D2 Ansible playbooks Documentation

Release 19.10.0

Dave Dittrich

Nov 13, 2019

Contents:

This document (version 19.10.0) describes the D2 Ansible playbooks*[0] (`ansible-dims-playbooks` for short) repository contents.

[0] D2 is a fork of the original DIMS Ansible Playbooks.

CHAPTER 1

Introduction

This chapter documents the D2 DIMS Ansible playbooks*[0] (`ansible-dims-playbooks` for short) repository.

This repository contains the Ansible playbooks and inventory for a development/test environment. It can be extended, enhanced, and customized for a production deployment using the features described in this document.

At a high level, these playbooks assist in composing a *small-scale distributed system* (i.e., a larger system composed of multiple hosts and/or containers that are configured to operate in concert.)

The resulting system supports any/all of the following features:

- Semi-automated provisioning and deployment of Digital Ocean droplets and DNS records using `terraform`.
- Support for SSH host key management allowing `StrictHostKeyChecking` to be left enabled, while avoiding manual host key validation or insertion/deletion.
- A Trident trust group management and communication portal behind an NGINX reverse proxy secured by TLS.
- A Jenkins build server behind an Nginx reverse proxy secured by TLS, with Jenkins CLI secured with SSH.
- Support for Letsencrypt SSL/TLS certificate generation, backup/restoration, renewal-hooks for deploying certificates to non-privileged services, and scheduled certificate renewal maintenance.
- Support for SPF, DKIM, and DMARC in Postfix SMTP email.
- Centralized `rsyslog` logging secured by TLS.
- AMQP (RabbitMQ) message bus for remote procedure call, log distribution, and simple text chat, all secured by TLS.

1.1 Installation Steps

Before diving into the details, it is helpful to understand the high level tasks that must be performed to bootstrap a functional deployment.

[0] D2 is a fork of the original DIMS Ansible Playbooks.

- Install the base operating system for the initial Ansible control host that will be used for configuring the deployment (e.g., on a development laptop or server).
- Set up host playbook and vars files for the Ansible control host.
- Pre-populate artifacts on the Ansible control host for use by virtual machines under Ansible control.
- Instantiate the virtual machines that will be used to provide the selected services and install the base operating system on them, including an `ansible` account with initial password and/or SSH `authorized_keys` files allowing access from the Ansible control host.
- Set up host playbooks, host vars files, and inventory definitions for the selected virtual machines.
- Validate that the Ansible control host is capable of connecting to all of the appropriate hosts defined in the inventory using Ansible *ad-hoc* mode.
- Finish customizing any templates, installed scripts, and secrets (e.g., passwords, certificates) unique to the deployment.

CHAPTER 2

Ansible and Configuration Management

Ansible is an open source tool that can be used to automate system administration tasks related to installation, configuration, account setup, and anything else required to manage system configurations across a large number of computers.

While it is possible to manually install and configure a hand-full of computer systems that do not change very often, this kind of system deployment and system administration quickly becomes a limiting factor. It does not scale very well, for one, and makes it very difficult to change or replicate. You may need to move hardware from one data center to another, requiring reconfiguration of both hosts and dozens of VM guests. You may need to move from one Linux distribution to another. You may wish to add another continuous integration build cluster to support another operating system major/minor version, or a new processor architecture like ARM. Even if the number of hosts is small, having the knowledge of how the systems were built and configured in the head of just one person (who may go on vacation, or permanently leave the project) increases the risk of total disruption of the project in the event of an outage.

Tip: If you are not familiar with Ansible, take some time to look at the Ansible Get Started page, and/or watch the following video series. While they are a little dated now (2015, pre- Ansible 2.x), they cover many useful concepts.

- 19 Minutes With Ansible (Part 1/4), Justin Weissig, sysadmincasts.com, January 13, 2015
- Learning Ansible with Vagrant (Part 2/4), Justin Weissig, sysadmincasts.com, March 19, 2015
- Configuration Management with Ansible (Part 3/4), Justin Weissig, sysadmincasts.com, March 26, 2015
- Zero-downtime Deployment with Ansible (Part 4/4), Justin Weissig, sysadmincasts.com, April 2, 2015

Also highly recommended is to immediately get and read all of Jeff Geerling's book, Ansible for DevOps. This book is more up-to-date in terms of covering Ansible 2.x features and coding style. It will save you countless hours of floundering around and Jeff's Ansible coding style is top quality.

Many more references can be found in Section bestpractices (originally collected at <https://staff.washington.edu/dittrich/home/unix.html#ansible>).

Attention: As of 2018-09-23, these playbooks have been successfully used with Ansible v2.5.5. Occassionally, changes in Ansible result in regression errors. Using a Python virtual environment, and explicitly installing a specific version of the Ansible package with `python -m pip` can work around such breaking changes.

2.1 Ansible fundamentals

Ansible allows you to document all of the steps necessary to perform the required tasks, organize sets of computers on which those steps should be performed, and then allow you to perform those tasks precisely and consistently across all of those hosts. Figure *Ansible Overview* (source: 19 Minutes with Ansible (Part 1/4)) illustrates this process.

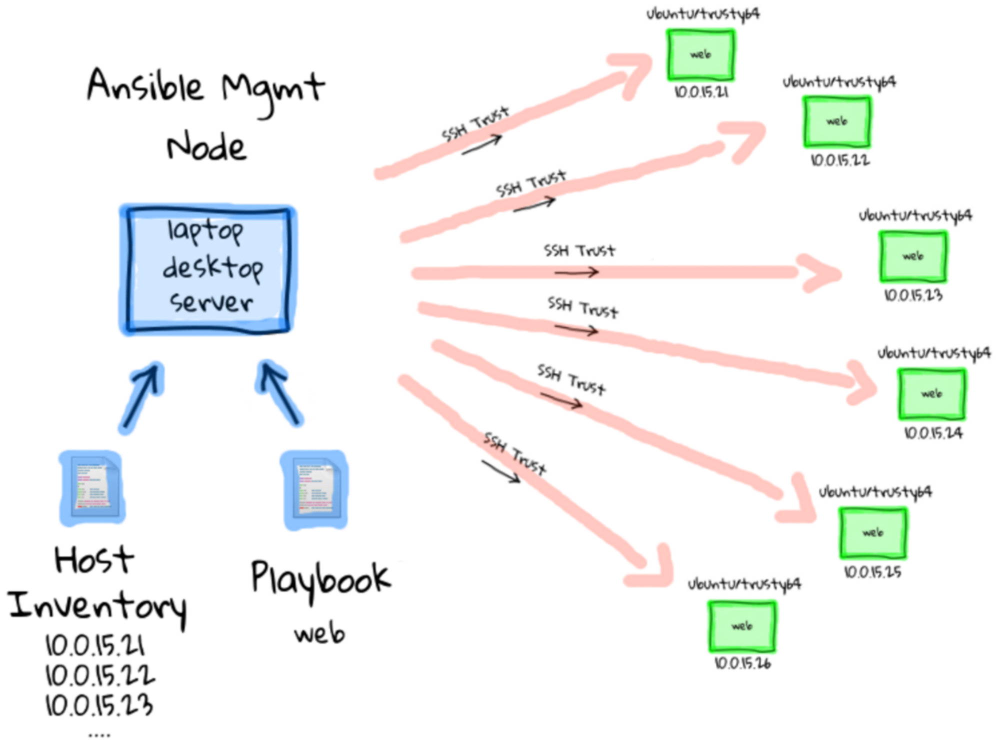


Fig. 1: Ansible Overview

At the center of the left side of Figure *Ansible Overview* is the **Ansible Management Node** (also called by some a **Control** node). This figure depicts the *push model* for using Ansible, where a Control machine holds the playbooks and inventory files necessary to drive Ansible, and that Control machine reaches out to *target* hosts on which the actions take place. Another way to use Ansible is by using a `localhost` connection for the Control machine to also be the Target machine (in which case the actions Ansible performs are done to the same computer on which Ansible is running.)

A set of hosts can be specified with an *inventory*. Ansible supports two styles for static inventories, an INI format style, and a YAML format style. The INI style format is known as a `hosts` file, by default stored in a file named `/etc/ansible/hosts`.

An INI style inventory that implements the example above could look like this:

```
[web]
10.0.15.21
10.0.15.22
10.0.15.23
10.0.15.24
10.0.15.25
10.0.15.26
```

Note: The `-i` flag can be used to specify the inventory file to use, rather than always having to over-write the file `/etc/ansible/hosts`. Alternatively, it can be specified in an `ansible.cfg` file, typically found in `/etc/ansible/ansible.cfg` for the global file. This is covered more in Section *Configuration and Customization of ansible and ansible-playbook*.)

Ansible has two main command line interface programs you can use. The first is just `ansible` and it allows you to run individual modules against a set of hosts (also known as "running a play"). Here is a very simple example of running the `ping` module against every host in the `all` group in the `development` inventory shown above:

```
$ ansible -i $GIT/ansible-playbooks/inventory/development all -m ping
floyd2-p.devops.develop | success >> {
    "changed": false,
    "ping": "pong"
}

hub.devops.develop | success >> {
    "changed": false,
    "ping": "pong"
}

u12-dev-svr-1.devops.develop | success >> {
    "changed": false,
    "ping": "pong"
}

linda-vm1.devops.develop | success >> {
    "changed": false,
    "ping": "pong"
}

u12-dev-ws-1.devops.develop | success >> {
    "changed": false,
    "ping": "pong"
}
```

Using the `command` module, and passing in arguments, you can run arbitrary commands on hosts as a form of distributed SSH:

```
$ ansible -i $GIT/ansible-playbooks/inventory/development all -m command -a /usr/bin/
↪uptime
floyd2-p.devops.develop | success | rc=0 >>
 01:02:52 up 22 days,  7:27,  1 user,  load average: 0.04, 0.12, 1.11

u12-dev-ws-1.devops.develop | success | rc=0 >>
 01:02:52 up 148 days, 14:58,  1 user,  load average: 0.00, 0.01, 0.05

u12-dev-svr-1.devops.develop | success | rc=0 >>
 01:02:45 up 144 days, 17:53,  1 user,  load average: 0.03, 0.05, 0.05

hub.devops.develop | success | rc=0 >>
 09:02:52 up 130 days, 15:14,  1 user,  load average: 0.00, 0.01, 0.05

linda-vm1.devops.develop | success | rc=0 >>
 01:02:53 up 148 days, 14:58,  1 user,  load average: 0.00, 0.01, 0.05
```

The other principal command line program is `ansible-playbook`, which is used to run more complex playbooks made up of multiple sequentially organized plays with all kinds of complex logic and other organizational techniques to manage complex processes. Examples of writing and running playbooks are found in the rest of this document.

Note: Ansible also has a Python API that can be used to embed Ansible functionality into other programs, or to write your own modules to perform tasks. This is explained in the video Alejandro Guirao Rodríguez - Extending and embedding Ansible with Python from EuroPython 2015.

Caution: Always remember that Ansible is used in a distributed system manner, meaning that it has two execution contexts:

(1) it runs with the chosen Python interpreter on the **control** host, which creates Python code that is then

(2) copied to and executed within the context of the **target** host.

Take another look at Figure *Ansible Overview* and realize that the arrows pointing away from the blue node (the control host) to the many green nodes (the targets) implicitly show this context switch.

This has ramifications for targets that run operating systems like CoreOS (that don't have Python installed, and don't have a package manager), and for use of modules like `apt` that call Python libraries to use operating system specific package managers like APT from within Ansible's Python code.

Since the DIMS project uses Python virtual environments to isolate the Python interpreter used by developers from the interpreter used by the system (to avoid breaking the system), this means by definition there are multiple Python interpreters on DIMS hosts. This requires that pay **very close attention** to configuration settings that affect the Python interpreter used by Ansible and consciously do things (and test the results of changes carefully to know when a change breaks something in Ansible.) The result of changes the Python interpreter used by Ansible can be random failures with cryptic error messages like these:

```
Traceback (most recent call last):
  File \"/home/core/.ansible/tmp/ansible-tmp-1462413293.33-173671562381843/file\",␣
↪line 114, in <module>
    exitcode = invoke_module(module, zipped_mod, ZIPLOADER_PARAMS)
  File \"/home/core/.ansible/tmp/ansible-tmp-1462413293.33-173671562381843/file\",␣
↪line 28, in invoke_module
    p = subprocess.Popen(['/opt/bin/python', module], env=os.environ, shell=False,␣
↪stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE)
  File \"/usr/lib/python2.7/subprocess.py\", line 710, in __init__
    errread, errwrite)
  File \"/usr/lib/python2.7/subprocess.py\", line 1335, in _execute_child
    raise child_exception
OSError: [Errno 2] No such file or directory
```

```
msg: Could not import python modules: apt, apt_pkg. Please install python-apt␣
↪package.
```

Both of these messages are due to the Python interpreter being used by Ansible on the **target** end being set to a non-system Python interpreter that does not have the necessary libraries or programs that Ansible needs. In the second case, commenters on blogs may say, "But I installed `python-apt` and I still get this message. Why?" Yes, you may have installed the `python-apt` package like it says, but it was installed into the **system** Python interpreter, which is **not** the one that Ansible is using if `ansible_python_interpreter` or `$PATH` would cause Ansible to use a different one!

2.2 Variables

Note: As of the release of this repository, the DIMS project has adopted Ansible 2.x and switched to using the little-documented (but much more powerful) YAML style inventory. This will be described in more detail elsewhere.

Ansible playbooks are general rules and steps for performing actions. These actions can be selected using logic ("If this is Redhat, do `A`, but if it is Ubuntu, do `B`"), or by using Jinja templating to apply variables to a generic template file, resulting in specific contents customized for a given host.

Some of these variables (known as "facts") are set by Ansible when it first starts to run on a target host, while others are defined in files that accompany playbooks and inventories. You can see the dictionary of `ansible_facts` for a given system using Ansible's `setup` module:

```
$ ansible -m setup localhost -c local
localhost | success >> {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.17.0.1",
            "10.88.88.5",
            "192.168.0.100",
            "10.86.86.7"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::d253:49ff:fed7:9ebd"
        ],
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "01/29/2015",
        "ansible_bios_version": "A13",
        "ansible_cmdline": {
            "BOOT_IMAGE": "/vmlinuz-3.16.0-30-generic",
            "quiet": true,
            "ro": true,
            "root": "/dev/mapper/hostname_vg-root_lv",
            "splash": true,
            "vt.handoff": "7"
        },
        "ansible_date_time": {
            "date": "2016-03-10",
            "day": "10",
            "epoch": "1457653607",
            "hour": "15",
            "iso8601": "2016-03-10T23:46:47Z",
            "iso8601_micro": "2016-03-10T23:46:47.246903Z",
            "minute": "46",
            "month": "03",
            "second": "47",
            "time": "15:46:47",
            "tz": "PST",
            "tz_offset": "-0800",
            "weekday": "Thursday",
            "year": "2016"
        },
        "ansible_default_ipv4": {
            "address": "192.168.0.100",
            "alias": "wlan0",
            "gateway": "192.168.0.1",
            "interface": "wlan0",
            "macaddress": "d0:53:49:d7:9e:bd",
```

(continues on next page)

(continued from previous page)

```
            "mtu": 1500,
            "netmask": "255.255.255.0",
            "network": "192.168.0.0",
            "type": "ether"
        },
        "ansible_default_ipv6": {},
        "ansible_devices": {
            "sda": {
                "holders": [],
                "host": "SATA controller: Intel Corporation 8 Series...",
                "model": "ST1000LM014-1EJ1",
                "partitions": {
                    "sda1": {
                        "sectors": "997376",
                        "sectorsize": 512,
                        "size": "487.00 MB",
                        "start": "2048"
                    },
                    "sda2": {
                        "sectors": "2",
                        "sectorsize": 512,
                        "size": "1.00 KB",
                        "start": "1001470"
                    },
                    "sda5": {
                        "sectors": "1952522240",
                        "sectorsize": 512,
                        "size": "931.04 GB",
                        "start": "1001472"
                    }
                },
                "removable": "0",
                "rotational": "1",
                "scheduler_mode": "deadline",
                "sectors": "1953525168",
                "sectorsize": "4096",
                "size": "7.28 TB",
                "support_discard": "0",
                "vendor": "ATA"
            },
            "sr0": {
                "holders": [],
                "host": "SATA controller: Intel Corporation 8 Series...",
                "model": "DVD-ROM SU-108GB",
                "partitions": {},
                "removable": "1",
                "rotational": "1",
                "scheduler_mode": "deadline",
                "sectors": "2097151",
                "sectorsize": "512",
                "size": "1024.00 MB",
                "support_discard": "0",
                "vendor": "TSSTcorp"
            }
        },
        "ansible_distribution": "Ubuntu",
        "ansible_distribution_major_version": "14",
```

(continues on next page)

(continued from previous page)

```
        "ansible_distribution_release": "trusty",
        "ansible_distribution_version": "14.04",
        "ansible_docker0": {
            "active": false,
            "device": "docker0",
            "id": "8000.0242a37d17a7",
            "interfaces": [],
            "ipv4": {
                "address": "172.17.0.1",
                "netmask": "255.255.0.0",
                "network": "172.17.0.0"
            },
            "macaddress": "02:42:a3:7d:17:a7",
            "mtu": 1500,
            "promisc": false,
            "stp": false,
            "type": "bridge"
        },
        "ansible_domain": "",
        "ansible_env": {
            "BASE": "bash",
            "BYOBU_ACCENT": "#75507B",
            "BYOBU_BACKEND": "tmux",
            "BYOBU_CHARMAP": "UTF-8",
            "BYOBU_CONFIG_DIR": "/home/dittrich/.byobu",
            "BYOBU_DARK": "#333333",
            "BYOBU_DATE": "%Y-%m-%d ",
            "BYOBU_DISTRO": "Ubuntu",
            "BYOBU_HIGHLIGHT": "#DD4814",
            "BYOBU_LIGHT": "#EEEEEE",
            "BYOBU_PAGER": "sensible-pager",
            "BYOBU_PREFIX": "/usr",
            "BYOBU_PYTHON": "python3",
            "BYOBU_READLINK": "readlink",
            "BYOBU_RUN_DIR": "/dev/shm/byobu-dittrich-0R38I1Mb",
            "BYOBU_SED": "sed",
            "BYOBU_TIME": "%H:%M:%S",
            "BYOBU_TTY": "/dev/pts/24",
            "BYOBU_ULIMIT": "ulimit",
            "BYOBU_WINDOW_NAME": "-",
            "CFG": "/opt/dims/nas/scd",
            "CLUTTER_IM_MODULE": "xim",
            "COLORTERM": "gnome-terminal",
            "COMMAND": "",
            "COMPIZ_BIN_PATH": "/usr/bin/",
            "COMPIZ_CONFIG_PROFILE": "ubuntu",
            "CONSUL_LEADER": "10.142.29.116",
            "DBUS_SESSION_BUS_ADDRESS": "unix:abstract=/tmp/dbus-sYbG5zmdUA",
            "DEBUG": "0",
            "DEFAULTS_PATH": "/usr/share/gconf/ubuntu.default.path",
            "DESKTOP_SESSION": "ubuntu",
            "DIMS": "/opt/dims",
            "DIMS_REV": "unspecified",
            "DIMS_VERSION": "1.6.129 (dims-ci-utils)",
            "DISPLAY": ":0",
            "GDMSESSION": "ubuntu",
            "GDM_LANG": "en_US",
```

(continues on next page)

(continued from previous page)

```
            "GIT": "/home/dittrich/dims/git",
            "GNOME_DESKTOP_SESSION_ID": "this-is-deprecated",
            "GNOME_KEYRING_CONTROL": "/run/user/1004/keyring-7kI0rA",
            "GNOME_KEYRING_PID": "2524",
            "GPG_AGENT_INFO": "/run/user/1004/keyring-7kI0rA/gpg:0:1",
            "GTK_IM_MODULE": "ibus",
            "GTK_MODULES": "overlay-scrollbar:unity-gtk-module",
            "HOME": "/home/dittrich",
            "IM_CONFIG_PHASE": "1",
            "INSTANCE": "",
            "JOB": "dbus",
            "LANG": "C",
            "LANGUAGE": "en_US",
            "LC_CTYPE": "C",
            "LESSCLOSE": "/usr/bin/lesspipe %s %s",
            "LESSOPEN": "| /usr/bin/lesspipe %s",
            "LOGNAME": "dittrich",
            "LS_COLORS": "rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:a....",
            "MANDATORY_PATH": "/usr/share/gconf/ubuntu.mandatory.path",
            "NAS": "/opt/dims/nas",
            "OLDPWD": "/home/dittrich",
            "OS": "Linux",
            "PATH": "/home/dittrich/dims/envs/dimsenv/bin:/home/di...",
            "PROGRAM": "/bin/bash",
            "PROJECT_HOME": "/home/dittrich/dims/devel",
            "PS1": "\\[\\033[1;34m\\][dimsenv]\\[\\e[0m\\] \\[\\03...",
            "PWD": "/vm/vagrant-run-devserver",
            "QT4_IM_MODULE": "xim",
            "QT_IM_MODULE": "ibus",
            "QT_QPA_PLATFORMTHEME": "appmenu-qt5",
            "RECIPIENTS": "dims-devops@...",
            "SELINUX_INIT": "YES",
            "SESSION": "ubuntu",
            "SESSIONTYPE": "gnome-session",
            "SESSION_MANAGER": "local/dimsdemo1:@/tmp/.ICE-unix/27...",
            "SHELL": "/bin/bash",
            "SHLVL": "3",
            "SSH_AUTH_SOCK": "/home/dittrich/.byobu/.ssh-agent",
            "STAT": "stat",
            "TERM": "screen-256color",
            "TEXTDOMAIN": "im-config",
            "TEXTDOMAINDIR": "/usr/share/locale/",
            "TMUX": "/tmp/tmux-1004/default,3276,1",
            "TMUX_PANE": "%16",
            "UPSTART_SESSION": "unix:abstract=/com/ubuntu/upstart-s...",
            "USER": "dittrich",
            "VERBOSE": "0",
            "VIRTUALENVWRAPPER_HOOK_DIR": "/home/dittrich/dims/envs",
            "VIRTUALENVWRAPPER_PROJECT_FILENAME": ".project",
            "VIRTUALENVWRAPPER_PYTHON": "/home/dittrich/dims/bin/python",
            "VIRTUALENVWRAPPER_SCRIPT": "/home/dittrich/dims/bin/vir...",
            "VIRTUALENVWRAPPER_WORKON_CD": "1",
            "VIRTUAL_ENV": "/home/dittrich/dims/envs/dimsenv",
            "VTE_VERSION": "3409",
            "WINDOWID": "23068683",
            "WORKON_HOME": "/home/dittrich/dims/envs",
            "XAUTHORITY": "/home/dittrich/.Xauthority",
```

(continues on next page)

(continued from previous page)

```
            "XDG_CONFIG_DIRS": "/etc/xdg/xdg-ubuntu:/usr/share/upstar...",
            "XDG_CURRENT_DESKTOP": "Unity",
            "XDG_DATA_DIRS": "/usr/share/ubuntu:/usr/share/gnome:/usr...",
            "XDG_GREETER_DATA_DIR": "/var/lib/lightdm-data/dittrich",
            "XDG_MENU_PREFIX": "gnome-",
            "XDG_RUNTIME_DIR": "/run/user/1004",
            "XDG_SEAT": "seat0",
            "XDG_SEAT_PATH": "/org/freedesktop/DisplayManager/Seat0",
            "XDG_SESSION_ID": "c2",
            "XDG_SESSION_PATH": "/org/freedesktop/DisplayManager/Session0",
            "XDG_VTNR": "7",
            "XMODIFIERS": "@im=ibus",
            "_": "/home/dittrich/dims/envs/dimsenv/bin/ansible"
        },
        "ansible_eth0": {
            "active": false,
            "device": "eth0",
            "macaddress": "34:e6:d7:72:0d:b0",
            "module": "e1000e",
            "mtu": 1500,
            "promisc": false,
            "type": "ether"
        },
        "ansible_form_factor": "Laptop",
        "ansible_fqdn": "dimsdemo1",
        "ansible_hostname": "dimsdemo1",
        "ansible_interfaces": [
            "docker0",
            "tun88",
            "lo",
            "tun0",
            "wlan0",
            "vboxnet2",
            "vboxnet0",
            "vboxnet1",
            "eth0"
        ],
        "ansible_kernel": "3.16.0-30-generic",
        "ansible_lo": {
            "active": true,
            "device": "lo",
            "ipv4": {
                "address": "127.0.0.1",
                "netmask": "255.0.0.0",
                "network": "127.0.0.0"
            },
            "ipv6": [
                {
                    "address": "::1",
                    "prefix": "128",
                    "scope": "host"
                }
            ],
            "mtu": 65536,
            "promisc": false,
            "type": "loopback"
        },
```

(continues on next page)

(continued from previous page)

```
        "ansible_lsb": {
            "codename": "trusty",
            "description": "Ubuntu 14.04.3 LTS",
            "id": "Ubuntu",
            "major_release": "14",
            "release": "14.04"
        },
        "ansible_machine": "x86_64",
        "ansible_memfree_mb": 2261,
        "ansible_memtotal_mb": 15988,
        "ansible_mounts": [
            {
                "device": "/dev/mapper/hostname_vg-root_lv",
                "fstype": "ext4",
                "mount": "/",
                "options": "rw,errors=remount-ro",
                "size_available": 859396513792,
                "size_total": 982859030528
            },
            {
                "device": "/dev/sda1",
                "fstype": "ext3",
                "mount": "/boot",
                "options": "rw",
                "size_available": 419035136,
                "size_total": 486123520
            }
        ],
        "ansible_nodename": "dimsdemo1",
        "ansible_os_family": "Debian",
        "ansible_pkg_mgr": "apt",
        "ansible_processor": [
            "Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz",
            "Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz",
            "Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz",
            "Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz",
            "Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz",
            "Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz",
            "Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz",
            "Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz"
        ],
        "ansible_processor_cores": 4,
        "ansible_processor_count": 1,
        "ansible_processor_threads_per_core": 2,
        "ansible_processor_vcpus": 8,
        "ansible_product_name": "Precision M4800",
        "ansible_product_serial": "NA",
        "ansible_product_uuid": "NA",
        "ansible_product_version": "01",
        "ansible_python_version": "2.7.6",
        "ansible_selinux": false,
        "ansible_ssh_host_key_dsa_public": "AAAA...==",
        "ansible_ssh_host_key_ecdsa_public": "AA...==",
        "ansible_ssh_host_key_rsa_public": "AAAA...",
        "ansible_swapfree_mb": 975,
        "ansible_swaptotal_mb": 975,
        "ansible_system": "Linux",
```

(continues on next page)

(continued from previous page)

```
        "ansible_system_vendor": "Dell Inc.",
        "ansible_tun0": {
            "active": true,
            "device": "tun0",
            "ipv4": {
                "address": "10.86.86.7",
                "netmask": "255.255.255.0",
                "network": "10.86.86.0"
            },
            "mtu": 1500,
            "promisc": false
        },
        "ansible_tun88": {
            "active": true,
            "device": "tun88",
            "ipv4": {
                "address": "10.88.88.5",
                "netmask": "255.255.255.0",
                "network": "10.88.88.0"
            },
            "mtu": 1500,
            "promisc": false
        },
        "ansible_user_id": "dittrich",
        "ansible_userspace_architecture": "x86_64",
        "ansible_userspace_bits": "64",
        "ansible_vboxnet0": {
            "active": false,
            "device": "vboxnet0",
            "macaddress": "0a:00:27:00:00:00",
            "mtu": 1500,
            "promisc": false,
            "type": "ether"
        },
        "ansible_vboxnet1": {
            "active": false,
            "device": "vboxnet1",
            "macaddress": "0a:00:27:00:00:01",
            "mtu": 1500,
            "promisc": false,
            "type": "ether"
        },
        "ansible_vboxnet2": {
            "active": false,
            "device": "vboxnet2",
            "macaddress": "0a:00:27:00:00:02",
            "mtu": 1500,
            "promisc": false,
            "type": "ether"
        },
        "ansible_virtualization_role": "host",
        "ansible_virtualization_type": "kvm",
        "ansible_wlan0": {
            "active": true,
            "device": "wlan0",
            "ipv4": {
                "address": "192.168.0.100",
```

(continues on next page)

(continued from previous page)

```
                "netmask": "255.255.255.0",
                "network": "192.168.0.0"
            },
            "ipv6": [
                {
                    "address": "fe80::d253:49ff:fed7:9ebd",
                    "prefix": "64",
                    "scope": "link"
                }
            ],
            "macaddress": "d0:53:49:d7:9e:bd",
            "module": "wl",
            "mtu": 1500,
            "promisc": false,
            "type": "ether"
        },
        "module_setup": true
    },
    "changed": false
}
```

Other variables are added from variables files found in `defaults/` and `vars/` directories in a role, from `group_vars/`, from `host_vars/`, from `vars` listed in a playbook, and from the command line with the `-e` flag.

You can run playbooks using the `ansible-playbook` command directly, by using a DIMS wrapper script (`dims.ansible-playbook`) which allows you to run playbooks, tasks, or roles by name, or via the `dimscli` Python CLI program.

> **Caution:** As a general rule, interrupting `ansible-playbook` with CTRL-C in the middle of a playbook run is a Bad Idea. The reason for this is that some playbooks will disable a service temporarily and notify a handler to restart the service at the end of the playbook run, or may successfully change only some configuration files (leaving the ones that would have been changed had the CTRL-C not been issued), either of which can leave the system in an inconsistent and/or potentially inoperable state.
>
> It is best to test playbooks on Vagrant hosts that are not critical if they are accidentally rendered inoperable rather than getting into an emergency debugging situation with a "production" server. If testing with a "live" system, having an active SSH terminal session as a fallback for local access helps, but not always. Be aware of this risk and act accordingly!

2.3 Configuration and Customization of `ansible` and `ansible-playbook`

Like any good Unix program, you can use a global or local *Configuration file* to customize default settings and/or program behavior. Ansible provides the following alternatives:

- `ANSIBLE_CONFIG` (an environment variable)
- `ansible.cfg` (in the current directory)
- `.ansible.cfg` (in the home directory)
- `/etc/ansible/ansible.cfg`

There are many reasons why this configuration customization is useful. The following subsections describe some.

Caution: Keep in mind that one or more of these configuration files may exist on a host causing Ansible to potentially behave differently than expected between different accounts, different systems, etc. If something appears to not work the way you expected, look for these files and see how they are set, add extra levels of verbosity with additional `-v` flags, or otherwise check how Ansible is being configured to work within scripts that run `ansible` or `ansible-playbook`.

To determine which one of these files might be used in a given working directory, you can use the following loop to show which file or files may exist:

```
$ pwd
/home/dittrich/dims/git/ansible-playbooks
$ for f in $ANSIBLE_CONFIG ansible.cfg ~/.ansible.cfg /etc/ansible/ansible.cfg; \
do [ -f $f ] && echo $f exists; done
/etc/ansible/ansible.cfg exists

$ cp /etc/ansible.cfg myconfig.cfg
$ vi myconfig.cfg
  [ ... ]
$ ANSIBLE_CONFIG=myconfig.cfg
$ for f in $ANSIBLE_CONFIG ansible.cfg ~/.ansible.cfg /etc/ansible/ansible.cfg; \
do [ -f $f ] && echo $f exists; done
myconfig.cfg exists
/etc/ansible/ansible.cfg exists
```

2.3.1 Controlling account, SSH port, etc.

There are several parameters that affect SSH connections and the number of simultaneous forked connections useful for accelerating parallel execution of Ansible tasks across a hosts in an inventory. In the example here, we set the number of forks to `10`, the default sudo user to `root`, the default SSH port to `8422` and the transport mechanism to `smart`:

```
# config file for ansible -- http://ansible.com/
# ==============================================

# nearly all parameters can be overridden in ansible-playbook
# or with command line flags. ansible will read ANSIBLE_CONFIG,
# ansible.cfg in the current working directory, .ansible.cfg in
# the home directory or /etc/ansible/ansible.cfg, whichever it
# finds first

[defaults]

# some basic default values...

#hostfile       = /etc/ansible/hosts
#library        = /usr/share/ansible
#remote_tmp     = $HOME/.ansible/tmp
#pattern        = *
forks          = 10
#poll_interval  = 15
sudo_user      = root
#ask_sudo_pass = True
#ask_pass      = True
```

(continues on next page)

(continued from previous page)

```
transport       = smart
remote_port     = 8422
module_lang     = C
 . . .
```

2.4 Lessons from the Fedora Project Ansible Playbooks

One of the better models identified during the second year of the DIMS Project was the Fedora Project's public ansible.git repo. The layout of their repo is described in a README file.

The Fedora Project puts very little in individual host_vars files to store minimal host-specific settings for use in playbooks. This will be examined here in the context of generating `iptables` rules files from Jinja templates.

Note: You can view all of the files in sequence using a Bash `for` loop, as follows:

```
$ cd ~/git/fedora-ansible/
$ for f in inventory/host_vars/*
> do
> echo $f
> echo "===================="
> cat $f
> echo "===================="
> echo ""
> done | less
```

The output looks like this:

```
  inventory/host_vars/aarch64-02a.arm.fedoraproject.org
  ====================
  fas_client_groups: sysadmin-noc,sysadmin-releng

  kojipkgs_url: armpkgs.fedoraproject.org
  kojihub_url: arm.koji.fedoraproject.org/kojihub
  kojihub_scheme: https
  eth0_ip: 10.5.78.75
  gw: 10.5.78.254

  koji_server_url: "http://arm.koji.fedoraproject.org/kojihub"
  koji_weburl: "http://arm.koji.fedoraproject.org/koji"
  koji_topurl: "http://armpkgs.fedoraproject.org/"

  nfs_mount_opts: rw,hard,bg,intr,noatime,nodev,nosuid,nfsvers=3,rsize=32768,
↪wsize=32768

  fedmsg_certs:
  - service: releng
    owner: root
    group: sysadmin-releng
    can_send:
    # pungi-koji stuff (ask dgilmore)
    - pungi.compose.phase.start
    - pungi.compose.phase.stop
    - pungi.compose.status.change
```

(continues on next page)

(continued from previous page)

```
    - pungi.compose.createiso.targets
    - pungi.compose.createiso.imagefail
    - pungi.compose.createiso.imagedone

  ====================

  inventory/host_vars/aarch64-03a.arm.fedoraproject.org
  ====================
  ---
  eth0_ip: 10.5.78.80
  ====================

  inventory/host_vars/aarch64-04a.arm.fedoraproject.org
  ====================
  ---
  eth0_ip: 10.5.78.85
  ====================

  ...

..
```

Let’s look at the file noc01.phx2.fedoraproject.org, specifically the blocks at lines 12-18.

