name of the Ethernet adapter will be generated (by default in Ubuntu 15.10+). Make sure to maintain the same name when editing this file.

- The IP address, netmask (subnet), network, broadcast, and gateway are suggestions. These can be changed to coordinate with your existing network structure.
- While not required, it's good practice to populate both the **dns-search** and **dns-nameservers** lines if applicable. The lines above contain examples and should be replaced or removed to coordinate with your existing network structure.

tftpd-hpa

The tftp protocol is fundamental to how PXE sends boot information. This needs to be installed and configured.

1. Install the prerequisite packages

```
apt-get install -y apache2 tftpd-hpa
```

2. Edit the configuration file for the tftp server.

```
nano /etc/default/tftpd-hpa
```

3. Add the following content to the file. (new lines in light green)

```
# /etc/default/tftpd-hpa

TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/var/lib/tftpboot"
TFTP_ADDRESS="[::]:69"
TFTP_OPTIONS="--secure"
RUN_DAEMON="yes"
OPTIONS="-l -s /var/lib/tftpboot"
```

4. Restart the tftpd-hpa service

```
service tftpd-hpa restart
```

PXE Server Bootstrap Files

In order to get the server to boot network clients using PXE, a few files must be copied from the Ubuntu ISO to the tftpboot folder.

1. Download the Ubuntu 16.04 x64 Server ISO to the local drive

```
wget -P /media/
http://releases.ubuntu.com/16.04.2/ubuntu-16.04.2-server-amd
64.iso
```

2. Mount the ISO to a temporary folder. We will use this path as the source from which to copy the required pxelinux files.

```
mkdir /tmp/bootfiles
mount -t iso9660 -o loop
/media/ubuntu-16.04.2-server-amd64.iso /tmp/bootfiles
```

3. Copy the (5) pxelinux files from the mounted ISO to the tftpboot folder

```
cp /tmp/bootfiles/install/netboot/pxelinux.0
/var/lib/tftpboot/
cp /tmp/bootfiles/install/netboot/ldlinux.c32
/var/lib/tftpboot/
cp
/tmp/bootfiles/install/netboot/ubuntu-installer/amd64/boot-s
creens/vesamenu.c32 /var/lib/tftpboot/
cp
/tmp/bootfiles/install/netboot/ubuntu-installer/amd64/boot-s
creens/libcom32.c32 /var/lib/tftpboot/
cp
/tmp/bootfiles/install/netboot/ubuntu-installer/amd64/boot-s
creens/libutil.c32 /var/lib/tftpboot/
```

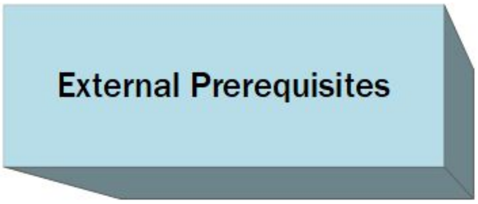
4. Check your work

```
ls /var/lib/tftpboot/
```

5. Remove the temporary mount point

```
umount /tmp/bootfiles
```

External Prerequisites



There is one requirement external to the PXE Server: DHCP changes. This is called out as two different efforts since the changes necessary often require a more advanced DHCP server than can be found in a basic network appliance. As such, a featureful DHCP server is required in order to correctly apply the DHCP scope options.

DHCP Server

Qualifying DHCP servers are those based on Linux, Windows, and more. Network appliances (commodity routers) with open source firmware can also be used. Commodity routers with factory firmware will oftentimes not be sufficiently featureful to accommodate the necessary changes required for setting DHCP scope options. While a detailed guide on installing a DHCP server is beyond the scope of this document, the following table should help guide users to the solution of their choice.

Platform for DHCP	**Does the DHCP Server need to be installed?**	**DHCP Scope Options**
Ubuntu	Install the DHCP service	Ubuntu DHCP Scope Options
Windows	DHCP on Server 2008 DHCP on Server 2008R2 DHCP on Server 2012 / R2 DHCP on Server 2016	Windows DHCP Scope Options
Qualifying Network Appliance	Yes	See Firmware specific configuration options
Basic Network Appliance	Yes, but it doesn't support PXE. 1. Install the DHCP service on the PXE Server itself 2. turn off DHCP on the Basic Network Appliance	Ubuntu DHCP Scope Options

DHCP Server Changes

Linux DHCP Scope Options

1. Edit the dhcp config file

   ```
   nano /etc/config/dhcpd.conf
   ```

2. Add the following changes to the end of the file:

   ```
   allow booting;
   allow bootp;
   option option-128 code 128 = string;
   option option-129 code 129 = text;
   next-server 10.1.1.XXX;
   filename "pxelinux.0";
   ```

   Where the **next-server** line specifies the IP Address of the PXE server

3. Save the file and exit nano
4. Restart the dhcp service

   ```
   service dhcpd restart
   ```

Windows DHCP Scope Options

DHCP Scope Option Name	Setting
066 Boot Server Host Name	IP Address of the PXE Server
067 Bootfile Name	pxelinux.0

What about DHCP scope option 060?

In some guides, there is a suggestion to enable DHCP scope option 060, which specifies the PXEClient option. This *only applies* if the DHCP server and the PXE server are on the *same* machine. In this case, the PXE service cannot exist on the same IP ports as the DCHP service. Setting option 060 tells the client to go look on port 4011 for the PXE Service.

Note that DHCP scope option 060 should therefore *not* be set where the DHCP server and PXE are on separate machines (as is the case with the examples above).

First menu

Now that the prerequisites for the PXE server are fulfilled, we can build a simple menu for the server.

Build a Basic Menu

1. First we need to make a config folder for our menu

   ```
   mkdir /var/lib/tftpboot/pxelinux.cfg
   ```

2. Create a new default menu file

   ```
   nano /var/lib/tftpboot/pxelinux.cfg/default
   ```

3. Populate this with the following lines

   ```
   # D-I config version 2.0
   # search path for the c32 support libraries (libcom32,
   libutil etc.)
   # path ubuntu-installer/amd64/boot-screens/
   # include ubuntu-installer/amd64/boot-screens/menu.cfg

   DEFAULT vesamenu.c32
   PROMPT 0
   TIMEOUT 300

   MENU TITLE DevOps Awesome PXE Boot Menu
   MENU AUTOBOOT Starting Local System in # seconds

   LABEL bootlocal
      MENU LABEL ^1) Boot to Local Drive
      MENU DEFAULT
      LOCALBOOT 0
   ```

   Let’s briefly cover this file structure:

   - **vesamenu.c32** is a menu type which provides basic graphical capability to the menu provided by the PXE server. Other menu options are also available. Both the **DEFAULT** and **PROMPT** lines should be used.

   - Providing a **TIMEOUT** is a good idea. Without a timeout, a machine that PXE boots will sit indefinitely waiting for user input. Note the number is counted in tenths of seconds, 300 being equal to 30 seconds.

- The **MENU TITLE** provides a banner for the menu
- The **MENU AUTOBOOT** line above provides a simple countdown timer for the user.
- The **LABEL bootlocal** helps delineate the first menu entry. Although not required, it's a good idea to indent the settings under a LABEL as shown above.
- The **MENU LABEL** line is the text presented to the user for the menu entry. The **^** character defines a hotkey for the menu, so the character after it should be unique within the scope of the menu. In this case we'll use a numbering scheme.
- **MENU DEFAULT** means that after our timeout expires, this menu option is chosen as the default option. Defaulting to boot the local OS is good practice.
- **LOCALBOOT 0** is the directive which will be applied if this menu option is chosen. In this case it will attempt to boot the local operating system.

4. Save the file and exit nano. A basic menu for the PXE server is now built (we'll add additional functionality later, following our layered approach)

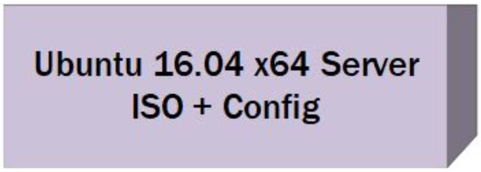
Tip: For a very in-depth look at the PXE menu, see the syslinux menu wiki page.

Testing the Basic Menu

Verification of our first menu can be completed using VMWare Workstation Player. We'll start with a blank VM and verify netboot functionality.

1. Start VMWare Player
2. Click the **Create a New Virtual Machine** button on the right.
3. Leave the radio button on **I will install the operating system later** and click **Next**.
4. On the **Guest Operating System** window, change the radio button to **Linux** and change the **Version dropdown** to **Ubuntu 64-bit**. Click **Next**.
5. On the **Name the Virtual Machine** window, change the values in the **Virtual Machine Name** field and the **Location** field to fit your needs. Click **Next**.
6. On the **Specify Disk Capacity** window, leave everything at the default and click **Next**.
7. On the **Name the Virtual Machine** window, click on **Customize Hardware**.
    a. In the Hardware list, select **Network Adapter**.
    b. On the right hand side, under **Network Connection** section, change the radio button to **Bridged**.
    c. Click the **Close** button to complete the modification of the Virtual Hardware.
    d. Click the **Finish** button to complete the creation of the Virtual Machine.
8. In the Virtual Machine list on the left hand side, double click on the new VM.
9. The VM should start and after a moment the PXE boot menu should be shown displaying only a single option to **boot locally**.

Ubuntu 16.04 x64 Server ISO + Config



All the manual steps to produce the foundation of a PXE Server have now been completed. We will now begin to automate much of the manual work from Part I, whereby a machine will boot and be automatically configured as an Ubuntu 16.04 x64 Server with integrated support for SaltStack. This automation will be integrated as a menu option on the PXE server.

Get and Mount the ISO image

Taking cues from the PXE Server Boot Files process above, we'll download and mount the Ubuntu ISO. Note that since we are never writing to the file system of the ISO file (we only read from the file system during OS installation), we can mount the ISO directly without needing to copy the contents to the mount point. This simplifies the mount process and minimizes the required space for file allocation.

1. If not already completed (since downloading of the Ubuntu ISO was performed above), download the Ubuntu 16.04 x64 Server ISO to the local drive

   ```
   wget -P /media/
   http://releases.ubuntu.com/16.04.2/ubuntu-16.04.2-server-amd
   64.iso
   ```

2. Create the mount point for the ISO file. Note that we intend to mount the ISO file inside the document root folder of apache. This will allow the entire filesystem of the ISO to be available via http.

   ```
   mkdir /var/www/html/ubuntu1604x64server
   ```

3. Add a mount line to the server's fstab file

   ```
   echo "/media/ubuntu-16.04.2-server-amd64.iso
   /var/www/html/ubuntu1604x64server       iso9660 loop    0 0"
   >> /etc/fstab
   ```

4. Mount all the points in the fstab file

   ```
   mount -a
   ```

5. Verify that the ISO is correctly mounted

   ```
   mount
   ```

SaltBase Bootstrap files

In order to boot the ISO, we first need to make the kernel and RAMdisk (which correspond to the ISO we are booting) available to the tftp server installed on PXE. These will be copied into a folder which will encapsulate this option for PXE.

1. Create the containing folder for this boot option

   ```
   mkdir /var/lib/tftpboot/ubuntu1604x64server
   ```

2. Copy the kernel and RAMdisk into this folder. We use http here a) to verify apache is functioning correctly and b) because http sources are easier to express in salt (we'll see the advantage of this later once we convert these manual steps to salt states).

   ```
   wget -P /var/lib/tftpboot/ubuntu1604x64server/ http://127.0.0.1/ubuntu1604x64server/install/netboot/ubuntu-installer/amd64/linux
   ```

   ```
   wget -P /var/lib/tftpboot/ubuntu1604x64server/ http://127.0.0.1/ubuntu1604x64server/install/netboot/ubuntu-installer/amd64/initrd.gz
   ```

Automated Ubuntu install using Preseed File

Built into Ubuntu (as a Debian derivative) is the ability to **preseed** the installation of the OS. This essentially provides a list of configuration choices usually made by the user during the standard install process. It's important to understand the capabilities as well as the limitations of the preseed approach. While preseed files can be specified at boot time or integrated into boot media, we will be integrating our pressed file directly into the PXE server. This preseed file, in conjunction with the OS media mount above, and the salt formula below, will serve to replace the steps required for the manual deployment of a SaltBase machine (as detailed in Part I) with the *automated* deployment of a SaltBase machine.

What can be completed in preseed?

- Setup of the root account, root password, and enable login for root
- Avoiding the default setup of a user other than root
- Setting of the timezone
- Setting of the partition scheme for the disc
- Installing some packages
- Set an update policy for the OS
- Hostname assignment
- Install SaltStack from external package sources
- Configure local file structure for Salt
- Dropping the SaltBase_Install formula onto the SaltBase machine
- Prepopulating the .bashrc with the final configuration steps using Salt

Add the saltbase_install folder

As the manual processes of Part I are replaced with automation, additional files are required as part of this transition. While these files could technically live in the root of the web server (/var/www/html), adding an additional folder to contain these files will keep this root folder uncluttered.

1. Create the containing folder for saltbase_install files

```
mkdir /var/www/html/saltbase_install
```

Apt source file

The apt source for the updated salt packages needs to be written to the PXE server. This file encapsulates the manual steps used in Part I for assigning this apt source.

1. Add the repo to the sources list (16.04)

```
echo deb http://repo.saltstack.com/apt/ubuntu/16.04/amd64/latest xenial main >> /var/www/html/saltbase_install/new1604saltstack.list
```

Build the Preseed File

The preseed file built for the PXE server is a modified version of the basic template offered by Ubuntu. The easiest way to detail the changes made is to take a diff between the original template and the **ub1604x64server.preseed** file, going over each change in detail.

Line	2
Original	`(NULL)`

ub1604x64server	`#### Found Here:`
	`https://help.ubuntu.com/lts/installation-guide/example-preseed.txt`

Details	Provided a trace to where the data was originally found

Line	126
Original	`#d-i passwd/root-login boolean false`
ub1604x64server	`d-i passwd/root-login boolean true`
Details	Uncommented to allow login from root upon first boot

Line	128
Original	`#d-i passwd/make-user boolean false`
ub1604x64server	`d-i passwd/make-user boolean false`
Details	Uncommented to avoid making a default user other than root

Line	131 - 132

Original	`#d-i passwd/root-password password r00tme`
	`#d-i passwd/root-password-again password r00tme`

ub1604x64server	`d-i passwd/root-password password INeedToBeChanged`
	`d-i passwd/root-password-again password INeedToBeChanged`

Details	Uncommented to automate the setting of the root password and changed it to 8 character minimum to avoid console notification of weak password

Line	163
Original	`d-i time/zone string US/Eastern`
ub1604x64server	`d-i time/zone string US/Pacific`
Details	Uncommented to automate setting of the timezone

Line	188
Original	`d-i partman-auto/method string lvm`
ub1604x64server	`d-i partman-auto/method string regular`
Details	Changed to move from using logical volume manager to a standard partition scheme.

Line	360
Original	`tasksel tasksel/first multiselect ubuntu-desktop`
ub1604x64server	`#tasksel tasksel/first multiselect ubuntu-desktop`
Details	Commented out since we only want to install server and not desktop.

Line	363
Original	`(NULL)`
ub1604x64server	`tasksel tasksel/first multiselect standard`
Details	Added a new line to install the standard packages.

Line	366
Original	`#d-i pkgsel/include string openssh-server build-essential`
ub1604x64server	`d-i pkgsel/include string openssh-server`
Details	Uncommented and changed the line to allow for adding of the openssh-server by default. Note that we will be adding the build tools using the salt formula (as part of the process that builds full gitfs support)

Line	377
Original	`#d-i pkgsel/update-policy select none`
ub1604x64server	`d-i pkgsel/update-policy select none`
Details	Uncommented, otherwise the installer will prompt for this setting.

Side Note: The remaining lines were added to the preseed file in order to:

a) minimally configure salt
b) run a salt highstate upon login to apply the remaining configuration
c) clean up afterward

Line	483 - 484
Original	(NULL)

ub1604x64server	`d-i preseed/late_command string \`
	`in-target sed -i 's|127.0.1.1.*|127.0.1.1        ub1604x64SvrSaltBase salt|g' /etc/hosts; \`

Details	Ensure the machine being built is self-mastered by adding the salt name to /etc/hosts

Line	485 - 487
Original	(NULL)

ub1604x64server	`in-target mkdir /srv; \`
	`in-target mkdir /srv/salt; \`
	`in-target mkdir /srv/pillar; \`

Details	Create the basic salt folder structure

Line	488 - 490
Original	`(NULL)`

ub1604x64server	`in-target wget -P /tmp/ https://repo.saltstack.com/apt/ubuntu/16.04/amd64/latest/SALTSTACK-GPG-KEY.pub; \`
	`in-target apt-key add /tmp/SALTSTACK-GPG-KEY.pub; \`
	`in-target wget -P /etc/apt/sources.list.d/ http://pxe.bobbarker.com/saltbase_install/new1604saltstack.list; \`

Details	The SaltStack packages in the Ubuntu repository are woefully out of date. Add the apt-key and the link to the latest salt packages. Note the new1604saltstack.list file contains the pointer to the SaltStack package repository as detailed in the Apt source changes section of Part I.

Line	491 - 492
Original	`(NULL)`

ub1604x64server	`in-target apt update; \`
	`in-target apt install -y salt-common salt-master salt-minion; \`

Details	Install the latest salt packages

Line	493 - 494
Original	`(NULL)`

ub1604x64server	`echo "ub1604x64SvrSaltBase" > /target/etc/hostname; \`
	`echo "ub1604x64SvrSaltBase" > /target/etc/salt/minion_id; \`

Details	Apply a new name to both the hostname and minion_id files

Line	495
Original	`(NULL)`
ub1604x64server	`in-target wget -P /etc/salt/master.d/ http://pxe.bobbarker.com/saltbase_install/saltbase_install.conf; \`
Details	Drop in the salt configuration for the installation process. The configuration file includes all the settings necessary for the initial setup of a self-mastered minion. See below for a comparison of the installation configuration file versus the running configuration file.

Line	496 - 497
Original	`(NULL)`

ub1604x64server

```
in-target wget -P /srv/salt/
http://pxe.bobbarker.com/saltbase_install/top.sls; \
in-target wget -P /srv/salt/
http://pxe.bobbarker.com/saltbase_install/saltbase_install.sls; \
```

Details	Drop in the top file, and salt formula. The top file points to the saltbase_install.sls, and the salt formula performs all the remaining configuration changes necessary to complete a SaltBase machine.

Line	498
Original	`(NULL)`

ub1604x64server

```
echo "# START salt blockreplace" >> /target/root/.bashrc; \
```

Details	Build the START delineator for the blockreplace salt state (everything between START and END will be removed if the salt highstate succeeds)

Line	499 - 505
Original	`(NULL)`

ub1604x64server

```
echo "echo" >> /target/root/.bashrc; \
echo "echo '***************************************'" >>
/target/root/.bashrc; \
echo "echo '*  SaltBase configuration completing  *'" >>
/target/root/.bashrc; \
echo "echo '*     This may take a few minutes     *'" >>
/target/root/.bashrc; \
echo "echo '* Machine will reboot upon completion *'" >>
/target/root/.bashrc; \
echo "echo '***************************************'" >>
/target/root/.bashrc; \
echo "echo" >> /target/root/.bashrc; \
```

Details	Add a notification to the root .bashrc so the user knows the final SaltBase configuration is taking place and manage expectations

Line	506
Original	`(NULL)`

ub1604x64server

```
echo "salt '*' state.highstate > /root/saltbase_install.log" >>
/target/root/.bashrc; \
```

Details	Automatically run a salt highstate upon root login

Line	507
Original	`(NULL)`
ub1604x64server	`echo "reboot" >> /target/root/.bashrc; \`
Details	Automatically reboot after the highstate is applied

Line	508
Original	`(NULL)`
ub1604x64server	`echo "# END salt blockreplace" >> /target/root/.bashrc`
Details	Build the END delineator for the blockreplace salt state (everything between START and END will be removed if the salt highstate succeeds)

Side Note: The simple way to set the root password uses plaintext input. A more secure way to accomplish this is by pre-hashing the password and using the root-password-crypted command. First, get the hash of the password using mkpasswd, then use the result as input for the root-password-crypted command. (mkpasswd requires the whois package be installed)

```
mkpasswd -H md5 "INeedToBeChanged"
```

Line	134
Original	#d-i passwd/root-password-crypted password [crypt(3) hash]
ub1604x64server	`d-i passwd/root-password-crypted password $1$ROnGTxWk$yaVNw1c07.K2y8VJfmFQF0`
Details	Uncommented to automate the setting of the root password using the pre-hashed value. Note: This is the hash of the default root password from above.

Drop the preseed file and dependent files into place

1. Download the preseed file directly from our github source

   ```
   wget -P /var/www/html/ https://raw.githubusercontent.com/love2scoot/pxesaltbase-formula/master/manualconfig/ub1604x64server.preseed
   ```

2. Download the saltstack package source link from our github source

   ```
   wget -P /var/www/html/saltbase_install https://raw.githubusercontent.com/love2scoot/pxesaltbase-formula/master/pxesaltbase/saltbase_install/new1604saltstack.list
   ```

3. Download the salt-master configuration files for the installation process directly from our github source

   ```
   wget -P /var/www/html/saltbase_install https://raw.githubusercontent.com/love2scoot/pxesaltbase-formula/master/pxesaltbase/saltbase_install/saltbase_install.conf
   ```

4. Download the salt top file directly from our github source

   ```
   wget -P /var/www/html/saltbase_install https://raw.githubusercontent.com/love2scoot/pxesaltbase-formula/master/pxesaltbase/saltbase_install/top.sls
   ```

5. Download the saltbase_isntall.sls salt formula directly from our github source

   ```
   wget -P /var/www/html/saltbase_install https://raw.githubusercontent.com/love2scoot/pxesaltbase-formula/master/manualconfig/saltbase_install.sls
   ```

Understanding the difference in salt-master configuration files

As detailed above, the salt-master configuration file, `saltbase_install.conf`, is installed as part of the preseed process. This configuration file differs from the salt-master configuration, `devops_standard.conf`, detailed in the Add config changes for gitfs section of Part I. Understanding how these two files differ helps give insight into the process of building an self-mastered SaltBase machine **using automation**.

`saltbase_install.conf`

```
#The root location for local salt source
file_roots:
  base:
    - /srv/salt

#The hash to use when discovering the hash of
a file on the master server
hash_type: sha256

#The root location for local pillar source
pillar_roots:
  base:
    - /srv/pillar

#Specify the search order of the backend file
systems
fileserver_backend:
  - roots

#Turn on auto accept mode during installation
auto_accept: True
```

`devops_standard.conf`

```
#The root location for local salt source
file_roots:
  base:
    - /srv/salt

#The hash to use when discovering the hash of
a file on the master server
hash_type: sha256

#The root location for local pillar source
pillar_roots:
  base:
    - /srv/pillar

#Specify the search order of the backend file
systems
fileserver_backend:
  - roots
  - git

#Specify which method provides gitfs access
to salt
gitfs_provider: pygit2
```

- With the compilation of libgit2 from source being performed in the salt formula (which happens *after* initial boot), the salt-master install config cannot include git as a backend nor call out a gitfs provider. These lines are therefore absent from the salt-master install config.
- Getting the salt-master to accept the salt-key of the salt-minion, **without user intervention**, is not a trivial process. In the Configure VM as Salf-Mastered section of Part I, the user completes this process manually using the salt-key command; however this command does not work well with a newly configured minion. The easiest way to automate this during installation is to temporarily enable auto_accept within the salt-master config. Upon first boot, the salt-master will accept the salt-key of the salt-minion and store this key going forward, *even after auto_accept is turned off*.
- Upon the first boot of the SaltBase machine, the SaltBase_Install formula will be applied. Part of this formula replaces this salt-master install config with the devops config, effectively locking down the salt-master configuration and opening up gitfs compatibility.

Complete Configuration Using Salt

While some of the configuration required for building a SaltBase machine is accomplished with the preseed file above, additional configuration is completed using a salt formula. The final lines of the preseed file configure SaltStack, download the salt formula, and pre-populate the .bashrc file with the commands required to complete the configuration of the SaltBase machine. The SaltBase_Install formula, in conjunction with the OS media mount and preseed file above, will complete the conversion of the manual steps in Part I to an automated build process of a SaltBase machine. This will result in a server which can auto-deploy a SaltBase machine.

What remains to be completed?

- Helper scripts written to /root on the SaltBase machine
- Build libgit2 from source and install pygit2
- Configure devops config file for Salt
- Build placeholder for gitfs based sources
- Cleanup: removing the notification and application of the salt highstate at login

Build the SaltBase_Install Formula

Although the SaltBase_Install Formula will be downloaded to the PXE Server directly from our github sources below, it's worth looking at the formula in detail. The content of the dependent files (those copied to the SaltBase machine using this salt formula) will be analyzed afterward.

```
# Drop in the helper scripts.
saltbase_helper1:
  file.managed:
    - name: /root/enable_ssh.sh
    - user: root
    - group: root
    - mode: 755
    - source:
http://pxe.bobbarker.com/saltbase_install/enable_ssh.sh
```

```
    - skip_verify: true
saltbase_helper2:
  file.managed:
    - name: /root/newsalthostname.sh
    - user: root
    - group: root
    - mode: 755
    - source:
http://pxe.bobbarker.com/saltbase_install/newsalthostname.sh
    - skip_verify: true
saltbase_helper3:
  file.managed:
    - name: /root/setupnetwork.sh
    - user: root
    - group: root
    - mode: 755
    - source:
http://pxe.bobbarker.com/saltbase_install/setupnetwork.sh
    - skip_verify: true
```

This section drops in the helper scripts into the correct location on the SaltBase machine.

```
# Build libgit2 from source to enable https access (No https access
using the main repo version)
build_prereq:
  pkg.installed:
    - refresh: True
    - pkgs:
      - build-essential
      - cmake
      - libssh2-1-dev
      - python-dev
      - python-pip
      - python-cffi
      - libssl-dev
      - libffi-dev
      - pkg-config
      - libcurl4-openssl-dev
      - libhttp-parser-dev
```

Install all the prerequisites enabling the build of libgit2 from source

```
build-libgit2:
  cmd.run:
    - name: |
        cd /tmp
        wget
https://github.com/libgit2/libgit2/archive/v0.25.0.tar.gz
        tar -zxf v0.25.0.tar.gz
        cd ./libgit2-0.25.0
        cmake .
        make
        make install
        ldconfig
    - cwd: /tmp
    - shell: /bin/bash
    - timeout: 300
    - unless: 'salt-call --versions-report | grep "libgit2:
0.25.0"'
  pip.installed:
    - name: pygit2==0.25.0
```

This section of the salt formula downloads and builds libgit2 from source. Notice the "unless" line which first checks to see if the correct version of libgit2 is installed before proceeding. The matching version of pygit2 is also installed in this section.

```
# Configure Salt on the local machine by removing the saltbase
install config, adding the devops saltbase config,
# adding the gitfs_remotes template, and cleaning up the .bashrc
file.

remove_install_config:
  file.absent:
    - name: /etc/salt/master.d/saltbase_install.conf

devops_config_install:
  file.managed:
    - name: /etc/salt/master.d/devops_standard.conf
    - user: root
    - group: root
    - mode: 644
    - source:
http://pxe.bobbarker.com/saltbase_install/devops_standard.conf
    - skip_verify: true

gitfs_remotes_template:
  file.managed:
    - name: /etc/salt/master.d/gitfs_remotes.conf
    - user: root
    - group: root
    - mode: 644
    - source:
http://pxe.bobbarker.com/saltbase_install/gitfs_remotes.conf
    - skip_verify: true
```

In this section the saltbase_install.conf is replaced by the devops_standard.conf configuration file and a gitfs_remotes.conf placeholder file is added. The devops_standard.conf and gitfs_remotes.conf files mirror the content found in the Add config changes for gitfs section of Part I.

```
cleanup_bashrc:
  file.blockreplace:
    - name: /root/.bashrc
    - marker_start: "# START salt blockreplace"
    - marker_end: "# END salt blockreplace"
    - content: '# Initial config completed'
    - backup: False
```

Leveraging the START and END markers added to .bashrc in the preseed file, this final state cleans things up by replacing the previous content with a completion comment. This essentially means, if the entire saltbase_install formula succeeds, the automatic call of the salt highstate in the .bashrc is removed and the next login will immediately proceed to a prompt.