Note: The custom_rules array in this example was split into separate lines here for better readability, as is found in other files such as db-fas01.phx2.fedoraproject.org. It is a single line in the original file (which is perfectly acceptable, though more difficult to read in a limited-column environment such as this documenation. The desire here was to show a file with all three of tcp_ports, udp_ports, and custom_rules variables, hence the cosmetic alteration.

```
1  ---
2  nm: 255.255.255.0
3  gw: 10.5.126.254
4  dns: 10.5.126.21
5
6  ks_url: http://10.5.126.23/repo/rhel/ks/kvm-rhel-7
7  ks_repo: http://10.5.126.23/repo/rhel/RHEL7-x86_64/
8  volgroup: /dev/vg_virthost
9  vmhost: virthost17.phx2.fedoraproject.org
10 datacenter: phx2
11
12 tcp_ports: ['22', '80', '443', '67', '68']
13 udp_ports: ['67','68','69']
14 custom_rules: [
15     '-A INPUT -p tcp -m tcp -s 192.168.1.20 --dport 5666 -j ACCEPT',
16     '-A INPUT -p tcp -m tcp -s 10.5.126.13 --dport 873 -j ACCEPT',
17     '-A INPUT -p tcp -m tcp -s 192.168.1.59 --dport 873 -j ACCEPT'
18 ]
19
20 eth0_ip: 10.5.126.41
21 csi_relationship: |
22     noc01 is the internal monitoring nagios instance to the phx datacenter.
23     it is also the dhcp server serving all computing nodes
```

(continues on next page)

(continued from previous page)

```

    * This host relies on:
    - the virthost it's hosted on (virthost17.phx2.fedoraproject.org)
    - FAS to authenticate users
    - VPN connectivity

    * Things that rely on this host:
    - Infrastructure team to be awair of the infra status. operations control␣
↪process will fail
    - if this host is down, it will be difficult to know the status of infra and␣
↪provide reactive/proactive support
    - if this host is down, dhcp/bootp leases/renew will fail. pxe booting will fail␣
↪as well
```

2.5 Generating `iptables` Rules from a Template

Note: Ansible suggests that Jinja templates use the extension `.j2`, though Ansible will process the template regardless of whether it has an extension or not. The example `iptables` template used by the Fedora Project has no `.j2` extension, while the DIMS project uses the `.j2` extension to more easily locate Jinja template files using `find . -name '*.j2'` or similar extension-based searching methods.

The template roles/base/templates/iptables/iptables (processed by roles/base/tasks/main.yml as part of the *base role*) provides a Jinja template for an `iptables` rule set.

Note: The complete template file roles/base/templates/iptables/iptables is over 100 lines. It is edited here to remove lines that are irrelevant to the discussion of Jinja templating.

Listing 1: Template for iptables rules

```
# {{ ansible_managed }}
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]

# allow ping and traceroute
-A INPUT -p icmp -j ACCEPT

# localhost is fine
-A INPUT -i lo -j ACCEPT

# Established connections allowed
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A OUTPUT -m state --state RELATED,ESTABLISHED -j ACCEPT

# allow ssh - always
-A INPUT -m conntrack --ctstate NEW -m tcp -p tcp --dport 22 -j ACCEPT

{% if env != 'staging' and datacenter == 'phx2' and inventory_hostname not in groups[
↪'staging-friendly'] %}
```

(continues on next page)

(continued from previous page)

```
#
# In the phx2 datacenter, both production and staging hosts are in the same
# subnet/vlan. We want production hosts to reject connectons from staging group hosts
# to prevent them from interfering with production. There are however a few hosts in
# production we have marked 'staging-friendly' that we do allow staging to talk to␣
↪for
# mostly read-only data they need.
#
{% for host in groups['staging']|sort %}
{% if 'eth0_ip' in hostvars[host] %}# {{ host }}
-A INPUT -s {{ hostvars[host]['eth0_ip'] }} -j REJECT --reject-with icmp-host-
↪prohibited
{% else %}# {{ host }} has no 'eth0_ip' listed
{% endif %}
{% endfor %}
{% endif %}

{% if ansible_domain == 'qa.fedoraproject.org' and inventory_hostname not in groups[
↪'qa-isolated'] %}
#
# In the qa.fedoraproject.org network, we want machines not in the qa-isolated group
# to block all access from that group. This is to protect them from any possible␣
↪attack
# vectors from qa-isolated machines.
#
{% for host in groups['qa-isolated']|sort %}
{% if 'eth0_ip' in hostvars[host] %}# {{ host }}
-A INPUT -s {{ hostvars[host]['eth0_ip'] }} -j REJECT --reject-with icmp-host-
↪prohibited
{% else %}# {{ host }} has no 'eth0_ip' listed
{% endif %}
{% endfor %}
{% endif %}
# if the host declares a fedmsg-enabled wsgi app, open ports for it
{% if wsgi_fedmsg_service is defined %}
{% for i in range(wsgi_procs * wsgi_threads) %}
-A INPUT -p tcp -m tcp --dport 30{{ '%02d' % i }} -j ACCEPT
{% endfor %}
{% endif %}

# if the host/group defines incoming tcp_ports - allow them
{% if tcp_ports is defined %}
{% for port in tcp_ports %}
-A INPUT -p tcp -m tcp --dport {{ port }} -j ACCEPT
{% endfor %}
{% endif %}

# if the host/group defines incoming udp_ports - allow them
{% if udp_ports is defined %}
{% for port in udp_ports %}
-A INPUT -p udp -m udp --dport {{ port }} -j ACCEPT
{% endfor %}
{% endif %}

# if there are custom rules - put them in as-is
{% if custom_rules is defined %}
{% for rule in custom_rules %}
```

(continues on next page)

(continued from previous page)

```
{{ rule }}
{% endfor %}
{% endif %}

# otherwise kick everything out
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

2.6 Customization of System and Service Configuration

Ansible supports variables in playbooks, allowing a generalization of steps to perform some task to be defined separate from the specifics of the content used. Rather than hard-coding a value like a port number into a command, a variable can be used to allow *any* port to be specified at run time. Where and how the variable is set is somewhat complicated in Ansible, as their are many places that variables can be set and a specific order of precedence that is followed. This can be seen in the Ansible documentation, Variable Precedence: Where Should I Put A Variable?.

The Fedora Project takes advantage of an advanced feature of Ansible in the form of the conditional `with_first_found` combined with the use of variables and variable precedence ordering. Ansible's own web page has a note saying, "This is an advanced topic that is infrequently used. You can probably skip this section."

(See Selecting Files And Templates Based On Variables).

An example of how this is used is found in roles/base/tasks/main.yml where the DNS resolver configuration file is applied:

Listing 2: Configuration file for DNS resolver

```
- name: /etc/resolv.conf
  copy: src={{ item }} dest=/etc/resolv.conf
  with_first_found:
    - "{{ resolvconf }}"
    - resolv.conf/{{ ansible_fqdn }}
    - resolv.conf/{{ host_group }}
    - resolv.conf/{{ datacenter }}
    - resolv.conf/resolv.conf
  tags: [ base, config, resolvconf ]
```

The first thing to notice is that the base name of file being installed here (`resolv.conf`) is used to name a directory in which all variations of that file will be stored. This keeps the directory for the role clean and organized.

The second thing to notice is well organized hierarchy of precedence from most specific to least specific.

Table 1: Search order for selecting customized file

File name	Derived from	Specific to	Relevant Content
`{{ resolvconf }}`	Extra var (`-e`)	A single file	Useful for dev/test
`{{ ansible_fqdn }}`	Ansible fact	A single host	Host-specific customizations
`{{ host_group }}`	Group name	Hosts of a specific class	Service-specific customizations
`{{ datacenter }}`	Defaults, vars, extra vars	Hosts in specific datacenter	Datacenter-specific customizations
`resolv.conf`	Not derived	Not specific	File of last resort

Using this kind of hierarchical naming scheme, it is easy to bring any host under Ansible control. Say that the `resolv.conf` file on a host that is not under Ansible control was created by editing it with `vi`. That file (which we will assume is working at the moment) can be copied into the `resolv.conf/` directory as-is, using the fully-qualified domain name of the host. Even better, place the `{{ ansible_managed }}` template into the file to make it clear that the file is now under Ansible control. If it later becomes clear that a more generic configuration is appropriate to make the host behave the same as other hosts in the same group, or same datacenter, you can simply remove the file with the fully-qualified domain name and the next file that is found (be it for host group, datacenter, or the fallback generic file) will be used.

Note: If the fallback `resolv.conf` file is a direct copy of the file installed by default from the original parent distribution package, having Ansible re-install a functionally equivalent version meets the objective of being *idempotent*.

2.7 Tags on Tasks

Ansible has a feature known as **tags**, that provides a fine-grained mechanism for isolating which plays within a playbook will be executed at runtime. This feature is advanced, and complicated, and you will find both people wholeheartedly embracing the use of tags (e.g., Ansible (Real Life) Good Practices and Tagging) and firmly shunning the use of tags (e.g., Best practices to build great Ansible playbooks)

While it is true that using tags within roles adds to complexity, the alternative of proliferating individual playbooks has its own drawbacks in terms of tight coupling of logic between playbooks that share some common actions, as well as an increased number of individual playbooks to be managed and invoked. By limiting the number of category tags to a minimum, a reasonable tradeoff is made between complexity within playbooks plus the need to write playbooks carefully vs. complexity in managing and invoking playbooks to perform system automation tasks.

2.7.1 DIMS Tagging Methodology

The `ansible-dims-playbooks` roles created by the DIMS Project, like those from the Fedora Project, uses tags to allow fine-grained control of which actions are applied during a given run of a complete host playbook. This allows all roles to be defined and fewer playbooks to be used to execute tasks, but requires that *all tasks have tags* in order for `--tags` and `--skip-tags` to work properly.

The Fedora Project also uses a monolithic playbook that include host playbooks for all hosts. Because tags are applied using an **OR** operation, rather than an **AND** operation, the selection of which tasks are to be applied is then made on either a per-host basis (using host tags or the `--limit` option), or a per-category basis (using category tags).

Figure *Ansible Tags for Vertical and Horizontal Control of Play Execution* illustrates the conceptual layering of tags for **roles** (vertical selection) and **categories** (horizontal selection) across a set of playbooks.

- Using **role** tags will cause all of the plays in the playbook for the specified roles to be executed (and no others).
- Using **category** tags will cause all of the plays in all playbooks that have that category tag to be executed (and no others).

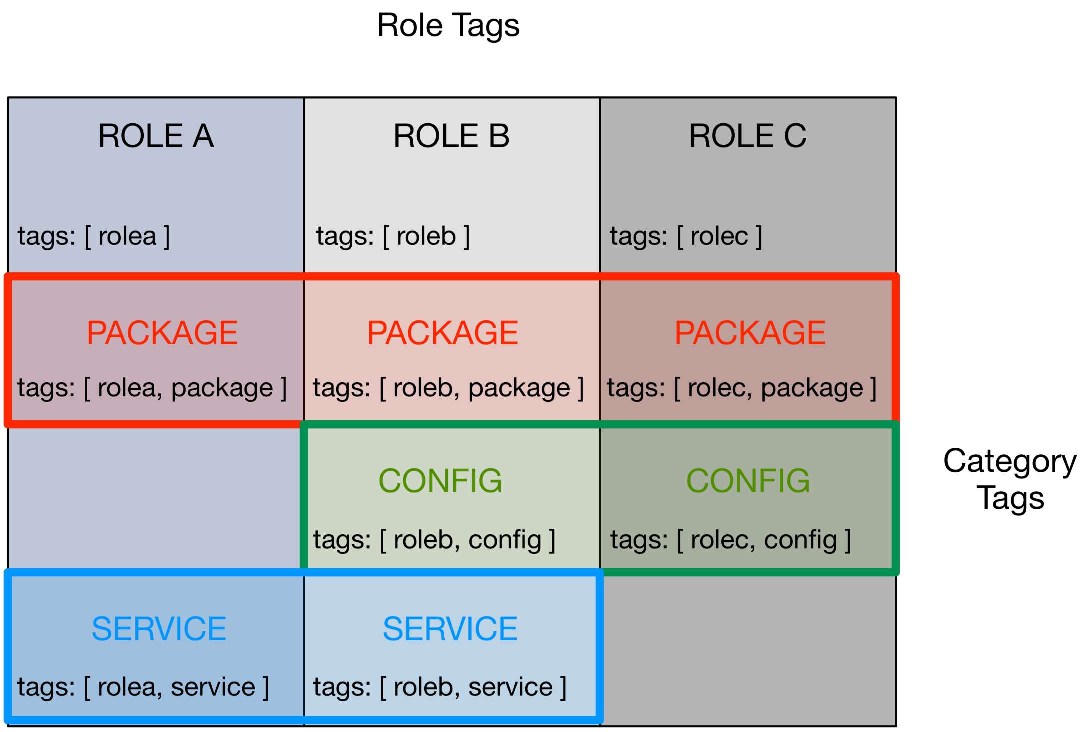


Fig. 2: Ansible Tags for Vertical and Horizontal Control of Play Execution

The concept of **vertical** application of role tags is seen in the vertical gray shaded boxes in Figure *Ansible Tags for Vertical and Horizontal Control of Play Execution* by the presence of a tag that matches the name of the role in every play in each role's playbook (e.g., `rolea` in every `tags` array for Role A, etc.) If every play in a playbook is tagged with an identifier that matches the name of the role, using that tag will limit execution to only those plays for the indicated role.

The concept of **horizontal** application of category tags is seen in Figure *Ansible Tags for Vertical and Horizontal Control of Play Execution* by the presence of a tag that matches a small set of general categories of actions that are common to multiple roles. Plays that are related to package installation are shown in the red horizontal box with `package`, plays related to system configuration files in the green box tagged with `config`, and service activation state plays in the blue horizontal box tagged with `service`. By changing variables that specify package versions and then running all plays tagged with `package`, you can update specific packages on all systems at once (seen in the red box in Figure *Ansible Tags for Vertical and Horizontal Control of Play Execution*).

Attention: The DIMS project uses special `pre_tasks.yml` and `post_tasks.yml` task playbooks that are included at the top and bottom of every role. This mechanism supports standardized actions that apply to every role, such as creating and cleaning up deployment directories, printing out debugging information at the start of each role to expose run-time variable contents, sending log messages to the continuous integration logging channel, etc.

In order to **only** run these pre- and post-tasks when tags are specified, each include line **must** include the union of all possible tags that are used. This can be seen in the following tasks file for the `vagrant` role.

Listing 3: Vagrant role `tasks/main.yml` file

```
---

# File: roles/vagrant/tasks/main.yml

# Prepare system for using Vagrant

- import_tasks: "{{ tasks_path }}/pre_tasks.yml"
  tags: [ vagrant, packages ]

- name: Get vagrant deb file
  get_url:
    url: "{{ vagrant_dist_url }}/{{ vagrant_artifact }}"
    dest: "{{ dims_deploy }}/{{ role_name }}"
    sha256sum: "{{ vagrant_deb_64bit_sum }}"
  become: yes
  when: ansible_os_family == "Debian"
  tags: [ vagrant, packages ]

- name: Ensure vagrant deb package present
  shell: "dpkg -i {{ dims_deploy }}/{{ role_name }}/{{ vagrant_artifact }}"
  become: yes
  when: ansible_os_family == "Debian"
  tags: [ vagrant, packages ]

- name: Ensure configure_networks.rb patch is present
  copy:
    src: "{{ patches }}/diffs.vagrant_configure_networks"
    dest: "{{ dims_deploy }}/{{ role_name }}diffs.vagrant_configure_networks"
    mode: 0o644
  when: ansible_os_family == "Debian"
  tags: [ vagrant, packages ]

- name: Hot patch CoreOS configure_networks.rb
  shell: >
    patch
    /opt/vagrant/embedded/gems/gems/vagrant-{{ vagrant_version }}/plugins/guests/
↪coreos/cap/configure_networks.rb
    {{ dims_deploy }}/{{ role_name }}diffs.vagrant_configure_networks
  become: yes
  tags: [ vagrant, packages ]
```

2.7.2 Examples of DIMS Tags

Some of the general and specific tags that are used frequently for performing regular system maintenance and development tasks are listed below. As a general rule, all roles have a tag that matches the role's name, allowing just that one role to be applied out of a general host playbook. (For example, you can apply all of the `base` role's tasks to the host you are currently logged in to using `run.playbook --tags base`.)

Attention: The tags listed in these tables are *not* the full set of tags that are applied to tasks within playbooks. To easily identify all of the tags that exist, a coding convention of placing all tags in an array on one line is used, allowing one to search for them using `grep` as seen here:

```
roles/base/tasks/coreos.yml:  tags: [ base, config ]
roles/base/tasks/dims_base.yml:  tags: [ base, config ]
roles/base/tasks/dims_base.yml:  tags: [ base, config, tests ]
roles/base/tasks/dnsmasq.yml:  tags: [ base, config ]
roles/base/tasks/dnsmasq.yml:  tags: [ base, config, dns ]
roles/base/tasks/dnsmasq.yml:  tags: [ base, packages, config ]
roles/base/tasks/main.yml:  tags: [ base ]
roles/base/tasks/main.yml:  tags: [ base, config ]
roles/base/tasks/main.yml:  tags: [ base, config, dns ]
roles/base/tasks/main.yml:  tags: [ base, config, dns, logrotate, packages,␣
↪rsyslogd, tests, triggers ]
roles/base/tasks/main.yml:  tags: [ base, config, iptables ]
roles/base/tasks/main.yml:  tags: [ base, config, journald ]
roles/base/tasks/main.yml:  tags: [ base, config, logrotate ]
roles/base/tasks/main.yml:  tags: [ base, config, packages, updates, triggers ]
roles/base/tasks/main.yml:  tags: [ base, config, rsyslogd ]
roles/base/tasks/main.yml:  tags: [ base, hosts, config ]
roles/base/tasks/main.yml:  tags: [ base, packages ]
roles/base/tasks/main.yml:  tags: [ base, packages, scripts, tests ]
roles/base/tasks/main.yml:  tags: [ base, packages, updates ]
roles/base/tasks/main.yml:  tags: [ base, services ]
roles/base/tasks/main.yml:  tags: [ base, tests ]
roles/base/tasks/main.yml:  tags: [ base, triggers ]
```

Tag	Description
`config`	Configuration files (usually requires `notify` of restart handlers to apply changes.
`dns`	Applies any DNS resolution settings and service restarts for `resolv.conf`, `dnsmasq`.
`packages`	Ensures package cache is updated and necessary packages (at specific pinned versions in some cases) are installed and/or held.
`rsyslogd`	Applies any `rsyslogd` related configuration and log handling tasks.
`tests`	Installs/updates `dims_functions.sh` and generates `bats` tests for applicable roles, etc.

Tag	Description
`hosts`	(Re)generates the `/etc/hosts` file and restarts `dnsmasq` server to apply. (Define `custom_hosts` to add IP address mappings in special cases, e.g., when bootstrapping a new deployment that does not yet have its own DNS server configured.)
`iptables`	(Re)generates `iptables` V4 and V6 rules files and reloads rules.
`updates`	Updates installed packages that are not held back.

2.8 Ansible Best Practices and Related Documentation

Before doing too much writing of Ansible playbooks, you should familiarize yourself with the recommended *best practices* for using Ansible for automating program installation and system configuration tasks in a general, repeatable, and scalable manner. Ansible provides recommended Playbooks Best Practices guidelines in the the Ansible Documentation, but sometimes these don't go far enough in guiding a new Ansible user.

Two other sources of highly useful information are the following books and related code examples:

- **Ansible for DevOps, by Jeff Geerling**
  - geerlingguy/ansible-for-devops
- **The DevOps 2.0 Toolkit, by Victor Farcic**
  - vfarcic/vfarcic.github.io

Other useful references collected over the years of using Ansible include:

- **Ansible (web site)**
  - Playbooks Best Practices
  - GitHub ansible/ansible-examples ("A few starter examples of ansible playbooks, to show features and how they work together. See http://galaxy.ansible.com for example roles from the Ansible community for deploying many popular applications.")
  - docker - manage docker containers
- **Alternate "Best Practices" (possibly conflicting, but helpful to consider none the less)**
  - Laying out roles, inventories and playbooks, by Michel Blanc, July 2, 2015
  - Best practices to build great Ansible playbooks, by Maxime Thoonsen, October 12, 2015
  - Ansible (Real Life) Good Practices, by Raphael Campardou, March 19, 2014 (has pre-commit Git hook for `ansible-vault`)
  - Lessons from using Ansible exclusively for 2 years, by Corban Raun, March 24, 2015
  - 6 practices for super smooth Ansible experience, by Maxim Chernyak, June 18, 2014
  - GitHub enginyoyen/ansible-best-practises ("A project structure that outlines some best practices of how to use ansible")
  - More Tips and Tricks, slideshare by bcoca, October 11, 2016 https://www.slideshare.net/bcoca/more-tips-n-tricks
- Episode #43 - 19 Minutes With Ansible (Part 1/4), Justin Weissig, sysadmincasts.com, January 13, 2015
- **Episode #45 - Learning Ansible with Vagrant (Part 2/4), Justin Weissig, sysadmincasts.com, March 19, 2015**
  - GitHub jweissig/episode-45 ("Episode #45 - Learning Ansible with Vagrant")
- Episode #46 - Configuration Management with Ansible (Part 3/4), Justin Weissig, sysadmincasts.com, March 26, 2015
- **Episode #47 - Zero-downtime Deployment with Ansible (Part 4/4), Justin Weissig, sysadmincasts.com, April 2, 2015**
  - GitHub jweissig/episode-47 ("Episode #47 - Zero-downtime Deployments with Ansible (Part 4/4)")
- **Graduating Past Playbooks: How to Use Ansible When Your Infrastructure Grows Up, by Rob McQueen**
  - GitHub nylas/ansible-flask-example ("Example using ansible-test and wrapper roles to implement a simple flask webapp")
- The Fedora Project ansible playbook/files/etc repository for fedora infrastructure
- How Twitter Uses Ansible, YouTube video by Ansible, May 21, 2014
- GitHub ePages-de/mac-dev-setup ("Automated provisioning of your Apple Mac (Java) development machine using Ansible")
- **Advanced Ansible concepts, gotchas, things to keep in mind…**

- **Security hardening for openstack-ansible, Openstack web site**
  - Automated Security Hardening with OpenStack-Ansible, by Major Hayden, Openstack Austin Summit, May 1, 2016
  - GitHub openstack/openstack-ansible-security ("Security Role for OpenStack-Ansible http://openstack.org")
- **Templating**
  - Jinja2 for better Ansible playbooks and templates, by Daniel Schneller, August 25, 2014
  - Ansible: "default" and "bool" filters, by dddpaul-github, November 30, 2015
  - Ansible loop through group vars in template, Stackoverflow post, November 18, 2014
  - Ansible loop over variables, Stackoverflow post, October 28, 2014
- **Dynamic Inventory**
  - Dynamic Inventory, Ansible documentation
  - Adapting inventory for Ansible, by Jan-Piet Mens
  - Creating custom dynamic inventories for Ansible, by Jeff Geerling, June 11, 2015
  - Writing a Custom Ansible Dynamic Inventory Script, by Adam Johnson, December 4, 2016
  - Using DNS as an Ansible dynamic inventory, by Remie Bolte, January 1, 2016
- **Facts vs. Variables**
  - Fact Caching and gathering, Ansible documentation
  - Fastest way to gather facts to fact cache, Stackoverflow post, September 1, 2015
  - Ansible Custom Facts, serverascode.com
- **Ansible Plugins**
  - Ansible module development in Python - 101, by Yves Fauser, Ansible Munich Meetup - going into 2016, February 23, 2016
  - Ansible: Modules and Action Plugins, by Nicholas Grisey Demengel, January 20, 2015
  - An action plugin for Ansible to handle SSH host keys and DNS SSHFP records, by Jan-Piet Mens, November 3, 2012
  - v2 callback plugin migration (thread), Google Groups
- **Front-ends for Ansible**
  - Ansible Tower
  - **DevOps Automation – Ansible+Semaphore is Indispensable!, by Thaddeus, code-complete.com**
    - GitHub ansible-semaphore/semaphore ("Open Source Alternative to Ansible Tower https://ansible-semaphore.github.io/semaphore")
  - Building an Automated Config Management Server using Ansible+Flask+Redis, by deepakmdas (beingsysadmin), April 21, 2015
  - rundeck ("Go fast. Be secure.")
  - **stackstorm ("Event-Driven Automation")**

      - GitHub StackStorm/st2 ("StackStorm (aka "IFTTT for Ops") is event-driven automation commonly used for auto-remediation, security responses, facilitated troubleshooting, complex deployments, and more. Includes rules engine, workflow,1800+ integrations (see /st2contrib), native ChatOps and so forth."
      - New In StackStorm: Ansible Integration, by Eugen C., June 5, 2015

  - **Handling multi-stage or multi-deployment environments**
    - Multistage environments with Ansible, by Ross Tuck, May 15, 2014
    - Multi-stage provisioning, by Victor Volle, Ansible Munich Meetup - going into 2016, February 23, 2016
  - Ansible Tips and Tricks on ReadTheDocs
  - How to Use Ansible Roles to Abstract your Infrastructure Environment, by Justin Ellingwood, February 11, 2014
  - Jinja2 for better Ansible playbooks and templates, by Daniel Schneller, August 25, 2014
  - In YAML, how do I break a string over multiple lines?, stackoverflow post, September 24, 2010
  - Ansible - some random useful things, by David Goodwin, August 4, 2014
  - Tagging, ThinkAnsible, June 4, 2014
  - Scalable and Understandable Provisioning with Ansible and Vagrant, by Julien Ponge, October 15, 2013
  - Alejandro Guirao Rodríguez - Extending and embedding Ansible with Python, YouTube video from EuroPython 2015
  - etcd + ansible = crazy delicious, by UnicornClouds
  - How I Fully Automated OS X Provisioning With Ansible, by Daniel Jaouen
  - Ansible tips, by Deni Bertović, October 13, 2014
  - Debugging Ansible Tasks, by Greg Hurrell, August 7, 2015
  - GitHub dellis23/ansible-toolkit ("Ansible toolkit hopes to solve [some Ansible playbook] problems by providing some simple visibility tools.")
  - GitHub ks888/ansible-playbook-debugger ("A Debugger for Ansible Playbook")
  - Hacking ansible, slideshare, October 15, 2014 ("a quick presentation on ansible internals and a focus on the ease of expansion through the plugin")
  - ansible-exec: ansible-playbook wrapper for executing playbooks, by Hagai Kariti, August 26, 2014
  - Using virtualenv Python in local Ansible, by Matt Behrens, April 5, 2014
  - Ansible: A Simple Rollback Strategy for Roles and Playbooks, by Valentino Gagliardi, June 25, 2014
  - Proposal for fixing playbooks with dynamic include problems, Ansible Development Google Group post

CHAPTER 3

Developing in the Cloud

This section covers deployment and development using a cloud service provider.

3.1 Developing on DigitalOcean

This chapter covers development or prototype deployment on the DigitalOcean platform using Hashicorp Terraform.

Control of Digital Ocean droplets using these playbooks is done in the subdirectory `deploy/do` beneath the root directory of the cloned repository. (This document assumes an environment variable `$PBR` points to the repository root.)

A helper `Makefile` and related scripts facilitate most of the steps used for creating and controlling DigitalOcean droplets using DigitalOcean's remote API. This remote API is accessible directly using programs like `dopy` or `curl`, or indirectly using `terraform`.

Note: The `Makefile` has built-in help text that serves as a reminder of capabilities it supports.

```
Usage: make [something]

Where 'something' is one of these:

 help - print this help text
 init - initialize terraform state (this is automatic)
 droplets - show list of active droplets using digital_ocean.py dynamic inventory
 hosts - show list of hosts in group 'do'
 images - lists available DigitalOcean images
 regions - lists available DigitalOcean regions
 provider - generate terraform provider.tf file for creating nodes in group 'do'

 plan - show the terraform plan for the current state
 graph - generate terraform graph (output is 'graph.png')
 newkeypair - generate a new SSH user keypair
 insertpubkey - insert SSH public key on DigitalOcean
```

(continues on next page)

(continued from previous page)

```
removepubkey - remove SSH public key on DigitalOcean
create - applies terraform plan to create droplets for hosts in group 'do'
bootstrap - create, then run bootstrap.yml and ca-certs.yml playbooks
installcerts - run 'certbot-installcert.yml' playbook to install SSL certs
deploy - run 'master.yml' playbook to deploy roles on droplets

update - update packages
reboot - reboot the system in +1 minute
cancel - cancel rebooting (if you REALLY DIDN'T MEAN IT)

addhostkeys - adds SSH host public keys to selected known_hosts files
removehostkeys - removes SSH host public keys from selected known_hosts files

dumpvars - produces Ansible debug output of vars for hosts in group 'do'
ping - does Ansible ad-hoc ping of hosts in group 'do'
pre.test - run pre-requisite tests for using terraform with DigitalOcean
post.test - run 'test.runner --terse' on all droplets

destroy - destroys droplets for hosts in group 'do'
spotless - remove terraform log and state files

* The default if you just type 'make' is the same as 'make help'
* To control Ansible, set DIMS_ANSIBLE_ARGS to the arguments you want
  to pass along on the command line, for example:
    $ make DIMS_ANSIBLE_ARGS="--tags tests --limit purple" deploy
    $ make DIMS_ANSIBLE_ARGS="--limit purple" post.test
```

3.1.1 Getting Started

Before being able to remotely create and control DigitalOcean droplets and related resource, you need to have credentials, programs, and specific configuration settings (including passwords) for your deployment.

Note: Some of these steps are performed by a `bootstrap` role, but you have to have already set up and configured Ansible and set variables for the host being set up in order to use that mechanism. Those steps are covered in Section *Developing Locally*. This section walks you through doing them manually. Once you are comfortable with managing the Ansible inventory, you can leverage Ansible for bootstrapping systems.

- Install pre-requisite programs on the system you will be using as the Ansible control host (see *Ansible fundamentals*).
  - Debian

    ```
    $ sudo apt-get install -y bats ansible git make
    ```

  - Mac OS X (using homebrew)

    ```
    $ brew install bats ansible git make
    ```

- Install Terraform for your OS.
- Set up an account on DigitalOcean.

Note: If you do not yet have a DigitalOcean account and want to try it out with $10 credit, you may use this referral link: https://m.do.co/c/a05d1634982e

- Create a DNS domain to use for your development deployment and configure the domain's **Nameservers** entries to point to DigitalOcean's NS servers (NS1.DIGITALOCEAN.COM, NS2.DIGITALOCEAN.COM and NS3.DIGITALOCEAN.COM). This is necessary for allowing Terraform to use DigitalOcean's API to create and set DNS A, MX, and TXT records for your droplets. (You will set an environment variable in a moment with this domain name.)

  Now go to your DigitalOcean control panel, select **Networking**, then **Domains**. Enter your new domain name in the **Add a Domain** field and select **Add Domain** to register the domain with Digital Ocean.

  After a short period of time, you should then be able to see the NS records:

```
$ dig example.com ns | grep NS
example.com.          1799    IN      NS      ns3.digitalocean.com.
example.com.          1799    IN      NS      ns1.digitalocean.com.
example.com.          1799    IN      NS      ns2.digitalocean.com.
```

Note: There are many domain name registrars you can use. Factors such as requirements for specific TLD names, longevity of use, cost, existing DNS services already available to you, etc., will guide your choice. For short-term development and testing, you can use one of the "free" TLD registrars (e.g., Freenom).

- Ensure that your ~/.bash_aliases (or ~/.bashrc, depending on how your operating system's Bash installation handles its resource files) has environment variables set up with the following variables.

```
export PBR="${HOME}/path_to_where_you_put/ansible-dims-playbooks"
export DIMS_DOMAIN="example.com"
export DIMS_SITE_ID="$(echo ${DIMS_DOMAIN} | sed 's/\./_/g')"

# For dopy
export DO_API_VERSION="2"
export DO_API_TOKEN="$(psec secrets get do_api_token)"

# For terraform
export DO_PAT=${DO_API_TOKEN}
export TF_VAR_do_api_token="${DO_PAT}"
export TF_VAR_region="sfo2"  # See output of "make regions" for available regions
export TF_VAR_environment="do"
export TF_VAR_domain="${DIMS_DOMAIN}"
export TF_VAR_datacenter="${TF_VAR_domain}"
export TF_VAR_private_key="${HOME}/.ssh/${DIMS_SITE_ID}"
export TF_VAR_public_key="${TF_VAR_private_key}.pub"
```

Note: Just editing this file does not change any currently set environment variables in active shells, so Bash must be forced to re-process this file. Either run exec bash in any active shell window to restart the Bash process, or log out and log back in. You may need to do this several times as you are configuring everything the first time.

- Make sure operating system software pre-requisites are present.

```
$ make prerequisites
```

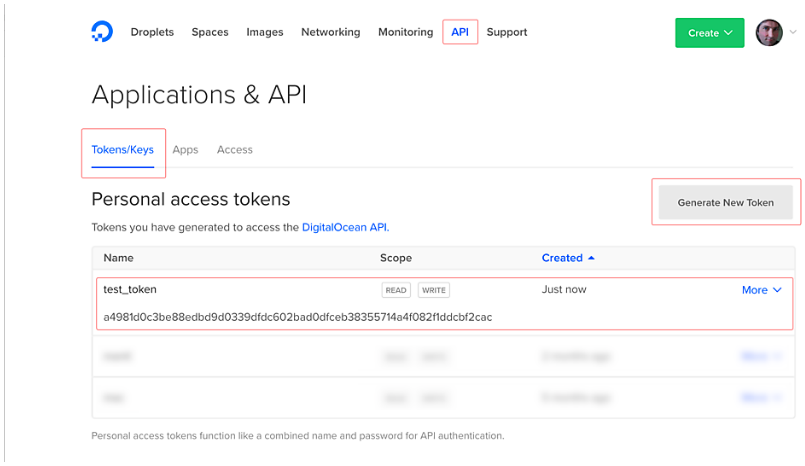
- Test the `terraform` installation and other tools by initializing the directory form within the `deploy/do` directory:

  ```
  $ cd $PBR/deploy/do
  $ make init
  ```

  This step does a few things, including initializing `terraform` and ensuring that a directory for storing secrets (with an empty `token` file) is created with the proper permissions. This "secrets" directory will later hold other secrets, such as passwords, TLS certificates and keys, backups of sensitive database components, etc.