Drop the saltbase_install.sls salt formula into place

1. Download the saltbase_install.sls salt formula directly from our github source

```
wget -P /var/www/html/
https://raw.githubusercontent.com/love2scoot/pxesaltbase-for
mula/master/manualconfig/saltbase_install.sls
```

Drop the helper scripts into place

1. Download the enable_ssh.sh script directly from our github source

```
wget -P /var/www/html/
https://raw.githubusercontent.com/love2scoot/pxesaltbase-formula/master/pxesaltbase/saltbase_install/enable_ssh.sh
```

2. Download the newsalthostname.sh script directly from our github source

```
wget -P /var/www/html/
https://raw.githubusercontent.com/love2scoot/pxesaltbase-formula/master/pxesaltbase/saltbase_install/newsalthostname.sh
```

3. Download the setupnetwork.sh script directly from our github source

```
wget -P /var/www/html/
https://raw.githubusercontent.com/love2scoot/pxesaltbase-formula/master/pxesaltbase/saltbase_install/setupnetwork.sh
```

Side Note: A detailed breakdown of the helper scripts can be found in Appendix A

Drop the salt config files into place

1. Download the devops_standard.conf configuration file directly from our github source

```
wget -P /var/www/html/
https://raw.githubusercontent.com/love2scoot/pxesaltbase-formula/master/pxesaltbase/saltbase_install/devops_standard.conf
```

2. Download the gitfs_remotes.conf configuration file directly from our github source

```
wget -P /var/www/html/
https://raw.githubusercontent.com/love2scoot/pxesaltbase-formula/master/pxesaltbase/saltbase_install/gitfs_remotes.conf
```

Side Note: The content of the devops_standard.conf file can be found in the Understanding the difference in Salt-Master configuration files in the preseed section above. The content of the gitfs_remtoes.conf file can be found in the Add config changes for gitfs section of Part I.

Add Ubuntu 16.04 x64 Server SaltBase to the PXE Boot Menu

PXE Menu Item

Now that the configuration of the Ubuntu 16.04 x64 Server SaltBase is completed, we will need to add a menu option on the PXE server for this configuration.

1. Edit the menu file

```
nano /var/lib/tftpboot/pxelinux.cfg/default
```

2. Add a section for the Ubuntu 16.04 x64 Server SaltBase install. (new lines in light green) (change the PXE URL to match the DNS name of the PXE server)

```
# D-I config version 2.0
# search path for the c32 support libraries (libcom32,
libutil etc.)
# path ubuntu-installer/amd64/boot-screens/
# include ubuntu-installer/amd64/boot-screens/menu.cfg

DEFAULT vesamenu.c32
PROMPT 0
TIMEOUT 300

MENU TITLE DevOps Awesome PXE Boot Menu
MENU AUTOBOOT Starting Local System in # seconds

LABEL bootlocal
   MENU LABEL ^1) Boot to Local Drive
   MENU DEFAULT
   LOCALBOOT 0

LABEL Ub1604x64Server
   MENU LABEL ^3) Ubuntu 16.04x64 Server SaltBase Install
   KERNEL ubuntu1604x64server/linux
   APPEND initrd=ubuntu1604x64server/initrd.gz
locale=en_US.UTF-8 keyboard-configuration/layoutcode=us
hostname=unassigned netcfg/choose_interface=auto
url=http://pxe.bobbarker.com/ub1604x64server.preseed
```

3. Save the file.

> For those of you who noticed that Ub1604x64Server is menu item 3)
> first: good attention to detail - second: *no spoilers!*

Testing the Installation of an Ubuntu 16.04 x64 Server SaltBase machine via PXE

Verification of our first menu can be completed using VMWare Workstation Player. We'll start with a blank VM and verify netboot functionality.

1. Start VMWare Player and click the **Create a New Virtual Machine** button on the right.
2. Leave the radio button on **I will install the operating system later** and click **Next**.
3. On the **Guest Operating System** window, change the radio button to **Linux** and change the **Version dropdown** to **Ubuntu 64-bit**. Click **Next**.
4. On the **Name the Virtual Machine** window, change the values in the **Virtual Machine Name** field and the **Location** field to fit your needs. Click **Next**.
5. On the **Specify Disk Capacity** window, leave everything at the default and click **Next**.
6. On the **Name the Virtual Machine** window, click on **Customize Hardware**.
    a. In the Hardware list, select **Network Adapter**.
    b. On the right hand side, under **Network Connection** section, change the radio button to **Bridged**.
    c. Click the **Close** button to complete the modification of the Virtual Hardware.
    d. Click the **Finish** button to complete the creation of the Virtual Machine.
7. In the Virtual Machine list on the left hand side, double click on the new VM.
8. The VM should start and, after a few moments, the PXE boot menu will appear.
9. Select the option to install the **Ubuntu 16.04 x64 Server SaltBase**.
10. Wait until OS installation is finished, and login using the root.
11. The notification of final configuration will appear and the salt highstate will complete the configuration of the machine. When completed, it will automatically reboot. Login again using root.
12. Run the command to check that the master has correctly added the minion key. The minion should be shown as accepted.

    ```
    salt-key -L
    ```

13. Verify the versions of the installed packages

    ```
    salt-master --version; salt-minion --version
    ```

14. Verify the status of the salt services

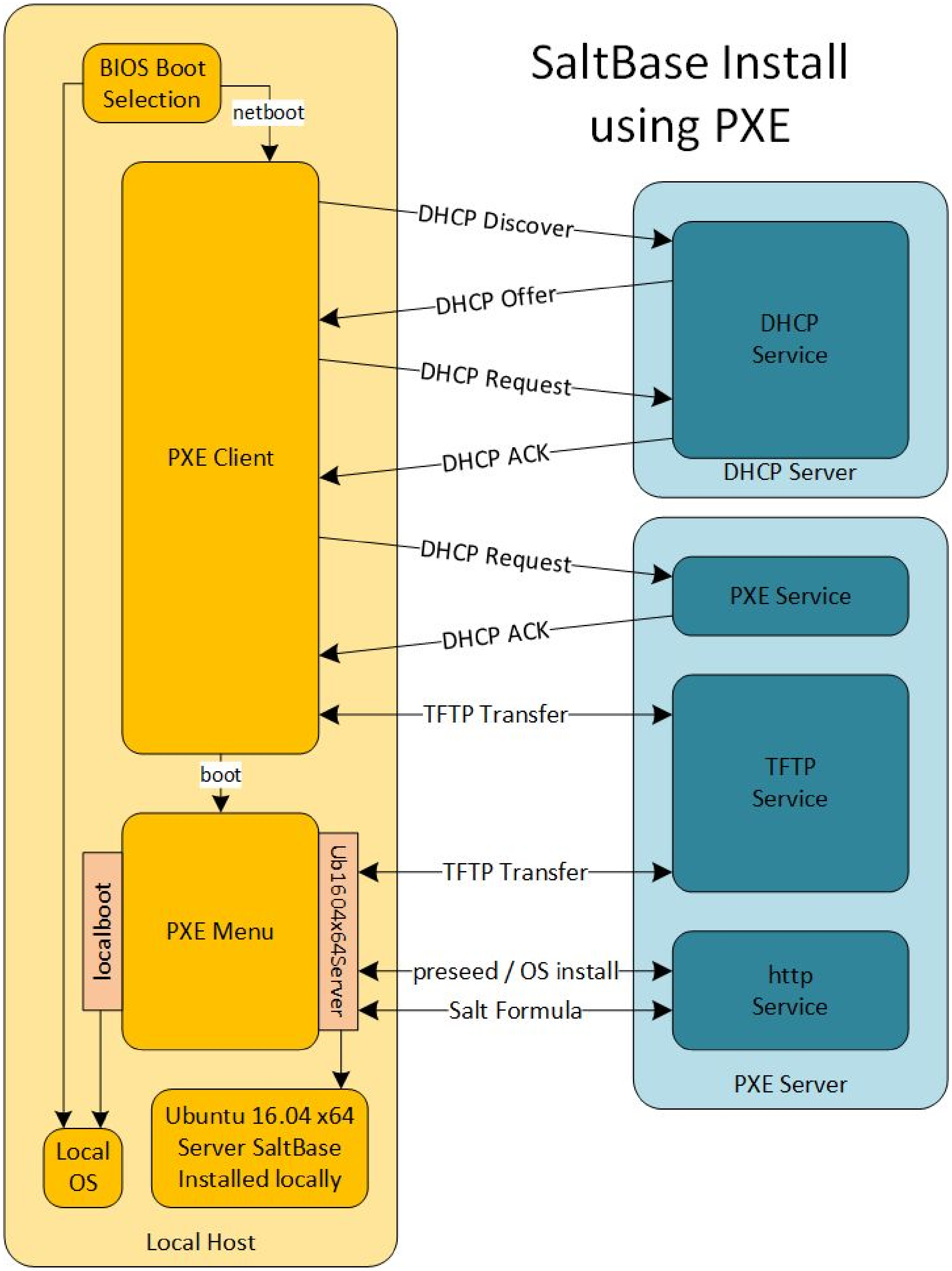
    ```
    service salt-master status
    service salt-minion status
    ```

15. Verify communication between the master and the minion using the command

    ```
    salt '*' test.ping
    ```

Understanding the SaltBase Install over PXE

Although the manual steps of the PXE server installation are outlined above, detailing how a machine would interact with the PXE server will help provide greater insight into the process as a whole. Below is a detailed diagram for installing a SaltBase machine via PXE. This is followed by a short key which adds detail to each of these steps.



Action	Source / Target	Description
BIOS Boot Selection	Local Host / N/A	The BIOS of the machine will have an option for booting from different sources. Shown here are booting to the Local OS as well as netbooting to PXE.
DHCP Discover	PXE Client / Broadcast	The PXE Client sends out a DHCP Discover broadcast to the network in an attempt to locate a local DHCP Server.
DHCP Offer	DHCP Server / PXE Client	The DHCP Server responds with a DHCP Offer of an available IP address and boot server parameters.
DHCP Request	PXE Client / DHCP Server	The PXE client receives the offer and responds with a DHCP Request for the IP address including an acknowledgement of the boot server
DHCP ACK	DHCP Server / PXE Client	The DHCP Server sends an acknowledgement of the request for the IP and issues a lease for the address.
DHCP Request	PXE Client / PXE Server	The PXE Client sends a DHCP request to the boot server (in this case the PXE Server) for the boot server IP and the bootstrap files.
DHCP ACK	PXE Server / PXE Client	The PXE Server sends an acknowledgement of the request with the boot server IP and bootstrap file
TFTP Transfer	PXE Client / PXE Server	The PXE Client downloads the bootstrap files from the PXE Server using the TFTP protocol. These will be used to boot into the PXE Boot Menu
Boot	N/A	Once downloaded, the PXE client boots using the bootstrap files into the PXE Menu..
PXE Menu	Local Host / N/A	The PXE Menu is hosted on the local machine. In this case there are options for local boot as well as the SaltBase install.
TFTP Transfer	Local Host / PXE Server	Once selected, the SaltBase option will copy bootstrap files from the PXE Server using the TFTP protocol. These will be used to boot into the target OS.
Preseed / OS Install	Local Host / PXE Server	After the bootstrap copy is complete, the local host will access the preseed file and the OS file system through the http service on the PXE server. See Detailing the SaltBase build over PXE below for a more detailed view of these steps.
Salt Formula	Local Host / PXE Server	Finally, the preseed file will chain load the Salt Formula from the http service on the PXE Server. This formula will complete the configuration of the SaltBase machine locally. See Detailing the SaltBase build over PXE below for a more detailed view of these steps.

Detailing the SaltBase Build over PXE

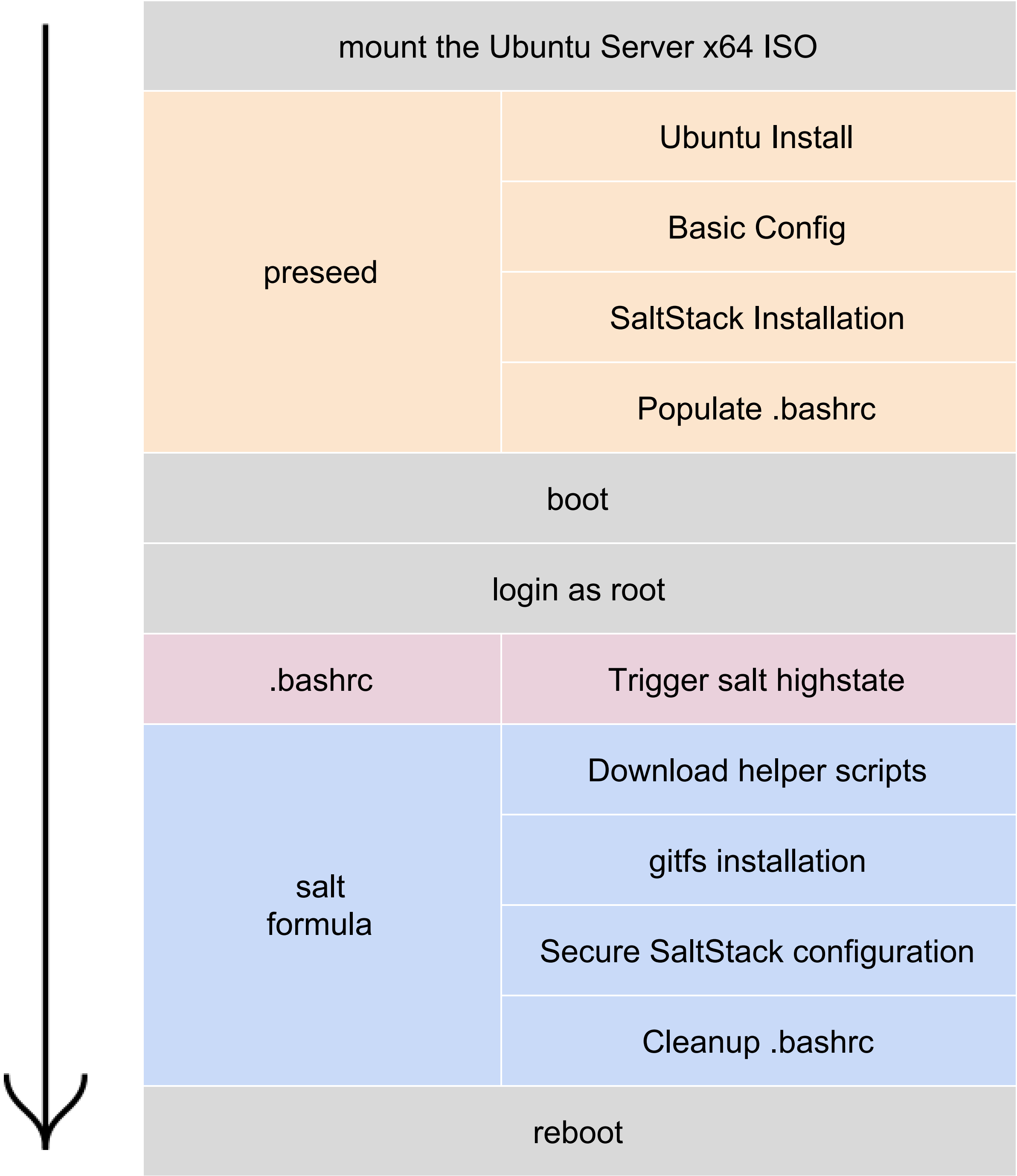
The table below tracks all the steps performed in Part I for the manual configuration of the SaltBase build process, as well as adding new steps created as part of the automation effort.

Manual Install Step	OS Media Mount	Preseed File	.bashrc	Salt Formula
Mounting the Ubuntu ISO for Installation	X			
Ubuntu Install: Keyboard Selection		X		
Ubuntu Install: Language Selection		X		
Ubuntu Install: Location / Timezone		X		
Ubuntu Install: Hostname		X		
Ubuntu Install: Config of temp user	No longer required			
Ubuntu Install: Partition Disk		X		
Ubuntu Install: Specify http proxy		X		
Ubuntu Install: Update Policy		X		
Ubuntu Install: Package Selection		X		
Ubuntu Install: Bootloader		X		
Basic Config: root password		X		
Basic Config: enable ssh for root	Included in helper scripts			
SaltStack: Add public repo key		X		
SaltStack: Add repo to sources list		X		

SaltStack: Install packages		X		
SaltStack: Update hosts file with salt entry		X		
SaltStack: Create the saltbase_install.conf		X		
SaltStack: Start the services	These start at first boot			
SaltStack: Accept minion keys on master	This is now automated			
SaltStack: Restart services	No longer required			
SaltStack: Create folder structure		X		
SaltStack: Add notification to .bashrc		X		
SaltStack: Add salt highstate to .bashrc		X		
.bashrc: Trigger salt highstate			X	
Helper Scripts: Download these				X
gitfs: install build tools				X
gitfs: download libgit2 source				X
gitfs: compile libgit2 source				X
gitfs: install pygit2				X
gitfs: Update devops_standard.conf	No longer required			
gitfs: Create gitfs_remotes.conf				X
gitfs: Restart services	No longer required			
SaltStack: Create the devops_standard.conf				X
New: Cleanup .bashrc				X

Graphing the SaltBase Build over PXE

The diagram below helps summarize the major steps required to complete the SaltBase installation over PXE



mount the Ubuntu Server x64 ISO	
preseed	Ubuntu Install
	Basic Config
	SaltStack Installation
	Populate .bashrc
boot	
login as root	
.bashrc	Trigger salt highstate
salt formula	Download helper scripts
	gitfs installation
	Secure SaltStack configuration
	Cleanup .bashrc
reboot	

Save Point: Export PXE Server to .ova

Creating a universal OVA template on Windows

Would you like to save here?

In order to create an OVA template that will be compatible with the maximum number of hypervisors, use the following guidelines with VMWare Workstation Player.

1. Ensure the VM version is no greater than 11. If it is greater, open the .vmx file in a text editor and change the following line:

```
virtualHW.version = "11"
```

2. Open the virtual machine in VMWare Workstation Player (do not start the VM), edit the VM settings, and remove the SATA CD-ROM device (if present). Close VMWare Workstation Player.
3. Browse to the folder containing the VM
4. Open the .vmx file using a text editor
5. Find and remove all lines that include SATA or IDE, then save the .vmx file. Here are some examples:

```
sata0.present = "TRUE"
sata0:1.present = "TRUE"
sata0:1.fileName = "C:\VMWare\ubuntu-16.04-server-amd64.iso"
sata0:1.deviceType = "cdrom-image"
```

Exporting the OVA Appliance on Windows

1. Find the path to your VM
2. Open a command prompt (elevated). **NOTE:** This does not work in PowerShell.
3. Change to the OVFTool folder to execute the ovftool command

```
cd "C:\Program Files (x86)\VMware\VMware Player\OVFTool"
```

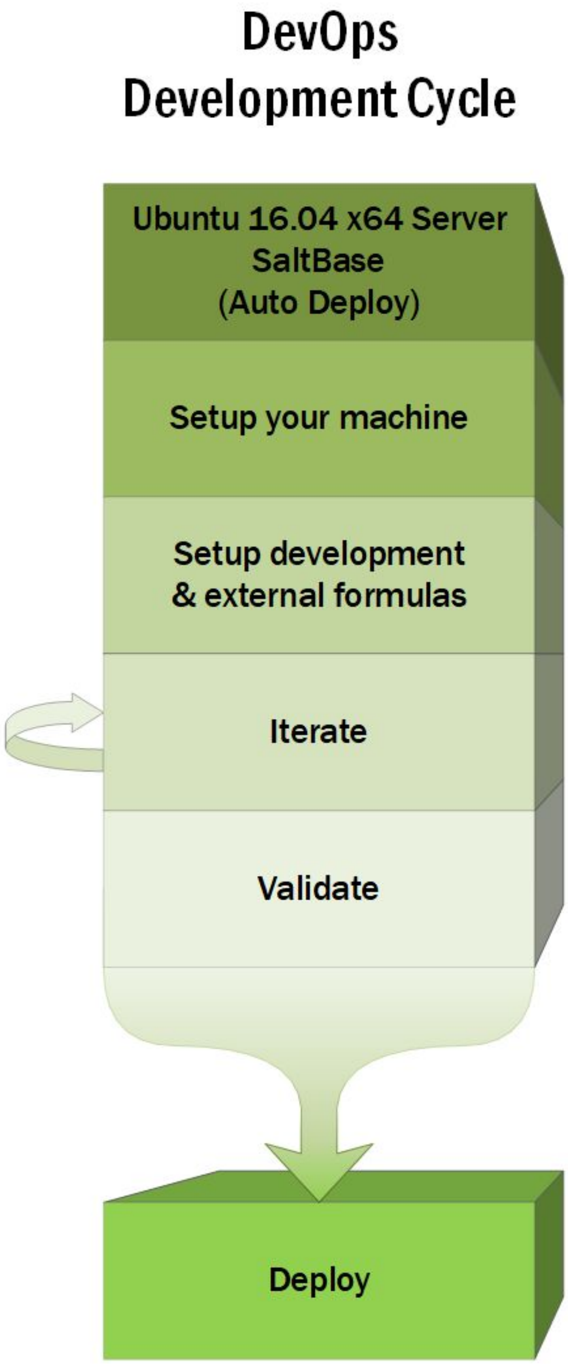
4. Convert your VM to an OVA Template

```
ovftool --compress=9 “C:\the path to your VM\your VMname.vmx”  "c:\new directory\name.ova"
```

The second finish line

The work above culminates in an infrastructure allowing any user to provision a SaltBase machine through selection from a simple menu. This automates the manual process outlined in Part I, ensuring a consistent and repeatable base on which to develop Salt. We can now consider how a **DevOps Development Cycle** could look within the context of a SaltBase machine, including local sources, remote gitfs sources, or a combination of both.

Part III: DevOps Development Cycle



Using the auto deployed SaltBase, the detailed **DevOps Development Cycle** will be documented. This development cycle will suggest best practices and articulate the benefits afforded by the Ubuntu +Salt + gitfs + PXE approach. This process can be broken down into:

- Auto deployment of an Ubuntu 16.04 x64 Server SaltBase VM using the PXE server
- Quick configuration of the basic settings for the VM
- Configuration of the development environment including both local and remote SaltStack sources.
- Iterative development using Salt
- Validation of the salt source by applying these states to a fresh VM
- Deployment of the application

Approach to Development

Leveraging the PXE server created in Part II, we can deploy to any machine that can netboot using PXE. That said, a development cycle is best served taking advantage of the benefits offered by virtualization- namely the ability to easily create multiple machines in rapid succession. As such, we will be deploying the SaltBase to a VM during the development cycle but will consider accommodations for other deployment options as part of this process.

Side Note: As a quick review, or for those who need a bit of context developing salt, this might be a good time to glance through the **Salt Development Backgrounder** in Appendix B.

Step 1: Auto Deploy a SaltBase Image

1. Start VMWare Player
2. Click the **Create a New Virtual Machine** button on the right.
3. Leave the radio button on **I will install the operating system later** and click **Next**.
4. On the **Guest Operating System** window, change the radio button to **Linux** and change the **Version dropdown** to **Ubuntu 64-bit**. Click **Next**.
5. On the **Name the Virtual Machine** window, change the values in the **Virtual Machine Name** field and the **Location** field to fit your needs. Click **Next**.
6. On the **Specify Disk Capacity** window, leave everything at the default and click **Next**.
7. On the **Name the Virtual Machine** window, click on **Customize Hardware**.
    a. In the Hardware list, select **Network Adapter**.
    b. On the right hand side, under **Network Connection** section, change the radio button to **Bridged**.
    c. Click the **Close** button to complete the modification of the Virtual Hardware.
    d. Click the **Finish** button to complete the creation of the Virtual Machine.
8. In the Virtual Machine list on the left hand side, double click on the new VM.
9. The VM should start and after a moment the PXE boot menu should be shown displaying two options, the first for **local boot**, and the second for installation of an **Ubuntu 16.04 x64 Server SaltBase**.
10. Select the option to install the **Ubuntu 16.04 x64 Server SaltBase**.
11. Wait until OS installation is finished, and login using the root.
12. The notification of final configuration will appear and the salt highstate will complete the configuration of the machine. When completed, it will automatically reboot. Login again using root.

Step 2: Setup the machine

Once the machine is imaged, it's time to login and customize. We will use the helper scripts to simplify this process and get the machine rapidly configured.

setupnetwork.sh

This script, located in /root, configures the eth0 (14.04) or enXXXX (16.04) adapter to either:

1. a static IP address
2. DHCP configuration

Once configured, the script restarts the Ethernet device in order to apply the configuration. **For Ubuntu v16.04 a reboot may be necessary to apply static IP address changes.**

Usage:

```
/root/setupnetwork.sh
```

enable_ssh.sh

By default in Ubuntu 14.04 and above, the root user is not allowed to connect via SSH by default. This script, located in /root, ensures openssh-server is installed, enables connections for the root user, and restarts the service.

Usage:

```
/root/enable_ssh.sh
```

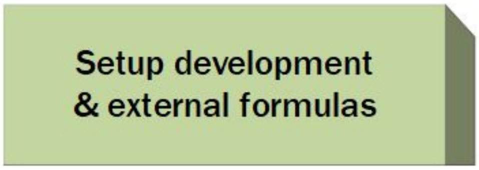
newsalthostname.sh

This script, located in /root, changes the hostname of the machine. This script simplifies this process by making the necessary changes to /etc/hostname, /etc/hosts, and /etc/salt/minion.id. Since these machines are self-mastered (both a Salt minion and a Salt master), this script also ensures that the security key associated between the two is updated as part of this process.

Usage:

```
/root/newsalthostname.sh
```

Step 3: Setup development and external formulas



While developers may have a process they prefer, we're going to start with a simple approach that leverages both code reuse and rapid development cycles. This section will also illustrate configuration of a development environment using salt formulas sourced from different gitfs repositories.

To help rapidly develop Salt, we need to be able to edit source and immediately test this Salt by applying it locally to the minion. Additionally, best practices should be followed where code is properly committed to a repository. The combination of these two requirements can be expressed using the workflow below.

Create an empty git repo

This will require access to a git server which can be either private or public. If you don't have access to a git server, several online services offer a free tier for open source projects, like GitHub. Once you have established access to a git server, create an empty repository, taking note of the clone URL. This repository will house the Salt source developed on this machine.

Setup a local development environment

We'll now create a local development folder which will be integrated with source control.

1. Although the SaltBase machine auto deploys with the /srv folder created, we will need to clone our git repository into an empty folder. As such, we'll move the contents of the /srv folder to a new location.

   ```
   mkdir /tmp/salttemp
   mv -R /srv/* /tmp/salttemp/
   ```

2. Next, the git repo created above should be cloned into this folder (replace the URL with the clone URL of the repo created above)

   ```
   git clone https://gitsource.bobbarker.com/happy.git /srv
   ```

3. The contents can now be moved back into place

   ```
   mv -R /tmp/salttemp/* /srv/
   ```

Build and Test

Now that the development environment is established, we can begin developing our first formula. We'll start with a single, simple state and test this formula to ensure everything is working as expected.

1. Create a new formula

   ```
   nano /srv/salt/packageupgrade.sls
   ```

2. Add the following content to the file and **save** the formula (note the spacing)

   ```
   # Refresh package manager
   update_os:
     pkg.uptodate:
     - refresh: true
   ```

3. Edit the **top.sls** file to add the name of the formula created above. This will direct Salt to use the new formula during a highstate.

   ```
   nano /srv/salt/top.sls
   ```

4. Add the following content to top.sls and **save** the file. (Note that we reference the *formula* name and not the *state* name) (new content in light green)

   ```
   base:
     '*':
       - packageupgrade
   ```

5. Restart the salt-master so that it picks up the new top.sls file. Note that the salt-master only needs to be restarted when new formulas are added to the top.sls file. This step is not required when simply making changes to formulas already called out by top.sls.

   ```
   service salt-master restart
   ```

6. Finally, we run a highstate to ensure that the formula is applied correctly

   ```
   salt '*' state.highstate
   ```

At this point, Salt should attempt to perform an "`apt update; apt upgrade`" and report back results. Assuming the results are positive, this validates the locally developed formula as well as the functionality of the salt-minion and salt-master.

Add an external formula

With the basic formula built and tested, we can now expand our Salt to include an external formula from a gitfs source. (This is similar to the process covered above in Apply a state from a gitfs source) For simplicity, this formula would have no dependencies and should not impact other software installed at a later time. In this case, we'll install fail2ban.

1. Open our gitfs_remotes.conf file for editing

   ```
   nano /etc/salt/master.d/gitfs_remotes.conf
   ```

2. Remove the # in front of gitfs_remotes:, add the following content, and **save** the file. (new content in light green)

   ```
   gitfs_remotes:
   #   - https://gitsource.bobbarker.com/happy.git:
   #     - user: for protected sources
   #     - password: for protected sources
   #     - root: salt
     - https://github.com/saltstack-formulas/fail2ban-formula.git
   ```

3. Edit the top file to include the new formula

   ```
   nano /srv/salt/top.sls
   ```

4. Add the following content to top.sls and **save** the file. (new content in light green)

   ```
   base:
     '*':
       - packageupgrade
       - fail2ban
   ```

5. Restart the Salt master in order to pick up these changes:

   ```
   service salt-master restart
   ```

6. Update the local cache of the remote sources (should return **TRUE**):

   ```
   salt-run fileserver.update
   ```

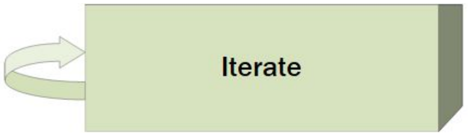
7. Finally, we run a highstate and ensure that the formula is applied correctly

   ```
   salt '*' state.highstate
   ```

At this point, Salt should attempt to apply both formulas to the salt-minion and report back results. Assuming the results are positive, this validates using formulas from both local sources and remote gitfs sources.

Step 4: Iterate

With the source now tested, active development can begin.



Build, Test, Add, Commit

The development cycle can essentially be expressed as

build	Edit the Salt source locally on the machine. *See the Build and Test section above.*
test	Validate the source by running a highstate and verifying the results. *See the Build and Test section above.*
add	In order to add new content to source control using git, the “git add” command should be used. For example, after a new formula is created within /srv/salt, it needs to be added to source control in order for changes to be tracked. While there are many different ways of running the “git add” command, the easiest is to simply add all files not already in source control to the next commit using the * wildcard.. Usage `git add *`
commit	The “git commit” command commits the staged snapshot to the project history. It is often easiest to use the “-m” switch, allowing the developer to directly add a commit message to the command itself. If “-m” is omitted, a text editor will be launched to record the commit message separately. Usage `git commit -m "<message>"`

Once the source is committed, the developer can return to the build step and resume developing the Salt. This process allows for rapid iteration on source code while ensuring revision history on the Salt in question.

Side Note: While developing, it can be easy to lose track of which files have been added, updated, and committed. In these cases, the helpful command `git status` will provide a detailed synopsis of the current file set within the context of git source control.

Usage

```
git status
```

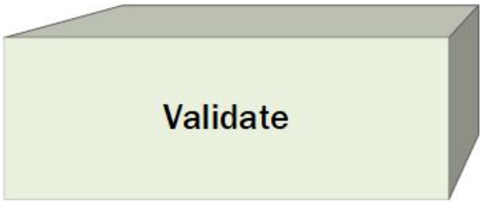
Push

Finally, once the Salt has reached a stable point, it's a good idea to update the git repository with all the commits made during the development cycle.

push	This will push all commits made to the source up to the git server. It is important to note that commits are *only stored on the local machine* until this command is run. Usage `git push origin master` *Origin* points to the repository from which the local copy of the repository was cloned. *Master* refers to the branch being pushed, in this case the master, or most recent branch. Note that this will also push all commits to the *Origin* which have been made since the last push command.

Side Note: The git workflow outlined above is kept purposefully lean. While this technically works for a single developer, multiple contributors will likely cause conflicts in the source. For a good understanding of managing more complex git workflows, see this great tutorial by Atlassian.

Step 5: Validate



Now that the source has been fully updated on the git server, we can validate the Salt using another machine.

Why Validate?

Developing using Salt is easy and flexible. States can be applied over and over until a machine reaches the desired configuration. While these advantages can streamline the development process, they don't protect against the developer manually configuring the machine *in tandem* with Salt. Validation of the Salt using a fresh machine ensures that the only configuration applied to the machine originates in the Salt source.

Auto Deploy

Follow the steps above in the Auto Deploy a SaltBase Image to setup a new VM.

Setup

Follow the steps above in the Setup the machine to configure the new VM.

Add Development Formula and Test

We now want to add the development formula pushed to the server in the previous step to the new VM. This will allow validation of the Salt by configuring the new VM exclusively from the Salt located on the git server

1. Open our gitfs_remotes.conf file for editing

   ```
   nano /etc/salt/master.d/gitfs_remotes.conf
   ```

2. Remove the # in front of gitfs_remotes:, add the following content, and **save** the file. (new content in light green) (replace the URL with the clone URL of the repo)

   ```
   gitfs_remotes:
   #   - https://gitsource.bobbarker.com/happy.git:
   #     - user: for protected sources
   #     - password: for protected sources
   #     - root: salt
     - https://gitsource.bobbarker.com/happy.git
   ```

3. Edit the top file to include the new formula

   ```
   nano /srv/salt/top.sls
   ```

4. Add all of the development formulas to top.sls and **save** the file. (new content in light green)

```
base:
  '*':
    - packageupgrade
    - getmcgavin
    - etc
```

5. Restart the Salt master in order to pick up these changes:

```
service salt-master restart
```

6. Update the local cache of the remote sources (should return **TRUE**):

```
salt-run fileserver.update
```

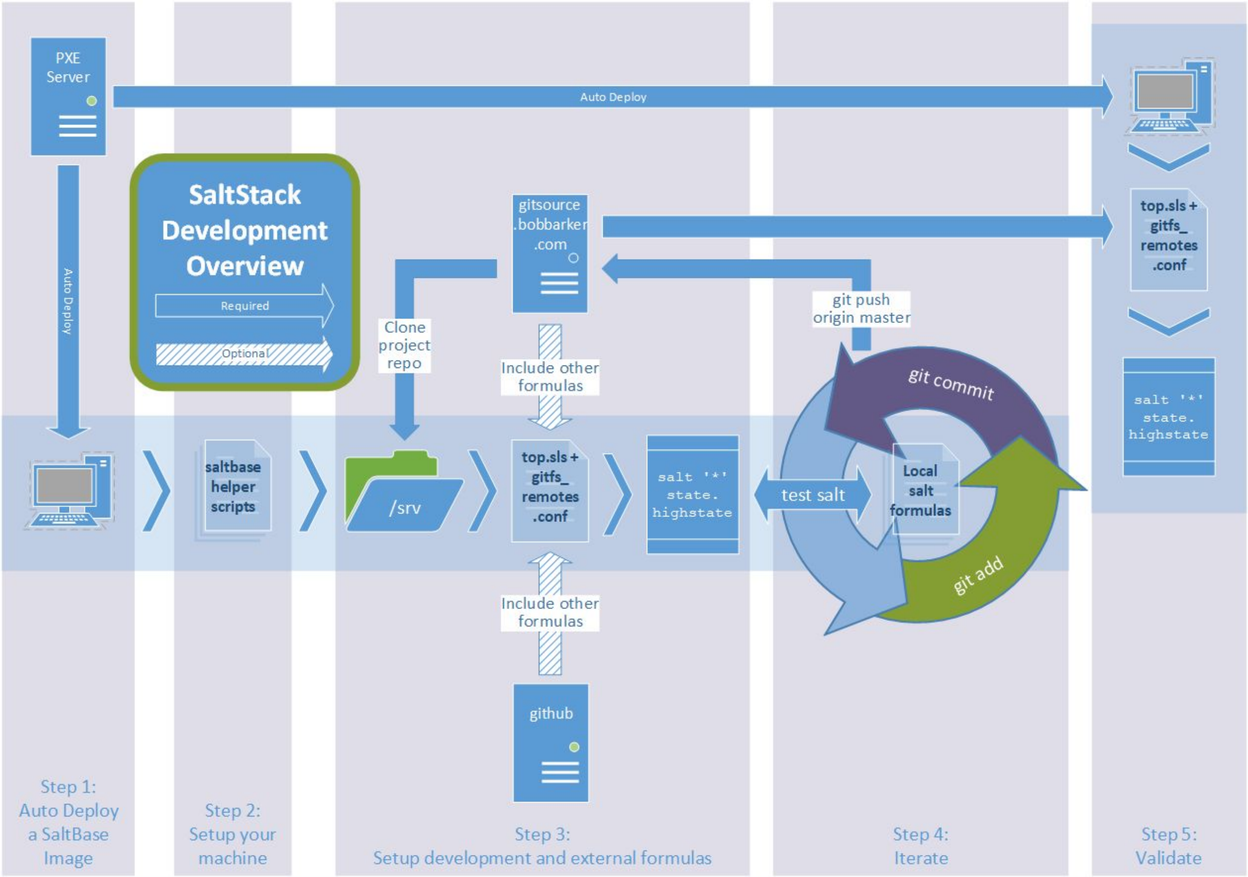
7. Finally, we run a highstate to complete the validation.

```
salt '*' state.highstate
```

At this point, Salt should attempt to apply all the remote formulas listed in top.sls to the fresh VM and return results. Assuming the results are positive, this validates the Salt and we can move onto the final step, **deploying to production**.

Overview

Before covering deployment scenarios for production, it is helpful to summarize the SaltBase development environment through use of a diagram.


PXE Server
Auto Deploy
Auto Deploy
SaltStack Development Overview
Required
Optional
gitsource .bobbarker .com
Clone project repo
Include other formulas
git push origin master
git commit
saltbase helper scripts
/srv
top.sls + gitfs_ remotes .conf
salt '*' state. highstate
test salt
Local salt formulas
git add
Include other formulas
github
top.sls + gitfs_ remotes .conf
salt '*' state. highstate
Step 1: Auto Deploy a SaltBase Image
Step 2: Setup your machine
Step 3: Setup development and external formulas
Step 4: Iterate
Step 5: Validate

Deployment Methods

With the Salt fully validated, the application can be deployed to production.

Deploying to a VM

While other deployments are possible, the most likely option is deployment to a VM.

The requirements are simple, the target hypervisor and VM only need to have netboot enabled for the virtual network adapter. Although each hypervisor may vary the approach for configuration, a good summary of the procedure can be found above in Step 1: Auto Deploy a SaltBase Image.

Deploying to Bare Metal

One of the advantages in using PXE for Auto deployment of a SaltBase machine, is the possibility to deploy to bare metal. With the availability of virtualization and containers, deployment to bare metal may seem a curious choice. In those situations where an engineer needs to directly connect a machine to a Device Under Test, a bare metal deployment can be quite useful.

The requirements for deploying to bare metal are quite simple: the hardware in question needs a network card capable of PXE boot. Almost all modern hardware will include this capability in the NIC, but enabling netboot may need to be performed in the BIOS before this feature is exposed.

Creating an .ova template

In cases where the target host for production deployment has strict network requirements (or limited network access), deploying first to an .ova and delivering this template to the target hardware is a possibility. This approach still has the advantage of the programmatic configuration of a machine through the use of Salt, which ensures consistency and stability for the deployment.

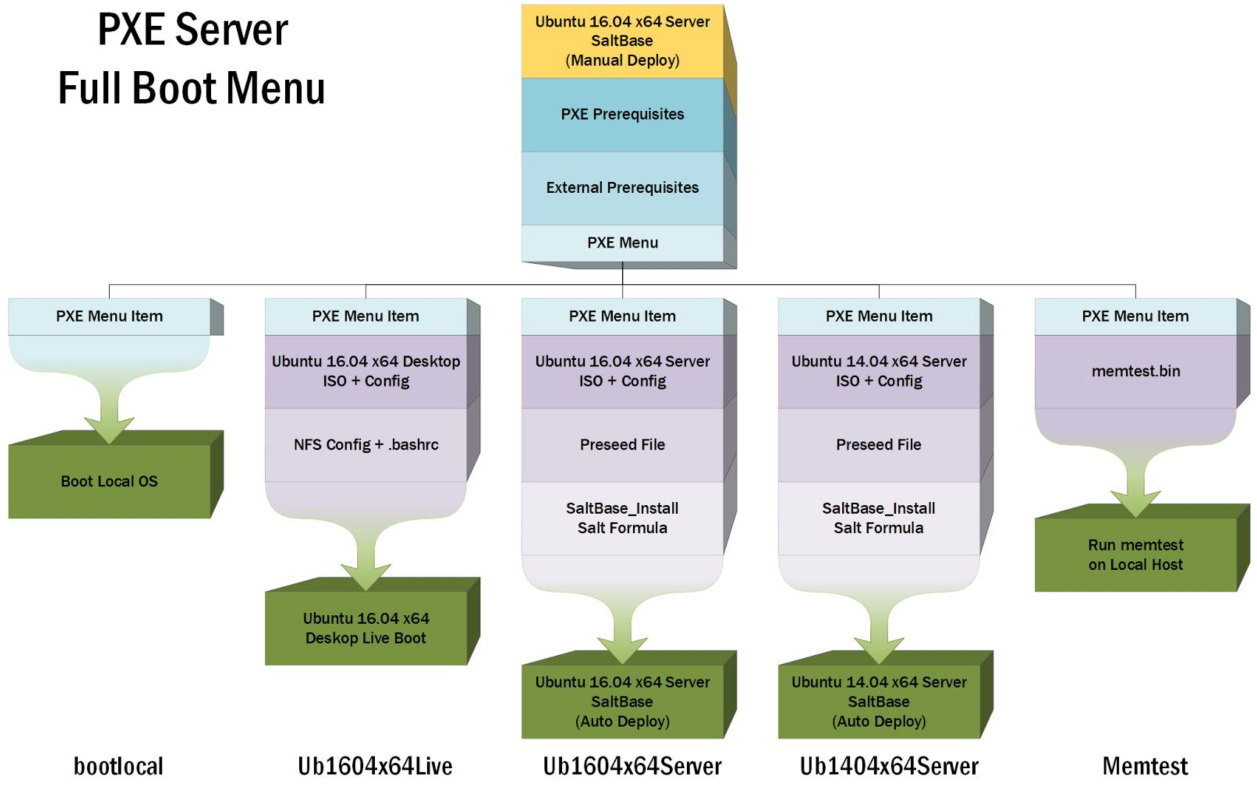
The procedure for deploying to an .ova has been covered above, both at the end of Part I as well as the end of Part II. These sections can be used as reference for the .ova template procedure.

PXE Feature Add

With the completed build of the Ub1604x64Server PXE boot option from Part II, we can use this effort as a template and extend the functionality of the PXE server by adding additional boot options. While a multitude of options exist, we have focused our approach on (3) new menu items which offer complimentary features and address a wide variety of needs within a DevOps environment. We will be adding options for:

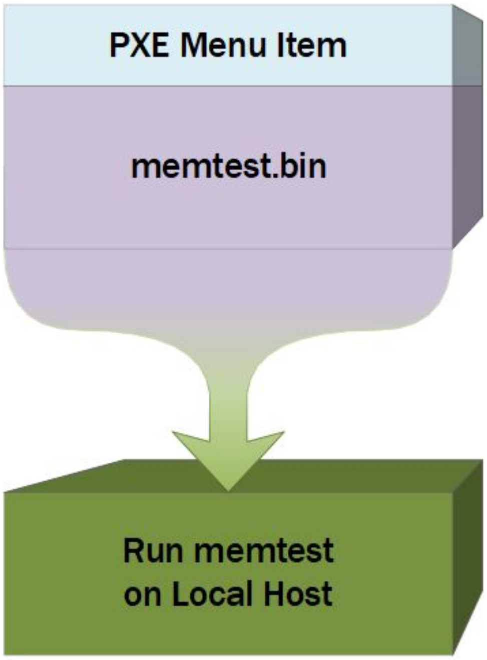
- A live boot for testing system memory (Memtest)
- A SaltBase build based on an older version of Ubuntu (Ub1404x64Server)
- A live boot of the most recent Ubuntu LTS desktop environment (Ub1604x64Live)

Looking at this approach from a high level, these features will be added to the existing PXE structure we completed in Part II. This means we need not cover any content on PXE configuration but can instead focus on the configuration of each new menu item. The following diagram summarizes the PXE boot Menu with the additional features.


PXE Server
Full Boot Menu
Ubuntu 16.04 x64 Server SaltBase (Manual Deploy)
PXE Prerequisites
External Prerequisites
PXE Menu
PXE Menu Item
Boot Local OS
bootlocal
PXE Menu Item
Ubuntu 16.04 x64 Desktop ISO + Config
NFS Config + .bashrc
Ubuntu 16.04 x64 Deskop Live Boot
Ub1604x64Live
PXE Menu Item
Ubuntu 16.04 x64 Server ISO + Config
Preseed File
SaltBase_Install Salt Formula
Ubuntu 16.04 x64 Server SaltBase (Auto Deploy)
Ub1604x64Server
PXE Menu Item
Ubuntu 14.04 x64 Server ISO + Config
Preseed File
SaltBase_Install Salt Formula
Ubuntu 14.04 x64 Server SaltBase (Auto Deploy)
Ub1404x64Server
PXE Menu Item
memtest.bin
Run memtest on Local Host
Memtest

Memtest

The simplest addition to our PXE boot menu is a live boot of memtest86+. Memtest86+ is a simple, open-source piece of software designed to detect and report problems with host memory. This software can be downloaded in several different configurations, one of which is a bootable binary kernel. Since this approach requires no mounting of an ISO, it requires only minimal effort to implement within PXE and is therefore the first additional feature we'll add to the PXESaltBase configuration.



Memtest

memtest.bin

1. In order to boot to the memtest86+ kernel, we simply need to download the bootable kernel to a known location on the PXE file system.

```
wget -P /var/lib/tftpboot/memtest/
http://www.memtest.org/download/5.01/memtest86+-5.01.bin
```

2. Next we simply rename the file to make it a neutral, non-versioned name.

```
mv /var/lib/tftpboot/memtest/memtest86+-5.01.bin
/var/lib/tftpboot/memtest/memtest
```

Add Memtest to the PXE Boot Menu

We need to add the memtest option to the PXE Boot Menu.

1. Edit the menu file

```
nano /var/lib/tftpboot/pxelinux.cfg/default
```

2. Add a section for the memtest boot option to the existing menu. (new lines in light green)

```
# D-I config version 2.0
# search path for the c32 support libraries (libcom32,
libutil etc.)
```

```
# path ubuntu-installer/amd64/boot-screens/
# include ubuntu-installer/amd64/boot-screens/menu.cfg

DEFAULT vesamenu.c32
PROMPT 0
TIMEOUT 300

MENU TITLE DevOps Awesome PXE Boot Menu
MENU AUTOBOOT Starting Local System in # seconds

LABEL bootlocal
   MENU LABEL ^1) Boot to Local Drive
   MENU DEFAULT
   LOCALBOOT 0

LABEL Ub1604x64Server
   MENU LABEL ^3) Ubuntu 16.04x64 Server SaltBase Install
   KERNEL ubuntu1604x64server/linux
   APPEND initrd=ubuntu1604x64server/initrd.gz
locale=en_US.UTF-8 keyboard-configuration/layoutcode=us
hostname=unassigned netcfg/choose_interface=auto
url=http://pxe.bobbarker.com/ub1604x64server.preseed

LABEL Memtest
   MENU LABEL ^5) Memtest
   ROOT (hd0,0)
   KERNEL memtest/memtest
```

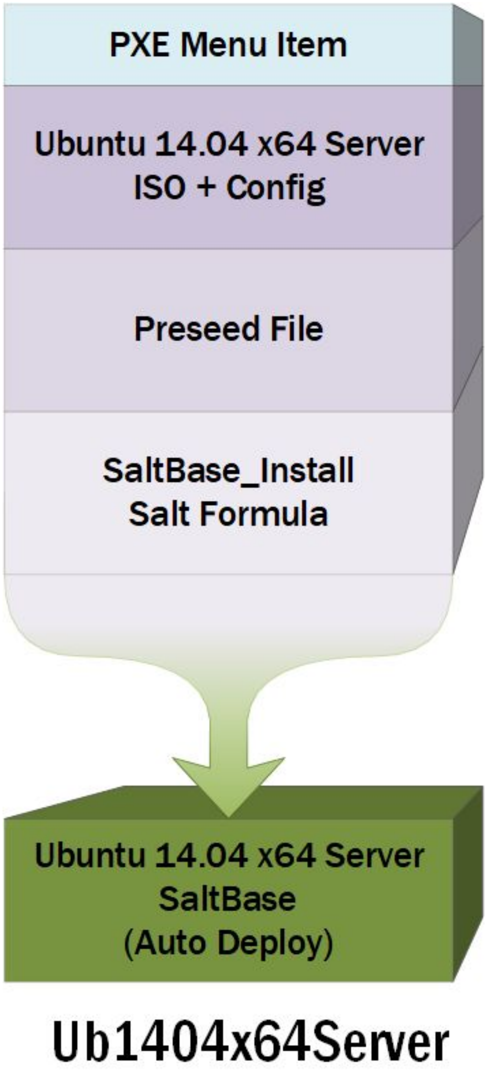
3. Save the file.

Test Memtest PXE Boot

1. On the network with the PXE server, boot any machine (it needs to be a physical machine and not a VM), making sure to use the network boot option.
2. After a few moments the PXE boot menu will appear.
3. Select the option to boot to **memtest**.
4. Let this program run, testing your system memory for errors.

Older Ubuntu Version

In some cases, the most recent LTS release of Ubuntu may not be the optimal choice. In this case, allowing installation of a SaltBase machine based on the previous LTS build is a good alternate solution. Installation of an Ubuntu 14.04 x64 Server SaltBase machine will include a large overlap with much of the content from Part II. In this case we need only call out those areas where the configuration of 14.04 is a departure from 16.04, while avoiding repetition of previous documentation.


PXE Menu Item
Ubuntu 14.04 x64 Server ISO + Config
Preseed File
SaltBase_Install Salt Formula
Ubuntu 14.04 x64 Server SaltBase (Auto Deploy)
Ub1404x64Server

Ubuntu 14.04 x64 Server ISO + Config

Get and Mount the ISO image

As explained above since we are never writing to the file system of the ISO file we can mount the ISO directly without needing to copy the contents to the mount point.

1. Download the Ubuntu 14.04 x64 Server ISO to the local drive

```
wget -P /media/ http://releases.ubuntu.com/14.04.5/ubuntu-14.04.5-server-amd64.iso
```

2. Create the mount point for the ISO file inside the root folder of apache.

```
mkdir /var/www/html/ubuntu1404x64server
```

3. Add a mount line to the server's fstab file

```
echo "/media/ubuntu-14.04.5-server-amd64.iso /var/www/html/ubuntu1404x64server       iso9660 loop    0 0" >> /etc/fstab
```

4. Mount all the points in the fstab file

```
mount -a
```

5. Verify that the ISO is correctly mounted

```
mount
```

SaltBase Bootstrap files

In order to boot the ISO, we first need to make the kernel and RAMdisk (which correspond to the ISO we are booting) available to the tftp server installed on PXE. These will be copied into a folder which will encapsulate this option for PXE.

1. Create the containing folder for this boot option

```
mkdir /var/lib/tftpboot/ubuntu1404x64server
```

2. Copy the kernel and RAMdisk into this folder. We use http here a) to verify apache is functioning correctly and b) because http sources are easier to express in salt (we'll see the advantage of this later once we convert these manual steps to salt states).

```
wget -P /var/lib/tftpboot/ubuntu1404x64server/
http://127.0.0.1/ubuntu1404x64server/install/netboot/ubuntu-
installer/amd64/linux
```

```
wget -P /var/lib/tftpboot/ubuntu1404x64server/
http://127.0.0.1/ubuntu1404x64server/install/netboot/ubuntu-
installer/amd64/initrd.gz
```

Apt source file

The apt source for the updated salt packages needs to be written to the PXE server.

1. Add the repo to the sources list (14.04)

```
echo deb
http://repo.saltstack.com/apt/ubuntu/14.04/amd64/latest trusty
main >> /var/www/html/saltbase_install/new1404saltstack.list
```

Automated Ubuntu install using Preseed File

The **preseed** file for Ubuntu 14.04 x64 Server is based off the same preseed template and with be functionally identical to that of the preseed file for Ubuntu 16.04 x64 Server. As such, we can distill our documentation down to only those areas where the 14.04x64 Server and the 16.04x64 Server preseed files diverge. (differences in light green)

Line	483 - 484
Original	`(NULL)`

ub1604x64server	`d-i preseed/late_command string \`
	`in-target sed -i 's|127.0.1.1.*|127.0.1.1        ub1404x64SvrSaltBase`
	`salt|g' /etc/hosts; \`

Details	Ensure the machine being built is self-mastered by adding the salt name to /etc/hosts

Line	488 - 490
Original	`(NULL)`

ub1604x64server	`in-target wget -P /tmp/`
	`https://repo.saltstack.com/apt/ubuntu/14.04/amd64/latest/SALTSTACK-GPG-K`
	`EY.pub; \`
	`in-target apt-key add /tmp/SALTSTACK-GPG-KEY.pub; \`
	`in-target wget -P /etc/apt/sources.list.d/`
	`http://pxe.bobbarker.com/saltbase_install/new1404saltstack.list; \`

Details	The SaltStack packages in the Ubuntu repository are woefully out of date. Add the apt-key and the link to the latest salt packages. Note the new1404saltstack.list file contains the pointer to the SaltStack package repository for Ubuntu 14.04 x64.

Line	493 - 494
Original	`(NULL)`

ub1604x64server	`echo "ub1404x64SvrSaltBase" > /target/etc/hostname; \`
	`echo "ub1404x64SvrSaltBase" > /target/etc/salt/minion_id; \`

Details	Apply a new name to both the hostname and minion_id files

Drop the Preseed File into place

1. Download the preseed file directly from our github source

```
wget -P /var/www/html/ https://raw.githubusercontent.com/love2scoot/pxesaltbase-formula/master/manualconfig/ub1404x64server.preseed
```

Complete Configuration Using the SaltBase_Install Formula

One of the many advantages to using salt is the obfuscation provided within the salt states. Whereas a script might differ slightly between Ubuntu 14.04 and Ubuntu 16.04, salt can allow for a single formula to perform the same functions across versions (or distributions). In this case, the salt formula for the Ubuntu 14.04x64 installation is identical to that of the Ubuntu 16.04x64 installation. As such, no additional files are required to be written to the host. The salt highstate will be initiated in the same way as with Ubuntu 16.04x64, through the .bashrc, completing the configuration of the SaltBase machine on 14.04x64.

Adding Ubuntu 14.04 x64 Server SaltBase to the PXE Boot Menu

We need to add the Ubuntu 14.04 x64 Server SaltBase option to the PXE Boot Menu.

1. Edit the menu file

```
nano /var/lib/tftpboot/pxelinux.cfg/default
```

2. Add a section for the memtest boot option to the existing menu. (new lines in light green)

```
# D-I config version 2.0
# search path for the c32 support libraries (libcom32,
libutil etc.)
# path ubuntu-installer/amd64/boot-screens/
# include ubuntu-installer/amd64/boot-screens/menu.cfg

DEFAULT vesamenu.c32
PROMPT 0
TIMEOUT 300

MENU TITLE DevOps Awesome PXE Boot Menu
MENU AUTOBOOT Starting Local System in # seconds

LABEL bootlocal
   MENU LABEL ^1) Boot to Local Drive
   MENU DEFAULT
   LOCALBOOT 0

LABEL Ub1604x64Server
   MENU LABEL ^3) Ubuntu 16.04x64 Server SaltBase Install
   KERNEL ubuntu1604x64server/linux
   APPEND initrd=ubuntu1604x64server/initrd.gz
locale=en_US.UTF-8 keyboard-configuration/layoutcode=us
```

```
hostname=unassigned netcfg/choose_interface=auto
url=http://pxe.bobbarker.com/ub1604x64server.preseed

LABEL Ub1404x64Server
   MENU LABEL ^4) Ubuntu 14.04x64 Server SaltBase Install
   KERNEL ubuntu1404x64server/linux
   APPEND initrd=ubuntu1404x64server/initrd.gz
locale=en_US.UTF-8 keyboard-configuration/layoutcode=us
hostname=unassigned netcfg/choose_interface=auto
url=http://pxe.bobbarker.com/ub1404x64server.preseed

LABEL Memtest
   MENU LABEL ^5) Memtest
   ROOT (hd0,0)
   KERNEL memtest/memtest
```

3. Save the file.

Testing the Installation of an Ubuntu 14.04 x64 Server SaltBase machine via PXE

Verification of our first menu can be completed using VMWare Workstation Player. We'll start with a blank VM and verify netboot functionality.