  ```
  $ tree -aifp ~ | grep ~/.secrets
  [drwx------]  /Users/dittrich/.secrets
  [drwx------]  /Users/dittrich/.secrets/digital-ocean
  [-rw-------]  /Users/dittrich/.secrets/digital-ocean/token
  ```

- The file that will hold the token is the last one listed in the `tree` output. To get the token to put in that file, go to your DigitalOcean control panel, select **API**, then select **Generate New Token** (see Figure *Digital Ocean Personal Access Token Generation*). Copy the token and place it in the file `~/.secrets/digital-ocean/token`.

  Applications & API

  Personal access tokens

  Tokens you have generated to access the DigitalOcean API.

Name	Scope	Created ▲	
test_token a4981d0c3be88edbd9d0339dfdc602bad0dfceb38355714a4f082f1ddcbf2cac	READ WRITE	Just now	More

  Personal access tokens function like a combined name and password for API authentication.

  Fig. 1: Digital Ocean Personal Access Token Generation

  After loading the token, you should be able to get a list of available regions with `make regions`:

  ```
  ["nyc1","sfo1","nyc2","ams2","sgp1","lon1","nyc3","ams3","fra1","tor1","sfo2
  ↪","blr1"]
  ```

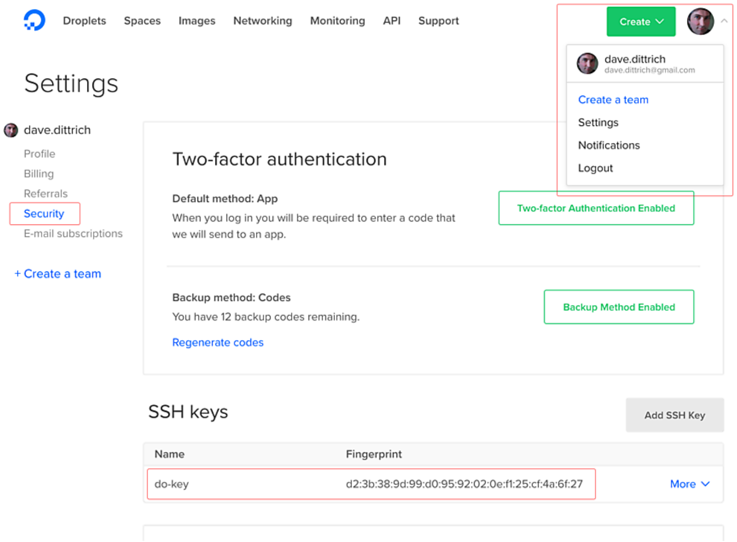
  You can get a list of available images (just the first 10 shown here) using `make images`:

  ```
  {"slug":"cassandra","distribution":"Ubuntu","name":"Cassandra on 14.04"}
  {"slug":"centos-6-5-x32","distribution":"CentOS","name":"6.7 x32"}
  {"slug":"centos-6-5-x64","distribution":"CentOS","name":"6.7 x64"}
  {"slug":"centos-6-x32","distribution":"CentOS","name":"6.9 x32"}
  {"slug":"centos-6-x64","distribution":"CentOS","name":"6.9 x64"}
  {"slug":"centos-7-x64","distribution":"CentOS","name":"7.4 x64"}
  {"slug":"coreos-alpha","distribution":"CoreOS","name":"1618.0.0 (alpha)"}
  {"slug":"coreos-beta","distribution":"CoreOS","name":"1590.2.0 (beta)"}
  {"slug":"coreos-stable","distribution":"CoreOS","name":"1576.4.0 (stable)"}
  {"slug":"debian-7-x32","distribution":"Debian","name":"7.11 x32"}
  ```

- Create an SSH key pair to use for secure remote access to your droplets. Run `make newkeypair` and answer the questions as appropriate. (Normally this is just pressing **Return** multiple times to accept defaults.) This will generate an SSH key pair in your account specifically for use with DigitalOcean.

Note: You can regenerate this key at any time you wish, provided that you do **not have** any active DigitalOcean droplets. Full live re-keying is not yet working, so destroying the SSH key that you are using to access your droplets will break if you switch private keys.

You can test the DigitalOcean API key by inserting the SSH key into your DigitalOcean account using `make insertkey` and then checking the **SSH Keys** section on the **Settings** > **Security** page (see Figure *Digital Ocean SSH Key*).



Two-factor authentication

Default method: App

When you log in you will be required to enter a code that we will send to an app.

Backup method: Codes

You have 12 backup codes remaining.

SSH keys

Name	Fingerprint
do-key	d2:3b:38:9d:99:d0:95:92:02:0e:f1:25:cf:4a:6f:27

Fig. 2: Digital Ocean SSH Key

Finally, you must set up a set of secrets (passwords, primarily) for the services that will be installed when you do `make deploy` after bootstrapping the droplets for Ansible control. These secrets are kept in a file `~/.secrets/digital-ocean/secrets.yml` that should contain at least the following variables:

```
---

ca_rootca_password: 'sposTeAsTo'
jenkins_admin_password: 'WeAsToXYLN'
rabbitmq_default_user_pass: 'xsTIoglYWe'
rabbitmq_admin_user_pass: 'oXYLNwspos'
trident_sysadmin_pass: 'glYWeAsTlo'
trident_db_pass: 'lZ4gDxsTlo'
vncserver_password: 'lYWeALNwsp'
```

Caution: **DO NOT** just cut and paste those passwords! They are just examples that should be replaced with similarly strong passwords. You can chose 5 random characters, separate them by one or two punctuation characters, followed by some string that reminds you of the service (e.g., “trident” for Trident) with some other punction or capitalization thrown in to strengthen the resulting password. This is relatively easy to remember, is not the same for all services, is lenghty enough to be difficult to brute-force, and is not something that is likely to be found in a dictionary of compromised passwords. (You may wish to use a program like `bashpass` to generate random strong passwords like `helpful+legmen~midnight`.)

A bats test file exists to validate *all* of the required elements necessary to create and control DigitalOcean droplets. When all pre-requisites are satisfied, all tests will succeed. If any fail, resolve the issue and try again.

```
$ make pre.test
bats do.bats
 ✓ [S][EV] terraform is found in $PATH
 ✓ [S][EV] Directory for secrets (~/.secrets/) exists
 ✓ [S][EV] Directory for secrets (~/.secrets/) is mode 700
 ✓ [S][EV] Directory for DigitalOcean secrets (~/.secrets/digital-ocean/) exists
 ✓ [S][EV] DigitalOcean token file (~/.secrets/digital-ocean/token) is not empty
 ✓ [S][EV] Secrets for DigitalOcean (~/.secrets/digital-ocean/secrets.yml) exist
 ✓ [S][EV] Variable DIMS_DOMAIN is defined in environment
 ✓ [S][EV] Variable DIMS_SITE_ID is defined in environment
 ✓ [S][EV] Variable DO_API_VERSION (dopy) is defined in environment
 ✓ [S][EV] Variable DO_API_TOKEN (dopy) is defined in environment
 ✓ [S][EV] Variable DO_PAT (terraform) is defined in environment
 ✓ [S][EV] Variable TF_VAR_do_api_token (terraform) is defined in environment
 ✓ [S][EV] Variable TF_VAR_region (terraform) is defined in environment
 ✓ [S][EV] Variable TF_VAR_environment (terraform) is defined in environment
 ✓ [S][EV] Variable TF_VAR_domain (terraform) is defined in environment
 ✓ [S][EV] Variable TF_VAR_datacenter (terraform) is defined in environment
 ✓ [S][EV] Variable TF_VAR_private_key (terraform) is defined in environment
 ✓ [S][EV] Variable TF_VAR_public_key (terraform) is defined in environment
 ✓ [S][EV] DO_API_TOKEN authentication succeeds
 ✓ [S][EV] File pointed to by TF_VAR_public_key exists and is readable
 ✓ [S][EV] File pointed to by TF_VAR_private_key exists and is readable
 ✓ [S][EV] Git user.name is set
 ✓ [S][EV] Git user.email is set

23 tests, 0 failures
```

The fundamentals are now in place for provisioning and deploying the resources for a D2 instance on DigitalOcean.

3.1.2 Bootstrapping DigitalOcean Droplets

Once remote access to DigitalOcean via the remote API is set up, you can create droplets. The target insertpubkey helps upload the SSH public key (though this is also done automatically by terraform apply). Test that this works (and get familiar with how DigitalOcean handles SSH keys) running make insertpubkey and then checking using the DigitalOcean dashboard to verify the key was inserted. You can find the **SSH Keys** section on the **Settings** > **Security** page (see Figure *Digital Ocean SSH Key*).

3.1.3 Creating DigitalOcean Resources

Running make create will update the provider.tf file from a Jinja template, then apply the plan. This is useful whenever you make changes to variables that affect things like droplet attributes (e.g., disk size, RAM, number of CPUs, etc.) and DNS records.

Caution: Some changes to droplet configuration settings will entice terraformm apply to destroy the resource and recreate it. This is not much of an issue for things like DNS entries, but if it causes a droplet to be destroyed you may – if you are not paying attention and say **No** when terraform asks for confirmation – destroy files you have created in the droplet being recreated.

3.1.4 Bootstrapping DigitalOcean Droplets

Running `make bootstrap` will apply the `bootstrap` role to the droplets, preparing them for full Ansible control. This is typically only necessary when the droplets are first created. After that, the specific host playbooks from the `deploy/do/playbooks/` directory are used to ensure the defined roles are applied to the droplets.

Note: You can limit the hosts being affected when running Ansible via the `Makefile` rules by defining the variable `DIMS_ANSIBLE_ARGS` on the command line to pass along any Ansible command line arguments to `ansible` or `ansible-playbook`. For example,

```
$ make DIMS_ANSIBLE_ARGS="--limit red" bootstrap
$ make DIMS_ANSIBLE_ARGS="--limit green,purple" ping
$ make DIMS_ANSIBLE_ARGS="--tags base -vv" deploy
```

3.1.5 Backing Up Certificates and Trident Portal Data

There are two `Makefile` helper targets that will create backups of either Letsencrypt certificate related files or Trident database files.

Using `make backup.letsencrypt` creates a backup of the `/etc/letsencrypt` directory tree, preserving the `certbot` account information used to generate the host's certificate, the most recently generated certificate, renewal information, etc. This backup can be restored the next time the droplet is destroyed and created again, allowing the host to immediately be used for SSL/TLS secured connections.

Using `make backup.postgres` creates a backup of the Trident `postgresql` database, preserving any manually-created portal content required for demonstration, testing, or debugging.

For more information on how these backups work, see Section *Backups and Restoration*.

3.1.6 Destroying DigitalOcean Resources

Doing `make destroy` will destroy *all* of the DigitalOcean resources you have created and remove the SSH host keys from the local `known_hosts` files.

To destroy specific resources, use `terraform destroy` and specify the resource using the `-target=` option. For example, here is how to destroy the droplet `purple`:

```
$ terraform destroy -target=digitalocean_droplet.purple
digitalocean_droplet.purple: Refreshing state... (ID: 79647375)

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  - destroy

Terraform will perform the following actions:

  - digitalocean_droplet.purple

  - digitalocean_record.purple


Plan: 0 to add, 0 to change, 2 to destroy.
```

(continues on next page)

(continued from previous page)

```
Do you really want to destroy?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

digitalocean_record.purple: Destroying... (ID: 33572623)
digitalocean_record.purple: Destruction complete after 1s
digitalocean_droplet.purple: Destroying... (ID: 79647375)
digitalocean_droplet.purple: Still destroying... (ID: 79647375, 10s elapsed)
digitalocean_droplet.purple: Destruction complete after 13s

Destroy complete! Resources: 2 destroyed.
```

CHAPTER 4

Developing Locally

This chapter walks through the process of bootstrapping a baremetal machine to serve as a Virtualbox hypervisor for hosting multiple Virtual Machine guests, serving as the Ansible control host for managing their configuration.

Note: Some of the examples here explicitly use `-i` to point to an inventory directory, and some do not. When there is no `-i` flag, it is assumed that `/etc/ansible/ansible.cfg`, or a perhaps `ansible.cfg` in the top level of a private customization directory, is configured to point to the correct inventory directory.

You can see what the default is using `ansible --help`:

```
Usage: ansible <host-pattern> [options]

Options:
  . . .
  -i INVENTORY, --inventory-file=INVENTORY
                        specify inventory host path
                        (default=/Users/dittrich/dims/git/ansible-dims-
                        playbooks/inventory) or comma separated host list.
  . . .

..
```

… or by using `ansible --version`:

```
ansible 2.3.0.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/dittrich/dims/git/private-develop/library',
  u'/home/dittrich/dims/git/ansible-dims-playbooks/library', u'/usr/share/ansible']
  python version = 2.7.13 (default, Jun 23 2017, 23:57:31) [GCC 4.8.4]
```

If this is set up properly, you should be able to list the `all` group and see results for the correct deployment:

```
$ ansible --list-hosts
hosts (11):
  blue14.devops.local
```

(continues on next page)

(continued from previous page)

```
purple.devops.local
node03.devops.local
vmhost.devops.local
node02.devops.local
yellow.devops.local
node01.devops.local
orange.devops.local
red.devops.local
blue16.devops.local
hub.devops.local
```

4.1 Initial Connectivity

The first step in putting hosts under Ansible control is to add them to an inventory, setting parameters allowing access to them. We will add them to a local "private" configuration repository, rooted at `$GIT/private-develop`. Since these are systems newly installed using an Ubuntu Kickstart USB drive, they only have a password on the ansible account that we set up, and were installed with IP addresses that were assigned by DHCP on the local subnet at installation time. Until they have been fully configured, they have been assigned an address on (the original DHCP assignments are commented out on lines 12 and 15, and the actively working addresses set on lines 24 and 26.) were manually set up on ports connected to an internal VLAN. The relevant portions of the YAML inventory file are shown here, listed in the `servers` inventory, with host variables defined in the `children` subgroup named `bootstrap` that we can refer to in Ansible ad-hoc mode:

```
 1  ---
 2
 3  # File: inventory/servers/nodes.yml
 4
 5  servers:
 6    vars:
 7      ansible_port: 8422
 8    hosts:
 9      'other-hosts-not-shown':
10      'stirling.devops.develop':
11        #ansible_host: '140.142.29.161'
12      'dellr510.devops.develop':
13        #ansible_host: '140.142.29.186'
14    children:
15      bootstrap:
16        vars:
17          ansible_port: 22
18          http_proxy: ''
19          https_proxy: ''
20        hosts:
21          'stirling.devops.develop':
22              ansible_host: '10.142.29.161'
23          'dellr510.devops.develop':
24              ansible_host: '10.142.29.186'
25
26  # vim: ft=ansible :
```

Validate the temporary `bootstrap` group that defines the two hosts we are setting up using the `debug` module to show the `ansible_host` variable and ensure they match what we set them to.

```
$ ansible -i inventory/ -m debug -a 'var=vars.ansible_host' bootstrap
stirling.devops.develop | SUCCESS => {
    "changed": false,
    "vars.ansible_host": "10.142.29.161"
}
dellr510.devops.develop | SUCCESS => {
    "changed": false,
    "vars.ansible_host": "10.142.29.186"
}
```

Now use the password that was set up at install time to validate that SSH is working using the `ping` or `raw` module (both are shown here, though only one test is necessary to validate connectivity).

Note: For this example, SSH host key checking is being temporarily disabled as we are using an internal VLAN. The host keys were written down in a journal when the installation was performed and SSH used manually to validate the key, which will be collected in a later step.

```
$ export ANSIBLE_HOST_KEY_CHECKING=False
$ ansible --ask-pass -m ping  bootstrap
SSH password:
dellr510.devops.develop | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
stirling.devops.develop | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
$ ansible -m raw -a uptime --ask-pass bootstrap
SSH password:
dellr510.devops.develop | SUCCESS | rc=0 >>
 22:21:50 up  3:37,  3 users,  load average: 0.78, 1.45, 1.29
Shared connection to 140.142.29.186 closed.


stirling.devops.develop | SUCCESS | rc=0 >>
 22:21:51 up  4:15,  3 users,  load average: 2.45, 1.49, 1.18
Shared connection to 140.142.29.161 closed.
```

Use the `ansible` account password with ad-hoc mode to invoke the `authorized_key` module to insert the `ansible` SSH private key in the account on the remote systems, using the `file` lookup and the `dims.function` shell utility function to derive the path to the private key, adding the `.pub` extension to get the public key.

```
$ ansible -m authorized_key -a "user=ansible state=present \
> key='{{ lookup('file', '$(dims.function get_ssh_private_key_file ansible).pub') }}'
↪" \
> --ask-pass bootstrap
SSH password:
dellr510.devops.develop | SUCCESS => {
    "changed": true,
    "exclusive": false,
    "key": "ssh-rsa AAAAB3NzaC1yc2...",
    "key_options": null,
    "keyfile": "/home/ansible/.ssh/authorized_keys",
    "manage_dir": true,
```

(continues on next page)

(continued from previous page)

```
    "path": null,
    "state": "present",
    "unique": false,
    "user": "ansible",
    "validate_certs": true
}
stirling.devops.develop | SUCCESS => {
    "changed": true,
    "exclusive": false,
    "key": "ssh-rsa AAAAB3NzaC1yc2...",
    "key_options": null,
    "keyfile": "/home/ansible/.ssh/authorized_keys",
    "manage_dir": true,
    "path": null,
    "state": "present",
    "unique": false,
    "user": "ansible",
    "validate_certs": true
}
```

4.2 Establishing Full Internet Connectivity

Now that the SSH public key is in the `authorized_keys` files, we can remove the `--ask-pass` option and present the SSH private key to validate that standard remote access with Ansible will now work. Let's also use this opportunity to test outbound network access by sending an ICMP packet to one of Google's DNS servers.

```
$ ansible -i inventory/ --ask-pass -m shell -a "ping -c 1 8.8.8.8"  bootstrap
SSH password:
dellr510.devops.develop | SUCCESS | rc=0 >>
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=57 time=1.39 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.395/1.395/1.395/0.000 ms

stirling.devops.develop | SUCCESS | rc=0 >>
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=57 time=1.44 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.446/1.446/1.446/0.000 ms
```

4.3 Bootstrapping Full Ansible Control

At this point we have verified Ansible can access the systems and that they can access the Internet. Those are the basics we need to now run the `bootstrap.yml` playbook to prepare the system for being a virtual machine hypervisor and Ansible control host. The tasks performed (at the high level) are seen here:

```
---

# File: roles/bootstrap/tasks/main.yml

# This role is intended to be run once after initial
# operating system installation to ensure that the system
# is ready to be controlled remotely using Ansible. That
# includes things like timezone setting and NTP time
# synchronization, installation of required packages,
# configuration of OpenSSH, initial firewall settings, 'sudo'
# access for the 'ansible' account, etc.

# This role can be applied using the generic
# 'playbooks/base_playbook.yml' file, setting the 'host'
# and 'role' variables appropriately for the target host(s).
# Be sure to use '--become' as well, as all of these tasks
# require root.
#
#  $ ansible-playbook $PBR/playbooks/bootstrap.yml \
#  > --ask-become-pass --ask-pass --become -e host=bootstrap

- name: Ensure hardware-specific packages present
  import_tasks: 'hardware.yml'
  tags: [ 'bootstrap', 'hardware' ]

- name: Ensure required packages are present
  import_tasks: '{{ tasks_path }}/packages.yml'
  tags: [ 'bootstrap', 'packages' ]

- name: Ensure timezone set
  import_tasks: 'timezone.yml'
  tags: [ 'bootstrap', 'timezone' ]

- name: Ensure NTP sync set up
  import_tasks: 'ntpcheck.yml'
  tags: [ 'bootstrap', 'ntpcheck' ]

- name: Establish sudo access
  import_tasks: 'sudo.yml'
  tags: [ 'bootstrap', 'sudo' ]

- name: Ensure hostname is set consistent with base role
  import_tasks: '{{ tasks_path }}/hostname.yml'
  tags: [ 'bootstrap', 'hostname' ]

- name: Ensure DIMS-specific resources present
  import_tasks: dims_base.yml
  tags: [ 'bootstrap', 'dims_base' ]

- name: Set up SSH access for Ansible control
  import_tasks: 'ssh.yml'
  tags: [ 'bootstrap', 'ssh' ]

- name: Set up monitoring features
  import_tasks: 'monitoring.yml'
  tags: [ 'bootstrap', 'monitoring' ]
```

(continues on next page)

(continued from previous page)

```
- name: Display diagnostic and validation information
  import_tasks: 'info.yml'
  tags: [ 'bootstrap', 'info' ]

# vim: ft=ansible :
```

Run the playbook as shown (or substitute the inventory host name directly, e.g., `dellr510.devops.develop`, instead of the group name `bootstrap`. Using the group, you can prepare as many hosts as you wish at one time, in this case we show configuration of two hosts simultaneously.

```
$ ansible-playbook -i inventory/ $PBR/playbooks/bootstrap.yml --ask-become-pass --ask-
↪pass --become -e host=bootstrap
SSH password:
SUDO password[defaults to SSH password]:

PLAY [Bootstrapping 'bootstrap'] **********************************************

TASK [Debugging] **************************************************************
Sunday 23 July 2017  12:41:06 -0700 (0:00:00.060)       0:00:00.060 ***********
skipping: [dellr510.devops.develop]
skipping: [stirling.devops.develop]

TASK [Include codename-specific variables] ************************************
Sunday 23 July 2017  12:41:07 -0700 (0:00:01.063)       0:00:01.124 ***********
ok: [dellr510.devops.develop] => (item=/home/dittrich/dims/git/ansible-dims-playbooks/
↪playbooks/../vars/trusty.yml)
ok: [stirling.devops.develop] => (item=/home/dittrich/dims/git/ansible-dims-playbooks/
↪playbooks/../vars/trusty.yml)

TASK [bootstrap : Check for Broadcom device 14e4:43b1] ************************
Sunday 23 July 2017  12:41:08 -0700 (0:00:01.075)       0:00:02.200 ***********
changed: [stirling.devops.develop]
changed: [dellr510.devops.develop]

TASK [bootstrap : Ensure Broadcom wireless kernel in place] *******************
Sunday 23 July 2017  12:41:10 -0700 (0:00:01.705)       0:00:03.905 ***********
skipping: [dellr510.devops.develop]
skipping: [stirling.devops.develop]

TASK [bootstrap : Make sure required APT packages are present (Debian)] *******
Sunday 23 July 2017  12:41:11 -0700 (0:00:01.633)       0:00:05.539 ***********
ok: [dellr510.devops.develop] => (item=[u'apt-transport-https', u'bash-completion', u
↪'ca-certificates', u'cpanminus', u'curl', u'dconf-tools', u'git-core', u'default-jdk
↪', u'gitk', u'gnupg2',
 u'htop', u'hunspell', u'iptables-persistent', u'ifstat', u'make', u'myrepos', u
↪'netcat', u'nfs-common', u'chrony', u'ntpdate', u'openssh-server', u'patch', u'perl
↪', u'postfix', u'python', u'
python-apt', u'remake', u'rsync', u'rsyslog', u'sshfs', u'strace', u'tree', u'vim', u
↪'xsltproc', u'chrony', u'nfs-kernel-server', u'smartmontools', u'unzip'])
ok: [stirling.devops.develop] => (item=[u'apt-transport-https', u'bash-completion', u
↪'ca-certificates', u'cpanminus', u'curl', u'dconf-tools', u'git-core', u'default-jdk
↪', u'gitk', u'gnupg2',
 u'htop', u'hunspell', u'iptables-persistent', u'ifstat', u'make', u'myrepos', u
↪'netcat', u'nfs-common', u'chrony', u'ntpdate', u'openssh-server', u'patch', u'perl
↪', u'postfix', u'python', u'
python-apt', u'remake', u'rsync', u'rsyslog', u'sshfs', u'strace', u'tree', u'vim', u
↪'xsltproc', u'chrony', u'nfs-kernel-server', u'smartmontools', u'unzip'])
```

(continues on next page)

(continued from previous page)

```

TASK [bootstrap : Make sure required APT packages are present (RedHat)] *******
Sunday 23 July 2017  12:41:26 -0700 (0:00:15.023)       0:00:20.562 ***********
skipping: [dellr510.devops.develop] => (item=[])
skipping: [stirling.devops.develop] => (item=[])

TASK [bootstrap : Ensure dims_timezone is set] ********************************
Sunday 23 July 2017  12:41:27 -0700 (0:00:01.168)       0:00:21.731 ***********
skipping: [dellr510.devops.develop]
skipping: [stirling.devops.develop]

TASK [bootstrap : Set timezone variables] *************************************
Sunday 23 July 2017  12:41:28 -0700 (0:00:01.069)       0:00:22.800 ***********
changed: [dellr510.devops.develop]
changed: [stirling.devops.develop]

TASK [bootstrap : Ensure Debian chrony package is installed] ******************
Sunday 23 July 2017  12:41:31 -0700 (0:00:02.035)       0:00:24.836 ***********
ok: [dellr510.devops.develop]
ok: [stirling.devops.develop]

TASK [bootstrap : Ensure chrony is running on Debian] *************************
Sunday 23 July 2017  12:41:33 -0700 (0:00:02.679)       0:00:27.515 ***********
ok: [dellr510.devops.develop]
ok: [stirling.devops.develop]

TASK [bootstrap : Ensure RedHat chrony package is installed] ******************
Sunday 23 July 2017  12:41:35 -0700 (0:00:01.601)       0:00:29.116 ***********
skipping: [dellr510.devops.develop]
skipping: [stirling.devops.develop]

TASK [bootstrap : Ensure chrony is running on RedHat] *************************
Sunday 23 July 2017  12:41:36 -0700 (0:00:01.067)       0:00:30.184 ***********
skipping: [dellr510.devops.develop]
skipping: [stirling.devops.develop]

TASK [bootstrap : Verify that the sudo group exists] **************************
Sunday 23 July 2017  12:41:37 -0700 (0:00:01.066)       0:00:31.250 ***********
ok: [dellr510.devops.develop]
ok: [stirling.devops.develop]

TASK [bootstrap : Set fact with temp sudoers filename] ************************
Sunday 23 July 2017  12:41:38 -0700 (0:00:01.462)       0:00:32.712 ***********
ok: [dellr510.devops.develop]
ok: [stirling.devops.develop]

TASK [bootstrap : Copy sudoers template to temporary file] ********************
Sunday 23 July 2017  12:41:39 -0700 (0:00:01.068)       0:00:33.781 ***********
changed: [dellr510.devops.develop]
changed: [stirling.devops.develop]

TASK [bootstrap : Back up sudoers file] ***************************************
Sunday 23 July 2017  12:41:41 -0700 (0:00:01.914)       0:00:35.695 ***********
changed: [dellr510.devops.develop]
changed: [stirling.devops.develop]

TASK [bootstrap : Verify sudoers before replacing] ****************************
```

(continues on next page)

(continued from previous page)

```
Sunday 23 July 2017  12:41:43 -0700 (0:00:01.398)       0:00:37.093 ************
changed: [dellr510.devops.develop]
changed: [stirling.devops.develop]

TASK [bootstrap : Define variable with ansible public key] *********************
Sunday 23 July 2017  12:41:44 -0700 (0:00:01.508)       0:00:38.602 ************
ok: [dellr510.devops.develop]
ok: [stirling.devops.develop]

TASK [bootstrap : Ensure ansible public key in authorized_keys] ****************
Sunday 23 July 2017  12:41:46 -0700 (0:00:02.083)       0:00:40.686 ************
ok: [dellr510.devops.develop]
changed: [stirling.devops.develop]

TASK [bootstrap : Show interface details (Debian)] *****************************
Sunday 23 July 2017  12:41:48 -0700 (0:00:01.710)       0:00:42.397 ************
changed: [dellr510.devops.develop]
changed: [stirling.devops.develop]

TASK [bootstrap : debug] *******************************************************
Sunday 23 July 2017  12:41:49 -0700 (0:00:01.397)       0:00:43.794 ************
ok: [dellr510.devops.develop] => {
    "_ifconfig.stdout_lines": [
        "em1       Link encap:Ethernet  HWaddr 78:2b:cb:57:9b:e1  ",
        "          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1",
        "",
        "em2       Link encap:Ethernet  HWaddr 78:2b:cb:57:9b:e2  ",
        "          inet addr:10.142.29.186  Bcast:10.142.29.255  Mask:255.255.255.0",
        "          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1",
        "",
        "lo        Link encap:Local Loopback  ",
        "          inet addr:127.0.0.1  Mask:255.0.0.0",
        "          UP LOOPBACK RUNNING  MTU:65536  Metric:1",
        "",
        "p2p1      Link encap:Ethernet  HWaddr 00:1b:21:c0:ff:30  ",
        "          UP BROADCAST MULTICAST  MTU:1500  Metric:1",
        "          Memory:de7c0000-de7dffff ",
        "",
        "p2p2      Link encap:Ethernet  HWaddr 00:1b:21:c0:ff:31  ",
        "          UP BROADCAST MULTICAST  MTU:1500  Metric:1",
        "          Memory:de7e0000-de7fffff ",
        "",
        "p3p1      Link encap:Ethernet  HWaddr 00:1b:21:c1:1c:34  ",
        "          UP BROADCAST MULTICAST  MTU:1500  Metric:1",
        "          Memory:dd7c0000-dd7dffff ",
        "",
        "p3p2      Link encap:Ethernet  HWaddr 00:1b:21:c1:1c:35  ",
        "          UP BROADCAST MULTICAST  MTU:1500  Metric:1",
        "          Memory:dd7e0000-dd7fffff "
    ],
    "changed": false
}
ok: [stirling.devops.develop] => {
    "_ifconfig.stdout_lines": [
        "em1       Link encap:Ethernet  HWaddr f0:4d:a2:40:92:1d  ",
        "          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1",
        "",
```

(continues on next page)

(continued from previous page)

```
        "em2       Link encap:Ethernet  HWaddr f0:4d:a2:40:92:1f  ",
        "          inet addr:10.142.29.161  Bcast:10.142.29.255  Mask:255.255.255.0",
        "          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1",
        "",
        "em3       Link encap:Ethernet  HWaddr f0:4d:a2:40:92:21  ",
        "          UP BROADCAST MULTICAST  MTU:1500  Metric:1",
        "",
        "em4       Link encap:Ethernet  HWaddr f0:4d:a2:40:92:23  ",
        "          UP BROADCAST MULTICAST  MTU:1500  Metric:1",
        "",
        "lo        Link encap:Local Loopback  ",
        "          inet addr:127.0.0.1  Mask:255.0.0.0",
        "          UP LOOPBACK RUNNING  MTU:65536  Metric:1"
    ],
    "changed": false
}

TASK [bootstrap : Show interface details (MacOSX)] ****************************
Sunday 23 July 2017  12:41:51 -0700 (0:00:01.071)       0:00:44.866 ***********
skipping: [dellr510.devops.develop]
skipping: [stirling.devops.develop]

TASK [bootstrap : debug] ******************************************************
Sunday 23 July 2017  12:41:52 -0700 (0:00:01.069)       0:00:45.936 ***********
skipping: [dellr510.devops.develop]
skipping: [stirling.devops.develop]

TASK [bootstrap : Determine SSH host MD5 key fingerprints] ********************
Sunday 23 July 2017  12:41:53 -0700 (0:00:01.068)       0:00:47.004 ***********
changed: [dellr510.devops.develop]
changed: [stirling.devops.develop]

TASK [bootstrap : debug] ******************************************************
Sunday 23 July 2017  12:41:54 -0700 (0:00:01.472)       0:00:48.477 ***********
ok: [dellr510.devops.develop] => {
    "_md5.stdout_lines": [
        "1024 c9:58:58:f3:90:a6:1f:1c:ab:fb:8e:18:42:77:a2:88  root@D-140-142-29-186␣
↪(DSA)",
        "256 a2:61:50:25:6b:c3:02:43:55:a7:35:32:cb:96:f5:82  root@D-140-142-29-186␣
↪(ECDSA)",
        "256 e6:c8:11:ac:48:28:1f:bc:fd:ad:06:f4:0f:26:9e:5b  root@D-140-142-29-186␣
↪(ED25519)",
        "2048 55:ae:94:22:e1:ce:d4:2a:b6:d3:8b:aa:09:70:d1:38  root@D-140-142-29-186␣
↪(RSA)"
    ],
    "changed": false
}
ok: [stirling.devops.develop] => {
    "_md5.stdout_lines": [
        "1024 b1:41:a2:bd:c2:e8:3b:bd:14:3b:3f:7d:eb:e5:ba:10  root@D-140-142-29-161␣
↪(DSA)",
        "256 41:68:1e:59:4e:bd:0c:5b:25:c8:24:60:a8:d6:f1:c6  root@D-140-142-29-161␣
↪(ECDSA)",
        "256 bb:4b:89:f5:6b:45:7c:d3:9e:56:54:ea:8c:1b:79:8f  root@D-140-142-29-161␣
↪(ED25519)",
        "2048 96:95:e2:45:01:d2:45:2e:49:a8:7c:f6:39:28:0a:a5  root@D-140-142-29-161␣
↪(RSA)"
```

(continues on next page)

(continued from previous page)

```
    ],
    "changed": false
}

TASK [bootstrap : Determine SSH host SHA256 key fingerprints] *****************
Sunday 23 July 2017  12:41:55 -0700 (0:00:01.076)       0:00:49.553 ***********
changed: [dellr510.devops.develop]
changed: [stirling.devops.develop]

TASK [bootstrap : debug] ******************************************************
Sunday 23 July 2017  12:41:57 -0700 (0:00:01.471)       0:00:51.025 ***********
ok: [dellr510.devops.develop] => {
    "_sha256.stdout_lines": [
        "ssh-dss dl/W3IeTv3aPGZdfX8q3L0yZE8gAbW6IbHw9uZlyYDU. root@D-140-142-29-186",
        "ecdsa-sha2-nistp256 8qqzBI22OGTY29T3WCKnpIPbyl1K0My9xwPiGEt9PmE. root@D-140-
↪142-29-186",
        "ssh-ed25519 K4Bc5IttYf5WHE2nzuxTr9w8QzTMzIKZYUewvwCcuPc. root@D-140-142-29-
↪186",
        "ssh-rsa rVUD1b6raug2Pp01pJLyWEHzxUfGbzOkwUxvhRzvH30. root@D-140-142-29-186"
    ],
    "changed": false
}
ok: [stirling.devops.develop] => {
    "_sha256.stdout_lines": [
        "ssh-dss EdHHaFS7LRtVqCKzlzYG68OpQNnKqEygWoEoM9lYtWs. root@D-140-142-29-161",
        "ecdsa-sha2-nistp256 3MicWfvhufEiPRiANS43Z/7MbcHHTythyOAhYluyD+w. root@D-140-
↪142-29-161",
        "ssh-ed25519 gT0duOWxArehJR08iR0iFO4gDUqDCjT6P+lJYPT0MwI. root@D-140-142-29-
↪161",
        "ssh-rsa MQl68HQR5Oip9MPlozLddlXA9Emcz9QTJLk0IJgVJOs. root@D-140-142-29-161"
    ],
    "changed": false
}

TASK [bootstrap : Determine SSH host SHA256 key fingerprints] *****************
Sunday 23 July 2017  12:41:58 -0700 (0:00:01.072)       0:00:52.097 ***********
skipping: [dellr510.devops.develop]
skipping: [stirling.devops.develop]

TASK [bootstrap : debug] ******************************************************
Sunday 23 July 2017  12:41:59 -0700 (0:00:01.069)       0:00:53.167 ***********
skipping: [dellr510.devops.develop]
skipping: [stirling.devops.develop]

RUNNING HANDLER [bootstrap : Update timezone] *********************************
Sunday 23 July 2017  12:42:00 -0700 (0:00:01.062)       0:00:54.229 ***********
changed: [dellr510.devops.develop]
changed: [stirling.devops.develop]

PLAY RECAP ********************************************************************
dellr510.devops.develop    : ok=20   changed=9    unreachable=0    failed=0
stirling.devops.develop    : ok=20   changed=10   unreachable=0    failed=0

Sunday 23 July 2017  12:42:02 -0700 (0:00:02.078)       0:00:56.307 ***********
===============================================================================
bootstrap : Make sure required APT packages are present (Debian) ------- 15.02s
bootstrap : Ensure Debian chrony package is installed ------------------- 2.68s
```

(continues on next page)

(continued from previous page)

```
bootstrap : Define variable with ansible public key --------------------- 2.08s
bootstrap : Update timezone --------------------------------------------- 2.08s
bootstrap : Set timezone variables -------------------------------------- 2.04s
bootstrap : Copy sudoers template to temporary file --------------------- 1.91s
bootstrap : Ensure ansible public key in authorized_keys ---------------- 1.71s
bootstrap : Check for Broadcom device 14e4:43b1 ------------------------- 1.71s
bootstrap : Ensure Broadcom wireless kernel in place -------------------- 1.63s
bootstrap : Ensure chrony is running on Debian -------------------------- 1.60s
bootstrap : Verify sudoers before replacing ----------------------------- 1.51s
bootstrap : Determine SSH host MD5 key fingerprints --------------------- 1.47s
bootstrap : Determine SSH host SHA256 key fingerprints ------------------ 1.47s
bootstrap : Verify that the sudo group exists --------------------------- 1.46s
bootstrap : Back up sudoers file ---------------------------------------- 1.40s
bootstrap : Show interface details (Debian) ----------------------------- 1.40s
bootstrap : Make sure required APT packages are present (RedHat) -------- 1.17s
bootstrap : debug ------------------------------------------------------- 1.08s
Include codename-specific variables ------------------------------------- 1.08s
bootstrap : debug ------------------------------------------------------- 1.07s
```

4.4 Integration into Working Inventory

After the `bootstrap` role has been applied, the host should now be ready for Ansible control. Create the host's playbook and ensure that any required variables are added to a more permanant inventory file. If this is anything beyond a basic development (i.e., `local`) deployment, create a new private customization repository (this will be discussed in more detail in Section *Customizing a Private Deployment*).

> **Attention:** Do not forget to add the host being bootstrapped to the `all` group in the inventory. While it may be accessible by simply being listed in the `children` subgroup with an `ansible_host` value like shown earlier, its `host_vars` file will not be loaded unless it is included in the `all` group.
>
> This problem would go away if all of the variables formerly placed in `host_vars` files were moved directly into the inventory files instead.

Set up the following to ensure that the host will be functional and under Ansible control for:

- `iptables` rules specified in `tcp_ports`, `udp_ports`, and/or `custom_rules` that will be templated into the rules files. These should lock the host down, while allowing access to hosts on internal VLANs for remote Ansible control, accessing internal repositories or source archives, etc.
- `/etc/network/interfaces` template or variables necessary to define all desired network interfaces. This file should start out reflecting the network settings used to install the system and provide access to the internal VLAN.
- Any `custom_hosts` that need to be defined in `/etc/hosts` to ensure connectivity out to remote systems (e.g., to an internal Git source repository host that is required to get private repositories, serve internal packages, etc.)

To separate these bootstrapping settings from normal settings, use a `children` sub-group named `bootstrap` for the host being set up. In this case, we are focusing on a host named `stirling.devops.develop`.

```
---

# File: inventory/servers/nodes.yml
```

(continues on next page)

(continued from previous page)

```
servers:
  vars:
    ansible_port: 8422
  hosts:
    'other-hosts-not-shown...':
    'stirling.devops.develop':
      #ansible_host: '10.142.29.182'
      #ansible_host: '140.142.29.161'
      ansible_user: 'ansible'
      zone_iface:
        'public': 'em2'
        'prisem': 'em1'
        'develop': 'em2'
        'swarm': 'vboxnet1'
        'consul': 'vboxnet1'
        'yellow_bridge': 'em1'
        'purple_bridge': 'em1'
      zones:
        - develop
      net:
        iface:
          'em1':
            #ip: '140.142.29.161'
            #ip: '140.142.13.171'
            #cidr_bits: 27
            ip: '0.0.0.0'
          'em2':
            ip: '10.142.29.161'
            netmask: '255.255.255.0'
            cidr_bits: 24
          'em3':
            ip: '10.3.0.1'
            netmask: '255.255.255.0'
            cidr_bits: 24
          'em4':
            ip: '10.4.0.1'
            netmask: '255.255.255.0'
            cidr_bits: 24
      tcp_ports: [ 9999 ]
      udp_ports: [ ]
      custom_hosts:
        - '10.142.29.98  source.devops.develop'
        - '10.142.29.115  eclipse.devops.develop'
  children:
    bootstrap:
      vars:
        ansible_port: 22
        http_proxy: ''
        https_proxy: ''
      hosts:
        'stirling.devops.develop':
            ansible_host: '10.142.29.161'
            private_develop: "{{ lookup('env','GIT') }}/private-develop"
            private_repository: "git@git.devops.develop:/var/opt/private-develop.git"
            private_repository_hostkey: "2048␣
↪78:82:74:66:56:93:a7:9d:54:ce:05:ed:8a:0d:fa:b4  root@git.devops.develop (RSA)"
```

(continues on next page)

(continued from previous page)

```
            private_repository_hostname: "git.devops.develop"
            ansible_ssh_private_key_file: "{{ lookup('dims_function', 'get_ssh_
↪private_key_file {{ ansible_user }} {{ private_develop }}') }}"
            install_ssh_keypair: true
            bootstrap_private: true
            artifacts_url: 'http://source.devops.develop/source/'
            ssh_config_hosts:
              - hostname_short: 'git'
                hostname: git.devops.develop
                user: git
                port: 8422

# vim: ft=ansible :
```

As for the host playbook, here is an example of a complete playbook for a virtual machine manager host with development capabilities.

```
---

# File: v2/playbooks/hosts/stirling.devops.develop.yml

- name: Configure host "stirling.devops.develop"
  hosts: stirling.devops.develop

  vars:
    playbooks_root: "{{ lookup('env', 'PBR') }}"
    dims_private: "{{ lookup('env', 'GIT') }}/private-{{ deployment }}"
    https_proxy: 'https://127.0.0.1:8000'

  vars_files:
   - "{{ playbooks_root }}/vars/global.yml"
   - "{{ playbooks_root }}/vars/trusty.yml"

  remote_user: "ansible"
  become: yes

  roles:
    - { role: base, packages_upgrade: true }
    - { role: hosts }
    - { role: dns }
    - { role: dims-ci-utils }
    - { role: python-virtualenv, use_sphinx: true }
    - { role: ansible-server }
    - { role: docker }
    - { role: consul }
    - { role: packer }
    - { role: vagrant }
    - { role: virtualbox }
    - { role: vncserver }
    - { role: nginx }
    - { role: byobu }
    - { role: apache-directory-studio }

  handlers:
   - include_tasks: "{{ handlers_path }}/restart_services.yml"

# vim: ft=ansible :
```

Some roles of note (highlighted above) are the following:

- `ansible-server` will set the host up for serving as an Ansible control host. This includes installing shared public roles that are being used for installing certain services, cloning the `ansible-dims-playbooks` repository (`master` branch by default), and installing the `ansible` user SSH key pair.
- `dns` will set up "split-horizon" DNS service, serving an internal domain used by virtual machines and the hypervisor host for looking up IP addresses on internal interfaces connected to private VLANs and/or virtual networks. The zone(s) that will be served by this host are defined by the `zones` array, which uses mappings to dictionaries holding interface information in order to derive the name-to-IP mappings for each zone.
- The roles `vagrant`, `packer`, and `virtualbox` set the host up for serving as a Virtualbox hypervisor that can use DIMS helper scripts for automated creation of Vagrant boxes. (This capability is useful for development and testing, but is not recommended for "production" use.)
- `vncserver` will configure the host for remotely running graphical user interface programs (e.g., the `virtualbox` management interface) using VNC tunneled over SSH. (It also creates a helper script on the control host running this playbook to facilitate setting up the SSH tunnel that we will use to manually create virtual machines in the following section).
- `nginx` sets up a reverse proxy web server that can be used to serve box files, operating system installation ISO image files, and packaged artifacts cached from public sources or non-public sources (e.g., from an internal Jenkins build server).

Note: As a ballpark estimate of time-to-deploy for an initial virtual machine host server, using a Dell R710 with a 1 Gbps ethernet connection, the initial Ubuntu Kickstart operating system installation took approximately 30 minutes. The `bootstrap` playbook to get the system to password-less Ansible control took about another 5 minutes. The first complete run of the host playbook (which, including the lengthy `python-virtualenv` build task) adds over a thousand new packages, took about 45 minutes to complete. This is a total time of just under 1.5 hours (and these steps could be done in parallel with multiple hosts with just a small additional overhead for setting variables for each host.)

4.5 Normal Playbook Operations

Now run the host's playbook to fully configure it and update packages. This can be done from the Ansible control host being used to remotely bootstrap the new server, or from within the server itself. If the desire is to hand the newly bootstrapped system off to a production operations group, the normal means of administering the system may be for them to log in to it using SSH and run the host's playbook locally. To make this easier (or for developers to keep their own systems up to date), a helper command `run.playbook` is set up. Running just this command will execute the full playbook. To only execute part of the playbook, use the `--tags` option to select the set of tags you wish to apply as described in Section *Tags on Tasks*. For example, to just apply any updated packages, use `run.playbook --tags updates`, or to just apply changes to `iptables` rules files and reload them, use `run.playbook --tags iptables`.

To run the playbook using Ansible directly, performing both of the example tasks just listed at once, the command would look like this:

```
$ ansible-playbook $DIMS_PRIVATE/playbooks/hosts/dellr510.devops.develop --tags␣
↪updates,iptables
```

4.6 Validating VNC over SSH Tunnelling

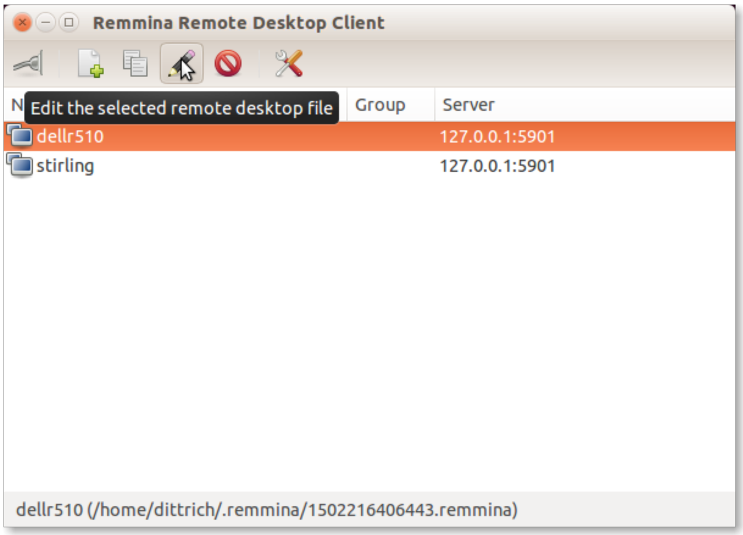
The last thing we will do to validate our VM hypervisor and Ansible control host is ready to use for managing virtual machines is to establish an SSH tunnelled VNC connection using Remmina. Run the helper script to establish the tunnel:

```
$ vnc.dellr510.devops.develop
[+] X11 forwarding connection to dellr510.devops.develop established.
[+] Remote X11 DISPLAY is :1
[+] Configure your VNC client to use 127.0.0.1:5901
[+] Use CTRL-C or ENTER to break tunnel connection...
```

Now run the following command in a shell window, or use the task bar to run the Remmina application:

```
$ remmina &
```

Note: The `&` at the end of the command line puts the application into the background. Remmina, like other X11 or Gnome applications, does not use the command line for keyboard input. Instead, it uses the X11 graphical user interface features. Leaving the `&` off will make the terminal window appear to "hang" as the prompt will not be returned until the Remmina graphical application quits. For more details, see How to clean launch a GUI app via the Terminal (so it doesn't wait for termination)?


Remmina Remote Desktop Client
Edit the selected remote desktop file

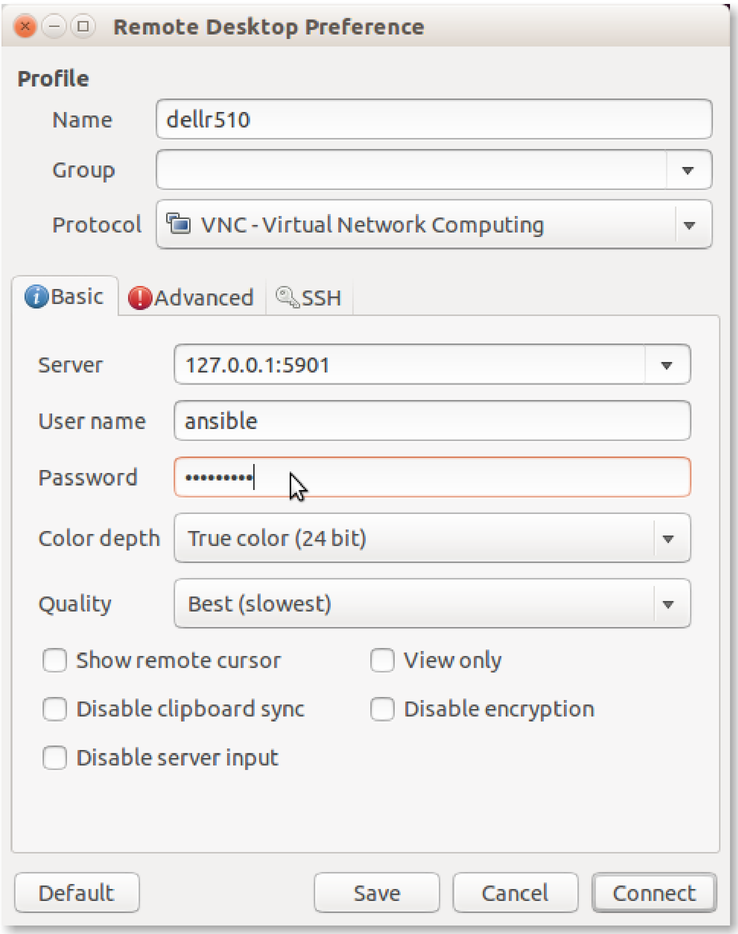
N	Group	Server
dellr510		127.0.0.1:5901
stirling		127.0.0.1:5901

dellr510 (/home/dittrich/.remmina/1502216406443.remmina)

Fig. 1: Remmina Main Screen

Select **Create a new remote desktop file** (the sheet of paper with a green + sign) if this is the first time you are running Remmina. In this case, a connection was already created so we will instead select **Edit** (the pencil icon) to edit the settings. Save them when you are done to get back to the main menu.

Note: The password to use here is one set by the variable `vnc_server_default` in the `roles/vncserver/defaults/main.yml` file. As long as the VNC server is bound to `localhost`, the risk is limited to the local system. For improved security, set this password to something strong by over-riding the default password with this variable in a private customization repository and/or Ansible Vault file using the techniques described in Section *Customizing a Private Deployment*.



Remote Desktop Preference

Profile

Name: dellr510

Group:

Protocol: VNC - Virtual Network Computing

Server: 127.0.0.1:5901

User name: ansible

Password: •••••••••

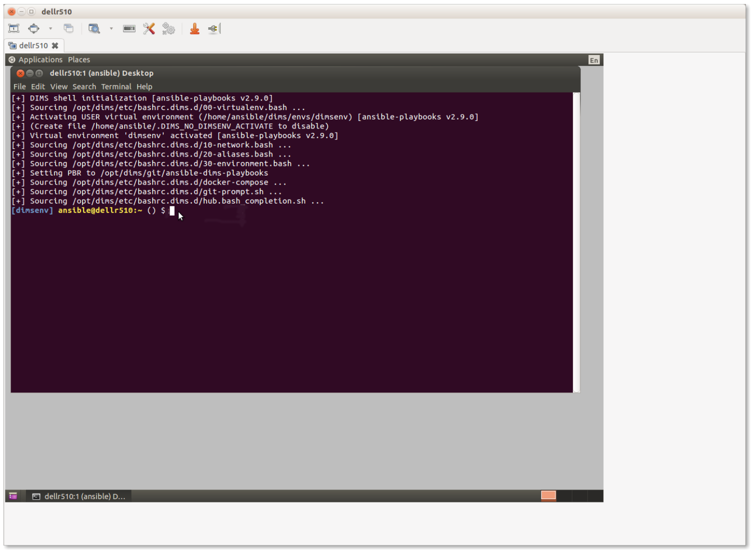
Color depth: True color (24 bit)

Quality: Best (slowest)

□ Show remote cursor
□ View only
□ Disable clipboard sync
□ Disable encryption
□ Disable server input

Fig. 2: Remmina Edit Screen

When you then select the item (`dellr510` in this case) and press **Open the connection to the selecetd remote desktop file** (the icon that looks like a light switch on the far left of the icon bar), you should now have a graphical desktop with a terminal window open on the remote host as seen here:



```
[+] DIMS shell initialization [ansible-playbooks v2.9.0]
[+] Sourcing /opt/dims/etc/bashrc.dims.d/00-virtualenv.bash ...
[+] Activating USER virtual environment (/home/ansible/dims/envs/dimsenv) [ansible-playbooks v2.9.0]
[+] (Create file /home/ansible/.DIMS_NO_DIMSENV_ACTIVATE to disable)
[+] Virtual environment 'dimsenv' activated [ansible-playbooks v2.9.0]
[+] Sourcing /opt/dims/etc/bashrc.dims.d/10-network.bash ...
[+] Sourcing /opt/dims/etc/bashrc.dims.d/20-aliases.bash ...
[+] Sourcing /opt/dims/etc/bashrc.dims.d/30-environment.bash ...
[+] Setting PBR to /opt/dims/git/ansible-dims-playbooks
[+] Sourcing /opt/dims/etc/bashrc.dims.d/docker-compose ...
[+] Sourcing /opt/dims/etc/bashrc.dims.d/git-prompt.sh ...
[+] Sourcing /opt/dims/etc/bashrc.dims.d/hub.bash_completion.sh ...
[dimsenv] ansible@dellr510:~ () $
```

Fig. 3: Initial Remmina VNC Connection

4.7 Creating VMs

The Ansible control host that was just set up can now be used to control a set of virtual machines, bare metal hosts, or a combination. It all depends on what services you wish to provide and how you chose to deploy them.

There are several options for creating a hybrid "private-cloud" comprised from a combination of bare-metal hosts, virtual machine hosts, and containerized microservices. This flexibility comes at a cost in added complexity and configuration management, but does afford for better linear horizontal scalability and/or addition of compute or storage resources as the system grows in size.

Hint: For the bigger picture of architectural design options considered while designing and building the DIMS system components, see Section DIMS architectural design of dimsad.

Figure *Pure Virtual Machine Architecture* shows a design similar to that being described in this and the previous chapters. The *Host* is shown at the bottom, comprised of a highly-provisioned server, a base operating system and a virtual machine hypervisor. Each virtual machine *Guest* is then created and installed with its own combination of base operating system, libraries and binaries, and application software. In this illustration, we see a single physical computer with a total of six servers (4 Ubuntu Linux, 1 Red Hat Enterprise Linux, and 1 Debian Linux).

The deployment we are currently creating is even simpler than Figure *Pure Virtual Machine Architecture*. There is the one bare-metal server acting as the Ansible control host and Virtualbox hypervisor (`dellr510`) and just two virtual machine guests (`yellow` and `purple`, a.k.a. the `trident` group). These guests will use bridged interfaces so they each have an Internet-facing IP address and domain name, as well as a private virtual LAN that is shared with the host for Ansible control and administration. For increased security, the bare-metal VM host will only be accessible through an internal VLAN.

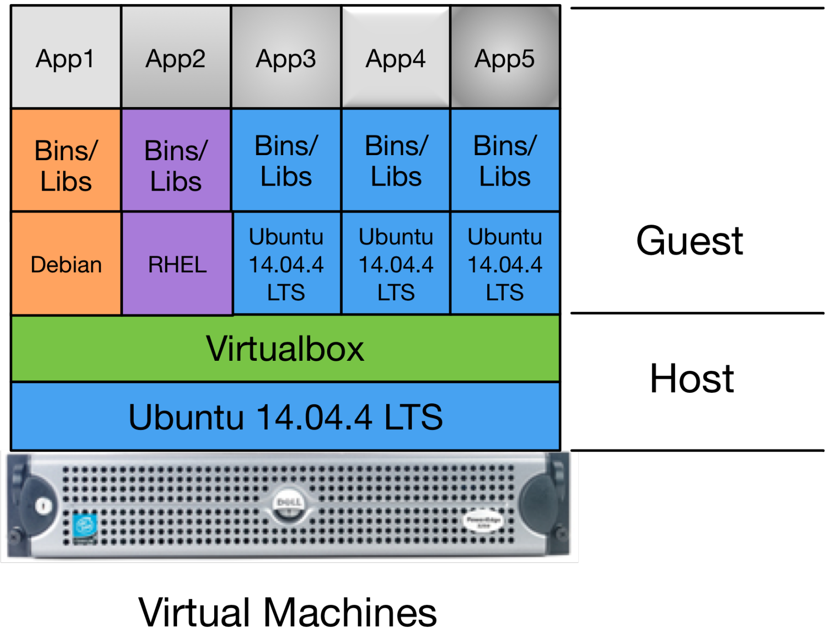


Fig. 4: Pure Virtual Machine Architecture

4.7.1 Manual Installation of Virtual Machines

This section walks through the process of manually creating two Debian 8.5 virtual machines to serve as Trident trust group portal servers. This deployment combines all of the Trident related services into one virtual machine. One of the two vitual machines (`yellow`) will serve as the "production" portal, and the other identical system (`purple`) will serve as a development/test server. The latter can be used to experiment with upgrades, test Ansible playbook changes, train system administrators and trust group administrators.

Start the Virtualbox management GUI in the Remmina VNC window.

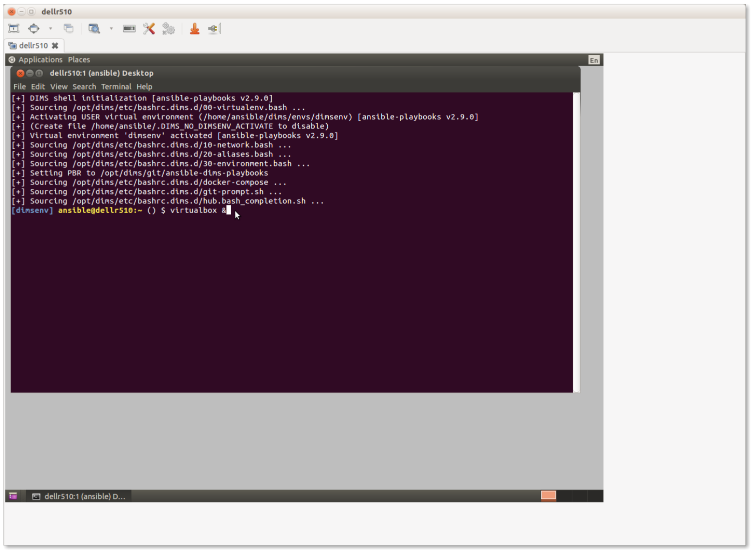
This should bring up the Virtualbox management GUI.

Select **New** to create a new virtual machine. Most tabs have a **Next>** button to go to the following tab, or select **Settings** after highlighting the VM you want to configure, or and press the **Right mouse button** and chose **Settings. . .** or use the keyboard shortcut **CTRL-S**.

Individual groupings of settings (e.g., **System** for boot order, processor settings, etc., **Storage** for virtual hard drives, **Network** for NICs) are on the left of the **Settings** panel.

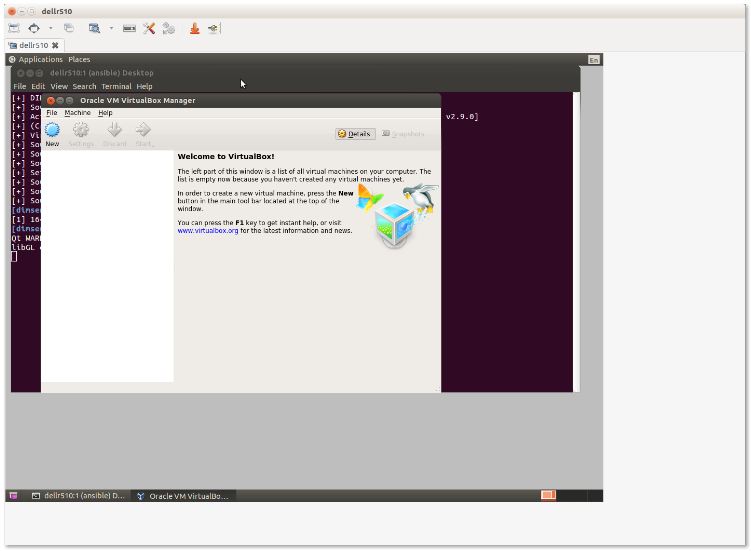
Navigate through the menus to set the following attributes:

- Set Name: `yellow`
- Set Type: **Linux**
- Set Version: **Ubuntu (64-bit)**
- Set memory (e.g., **4096 MB**)
- Create a virtual disk, type **VDI (VirtualBox Disk Image)**, dynamically allocated, making it generously large in relation to available disk space to provide adequate storage space for Trident upload files (e.g., **200GB**).
- Configure three NICs:


dellr510
dellr510
Applications Places
En
dellr510:1 (ansible) Desktop
File Edit View Search Terminal Help
dellr510:1 (ansible) D...

```
[+] DIMS shell initialization [ansible-playbooks v2.9.0]
[+] Sourcing /opt/dims/etc/bashrc.dims.d/00-virtualenv.bash ...
[+] Activating USER virtual environment (/home/ansible/dims/envs/dimsenv) [ansible-playbooks v2.9.0]
[+] (Create file /home/ansible/.DIMS_NO_DIMSENV_ACTIVATE to disable)
[+] Virtual environment 'dimsenv' activated [ansible-playbooks v2.9.0]
[+] Sourcing /opt/dims/etc/bashrc.dims.d/10-network.bash ...
[+] Sourcing /opt/dims/etc/bashrc.dims.d/20-aliases.bash ...
[+] Sourcing /opt/dims/etc/bashrc.dims.d/30-environment.bash ...
[+] Setting PBR to /opt/dims/git/ansible-dims-playbooks
[+] Sourcing /opt/dims/etc/bashrc.dims.d/docker-compose ...
[+] Sourcing /opt/dims/etc/bashrc.dims.d/git-prompt.sh ...
[+] Sourcing /opt/dims/etc/bashrc.dims.d/hub.bash_completion.sh ...
[dimsenv] ansible@dellr510:~ () $ virtualbox &
```

Fig. 5: Running Virtualbox management GUI over VNC


dellr510
dellr510
Applications Places
En
dellr510:1 (ansible) Desktop
File Edit View Search Terminal Help
Oracle VM VirtualBox Manager
File Machine Help
New
Settings
Discard
Start
Details
Snapshots
dellr510:1 (ansible) D...
Oracle VM VirtualBo...

```
[+] DI
[+] So
[+] Ac
[+] (C
[+] Vi
[+] So
[+] So
[+] So
[+] Se
[+] So
[+] So
[+] So
[dimse
[1] 16
[dimse
Qt WAR
libGL
```

```
v2.9.0]
```

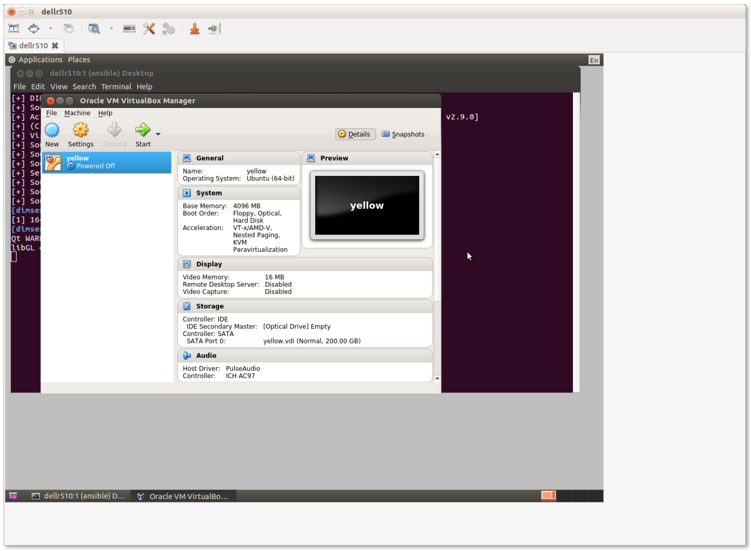
Welcome to VirtualBox!

The left part of this window is a list of all virtual machines on your computer. The list is empty now because you haven't created any virtual machines yet.

In order to create a new virtual machine, press the **New** button in the main tool bar located at the top of the window.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for the latest information and news.

Fig. 6: Virtualbox management GUI



General
Name: yellow
Operating System: Ubuntu (64-bit)

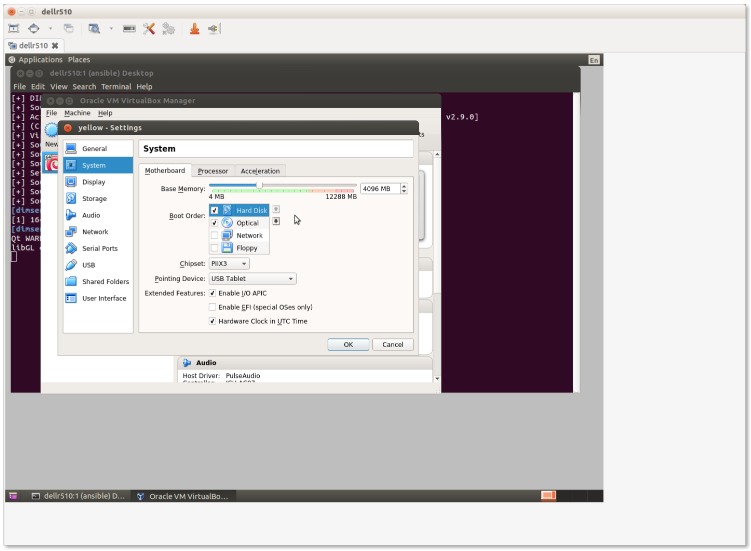
System
Base Memory: 4096 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

Display
Video Memory: 16 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage
Controller: IDE
IDE Secondary Master: [Optical Drive] Empty
Controller: SATA
SATA Port 0: yellow.vdi (Normal, 200.00 GB)

Audio
Host Driver: PulseAudio
Controller: ICH AC97

Fig. 7: Initial `yellow` VM



System

Base Memory: 4096 MB (4 MB – 12288 MB)

Boot Order:
☑ Hard Disk
☑ Optical
☐ Network
☐ Floppy

Chipset: PIIX3

Pointing Device: USB Tablet

Extended Features:
☑ Enable I/O APIC
☐ Enable EFI (special OSes only)
☑ Hardware Clock in UTC Time

Host Driver: PulseAudio

Fig. 8: VM System Settings

  - Adapter **1** should be attached to **NAT** to provide host-only access with NAT to get to the Internet while setting up the VM.
  - Adapter **2** should be attached to **Bridged Adapter**, name `em2` in this case. (This is the host NIC attached to the internal VLAN in this configuration). This interface in the guest (`eth1`) will be used for local Ansible control and communication with internal hosts.
  - Adapter **3** should be attached to **Bridged Adapter**, name `em1` in this case. (This is the host NIC attached to the Internet in this configuration, which will be set to `0.0.0.0` to prevent direct communication from the Internet to the VM host using this interface). This interface in the guest (`eth2`) will have the public IP address for the Trident portal, email delivery, etc.

- Set the system boot order to be **Hard Disk** first, followed by **Optical** drive. The first boot with an empty hard drive will boot from the **Optical** drive, while subsequent reboots will use the operating system installed on the **Hard Disk**.
- Increase the number of CPUs (for a 16 core VM host, 3 or 4 cores is reasonable.)

Note: All of these settings can be tuned later on if it is determined that they are too low (or too high). Use a program like `htop` on the virtual machine host to watch things like CPU saturation, memory saturation, swap usage, etc.

After configuring the first VM `yellow`, produce a full clone of the VM and name it `purple`. This will be the backup Trident server. Check the box to regenerate MAC addresses for the network interfaces to ensure that they are separable at the packet level in case network diagnostics need to be performed using `tcpdump` or other network tools.

Once both of the VMs are set up, start them to boot from the Debian installation ISO attached to the virtual DVD drive.

Note: We are not using Kickstart here, as we did for the baremetal host in Section *Bootstrapping DigitalOcean Droplets*, which means that a number of steps that were automatically performed during system installation will need to be performed manually. This is an area of automation that needs further work to unify and standardize the boot process using Kickstart from Jinja templates and inventory variables, allowing a consistent, configurable, repeatable, and much faster system setup. This will result in time and cost savings that scale better and help new teams more quickly deploy a full system.

- Use LVM on the entire drive, with separate partitions for `/tmp`, `/home`, and `/var`.
- Choose **Debian desktop environment**, with **Gnome**, de-select **print server** and select **SSH server**, leaving **standard system utilities** selected, and press **Tab** and **Enter** to **Continue**.
- Create the `ansible` account using the password you created for this deployment. Also set the `root` password (ideally to a different password than the `ansible` account, to be used for emergency console access when and if something disables access to the `ansible` account.)

At the end of the operating system installation process, it will ask you to reboot. The guest should then show the Grub boot menu and proceed to boot into Debian, presenting the login screen when the system is up and running.

4.7.2 Bootstrapping the New VMs

Before you can perform the bootstrapping process using Ansible, you must configure at least one network interface on each VM guest (as well as setting an IP address in the same network block on the bridged interface of the host) to allow host-to-guest SSH access.

Manually edit the `/etc/network/interfaces` file to configure the initial `eth1` NIC to have the IP addresses assigned for the hosts in the inventory file. Bring the interface up using `ifup eth1` and test after setting up all of the interfaces using the same steps as shown in Section *Establishing Full Internet Connectivity*.

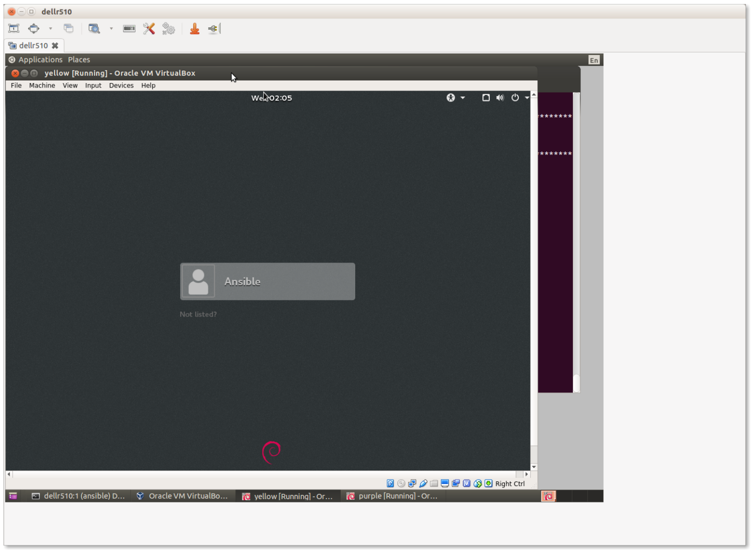
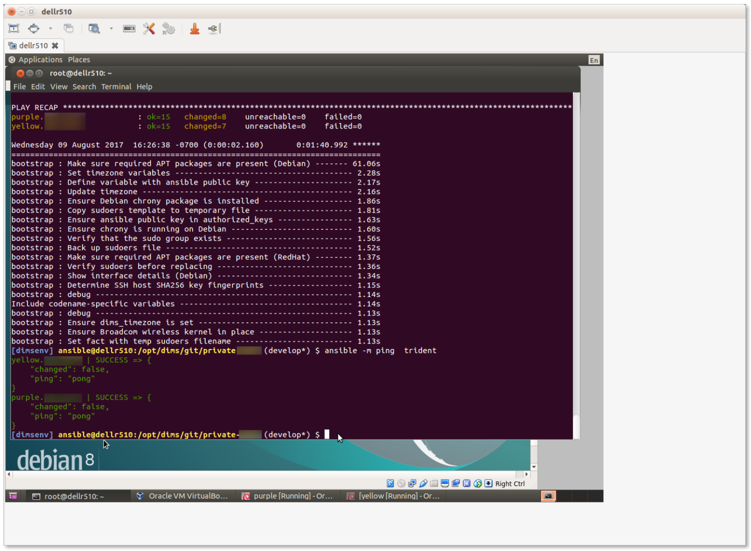


Fig. 9: Login screen for `yellow` VM

Once connectivity has been verified, apply the `bootstrap.yml` playbook as shown in Section *Bootstrapping Full Ansible Control*, using the `trident` group this time to bootstrap both VMs at the same time.