1. Start VMWare Player and click the **Create a New Virtual Machine** button on the right.
2. Leave the radio button on **I will install the operating system later** and click **Next**.
3. On the **Guest Operating System** window, change the radio button to **Linux** and change the **Version dropdown** to **Ubuntu 64-bit**. Click **Next**.
4. On the **Name the Virtual Machine** window, change the values in the **Virtual Machine Name** field and the **Location** field to fit your needs. Click **Next**.
5. On the **Specify Disk Capacity** window, leave everything at the default and click **Next**.
6. On the **Name the Virtual Machine** window, click on **Customize Hardware**.
    a. In the Hardware list, select **Network Adapter**.
    b. On the right hand side, under **Network Connection** section, change the radio button to **Bridged**.
    c. Click the **Close** button to complete the modification of the Virtual Hardware.
    d. Click the **Finish** button to complete the creation of the Virtual Machine.
7. In the Virtual Machine list on the left hand side, double click on the new VM.
8. The VM should start and after a few moments the PXE boot menu will appear.
9. Select the option to install the **Ubuntu 14.04 x64 Server SaltBase**.
10. Wait until OS installation is finished, and login using the root.
11. The notification of final configuration will appear and the salt highstate will complete the configuration of the machine. When completed, it will automatically reboot. Login again using root.
12. Run the command to check that the master has correctly added the minion key. The minion should be shown as accepted.

    ```
    salt-key -L
    ```

13. Verify the versions of the installed packages

    ```
    apt-show-versions salt-master salt-minion
    ```

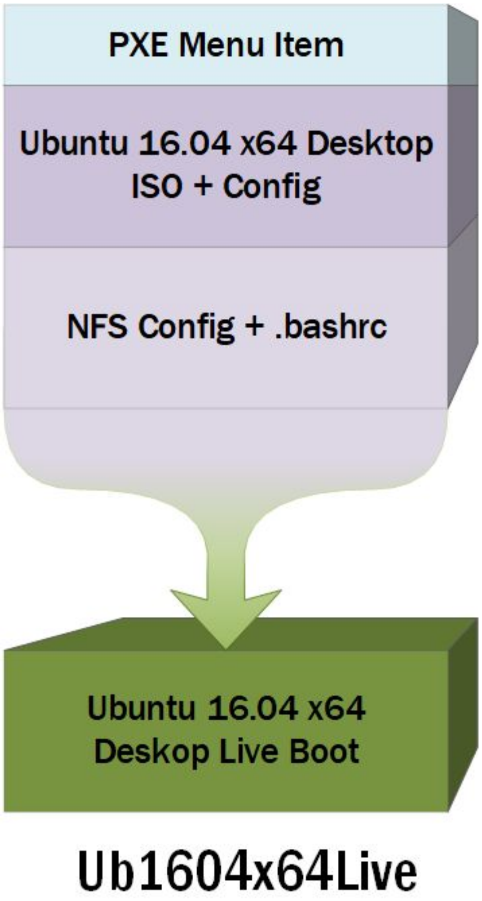
14. Verify the status of the salt services

    ```
    service salt-master status
    service salt-minion status
    ```

15. Verify communication between the master and the minion using the command

    ```
    salt '*' test.ping
    ```

Live Boot Ubuntu 16.04 x64 Desktop


PXE Menu Item
Ubuntu 16.04 x64 Desktop ISO + Config
NFS Config + .bashrc
Ubuntu 16.04 x64 Deskop Live Boot
Ub1604x64Live

One of the most useful additions to the PXE Server is inclusion of an Ubuntu Live Boot option. With live boot, a user can perform maintenance on an existing OS drive (like manipulating the partition table with gparted) along with many other helpful tasks. Configuration and use of this option through PXE requires some additional configuration beyond that with which the previous options have been built. As such, this section was left until last as it adds one final layer to our configuration.

Ubuntu 16.04 x64 Desktop ISO + Config

Get and Mount the ISO image

As explained above since we are never writing to the file system of the ISO file we can mount the ISO directly without needing to copy the contents to the mount point.

1. Download the Ubuntu 16.04 x64 Desktop ISO to the local drive

```
wget -P /media/ http://releases.ubuntu.com/16.04.2/ubuntu-16.04.2-desktop-amd64.iso
```

2. Create the mount point for the ISO file inside the root folder of apache.

```
mkdir /var/www/html/ubuntu1604x64live
```

3. Add a mount line to the server’s fstab file

```
echo "/media/ubuntu-16.04.2-desktop-amd64.iso /var/www/html/ubuntu1604x64live       iso9660 loop    0 0" >> /etc/fstab
```

4. Mount all the points in the fstab file

```
mount -a
```

5. Verify that the ISO is correctly mounted

```
mount
```

SaltBase Bootstrap files

In order to boot the ISO, we first need to make the kernel and RAMdisk (which correspond to the ISO we are booting) available to the tftp server installed on PXE. These will be copied into a folder which will encapsulate this option for PXE.

1. Create the containing folder for this boot option

   ```
   mkdir /var/lib/tftpboot/ubuntu1604x64live
   ```

2. Copy the kernel and RAMdisk into this folder. We use http here a) to verify apache is functioning correctly ans b) http sources are easier to express in salt.

   ```
   wget -P /var/lib/tftpboot/ubuntu1604x64live/ http://127.0.0.1/ubuntu1604x64live/install/netboot/ubuntu-installer/amd64/linux
   ```

   ```
   wget -P /var/lib/tftpboot/ubuntu1604x64live/ http://127.0.0.1/ubuntu1604x64server/install/netboot/ubuntu-installer/amd64/initrd.gz
   ```

NFS Config + .bashrc

Install and Configure nfs

In addition to having files mounted on tftp and http (apache), the live boot configuration also requires nfs. As such we'll need to install the nfs package and perform some basic configuration.

1. Install the nfs package

   ```
   apt install -y nfs-kernel-server
   ```

2. Add a line to /etc/exports for mounting the Ubuntu 16.04x64 Desktop ISO using nfs.

   ```
   echo "/var/www/html/ubuntu1604x64live/ *
   (ro,sync,no_wdelay,insecure_locks,no_root_squash,insecure)"
   >> /etc/exports
   ```

3. Verify the file contains the expected content

   ```
   tail /etc/exports
   ```

4. Reload the nfs service in order to pick up the new mount point

   ```
   service nfs-kernel-server restart
   ```

Fix the nfs mount in the PXE boot menu with .bashrc

When using nfs for mounting the live boot media, there is a limitation on the mount point within the PXE Boot Menu. BusyBox, which is leveraged as part of the live boot process, will *not* work with DNS names when looking for nfs mount points. As such, we need to replace the nfsroot line in the PXE menu with one that contains the *IP address* of the server (which is resolved as part of the replacement process). Dropping this into .bashrc of the PXE boot server ensures that each time the machine is rebooted, the IP of the system is resolved and the PXE boot menu is correctly updated. Also note that we are performing some logic to determine the name of the ethernet adapter. This code should be backward compatible for older versions of Ubuntu which use “eth0” for the default network adapter name.

1. Edit /root/.bashrc

```
nano /root/.bashrc
```

2. Add the following contents to the end of the file and save:

```
# The following commands fix the nfsroot call in defaults.
Busybox will not resolve the hostname so we are restricted
to using the IP only.
if ( grep eth0 /etc/network/interfaces )
then
     ethname="eth0" #Found eth0
else
     ethname=`ifconfig | grep -w -m 1 en.... | awk '{print
$1;}'` #Derived Ethernet interface name
fi
localpxeip=`/sbin/ifconfig $ethname | grep 'inet addr:' |
cut -d: -f2 | awk '{ print $1 }'`
sed -i "s|nfsroot.*:|nfsroot=$localpxeip:|g"
/var/lib/tftpboot/pxelinux.cfg/default
```

Add Ubuntu 16.04 x64 Live Boot to the PXE Boot Menu

We need to add the Ubuntu 16.04 x64 Desktop Live Boot option to the PXE Boot Menu.

1. Edit the menu file

```
nano /var/lib/tftpboot/pxelinux.cfg/default
```

2. Add a section for the memtest boot option to the existing menu. (new lines in light green)

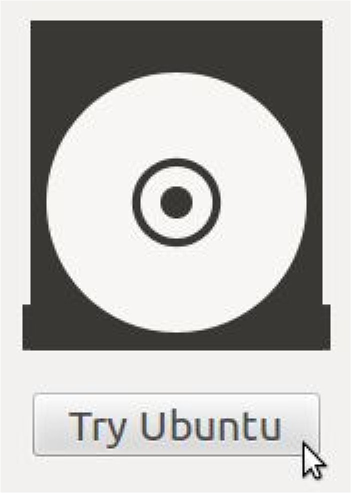
```
# D-I config version 2.0
# search path for the c32 support libraries (libcom32,
libutil etc.)
```

```
# path ubuntu-installer/amd64/boot-screens/
# include ubuntu-installer/amd64/boot-screens/menu.cfg

DEFAULT vesamenu.c32
PROMPT 0
TIMEOUT 300

MENU TITLE DevOps Awesome PXE Boot Menu
MENU AUTOBOOT Starting Local System in # seconds

LABEL bootlocal
   MENU LABEL ^1) Boot to Local Drive
   MENU DEFAULT
   LOCALBOOT 0

LABEL Ub1604x64Live
   MENU LABEL ^2) Ubuntu 16.04x64 Desktop Live Boot
   KERNEL ubuntu1604x64live/vmlinuz.efi
   APPEND vga=normal boot=casper rootfstype=nfs netboot=nfs
nfsroot=pxe.bobbarker.com:/var/www/html/ubuntu1604x64live/
initrd=ubuntu1604x64live/initrd.lz splash --

LABEL Ub1604x64Server
   MENU LABEL ^3) Ubuntu 16.04x64 Server SaltBase Install
   KERNEL ubuntu1604x64server/linux
   APPEND initrd=ubuntu1604x64server/initrd.gz
locale=en_US.UTF-8 keyboard-configuration/layoutcode=us
hostname=unassigned netcfg/choose_interface=auto
url=http://pxe.bobbarker.com/ub1604x64server.preseed

LABEL Ub1404x64Server
   MENU LABEL ^4) Ubuntu 14.04x64 Server SaltBase Install
   KERNEL ubuntu1404x64server/linux
   APPEND initrd=ubuntu1404x64server/initrd.gz
locale=en_US.UTF-8 keyboard-configuration/layoutcode=us
hostname=unassigned netcfg/choose_interface=auto
url=http://pxe.bobbarker.com/ub1404x64server.preseed

LABEL Memtest
   MENU LABEL ^5) Memtest
   ROOT (hd0,0)
   KERNEL memtest/memtest
```

4. Save the file.
5. Logoff / Login to update the PXE menu using .bashrc

Test Live Boot of Ubuntu 16.04 x64 using PXE

1. On the network with the PXE server, boot any machine, making sure to use the network boot option.
2. After a few moments the PXE boot menu will appear.
3. Select the option to boot to **Ubuntu 16.04x64 Desktop Live Boot**.
4. If prompted, choose **Try Ubuntu** to boot into the Live OS.



Dogfooding: SaltStack build of PXE

What better way to fully test the procedures above, than to build a *new* PXESaltBase machine using this very infrastructure? In this section we’ll migrate from our manual build to an automated build using salt, step through the required changes in the pillar, walk through the build process, and verify the deployment through testing.

The SaltBase PXE Server File Tree

A good way to start conceptualizing the automated build of the SaltBase PXE server, is to list all the files required and consider a) what can be brought over directly from manual processes b) what can be brought over but variablized using jinja and c) what needs to be created new for automation. In addition to color coding based on these categories, the files below contain links to the creation process for each file in this document.

<table>
<tr><th>File Tree</th><th>Key</th></tr>
<tr><td rowspan="3">pxesaltbase
│ .bashrc
│ default
│ exports
│ init.sls
│ tftpd-hpa
│ ub1404x64server.preseed
│ ub1604x64server.preseed
│
└──saltbase_install
devops_standard.conf
enable_ssh.sh
gitfs_remotes.conf
new1404saltstack.list
new1604saltstack.list
newsalthostname.sh
saltbase_install.conf
saltbase_install.sls
setupnetwork.sh
top.sls</td><td>🟦 Carried over from manual processes</td></tr>
<tr><td>🟩 Variablized using jinja</td></tr>
<tr><td>🟧 New for Automation</td></tr>
</table>

Migrating PXE from manual to auto deployment

Using the entirety of the manual steps within this document, we will migrate all build processes to salt, enabling a PXE server which can be automatically deployed directly from gitfs. The following sections will detail the files added or changed as part of this automation process (orange and green highlights above)

Salt for PXE server (init.sls)

The easiest way to map the conversion of this process from manual steps to salt is to simply study each section of the salt formula and ensure that all steps in the manual process are accounted for. The salt for our pxesaltbase formula is located here: `https://raw.githubusercontent.com/love2scoot/pxesaltbase-formula/master/pxesaltbase/init.sls`.

Salt: PXE Package Install

```
# Install the packages required for PXE
pxe_packages:
  pkg.installed:
    - refresh: True
    - pkgs:
      - apache2
      - tftpd-hpa
      - nfs-kernel-server
```

- `pkg.installed` simply installs packages to the OS using the built in package manager.
- `refresh: True` refreshes the available package list before performing the installation.
- Apache 2 is required to allow for mounting of ISOs, pressed files, salt formulas, and other files
- tftpd-hpa is required for the basic functionality of any PXE server. See Part II manual install
- nfs-kernel-server is required for ISO mounting for the live boot option

Salt: Configure the TFTP Server

```
# Configure the TFTPD-hpa service for serving the menu and kernels
tftpd_config:
  file.managed:
    - name: /etc/default/tftpd-hpa
    - user: root
    - group: root
    - mode: 644
    - source: salt://pxesaltbase/tftpd-hpa
  service.running:
    - name: tftpd-hpa
    - reload: true
    - watch:
      - file: /etc/default/tftpd-hpa
```

- `file.managed` pulls a file from source, writes it to the minion (if it is different from the local copy), and applies ownership and permission changes. In this case the configuration of the tftpd-hpa server is ready to be dropped into place. Looking at the source line, we see that it uses a `salt://pxesaltbase` URL, which means the file is stored in the same folder as the salt formula.

- `service.running` watches the tftpd-hpa config file and, if salt changes it, it will restart the tftpd-hpa service. Since the above `file.managed` state will only apply changes to the file the first time the salt is applied, the tftpd-hpa service only needs to be restarted on this trigger event.

Salt: Ubuntu 16.04 Live Boot

```
# Setting up ubuntu-16.04-desktop-amd64 live as a boot option
# Grab the ISO from the Ubuntu site
get_ub1604x64desktop_iso:
  file.managed:
    - name: /media/ubuntu-16.04.2-desktop-amd64.iso
    - user: root
    - group: root
    - mode: 644
    - source:
http://releases.ubuntu.com/16.04.2/ubuntu-16.04.2-desktop-amd64.iso
    - source_hash:
sha256=0f3086aa44edd38531898b32ee3318540af9c643c27346340deb2f9bc1c3
de7e
# Mount the ISO using the loop option.  Since this is supposed to
be read only media, we don't need to copy files locally.
mount_ub1604x64desktop_iso:
  file.directory:
    - name: /var/www/html/ubuntu1604x64live
  mount.mounted:
    - name: /var/www/html/ubuntu1604x64live
    - device: /media/ubuntu-16.04.2-desktop-amd64.iso
    - fstype: iso9660
    - opts:
      - loop
# After the ISO is mounted, the kernel and ramdisk need to be
copied from apache to the local TFTP Server to enable this image to
PXE boot
ub1604x64desktop_kernel:
  file.managed:
    - name: /var/lib/tftpboot/ubuntu1604x64live/vmlinuz.efi
    - user: root
    - group: root
    - mode: 644
    - source: http://127.0.0.1/ubuntu1604x64live/casper/vmlinuz.efi
    - makedirs: true
    - skip_verify: true
ub1604x64desktop_ramdisk:
  file.managed:
    - name: /var/lib/tftpboot/ubuntu1604x64live/initrd.lz
    - user: root
    - group: root
    - mode: 644
    - source: http://127.0.0.1/ubuntu1604x64live/casper/initrd.lz
    - skip_verify: true
```

- Since the live boot option is the first selection on the PXE boot menu, these states come next in the salt file
- `file.managed` pulls the desktop ISO from the Ubuntu website. With sources which do *not* use a `salt://` URL, verification steps have been added to ensure the file is downloaded correctly. This can be a verification using a checksum or (less desirable) the verification can simply be skipped. In this case we have supplied the sha256 checksum of the ISO file.
- Next we ensure the folder for mounting the ISO is created using `file.directory`.
- Using the state `mount.mounted`, the salt ensures that the ISO is mounted using the correct mount options. These steps replace the manual configuration of the live boot ISO image download and mount.
- Next, the kernel and ramdisk files need to be copied to tftpboot folder. Since the Ubuntu 16.04x64 Desktop ISO is already mounted, these are pulled from the mount point on the local apache server and *not* from our salt source. In the case where the downloaded ISO

is updated, this allows us to easily grab updated kernel and ramdisk files from the mounted source, avoiding the requirement to download these files to the local salt folder each time the ISO changes. Notice here that we choose to skip the verification of the files since they are sourced from localhost (127.0.0.1). It's also important to note that the kernel and ramdisk files for Ubuntu desktop are both named differently and are located within a different path than these same files sourced from their Ubuntu server counterparts. These two states replace the manual steps for downloading the kernel and ramdisk files.

Salt: Ubuntu 16.04 Server SaltBase

```
# Setting up ubuntu-16.04-server-amd64 install as a boot option
# Grab the ISO from the Ubuntu site
get_ub1604x64server_iso:
  file.managed:
    - name: /media/ubuntu-16.04.2-server-amd64.iso
    - user: root
    - group: root
    - mode: 644
    - source:
http://releases.ubuntu.com/16.04.2/ubuntu-16.04.2-server-amd64.iso
    - source_hash:
sha256=737ae7041212c628de5751d15c3016058b0e833fdc32e7420209b76ca3d0
a535
# Mount the ISO using the loop option.  Since this is supposed to
be read only media, we don't need to copy files locally.
mount_ub1604x64server_iso:
  file.directory:
    - name: /var/www/html/ubuntu1604x64server
  mount.mounted:
    - name: /var/www/html/ubuntu1604x64server
    - device: /media/ubuntu-16.04.2-server-amd64.iso
    - fstype: iso9660
    - opts:
      - loop
# After the ISO is mounted, the kernel and ramdisk need to be
copied from apache to the local TFTP Server to enable this image to
PXE boot
ub1604x64server_kernel:
  file.managed:
    - name: /var/lib/tftpboot/ubuntu1604x64server/linux
    - user: root
    - group: root
    - mode: 644
    - source:
http://127.0.0.1/ubuntu1604x64server/install/netboot/ubuntu-install
er/amd64/linux
    - makedirs: true
    - skip_verify: true
ub1604x64server_ramdisk:
  file.managed:
    - name: /var/lib/tftpboot/ubuntu1604x64server/initrd.gz
    - user: root
    - group: root
    - mode: 644
    - source:
http://127.0.0.1/ubuntu1604x64server/install/netboot/ubuntu-install
er/amd64/initrd.gz
    - skip_verify: true
# The preseed file specifies all the values necessary in order to
fully automate the OS install
ub1604x64server_preseed:
  file.managed:
    - name: /var/www/html/ub1604x64server.preseed
    - user: root
    - group: root
    - mode: 644
    - source: salt://pxesaltbase/ub1604x64server.preseed
    - template: jinja
```

- `file.managed` pulls the 16.04x64 Server ISO from the Ubuntu website and verifies it using the sha256 hash.
- In much the same way as above, the mount folder is created (using `file.directory`) and the ISO is mounted (using `mount.mounted`).
- Also just as above, both the kernel and ramdisk are pulled from the mounted media using `file.managed`. As noted above, the location and name of the kernel and ramdisk files differ between Ubuntu Desktop and Ubuntu Server.
- Finally, we pull in the last file. ub1604x64server.preseed. What's different here vs the other states which use `file.managed`, is the inclusion of the `template: jinja` line. This means that sections of the file include some jinja, which inserts pillar content at the time the file is written to disk. This is more fully detailed in the preseed files section below.

Salt: Ubuntu 14.04 Server SaltBase

```
# Setting up ubuntu-14.04.5-server-amd64 install as a boot option
# Grab the ISO from the Ubuntu site
get_ub1404x64server_iso:
  file.managed:
    - name: /media/ubuntu-14.04.5-server-amd64.iso
    - user: root
    - group: root
    - mode: 644
    - source:
http://releases.ubuntu.com/14.04.5/ubuntu-14.04.5-server-amd64.iso
    - source_hash:
sha256=dde07d37647a1d2d9247e33f14e91acb10445a97578384896b4e1d985f75
4cc1
# Mount the ISO using the loop option.  Since this is supposed to
be read only media, we don't need to copy files locally.
mount_ub1404x64server_iso:
  file.directory:
    - name: /var/www/html/ubuntu1404x64server
  mount.mounted:
    - name: /var/www/html/ubuntu1404x64server
    - device: /media/ubuntu-14.04.5-server-amd64.iso
    - fstype: iso9660
    - opts:
      - loop
# After the ISO is mounted, the kernel and ramdisk need to be
copied from apache to the local TFTP Server to enable this image to
PXE boot
ub1404x64server_kernel:
  file.managed:
    - name: /var/lib/tftpboot/ubuntu1404x64server/linux
    - user: root
    - group: root
    - mode: 644
    - source:
http://127.0.0.1/ubuntu1404x64server/install/netboot/ubuntu-install
er/amd64/linux
    - makedirs: true
    - skip_verify: true
ub1404x64server_ramdisk:
  file.managed:
    - name: /var/lib/tftpboot/ubuntu1404x64server/initrd.gz
    - user: root
    - group: root
    - mode: 644
    - source: 
http://127.0.0.1/ubuntu1404x64server/install/netboot/ubuntu-install
er/amd64/initrd.gz
    - skip_verify: true
# The preseed file specifies all the values necessary in order to
fully automate the OS intall
ub1404x64server_preseed:
  file.managed:
    - name: /var/www/html/ub1404x64server.preseed
    - user: root
    - group: root
    - mode: 644
    - source: salt://pxesaltbase/ub1404x64server.preseed
    - template: jinja
```

- `file.managed` pulls the 14.04x64 Server ISO from the Ubuntu website and verifies it using the sha256 hash.
- Just as above, the mount folder is created (using `file.directory`) and the ISO is mounted (using `mount.mounted`).
- Also just as above, both the kernel and ramdisk are pulled from the mounted media using `file.managed`. As noted above, the location and name of the kernel and ramdisk files differ between Ubuntu Desktop and Ubuntu Server.
- Finally, we pull in the last file, ub1404x64server.preseed using `file.managed`. This file also includes jinja templating.

Salt: Memtest Boot

```
# Setting up memtest as a boot option
get_memtest:
  file.managed:
    - name: /var/lib/tftpboot/memtest/memtest
    - user: root
    - group: root
    - mode: 644
    - source:
http://www.memtest.org/download/5.01/memtest86+-5.01.bin
    - makedirs: true
    - skip_verify: true
```

- For the final menu item, we use `file.managed` to download the bootable image for memtest (since it's small we skip verification). Also note that we make use of `makedirs: true` for creating the containing folder(s) as part of the state.

Salt: NFS Configuration

```
# Configuring NFS for those boot options which require local media
to be mounted
# In this case, only the Ubuntu live boot option relies on NFS
nfs_config:
  file.managed:
    - name: /etc/exports
    - user: root
    - group: root
    - mode: 644
    - source: salt://pxesaltbase/exports
  service.running:
    - name: nfs-kernel-server
    - reload: true
    - watch:
      - file: /etc/exports
```

- Here we use `file.managed` to grab the nfs config file "exports" from our local source.
- We also use the same "watch" technique (as part of `service.running`) for restarting the NFS service as we did for the tftpd-hpa service.
- Although technically only the live boot requires NFS, it is possible that a future menu item will need to make use of this configuration as well. As such, we have decoupled the states for nfs from those of live boot.

Salt: PXE Boot Menu & Bootstrap Files

```
# Building the PXE Boot menu and prerequisites
pxe_menu_default:
  file.managed:
    - name: /var/lib/tftpboot/pxelinux.cfg/default
    - user: root
    - group: root
    - mode: 644
    - source: salt://pxesaltbase/default
    - template: jinja
    - makedirs: true
pxe_req1:
  file.managed:
    - name: /var/lib/tftpboot/pxelinux.0
    - user: root
    - group: root
    - mode: 644
    - source:
http://127.0.0.1/ubuntu1604x64server/install/netboot/pxelinux.0
    - skip_verify: true
pxe_req2:
  file.managed:
    - name: /var/lib/tftpboot/ldlinux.c32
    - user: root
    - group: root
    - mode: 644
    - source:
http://127.0.0.1/ubuntu1604x64server/install/netboot/ldlinux.c32
    - skip_verify: true
pxe_req3:
  file.managed:
    - name: /var/lib/tftpboot/vesamenu.c32
    - user: root
    - group: root
    - mode: 644
    - source:
http://127.0.0.1/ubuntu1604x64server/install/netboot/ubuntu-install
er/amd64/boot-screens/vesamenu.c32
    - skip_verify: true
pxe_req4:
  file.managed:
    - name: /var/lib/tftpboot/libcom32.c32
    - user: root
    - group: root
    - mode: 644
    - source:
http://127.0.0.1/ubuntu1604x64server/install/netboot/ubuntu-installer/amd64/boot-screens/libcom32.c32
    - skip_verify: true
pxe_req5:
  file.managed:
    - name: /var/lib/tftpboot/libutil.c32
    - user: root
    - group: root
    - mode: 644
    - source:
http://127.0.0.1/ubuntu1604x64server/install/netboot/ubuntu-installer/amd64/boot-screens/libutil.c32
    - skip_verify: true
```

- Our first state uses `file.managed` to grab the PXE boot menu and apply the jinja.
- The remaining states copy the PXE Server bootstrap files to the correct location within the tftpboot folder. Since the Ubuntu Server 16.04x64 media is already mounted, we can simply use this as the source for the files.

Salt: Add the Salt Config files and Formula to PXE

```
# Create the saltbase_install folder and drop in the supporting
files and salt
sbinstall_folder:
  file.directory:
    - name: /var/www/html/saltbase_install
sbinstall_1604list:
  file.managed:
    - name: /var/www/html/saltbase_install/new1604saltstack.list
    - user: root
    - group: root
    - mode: 644
    - source:
salt://pxesaltbase/saltbase_install/new1604saltstack.list
sbinstall_1404list:
  file.managed:
    - name: /var/www/html/saltbase_install/new1404saltstack.list
    - user: root
    - group: root
    - mode: 644
    - source:
salt://pxesaltbase/saltbase_install/new1404saltstack.list
sbinstall_installconf:
  file.managed:
    - name: /var/www/html/saltbase_install/saltbase_install.conf
    - user: root
    - group: root
    - mode: 644
    - source:
salt://pxesaltbase/saltbase_install/saltbase_install.conf
sbinstall_devopsconf:
  file.managed:
    - name: /var/www/html/saltbase_install/devops_standard.conf
    - user: root
    - group: root
    - mode: 644
    - source:
salt://pxesaltbase/saltbase_install/devops_standard.conf
sbinstall_gitfsconf:
  file.managed:
    - name: /var/www/html/saltbase_install/gitfs_remotes.conf
    - user: root
    - group: root
    - mode: 644
    - source: salt://pxesaltbase/saltbase_install/gitfs_remotes.conf
sbinstall_topsls:
  file.managed:
    - name: /var/www/html/saltbase_install/top.sls
    - user: root
    - group: root
    - mode: 644
    - source: salt://pxesaltbase/saltbase_install/top.sls
sbinstall_salt:
  file.managed:
    - name: /var/www/html/saltbase_install/saltbase_install.sls
    - user: root
    - group: root
    - mode: 644
    - source:
salt://pxesaltbase/saltbase_install/saltbase_install.sls
    - template: jinja
```

- The first state `sbinstall_folder` builds the folder for collecting the salt files that will be applied across both Ubuntu Server installations.
- The next two states (`sbinstall_1604list & sbinstall_1404list`) are used to import the .list files. These are used to add the apt source for the updated salt packages to the SaltBase machines during the application of the preseed file.
- The next two states (`sbinstall_installconf & sbinstall_devopsconf`) import the salt-master configurations. A detailed look into these files and how they are applied to the SaltBase machine can be found in the Understanding the difference in salt-master configuration files in Part II.
- The `sbinstall_gitfsconf` state imports the gitfs_remotes.conf placeholder file used for adding remote gitfs sources to a SaltBase machine.
- The `sbinstall_topsls` state imports a bare top.sls file used for new SaltBase machines.
- The `sbinstall_salt` state imports the salt formula used to complete the SaltBase configuration on target hosts. Note that this file is the only one which contains jinja, in this case for inserting the correct DNS name of the PXE server into the formula for file source paths.