```
PLAY RECAP *******************************************************************************************************
purple.[redacted]            : ok=15   changed=8    unreachable=0    failed=0
yellow.[redacted]            : ok=15   changed=7    unreachable=0    failed=0

Wednesday 09 August 2017  16:26:38 -0700 (0:00:02.160)       0:01:40.992 ******
===============================================================================
bootstrap : Make sure required APT packages are present (Debian) ------- 61.06s
bootstrap : Set timezone variables ------------------------------------- 2.28s
bootstrap : Define variable with ansible public key -------------------- 2.17s
bootstrap : Update timezone -------------------------------------------- 2.16s
bootstrap : Ensure Debian chrony package is installed ------------------ 1.86s
bootstrap : Copy sudoers template to temporary file -------------------- 1.81s
bootstrap : Ensure ansible public key in authorized_keys --------------- 1.63s
bootstrap : Ensure chrony is running on Debian ------------------------- 1.60s
bootstrap : Verify that the sudo group exists -------------------------- 1.56s
bootstrap : Back up sudoers file --------------------------------------- 1.52s
bootstrap : Make sure required APT packages are present (RedHat) -------- 1.37s
bootstrap : Verify sudoers before replacing ---------------------------- 1.36s
bootstrap : Show interface details (Debian) ---------------------------- 1.34s
bootstrap : Determine SSH host SHA256 key fingerprints ----------------- 1.15s
bootstrap : debug ------------------------------------------------------ 1.14s
Include codename-specific variables ------------------------------------ 1.14s
bootstrap : debug ------------------------------------------------------ 1.13s
bootstrap : Ensure dims_timezone is set -------------------------------- 1.13s
bootstrap : Ensure Broadcom wireless kernel in place ------------------- 1.13s
bootstrap : Set fact with temp sudoers filename ------------------------ 1.13s
[dimsenv] ansible@dellr510:/opt/dims/git/private[redacted] (develop*) $ ansible -m ping  trident
yellow.[redacted] | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
purple.[redacted] | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
[dimsenv] ansible@dellr510:/opt/dims/git/private-[redacted] (develop*) $
```

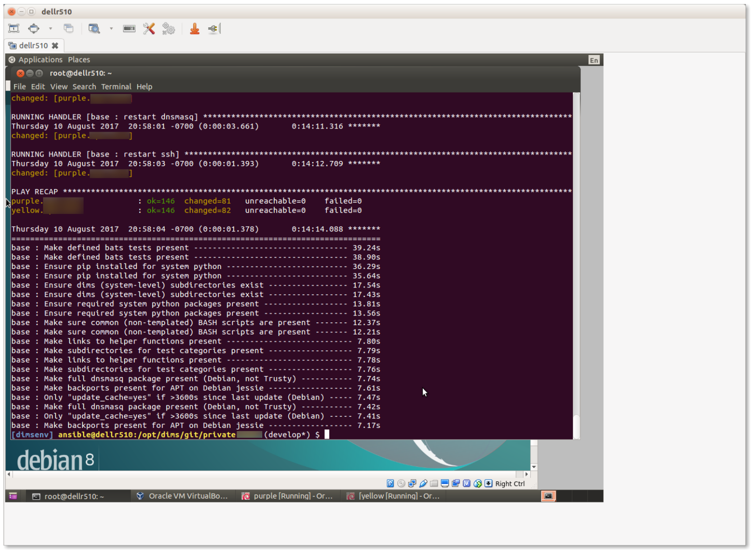
Fig. 10: Bootstrapping the `trident` group

4.7.3 Initial Provisioning of the New VMs

Lastly, we will run the initial provisioning steps to install and configure the two new VMs. For the purposes of this example, we will start by only applying the `base` role tasks to make sure the fundamentals of our customized configuration will work. The command we use is:

```
$ ansible-playbook $GIT/private-develop/master.yml --tags base --limit trident
```

Having applied the `base` role, network interfaces are set up, `iptables` rules are in place, `/etc/hosts` file and



```
changed: [purple.[redacted]

RUNNING HANDLER [base : restart dnsmasq] *************************************************************************
Thursday 10 August 2017  20:58:01 -0700 (0:00:03.661)       0:14:11.316 *******
changed: [purple.[redacted]]

RUNNING HANDLER [base : restart ssh] *****************************************************************************
Thursday 10 August 2017  20:58:03 -0700 (0:00:01.393)       0:14:12.709 *******
changed: [purple.[redacted]]

PLAY RECAP *******************************************************************************************************
purple.[redacted]            : ok=146  changed=81   unreachable=0    failed=0
yellow.[redacted]            : ok=146  changed=82   unreachable=0    failed=0

Thursday 10 August 2017  20:58:04 -0700 (0:00:01.378)       0:14:14.088 *******
===============================================================================
base : Make defined bats tests present --------------------------------- 39.24s
base : Make defined bats tests present --------------------------------- 38.90s
base : Ensure pip installed for system python -------------------------- 36.29s
base : Ensure pip installed for system python -------------------------- 35.64s
base : Ensure dims (system-level) subdirectories exist ----------------- 17.54s
base : Ensure dims (system-level) subdirectories exist ----------------- 17.43s
base : Ensure required system python packages present ------------------ 13.81s
base : Ensure required system python packages present ------------------ 13.56s
base : Make sure common (non-templated) BASH scripts are present ------- 12.37s
base : Make sure common (non-templated) BASH scripts are present ------- 12.21s
base : Make links to helper functions present --------------------------- 7.80s
base : Make subdirectories for test categories present ------------------ 7.79s
base : Make links to helper functions present --------------------------- 7.78s
base : Make subdirectories for test categories present ------------------ 7.76s
base : Make full dnsmasq package present (Debian, not Trusty) ----------- 7.74s
base : Make backports present for APT on Debian jessie ------------------ 7.61s
base : Only "update_cache=yes" if >3600s since last update (Debian) ----- 7.47s
base : Make full dnsmasq package present (Debian, not Trusty) ----------- 7.42s
base : Only "update_cache=yes" if >3600s since last update (Debian) ----- 7.41s
base : Make backports present for APT on Debian jessie ------------------ 7.17s
[dimsenv] ansible@dellr510:/opt/dims/git/private[redacted](develop*) $
```

Fig. 11: Applying base role to trident group

DNS resolution are configured, and packages have been updated. This would be a good time to reboot both systems to ensure everything is applied and functions. You can use Ansible ad-hoc mode to do this with the command:

```
$ ansible -m shell --become -a 'shutdown -r now' trident`
```

After a minute or two, you can test connectivity again with the command:

```
$ ansible -m shell -a 'uptime' trident`
purple.devops.develop | SUCCESS | rc=0 >>
 14:22:33 up 0 min,  1 user,  load average: 0.86, 0.22, 0.07

yellow.devops.develop | SUCCESS | rc=0 >>
 14:22:33 up 0 min,  1 user,  load average: 0.79, 0.25, 0.09
```

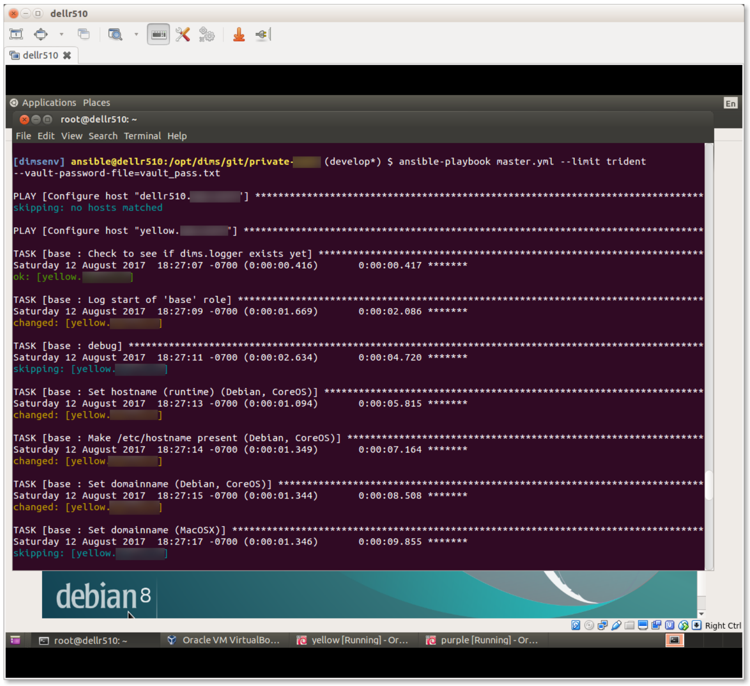
At this point, the hosts are ready for application of their full playbooks. Use `--limit trident` when running the `master.yml` playbook to only operate on the two VMs in question.

Note: If Ansible Vault is being used to encrypt any secrets on disk, you will need to either provide the password using the `--ask-vault-pass` command line option or provide a path to the Vault password file using the `--vault-password-file` command line option. We will use the latter in this example:

Attention: The `nginx` role is designed to support use of Letsencrypt for SSL certificate generation. Because Letsencrypt imposes a limit on the number of certificates that can be generated for a given DNS domain name per week, the default is to use the "staging" facility (i.e., the default is `certbot_staging: yes` globally.) It may take a few full playbook runs to ensure that all variables are defined and set properly, which could exhaust the limit of certificates if the default was to generate real certificates each time the `nginx` role gets applied.

After you are sure things are working properly, edit the `inventory/trident/nodes.yml` file and change the setting to `certbot_staging: no` and apply the `nginx` role one more time to get valid certificates.

Once valid certificates have been generated once, you can create a backup that can be restored later for development testing purposes in case you have to destroy the `/etc/letsencrypt` directory and start again (as occurs when



```
[dimsenv] ansible@dellr510:/opt/dims/git/private-[redacted] (develop*) $ ansible-playbook master.yml --limit trident
--vault-password-file=vault_pass.txt

PLAY [Configure host "dellr510.[redacted]"] ****************************************************************************
skipping: no hosts matched

PLAY [Configure host "yellow.[redacted]"] *****************************************************************************

TASK [base : Check to see if dims.logger exists yet] *****************************************************************
Saturday 12 August 2017  18:27:07 -0700 (0:00:00.416)       0:00:00.417 *******
ok: [yellow.[redacted]]

TASK [base : Log start of 'base' role] *******************************************************************************
Saturday 12 August 2017  18:27:09 -0700 (0:00:01.669)       0:00:02.086 *******
changed: [yellow.[redacted]]

TASK [base : debug] **************************************************************************************************
Saturday 12 August 2017  18:27:11 -0700 (0:00:02.634)       0:00:04.720 *******
skipping: [yellow.[redacted]]

TASK [base : Set hostname (runtime) (Debian, CoreOS)] ****************************************************************
Saturday 12 August 2017  18:27:13 -0700 (0:00:01.094)       0:00:05.815 *******
changed: [yellow.[redacted]]

TASK [base : Make /etc/hostname present (Debian, CoreOS)] ************************************************************
Saturday 12 August 2017  18:27:14 -0700 (0:00:01.349)       0:00:07.164 *******
changed: [yellow[redacted]]

TASK [base : Set domainname (Debian, CoreOS)] ************************************************************************
Saturday 12 August 2017  18:27:15 -0700 (0:00:01.344)       0:00:08.508 *******
changed: [yellow.[redacted]]

TASK [base : Set domainname (MacOSX)] ********************************************************************************
Saturday 12 August 2017  18:27:17 -0700 (0:00:01.346)       0:00:09.855 *******
skipping: [yellow.[redacted]]
```

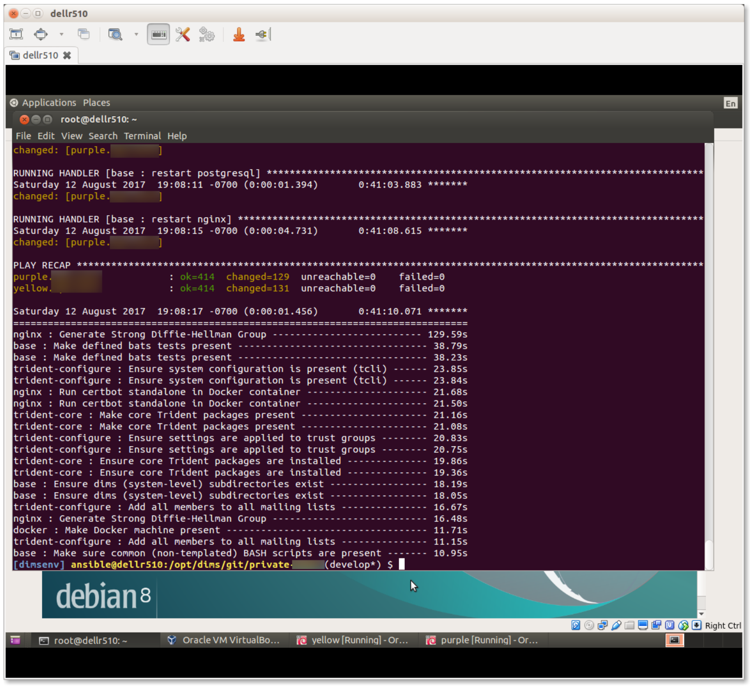
Fig. 12: Applying full playbook to `trident` group

using Vagrants and doing `vagrant destroy`, or terminating virtual machines in cloud service providers.) This process is described in Chapter *Creating a Backup*.

This completes the installation of the two VMs.

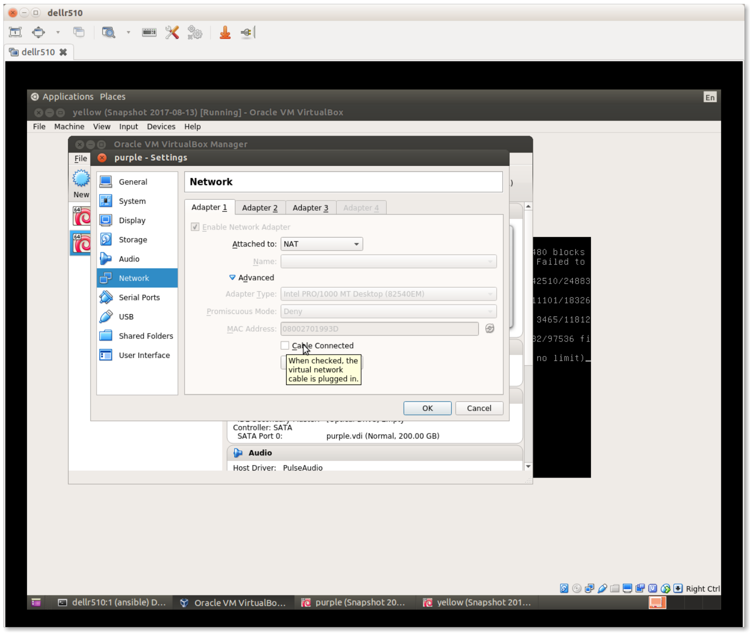
Attention: As these VMs were created using a NAT interface, but are meant to normally operate using a bridged adapter for Internet facing access to the portal and for email processing, one last configuration change is to disable the `eth0` NAT interface so its DHCP assigned default route does not conflict with the default gateway setting of the `eth2` interface. To do this, you will need to go the **Settings** tab, then unselect **Cable connected** for **Adapter 1** on each VM as shown in Figure *Disconecting cable to NAT interface*.

At this point, it would be a good idea to create snapshots of the VMs in this initial working state to have something to fall back on in case of mistakes at a later date. This is shown in Figure *Creating Snapshots in Virtualbox* and the steps to perform are described in How to use snapshots in VirtualBox and the Virtualbox document, Chapter 1. First steps.



```
changed: [purple.[redacted]]

RUNNING HANDLER [base : restart postgresql] *************************************************************************
Saturday 12 August 2017  19:08:11 -0700 (0:00:01.394)       0:41:03.883 *******
changed: [purple.[redacted]]

RUNNING HANDLER [base : restart nginx] ******************************************************************************
Saturday 12 August 2017  19:08:15 -0700 (0:00:04.731)       0:41:08.615 *******
changed: [purple.[redacted]]

PLAY RECAP **********************************************************************************************************
purple.[redacted]              : ok=414  changed=129  unreachable=0    failed=0
yellow.[redacted]              : ok=414  changed=131  unreachable=0    failed=0

Saturday 12 August 2017  19:08:17 -0700 (0:00:01.456)       0:41:10.071 *******
===============================================================================
nginx : Generate Strong Diffie-Hellman Group ------------------------- 129.59s
base : Make defined bats tests present -------------------------------- 38.79s
base : Make defined bats tests present -------------------------------- 38.23s
trident-configure : Ensure system configuration is present (tcli) ------ 23.85s
trident-configure : Ensure system configuration is present (tcli) ------ 23.84s
nginx : Run certbot standalone in Docker container -------------------- 21.68s
nginx : Run certbot standalone in Docker container -------------------- 21.50s
trident-core : Make core Trident packages present --------------------- 21.16s
trident-core : Make core Trident packages present --------------------- 21.08s
trident-configure : Ensure settings are applied to trust groups -------- 20.83s
trident-configure : Ensure settings are applied to trust groups -------- 20.75s
trident-core : Ensure core Trident packages are installed ------------- 19.86s
trident-core : Ensure core Trident packages are installed ------------- 19.36s
base : Ensure dims (system-level) subdirectories exist ---------------- 18.19s
base : Ensure dims (system-level) subdirectories exist ---------------- 18.05s
trident-configure : Add all members to all mailing lists -------------- 16.67s
nginx : Generate Strong Diffie-Hellman Group -------------------------- 16.48s
docker : Make Docker machine present ---------------------------------- 11.71s
trident-configure : Add all members to all mailing lists -------------- 11.15s
base : Make sure common (non-templated) BASH scripts are present ------- 10.95s
[dimsenv] ansible@dellr510:/opt/dims/git/private-[redacted](develop*) $
```

Fig. 13: Summary of full playbook run



☑ Enable Network Adapter

Attached to: NAT

Name:

Advanced

Adapter Type: Intel PRO/1000 MT Desktop (82540EM)

Promiscuous Mode: Deny

MAC Address: 08002701993D

☐ Cable Connected

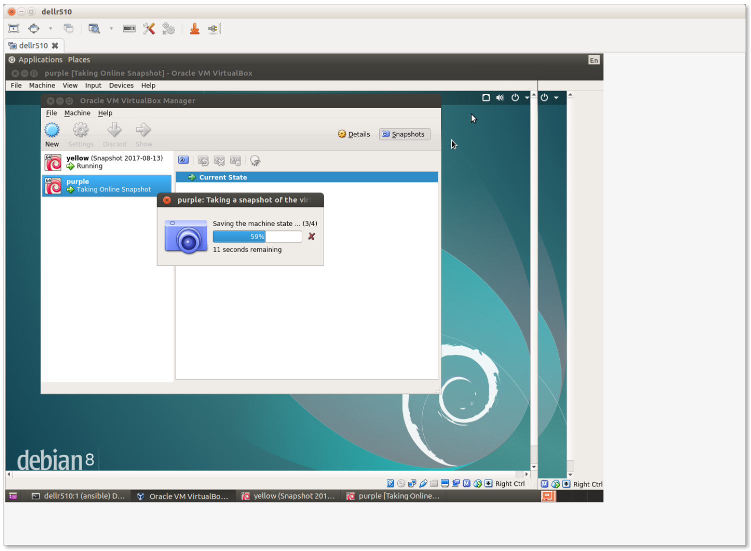
When checked, the virtual network cable is plugged in.

Controller: SATA
SATA Port 0: purple.vdi (Normal, 200.00 GB)

Host Driver: PulseAudio

```
480 blocks
 Failed to
42510/24883
11101/18326
 3465/11812
32/97536 fi
 no limit)_
```

Fig. 14: Disconecting cable to NAT interface



yellow (Snapshot 2017-08-13)
Running

purple
Taking Online Snapshot

Saving the machine state ... (3/4)

11 seconds remaining

Fig. 15: Creating Snapshots in Virtualbox

CHAPTER 5

Customizing a Private Deployment

The public Ansible playbooks in the ansible-dims-playbooks repository are designed to be public, which means they must (by definition) not contain real secrets. What is more, if someone wants to deploy their own instance of DIMS subsystems, they will need to maintain their own copies of inventory files, templates, and yes, secret files like Ansible vault, certificates, private keys, etc. These files obviously can't be committed to the public repository master or develop branches.

To facilitate keeping everything above (and more files, like backups of databases) completely separate, the ansible-dims-playbooks roles allow a second parallel repository that shares some of same subdirectories is used. The common directories are files/, roles/, and vars/. By convention, the directory is named private- followed by an identifier of your choice (e.g., private-devtest could be your development test deployment). This location is pointed to by the environment variable DIMS_PRIVATE and the Ansible variable dims_private which is set in the inventory, playbooks, or command line.

Note: Some wrapper scripts will automatically set dims_private from the environment variable DIMS_PRIVATE. There is a helper function in dims_functions.sh called get_private that returns the directory path based on the DIMS_PRIVATE environment variable or falling back to the ansible-dims-playbooks directory for a pure local development environment.

To facilitate creating the private customization directory repository, the cookiecutter program can be used.

5.1 Cookiecutter

Cookiecutter is a command-line utility used to template project structures. It uses Jinja2 to take generalized templates of file names and file contents and render them to create a new, unique directory tree. A popular Cookiecutter template, used by Cookiecutter in their documentation, is a Python package project template.

Cookiecutter can be used to template more than Python packages and can do so for projects using languages other than Python.

Cookiecutter documentation and examples:

- Latest Cookiecutter Docs
- Python Package Project Template Example
- Cookiecutter Tutorial

Cookiecutter is being integrated into the DIMS project as a Continuous Integration Utility. It's command line interface, `cookiecutter`, is installed along with other tools used in the DIMS project in the `dimsenv` Python virtual environment.

```
$ which cookiecutter
/home/dittrich/dims/envs/dimsenv/bin/cookiecutter
```

The source files used by `cookiecutter` can be found in `ansible-dims-playbooks/files/cookiecutter`.

The directory `ansible-dims-playbooks/files/cookiecutter/dims-new-repo/` provides a template for a new Git source code repository that contains a Sphinx documentation directory suitable for publication on ReadTheDocs.

Note: This template is usable for a source code repository with documentation, but can also be used for a documentation-only repository. If no Sphinx documentation is necessary, simply delete the `docs/` directory prior to making the initial commit to Git. Documenting how to use the repository is recommended.

5.1.1 Top Level Files and Directories

The `cookiecutter` template directory used for creating DIMS project Git repositories contains the following files and directories:

```
../cookiecutter/
+-- dims-new-repo
+-- dims-new-repo.yml
+-- dims-private
+-- README.txt

1 directory, 4 files
```

- The directory `dims-new-repo` is the templated Cookiecutter.
- The directory `dims-private` adds additional files by overlaying them into the appropriate places created by the main `dims-new-repo` templated Cookiecutter. It marks the repo as being non-public with warnings in documentation and a file named `DO_NOT_PUBLISH_THIS_REPO` in the top level directory to remind against publishing. It also includes hooks to ensure proper modes on SSH private key files.
- The file `dims-new-repo.yml` is a template for variables that can be used to over-ride the defaults contained in the Cookiecutter directory.
- The file `README.txt` is an example of how to use this Cookiecutter.

Files at this top level are not propagated to the output by `cookiecutter`, only the contents of the slug directory tree rooted at `{{cookiecutter.project_slug}}` will be included.

5.1.2 The `dims-new-repo` Cookiecutter

Going one level deeper into the Cookiecutter template directory `dims-new-repo`, you will find the following files and directories:

```
$ tree -a cookiecutter/dims-new-repo/
cookiecutter/dims-new-repo/
+── cookiecutter.json
+── {{cookiecutter.project_slug}}
│   +── .bumpversion.cfg
│   +── docs
│   │   +── build
│   │   │   +── .gitignore
│   │   +── Makefile
│   │   +── source
│   │       +── conf.py
│   │       +── index.rst
│   │       +── introduction.rst
│   │       +── license.rst
│   │       +── license.txt
│   │       +── static
│   │       │   +── .gitignore
│   │       +── templates
│   │       │   +── .gitignore
│   │       +── d2-logo.ico
│   │       +── d2-logo.png
│   +── README.rst
│   +── VERSION
+── hooks
    +── post_gen_project.sh

7 directories, 16 files
```

- The directory `{{cookiecutter.project_slug}}` is what is called the *slug* directory, a directory that will be processed as a template to produce a new directory with specific content based on variable expansion. It contains all the other files, pre-configured for use by programs like Sphinx, `bumpversion`, and Git.

Note: Note the name of this directory includes paired curly braces (`{{` and `}}`) that tell Jinja to substitute the value of a variable into the template. In the Ansible world, some people call these "mustaches" (tilt you head and squint a little and you'll get it.)

The thing inside the mustaches in this directory name is a Jinja dictionary variable reference, with `cookiecutter` being the top level dictionary name and `project_slug` being the key to an entry in the dictionary. You will see this variable name below in the `cookiecutter.json` default file and `dims-new-repo.yml` configuration file.

The curly brace characters (`{}`) are also Unix shell metacharacters used for advanced filename globbing, so you may need to escape them using `''` or `\` on a shell command line "remove the magic." For example, if you `cd` into the `dims-new-repo` directory, type `ls {` and then press `TAB` for file name completion, you will see the following:

```
$ ls \{\{cookiecutter.project_slug\}\}/
```

- The file `cookiecutter.json` is the set of defaults in JSON format. Templated files in the *slug* directory will be substituted from these variables. If desired, `cookiecutter` will use these to produce prompts that you can fill in with specifics at run time.

- The directory `hooks` is holds scripts that are used for pre- and post-processing of the template output directory. (You may not need to pay any attention to this directory.)

Project Slug Directory

Path: `$GIT/ansible-dims-playbooks/files/cookiecutter/dims-new-repo/{{ cookiecutter.project_slug }}`

Every Cookiecutter includes a directory with a name in the format of `{{cookiecutter.project_slug}}`. This is how the `cookiecutter` program knows where to start templating. Everything outside of this directory is ignored in the creation of the new project. The directory hierarchy of this directory will be used to make the directory hierarchy of the new project. The user can populate the `{{cookiecutter.project_slug}}` directory with any subdirectory structure and any files they will need to instantiate templated versions of their project. Any files in this directory can similarly use variables of the same format as the slug directory. These variables must be defined by either defaults or a configuration file or an undefined variable error will occur.

Look back at the example `cookiecutter.json` file. For that Cookiecutter, a new repo with the project name `DIMS Test Repo` would be found in a directory called `dims-test-repo` (this is the `{{cookiecutter.project_name}}` to `{{cookiecutter.project_slug}}` conversion).

Look back at the `tree -a` output. For that cookiecutter, a new directory would have a `docs/` subdirectory, with its own subdirectories and files, a `.bumpversion.cfg` file, and a `VERSION` file. Any time this cookiecutter is used, this is the hierarchy and files the new repo directory will have.

```
{{cookiecutter.project_slug}}/
+-- .bumpversion.cfg
+-- docs
|   +-- Makefile
|   +-- source
|       +-- conf.py
|       +-- index.rst
|       +-- license.rst
|       +-- license.txt
|       +-- d2-logo.ico
|       +-- d2-logo.png
+-- VERSION

4 directories, 8 files
```

- `.bumpversion.cfg`: used to keep track of the version in various locations in the repo.
- `VERSION`: file containing current version number
- `docs/`:
    - `Makefile`: used to build HTML and LaTeX documents
    - `source/`:
        - minimal doc set (`index.rst` and `license.rst`)
        - `.ico` and `.png` files for branding the documents
        - `conf.py` which configures the document theme, section authors, project information, etc. Lots of variables used in this file, set from `cookiecutter.json` values.

Template Defaults

Path: `$GIT/ansible-dims-playbooks/files/cookiecutter/dims-new-repo/cookiecutter.json`

Every cookiecutter has a cookiecutter.json file. This file contains the default variable definitions for a template. When the user runs the cookiecutter command, they can be prompted for this information. If the user provides no information, the defaults already contained in the .json file will be used to create the project.

The cookiecutter.json file in the dims-new-repo Cookiecutter slug directory contains the following:

```
{
  "full_name": "_NAME_",
  "email": "_EMAIL_",
  "project_name": "_PROJECT_NAME_",
  "project_slug": "_PROJECT_SLUG_",
  "project_short_description": "_PROJECT_DESCRIPTION_",
  "release_date": "{% now 'utc', '%Y-%m-%d' %}",
  "project_version": "0.1.0",
  "project_copyright_name": "_COPYRIGHT_NAME_",
  "project_copyright_date": "{% now 'utc', '%Y' %}",
   "_extensions": ["jinja2_time.TimeExtension"]
}
```

Python commands can be used to manipulate the values of one field to create the value of another field.

For example, you can generate the project slug from the repository name using the following:

```
{

"project_name": "DIMS New Repo Boilerplate",
"project_slug": "{{ cookiecutter.project_name.lower().replace(' ', '-') }}",

}
```

The resulting slug would look like dims-new-repo-boilerplate.

You can also load Jinja extensions by including an array named _extensions (shown array at the bottom of the JSON defaults file.) The variables release_date and project_copyright_date are produced programmatically using the Jinja2_time.TimeExtension extension. These are filled with the current date/time as defined. You can over-ride them using the dims-new-repo.yml YAML file adding the variables by name.

Custom Configuration File

Path: $GIT/ansible-dims-playbooks/files/cookiecutter/dims-new-repo/dims-new-repo.yml

The file dims-new-repo.yml is a configuration file that can be passed to cookiecutter using the --config-file command line option. It sets the dictionary default_context for cookiecutter at runtime, over-riding the defaults from the cookiecutter.json file.

```
---

default_context:
  full_name: "_NAME_"
  email: "_EMAIL_"
  project_name: "_PROJECT_NAME_"
  project_slug: "_PROJECT_SLUG_"
  project_short_description: "_PROJECT_DESCRIPTION_"
  project_copyright_name: "_COPYRIGHT_NAME_"
```

To use this file, copy the file dims-new-repo.yml and give it a unique name to differentiate it from other configuration files. This allows you to easily create more than one repository directory at a time, as well as save the settings to easily repeat the process for development and testing of the slug directory when you need to update it. For this example, we will use `testrepo.yml` for the configuration file.

```
$ cp dims-new-repo.yml testrepo.yml
$ vi testrepo.yml
```

Edit the template to customize is at necessary. It should end up looking something like this:

```
$ cat testrepo.yml
---

default_context:
  full_name: "Dave Dittrich"
  email: "dave.dittrich@gmail.com"
  project_name: "D2 Ansible Playbooks"
  project_slug: "ansible-dims-playbooks"
  project_short_description: "Ansible Playbooks for D2 System Configuration"
  project_copyright_name: "David Dittrich"
```

Usage

By default, `cookiecutter` will generate the new directory with the name specified by the `cookiecutter.project_slug` variable in the current working directory. Provide a relative or absolute path to another directory (e.g., `$GIT`, so place the new directory in the standard DIMS repo directory) using the `-o` command line option. In this example, we will let `cookiecutter` prompt for alternatives to the defaults from the `cookiecutter.json` file:

```
$ cd $GIT/dims-ci-utils/cookiecutter
$ cookiecutter -o ~/ dims-new-repo/
full_name [DIMS User]: Dave Dittrich
email []: dave.dittrich@gmail.com
project_name [DIMS New Repo Boilerplate]: Test Repo
project_short_description [DIMS New Repo Boilerplate contains docs/ setup, conf.py␣
↪template, .bumpversion.cfg, LICENSE file, and other resources needed for␣
↪instantiating a new repo.]: This is just a test
release_date [20YY-MM-DD]: 2018-01-01
project_version [1.0.0]:
project_slug [test-repo]:
project_copyright_name [David Dittrich]:
project_copyright_date [2018]:
$ cd ~/test-repo
$ ls
docs  VERSION
$ tree -a
.
+-- .bumpversion.cfg
+-- docs
│   +-- build
│   │   +-- .gitignore
│   +-- Makefile
│   +-- source
│       +-- conf.py
│       +-- images
│       +-- index.rst
│       +-- license.rst
│       +-- license.txt
```

(continues on next page)

(continued from previous page)

```
│       +── static
│       │   +── .gitignore
│       +── templates
│       │   +── .gitignore
│       +── d2-logo.ico
│       +── d2-logo.png
+── VERSION

3 directories, 12 files
```

The highlighted section in the above code block is the prompts for cookiecutter.json configuring. As you can see, I answer the first five prompts, the ones which require user input, and leave the rest blank because they don't require user input.

Following that, you can see the `tree` structure of the newly created repo called "test-repo". Once this is done, you can finish following repo setup instructions found in dimsdevguide:sourcemanagement.

Alternatively, you can change your current working directory to be the location where you want the templated directory to be created and specify the template source using an absolute path. In this example, we also use a configuration file, also specified with an absolute path:

```
$ mkdir -p /tmp/new/repo/location
$ cd /tmp/new/repo/location
$ cookiecutter --no-input \
> --config-file /home/dittrich/dims/git/dims-ci-utils/cookiecutter/testrepo.yml \
> /home/dittrich/dims/git/dims-ci-utils/cookiecutter/dims-new-repo
[+] Fix underlining in these files:
/tmp/new/repo/location/ansible-dims-playbooks/docs/source/index.rst
/tmp/new/repo/location/ansible-dims-playbooks/README.rst
$ tree
.
+── ansible-dims-playbooks
    +── docs
    │   +── build
    │   +── Makefile
    │   +── source
    │       +── conf.py
    │       +── index.rst
    │       +── introduction.rst
    │       +── license.rst
    │       +── license.txt
    │       +── _static
    │       +── _templates
    │       +── d2-logo.ico
    │       +── d2-logo.png
    +── README.rst
    +── VERSION

6 directories, 10 files
```

Note the lines that show up right after the command line (highlighted here):

```
$ cookiecutter --no-input -f -o /tmp --config-file testrepo.yml dims-new-repo
[+] Fix underlining in these files:
/tmp/ansible-dims-playbooks/docs/source/index.rst
/tmp/ansible-dims-playbooks/README.rst
```

ReStructureText (RST) files *must* have section underlines that are exactly the same length as the text for the section.

Since the templated output length is not known when the template is written, it is impossible to correctly guess 100% of the time how many underline characters are needed. This could be handled with post-processing using `awk`, `perl`, etc., or it can just be called out by identifying a fixed string. The latter is what this Cookiecutter uses.

To produce one of these warning messages, simply place a line containing the string `FIX_UNDERLINE` in the template file, as shown here:

```
.. {{ cookiecutter.project_slug }} documentation master file, created by
   cookiecutter on {{ cookiecutter.release_date }}.
   You can adapt this file completely to your liking, but it should at least
   contain the root `toctree` directive.

{{ cookiecutter.project_name }} v |release|
.. FIX_UNDERLINE

This document (version |release|) describes the
{{ cookiecutter.project_name }} (``{{ cookiecutter.project_slug }}``
for short) repository contents.

.. toctree::
   :maxdepth: 3
   :numbered:
   :caption: Contents:

   introduction
   license

.. sectionauthor:: {{ cookiecutter.full_name }} {{ cookiecutter.email }}

.. include:: <isonum.txt>

Copyright |copy| {{ cookiecutter.project_copyright_date }} {{ cookiecutter.project_
↪copyright_name }}. All rights reserved.
```

Edit these files to fix the underline before committing them to Git, as shown here:

```
DIMS Ansible Playbooks v |release|
==================================
```

5.1.3 The `dims-private` Cookiecutter

If the repo is supposed to be non-public, use the same configuration file to overlay files from the `dims-private` Cookiecutter onto the same output directory as the main repo directory. It uses a symbolic link for the `cookiecutter.json` file to have exactly the same defaults and using the same configuration file ensures the same output directory and templated values are output as appropriate.

```
$ cookiecutter --no-input -f -o /tmp --config-file testrepo.yml dims-private
[+] Fix underlining in these files:
/tmp/ansible-dims-playbooks/docs/source/index.rst
/tmp/ansible-dims-playbooks/README.rst
```

The `dims-private` Cookiecutter also adds a directory `hooks/` and a `Makefile` that installs `post-checkout` and `post-merge` hooks that Git will run after checking out and merging branches to fix file permissions on SSH private keys. Git has a limitation in its ability to track all Unix mode bits. It only tracks whether the execute bit is set or not. This causes the wrong mode bits for SSH keys that will prevent them from being used. These hooks fix this in a very simplistic way (though it does work.)

The very first time after the repository is cloned, the hooks will not be installed as they reside in the `.git` directory. Install them by typing `make` at the top level of the repository:

```
$ make
[+] Installing .git/hooks/post-checkout
[+] Installing .git/hooks/post-merge
```

The hooks will be triggered when needed and you will see an added line in the Git output:

```
$ git checkout master
Switched to branch 'master'
[+] Verifying private key permissions and correcting if necessary
```

5.2 Populating the Private Configuration Repository

Start creating your local customization repository using the `cookiecutter` template discussed in the previous section. We will call this private deployment `devtest`, thus creating a repository in a the directory named `$GIT/private-devtest`. Here is the configuration file we will use:

```
$ cd $GIT
$ cat private-devtest.yml
---

default_context:
  full_name: "Dave Dittrich"
  email: "dave.dittrich@gmail.com"
  project_name: "Deployment \"devtest\" private configuration"
  project_slug: "private-devtest"
  project_short_description: "Ansible playbooks private content for \"devtest\"␣
↪deployment"
  project_copyright_name: "David Dittrich"
```

First, generate the new repository from the `dims-new-repo` template, followed by adding in the files from the `dims-private` template.

```
$ cookiecutter --no-input -f -o . --config-file private-devtest.yml $GIT/dims-ci-
↪utils/cookiecutter/dims-new-repo
[+] Fix underlining in these files:
./private-devtest/docs/source/index.rst
./private-devtest/README.rst
$ cookiecutter --no-input -f -o . --config-file private-devtest.yml $GIT/dims-ci-
↪utils/cookiecutter/dims-private
```

Note: Be sure to edit the two documents that are mentioned above right now to fix the headings, and possibly to change the documentation in the `README.rst` file to reference the actual location of the private GIT repository.

You now have a directory ready to be turned into a Git repository with all of the requisite files for `bumpversion` version number tracking, Sphinx documentation, and hooks for ensuring proper permissions on SSH private key files.

```
    $ tree -a private-devtest
    private-devtest
    +── .bumpversion.cfg
    +── docs
```

(continues on next page)

(continued from previous page)

```
    │   +── .gitignore
    │   +── Makefile
    │   +── source
    │       +── conf.py
    │       +── index.rst
    │       +── introduction.rst
    │       +── license.rst
    │       +── license.txt
    │       +── _static
    │       │   +── .gitignore
    │       +── _templates
    │       │   +── .gitignore
    │       +── d2-logo.ico
    │       +── d2-logo.png
    +── DO_NOT_PUBLISH_THIS_REPO
    +── hooks
    │   +── post-checkout
    │   +── post-merge
    +── Makefile
    +── README.rst
    +── VERSION

    5 directories, 18 files

..
```

Next, begin by creating the Ansible `inventory/` directory that will describe your deployment. Copy the `group_vars`, `host_vars`, and `inventory` directory trees to the new custom directory.

```
$ cp -vrp $PBR/{group_vars,host_vars,inventory} -t private-devtest
'/home/dittrich/dims/git/ansible-dims-playbooks/group_vars' -> 'private-devtest/group_
↪vars'
'/home/dittrich/dims/git/ansible-dims-playbooks/group_vars/all' -> 'private-devtest/
↪group_vars/all'
'/home/dittrich/dims/git/ansible-dims-playbooks/group_vars/all/consul.yml' ->␣
↪'private-devtest/group_vars/all/consul.yml'
'/home/dittrich/dims/git/ansible-dims-playbooks/group_vars/all/rsyslog.yml' ->␣
↪'private-devtest/group_vars/all/rsyslog.yml'
'/home/dittrich/dims/git/ansible-dims-playbooks/group_vars/all/prisem_rpc.yml' ->␣
↪'private-devtest/group_vars/all/prisem_rpc.yml'
'/home/dittrich/dims/git/ansible-dims-playbooks/group_vars/all/trident.yml' ->␣
↪'private-devtest/group_vars/all/trident.yml'
'/home/dittrich/dims/git/ansible-dims-playbooks/group_vars/all/postgresql.yml' ->␣
↪'private-devtest/group_vars/all/postgresql.yml'
'/home/dittrich/dims/git/ansible-dims-playbooks/group_vars/all/nginx.yml' -> 'private-
↪devtest/group_vars/all/nginx.yml'
'/home/dittrich/dims/git/ansible-dims-playbooks/group_vars/all/.dims.yml.swp' ->␣
↪'private-devtest/group_vars/all/.dims.yml.swp'
'/home/dittrich/dims/git/ansible-dims-playbooks/group_vars/all/networks.yml' ->␣
↪'private-devtest/group_vars/all/networks.yml'
'/home/dittrich/dims/git/ansible-dims-playbooks/group_vars/all/docker.yml' ->␣
↪'private-devtest/group_vars/all/docker.yml'
'/home/dittrich/dims/git/ansible-dims-playbooks/group_vars/all/dnsmasq.yml' ->␣
↪'private-devtest/group_vars/all/dnsmasq.yml'
'/home/dittrich/dims/git/ansible-dims-playbooks/group_vars/all/dims.yml' -> 'private-
↪devtest/group_vars/all/dims.yml'
```

(continues on next page)

(continued from previous page)

```
‘/home/dittrich/dims/git/ansible-dims-playbooks/group_vars/all/swarm.yml’ -> ‘private-
↪devtest/group_vars/all/swarm.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/group_vars/all/squid-deb-proxy.yml’ ->
↪ ‘private-devtest/group_vars/all/squid-deb-proxy.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/group_vars/all/vagrant.yml’ ->␣
↪‘private-devtest/group_vars/all/vagrant.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/group_vars/all/go.yml’ -> ‘private-
↪devtest/group_vars/all/go.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/group_vars/vault.yml’ -> ‘private-
↪devtest/group_vars/vault.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/group_vars/README.txt’ -> ‘private-
↪devtest/group_vars/README.txt’
‘/home/dittrich/dims/git/ansible-dims-playbooks/host_vars’ -> ‘private-devtest/host_
↪vars’
‘/home/dittrich/dims/git/ansible-dims-playbooks/host_vars/purple.devops.local.yml’ ->␣
↪‘private-devtest/host_vars/purple.devops.local.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/host_vars/.gitignore’ -> ‘private-
↪devtest/host_vars/.gitignore’
‘/home/dittrich/dims/git/ansible-dims-playbooks/host_vars/node02.devops.local.yml’ ->␣
↪‘private-devtest/host_vars/node02.devops.local.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/host_vars/yellow.devops.local.yml’ ->␣
↪‘private-devtest/host_vars/yellow.devops.local.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/host_vars/green.devops.local.yml’ ->␣
↪‘private-devtest/host_vars/green.devops.local.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/host_vars/red.devops.local.yml’ ->␣
↪‘private-devtest/host_vars/red.devops.local.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/host_vars/orange.devops.local.yml’ ->␣
↪‘private-devtest/host_vars/orange.devops.local.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/host_vars/vmhost.devops.local.yml’ ->␣
↪‘private-devtest/host_vars/vmhost.devops.local.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/host_vars/node03.devops.local.yml’ ->␣
↪‘private-devtest/host_vars/node03.devops.local.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/host_vars/node01.devops.local.yml’ ->␣
↪‘private-devtest/host_vars/node01.devops.local.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/host_vars/blue14.devops.local.yml’ ->␣
↪‘private-devtest/host_vars/blue14.devops.local.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/host_vars/blue16.devops.local.yml’ ->␣
↪‘private-devtest/host_vars/blue16.devops.local.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/host_vars/hub.devops.local.yml’ ->␣
↪‘private-devtest/host_vars/hub.devops.local.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory’ -> ‘private-devtest/
↪inventory’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/dns_zones’ -> ‘private-
↪devtest/inventory/dns_zones’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/dns_zones/nodes.yml’ ->␣
↪‘private-devtest/inventory/dns_zones/nodes.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/trident’ -> ‘private-
↪devtest/inventory/trident’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/trident/nodes.yml’ ->␣
↪‘private-devtest/inventory/trident/nodes.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/vagrants’ -> ‘private-
↪devtest/inventory/vagrants’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/vagrants/nodes.yml’ ->␣
↪‘private-devtest/inventory/vagrants/nodes.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/ci-server’ -> ‘private-
↪devtest/inventory/ci-server’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/ci-server/nodes.yml’ ->␣
↪‘private-devtest/inventory/ci-server/nodes.yml’
```

(continues on next page)

(continued from previous page)

```
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/swarm’ -> ‘private-devtest/
↪inventory/swarm’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/swarm/nodes.yml’ ->␣
↪‘private-devtest/inventory/swarm/nodes.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/private’ -> ‘private-
↪devtest/inventory/private’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/private/nodes.yml’ ->␣
↪‘private-devtest/inventory/private/nodes.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/host_vars’ -> ‘private-
↪devtest/inventory/host_vars’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/group_vars’ -> ‘private-
↪devtest/inventory/group_vars’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/all.yml’ -> ‘private-
↪devtest/inventory/all.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/nameserver’ -> ‘private-
↪devtest/inventory/nameserver’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/nameserver/nodes.yml’ ->␣
↪‘private-devtest/inventory/nameserver/nodes.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/ansible-server’ -> ‘private-
↪devtest/inventory/ansible-server’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/ansible-server/nodes.yml’ ->
↪ ‘private-devtest/inventory/ansible-server/nodes.yml’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/coreos’ -> ‘private-devtest/
↪inventory/coreos’
‘/home/dittrich/dims/git/ansible-dims-playbooks/inventory/coreos/nodes.yml’ ->␣
↪‘private-devtest/inventory/coreos/nodes.yml’
```

The names of hosts in the inventory need to be changed to match the new deployment name. This is necessary for mapping inventory host names to `host_vars` files, as well as to generate the proper split-DNS name to IP address mappings (among other things). In the inventory, this process can be automated a little bit.

Start by verifying that the word `local` only occurs in the inventory in places where it can be cleanly edited using a simple inline string editing (`sed` style) regular expression.

```
$ grep -r local inventory
inventory/dns_zones/nodes.yml:      'local':
inventory/dns_zones/nodes.yml:          - 'red.devops.local'
inventory/dns_zones/nodes.yml:          - 'vmhost.devops.local'
inventory/dns_zones/nodes.yml:        mxserver: 'vmhost.devops.local'
inventory/dns_zones/nodes.yml:    local:
inventory/dns_zones/nodes.yml:        'vmhost.devops.local':
inventory/dns_zones/nodes.yml:        'red.devops.local':
inventory/dns_zones/nodes.yml:        'orange.devops.local':
inventory/dns_zones/nodes.yml:        'blue14.devops.local':
inventory/dns_zones/nodes.yml:        'blue16.devops.local':
inventory/dns_zones/nodes.yml:        'yellow.devops.local':
inventory/dns_zones/nodes.yml:        'purple.devops.local':
inventory/dns_zones/nodes.yml:        'hub.devops.local':
inventory/dns_zones/nodes.yml:        'node01.devops.local':
inventory/dns_zones/nodes.yml:        'node02.devops.local':
inventory/dns_zones/nodes.yml:        'node03.devops.local':
inventory/dns_zones/nodes.yml:        'node01.devops.local':
inventory/dns_zones/nodes.yml:        'node02.devops.local':
inventory/dns_zones/nodes.yml:        'node03.devops.local':
inventory/trident/nodes.yml:    'yellow.devops.local':
inventory/trident/nodes.yml:    'purple.devops.local':
inventory/vagrants/nodes.yml:      'local': 'eth1'
```

(continues on next page)

(continued from previous page)

```
inventory/vagrants/nodes.yml:    'red.devops.local':
inventory/vagrants/nodes.yml:    'node01.devops.local':
inventory/vagrants/nodes.yml:    'node02.devops.local':
inventory/vagrants/nodes.yml:    'node03.devops.local':
inventory/vagrants/nodes.yml:    'yellow.devops.local':
inventory/vagrants/nodes.yml:    'purple.devops.local':
inventory/vagrants/nodes.yml:    'blue14.devops.local':
inventory/vagrants/nodes.yml:    'orange.devops.local':
inventory/vagrants/nodes.yml:        'red.devops.local':
inventory/vagrants/nodes.yml:        'node01.devops.local':
inventory/vagrants/nodes.yml:        'node02.devops.local':
inventory/vagrants/nodes.yml:        'node03.devops.local':
inventory/vagrants/nodes.yml:        'yellow.devops.local':
inventory/vagrants/nodes.yml:        'orange.devops.local':
inventory/vagrants/nodes.yml:        'purple.devops.local':
inventory/vagrants/nodes.yml:        'blue14.devops.local':
inventory/vagrants/nodes.yml:        'red.devops.local':
inventory/vagrants/nodes.yml:        'yellow.devops.local':
inventory/vagrants/nodes.yml:        'orange.devops.local':
inventory/ci-server/nodes.yml:#     jenkins_hostname: jenkins.devops.local
inventory/ci-server/nodes.yml:    jenkins_hostname: localhost
inventory/ci-server/nodes.yml:    'orange.devops.local':
inventory/swarm/nodes.yml:    'red.devops.local':
inventory/swarm/nodes.yml:    'yellow.devops.local':
inventory/swarm/nodes.yml:    'purple.devops.local':
inventory/swarm/nodes.yml:    'node01.devops.local':
inventory/swarm/nodes.yml:    'node02.devops.local':
inventory/swarm/nodes.yml:    'node03.devops.local':
inventory/swarm/nodes.yml:        'node01.devops.local':
inventory/swarm/nodes.yml:        'node02.devops.local':
inventory/swarm/nodes.yml:        'node03.devops.local':
inventory/swarm/nodes.yml:        'red.devops.local':
inventory/swarm/nodes.yml:        'yellow.devops.local':
inventory/swarm/nodes.yml:        'purple.devops.local':
inventory/private/nodes.yml:    'vmhost.devops.local':
inventory/private/nodes.yml:    'red.devops.local':
inventory/private/nodes.yml:    'orange.devops.local':
inventory/private/nodes.yml:    'blue14.devops.local':
inventory/private/nodes.yml:    'blue16.devops.local':
inventory/private/nodes.yml:    'yellow.devops.local':
inventory/private/nodes.yml:    'purple.devops.local':
inventory/private/nodes.yml:    'hub.devops.local':
inventory/private/nodes.yml:    'node01.devops.local':
inventory/private/nodes.yml:    'node02.devops.local':
inventory/private/nodes.yml:    'node03.devops.local':
inventory/all.yml:    deployment: 'local'
inventory/all.yml:    dims_domain: 'devops.local'
inventory/all.yml:    'vmhost.devops.local':
inventory/all.yml:    'orange.devops.local':
inventory/all.yml:    'red.devops.local':
inventory/all.yml:    'node01.devops.local':
inventory/all.yml:    'node02.devops.local':
inventory/all.yml:    'node03.devops.local':
inventory/all.yml:    'yellow.devops.local':
inventory/all.yml:    'purple.devops.local':
inventory/all.yml:    'blue14.devops.local':
inventory/nameserver/nodes.yml:    'red.devops.local':
```

(continues on next page)

(continued from previous page)

```
inventory/nameserver/nodes.yml:    'vmhost.devops.local':
inventory/ansible-server/nodes.yml:    'vmhost.devops.local':
inventory/ansible-server/nodes.yml:    'orange.devops.local':
inventory/coreos/nodes.yml:    iptables_rules: rules.v4.coreos-local.j2
inventory/coreos/nodes.yml:    dims_environment: environment.coreos-local.j2
inventory/coreos/nodes.yml:    # This is not specific to "local" deployment, but is␣
↪specific to coreos
inventory/coreos/nodes.yml:    'node01.devops.local':
inventory/coreos/nodes.yml:    'node02.devops.local':
inventory/coreos/nodes.yml:    'node03.devops.local':
```

Doing this on a BASH command line in Linux would highlight the word `local`, making it easier to see, but there is no need to do anything other than simply substitute `local` with `devtest`.

Caution: The kind of bulk editing that will be shown next is powerful, which means it is also risky. Accidental damage from typos on the command line can be very difficult to recover from. For example, if you were to blindly change the word `local` to `devtest` in the following files, you would break many things:

```
. . .
../roles/postgresql/templates/postgresql/postgresql.conf.j2:listen_addresses =
↪'localhost'              # what IP address(es) to listen on;
../roles/postgresql/templates/postgresql/postgresql.conf.j2:log_timezone =
↪'localtime'
../roles/postgresql/templates/postgresql/postgresql.conf.j2:timezone = 'localtime'
. . .
../roles/postgresql/templates/postgresql/pg_hba.conf.j2:local   all             all␣
↪                                     trust
../roles/postgresql/templates/postgresql/pg_hba.conf.j2:local   replication    ␣
↪postgres                                 trust
  . . .
../roles/nginx/templates/nginx/nginx.conf.j2:   access_log syslog:server=localhost,
↪facility={{ syslog_facility }},tag=nginx,severity={{ syslog_severity }};
../roles/nginx/templates/nginx/nginx.conf.j2:   error_log syslog:server=localhost,
↪facility={{ syslog_facility }},tag=nginx,severity={{ syslog_severity }};
. . .
../roles/base/files/hub.bash_completion.sh:    local line h k v host=${1:-github.
↪com} config=${HUB_CONFIG:-~/.config/hub}
../roles/base/files/hub.bash_completion.sh:    local f format=$1
../roles/base/files/hub.bash_completion.sh:    local i remote repo branch dir=$(__
↪gitdir)
../roles/base/files/hub.bash_completion.sh:    local i remote=${1:-origin} dir=$(__
↪gitdir)
. . .
../roles/base/files/git-prompt.sh:      local upstream=git legacy="" verbose=""␣
↪name=""
../roles/base/files/git-prompt.sh:      local output="$(git config -z --get-regexp
↪'^(svn-remote\..*\.url|bash\.showupstream)$' 2>/dev/null | tr '\0\n' '\n ')"
../roles/base/files/git-prompt.sh:              local -a svn_upstream
../roles/base/files/git-prompt.sh:                      local n_stop="${#svn_
↪remote[@]}"
../roles/base/files/git-prompt.sh:              local commits
. . .
../roles/base/files/dims_functions.sh:    local retval=$1 && shift
../roles/base/files/dims_functions.sh:    local script=$1
../roles/base/files/dims_functions.sh:    local n=${#on_exit_items[*]}
../roles/base/files/dims_functions.sh:    local _deployment=${1:-${DEPLOYMENT}}
. . .
```

If you are not comfortable and confident that you know what you are doing, practice first by making a copy of the directory tree to the `/tmp` directory and trying the edits there. Using `diff -r` against both the original directory and the temporary directory will show you the effects and allow you to validate they reflect what you desire before applying to the actual files.

Use the `-l` option of `grep` to get just the file names, save them to a file, and use that file in an inline command substitution in a `for` loop to edit the files inline using `perl`.

```
$ grep -lr local inventory > /tmp/files
$ for F in $(cat /tmp/files); do perl -pi -e 's/local/devtest/' $F; done
```

Next, rename all of the `host_vars` files to have names that match the deployment name `devtest` and the changes made to the inventory files, and carefully change the internal contents like the last step so they match as well.

```
$ cd private-devtest/host_vars/
$ ls
blue14.devops.local.yml  green.devops.local.yml  node01.devops.local.yml
node03.devops.local.yml  purple.devops.local.yml vmhost.devops.local.yml
blue16.devops.local.yml  hub.devops.local.yml    node02.devops.local.yml
orange.devops.local.yml  red.devops.local.yml    yellow.devops.local.yml
$ for F in *.yml; do mv $F $(echo $F | sed 's/local/devtest/'); done
$ ls
blue14.devops.devtest.yml  green.devops.devtest.yml  node01.devops.devtest.yml
node03.devops.devtest.yml  purple.devops.devtest.yml vmhost.devops.devtest.yml
blue16.devops.devtest.yml  hub.devops.devtest.yml    node02.devops.devtest.yml
orange.devops.devtest.yml  red.devops.devtest.yml    yellow.devops.devtest.yml
$ grep local *
blue14.devops.devtest.yml:# File: host_vars/blue14.devops.local.yml
blue16.devops.devtest.yml:# File: host_vars/blue16.devops.local.yml
green.devops.devtest.yml:# File: host_vars/green.devops.local.yml
hub.devops.devtest.yml:# File: host_vars/hub.devops.local.yml
node01.devops.devtest.yml:# File: host_vars/node01.devops.local.yml
node02.devops.devtest.yml:# File: host_vars/node02.devops.local.yml
node03.devops.devtest.yml:# File: host_vars/node03.devops.local.yml
orange.devops.devtest.yml:# file: host_vars/orange.devops.local
orange.devops.devtest.yml:jenkins_url_external: 'http://orange.devops.local:8080'
purple.devops.devtest.yml:# File: host_vars/purple.devops.local.yml
red.devops.devtest.yml:# File: host_vars/red.devops.local.yml
vmhost.devops.devtest.yml:  'local': 'vboxnet3'
yellow.devops.devtest.yml:# File: host_vars/yellow.devops.local.yml
$ grep -l local *.yml > /tmp/files
$ for F in $(cat /tmp/files); do perl -pi -e 's/local/devtest/' $F; done
$ cd -
/home/dittrich/dims/git
```

CHAPTER 6

Testing System Components

The DIMS project has adopted use of the Bats: Bash Automated Testing System (known as bats) to perform simple tests in a manner that produces parsable output following the Test Anything Protocol (TAP).

Bats is a **TAP Producer**, whose output can be processed by one of many TAP Consumers, including the Python program tap.py.

6.1 Organizing Bats Tests

This section covers the basic functionality of bats and how it can be used to produce test results.

We should start by looking at the --help output for bats to understand how it works in general.

```
$ bats -h
Bats 0.4.0
Usage: bats [-c] [-p | -t] <test> [<test> ...]

  <test> is the path to a Bats test file, or the path to a directory
  containing Bats test files.

  -c, --count    Count the number of test cases without running any tests
  -h, --help     Display this help message
  -p, --pretty   Show results in pretty format (default for terminals)
  -t, --tap      Show results in TAP format
  -v, --version  Display the version number

  For more information, see https://github.com/sstephenson/bats
```

As is seen, multiple tests – files that end in .bats – can be passed as a series of arguments on the command line. This can be either individual arguments, or a wildcard shell expression like *.bats.

If the argument evaluates to being a directory, bats will look through that directory and run all files in it that end in .bats.

Caution: As we will see, bats has some limitations that do not allow mixing file arguments and directory arguments. You can either give bats one or more files, or you can give it one or more directories, but you **cannot mix files and directories**.

To see how this works, let us start with a simple example that has tests that do nothing other than report success with their name. In this case, test a.bats looks like this:

```
#!/usr/bin/env bats

@test "a" {
    [[ true ]]
}
```

We produce three such tests, each in their own directory, following this organizational structure:

```
$ tree tests
tests
+—— a
│   +—— a.bats
+—— b
    +—— b.bats
    +—— c
        +—— c.bats

3 directories, 3 files
```

Since the hierarchy shown here does not contain tests itself, but rather holds directories that in turn hold tests, how does we run the tests?

Running bats with an argument that includes the highest level of the directory hierarchy does not work to run any of the tests in subordinate directories:

```
$ bats tests

0 tests, 0 failures
```

Running bats and passing a directory that contains files with names that end in .bats runs all of the tests in *that directory*.

```
$ bats tests/a
 ✓ a

1 test, 0 failures
```

If we specify the next directory tests/b, then bats will run the tests in that directory that end in .bats, but will not traverse down into the tests/b/c/ directory.

```
$ bats tests/b
 ✓ b

1 test, 0 failures
```

To run the tests in the lowest directory, that specific directory must be given on the command line:

```
$ bats tests/b/c
 ✓ c
```

(continues on next page)

(continued from previous page)

```

1 test, 0 failures
```

Attempting to pass all of the directories along as arguments does not work, as seen here:

```
$ bats tests/a /tests/b tests/b/c
bats: /tmp/b does not exist
/usr/local/Cellar/bats/0.4.0/libexec/bats-exec-suite: line 20: let: count+=: syntax␣
↪error: operand expected (error token is "+=")
```

This means that we *can* separate tests into subdirectories, to any depth or directory organizational structure, as needed, but tests must be run on a per-directory basis, or identified and run as a group of tests passed as file arguments using wildcards:

```
$ bats tests/a/*.bats tests/b/*.bats tests/b/c/*.bats
 ✓ a
 ✓ b
 ✓ c

3 tests, 0 failures
```

Because specifying wildcards in this way, with arbitrary depths in the hierarchy of directories below `tests/` is too hard to predict, use a program like `find` to identify tests by name (possibly using wildcards or `grep` filters for names), passing the results on to a program like `xargs` to invoke `bats` on each identified test:

```
$ find tests -name '*.bats' | xargs bats
1..3
ok 1 a
ok 2 b
ok 3 c
```

Note: Note that the output changed from the examples above, which include the arrow ("✓") character, to now include the word `ok` instead in TAP format. This is because the default for terminals (i.e., a program that is using a TTY device, not a simple file handle to something like a pipe). To get the pretty-print output, add the `-p` flag, like this:

```
$ find tests -name '*.bats' | xargs bats -p
 ✓ a
 ✓ b
 ✓ c

3 tests, 0 failures
```

A more realistic test is seen here. This file, `pycharm.bats`, is the product of a Jinja template that is installed by Ansible along with the PyCharm Community Edition Python IDE.

```
#!/usr/bin/env bats
#
# Ansible managed: /home/dittrich/dims/git/ansible-playbooks/v2/roles/pycharm/
↪templates/../templates/tests/./system/pycharm.bats.j2 modified on 2016-09-15␣
↪20:14:38 by dittrich on dimsdemo1 [ansible-playbooks v1.3.33]
#
# vim: set ts=4 sw=4 tw=0 et :

load helpers
```

(continues on next page)

(continued from previous page)

```
@test "[S][EV] Pycharm is not an installed apt package." {
    ! is_installed_package pycharm
}

@test "[S][EV] Pycharm Community edition is installed in /opt" {
    results=$(ls -d /opt/pycharm-community-* | wc -l)
    echo $results >&2
    [ $results -ne 0 ]
}

@test "[S][EV] \"pycharm\" is /opt/dims/bin/pycharm" {
    assert "pycharm is /opt/dims/bin/pycharm" type pycharm
}

@test "[S][EV] /opt/dims/bin/pycharm is a symbolic link to installed pycharm" {
    [ -L /opt/dims/bin/pycharm ]
}

@test "[S][EV] Pycharm Community installed version number is 2016.2.3" {
    assert "2016.2.3" bash -c "file $(which pycharm) | sed 's|\(.*/pycharm-community-
↪\)\([^/]*\)\(/.*$\)|\2|'"
}
```

```
$ test.runner --level system --match pycharm
[+] Running test system/pycharm
 ✓ [S][EV] Pycharm is not an installed apt package.
 ✓ [S][EV] Pycharm Community edition is installed in /opt
 ✓ [S][EV] "pycharm" is /opt/dims/bin/pycharm
 ✓ [S][EV] /opt/dims/bin/pycharm is a symbolic link to installed pycharm
 ✓ [S][EV] Pycharm Community installed version number is 2016.2.3

5 tests, 0 failures
```

6.2 Organizing tests in DIMS Ansible Playbooks Roles

The DIMS project uses a more elaborate version of the above example, which uses a *drop-in* model that allows any Ansible role to drop its own tests into a structured hierarchy that supports fine-grained test execution control. This drop-in model is implemented by the `tasks/bats-tests.yml` task playbook.

To illustrate how this works, we start with an empty test directory:

```
$ tree /opt/dims/tests.d
/opt/dims/tests.d

0 directories, 0 files
```

The `base` role has the largest number of tests, since it does the most complex foundational setup work for DIMS computer systems. The `template/tests` directory is filled with Jinja template Bash scripts and/or `bats` tests, in a hierarchy that includes subdirectories for each of the defined test levels from Section Test levels of dimstp.