Salt: Helper Scripts

```
# Drop in the helper scripts.  Note that these will be accessible
from the webserver, where they can be copied to the target host by
the saltbase_install.sls salt formula.
saltbase_helper1:
  file.managed:
    - name: /var/www/html/saltbase_install/enable_ssh.sh
    - user: root
    - group: root
    - mode: 644
    - source: salt://pxesaltbase/saltbase_install/enable_ssh.sh
saltbase_helper2:
  file.managed:
    - name: /var/www/html/saltbase_install/newsalthostname.sh
    - user: root
    - group: root
    - mode: 644
    - source:
salt://pxesaltbase/saltbase_install/newsalthostname.sh
saltbase_helper3:
  file.managed:
    - name: /var/www/html/saltbase_install/setupnetwork.sh
    - user: root
    - group: root
    - mode: 644
    - source: salt://pxesaltbase/saltbase_install/setupnetwork.sh
```

- We use (3) `file.managed` states to copy our helper scripts into the `saltbase_install` folder of the web server. As noted in the inline comments, it is important these files are available from the web server so that an auto-deployed SaltBase machine can grab them as part of the saltbase_install.sls salt formula highstate.

- Note that in the manual process, the helper scripts are downloaded from the web using URLs. Auto-deploying PXE using SaltStack allows these scripts to be grabbed directly from the salt folder. This provides additional flexibility by allowing an administrator to make changes to these scripts and have those changes preserved for all auto-deployed SaltBase machines without the need to change the salt itself.

Salt: .bashrc

```
# Drop in a new .bashrc file.  This contains post install
instructions and fixes the nfsroot hostname limitation in
/var/lib/tftpboot/pxelinux.cfg/default
update_bashrc:
  file.managed:
    - name: /root/.bashrc
    - user: root
    - group: root
    - mode: 644
    - source: salt://pxesaltbase/.bashrc
    - template: jinja
```

- We use a single `file.managed` state to copy in the last file from our local source and apply jinja templating.

- In addition to helping fix the limitation of nfs mounting as it pertains to the live boot, `.bashrc` is slightly different from the manual steps outlined above. These changes are tracked in more detail in the `.bashrc` section below.

PXE boot menu (default)

The most basic example of salt with jinja templating can be found in the PXE Boot Menu file, *default*. Comparing this file to our manually created PXE boot menu, we see a single line change.

Line	10
default manual	MENU TITLE DevOps Awesome PXE Boot Menu
default auto-deploy	MENU TITLE {{ pillar.get('PXE_MENU_TITLE') }}
Details	In this case we pull the PXE Boot Menu Title into a pillar, allowing for easy customization..

Preseed files (ub1604x64server.preseed & ub1404x64server.preseed)

The preseed files are *slightly* more complex, based on the jinja templating applied as part of the Salt auto-deploy process. We'll go over the differences on a per line basis to evaluate the changes.

Line	131-132
ub1604x64server ub1404x64server manual	d-i passwd/root-password password r00tme d-i passwd/root-password-again password r00tme

ub1604x64server ub1404x64server auto-deploy	d-i passwd/root-password password {{ pillar.get('DEFAULT_ROOT_PASSWORD') }}
	d-i passwd/root-password-again password {{ pillar.get('DEFAULT_ROOT_PASSWORD') }}

Details	This is an easy and simple replacement where a default password is replaced with a value stored in the pillar. Salt inserts the value from the pillar when the preseed file is written to the filesystem. Note this change is valid for both ub1604x64server and ub1404x64server preseed files.

Line	490
ub1604x64server manual	in-target wget -P /etc/apt/sources.list.d/ http://pxe.bobbarker.com/saltbase_install/new1604saltstack.list; \
ub1604x64server auto-deploy	in-target wget -P /etc/apt/sources.list.d/ http://{{ pillar.get('PXE_DNS_NAME') }}/saltbase_install/new1604saltstack.list; \
Details	Again, this is an easy and simple replacement where the DNS name is pulled from the pillar and applied to the preseed file. This line downloads the updated apt source for SaltStack directly from the PXE server.

Line	490
ub1404x64server manual	in-target wget -P /etc/apt/sources.list.d/ http://pxe.bobbarker.com/saltbase_install/new1404saltstack.list; \
ub1404x64server auto-deploy	in-target wget -P /etc/apt/sources.list.d/ http://{{ pillar.get('PXE_DNS_NAME') }}/saltbase_install/new1404saltstack.list; \
Details	Exactly the same as the lines above, but replace 16.04 with 14.04.

Line	495- 497
ub1604x64server ub1404x64server manual	in-target wget -P /etc/salt/master.d/ http://pxe.bobbarker.com/saltbase_install/saltbase_install.conf; \ in-target wget -P /srv/salt/ http://pxe.bobbarker.com/saltbase_install/top.sls; \ in-target wget -P /srv/salt/ http://pxe.bobbarker.com/saltbase_install/saltbase_install.sls; \
ub1604x64server ub1404x64server auto-deploy	in-target wget -P /etc/salt/master.d/ http://{{ pillar.get('PXE_DNS_NAME') }}/saltbase_install/saltbase_install.conf; \ in-target wget -P /srv/salt/ http://{{ pillar.get('PXE_DNS_NAME') }}/saltbase_install/top.sls; \ in-target wget -P /srv/salt/ http://{{ pillar.get('PXE_DNS_NAME') }}/saltbase_install/saltbase_install.sls; \
Details	Again, we variabilize the DNS name for the PXE Server, in this case to allow for downloading of the salt files and the salt install config directly from the PXE server itself. Note this change is valid for both ub1604x64server and ub1404x64server preseed files.

Side Note: As detailed above in the manual process, the simple way to set the root password uses plaintext input, but a more secure way to accomplish this is by pre-hashing the password and using the root-password-crypted command. First, get the hash of the password using mkpasswd, drop the value into the pillar, then make the following changes to the root-password-crypted command. (mkpasswd requires the whois package be installed)

```
mkpasswd -H md5 "yourpassword"
```

Line	134
Original	#d-i passwd/root-password-crypted password [crypt(3) hash]
ub1604x64server	d-i passwd/root-password-crypted password {{ pillar.get('CRYPTED_ROOT_PASSWORD') }}
Details	Uncommented to automate the setting of the root password using the pre-hashed value stored in the pillar..

The jinja templating in the salt version of the preseed files adds a small amount of complexity vs. the manual process. From an overall scope, this is balanced by the simple customizability offered through the use of the pillar data.

.bashrc

Although it seems like this would be a simpler file (with only a single line of jinja) .bashrc is the first real divergence from the manual process that we've outlined above. The manual process is still represented by the code to fix nfs, but the remaining lines are meant to notify the user of the steps which must be performed before the PXE server configuration is complete. These steps were performed manually above as part of the PXE prerequisites and External prerequisites sections, but cannot be automated and are therefore called out using a block of reminders echoed to the console.

Line	111-119
ub1604x64server ub1404x64server manual	Lines do not exist

ub1604x64server ub1404x64server auto-deploy	# Add the following reminders for the configuration of the PXE server after an automated salt build echo "" echo "********************" echo "On initial boot, you should perform the following to complete the PXE Server configuration:" echo " - Assign a static IP to the PXE server using the /root/setupnetwork.sh script" echo " - Change the hostname using the /root/newsalthostname.sh script" echo " - Add a {{ pillar.get('PXE_DNS_NAME') }} DNS record" echo "********************" echo ""
Details	These lines are added to remind the user (after the imaging of the PXE server is complete) of the remaining tasks which must be completed before the PXE server will perform as expected. This pulls the same DNS name from the pillar as is used above in other files with jinja templating..

salt formula (saltbase_install.sls)

In much the same way as the preseed files, the salt formula includes jinja templating as well

Line	8, 9, 10
ub1604x64server ub1404x64server manual	wget -P /root/ https://raw.githubusercontent.com/love2scoot/pxesaltbase-formula/master/pxesaltbase/enable_ssh.sh wget -P /root/ https://raw.githubusercontent.com/love2scoot/pxesaltbase-formula/master/pxesaltbase/setupnetwork.sh wget -P /root/ https://raw.githubusercontent.com/love2scoot/pxesaltbase-formula/master/pxesaltbase/newsalthostname.sh
ub1604x64server ub1404x64server auto-deploy	wget -P /root/ http://{{ pillar.get('PXE_DNS_NAME') }}/enable_ssh.sh wget -P /root/ http://{{ pillar.get('PXE_DNS_NAME') }}/setupnetwork.sh wget -P /root/ http://{{ pillar.get('PXE_DNS_NAME') }}/newsalthostname.sh
Details	A simple replacement of the source URL for the helper scripts. Since we know that these scripts will be served from the PXE server, we can use the DNS name in the copy process. Note this change is valid for both ub1604x64server and ub1404x64server SaltBase machines..

Static Files

The content of the remaining files do not change between the manual and automated PXE build processes:

exports
tftpd-hpa
top.sls
new1404saltstack.list
new1604saltstack.list
newsalthostname.sh
saltbase_install.conf
devops_standard.conf
gitfs_remotes.conf
setupnetwork.sh
enable_ssh.sh

Pillar Customization

In this section we will step through the pillar and see how easy it is to customize. Note that by convention, the pillar itself will be stored as an example file in the repo. In this case we've named the pillar `pxesaltbase.sls.pillarexample`.

PXESaltBase Pillar Contents

```
PXE_MENU_TITLE: 'DevOps Awesome PXE Boot Menu'
PXE_DNS_NAME: 'pxe.bobbarker.com'
DEFAULT_ROOT_PASSWORD: 'INeedToBeChanged'
CRYPTED_ROOT_PASSWORD: '$1$ROnGTxWk$yaVNw1c07.K2y8VJfmFQF0'
```

- `PXE_MENU_TITLE` is only used once in *default*, the PXE Boot Menu file. This can be left as is or customized to the user's preference. Note the use of single quotes here: **the title should not include anything with single quotes as this will break the markup.** For example, `'Bob's Menu'` would not work.
- `PXE_DNS_NAME` is used in the preseed files, .bashrc, and the saltbase_install.sls salt formula. This must be changed to the DNS name chosen for the PXESaltBase server.
- `DEFAULT_ROOT_PASSWORD` is used only once, in the preseed files. This can be left as is or customized to the user's preference. **Note that a password under 8 characters will elicit a warning from the Ubuntu installer during all install processes to PXE clients.** It's a good practice to keep this password at 8 characters or above.
- `CRYPTED_ROOT_PASSWORD` is used only optionally in the preseed files (if the more secure method of setting the root password is used). This can be left as is or customized to the user's preference. This is the hash of the default password.

How to use the pillar

Salt can be accessed from any number of places depending on how a SaltBase machine is configured, basically falling into two categories: a) *local* salt in /srv/salt and b) *remote* salt in remote gitfs repos. Interestingly, the pillar should always be *local* since it contains values that are either sensitive or specific to the deployed machines. Technically a pillar *could* reside within a repo (with edits being committed) but this is not considered good practice. The good news here is that the pillar will work regardless of where the salt resides. Since formula names are unique across all salt sources, the pillar will match one (and only one) formula. To use the pillar simply customize the content of the file before running a salt highstate. See the Set up the Pillar section below for more detail.

Build Walkthrough

Now that we have looked at both the salt and pillar, it is time to set up the process for building the PXESaltBase machine.

Deploy a new VM

If you didn't skip ahead, you'll know you have two options here:

- Deploy a SaltBase machine from an .ova
- Deploy a SaltBase machine from PXE

Set up the Salt

After you have logged into the machine, you will need to set the remote gitfs repository for pxesaltbase on the host.

1. Open our gitfs_remotes.conf file for editing

   ```
   nano /etc/salt/master.d/gitfs_remotes.conf
   ```

2. Remove the # in front of gitfs_remotes:, add the following content, and **save** the file. (new content in light green)

   ```
   gitfs_remotes:
   #  - https://gitsource.bobbarker.com/happy.git:
   #    - user: for protected sources
   #    - password: for protected sources
   #    - root: salt
     - https://github.com/love2scoot/pxesaltbase-formula.git
   ```

3. Edit the top file to include the new formula

   ```
   nano /srv/salt/top.sls
   ```

4. Add the following content to top.sls and **save** the file. (new content in light green)

   ```
   base:
     '*':
       - pxesaltbase
   #    - saltbase_install
   ```

5. Restart the Salt master in order to pick up these changes:

```
service salt-master restart
```

6. Update the local cache of the remote sources (should return **TRUE**):

```
salt-run fileserver.update
```

Set up the Pillar

1. Copy the pillar into the correct local folder

```
wget -P /srv/pillar
https://raw.githubusercontent.com/love2scoot/pxesaltbase-for
mula/master/pxesaltbase.sls.pillarexample
```

2. Rename the pillar

```
mv /srv/pillar/pxesaltbase.sls.pillarexample
/srv/pillar/pxesaltbase.sls
```

3. Open the pillar for editing, customize it (watch for quoting), and save the file.

```
nano /srv/pillar/pxesaltbase.sls
```

4. Copy the salt top file to the pillar top file

```
cp /srv/salt/top.sls /srv/pillar/top.sls
```

Deployment and Testing

Everything should be ready to go! Let's...

Make it so!

1. Run a highstate to deploy the PXESaltBase to the machine, but redirect the output to a file for inspection.

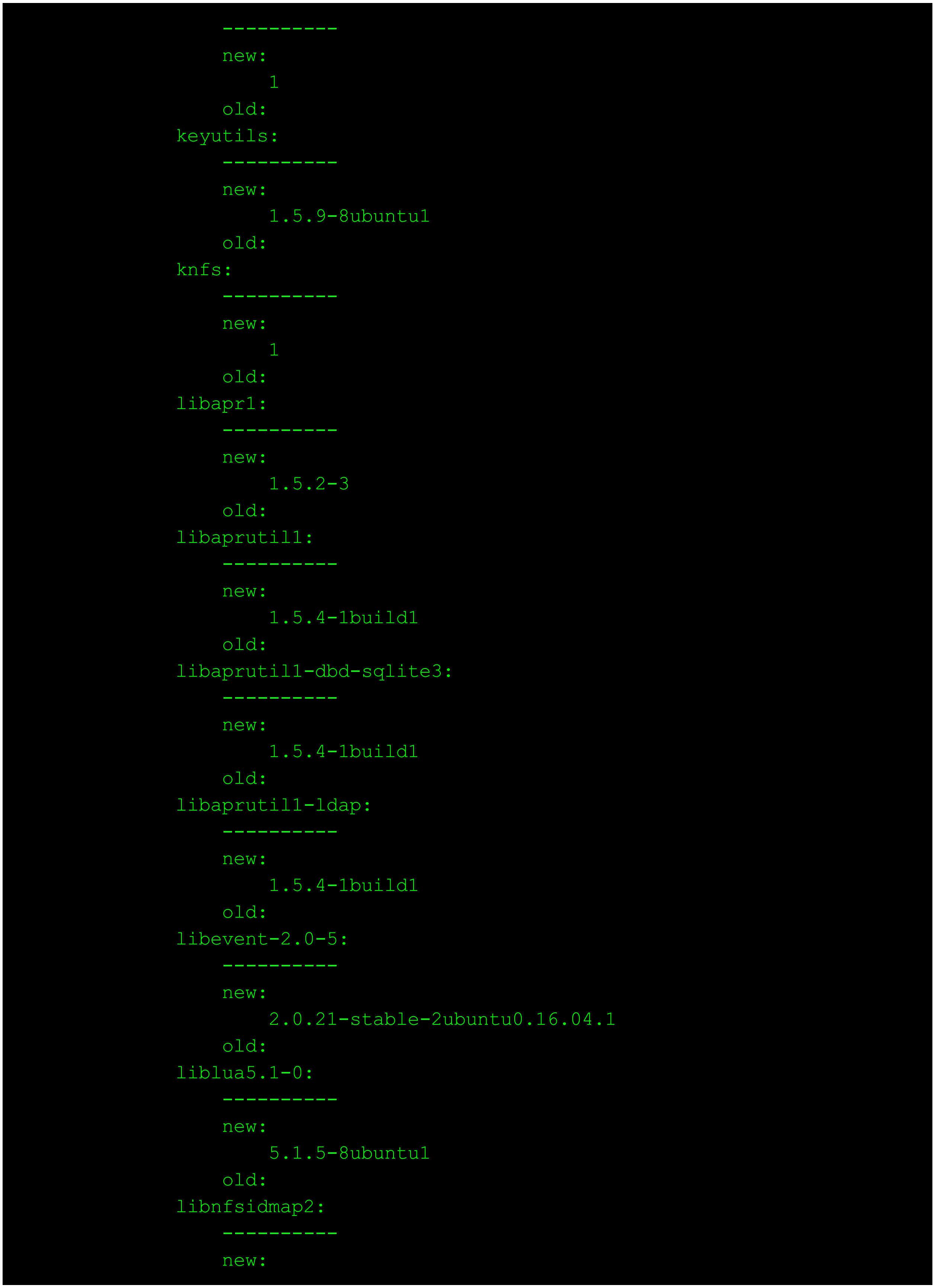
```
salt '*' state.highstate > ~/pxesaltbase.log
```

Side Note: Due to the fact that the salt for the PXE Server downloads (3) ISO files and (1) boot image, the highstate will require several minutes to complete. This timing can be highly variable based on internet connection throughput as well as load on the servers hosting the ISO files. A good rule of thumb is to check back after approximately 15 minutes.

Stepping through the Salt-Master report

Below is the full report echoed by the Salt-Master after the highstate command was run on a SaltBase machine set to deploy the pxesaltbase formula. We'll step through this report in sections to help clarify what to expect and call attention to some interesting details.

```
pxesaltbasetest:
----------
          ID: pxe_packages
    Function: pkg.installed
      Result: True
     Comment: 3 targeted packages were installed/updated.
     Started: 16:07:50.855672
    Duration: 18976.478 ms
     Changes:
              ----------
              apache2:
                  ----------
                  new:
                      2.4.18-2ubuntu3.4
                  old:
              apache2-api-20120211:
                  ----------
                  new:
                      1
                  old:
              apache2-bin:
                  ----------
                  new:
                      2.4.18-2ubuntu3.4
                  old:
              apache2-data:
                  ----------
                  new:
                      2.4.18-2ubuntu3.4
                  old:
              apache2-utils:
                  ----------
                  new:
                      2.4.18-2ubuntu3.4
                  old:
              httpd:
                  ----------
                  new:
                      1
                  old:
              httpd-cgi:
```



```
    ----------
    new:
        1
    old:
keyutils:
    ----------
    new:
        1.5.9-8ubuntu1
    old:
knfs:
    ----------
    new:
        1
    old:
libapr1:
    ----------
    new:
        1.5.2-3
    old:
libaprutil1:
    ----------
    new:
        1.5.4-1build1
    old:
libaprutil1-dbd-sqlite3:
    ----------
    new:
        1.5.4-1build1
    old:
libaprutil1-ldap:
    ----------
    new:
        1.5.4-1build1
    old:
libevent-2.0-5:
    ----------
    new:
        2.0.21-stable-2ubuntu0.16.04.1
    old:
liblua5.1-0:
    ----------
    new:
        5.1.5-8ubuntu1
    old:
libnfsidmap2:
    ----------
    new:
```

```
                        0.25-5
                    old:
                libtirpc1:
                    ----------
                    new:
                        0.2.5-1
                    old:
                nfs-client:
                    ----------
                    new:
                        1
                    old:
                nfs-common:
                    ----------
                    new:
                        1:1.2.8-9ubuntu12.1
                    old:
                nfs-kernel-server:
                    ----------
                    new:
                        1:1.2.8-9ubuntu12.1
                    old:
                nfs-server:
                    ----------
                    new:
                        1
                    old:
                portmap:
                    ----------
                    new:
                        1
                    old:
                rpcbind:
                    ----------
                    new:
                        0.2.3-0.2
                    old:
                ssl-cert:
                    ----------
                    new:
                        1.0.37
                    old:
                tftp-server:
                    ----------
                    new:
                        1
                    old:
```

```
              tftpd-hpa:
                  ----------
                  new:
                      5.2+20150808-1ubuntu1.16.04.1
                  old:
----------
```

The significant content above is the result of a single state, that of the package install for the PXE server. Although only (3) packages are directed to be installed, Salt uses the system package manager for installation, which will properly install all dependencies as well (thus the install / update of 26 packages). Also note the time of execution, which is ~19 seconds.

```
          ID: tftpd_config
    Function: file.managed
        Name: /etc/default/tftpd-hpa
      Result: True
     Comment: File /etc/default/tftpd-hpa updated
     Started: 16:08:09.838375
    Duration: 119.653 ms
     Changes:
              ----------
              diff:
                  ---
                  +++
                  @@ -2,5 +2,7 @@

                   TFTP_USERNAME="tftp"
                   TFTP_DIRECTORY="/var/lib/tftpboot"
                  -TFTP_ADDRESS=":69"
                  +TFTP_ADDRESS="[::]:69"
                   TFTP_OPTIONS="--secure"
                  +RUN_DAEMON="yes"
                  +OPTIONS="-l -s /var/lib/tftpboot"
----------
          ID: tftpd_config
    Function: service.running
        Name: tftpd-hpa
      Result: True
     Comment: Running scope as unit
run-rd9003b6351824924a958563791f54032.scope.
              Failed to reload tftpd-hpa.service: Job type reload is not
applicable for unit tftpd-hpa.service.
              See system logs and 'systemctl status tftpd-hpa.service' for
details.
```

```
     Started: 16:08:10.465685
    Duration: 60.917 ms
     Changes:
----------
```

The states above are those which correspond to the configuration of the TFTP server. In the first state results, we can see the diff of the salt managed file vs. the file on the disk. This clocks in at ~120ms.

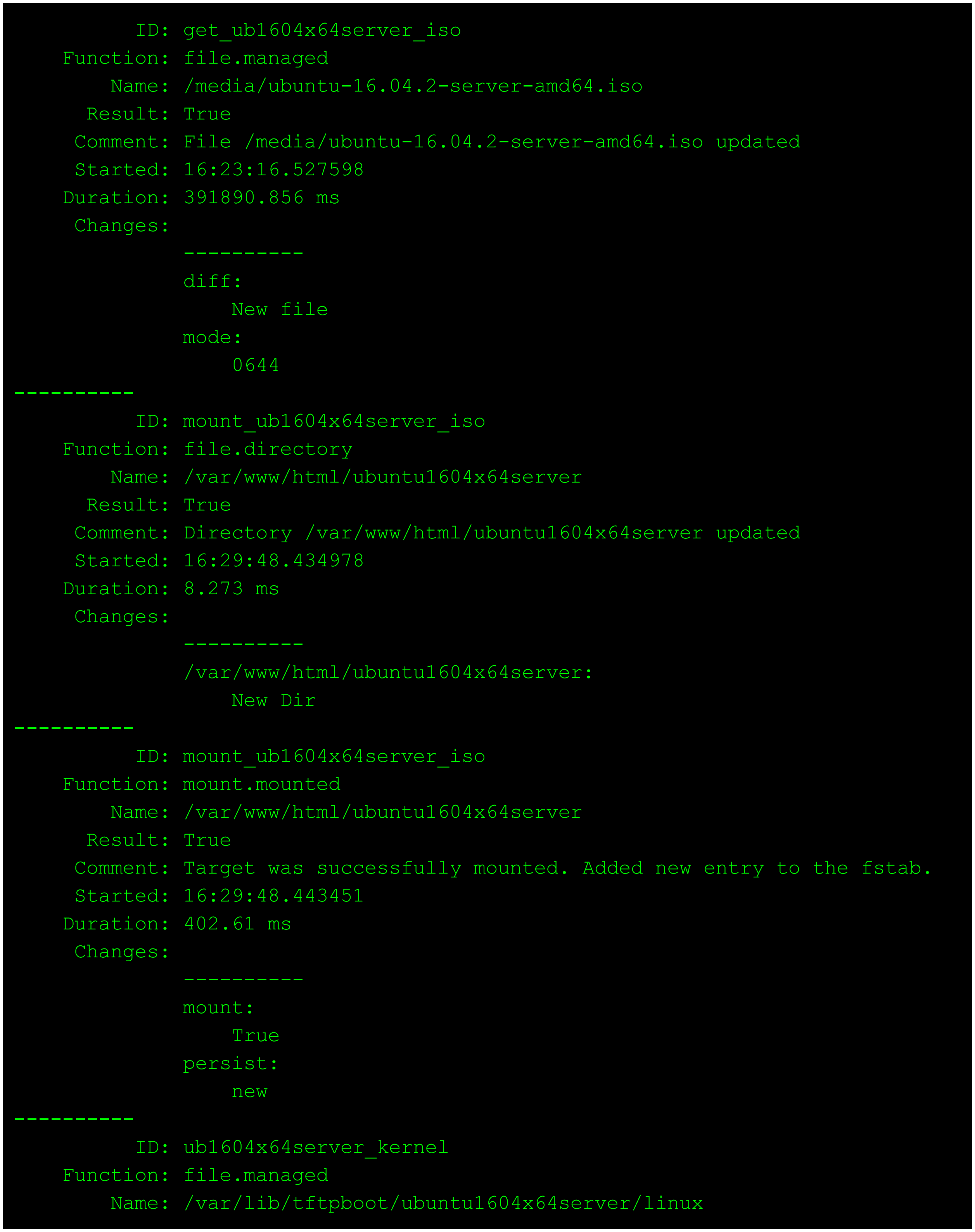
The second set is the attempt by Salt to restart the tftpd-hpa service (since the tftpd-hpa file changed). It is unclear why this does not work, since these steps complete without incident in the manual TFTP process. This does not report an error, but simply does not complete since the specified option does not work for this service. This state was quick at ~61ms.

```
          ID: get_ub1604x64desktop_iso
    Function: file.managed
        Name: /media/ubuntu-16.04.2-desktop-amd64.iso
      Result: True
     Comment: File /media/ubuntu-16.04.2-desktop-amd64.iso updated
     Started: 16:08:10.526839
    Duration: 904962.673 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
          ID: mount_ub1604x64desktop_iso
    Function: file.directory
        Name: /var/www/html/ubuntu1604x64live
      Result: True
     Comment: Directory /var/www/html/ubuntu1604x64live updated
     Started: 16:23:15.490023
    Duration: 6.253 ms
     Changes:
              ----------
              /var/www/html/ubuntu1604x64live:
                  New Dir
----------
          ID: mount_ub1604x64desktop_iso
    Function: mount.mounted
        Name: /var/www/html/ubuntu1604x64live
      Result: True
     Comment: Target was successfully mounted. Added new entry to the fstab.
     Started: 16:23:15.536178
    Duration: 602.536 ms
     Changes:
              ----------
              mount:
                  True
              persist:
                  new
----------
          ID: ub1604x64desktop_kernel
    Function: file.managed
        Name: /var/lib/tftpboot/ubuntu1604x64live/vmlinuz.efi
      Result: True
     Comment: File /var/lib/tftpboot/ubuntu1604x64live/vmlinuz.efi updated
     Started: 16:23:16.138977
    Duration: 173.077 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
          ID: ub1604x64desktop_ramdisk
    Function: file.managed
        Name: /var/lib/tftpboot/ubuntu1604x64live/initrd.lz
      Result: True
     Comment: File /var/lib/tftpboot/ubuntu1604x64live/initrd.lz updated
     Started: 16:23:16.312228
    Duration: 215.207 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
```

This section consists of those states corresponding to the Ubuntu 16.04x64 Desktop Live Boot.

Our first state is the downloading of the ISO directly from Ubuntu, which is quite time intensive at approximately 905 seconds.

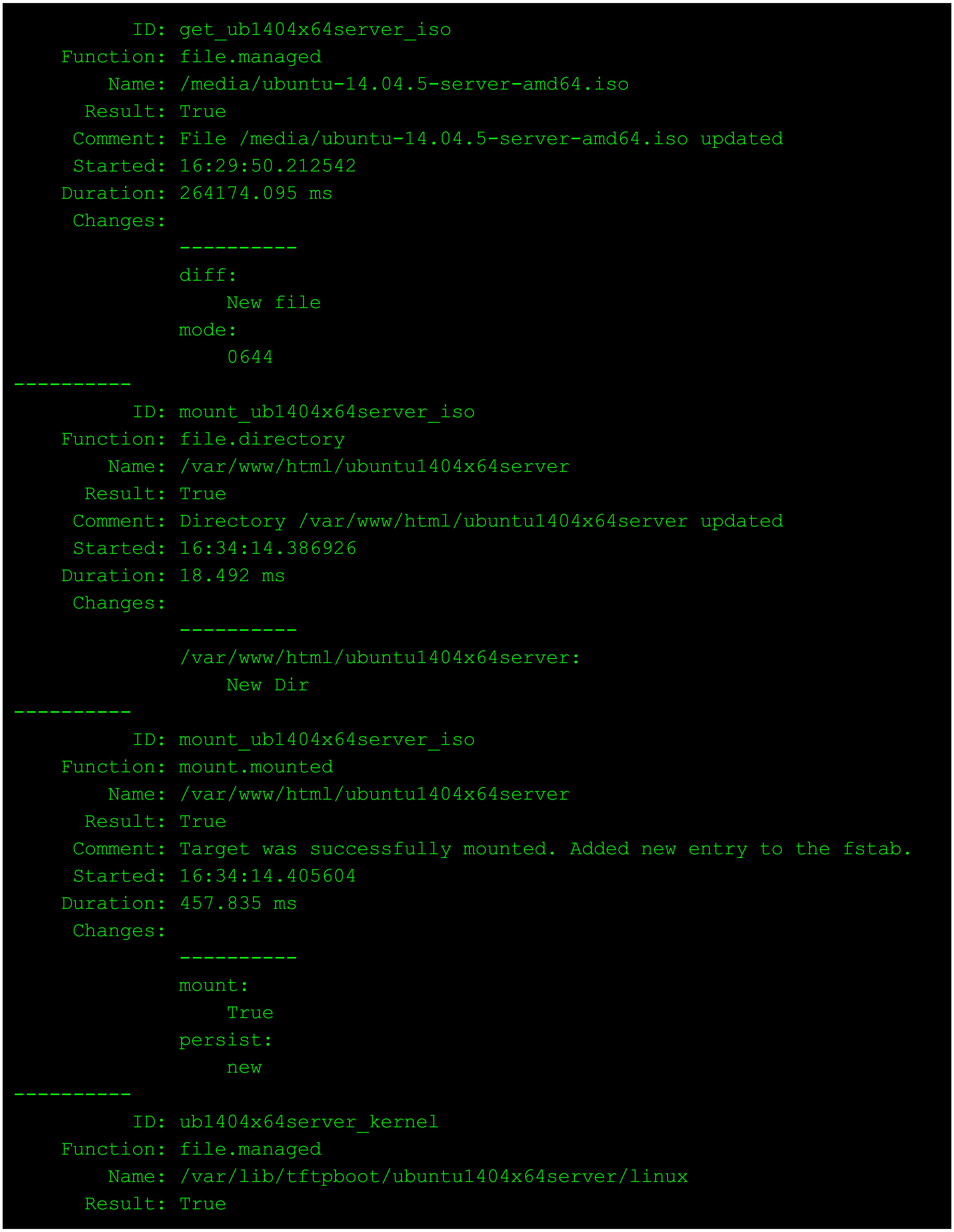
The remaining states create the folder, mount the ISO, and copy two files from the mounted ISO using apache. Cumulatively these states require less than 1 second.



```
          ID: get_ub1604x64server_iso
    Function: file.managed
        Name: /media/ubuntu-16.04.2-server-amd64.iso
      Result: True
     Comment: File /media/ubuntu-16.04.2-server-amd64.iso updated
     Started: 16:23:16.527598
    Duration: 391890.856 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
          ID: mount_ub1604x64server_iso
    Function: file.directory
        Name: /var/www/html/ubuntu1604x64server
      Result: True
     Comment: Directory /var/www/html/ubuntu1604x64server updated
     Started: 16:29:48.434978
    Duration: 8.273 ms
     Changes:
              ----------
              /var/www/html/ubuntu1604x64server:
                  New Dir
----------
          ID: mount_ub1604x64server_iso
    Function: mount.mounted
        Name: /var/www/html/ubuntu1604x64server
      Result: True
     Comment: Target was successfully mounted. Added new entry to the fstab.
     Started: 16:29:48.443451
    Duration: 402.61 ms
     Changes:
              ----------
              mount:
                  True
              persist:
                  new
----------
          ID: ub1604x64server_kernel
    Function: file.managed
        Name: /var/lib/tftpboot/ubuntu1604x64server/linux
      Result: True
     Comment: File /var/lib/tftpboot/ubuntu1604x64server/linux updated
     Started: 16:29:48.846324
    Duration: 220.305 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
          ID: ub1604x64server_ramdisk
    Function: file.managed
        Name: /var/lib/tftpboot/ubuntu1604x64server/initrd.gz
      Result: True
     Comment: File /var/lib/tftpboot/ubuntu1604x64server/initrd.gz updated
     Started: 16:29:49.066787
    Duration: 927.321 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
          ID: ub1604x64server_preseed
    Function: file.managed
        Name: /var/www/html/ub1604x64server.preseed
      Result: True
     Comment: File /var/www/html/ub1604x64server.preseed updated
     Started: 16:29:49.994289
    Duration: 218.122 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
```

This section consists of those states corresponding to the Ubuntu 16.04x64 Server SaltBase install process.

Just as above, the first state downloads the ISO directly from Ubuntu, requiring ~392 seconds to complete.The remaining states are responsible for creating the new folder, mounting the ISO, copying the bootstrap files, and the preseed file over http using apache. Cumulatively these states required ~1.78 seconds to complete.



```
          ID: get_ub1404x64server_iso
    Function: file.managed
        Name: /media/ubuntu-14.04.5-server-amd64.iso
      Result: True
     Comment: File /media/ubuntu-14.04.5-server-amd64.iso updated
     Started: 16:29:50.212542
    Duration: 264174.095 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
          ID: mount_ub1404x64server_iso
    Function: file.directory
        Name: /var/www/html/ubuntu1404x64server
      Result: True
     Comment: Directory /var/www/html/ubuntu1404x64server updated
     Started: 16:34:14.386926
    Duration: 18.492 ms
     Changes:
              ----------
              /var/www/html/ubuntu1404x64server:
                  New Dir
----------
          ID: mount_ub1404x64server_iso
    Function: mount.mounted
        Name: /var/www/html/ubuntu1404x64server
      Result: True
     Comment: Target was successfully mounted. Added new entry to the fstab.
     Started: 16:34:14.405604
    Duration: 457.835 ms
     Changes:
              ----------
              mount:
                  True
              persist:
                  new
----------
          ID: ub1404x64server_kernel
    Function: file.managed
        Name: /var/lib/tftpboot/ubuntu1404x64server/linux
      Result: True
     Comment: File /var/lib/tftpboot/ubuntu1404x64server/linux updated
     Started: 16:34:14.863701
    Duration: 245.897 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
          ID: ub1404x64server_ramdisk
    Function: file.managed
        Name: /var/lib/tftpboot/ubuntu1404x64server/initrd.gz
      Result: True
     Comment: File /var/lib/tftpboot/ubuntu1404x64server/initrd.gz updated
     Started: 16:34:15.109839
    Duration: 911.583 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
          ID: ub1404x64server_preseed
    Function: file.managed
        Name: /var/www/html/ub1404x64server.preseed
      Result: True
     Comment: File /var/www/html/ub1404x64server.preseed updated
     Started: 16:34:16.021608
    Duration: 120.391 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
```

This section consists of those states corresponding to the Ubuntu 14.04x64 Server SaltBase install process.

Just as above, the first state downloads the ISO directly from Ubuntu. Despite the 1404x64 server ISO being approximately the same size as the 16.04x64 ISO, this state clocks in at ~264 seconds (vs ~392 above). The remaining states are responsible for creating the new folder, mounting the ISO, copying the bootstrap files, and the preseed file over http using apache. Cumulatively these states required ~1.75 second to complete.

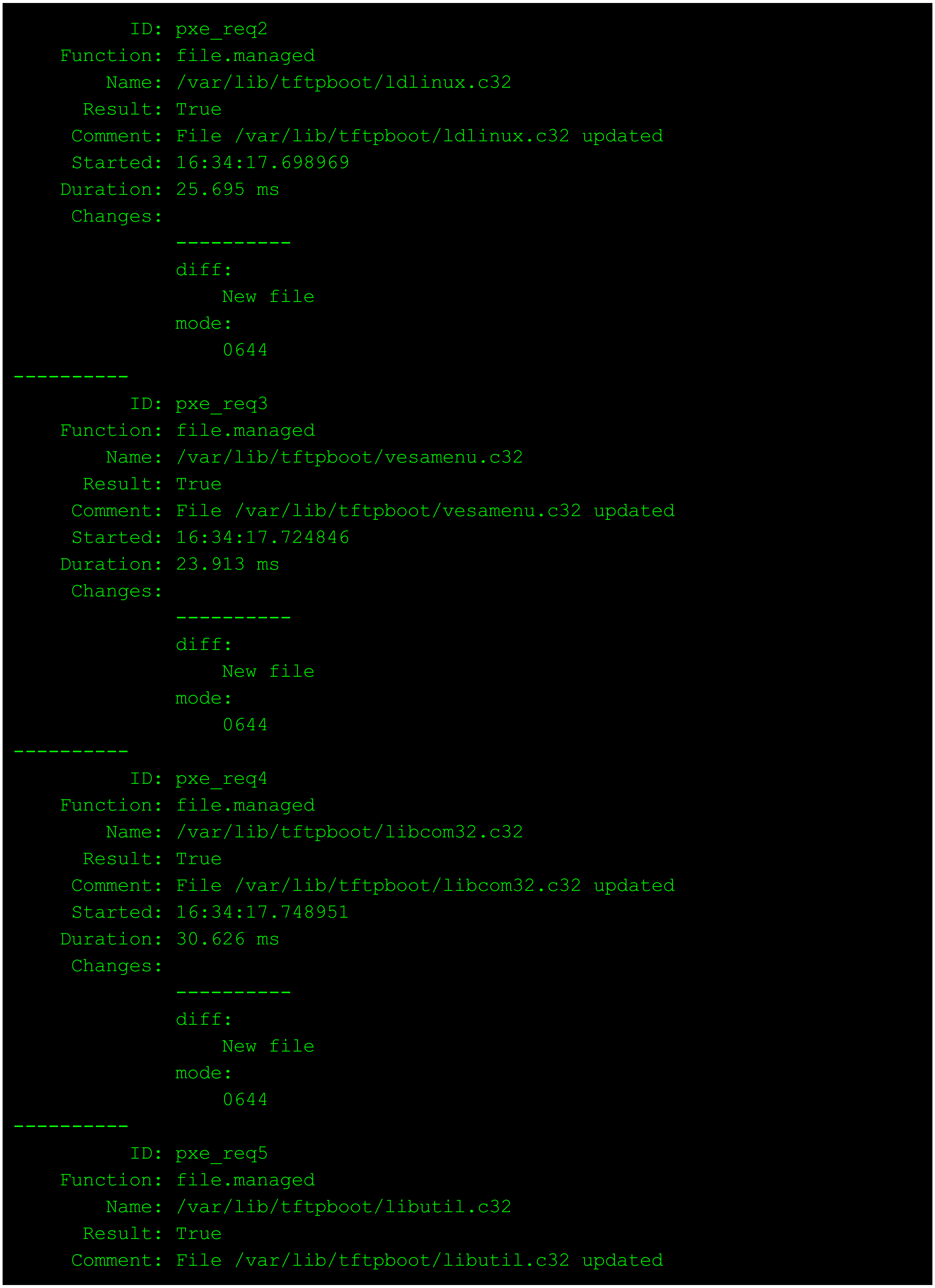
```
          ID: get_memtest
    Function: file.managed
        Name: /var/lib/tftpboot/memtest/memtest
      Result: True
     Comment: File /var/lib/tftpboot/memtest/memtest updated
     Started: 16:34:16.142191
    Duration: 980.782 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
```

This single state handles the download of memtest directly to the TFTP share, making it available as a boot option. This was quite fast at under 1 second to complete.

```
          ID: nfs_config
    Function: file.managed
        Name: /etc/exports
      Result: True
     Comment: File /etc/exports updated
     Started: 16:34:17.123184
    Duration: 49.944 ms
     Changes:
              ----------
              diff:
                  ---
                  +++
                  @@ -8,3 +8,4 @@
                   # /srv/nfs4
gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
                   # /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
                   #
                  +/var/www/html/ubuntu1604x64live/ *
(ro,sync,no_wdelay,insecure_locks,no_root_squash,insecure)
----------
          ID: nfs_config
    Function: service.running
        Name: nfs-kernel-server
      Result: True
     Comment: Service reloaded
     Started: 16:34:17.437769
    Duration: 152.644 ms
     Changes:
              ----------
              nfs-kernel-server:
                  True
----------
```

These two states manage the configuration of NFS. In the first state we see the diff between the salt managed file and the local file while the second reloads the service. These changes are applied quickly with a combined time of ~203ms.

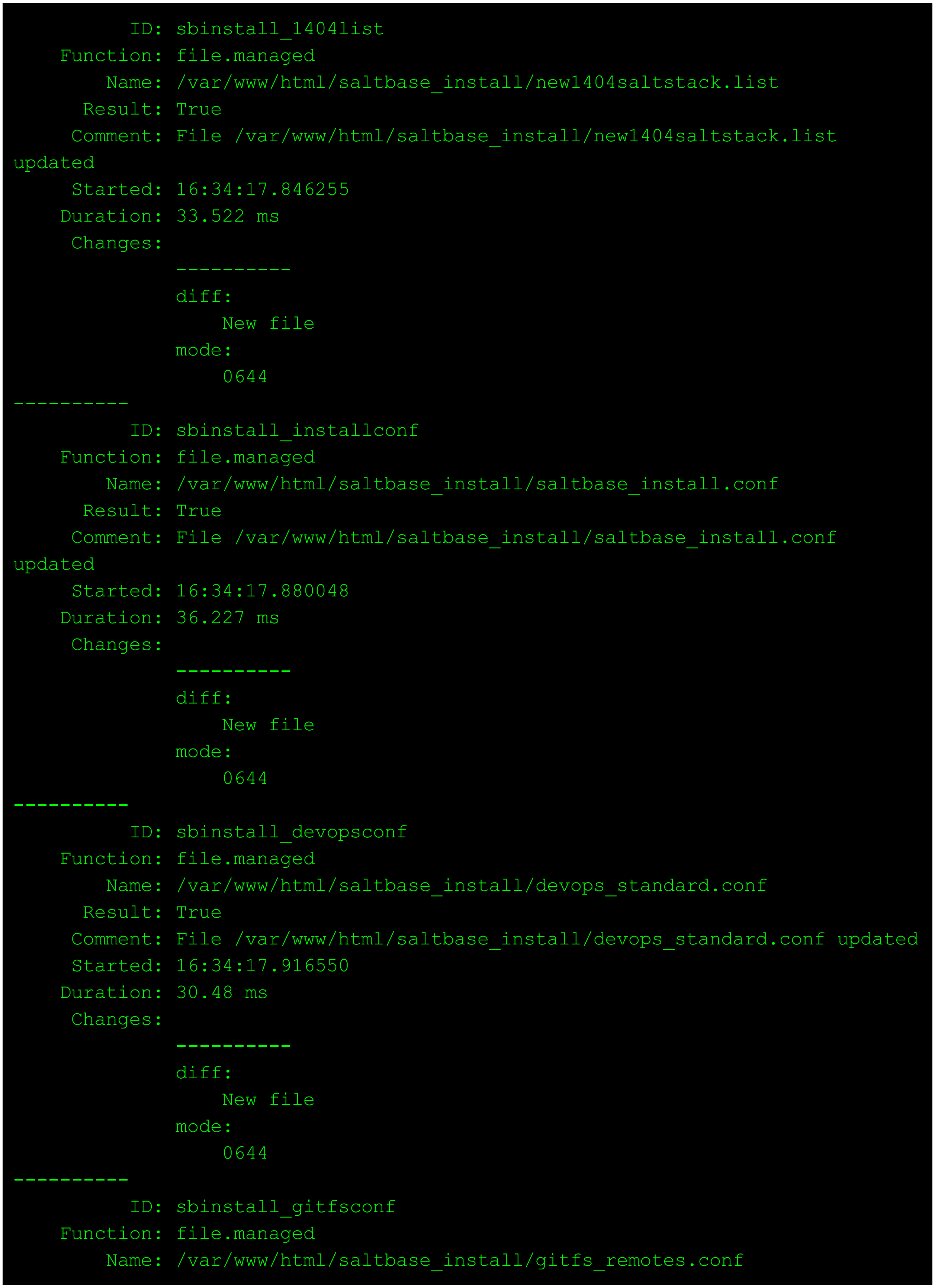
```
          ID: pxe_menu_default
    Function: file.managed
        Name: /var/lib/tftpboot/pxelinux.cfg/default
      Result: True
     Comment: File /var/lib/tftpboot/pxelinux.cfg/default updated
     Started: 16:34:17.590770
    Duration: 51.563 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
          ID: pxe_req1
    Function: file.managed
        Name: /var/lib/tftpboot/pxelinux.0
      Result: True
     Comment: File /var/lib/tftpboot/pxelinux.0 updated
     Started: 16:34:17.642495
    Duration: 56.29 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
```



```
          ID: pxe_req2
    Function: file.managed
        Name: /var/lib/tftpboot/ldlinux.c32
      Result: True
     Comment: File /var/lib/tftpboot/ldlinux.c32 updated
     Started: 16:34:17.698969
    Duration: 25.695 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
          ID: pxe_req3
    Function: file.managed
        Name: /var/lib/tftpboot/vesamenu.c32
      Result: True
     Comment: File /var/lib/tftpboot/vesamenu.c32 updated
     Started: 16:34:17.724846
    Duration: 23.913 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
          ID: pxe_req4
    Function: file.managed
        Name: /var/lib/tftpboot/libcom32.c32
      Result: True
     Comment: File /var/lib/tftpboot/libcom32.c32 updated
     Started: 16:34:17.748951
    Duration: 30.626 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
          ID: pxe_req5
    Function: file.managed
        Name: /var/lib/tftpboot/libutil.c32
      Result: True
     Comment: File /var/lib/tftpboot/libutil.c32 updated
     Started: 16:34:17.779765
    Duration: 18.129 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
```

These states simply copy new files to the PXESaltBase machine, namely the PXE Boot Menu and the Bootstrap files. These operations require minimal resources, finishing up all (6) states in ~205ms.

```
          ID: sbinstall_folder
    Function: file.directory
        Name: /var/www/html/saltbase_install
      Result: True
     Comment: Directory /var/www/html/saltbase_install updated
     Started: 16:34:17.798174
    Duration: 1.511 ms
     Changes:
              ----------
              /var/www/html/saltbase_install:
                  New Dir
----------
          ID: sbinstall_1604list
    Function: file.managed
        Name: /var/www/html/saltbase_install/new1604saltstack.list
      Result: True
     Comment: File /var/www/html/saltbase_install/new1604saltstack.list
updated
     Started: 16:34:17.799916
    Duration: 46.184 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
```



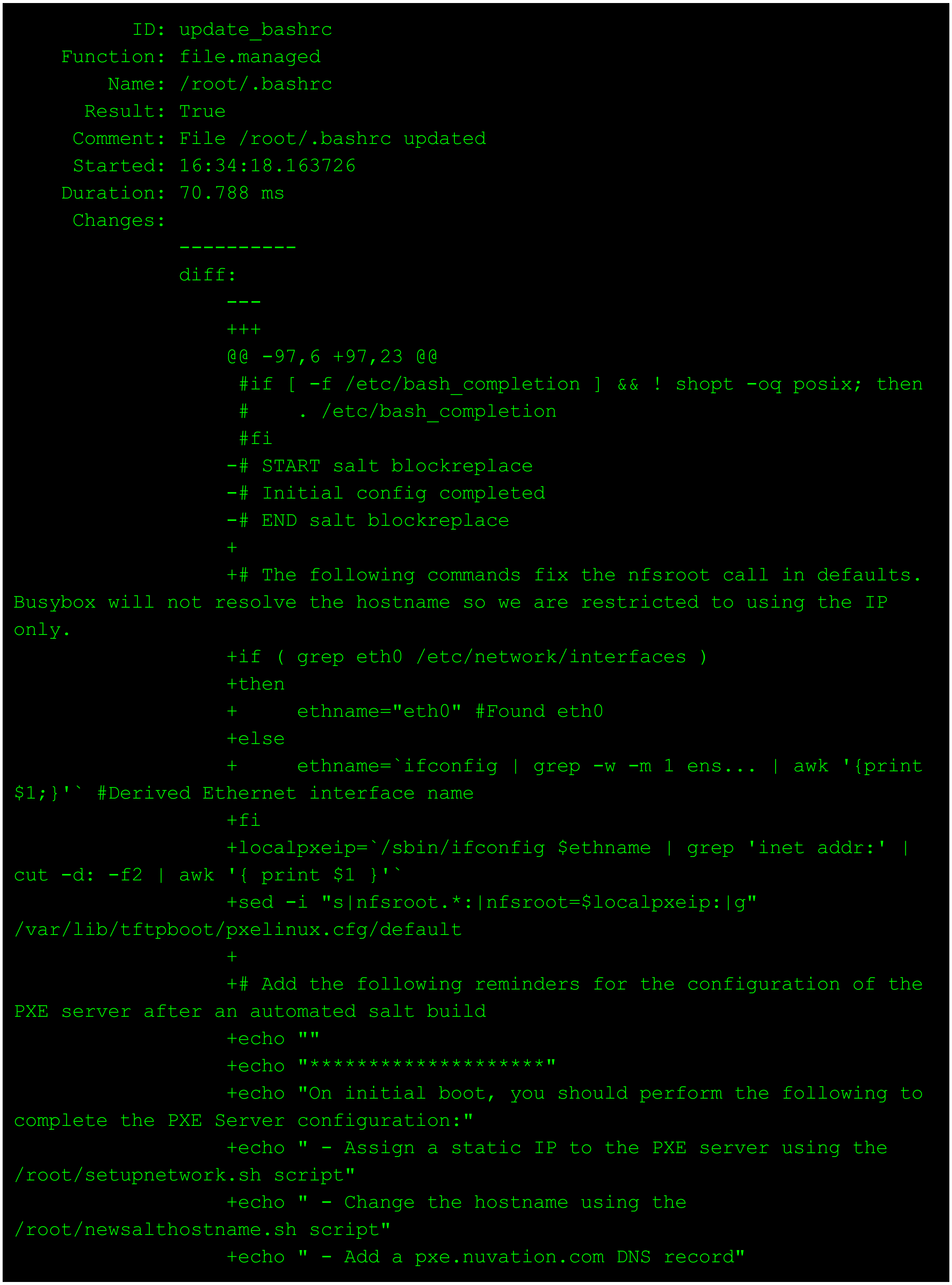
```
          ID: sbinstall_1404list
    Function: file.managed
        Name: /var/www/html/saltbase_install/new1404saltstack.list
      Result: True
     Comment: File /var/www/html/saltbase_install/new1404saltstack.list
updated
     Started: 16:34:17.846255
    Duration: 33.522 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
          ID: sbinstall_installconf
    Function: file.managed
        Name: /var/www/html/saltbase_install/saltbase_install.conf
      Result: True
     Comment: File /var/www/html/saltbase_install/saltbase_install.conf
updated
     Started: 16:34:17.880048
    Duration: 36.227 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
          ID: sbinstall_devopsconf
    Function: file.managed
        Name: /var/www/html/saltbase_install/devops_standard.conf
      Result: True
     Comment: File /var/www/html/saltbase_install/devops_standard.conf updated
     Started: 16:34:17.916550
    Duration: 30.48 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
          ID: sbinstall_gitfsconf
    Function: file.managed
        Name: /var/www/html/saltbase_install/gitfs_remotes.conf
      Result: True
     Comment: File /var/www/html/saltbase_install/gitfs_remotes.conf updated
     Started: 16:34:17.947184
    Duration: 31.504 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
          ID: sbinstall_topsls
    Function: file.managed
        Name: /var/www/html/saltbase_install/top.sls
      Result: True
     Comment: File /var/www/html/saltbase_install/top.sls updated
     Started: 16:34:17.978872
    Duration: 35.276 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
          ID: sbinstall_salt
    Function: file.managed
        Name: /var/www/html/saltbase_install/saltbase_install.sls
      Result: True
     Comment: File /var/www/html/saltbase_install/saltbase_install.sls updated
     Started: 16:34:18.014331
    Duration: 36.796 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
```

These states copy the files required for the functionality of salt on the SaltBase machine. These files are copied to the PXESaltbase server where they will be used as source files during SaltBase builds from the PXE server. These states completed in ~250ms.

```
          ID: saltbase_helper1
```

```
    Function: file.managed
        Name: /var/www/html/saltbase_install/enable_ssh.sh
      Result: True
     Comment: File /var/www/html/saltbase_install/enable_ssh.sh updated
     Started: 16:34:18.051313
    Duration: 30.328 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
          ID: saltbase_helper2
    Function: file.managed
        Name: /var/www/html/saltbase_install/newsalthostname.sh
      Result: True
     Comment: File /var/www/html/saltbase_install/newsalthostname.sh updated
     Started: 16:34:18.081815
    Duration: 37.449 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
          ID: saltbase_helper3
    Function: file.managed
        Name: /var/www/html/saltbase_install/setupnetwork.sh
      Result: True
     Comment: File /var/www/html/saltbase_install/setupnetwork.sh updated
     Started: 16:34:18.119460
    Duration: 43.99 ms
     Changes:
              ----------
              diff:
                  New file
              mode:
                  0644
----------
```

Just as above, these states simply copy the helper scripts to the PXESaltbase machine, and do so in ~112ms.



```
          ID: update_bashrc
    Function: file.managed
        Name: /root/.bashrc
      Result: True
     Comment: File /root/.bashrc updated
     Started: 16:34:18.163726
    Duration: 70.788 ms
     Changes:
              ----------
              diff:
                  ---
                  +++
                  @@ -97,6 +97,23 @@
                   #if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
                   #    . /etc/bash_completion
                   #fi
                  -# START salt blockreplace
                  -# Initial config completed
                  -# END salt blockreplace
                  +
                  +# The following commands fix the nfsroot call in defaults.
Busybox will not resolve the hostname so we are restricted to using the IP
only.
                  +if ( grep eth0 /etc/network/interfaces )
                  +then
                  +     ethname="eth0" #Found eth0
                  +else
                  +     ethname=`ifconfig | grep -w -m 1 ens... | awk '{print
$1;}'` #Derived Ethernet interface name
                  +fi
                  +localpxeip=`/sbin/ifconfig $ethname | grep 'inet addr:' |
cut -d: -f2 | awk '{ print $1 }'`
                  +sed -i "s|nfsroot.*:|nfsroot=$localpxeip:|g"
/var/lib/tftpboot/pxelinux.cfg/default
                  +
                  +# Add the following reminders for the configuration of the
PXE server after an automated salt build
                  +echo ""
                  +echo "********************"
                  +echo "On initial boot, you should perform the following to
complete the PXE Server configuration:"
                  +echo " - Assign a static IP to the PXE server using the
/root/setupnetwork.sh script"
                  +echo " - Change the hostname using the
/root/newsalthostname.sh script"
                  +echo " - Add a pxe.nuvation.com DNS record"
                  +echo "********************"
                  +echo ""
```

The final state, updating the .bashrc file, shows the full diff between the salt managed file and that of the local file. This is again a quick process, completing in ~71ms.

```
Summary for pxesaltbasetest
-------------
Succeeded: 41 (changed=40)
Failed:     0
-------------
Total states run:     41
Total run time: 1586.536 s
```

There are some interesting takeaways from this summary:

- There are no failures, they would have been easily spotted as red text.
- States which precipitate some change are called out next to the succeeded total, this is helpful when applying a highstate multiple times as it will be clear which states in the salt make changes at each iteration.
- This salt formula required quite a long time to complete at more than 26 minutes! However, if the total time required to download the ISO files (~1561 seconds) is subtracted from the whole, the entire configuration of the PXESaltBase server requires only **25 seconds**, of which almost **19 seconds** are spent on package installations and updates. This illustrates not only the incredible performance of salt, but that the execution time can be dramatically cut by locally caching the ISO files and updating the salt accordingly.