```
$ tree roles/base/templates/tests
roles/base/templates/tests
+── component
```

(continues on next page)

(continued from previous page)

```
+-- helpers.bash.j2
+-- integration
│   +-- dims-coreos.bats.j2
│   +-- proxy.bats.j2
+-- README.txt
+-- rsyslog.bats
+-- system
│   +-- deprecated.bats.j2
│   +-- dims-accounts.bats.j2
│   +-- dims-accounts-sudo.bats.j2
│   +-- dims-base.bats.j2
│   +-- dims-coreos.bats.j2
│   +-- dns.bats.j2
│   +-- iptables-sudo.bats.j2
│   +-- reboot.bats.j2
│   +-- updates.bats.j2
+-- unit
│   +-- ansible-yaml.bats.j2
│   +-- bats-helpers.bats.j2
│   +-- dims-filters.bats.j2
│   +-- dims_functions.bats.j2
+-- user
    +-- user-account.bats.j2
    +-- user-deprecated.bats.j2

5 directories, 20 files
```

After running just the `base` role, the highlighted subdirectories that correspond to each of the test levels are now present in the `/opt/dims/tests.d/` directory:

```
$ tree /opt/dims/tests.d/
/opt/dims/tests.d
+-- component
│   +-- helpers.bash -> /opt/dims/tests.d/helpers.bash
+-- helpers.bash
+-- integration
│   +-- dims-coreos.bats
│   +-- helpers.bash -> /opt/dims/tests.d/helpers.bash
│   +-- proxy.bats
+-- system
│   +-- deprecated.bats
│   +-- dims-accounts.bats
│   +-- dims-accounts-sudo.bats
│   +-- dims-base.bats
│   +-- dims-ci-utils.bats
│   +-- dims-coreos.bats
│   +-- dns.bats
│   +-- helpers.bash -> /opt/dims/tests.d/helpers.bash
│   +-- iptables-sudo.bats
│   +-- reboot.bats
│   +-- updates.bats
+-- unit
│   +-- ansible-yaml.bats
│   +-- bats-helpers.bats
│   +-- dims-filters.bats
│   +-- dims_functions.bats
```

(continues on next page)

(continued from previous page)

```
│   +-- helpers.bash -> /opt/dims/tests.d/helpers.bash
+-- user
    +-- helpers.bash -> /opt/dims/tests.d/helpers.bash
    +-- user-account.bats
    +-- user-deprecated.bats

5 directories, 24 files
```

Here is the directory structure for tests in the `docker` role:

```
/docker/templates/tests
+-- system
    +-- docker-core.bats.j2
    +-- docker-network.bats.j2

1 directories, 2 files
```

If we now run the `docker` role, it will drop these files into the `system` subdirectory. There are now 3 additional files (see emphasized lines for the new additions):

```
$ tree /opt/dims/tests.d
/opt/dims/tests.d
+-- component
│   +-- helpers.bash -> /opt/dims/tests.d/helpers.bash
+-- helpers.bash
+-- integration
│   +-- dims-coreos.bats
│   +-- helpers.bash -> /opt/dims/tests.d/helpers.bash
│   +-- proxy.bats
+-- system
│   +-- deprecated.bats
│   +-- dims-accounts.bats
│   +-- dims-accounts-sudo.bats
│   +-- dims-base.bats
│   +-- dims-ci-utils.bats
│   +-- dims-coreos.bats
│   +-- dns.bats
│   +-- docker-core.bats
│   +-- docker-network.bats
│   +-- helpers.bash -> /opt/dims/tests.d/helpers.bash
│   +-- iptables-sudo.bats
│   +-- reboot.bats
│   +-- updates.bats
+-- unit
│   +-- ansible-yaml.bats
│   +-- bats-helpers.bats
│   +-- dims-filters.bats
│   +-- dims_functions.bats
│   +-- helpers.bash -> /opt/dims/tests.d/helpers.bash
+-- user
    +-- helpers.bash -> /opt/dims/tests.d/helpers.bash
    +-- user-account.bats
    +-- user-deprecated.bats

5 directories, 26 files
```

You will see the tests being installed during `ansible-playbook` runs, for example (from the `base` role):

```
. . .

TASK [base : Identify bats test templates] ************************************
Sunday 03 September 2017  13:05:45 -0700 (0:00:05.496)       0:03:48.846 ******
ok: [dimsdemo1.devops.develop -> 127.0.0.1]

TASK [base : Initialize bats_test_templates list] *****************************
Sunday 03 September 2017  13:05:46 -0700 (0:00:01.152)       0:03:49.998 ******
ok: [dimsdemo1.devops.develop]

TASK [base : Set fact with list of test templates] ****************************
Sunday 03 September 2017  13:05:47 -0700 (0:00:01.047)       0:03:51.046 ******
ok: [dimsdemo1.devops.develop] => (item=/home/dittrich/dims/git/ansible-dims-
↪playbooks/roles/base/templates/tests/system/dims-coreos.bats.j2)
ok: [dimsdemo1.devops.develop] => (item=/home/dittrich/dims/git/ansible-dims-
↪playbooks/roles/base/templates/tests/system/dims-base.bats.j2)
ok: [dimsdemo1.devops.develop] => (item=/home/dittrich/dims/git/ansible-dims-
↪playbooks/roles/base/templates/tests/system/dims-accounts-sudo.bats.j2)
ok: [dimsdemo1.devops.develop] => (item=/home/dittrich/dims/git/ansible-dims-
↪playbooks/roles/base/templates/tests/system/updates.bats.j2)
ok: [dimsdemo1.devops.develop] => (item=/home/dittrich/dims/git/ansible-dims-
↪playbooks/roles/base/templates/tests/system/reboot.bats.j2)
ok: [dimsdemo1.devops.develop] => (item=/home/dittrich/dims/git/ansible-dims-
↪playbooks/roles/base/templates/tests/system/dns.bats.j2)
ok: [dimsdemo1.devops.develop] => (item=/home/dittrich/dims/git/ansible-dims-
↪playbooks/roles/base/templates/tests/system/deprecated.bats.j2)
ok: [dimsdemo1.devops.develop] => (item=/home/dittrich/dims/git/ansible-dims-
↪playbooks/roles/base/templates/tests/system/dims-accounts.bats.j2)
ok: [dimsdemo1.devops.develop] => (item=/home/dittrich/dims/git/ansible-dims-
↪playbooks/roles/base/templates/tests/system/iptables-sudo.bats.j2)
ok: [dimsdemo1.devops.develop] => (item=/home/dittrich/dims/git/ansible-dims-
↪playbooks/roles/base/templates/tests/integration/dims-coreos.bats.j2)
ok: [dimsdemo1.devops.develop] => (item=/home/dittrich/dims/git/ansible-dims-
↪playbooks/roles/base/templates/tests/integration/proxy.bats.j2)
ok: [dimsdemo1.devops.develop] => (item=/home/dittrich/dims/git/ansible-dims-
↪playbooks/roles/base/templates/tests/user/user-account.bats.j2)
ok: [dimsdemo1.devops.develop] => (item=/home/dittrich/dims/git/ansible-dims-
↪playbooks/roles/base/templates/tests/user/user-deprecated.bats.j2)
ok: [dimsdemo1.devops.develop] => (item=/home/dittrich/dims/git/ansible-dims-
↪playbooks/roles/base/templates/tests/unit/bats-helpers.bats.j2)
ok: [dimsdemo1.devops.develop] => (item=/home/dittrich/dims/git/ansible-dims-
↪playbooks/roles/base/templates/tests/unit/dims-filters.bats.j2)
ok: [dimsdemo1.devops.develop] => (item=/home/dittrich/dims/git/ansible-dims-
↪playbooks/roles/base/templates/tests/unit/ansible-yaml.bats.j2)
ok: [dimsdemo1.devops.develop] => (item=/home/dittrich/dims/git/ansible-dims-
↪playbooks/roles/base/templates/tests/unit/dims_functions.bats.j2)

TASK [base : debug] ***********************************************************
Sunday 03 September 2017  13:06:04 -0700 (0:00:17.532)       0:04:08.578 ******
ok: [dimsdemo1.devops.develop] => {
    "bats_test_templates": [
        "system/dims-coreos.bats.j2",
        "system/dims-base.bats.j2",
        "system/dims-accounts-sudo.bats.j2",
        "system/updates.bats.j2",
        "system/reboot.bats.j2",
        "system/dns.bats.j2",
```

(continues on next page)

(continued from previous page)

```
        "system/deprecated.bats.j2",
        "system/dims-accounts.bats.j2",
        "system/iptables-sudo.bats.j2",
        "integration/dims-coreos.bats.j2",
        "integration/proxy.bats.j2",
        "user/user-account.bats.j2",
        "user/user-deprecated.bats.j2",
        "unit/bats-helpers.bats.j2",
        "unit/dims-filters.bats.j2",
        "unit/ansible-yaml.bats.j2",
        "unit/dims_functions.bats.j2"
    ]
}

TASK [base : Make defined bats tests present] *********************************
Sunday 03 September 2017  13:06:05 -0700 (0:00:01.053)       0:04:09.631 ******
changed: [dimsdemo1.devops.develop] => (item=system/dims-coreos.bats.j2)
changed: [dimsdemo1.devops.develop] => (item=system/dims-base.bats.j2)
changed: [dimsdemo1.devops.develop] => (item=system/dims-accounts-sudo.bats.j2)
changed: [dimsdemo1.devops.develop] => (item=system/updates.bats.j2)
changed: [dimsdemo1.devops.develop] => (item=system/reboot.bats.j2)
changed: [dimsdemo1.devops.develop] => (item=system/dns.bats.j2)
changed: [dimsdemo1.devops.develop] => (item=system/deprecated.bats.j2)
changed: [dimsdemo1.devops.develop] => (item=system/dims-accounts.bats.j2)
changed: [dimsdemo1.devops.develop] => (item=system/iptables-sudo.bats.j2)
changed: [dimsdemo1.devops.develop] => (item=integration/dims-coreos.bats.j2)
changed: [dimsdemo1.devops.develop] => (item=integration/proxy.bats.j2)
changed: [dimsdemo1.devops.develop] => (item=user/user-account.bats.j2)
changed: [dimsdemo1.devops.develop] => (item=user/user-deprecated.bats.j2)
changed: [dimsdemo1.devops.develop] => (item=unit/bats-helpers.bats.j2)
changed: [dimsdemo1.devops.develop] => (item=unit/dims-filters.bats.j2)
changed: [dimsdemo1.devops.develop] => (item=unit/ansible-yaml.bats.j2)
changed: [dimsdemo1.devops.develop] => (item=unit/dims_functions.bats.j2)

. . .
```

Tests can now be run by level, multiple levels at the same time, or more fine-grained filtering can be performed using `find` and `grep` filtering.

6.3 Running Bats Tests Using the DIMS `test.runner`

A test runner script (creatively named `test.runner`) is available to This script builds on and extends the capabilities of scipts like test_runner.sh from the GitHub docker/swarm/test/integration repository.

```
$ base/templates/tests/test.runner --help
usage: test.runner [options] args
flags:
  -d,--[no]debug:  enable debug mode (default: false)
  -E,--exclude:  tests to exclude (default: '')
  -L,--level:  test level (default: 'system')
  -M,--match:  regex to match tests (default: '.*')
  -l,--[no]list-tests:  list available tests (default: false)
  -t,--[no]tap:  output tap format (default: false)
  -S,--[no]sudo-tests:  perform sudo tests (default: false)
```

(continues on next page)

(continued from previous page)

```
  -T,--[no]terse:  print only failed tests (default: false)
  -D,--testdir:  test directory (default: '/opt/dims/tests.d/')
  -u,--[no]usage:  print usage information (default: false)
  -v,--[no]verbose:  be verbose (default: false)
  -h,--help:  show this help (default: false)
```

To see a list of all tests under a given test level, specify the level using the `--level` option. (The default is `system`). The following example shows a list of all the available `system` level tests:

```
$ test.runner --list-tests
system/dims-base.bats
system/pycharm.bats
system/dns.bats
system/docker.bats
system/dims-accounts.bats
system/dims-ci-utils.bats
system/deprecated.bats
system/coreos-prereqs.bats
system/user/vpn.bats
system/proxy.bats
```

To see all tests under any level, use `*` or a space-separated list of levels:

```
$ test.runner --level "*" --list-tests
system/dims-base.bats
system/pycharm.bats
system/dns.bats
system/docker.bats
system/dims-accounts.bats
system/dims-ci-utils.bats
system/deprecated.bats
system/coreos-prereqs.bats
system/user/vpn.bats
system/proxy.bats
unit/dims-filters.bats
unit/bats-helpers.bats
```

Certain tests that require elevated privileges (i.e., use of `sudo`) are handled separately. To list or run these tests, use the `--sudo-tests` option:

```
$ test.runner --list-tests --sudo-tests
system/dims-accounts-sudo.bats
system/iptables-sudo.bats
```

A subset of the tests can be selected using the `--match` option. To see all tests that include the word `dims`, do:

```
$ test.runner --level system --match dims --list-tests
system/dims-base.bats
system/dims-accounts.bats
system/dims-ci-utils.bats
```

The `--match` option takes a an `egrep` expression to filter the selected tests, so multiple substrings (or regular expressions) can be passed with pipe separation:

```
$ test.runner --level system --match "dims|coreos" --list-tests
system/dims-base.bats
```

(continues on next page)

(continued from previous page)

```
system/dims-accounts.bats
system/dims-ci-utils.bats
system/coreos-prereqs.bats
```

There is a similar option `--exclude` that filters out tests by `egrep` regular expression. Two of the four selected tests are then excluded like this:

```
$ test.runner --level system --match "dims|coreos" --exclude "base|utils" --list-tests
system/dims-accounts.bats
system/coreos-prereqs.bats
```

6.4 Controlling the Amount and Type of Output

The default for the `bats` program is to use `--pretty` formatting when standard output is being sent to a terminal. This allows the use of colors and characters like ✓ and to be used for passed and failed tests (respectively).

```
$ bats --help

[No write since last change]
Bats 0.4.0
Usage: bats [-c] [-p | -t] <test> [<test> ...]

  <test> is the path to a Bats test file, or the path to a directory
  containing Bats test files.

  -c, --count    Count the number of test cases without running any tests
  -h, --help     Display this help message
  -p, --pretty   Show results in pretty format (default for terminals)
  -t, --tap      Show results in TAP format
  -v, --version  Display the version number

  For more information, see https://github.com/sstephenson/bats


Press ENTER or type command to continue
```

We will limit the tests in this example to just those for `pycharm` and `coreos` in their names. These are relatively small tests, so it is easier to see the effects of the options we will be examining.

```
$ test.runner --match "pycharm|coreos" --list-tests
system/pycharm.bats
system/coreos-prereqs.bats
```

The DIMS `test.runner` script follows this same default output style of `bats`, so just running the two tests above gives the following output:

```
$ test.runner --match "pycharm|coreos"
[+] Running test system/pycharm.bats
 ✓ [S][EV] Pycharm is not an installed apt package.
 ✓ [S][EV] Pycharm Community edition is installed in /opt
 ✓ [S][EV] "pycharm" is /opt/dims/bin/pycharm
 ✓ [S][EV] /opt/dims/bin/pycharm is a symbolic link to installed pycharm
 ✓ [S][EV] Pycharm Community installed version number is 2016.2.2

```

(continues on next page)

(continued from previous page)

```
5 tests, 0 failures
[+] Running test system/coreos-prereqs.bats
 ✓ [S][EV] consul service is running
 ✓ [S][EV] consul is /opt/dims/bin/consul
 ✓ [S][EV] 10.142.29.116 is member of consul cluster
 ✓ [S][EV] 10.142.29.117 is member of consul cluster
 ✓ [S][EV] 10.142.29.120 is member of consul cluster
 ✓ [S][EV] docker overlay network "ingress" exists
  [S][EV] docker overlay network "app.develop" exists
   (from function `assert' in file system/helpers.bash, line 18,
    in test file system/coreos-prereqs.bats, line 41)
     `assert 'app.develop' bash -c "docker network ls --filter driver=overlay | awk '/
↪app.develop/ { print \$2; }'"' failed
   expected: "app.develop"
   actual:   ""
  [S][EV] docker overlay network "data.develop" exists
   (from function `assert' in file system/helpers.bash, line 18,
    in test file system/coreos-prereqs.bats, line 45)
     `assert 'data.develop' bash -c "docker network ls --filter driver=overlay | awk
↪'/data.develop/ { print \$2; }'"' failed
   expected: "data.develop"
   actual:   ""

8 tests, 2 failures
```

To get TAP compliant output, add the `--tap` (or `-t`) option:

```
$ test.runner --match "pycharm|coreos" --tap
[+] Running test system/pycharm.bats
1..5
ok 1 [S][EV] Pycharm is not an installed apt package.
ok 2 [S][EV] Pycharm Community edition is installed in /opt
ok 3 [S][EV] "pycharm" is /opt/dims/bin/pycharm
ok 4 [S][EV] /opt/dims/bin/pycharm is a symbolic link to installed pycharm
ok 5 [S][EV] Pycharm Community installed version number is 2016.2.2
[+] Running test system/coreos-prereqs.bats
1..8
ok 1 [S][EV] consul service is running
ok 2 [S][EV] consul is /opt/dims/bin/consul
ok 3 [S][EV] 10.142.29.116 is member of consul cluster
ok 4 [S][EV] 10.142.29.117 is member of consul cluster
ok 5 [S][EV] 10.142.29.120 is member of consul cluster
ok 6 [S][EV] docker overlay network "ingress" exists
not ok 7 [S][EV] docker overlay network "app.develop" exists
# (from function `assert' in file system/helpers.bash, line 18,
#  in test file system/coreos-prereqs.bats, line 41)
#   `assert 'app.develop' bash -c "docker network ls --filter driver=overlay | awk '/
↪app.develop/ { print \$2; }'"' failed
# expected: "app.develop"
# actual:   ""
not ok 8 [S][EV] docker overlay network "data.develop" exists
# (from function `assert' in file system/helpers.bash, line 18,
#  in test file system/coreos-prereqs.bats, line 45)
#   `assert 'data.develop' bash -c "docker network ls --filter driver=overlay | awk '/
↪data.develop/ { print \$2; }'"' failed
# expected: "data.develop"
# actual:   ""
```

When running a large suite of tests, the total number of individual tests can get very large (along with the resulting output). To increase the signal to noise ratio, you can use the `--terse` option to filter out all of the successful tests, just focusing on the remaining failed tests. This is handy for things like validation of code changes and regression testing of newly provisioned Vagrant virtual machines.

```
$ test.runner --match "pycharm|coreos" --terse
[+] Running test system/pycharm.bats

5 tests, 0 failures
[+] Running test system/coreos-prereqs.bats
  [S][EV] docker overlay network "app.develop" exists
   (from function `assert' in file system/helpers.bash, line 18,
    in test file system/coreos-prereqs.bats, line 41)
     `assert 'app.develop' bash -c "docker network ls --filter driver=overlay | awk '/
↪app.develop/ { print \$2; }'"' failed
   expected: "app.develop"
   actual:   ""
  [S][EV] docker overlay network "data.develop" exists
   (from function `assert' in file system/helpers.bash, line 18,
    in test file system/coreos-prereqs.bats, line 45)
     `assert 'data.develop' bash -c "docker network ls --filter driver=overlay | awk
↪'/data.develop/ { print \$2; }'"' failed
   expected: "data.develop"
   actual:   ""

8 tests, 2 failures
```

Here is the same examples as above, but this time using the TAP compliant output:

```
$ test.runner --match "pycharm|coreos" --tap
[+] Running test system/pycharm.bats
1..5
ok 1 [S][EV] Pycharm is not an installed apt package.
ok 2 [S][EV] Pycharm Community edition is installed in /opt
ok 3 [S][EV] "pycharm" is /opt/dims/bin/pycharm
ok 4 [S][EV] /opt/dims/bin/pycharm is a symbolic link to installed pycharm
ok 5 [S][EV] Pycharm Community installed version number is 2016.2.2
[+] Running test system/coreos-prereqs.bats
1..8
ok 1 [S][EV] consul service is running
ok 2 [S][EV] consul is /opt/dims/bin/consul
ok 3 [S][EV] 10.142.29.116 is member of consul cluster
ok 4 [S][EV] 10.142.29.117 is member of consul cluster
ok 5 [S][EV] 10.142.29.120 is member of consul cluster
ok 6 [S][EV] docker overlay network "ingress" exists
not ok 7 [S][EV] docker overlay network "app.develop" exists
# (from function `assert' in file system/helpers.bash, line 18,
#  in test file system/coreos-prereqs.bats, line 41)
#   `assert 'app.develop' bash -c "docker network ls --filter driver=overlay | awk '/
↪app.develop/ { print \$2; }'"' failed
# expected: "app.develop"
# actual:   ""
not ok 8 [S][EV] docker overlay network "data.develop" exists
# (from function `assert' in file system/helpers.bash, line 18,
#  in test file system/coreos-prereqs.bats, line 45)
#   `assert 'data.develop' bash -c "docker network ls --filter driver=overlay | awk '/
↪data.develop/ { print \$2; }'"' failed
# expected: "data.develop"
```

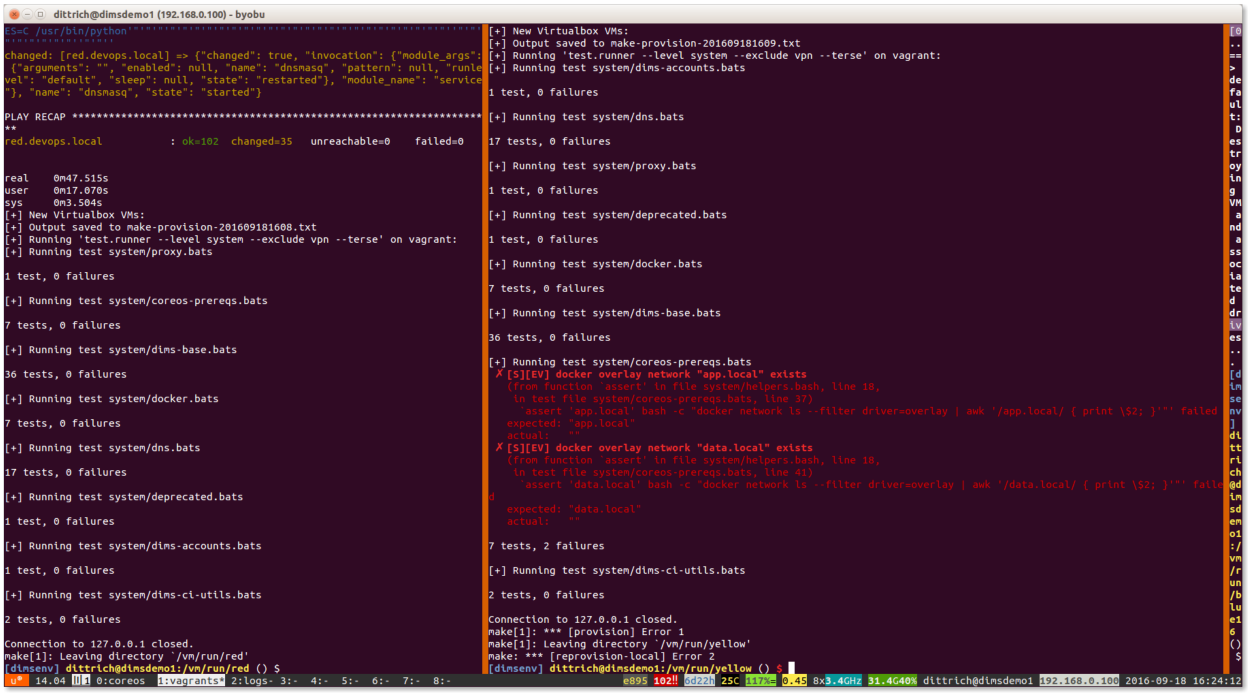
(continues on next page)

(continued from previous page)

```
# actual:   ""
```

```
$ test.runner --match "pycharm|coreos" --tap --terse
[+] Running test system/pycharm.bats
1..5
[+] Running test system/coreos-prereqs.bats
1..8
not ok 7 [S][EV] docker overlay network "app.develop" exists
# (from function `assert' in file system/helpers.bash, line 18,
#  in test file system/coreos-prereqs.bats, line 41)
#   `assert 'app.develop' bash -c "docker network ls --filter driver=overlay | awk '/
↪app.develop/ { print \$2; }'"' failed
# expected: "app.develop"
# actual:   ""
not ok 8 [S][EV] docker overlay network "data.develop" exists
# (from function `assert' in file system/helpers.bash, line 18,
#  in test file system/coreos-prereqs.bats, line 45)
#   `assert 'data.develop' bash -c "docker network ls --filter driver=overlay | awk '/
↪data.develop/ { print \$2; }'"' failed
# expected: "data.develop"
# actual:   ""
```

Figure *Using test.runner in Vagrant Provisioning* shows the output of `test.runner --level system --terse` at the completion of provisioning of two Vagrants. The one on the left has passed all tests, while the Vagrant on the right has failed two tests. Note that the error result has been passed on to `make`, which reports the failure and passes it along to the shell (as seen by the red `$` prompt on the right, indicating a non-zero return value).



```
ES=C /usr/bin/python'"'"'"'"'"'"'"'"'"'"'"'"'"'"'"'"'"'"'"'"'"'"'"'"'"'"'"'"'"'"'
"'"'"'"'"'"'"'"'"'
changed: [red.devops.local] => {"changed": true, "invocation": {"module_args":
 {"arguments": "", "enabled": null, "name": "dnsmasq", "pattern": null, "runle
vel": "default", "sleep": null, "state": "restarted"}, "module_name": "service
"}, "name": "dnsmasq", "state": "started"}

PLAY RECAP *********************************************************************
**
red.devops.local           : ok=102  changed=35   unreachable=0    failed=0


real    0m47.515s
user    0m17.070s
sys     0m3.504s
[+] New Virtualbox VMs:
[+] Output saved to make-provision-201609181608.txt
[+] Running 'test.runner --level system --exclude vpn --terse' on vagrant:
[+] Running test system/proxy.bats

1 test, 0 failures

[+] Running test system/coreos-prereqs.bats

7 tests, 0 failures

[+] Running test system/dims-base.bats

36 tests, 0 failures

[+] Running test system/docker.bats

7 tests, 0 failures

[+] Running test system/dns.bats

17 tests, 0 failures

[+] Running test system/deprecated.bats

1 test, 0 failures

[+] Running test system/dims-accounts.bats

1 test, 0 failures

[+] Running test system/dims-ci-utils.bats

2 tests, 0 failures

Connection to 127.0.0.1 closed.
make[1]: Leaving directory `/vm/run/red'
[dimsenv] dittrich@dimsdemo1:/vm/run/red () $
```

```
[+] New Virtualbox VMs:
[+] Output saved to make-provision-201609181609.txt
[+] Running 'test.runner --level system --exclude vpn --terse' on vagrant:
[+] Running test system/dims-accounts.bats

1 test, 0 failures

[+] Running test system/dns.bats

17 tests, 0 failures

[+] Running test system/proxy.bats

1 test, 0 failures

[+] Running test system/deprecated.bats

1 test, 0 failures

[+] Running test system/docker.bats

7 tests, 0 failures

[+] Running test system/dims-base.bats

36 tests, 0 failures

[+] Running test system/coreos-prereqs.bats
 ✗ [S][EV] docker overlay network "app.local" exists
   (from function `assert' in file system/helpers.bash, line 18,
    in test file system/coreos-prereqs.bats, line 37)
     `assert 'app.local' bash -c "docker network ls --filter driver=overlay | awk '/app.local/ { print \$2; }'"' failed
   expected: "app.local"
   actual:   ""
 ✗ [S][EV] docker overlay network "data.local" exists
   (from function `assert' in file system/helpers.bash, line 18,
    in test file system/coreos-prereqs.bats, line 41)
     `assert 'data.local' bash -c "docker network ls --filter driver=overlay | awk '/data.local/ { print \$2; }'"' faile
d
   expected: "data.local"
   actual:   ""

7 tests, 2 failures

[+] Running test system/dims-ci-utils.bats

2 tests, 0 failures

Connection to 127.0.0.1 closed.
make[1]: *** [provision] Error 1
make[1]: Leaving directory `/vm/run/yellow'
make: *** [reprovision-local] Error 2
[dimsenv] dittrich@dimsdemo1:/vm/run/yellow () $
```

```
[0
..
==
>
de
fa
ul
t:
 D
es
tr
oy
in
g
VM
 a
nd
 a
ss
oc
ia
te
d
dr
iv
es
..
.
[d
im
se
nv
]
di
tt
ri
ch
@d
im
sd
em
o1
:/
vm
/r
un
/b
lu
e1
6
()
 $
```

u 14.04 1 0:coreos 1:vagrants* 2:logs- 3:- 4:- 5:- 6:- 7:- 8:-
e895 102!! 6d22h 25C 117%= 0.45 8x3.4GHz 31.4G40% dittrich@dimsdemo1 192.168.0.100 2016-09-18 16:24:12

Fig. 1: Using `test.runner` in Vagrant Provisioning

6.4.1 Using DIMS Bash functions in Bats tests

The DIMS project Bash shells take advantage of a library of functions that are installed by the `base` role into `$DIMS/bin/dims_functions.sh`.

Bats has a pre- and post-test hooking feature that is very tersely documented (see setup and teardown: Pre- and post-test hooks):

> You can define special setup and teardown functions, which run before and after each test case, respectively. Use these to load fixtures, set up your environment, and clean up when you're done.

What this means is that if you define a `setup()` function, it will be run *before* every `@test`, and if you define a `teardown()` function, it will be run *after* every `@test`.

We can take advantage of this to source the common DIMS `dims_functions.sh` library, making any defined functions in that file available to be called directly in a `@TEST` the same way it would be called in a Bash script.

An example of how this works can be seen in the unit tests for the `dims_functions.sh` library itself. (We are only showing a sub-set of the tests.)

- Lines 9-16 perform the `setup()` actions (e.g., creating directories used in a later test.)
- Lines 18-21 perform the `teardown()` actions.
- All of the remaining highlighted lines use functions defined in `dims_functions.sh` just as if sourced in a normal Bash script.