Enable the PXESaltBase machine

OK, it's deployed, now what? Well, upon reboot (and login) the following message will be displayed:

```
********************
On initial boot, you should perform the following to complete the PXE Server configuration:
 - Assign a static IP to the PXE server using the /root/setupnetwork.sh script"
 - Change the hostname using the /root/newsalthostname.sh script"
 - Add a pxe.bobbarker.com DNS record"
********************
```

These lines were added to the end of .bashrc to remind the user of the remaining prerequisites for getting the PXESaltBase machine enabled. Once these quick points are completed, the PXESaltBase Server should be able to auto-deploy new SaltBase machines within the target infrastructure.

Finish Line / The Real Start

Completing the build of our Ubuntu + SaltStack + gitfs + PXE infrastructure is really just the beginning. This infrastructure opens up new avenues for rapid development and repeatable configuration. It allows for simplified (re)use of vetted code, and easy deployment to templates, hypervisors, and bare metal alike. Removing the complexity of OS configuration while streamlining the development process unleashes both Developers and IT to focus on bigger challenges.

Looking Forward

With this infrastructure built and fully automated, we can look toward expanding this work by extending options on the PXE server itself as well as looking at complementary systems.

Example: Build a Complementary apt-cache

One of the things that can have a direct impact on SaltBase machines (or all Linux machines for that matter) is the inclusion of an apt-cache server. An apt-cache server acts as a local network cache of updated Linux packages which can be provided to clients at local network speeds. In this section we will build an apt-cache server to complement the PXE server, and in the process provide an example of a SaltBase machine deployment using simple formulas to rapidly expand an infrastructure. This example will skip the manual configuration steps and jump straight into development using salt.

Apt-cache Deployment

The PXE server and SaltBase clients will benefit from the apt-cache, but consideration should be made on where to deploy this server. On one hand, this server can be deployed to a fresh SaltBase machine and stand on its own. On the other, the apt-cache server could be added directly to the existing PXE server. In either case, the configuration and implementation using a salt formula will be the same. If this is to be a standalone server, a new SaltBase machine should be deployed before continuing. If the apt-cache is to be added to the existing PXE server, this machine should be attached to via ssh (or console) before continuing.

Apt-cache Overview

Building an apt-cache is actually quite simple and can be broken down into the following major tasks:

- Install prerequisites (Apache)
- Install apt-cacher package
- Add a configuration file
- Update DNS records
- Point hosts to apt-cache server
- Extra Credit: Configure apache and link to apt-cache status page

Let's take these one at time and develop within the context of a salt formula. First, we'll need to setup our salt environment for a new "myaptcache" salt formula and enable the formula.

```
mkdir /srv/salt/myaptcache
touch /srv/salt/myaptcache/init.sls
echo "    - myaptcache" >> /srv/salt/top.sls
```

Install prerequisites (Apache)

This is can be easily implemented within a single state. Add this state to `init.sls`.

```
aptcache_prereq:
  pkg.installed:
    - refresh: True
    - pkgs:
      - apache2
```

Install apt-cacher package

In much the same way as the prerequisite, we can install the apt-cacher package. Note that the apt-cache service should be changed to run at boot time. Add this state to `init.sls`.

```
# Install the apt-cacher service and ensure it starts at boot
aptcache_install:
  pkg.installed:
    - refresh: True
    - pkgs:
      - apt-cacher
  service.running:
    - name: apt-cacher
    - enable: True
```

Add a configuration file

Next we'll want to build a configuration file and install it to the correct location.

Build the apt-cache configuration file

We'll want to build a basic configuration file for the apt-cache server which includes all fields we need and some we may want to use in the future. Also note that a field called "`admin_email`" exists in the file. We'll want to specify this field from the pillar (via jinja) to ensure this information can be easily configured or updated in the future.

The following content can be saved as `/srv/salt/myatpcache/myaptcache.conf`.

```
### GENERAL ###
cache_dir=/var/cache/apt-cacher
logdir=/var/log/apt-cacher
admin_email={{ pillar.get('ADMIN_EMAIL') }}
daemon_port=3142
group=www-data
user=www-data
offline_mode=0

### UPSTREAM PROXY ###
limit=0

### ACCESS and SECURITY ###
allowed_hosts=*
denied_hosts=
allowed_hosts_6=fec0::/16
denied_hosts_6=
supported_archs = i386, amd64

### HOUSEKEEPING ###
generate_reports=1
clean_cache=1

### INTERNALS ###
debug=0

### EXTRA ###
expire_hours=0
use_proxy=0
use_proxy_auth=0
```

Install the apt-cache server config file

Next we'll build some salt to install this config file into place. This state should manage the file using jinja templating and should restart the apt-cacher service if this config file changes when the state is applied. Note that best practices are to drop this file into `/etc/apt-cacher/conf.d/` instead of directly manipulating the global apt-cache config file (`/etc/apt-cacher/apt-cacher.conf`). Add this state to `init.sls`.

```
# Drop in the apt-cacher config file
aptcache_config:
  file.managed:
    - name: /etc/apt-cacher/conf.d/myaptcache.conf
    - user: root
    - group: root
    - mode: 644
    - source: salt://aptcache/myaptcache.conf
    - template: jinja
  service.running:
    - watch:
      - file: /etc/apt-cacher/conf.d/myaptcache.conf
    - name: apt-cacher
```

Enable the pillar

The pillar needs to be created and have the content added to it for insertion into the config file above.

1. Edit the pillar

   ```
   nano /srv/pillar/myaptcache.sls
   ```

2. Add the variable value to the pillar and save the pillar before exiting nano

.

   ```
   ADMIN_EMAIL: 'bob@bobbarker.com'
   ```

3. Enable the pillar (the top.sls for salt already has the call to myaptcache)

   ```
   cp /srv/salt/top.sls /srv/pillar/top.sls
   ```

Run a highstate to build the apt-cache server

1. Run the highstate

   ```
   salt '*' state.highstate > ~/aptcacheinstall.log
   ```

Update DNS Records

Technically, the configuration file and the salt states above are sufficient to get the apt-cache running. The easiest way to ensure that hosts can access this server is to add a DNS record (regardless of whether the apt-cache was deployed independently or directly to the PXE server).

1. Login to your local DNS server
2. Add a record for the aptcache server. e.g.: `aptcache.bobbarker.com`

Point hosts to apt-cache server

Finally, we'll need to point hosts to the new apt-cache server.

Manually point hosts to aptcache

For hosts which already have a fully configured OS, they can be pointed to the apt-cache server manually. Replace "`aptcache.bobbarker.com`" with the DNS name entered above.

```
echo 'Acquire::http::Proxy "http://aptcache.bobbarker.com:3142";' >
/etc/apt/apt.conf.d/01proxy
```

Add apt-cache server via salt

Enabling the apt-cache can also be completed with a small salt formula on salt enabled hosts.. Create the following files, add an entry to /srv/salt/top/sls, and run a highstate to complete the change. Replace "`aptcache.bobbarker.com`" with the DNS name entered above.

/srv/salt/enableaptcache/01proxy

```
Acquire::http::Proxy
"http://aptcache.bobbarker.com:3142";
```

/srv/salt/enableaptcache/init.sls

```
# Enable the local apt cache
local_apt_cache:
  file.managed:
  - name:
/etc/apt/apt.conf.d/01proxy
  - user: root
  - group: root
  - mode: 644
  - source:
salt://enableaptcache/01proxy
```

Add apt-cache server to SaltBase preseed file

Adding the configuration change to the preseed file on the PXE server will allow all SaltBase machines built using the PXE server to utilize the apt-cache by default.

1. Add / create the `01proxy` file in `/var/www/html/saltbase_install/`
2. Add the following line somewhere in the "late_commard" section at the end of the preseed file(s):

   ```
   in-target wget -P /etc/apt/apt.conf.d/ http://{{
   pillar.get('PXE_DNS_NAME') }}/saltbase_install/01proxy; \
   ```

Extra Credit: Configure apache and link to apt-cache status page

With a little extra effort we can add some polish to the apt-cache server, making it easier to access the status of apt-cache service through apache.

Replace default apache site

Replacing the default apache site will add a few small configuration changes, allowing for ignoring of file types during a folder listing, customizing folder view options, and adding a legitimate admin email address (we'll take advantage of the ADMIN_EMAIL variable defined above). Save the following content to `/srv/salt/myaptcache/000-default.conf`.

```
<VirtualHost *:80>
        ServerAdmin {{ pillar.get('ADMIN_EMAIL') }}
        DocumentRoot /var/www/html

        # List of files to ignore
        IndexIgnore *.preseed

        <Directory />
                Options FollowSymLinks
                AllowOverride None
        </Directory>
        <Directory /var/www/html>
                Options Indexes FollowSymLinks MultiViews
                AllowOverride None
                Order allow,deny
                allow from all
        </Directory>

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
        LogLevel warn
</VirtualHost>
```

This site configuration will need to be enabled on the apt-cache server. If this state changes the apache site then apache should also be reloaded. Add this state to `init.sls`.

```
# Replace the default website config
aptcache_default_site:
  file.managed:
    - name: /etc/apache2/sites-available/000-default.conf
    - user: root
    - group: root
    - mode: 644
    - source: salt://aptcache/000-default.conf
    - template: jinja
  service.running:
    - name: apache2
    - watch:
      - file: /etc/apache2/sites-available/000-default.conf
```

Add apt-cache status pointer

Adding a pointer which redirects directly to the apt-cache status page makes for a more convenient user experience. In this case we'll need to add a variable for the aptcache DNS record to the pillar and insert the corresponding jinja to the pointer file. First, save the following content to `/srv/salt/myaptcache/aptcache_status.html`.

```
<head>
<meta http-equiv="refresh" content="0; URL='http://{{
pillar.get('APTCACHE_DNS') }}:3142/aptcache'" />
</head>
```

Next we'll need to add the variable to the pillar (change the DNS name to match the one created above)

```
echo "APTCACHE_DNS: 'aptcache.bobbarker.com'" >>
/srv/pillar/myaptcache.sls
```

Finally we'll need to add a salt state to deploy the pointer file. Add this state to `init.sls`.

```
# Drop in a pointer to the aptcache status page and
aptcache_pointer:
  file.managed:
    - name: /var/www/aptcache_status.html
    - user: root
    - group: root
    - mode: 644
    - source: salt://aptcache/aptcache_status.html
    - template: jinja
```

Clean out the document root

Lastly, we'll need to clean the `index.html` out of `/var/www/html` which will allow apache to create a folder listing as the default apache view for the apt-cache server. Add this state to `init.sls`.

```
# Ensure the default index.html is wiped out, this should allow
apache to create a folder listing by default.
aptcache_cleanout:
  file.absent:
    - name: /var/www/html/index.html
```

Run a highstate to add features to the apt-cache server

1. Run the highstate

```
salt '*' state.highstate > ~/aptcacheinstall.log
```

apt-cache source

The full source of the apt-cache salt formula and associated files can be found on github: https://github.com/love2scoot/pxesaltbase-formula/tree/master/myaptcache.

apt-cache summary

Leveraging the deployment capacity of the PXE server, the process above helps build an apt-cache server to provide a complementary service within the same infrastructure. This exemplifies the benefits of the SaltBase PXE approach, showing the simplicity and power of scaling services on top of SaltStack enabled hosts.

Expanding on this work

We can now look at expanding on the options of the PXE server. Each of these options will be covered at a high level, exploring briefly the benefits they could potentially offer.

Other Linux Variants

While we have chosen to use Ubuntu as the Linux variant of choice, there are no limitations on booting to or installing other Linux variants from the PXE server. For example, the installation process for CentOS (RHEL) should be a simple sideways port from Ubuntu where a kickstart file replaces the preseed file, and the salt should port over easily (if not identically). It's even possible to install FreeBSD (I know- *not* a Linux variant) from an Ubuntu based PXE server. With tftp, http, and nfs mount options installed to the PXE server by default, most installation methods should be available without the need to dramatically modify the PXE server itself.

Targeted TFTP Boot Options

The PXE menu system offers excellent flexibility in selecting the the desired boot configuration for the host. There may, however, be times when a subset of machines need to be booted using a specific configuration. Using the configuration options built into tftp, single hosts or groups of hosts can be set to directly boot to a specific configuration, circumventing the need to select the same menu option each time. This topic is covered well on the syslinux.org page.

Acronis PXE Boot

With the power to image entire systems from either local or network resources, Acronis can be a powerful addition to the PXE server. Although not officially supported, it appears that some industrious users have managed to get it working.

Windows PE

Windows PE is a small operating system used to install, deploy, and repair Windows. In much the same way that the Ubuntu Live Boot adds utility for Linux machines, Windows PE offers the same benefit for Windows based hosts. Microsoft offers good documentation around Windows PE from a great introductory page to specific documentation for implementing Windows PE on a PXE server.

Where to now?

Looking beyond the PXE server, we can consider other technologies which could provide complementary capabilities and allow for even greater levels of automation.

gitfs Integration with a Local git Server

While beyond the scope of this section, a great way to further streamline development is to integrate git server credentials directly into SaltBase machines produced by the PXE server. This can be realized by pre-populating the **gitfs_remotes** file on the PXE server (by default, this file includes a commented out example including a sample URL, user, password, and root path). All SaltBase machines built using this server will then inherit the ability to seamlessly use the referenced git server as a gitfs source, allowing users to immediately leverage formulas which can be standardized across the organization.

For example, if your organization has a an apt-cache server (like the example above), this configuration can be built as a salt formula (let’s call it `myaptcache.sls`), committed to a private git server, and therefore made available to new SaltBase machines by simply adding

```
    - myaptcache
```

to the top.sls file. When a high state is run, Salt will use the formula directly from the git server. This allows common configuration tasks in an organization to be centralized into a single common source repo and easily leveraged across all SaltBase machines.

pygit2 solution

While it is not a problem to compile from source, it would certainly be easier to install pygit2 (and dependencies) from public package repositories if http support was baked in. In an ideal case, the maintainer of the package would include both ssh and http capabilities in pygit2 in the future. If this comes to pass it will obviate the complexity associated with the compilation solution required at this time.

build git from a formula on a SaltBase machine

If the benefits espoused above for a private git server are attractive, a gitlab-formula is available currently on github. This formula will install gitlab and can be leveraged directly from a SaltBase machine. Alternately a gitolite-formula is available as well as a plain git-formula.

Migrating to a central salt-master

SaltBase machines are self mastered minions. This allows for flexible configuration options for the host, but scaling this infrastructure requires some additional work; enter a central salt-master. Once a central master is built and configured, all Salt enabled hosts can be pointed to this master for centralized, coordinated configuration. In this way, an infrastructure can be easily scaled up with centralized configuration of nodes on a given network.

To enable this feature on SaltBase machines requires (2) changes: 1) the local DNS server requires a host called "salt" (excluding DNS suffix) which points to the salt-master and 2) the "salt" entry in **/etc/hosts** needs to be removed. Without a hosts file entry, machines will reach out to the local DNS server for resolution of the "salt" name whereupon the salt-master IP will be returned and the minion will attempt to contact the salt-master. Once communication between the minions and the central master is established, the salt-master will need to accept the keys for the connected minions. These steps will enable the salt-master to apply configuration changes to minions which match the criteria specified by the salt-master in top.sls.

Layering Containerization

DevOps has fully embraced containerization solutions (like Docker) to streamline the development process. From a developer's standpoint this approach allows for rapid iteration, dramatic scaling, and more. When moving from development to production, a developer may want to output an application as a container. Reflexively, Operations (IT) will need to provide a target for this container on the production server. Current standards often suggest that production environments use a 1:1 correspondence between a container and the underlying kernel as this minimizes the chance that any one container will affect any other in production.

When considering these scenarios, the simple solution is to build a standard containerized configuration using SaltStack. While Docker (or similar) would live directly on top of an OS, its configuration would be applied by salt formula, ensuring consistency across both development and production environments. As a launching point, github hosts a docker-formula.

The Road Ahead

Cresting a hill allows you to look back at all you've accomplished while giving you a greater perspective on those challenges which lie ahead. My hope is that *Getting DevOps off the Ground* helps deliver on the promise of automation, and is a foundation on which further gains in automation and efficiency can be built. Over the years, I have benefitted immensely from the generosity of those who have taken the time to share ideas and solutions on forums, subreddits, and within documentation. I'm hoping that this, at least in some small way, gives back to the community to which I have borrowed so much.

Gratitude

- Thanks to Man-Yee for helping isolate the pygit2 bug
- Thanks to a great group of peers, coworkers, friends, and family who volunteered their time to help review this work
- Thanks to Trevor for suggesting The Phoenix Project, a great work of fictionalized nonfiction.

Appendix A: Detailed Breakdown of Helper Scripts

Added to each SaltBase machine deployed from the PXE server, the (3) helper scripts simplify some of the more common tasks required for initial configuration of deployed hosts. In this appendix, each of these (3) scripts will be broken down to help the reader understand the configuration changes being made in each case.

enable_ssh.sh

Analysis of enable_ssh.sh

By default, ssh access for the root user is not allowed in Ubuntu v14.04+. The enable_ssh.sh script attempts to complete (3) major actions:

- Installation of the openssh-server on the host
- Altering of the openssh-server configuration to allow for ssh access to root
- A restart of the openssh-server service if alterations to the configuration have been made

Additionally, results are reported back to the user, with one of three outcomes:

- No change required
- Enabling of ssh succeeded
- Something went wrong

Some additional points may be relevant here:

- Although the line *allowing* ssh access for the root user is the same between Ubuntu 14.04 and 16.04, the line *restricting* access is different. As such, we first search for an allowed state using `if` and `grep "^PermitRootLogin yes"`. If this is not found, we search for the line with a wildcard using `elif` and `grep "^PermitRootLogin.*"`. We assume if this line is present that it is a restriction line and we replace this entire line using the `sed` command (listed below).
- This script works for Ubuntu versions 14.04 and 16.04, but an additional case using `else` was built into the script such that if both the `if` and `elif` conditionals fail, that a failure state will be returned to the user. This would most likely be due to a change in the openssh-server configuration file and a change to the script would then be necessary in order to enable ssh access for root. This `else` conditional was added to ensure that the script did not silently fail.

Full source of enable_ssh.sh

```
#!/bin/bash

apt-get install -y openssh-server

if ( grep "^PermitRootLogin yes" /etc/ssh/sshd_config ) # Format
with root SSH enabled
then
     echo ""
     echo "No change required: SSH appears to already be enabled
for root"
     echo ""
elif ( grep "^PermitRootLogin.*" /etc/ssh/sshd_config ) # If it is
present at the beginning of the line, but not set to yes, we assume
it is restricted
then
     sed -i "s/^PermitRootLogin.*/PermitRootLogin yes/g"
/etc/ssh/sshd_config
    service ssh restart
     echo ""
     echo "SSH enabled for root"
     echo ""
else # Something went wrong
     echo ""
     echo "There was a problem when attempting to enable SSH access
for the root user"
     echo ""
fi
```

The source of enable_ssh.sh can also be found on github: https://raw.githubusercontent.com/love2scoot/pxesaltbase-formula/master/pxesaltbase/saltbase_install/enable_ssh.sh.

newsalthostname.sh

Analysis of newsalthostname.sh

While changing a hostname on linux is only a matter of manipulating the contents of /etc/hostsname, completing this task on a self-mastered salt minion requires some additional logic:

- The new hostname is applied to /etc/hostname
- The new hostname is applied to /etc/salt/minion_id
- The new hostname is applied to /etc/hosts
- The salt-key associated with the old hostname must be deleted
- The salt-minion must recognize the new minion_id
- The new minion_id (and salt key) must be recognized by the salt-master
- The salt-master must accept the new salt-key
- The networking subsystem must be restarted to associate with the new hostname

Implementing these steps in a script is fairly straightforward:

- The script first gathers the hostname from the user and allows the user to confirm their choice.
- The steps above are then completed using a combination of echo, sed, salt-key, and service commands.
- Results are displayed for the user to verify the hostname change was completed for salt
- A message requesting a reboot for process completions is echoed to the user.

Full source of newsalthostname.sh

```
#!/bin/bash

# Get the new hostname
echo "Enter the new hostname followed by [ENTER]:"
read newhostname

# Confirm the hostname
while true; do
    read -p "Is $newhostname the new hostname you wish to use?
(y/n)" confirm
    case $confirm in
        [Yy]* ) break;;
        [Nn]* ) echo "Aborting..."; exit;;
        * ) echo "Please answer yes or no.";;
    esac
```

```
done

# Apply the hostname including updating the salt minion_id
echo $newhostname > /etc/hostname
echo $newhostname > /etc/salt/minion_id
sed -i "s|127.0.1.1.*|127.0.1.1        $newhostname salt|g" /etc/hosts

# ---Completing the steps for the changes to salt
# First, we need to delete the old key (all keys actually) because it is associated with the old name
salt-key -y -D
# Restarting the minion service will grab the new minion_id
service salt-minion restart
# displaying the list of all keys will force the master to see the new hostname under unaccepted keys
salt-key -L
# We auto-accept the new key and minion name
salt-key -y -A
# We output the list of keys to verify the key was properly accepted
salt-key -L
# We remind the user to reboot the machine to apply the change
echo ""
echo "*** Please reboot this machine to complete the hostname change"
```

The source of newsalthostname.sh can also be found on github: https://raw.githubusercontent.com/love2scoot/pxesaltbase-formula/master/pxesaltbase/saltbase_install/newsalthostname.sh.

setupnetwork.sh

Analysis of setupnetwork.sh

The setupnetwork.sh script attempts to simplify the process of changing the network configuration of an Ubuntu Server OS. It provides the following features:

- a simple wizard which allows the user to easily change between DHCP and static IP configurations
- validation of all IP addresses to ensure they conform with IPv4 address standards
- feedback to the user on relative success or failure of the procedures

Manually changing network configurations on an Ubuntu Server can be a *relatively* simple task, but automating this process to work across different versions of Ubuntu can be a challenge. Starting in Ubuntu v15.10, the networking system began to use "Predictable Network Interface Names". This results in version 14.04 and 16.04 using different naming standards for Ethernet adapters within the `/etc/network.interfaces` file. As a result, additional complexity is required in order to derive the name of the active Ethernet adapter and correctly apply network changes across Ubuntu versions. After an echoed intro, this is the very first task of the script starting at the comment:

```
# Figure out the name of the Ethernet adapter
```

In Step 1, the user is presented with a menu which allows a selection of DHCP or Static IP. If any other input is specified the wizard will report an invalid option and ask again.

- In the case of DHCP, a simple configuration is built using the `basicoutput ()` and `dhcpoutput ()` functions, the config is written to the interfaces file, the Ethernet adapter is restarted, and the script exits with success.
- In the case of Static IP, the wizard then continues to Step 2 and begins to query the user for the required inputs

In Step 2, the user will be asked for the static IP address. The input is sent to the `isipvalid ()` function, where it is checked for compliance with IPv4 address standards. If the IP is not valid, the wizard will report invalid input and repeat the step.

In Step 3, the user will be asked to specify the size of the subnet. A helpful table with key is displayed, allowing the user to select an option by subnet size. The input is checked and if invalid the subnet size will be requested again.

In Step 4, the user will be asked for the IP of the gateway. The input is sent to the `isipvalid ()` function, where it is checked for compliance with IPv4 address standards. If the IP is not valid, the wizard will report invalid input and repeat the step.

In Step 5, the user will specify the DNS search domain. If the user skips input on this step, the wizard will pass a # value for this step so that variable numbering is maintained.

In Step 6a, the user will specify the IP of the primary DNS nameserver. The input is sent to the `isipvalid ()` function, where it is checked for compliance with IPv4 address standards. If the IP is not valid, the wizard will report invalid input and repeat the step.

In Step 6b, the user will specify the IP of the secondary DNS nameserver. If the user skips input on this step, the wizard will pass a # value for this step so that variable numbering is maintained. If the user passes a value, the input is sent to the `isipvalid ()` function, where it is checked for compliance with IPv4 address standards. If the IP is not valid, the wizard will report invalid input and repeat the step.

Once all input is complete, the wizard will echo the configuration as specified by the user and ask for confirmation.

- If the user specified an invalid choice, the wizard will report such and repeat the confirmation request.
- If the user rejects the configuration, the script exits reporting an abort.
- If the user accepts the configuration, the configuration is built using the `basicoutput ()` and `staticoutput ()` functions, the config is written to the interfaces file, the Ethernet adapter is restarted, the network configuration is echoed to the user, and the script exits with:

  ```
  echo "Check the adapter information above.  If the IP has *not*
  changed, you will need to reboot in order to apply the change
  to a static IP"
  ```

Full source of setupnetwork.sh

```
#!/bin/bash

#---CONSTANTS
outputfile="/etc/network/interfaces"

#---FUNCTIONS
basicoutput () # All interface files start with this
{
     echo "# This file describes the network interfaces available
on your system"
     echo "# and how to activate them. For more information, see
interfaces(5)."
     echo ""
     echo "# The loopback network interface"
     echo "auto lo"
     echo "iface lo inet loopback"
     echo ""
     echo "# The primary network interface"
}

dhcpoutput () # Output for DHCP configuration
{
     echo "# DHCP Configuration"
     echo "auto ${ethname}"
     echo "iface ${ethname} inet dhcp"
}

staticoutput () # Output for Static IP configuration
{
     echo "# Static IP in WDC"
     echo "auto ${1}"
     echo "iface ${1} inet static"
     echo "     address ${2}"
     echo "     netmask ${3}"
#    echo "     network "
#    echo "     broadcast "
     echo "     gateway ${4}"
     echo "     dns-search ${5}"
     echo "     dns-nameservers ${6} ${7}"
}

isipvalid () # Check to see if the IP is valid
{
    separator="\." # Make sure to escape the .
    dotcount=$(grep -o "$separator" <<< "$1" | grep -c .) # Check
for extra octets
    valid=`grep -oP
'\b(?:(?:25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9])[.](?:25[
0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9])[.](?:25[0-5]|2[0-4][
0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9])[.](?:25[0-5]|2[0-4][0-9]|1[0-9][
0-9]|[1-9][0-9]|[0-9]))\b' <<< "$1"`
    if [ "$dotcount" -eq 3 ] && [ "$valid" ]
    then
        return 0 # We have a valid IP address
    else
        return 1 # Not a valid IP
    fi
}

#---START

# Intro
echo "This wizard will help you change the network configuration on
your machine"

# Figure out the name of the Ethernet adapter
if ( grep eth0 $outputfile )
then
     ethname="eth0" #Found eth0
else
     ethname=`ifconfig | grep -w -m 1 en.... | awk '{print $1;}'`
#Derived Ethernet interface name
fi

# Configure as Static or DHCP?
echo ""
echo "---STEP 1: STATIC vs DHCP"
while true; do
    echo ""
    echo "How would you like to configure adapter ${ethname}?"
     read -r -p "(S)tatic IP / (D)HCP [S/D] " answer
     case "${answer}" in
           (s* | S*)
                break
                ;;
           (d* | D*)
                basicoutput > $outputfile
                dhcpoutput >> $outputfile
                ifdown ${ethname}
                ifup ${ethname}
                echo ""
                echo "---IP configuration change complete"
                exit 0
                ;;
           (*)
                echo "Unknown option: ${answer}"
                ;;
     esac
done

# OK, configure as static
echo ""
echo "---STEP 2: IP ADDRESS"
while true; do
    echo ""
    read -r -p "Please supply the desired IP address (no leading
zeros): " myip
    echo "Checking IP..."
    isipvalid $myip
    if [ $? -eq 0 ]
    then
        echo "That looks like a valid IP"
        break
    else
        echo "$myip is not a valid IP address" # Something is wrong
with the octets
    fi
done

# We need a subnet now
echo ""
echo "---STEP 3: SUBNET MASK"
while true; do
    echo ""
    echo "-------------------------------------"
    echo "| Bits | Mask            | Hosts    |"
    echo "-------------------------------------"
    echo "| /8   | 255.0.0.0       | 16777214 |"
    echo "| /9   | 255.128.0.0     | 8388606  |"
    echo "| /10  | 255.192.0.0     | 4194302  |"
    echo "| /11  | 255.224.0.0     | 2097150  |"
    echo "| /12  | 255.240.0.0     | 1048574  |"
    echo "| /13  | 255.248.0.0     | 524286   |"
    echo "| /14  | 255.252.0.0     | 262142   |"
    echo "| /15  | 255.254.0.0     | 131070   |"
    echo "| /16  | 255.255.0.0     | 65534    |"
    echo "| /17  | 255.255.128.0   | 32766    |"
    echo "| /18  | 255.255.192.0   | 16382    |"
    echo "| /19  | 255.255.224.0   | 8190     |"
    echo "| /20  | 255.255.240.0   | 4094     |"
    echo "| /21  | 255.255.248.0   | 2046     |"
    echo "| /22  | 255.255.252.0   | 1022     |"
    echo "| /23  | 255.255.254.0   | 510      |"
    echo "| /24  | 255.255.255.0   | 254      |"
    echo "| /25  | 255.255.255.128 | 126      |"
    echo "| /26  | 255.255.255.192 | 62       |"
    echo "| /27  | 255.255.255.224 | 30       |"
    echo "| /28  | 255.255.255.240 | 14       |"
    echo "| /29  | 255.255.255.248 | 6        |"
    echo "| /30  | 255.255.255.252 | 2        |"
    echo "-------------------------------------"
    echo ""
     read -r -p "Please supply the bits of the desired subnet (see
table above): " mysub
    case "${mysub}" in
           (/8 | 8) mymask="255.0.0.0"; break;;
           (/9 | 9) mymask="255.128.0.0"; break;;
           (/10 | 10) mymask="255.192.0.0"; break;;
           (/11 | 11) mymask="255.224.0.0"; break;;
           (/12 | 12) mymask="255.240.0.0"; break;;
           (/13 | 13) mymask="255.248.0.0"; break;;
           (/14 | 14) mymask="255.252.0.0"; break;;
           (/15 | 15) mymask="255.254.0.0"; break;;
           (/16 | 16) mymask="255.255.0.0"; break;;
           (/17 | 17) mymask="255.255.128.0"; break;;
           (/18 | 18) mymask="255.255.192.0"; break;;
           (/19 | 19) mymask="255.255.224.0"; break;;
           (/20 | 20) mymask="255.255.240.0"; break;;
           (/21 | 21) mymask="255.255.248.0"; break;;
           (/22 | 22) mymask="255.255.252.0"; break;;
           (/23 | 23) mymask="255.255.254.0"; break;;
           (/24 | 24) mymask="255.255.255.0"; break;;
           (/25 | 25) mymask="255.255.255.128"; break;;
           (/26 | 26) mymask="255.255.255.192"; break;;
           (/27 | 27) mymask="255.255.255.224"; break;;
           (/28 | 28) mymask="255.255.255.240"; break;;
           (/29 | 29) mymask="255.255.255.248"; break;;
           (/30 | 30) mymask="255.255.255.252"; break;;
           (*) echo "Unknown option: ${mysub}";;
     esac
done

# Get the gateway
echo ""
echo "---STEP 4: GATEWAY"
```

```
while true; do
    echo ""
    read -r -p "Now we'll need the IP Address of the gateway (no
leading zeros): " mygate
    echo "Checking IP..."
    isipvalid $mygate
    if [ $? -eq 0 ]
    then
        echo "That looks like a valid IP"
        break
    else
        echo "$mygate is not a valid IP address" # Something is
wrong with the octets
    fi
done

# Get the dnsdomain
echo ""
echo "---STEP 5: DNS Domain(s)"
echo "(Optional, but suggested)"
echo 'Example: "bobbarker.com"'
echo ""
read -r -p "Please provide a dns search domain: " dnsdomain
if [ -z "$dnsdomain" ]
then
    dnsdomain="#" # Passing a hash to retain the numbering of
passed vales to the static IP function
fi

#Get the DNS Server IP
echo ""
echo "---STEP 6a: DNS NAMESERVER"
while true; do
    echo ""
    read -r -p "We'll need the IP Address of the local DNS Server
(no leading zeros): " dnsnameserver1
    echo "Checking IP..."
    isipvalid $dnsnameserver1
    if [ $? -eq 0 ]
    then
        echo "That looks like a valid IP"
        break
    else
        echo "$dnsnameserver1 is not a valid IP address" #
Something is wrong with the octets
    fi
done
```

```
#Get the DNS Server IP
echo ""
echo "---STEP 6b: DNS NAMESERVER (Secondary)"
while true; do
    echo ""
    read -r -p "Secondary DNS Server IP / Enter to skip (no leading
zeros): " dnsnameserver2
    if [ -z "$dnsnameserver2" ]
    then
        dnsnameserver2="#" # Passing a hash to retain the numbering
of passed vales to the static IP function
        break
    fi
    echo "Checking IP..."
    isipvalid $dnsnameserver2
    if [ $? -eq 0 ]
    then
        echo "That looks like a valid IP"
        break
    else
        echo "$dnsnameserver2 is not a valid IP address" #
Something is wrong with the octets
    fi
done

# Confirm the config
while true; do
    echo ""
    echo "---Validate the Configuration----"
    basicoutput
    staticoutput ${ethname} ${myip} ${mymask} ${mygate}
${dnsdomain} ${dnsnameserver1} ${dnsnameserver2}
    echo "--------------------------------"
    echo ""
    read -r -p "Would you like to apply this configuration? (Y/N):
" pleaseconfirm
     case "${pleaseconfirm}" in
           (y* | Y*)
            # Output Static IP settings to file
            basicoutput > $outputfile
            staticoutput ${ethname} ${myip} ${mymask} ${mygate}
${dnsdomain} ${dnsnameserver1} ${dnsnameserver2} >> $outputfile
            ifdown ${ethname}
            ifup ${ethname}
            ifconfig ${ethname}
            echo ""
            echo "*********"
            echo "Check the adapter information above.  If the IP has *not* changed, you will need to reboot in order to apply the change to a static IP"
            break
                ;;
           (n* | N*)
            echo ""
            echo "---Aborting configuration"
                exit 1
                ;;
           (*)
                echo "Unknown option: ${answer}"
                ;;
     esac
done

echo ""
echo "---IP configuration change complete"
```

The source of setupnetwork.sh can also be found on github: https://raw.githubusercontent.com/love2scoot/pxesaltbase-formula/master/pxesaltbase/saltbase_install/setupnetwork.sh.

Appendix B: Quick SaltStack Primer

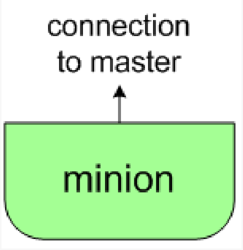
Although plenty of guides on using SaltStack exist, the documentation below provides a simple primer for understanding the basic concepts of the tool.

Masters and Minions

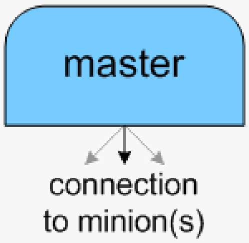
Defining these roles

Configuration of machines using SaltStack follow the association of masters and minions.

- Machines that are being configured are **minions**.
    - Minions are authenticated to their master
    - Minions report back configuration results to their master
    - Minions can only associate with a single master at a time

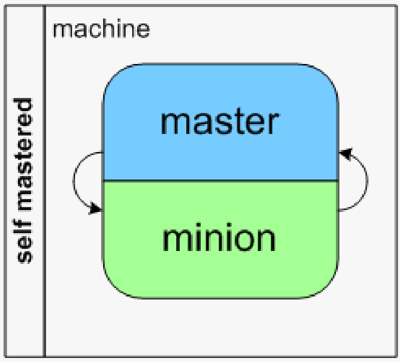


- Machines that are applying the configuration are **masters**
    - Masters initiate a configuration in one or more minions
    - Masters centralize reporting from their associated minions
    - Masters can associate with one or more minions
    - Masters can apply configurations to a subset of their minions through conditional statements based on a wide variety of data. This allows masters to easily apply configurations to a targeted subset of associated minion machines.



Self Mastered

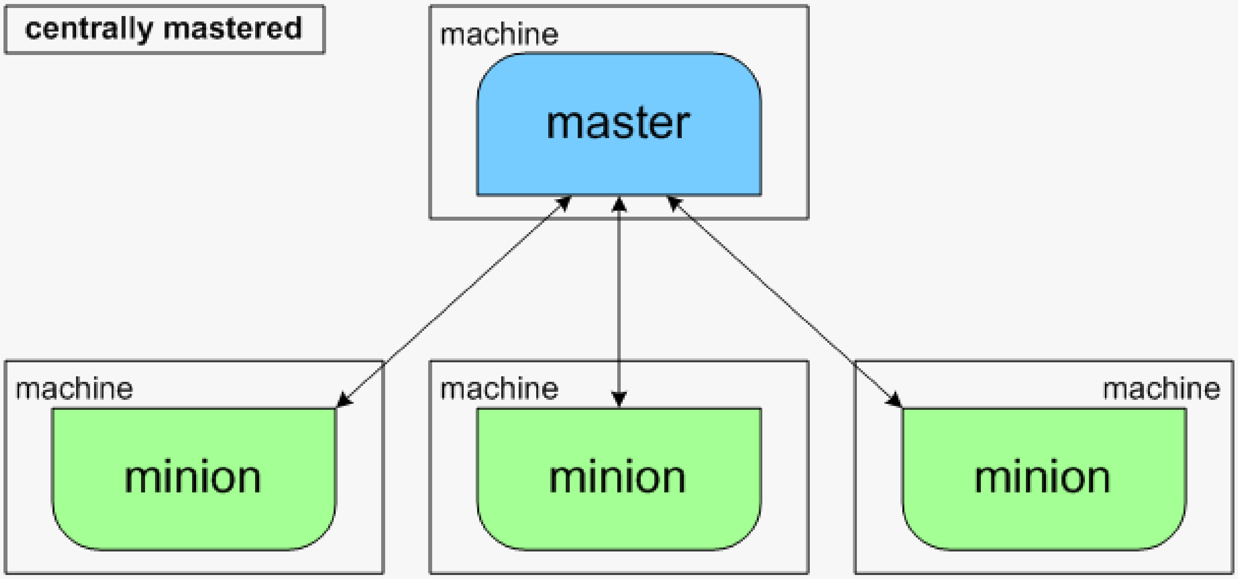
Looking at the roles above, it would be easy to assume that master and minion roles require separate machines, but that is not the case. The simplest approach to developing with salt is to keep both roles on the same machine in a configuration called *self-mastered*. Having the master and minion on the same machine allows for rapid development of configuration code by allowing development without the need to rely on any external sources.



Centrally Mastered

After the initial configuration of a minion is complete, the machine can remain self mastered, or can be associated with a new master. In some cases, the association with a centralized master can be advantageous. If, for example, a pool of identically configured minions are required for a project, having the minions associated with a centralized master will allow them to remain in sync.

Although outside the scope of this document, it is important to note how salt can be leveraged beyond the configurations detailed herein. The documentation from this point describes salt from within the context of a self mastered approach.

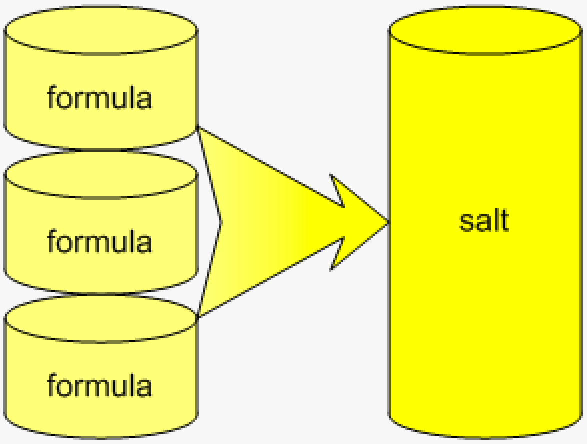


Structures

The source for SaltStack is essentially broken into (2) main structures, Salt and Pillars.

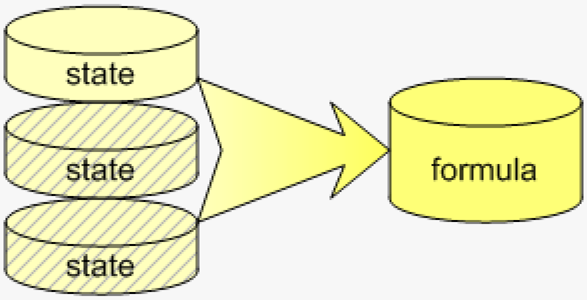
Salt

The source code that describes the changes to apply to the minion is called **salt**. This source is organized into files called formulas.



Formulas

Salt **formulas** are files that contain one or more states to apply to a minion.



Formulas contain one or more changes (or states) that will be applied to the minion. The developer chooses how to build these formulas by segmenting these states into formulas. While this is somewhat arbitrary, it’s a good idea to consider how the code may be maintained going forward when making this determination.

For example, if the goal is to configure a minion as a web server, all states to configure this machine can be collected within a single formula. However, the developer may want to consider segmenting these states up into several different formulas like apache_changes, php_changes, http_content_export, etc. each of which contain a subset of the configuration states.

States

Unlike scripting languages which will run in their entirely on each launch, Salt leverages a concept known as *states*. States allow for a conditional check of the minion before attempting to apply the configuration change. If the state has already been applied, no action is taken and the minion reports back that the state was already correctly configured. If not, application of the state is attempted and the success or failure is reported.

Within a formula, all states must be uniquely named. If, for example, a state was named `file_owner_change:` and another owner change were required within the same formula, the state could simply be built using a different name; for example: `file_owner_again:`. Naming of states inside a formula is arbitrary, but it's probably a good idea to imply some description of the state's functionality in the name.

States are applied to the minion in the order in which they appear in the formula. Exceptions exist, often based on dependencies that may be applied between states.

Salt Examples

For example, let's say we need to change the owner and permissions on a specific file on the minion. In bash this could be represented by the commands:

```
chown root:root /etc/cron.daily/backup
chmod 644 /etc/cron.daily/backup
```

Implemented in salt, we could build a simple formula with a single state called `sample_file_permission:`. This state would leverage the file.managed formula and use input parameters for the path to the file and the desired permissions.

Here's how it might look:

```
# change owner and permissions on backup
sample_file_permission:
  file.managed:
    - name: /etc/cron.daily/backup
    - user: root
    - group: root
    - mode: 644
```

This state will check to ensure that the file exists, has the correct ownership, and the correct permissions assigned. If not, it will attempt to apply changes to the file to bring it into this configuration. Note that both bash commands are represented here in a single state. This source could then be saved as a file called “sample.sls”, and would stand as its own formula.

Let's look at more advanced example:

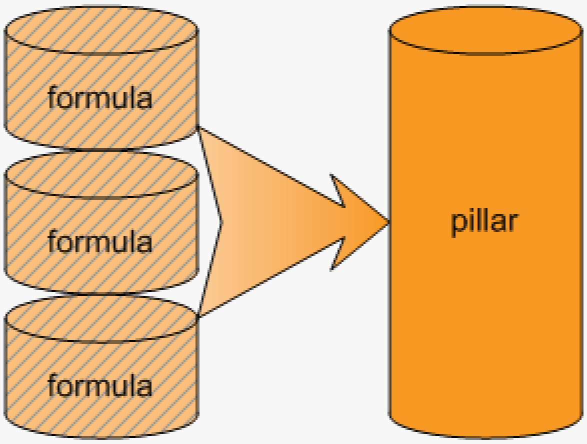
```
# Export a file from SVN
svn_example_export:
  svn.export:
    - name: https://svn.bobbarker.com/SANDBOX/trunk/test.txt
    - target: /tmp/test.txt
    - username: bbarker
    - password: PriceIsWrong!
    - force: true
    - overwrite: true
    - trust: true
```

In this example we can see that salt functionality can extend beyond simple OS changes to include functions such as interfacing with an SVN server. This source can then be saved as a file called `svnexport.sls` as its own formula.

This example also shows one of the common downsides to interpreted code, namely inclusion of secure data within the source. SaltStack makes a provision for this, allowing storage of more sensitive data within a *pillar*.

Pillars

A **pillar** allows for storage of sensitive data within a structure external to our salt. In their most basic form, pillars are simply a list of variables that allow for insertion into the salt when the salt is interpreted. Note that the use of pillars is optional.



Pillar Examples

Following from our previous example, the more secure way of writing the above state would be to separate out the username and password into a pillar and replace it with some simple code that tells salt to go look for the data in the pillar (using Jinja).

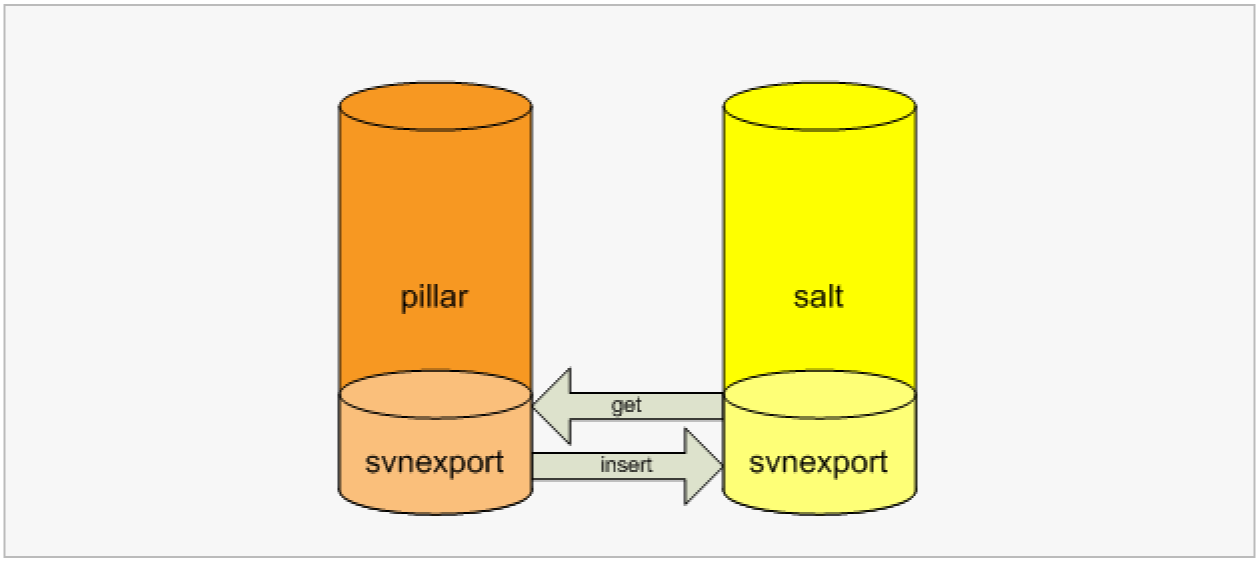
The pillar would look like:

```
SVN_USERNAME: "bbarker"
SVN_PASSWORD: "PriceIsWrong!"
```

The resulting salt would appear as:

```
# Export a file from SVN
svn_example_export:
  svn.export:
    - name: https://svn.bobbarker.com/SANDBOX/trunk/test.txt
    - target: /tmp/test.txt
    - username: {{ pillar.get('SVN_USERNAME') }}
    - password: {{ pillar.get('SVN_PASSWORD') }}
    - force: true
    - overwrite: true
    - trust: true
```

Note that these files would both be saved as “svnexport.sls” (see file topology below for more information)



At this point, the salt could be securely checked into a repository. The pillar, while it *could* be checked into a repository as is, should have the two variables replaced with examples and then committed to the repo.

For example:

```
SVN_USERNAME: "svnuser"
SVN_PASSWORD: "changeme"
```

In this way, secure data is never committed and all variables for a given salt are now centralized within the pillar.

Content

Inside of our salt and pillars is our source, which is segregated into formulas which contain states.

Although the examples above use the general structure:

```
statename:
  function:
    - parameters
```

States can also be named in a more direct manner:

```
parametername:
  function
```

If, for example, we want to touch a file within the minion:

```
create_test:
  file.touch:
    - name: /tmp/test.txt
```

or we could use the short form:

```
/tmp/test.txt:
  file.touch
```

In certain cases this works very well. The only downside here is the state naming limitation noted above. Let's say we wanted to test for file existence within the same formula. We could use:

```
test_exists:
  file.exists:
    - name: /tmp/test.txt
```

but we could *not* name the state:

```
/tmp/test.txt:
  file.exists
```

since we already will have a state named /tmp/test.txt: using the short form above.

File Topology

Now that we have a concept of how the salt and pillar are formatted and interact, we need details on how to store this source within the topology required by SaltStack.

Root Folders

Salt can be configured to use any folder as the container for either salt or pillars, but best practices put these folders in:

```
/srv/salt
/srv/pillar
```

top file

The salt and pillar folders will each need to have a top file. This top file, named top.sls, describes how formulas are applied to minions. For our purposes, we will be taking the most basic example that states are applied to *all* minions, which for a self mastered machine means states apply only to itself.

The top.sls file must be formatted in a very specific way. Here's an example of a top.sls file which includes a single salt file that enables ntp on a minion:

```
base:
  '*':
    - timesync
```

Each salt formula which should be run on the minion must be included on a line in the top.sls file. From our first example above, not all salt files need to have a corresponding pillar file, so it's possible to have an entry in the salt top.sls but not in the pillar top.sls. It is essential, however, to ensure that a top.sls file exists within the pillar folder if there are calls to the pillar within the salt source. If a pillar is not required, the top.sls file in /srv/pillar should be omitted.

naming salt and pillar files

From our sample top.sls above, we see that a formula called "timesync" should be applied to the minion. There are two ways to name formulas within salt:

- as a standalone .sls file
- as a folder

Standalone

For the simplest formulas, we can use a simple .sls file. From the example above, the file would be named timesync.sls and be contained in the root of the salt folder. So the full path would be:

```
/srv/salt/timesync.sls
```

Pillars will always appear as a simple .sls file and have the same name as the salt file / folder. If the timesync.sls salt formula above made calls to the pillar, the corresponding pillar would have the same name, but be located in the root of the pillar folder. So the full path would be:

```
/srv/pillar/timesync.sls
```

Folder

In some cases we will want use a folder for the formula instead of the simple .sls file. This is typically due to the inclusion of external files that should be included along with the salt formulas. In this case the nomenclature is a little different: the folder takes the name of the formula and the salt source is put into a file named init.sls within that folder. So the full path would be:

```
/srv/salt/timesync/init.sls
```

external file inclusion

As mentioned above, there are times when external files should be included along with a formula. This typically manifests as files that should be inserted into the minion when the states are applied.

Using our first example, let's say that `/etc/cron.daily/backup` does not yet exist, but we want to create it on the minion at the time we apply the state. In this case we would use the same `file.managed` function but grab the file from the same folder.

The state could look like:

```
create_backup_file:
  file.managed:
    - name: /etc/cron.daily/backup
    - user: root
    - group: root
    - mode: 644
    - source: salt://sample/backup
```

and the folder would have both the formula and the file. The full paths would be

```
/srv/salt/sample/init.sls
/srv/salt/sample/backup
```

Taking these ideas one step further, it is possible to include the Jinja script snippets *in the external files*. In this case, let's say we want to change how the backup script works, allowing it to use a database name specified in the pillar. The backup file could have a snippet:

```
databasename="{{ pillar.get('MY_DATABASE') }}"
```

The pillar file would then need to have the associated value. The pillar could look like:

```
MY_DATABASE: "SQLTEST"
```

The only change required would be an additional parameter on the file.managed function telling salt to interpret the attached file as containing Jinja. The new state could look like:

```
create_backup_file:
  file.managed:
    - name: /etc/cron.daily/backup
    - user: root
    - group: root
    - mode: 644
    - source: salt://sample/backup
    - template: jinja
```

Note the "template" line. The full paths of all files for this approach would be:

```
/srv/salt/top.sls
/srv/salt/sample/init.sls
/srv/salt/sample/backup
/srv/pillar/top.sls
/srv/pillar/sample.sls
```

Where:

`/srv/salt/top.sls`	Top file with a single entry for sample, the only formula we are applying.
`/srv/salt/sample/init.sls`	The state described above which pulls the file “backup” from the sample folder and writes it to the target path, inserting the value from the pillar specified by the Jinja, applying the ownership and permissions to the file at the same time.
`/srv/salt/sample/backup`	The file which will be written to the target path by the formula. This file contains Jinja, which will have values inserted from the pillar at the time the file is written to the target path by the salt state.
`/srv/pillar/top.sls`	Top file with a single entry for sample, the only pillar we are using.
`/srv/pillar/sample.sls`	The pillar file containing the database value to insert into the backup file when written to the target path

Formatting

Salt requires that content adhere to specific formatting rules. Much like other scripting and programming languages, a small mistake will break the formulas and not allow the states to be applied.

YAML

Salt uses **YAML** which requires specific indent spacing, usage of colon, and usage of dashes. Note in all the examples above:

- Each "level" of states are indented by two additional spaces
- A colon is required to list elements under an line
- A dash with space after it is required for listed parameters

These are by no means the only requirements of salt, but should be sufficient for this document.

Jinja

The **Jinja** code snippets use double curly brackets separated by spaces to encapsulate the code. You can see examples of this above such as

```
databasename="{{ pillar.get('MY_DATABASE') }}"
```

Also note that quotation marks cannot be nested if they are the same kind. If nested quotes are needed, single and double quotes can be nested within one another.

Salt Development Backgrounder

This section touches on code reuse as well as some subtleties around the development of salt.

Leveraging Code Reuse

One of the great benefits of using Salt is the low effort required to leverage the work of others. With the ability to pull salt formulas from varied sources, development can be modularized and easily replicated. However, unless the developer is aware of what is already available, there will be little opportunity to leverage this advantage. A great way to become acquainted with existing formulas is to simply browse repositories. Once you identify a promising formula, simply add it to your top file and gitfs_remotes and give it a try. An example of this approach can be seen in Part I above in Apply a state from a gitfs source.

Understanding Salt Hierarchy

Salt uses a relatively simple structure for execution order and naming conventions. This information is important to cover in order to help avoid scenarios where a potential conflict may occur.

Formula Execution Order

A salt formula is interpreted using the order in which the states appear. For example:

```
# Install my desired packages
install_my_packages:
  pkg.installed:
    - refresh: True
    - pkgs:
      - subversion
      - htop

# Enable the local apt cache
local_apt_cache:
  file.managed:
  - name: /etc/apt/apt.conf.d/01proxy
  - user: root
  - group: root
  - mode: 644
  - source: salt://localaptcache/01proxy
```

```
# Refresh package manager
update_os:
  pkg.uptodate:
  - refresh: true
```

- First, the `install_my_packages` state would run and install subversion and htop
- Second, the `local_apt_cache` state would run, defining the apt cache proxy
- Third, the `update_os` state would run, performing an apt update / upgrade

This execution order is the same every time the salt is interpreted. The only exception to this execution order is when states use requisites, allowing logical dependencies to be built between states. You can see details on this topic in the SaltStack Documentation

State Name Conflicts

No two states can be named the same. For example:

```
# Install my desired packages
install_my_packages:
  pkg.installed:
    - refresh: True
    - pkgs:
      - subversion

# Install more packages
install_my_packages:
  pkg.installed:
    - refresh: True
    - pkgs:
      - htop
```

This would cause an error to be thrown and the salt would not be executed. If, however, the second state was renamed from `install_my_packages` to `install_more_packages` this would resolve the conflict.

Formula Name Conflicts

No two formulas can be named the same. For example, if your local salt folder contained a formula named `mysettings.sls` and this same filename was also contained in a connected gitfs repository, an error to be thrown and the salt would not be executed. It is important to ensure that all formula names are without conflict across the entire state tree (the combined salt across all file sources). This would typically come into play once a new gitfs source is added, whereupon new formulas would be added to the state tree.