```
1  #!/usr/bin/env bats
2  #
3  # {{ ansible_managed }} [ansible-playbooks v{{ ansibleplaybooks_version }}]
4  #
5  # vim: set ts=4 sw=4 tw=0 et :
6
7  load helpers
8
9  function setup() {
10     source $DIMS/bin/dims_functions.sh
11     touch --reference=/bin/ls /tmp/bats.ls-marker
12     for name in A B; do
13         mkdir -p /tmp/bats.tmp/${name}.dir
14         touch /tmp/bats.tmp/${name}.txt
15     done
16 }
17
18 function teardown() {
19     rm -f /tmp/bats.ls-marker
20     rm -rf /tmp/bats.tmp
21 }
22
23 @test "[U][EV] say() strips whitespace properly" {
24     assert '[+] unce, tice, fee times a madie...' say 'unce, tice,  fee    times a␣
   ↪madie...      '
25 }
26
27 @test "[U][EV] say_raw() does not strip whitespace" {
28     assert '[+] unce, tice,  fee   times a madie...      ' say_raw 'unce, tice,  fee ␣
   ↪ times a madie...      '
29 }
30
31 # This test needs to directly source dims_functions in bash command string because of␣
   ↪multi-command structure.
```

(continues on next page)

(continued from previous page)

```
32 @test "[U][EV] add_on_exit() saves and get_on_exit() returns content properly" {
33     assert "'([0]=\"cat /dev/null\")'" bash -c ". $DIMS/bin/dims_functions.sh; touch /tmp/foo; add_on_exit cat /dev/null; get_on_exit"
34 }
35
36 @test "[U][EV] get_hostname() returns hostname" {
37     assert "$(hostname)" get_hostname
38 }
39
40 @test "[U][EV] is_fqdn host.category.deployment returns success" {
41     is_fqdn host.category.deployment
42 }
43
44 @test "[U][EV] is_fqdn host.subdomain.category.deployment returns success" {
45     is_fqdn host.subdomain.category.deployment
46 }
47
48 @test "[U][EV] is_fqdn 12345 returns failure" {
49     ! is_fqdn 12345
50 }
51
52 @test "[U][EV] parse_fqdn host.category.deployment returns 'host category deployment'" {
53     assert "host category deployment" parse_fqdn host.category.deployment
54 }
55
56 @test "[U][EV] get_deployment_from_fqdn host.category.deployment returns 'deployment'" {
57     assert "deployment" get_deployment_from_fqdn host.category.deployment
58 }
59
60 @test "[U][EV] get_category_from_fqdn host.category.deployment returns 'category'" {
61     assert "category" get_category_from_fqdn host.category.deployment
62 }
63
64 @test "[U][EV] get_hostname_from_fqdn host.category.deployment returns 'host'" {
65     assert "host" get_hostname_from_fqdn host.category.deployment
66 }
```

> **Attention:** Note that there is one test, shown on lines 31 through 34, that has multiple commands separated by semicolons. That compound command sequence needs to be run as a single command string using `bash -c`, which means it is going to be run as a new sub-process to the `assert` command line. Sourcing the functions in the outer shell does not make them available in the sub-process, so that command string must itself also source the `dims_functions.sh` library in order to have the functions defined at that level.

Another place that a `bats` unit test is employed is the `python-virtualenv` role, which loads a number of `pip` packages and utilities used for DIMS development. This build process is quite extensive and produces thousands of lines of output that may be necessary to debug a problem in the build process, but create a *huge* amount of noise if no needed. To avoid spewing out so much noisy text, it is only shown if `-v` (or higher verbosity level) is selected.

Here is the output when a failure occurs without verbosity:

```
$ run.playbook --tags python-virtualenv
. . .
```

(continues on next page)

(continued from previous page)

```
TASK [python-virtualenv : Run dimsenv.build script] ***************************
Tuesday 01 August 2017  19:00:10 -0700 (0:00:02.416)       0:01:13.310 ********
changed: [dimsdemo1.devops.develop]

TASK [python-virtualenv : Run unit test for Python virtualenv] ****************
Tuesday 01 August 2017  19:02:16 -0700 (0:02:06.294)       0:03:19.605 ********
fatal: [dimsdemo1.devops.develop]: FAILED! => {
    "changed": true,
    "cmd": [
        "/opt/dims/bin/test.runner",
        "--tap",
        "--level",
        "unit",
        "--match",
        "python-virtualenv"
    ],
    "delta": "0:00:00.562965",
    "end": "2017-08-01 19:02:18.579603",
    "failed": true,
    "rc": 1,
    "start": "2017-08-01 19:02:18.016638"
}

STDOUT:

# [+] Running test unit/python-virtualenv
1..17
ok 1 [S][EV] Directory /opt/dims/envs/dimsenv exists
ok 2 [U][EV] Directory /opt/dims/envs/dimsenv is not empty
ok 3 [U][EV] Directories /opt/dims/envs/dimsenv/{bin,lib,share} exist
ok 4 [U][EV] Program /opt/dims/envs/dimsenv/bin/python exists
ok 5 [U][EV] Program /opt/dims/envs/dimsenv/bin/pip exists
ok 6 [U][EV] Program /opt/dims/envs/dimsenv/bin/easy_install exists
ok 7 [U][EV] Program /opt/dims/envs/dimsenv/bin/wheel exists
ok 8 [U][EV] Program /opt/dims/envs/dimsenv/bin/python-config exists
ok 9 [U][EV] Program /opt/dims/bin/virtualenvwrapper.sh exists
ok 10 [U][EV] Program /opt/dims/envs/dimsenv/bin/activate exists
ok 11 [U][EV] Program /opt/dims/envs/dimsenv/bin/logmon exists
not ok 12 [U][EV] Program /opt/dims/envs/dimsenv/bin/blueprint exists
# (in test file unit/python-virtualenv.bats, line 54)
#   `[[ -x /opt/dims/envs/dimsenv/bin/blueprint ]]' failed
not ok 13 [U][EV] Program /opt/dims/envs/dimsenv/bin/dimscli exists
# (in test file unit/python-virtualenv.bats, line 58)
#   `[[ -x /opt/dims/envs/dimsenv/bin/dimscli ]]' failed
not ok 14 [U][EV] Program /opt/dims/envs/dimsenv/bin/sphinx-autobuild exists
# (in test file unit/python-virtualenv.bats, line 62)
#   `[[ -x /opt/dims/envs/dimsenv/bin/sphinx-autobuild ]]' failed
not ok 15 [U][EV] Program /opt/dims/envs/dimsenv/bin/ansible exists
# (in test file unit/python-virtualenv.bats, line 66)
#   `[[ -x /opt/dims/envs/dimsenv/bin/ansible ]]' failed
not ok 16 [U][EV] /opt/dims/envs/dimsenv/bin/dimscli version is 0.26.0
# (from function `assert' in file unit/helpers.bash, line 13,
#  in test file unit/python-virtualenv.bats, line 71)
#   `assert "dimscli 0.26.0" bash -c "/opt/dims/envs/dimsenv/bin/dimscli --version 2>&
↪1"' failed with status 127
not ok 17 [U][EV] /opt/dims/envs/dimsenv/bin/ansible version is 2.3.1.0
# (from function `assert' in file unit/helpers.bash, line 18,
```

(continues on next page)

(continued from previous page)

```
#  in test file unit/python-virtualenv.bats, line 76)
#   `assert "ansible 2.3.1.0" bash -c "/opt/dims/envs/dimsenv/bin/ansible --version 2>
↪&1 | head -n1"' failed
# expected: "ansible 2.3.1.0"
# actual:   "bash: /opt/dims/envs/dimsenv/bin/ansible: No such file or directory"
#

PLAY RECAP *********************************************************************
dimsdemo1.devops.develop   : ok=49   changed=7    unreachable=0    failed=1
. . .
```

To find out what the problem is, run the build again and add at least one `-v`:

```
 $ run.playbook -v --tags python-virtualenv
 . . .
 TASK [python-virtualenv : Run dimsenv.build script] ***************************
 Tuesday 01 August 2017  18:54:22 -0700 (0:00:02.437)       0:01:32.394 ********
 changed: [dimsdemo1.devops.develop] => {
     "changed": true,
     "cmd": [
         "bash",
         "/opt/dims/bin/dimsenv.build",
         "--verbose",
         "2>&1"
     ],
     "delta": "0:02:08.917329",
     "end": "2017-08-01 18:56:32.631252",
     "rc": 0,
     "start": "2017-08-01 18:54:23.713923"
 }

 STDOUT:

 [+] Starting /opt/dims/bin/dimsenv.build
 [+] Unpacking /opt/dims/src/Python-2.7.13.tgz archive
 [+] Configuring/compiling Python-2.7.13
 checking build system type... x86_64-pc-linux-gnu
 checking host system type... x86_64-pc-linux-gnu
 . . .
 [ 10129 lines deleted! ]
 . . .
 virtualenvwrapper.user_scripts creating /opt/dims/envs/dimsenv/bin/get_env_details
   Retrying (Retry(total=4, connect=None, read=None, redirect=None)) after connection␣
↪broken by 'ProxyError('Cannot connect to proxy.', error('Tunnel connection failed:␣
↪503 Service Unavailable',))': /source/python_dimscli-0.26.0-py2.py3-none-any.whl
   Retrying (Retry(total=3, connect=None, read=None, redirect=None)) after connection␣
↪broken by 'ProxyError('Cannot connect to proxy.', error('Tunnel connection failed:␣
↪503 Service Unavailable',))': /source/python_dimscli-0.26.0-py2.py3-none-any.whl
   Retrying (Retry(total=2, connect=None, read=None, redirect=None)) after connection␣
↪broken by 'ProxyError('Cannot connect to proxy.', error('Tunnel connection failed:␣
↪503 Service Unavailable',))': /source/python_dimscli-0.26.0-py2.py3-none-any.whl
   Retrying (Retry(total=1, connect=None, read=None, redirect=None)) after connection␣
↪broken by 'ProxyError('Cannot connect to proxy.', error('Tunnel connection failed:␣
↪503 Service Unavailable',))': /source/python_dimscli-0.26.0-py2.py3-none-any.whl
   Retrying (Retry(total=0, connect=None, read=None, redirect=None)) after connection␣
↪broken by 'ProxyError('Cannot connect to proxy.', error('Tunnel connection failed:␣
↪503 Service Unavailable',))': /source/python_dimscli-0.26.0-py2.py3-none-any.whl
```

(continues on next page)

(continued from previous page)

```
Exception:
Traceback (most recent call last):
  File "/opt/dims/envs/dimsenv/lib/python2.7/site-packages/pip/basecommand.py", line␣
↪215, in main
    status = self.run(options, args)
  File "/opt/dims/envs/dimsenv/lib/python2.7/site-packages/pip/commands/install.py",␣
↪line 335, in run
    wb.build(autobuilding=True)
  File "/opt/dims/envs/dimsenv/lib/python2.7/site-packages/pip/wheel.py", line 749,␣
↪in build
    self.requirement_set.prepare_files(self.finder)
  File "/opt/dims/envs/dimsenv/lib/python2.7/site-packages/pip/req/req_set.py", line␣
↪380, in prepare_files
    ignore_dependencies=self.ignore_dependencies))
  File "/opt/dims/envs/dimsenv/lib/python2.7/site-packages/pip/req/req_set.py", line␣
↪620, in _prepare_file
    session=self.session, hashes=hashes)
  File "/opt/dims/envs/dimsenv/lib/python2.7/site-packages/pip/download.py", line␣
↪821, in unpack_url
    hashes=hashes
  File "/opt/dims/envs/dimsenv/lib/python2.7/site-packages/pip/download.py", line␣
↪659, in unpack_http_url
    hashes)
  File "/opt/dims/envs/dimsenv/lib/python2.7/site-packages/pip/download.py", line␣
↪853, in _download_http_url
    stream=True,
  File "/opt/dims/envs/dimsenv/lib/python2.7/site-packages/pip/_vendor/requests/
↪sessions.py", line 488, in get
    return self.request('GET', url, **kwargs)
  File "/opt/dims/envs/dimsenv/lib/python2.7/site-packages/pip/download.py", line␣
↪386, in request
    return super(PipSession, self).request(method, url, *args, **kwargs)
  File "/opt/dims/envs/dimsenv/lib/python2.7/site-packages/pip/_vendor/requests/
↪sessions.py", line 475, in request
    resp = self.send(prep, **send_kwargs)
  File "/opt/dims/envs/dimsenv/lib/python2.7/site-packages/pip/_vendor/requests/
↪sessions.py", line 596, in send
    r = adapter.send(request, **kwargs)
  File "/opt/dims/envs/dimsenv/lib/python2.7/site-packages/pip/_vendor/cachecontrol/
↪adapter.py", line 47, in send
    resp = super(CacheControlAdapter, self).send(request, **kw)
  File "/opt/dims/envs/dimsenv/lib/python2.7/site-packages/pip/_vendor/requests/
↪adapters.py", line 485, in send
    raise ProxyError(e, request=request)
ProxyError: HTTPSConnectionPool(host='source.devops.develop', port=443): Max retries␣
↪exceeded with url: /source/python_dimscli-0.26.0-py2.py3-none-any.whl (Caused by␣
↪ProxyError('Cannot connect to proxy.', error('Tunnel connection failed: 503 Service␣
↪Unavailable',)))
zip_safe flag not set; analyzing archive contents...
rpc.rpc_common: module MAY be using inspect.getouterframes
/opt/dims/envs/dimsenv/lib/python2.7/site-packages/setuptools/dist.py:341:␣
↪UserWarning: Normalizing '1.0.0-dev' to '1.0.0.dev0'
  normalized_version,
warning: install_lib: 'build/lib' does not exist -- no Python modules to install

zip_safe flag not set; analyzing archive contents...
```

(continues on next page)

(continued from previous page)

```
TASK [python-virtualenv : Run unit test for Python virtualenv] ****************
. . .
PLAY RECAP *********************************************************************
dimsdemo1.devops.develop   : ok=50   changed=10   unreachable=0    failed=1
```

The highlighted lines show the problem, which is a proxy failure. This is typically due to the Docker container used as a local `squid-deb-proxy` to optimize Vagrant installations being hung and non-responsive. (To resolve this, see restartProxy in the Appendix.)

A final example here of a `bats` unit test being used to avoid hidden problems resulting from subtle errors in Ansible playbooks is the unit test `ansible-yaml`. This test is intended to perform validations checks of YAML syntax and Ansible module requirements.

Perhaps the most important test (and the only shown here) has to do with avoiding the way `mode` attributes to modules like `copy` and `template` are used. Ansible is very powerful, but it has some pedantic quirks related to the use of YAML to create Python data structures that are used to run Unix command line programs. This is perhaps nowhere more problematic than file permissions (which can horribly break things on a Unix system). The problem has to do with the way Unix modes (i.e., file permissions) were historically defined using bit-maps expressed in numeric form using **octal** (i.e., base-8) form, not decimal (i.e., base-10) form that is more universally used for expressing numbers. An octal value is expressed using the digits `0` through `7`. Many programming languages assume that if a given string representing a numeric value starts with an alpha-numeric character in the range `[1..9]`, that string represents a decimal value, while a string starting with a `0` instead represents an octal value. Since YAML and JSON are text representations of data structures that are interpretted to create binary data structures used internally to Python

Note: Two other numeric bases used commonly in programming are binary (base-2) and hexadecimal (base-16). Binary is used so rarely that we can ignore it here. Because hexadecimal goes beyond `9`, using the letters in the range `[A..F]`, it doesn't have the same conflict in being recognized as decimal vs. octal, so its values typically start with `0x` followed by the number (e.g., `0x123ABCD`).

You can find issues describing this decimal vs. octal/string vs. number problem and related discussion of ways to deal with it in many places:

- 4-digit octal mode works incorrectly unless quoted #9196
- Why am I getting this error from Ansible when trying to change the permissions on a file?
- Setting the setuid / setgid Bit with Ansible

The `ansible-dims-playbooks` repository uses the convention of expressing modes using the `0o` format to explicitly ensure proper octal modes, e.g.:

```
- name: Ensure /etc/rsyslog.d present
  file:
    path: /etc/rsyslog.d
    state: directory
    mode: 0o644
  become: yes
  when: ansible_os_family != "Container Linux by CoreOS"
  tags: [ base, config, rsyslogd ]
```

Running the `ansible-yaml` unit test will detect when modes deviate from this standard policy, as seen here:

```
$ test.runner unit/ansible-yaml
[+] Running test unit/ansible-yaml
 ✓ [U][EV] Modes in tasks under $PBR use valid '0o' notation
   [U][EV] Modes in tasks under $DIMS_PRIVATE use valid '0o' notation
```

(continues on next page)

(continued from previous page)

```
   (in test file unit/ansible-yaml.bats, line 30)
     `[ -z "$DIMS_PRIVATE" ] && skip 'DIMS_PRIVATE is not defined'' failed
   /home/dittrich/dims/git/private-develop/inventory/all.yml:    mode: 755

2 tests, 1 failure
```

Note: Exceptions to this check will likely need to be made, since the test is for strings of the form `mode: 0o` or `mode=0o` (followed by a few more numbers). These exceptions are put into a whitelist file that the test uses to ignore them. To update the whitelist, add any false positive failure strings to the variable `yaml_mode_whitelist` in `group_vars/all/dims.yml`.

CHAPTER 7

Debugging with Ansible and Vagrant

This chapter covers some tactics and procedures used for testing and debugging Ansible inventories, playbooks, roles, etc. using Vagrants with `bats` tests as the test and validation mechanisms.

More general debugging strategies and techniques are covered in Section dimsdevguide:debugging of the dimsdevguide:dimsdevguide.

7.1 Debugging Ansible

Ansible has two primary methods with which it is invoked – `ansible-playbook` to run playbooks, and `ansible` (a.k.a., "ad-hoc mode") to run individual modules one at a time.

- Debugging using the `debug` module in "ad-hoc" mode can be used to explore the value of variables after processing of the inventory (i.e., group definitions, group variables, host variables, etc.) This does not require any remote connections, or even an internet connection at all, since the `debug` module is processed locally on the Ansible control host. (The flip side of this is that no Ansible "facts" are available, *because* of the fact that no remote connections are made.)
- Debugging playbook execution with `ansible-playbook` involves controlling the level of verbosity in output of program execution and/or exposing the runtime state of variables (possibly obtaining that state remotely from running systems) using the `debug` module. There is also "single-stepping" of playbooks that can be used in conjunction with these mechanisms.

7.1.1 Examining Variables

To see the value of the variable `inventory_hostname` for a group of hosts named `manager`, use the `debug` module, the specific inventory to look at, and pass the group name:

```
$ ansible -m debug -a "var=inventory_hostname" manager
node03.devops.local | SUCCESS => {
    "inventory_hostname": "node03.devops.local"
}
```

(continues on next page)

(continued from previous page)

```
node01.devops.local | SUCCESS => {
    "inventory_hostname": "node01.devops.local"
}
node02.devops.local | SUCCESS => {
    "inventory_hostname": "node02.devops.local"
}
```

Ansible variables are sometimes not as straightforward as that. Often variables are composed from other variables using Jinja templating expressions in strings which are recursively processed at run time during template rendering. This means that you must either be really good at resolving the nested variable references in your head, or get used to using Ansible's `debug` module with `msg` to do the templating for you. What is more, Ansible variables are all effectively in a deeply-nested Python dictionary structure that takes some getting used to. Using data structures properly helps iterate over lists or dictionary keys using clean algorithms involving Loops.

To see how this works, take a look at the following example of the bundle of Trident packages that are part of a Trident deployment. We want to validate each package using a common cryptographic hash, so a simple dictionary keyed on `url` and `sha256sum` will work.

```
# Trident components are all loaded at once as a bundle
trident_dist_bundle:
  - { 'url': '{{ trident_server_disturl }}', 'sha256sum': '{{ trident_server_
↪sha256sum }}' }
  - { 'url': '{{ trident_cli_disturl }}', 'sha256sum': '{{ trident_cli_sha256sum }}' }
  - { 'url': '{{ trident_all_disturl }}', 'sha256sum': '{{ trident_all_sha256sum }}' }
  - { 'url': '{{ pitchfork_disturl }}', 'sha256sum': '{{ pitchfork_sha256sum }}' }
  - { 'url': '{{ trident_wikiexport_disturl }}', 'sha256sum': '{{ trident_wikiexport_
↪sha256sum }}' }

trident_cli_version: '{{ trident_version }}'
trident_cli_archive: 'trident-cli_{{ trident_cli_version }}_amd64.deb'
trident_cli_disturl: '{{ trident_download_dir }}/{{ trident_cli_archive }}'
trident_cli_sha256sum:
↪'15f11c986493a67e85aa9cffe6719a15a8c6a65b739a2b0adf62ce61e53f4203'
trident_cli_opts: ''

trident_server_version: '{{ trident_version }}'
trident_server_archive: 'trident-server_{{ trident_server_version }}_amd64.deb'
trident_server_disturl: '{{ trident_download_dir }}/{{ trident_server_archive }}'
trident_server_sha256sum:
↪'a8af27833ada651c9d15dc29d04451250a335ae89a0d2b66bf97a787dced9956'
trident_server_opts: '--syslog'

trident_all_version: '{{ trident_version }}'
trident_all_archive: 'trident_{{ trident_all_version }}_all.deb'
trident_all_disturl: '{{ trident_download_dir }}/{{ trident_all_archive }}'
trident_all_sha256sum:
↪'67f57337861098c4e9c9407592c46b04bbc2d64d85f69e8c0b9c18e8d5352ea6' #trident_1.4.5_
↪all.deb

trident_wikiexport_version: '{{ trident_version }}'
trident_wikiexport_archive: 'trident-wikiexport_{{ trident_wikiexport_version }}_
↪amd64.deb'
trident_wikiexport_disturl: '{{ trident_download_dir }}/{{ trident_wikiexport_archive␣
↪}}'
trident_wikiexport_sha256sum:
↪'4d2f9d62989594dc5e839546da596094c16c34d129b86e4e323556f1ca1d8805'
```

(continues on next page)

(continued from previous page)

```

# Pitchfork tracks its own version
pitchfork_version: '1.9.4'
pitchfork_archive: 'pitchfork-data_{{ pitchfork_version }}_all.deb'
pitchfork_disturl: '{{ trident_download_dir }}/{{ pitchfork_archive }}'
pitchfork_sha256sum: '5b06ae4a20a16a7a5e59981255ba83818f67224b68f6aaec014acf51ca9d1a44
↪'

# Trident perl tracks its own version
# TODO(dittrich): trident-perl is private artifact - using our cached copy
trident_perl_version: '0.1.0'
trident_perl_archive: 'trident-perl_{{ trident_perl_version }}_amd64.deb'
trident_perl_disturl: '{{ artifacts_url }}/{{ trident_perl_archive }}'
trident_perl_sha256sum:
↪'2f120dc75f75f8b2c8e5cdf55a29984e24ee749a75687a10068ed8f353098ffb'
```

To see what the `trident_dist_bundle` looks like to better visualize how to loop on it and process the values, we can use the following command:

```
$ ansible -i inventory/ -m debug -a "msg={{ trident_dist_bundle }}" yellow.devops.
↪local
yellow.devops.local | SUCCESS => {
    "changed": false,
    "msg": [
        {
            "sha256sum":
↪"a8af27833ada651c9d15dc29d04451250a335ae89a0d2b66bf97a787dced9956",
            "url": "https://github.com/tridentli/trident/releases/download/v1.4.5/
↪trident-server_1.4.5_amd64.deb"
        },
        {
            "sha256sum":
↪"15f11c986493a67e85aa9cffe6719a15a8c6a65b739a2b0adf62ce61e53f4203",
            "url": "https://github.com/tridentli/trident/releases/download/v1.4.5/
↪trident-cli_1.4.5_amd64.deb"
        },
        {
            "sha256sum":
↪"67f57337861098c4e9c9407592c46b04bbc2d64d85f69e8c0b9c18e8d5352ea6",
            "url": "https://github.com/tridentli/trident/releases/download/v1.4.5/
↪trident_1.4.5_all.deb"
        },
        {
            "sha256sum":
↪"5b06ae4a20a16a7a5e59981255ba83818f67224b68f6aaec014acf51ca9d1a44",
            "url": "https://github.com/tridentli/trident/releases/download/v1.4.5/
↪pitchfork-data_1.9.4_all.deb"
        },
        {
            "sha256sum":
↪"4d2f9d62989594dc5e839546da596094c16c34d129b86e4e323556f1ca1d8805",
            "url": "https://github.com/tridentli/trident/releases/download/v1.4.5/
↪trident-wikiexport_1.4.5_amd64.deb"
        }
    ]
}
```

7.1.2 Debugging Filter Logic

Ansible supports Filters in template expressions. These use not only the default builtin Jinja filters, but also added Ansible filters and custom filters that user can easily add.

In general, these filters take some data structure as input and perform operations on it to produce some desired output, such as replacing strings based on regular expressions or turning keys in dictionary into a list.

Jinja filters can be chained when maniplating complex data structures. In some cases they must be chained to achieve the desired result.

For example, take the following example data structure, which is an array named trident_site_trust_groups that holds dictionaries containing a name, initial_users, and additional_lists:

```
trident:
  vars:
    trident_site_trust_groups:
      - name: 'main'
        initial_users:
          - ident: 'dims'
            descr: 'DIMS Mail (no-reply)'
            email: 'noreply@{{ trident_site_email_domain }}'
          - ident: 'dittrich'
            descr: 'Dave Dittrich'
            email: 'dittrich@{{ trident_site_email_domain }}'
        additional_lists:
          - ident: 'demo'
            descr: 'LOCAL Trident Demonstration'
          - ident: 'warroom'
            descr: 'LOCAL Trust Group War Room'
          - ident: 'exercise'
            descr: 'LOCAL Trust Group Exercise Comms'
          - ident: 'events'
            descr: 'LOCAL Trust Group Social Events'
```

Start by just examining the variable using Ansible's debug module and var to select the top level variable in the vars structure.

```
$ ansible -m debug -a "var=vars.trident_site_trust_groups" yellow.devops.local
yellow.devops.local | SUCCESS => {
    "changed": false,
    "vars.trident_site_trust_groups": [
        {
            "additional_lists": [
                {
                    "descr": "LOCAL Trident Demonstration",
                    "ident": "demo"
                },
                {
                    "descr": "LOCAL Trust Group War Room",
                    "ident": "warroom"
                },
                {
                    "descr": "LOCAL Trust Group Exercise Comms",
                    "ident": "exercise"
                },
                {
                    "descr": "LOCAL Trust Group Social Events",
```

(continues on next page)

(continued from previous page)

```
                    "ident": "events"
                }
            ],
            "initial_users": [
                {
                    "descr": "DIMS Mail (no-reply)",
                    "email": "noreply@{{ trident_site_email_domain }}",
                    "ident": "dims"
                },
                {
                    "descr": "Dave Dittrich",
                    "email": "dittrich@{{ trident_site_email_domain }}",
                    "ident": "dittrich"
                }
            ],
            "name": "main",
        }
    ]
}
```

Next, we can isolate just the `additional_lists` sub-dictionary:

```
$ ansible -m debug -a "var=vars.trident_site_trust_groups[0].additional_lists" yellow.
↪devops.local
yellow.devops.local | SUCCESS => {
    "changed": false,
    "vars.trident_site_trust_groups[0].additional_lists": [
        {
            "descr": "LOCAL Trident Demonstration",
            "ident": "demo"
        },
        {
            "descr": "LOCAL Trust Group War Room",
            "ident": "warroom"
        },
        {
            "descr": "LOCAL Trust Group Exercise Comms",
            "ident": "exercise"
        },
        {
            "descr": "LOCAL Trust Group Social Events",
            "ident": "events"
        }
    ]
}
```

The `map` filter is then used to extract just the key `ident` from each dictionary, followed by `list` to turn the extracted sub-dictionary into an array, followed by `sort` to put the list in alphabetic order for good measure.

```
$ ansible -m debug -a msg="{{ trident_site_trust_groups[0].additional_
↪lists|map(attribute='ident')|list|sort }}" yellow.devops.local
yellow.devops.local | SUCCESS => {
    "changed": false,
    "msg": [
        "demo",
        "events",
```

(continues on next page)

(continued from previous page)

```
        "exercise",
        "warroom"
    ]
}
```

In an Ansible playbook, it might look like this:

```
 - name: Create list of defined mailing lists
   set_fact: _additional_lists={{ trident_site_trust_groups[0].additional_
↪lists|map(attribute='ident')|list|sort }}"

 - debug: var=_additional_lists
```

This will give the following results:

```
 TASK [Create list of defined mailing lists] ************************************
 Monday 13 February 2017  09:20:38 -0800 (0:00:01.037)       0:00:01.093 *******
 ok: [yellow.devops.local]

 TASK [debug] *******************************************************************
 Monday 13 February 2017  09:20:38 -0800 (0:00:00.043)       0:00:01.136 *******
 ok: [yellow.devops.local] => {
     "_additional_lists": [
         "demo",
         "events",
         "exercise",
         "warroom"
     ]
 }

 PLAY RECAP *********************************************************************
 yellow.devops.local        : ok=3    changed=0    unreachable=0    failed=0
```

Our final example illustrates forced type conversion with a filter to drive the proper logic of a boolean filter known as the *ternary* operator. This is a useful, but somewhat terse, operator that takes a boolean expression as the input and produces one of two outputs based on the value of the boolean expression. This prevents having to do two separate tasks, one with the `true` conditional and a second with the `false` conditional. In the example we are about to see, the goal is to produce a ternary filter expression that results in creating a variable that will be added to a command line invoking `certbot-auto` that adds the `--staging` option when an Ansible variable holds a boolean `true` value.

A conditional operation in Jinja is an expression in parentheses (`()`). Our first attempt looks like this:

```
$ ansible -m debug -e debug=true -a 'msg={{ (debug)|ternary("yes", "no") }}' yellow.
 ↪devops.local
yellow.devops.local|SUCCESS => {
    "changed": false,
    "msg": "yes"
}
```

That looks perfect! Go!

No, that is not robust. It is unwise to try something, get the result you expect, and run with it. Let's try setting `debug` to `false` and see what happens.

```
$ ansible -m debug -e debug=false -a 'msg={{ (debug)|ternary("yes", "no") }}' yellow.
 ↪devops.local
yellow.devops.local|SUCCESS => {
```

(continues on next page)

(continued from previous page)

```
    "changed": false,
    "msg": "yes"
}
```

False is true? Fake news! What is happening here? Do we need to actually do an equivalence test using == to get the right result? Let's try it.

```
$ ansible -m debug -e debug=false -a 'msg={{ (debug == True)|ternary("yes", "no") }}'␣
↪yellow.devops.local
yellow.devops.local|SUCCESS => {
    "changed": false,
    "msg": "no"
}
$ ansible -m debug -e debug=true -a 'msg={{ (debug == True)|ternary("yes", "no") }}'␣
↪yellow.devops.local
yellow.devops.local|SUCCESS => {
    "changed": false,
    "msg": "no"
}
```

OK. Now we get the exact same result again, but this time it is the exact *opposite* always-the-same result. What?!?! Ansible allows us to use `yes`, `true`, or even `on` to set a boolean variable. The Gotcha here is that the variable is being set on the command line, which sets the variable to be a *string* rather than a *boolean*, and a non-null string (*any string*) resolves to `true`.

Wait! Maybe the problem is we defined `debug=true` instead of `debug=True`? That's got to be it, yes?

```
$ ansible -m debug -e "debug=True" -a 'msg={{ (debug == True)|ternary("yes", "no") }}
↪' yellow.devops.local
yellow.devops.local|SUCCESS => {
    "changed": false,
    "msg": "no"
}
```

As the `msg` says, `no`.

Let's go back to the simple `(debug)` test and systematically try a bunch of alternatives and see what actually happens in real-world experimentation.

```
$ ansible -m debug -e "debug=True" -a 'msg={{ (debug)|ternary("yes", "no") }}' yellow.
↪devops.local
yellow.devops.local|SUCCESS => {
    "changed": false,
    "msg": "yes"
}
$ ansible -m debug -e "debug=False" -a 'msg={{ (debug)|ternary("yes", "no") }}'␣
↪yellow.devops.local
yellow.devops.local|SUCCESS => {
    "changed": false,
    "msg": "yes"
}
$ ansible -m debug -e "debug=yes" -a 'msg={{ (debug)|ternary("yes", "no") }}' yellow.
↪devops.local
yellow.devops.local|SUCCESS => {
    "changed": false,
    "msg": "yes"
}
```

(continues on next page)

(continued from previous page)

```
$ ansible -m debug -e "debug=no" -a 'msg={{ (debug)|ternary("yes", "no") }}' yellow.
↪devops.local
yellow.devops.local|SUCCESS => {
    "changed": false,
    "msg": "yes"
}
```

Spoiler Alert

It is not obvious at all, but the behavior hints at the problem which is a typing conflict between boolean and string types, combined with the way strings are interpreted in a conditional expression. Pretty much every interpreted programming language, and even some compiled languages without mandatory strong typing, have their own variation on this problem. It takes programming experience with perhaps a dozen or more programming languages to internalize this problem enough to reflexively avoid it it seems (and even then it can still bite you!) The answer is to be explicit about boolean typing and/or casting.

Jinja has a filter called `bool` that converts a string to a boolean the way we expect from the Ansible documentation. Adding `|bool` results in the behavior we expect:

```
$ ansible -m debug -e "debug=no" -a 'msg={{ (debug|bool)|ternary("yes", "no") }}'␣
↪yellow.devops.local
yellow.devops.local|SUCCESS => {
    "changed": false,
    "msg": "no"
}
$ ansible -m debug -e "debug=yes" -a 'msg={{ (debug|bool)|ternary("yes" , "no") }}'␣
↪yellow.devops.local
yellow.devops.local|SUCCESS => {
    "changed": false,
    "msg": "yes"
}
$ ansible -m debug -e "debug=False" -a 'msg={{ (debug|bool)|ternary("yes", "no") }}'␣
↪yellow.devops.local
yellow.devops.local|SUCCESS => {
    "changed": false,
    "msg": "no"
}
$ ansible -m debug -e "debug=True" -a 'msg={{ (debug|bool)|ternary("yes" , "no") }}'␣
↪yellow.devops.local
yellow.devops.local|SUCCESS => {
    "changed": false,
    "msg": "yes"
}
$ ansible -m debug -e "debug=off" -a 'msg={{ (debug|bool)|ternary("yes", "no") }}'␣
↪yellow.devops.local
yellow.devops.local|SUCCESS => {
    "changed": false,
    "msg": "no"
}
$ ansible -m debug -e "debug=on" -a 'msg={{ (debug|bool)|ternary("yes" , "no") }}'␣
↪yellow.devops.local
yellow.devops.local|SUCCESS => {
    "changed": false,
    "msg": "yes"
}
```

OK, *that's better*!! Now that we have the syntax down to get the logic that we expect, we can set the `certbot_staging` variable they way we want:

```
$ ansible -m debug -e "certbot_staging=no" -a 'msg={{ (certbot_staging|bool)|ternary(
↪"--staging", "") }}' yellow.devops.local
yellow.devops.local|SUCCESS => {
    "changed": false,
    "msg": ""
}
$ ansible -m debug -e "certbot_staging=yes" -a 'msg={{ (certbot_staging|bool)|ternary(
↪"--staging", "") }}' yellow.devops.local
yellow.devops.local|SUCCESS => {
    "changed": false,
    "msg": "--staging"
}
```

Attention: Hopefully this shows the importance of using Ansible's `debug` module to develop tasks in playbooks such that they don't result in hidden bugs that cause silent failures deep within hundreds of tasks that blast by on the screen when you run a complex Ansible playbook. Doing this every time a complex Jinja expression, or a deeply nested complex data structure, will take a little extra time. But it is *almost guaranteed* to be *much less time* (and less stress, less friction) than debugging the playbook later on when something isn't working right and it isn't clear why. Robust coding practice is good coding practice!

7.1.3 Developing Custom Jinja Filters

Here is a minimal sub-set of the DIMS filters module, `dims_filters.py`, that implements a filter that converts an array into a string usable with Consul for establishing an `initial-cluster` command line argument.

```
# vim: set ts=4 sw=4 tw=0 et :

from netaddr import *
import socket
from ansible import errors

def _initial_cluster(_list, port=2380):
    '''
    Return a comma (no spaces!) separated list of Consul initial cluster
    members from fully qualified domain names (e.g., Ansible group member
    names). The "no spaces" is because this is used as a single command line
    argument.

    a = ['node01.devops.local','node02.devops.local','node03.devops.local']
    _initial_cluster(a)
    'node01=http://node01.devops.local:2380,node02=http://node02.devops.local:2380,
↪node03=http://node03.devops.local:2380'

    '''

    if type(_list) == type([]):
        try:
            return ','.join(
                ['{0}=http://{1}:{2}'.format(
                    i.decode('utf-8').split('.')[0],
                    i.decode('utf-8'),
```

(continues on next page)

(continued from previous page)

```
                    port) for i in _list]
            )
        except Exception as e:
            #raise errors.AnsibleFilterError(
            #    'initial_cluster() filed to convert: {0}'.format(str(e))
            #)
            return ''
    else:
        raise errors.AnsibleFilterError('Unrecognized input arguments to initial_
↪cluster()')

class FilterModule(object):
    '''DIMS Ansible filters.'''

    def filters(self):
        return {
            # Docker/Consul/Swarm filters
            'initial_cluster': _initial_cluster,
        }
```

Here is how it works with the `debug` module:

```
$ ansible -m debug -a msg="{{ groups.consul|initial_cluster() }}" node01.devops.local
node01.devops.local | SUCCESS => {
    "changed": false,
    "msg": "node03=http://node03.devops.local:2380,node02=http://node02.devops.
↪local:2380,node01=http://node01.devops.local:2380"
}
```

CHAPTER 8

Regular System Maintenance

This chapter covers regular system maintenance tasks, such as updating the `ansible-dims-playbooks` repo and related private customization repository, upgrading operating system packages, and generally keeping system components clean and up to date.

8.1 Updating Operating System Packages

Updating system packages, especially security patches, is an important part of ensuring the integrity, availability, and confidentiality of information and information systems. The availability aspect is sometimes a concern when applying updates, so using the multi-deployment model adopted by the DIMS Project to allow easier testing of system components after patching on a test deployment before applying updates to "production" deployment systems helps allay concerns.

There are two `bats` system tests that are designed to make the normal system updating process easier to automate and apply across the entire deployment: the `system/updates` and `system/reboot` tests. Both of these tests can be run at once using the following command line:

```
$ test.runner --match "updates|reboot"
[+] Running test system/updates
  [S][EV] All APT packages are up to date (Ubuntu)
   (from function `assert' in file system/helpers.bash, line 18,
    in test file system/updates.bats, line 11)
     `assert "0 packages can be updated. 0 updates are security updates." bash -c "/
↪usr/lib/update-notifier/apt-check --human-readable"' failed
   linux-headers-4.4.0-92-generic
   google-chrome-stable
   xul-ext-ubufox
   firefox
   linux-image-4.4.0-92-generic
   linux-generic-lts-xenial
   libgd3
   linux-headers-4.4.0-92
   linux-headers-generic-lts-xenial
```

(continues on next page)

(continued from previous page)

```
   linux-image-extra-4.4.0-92-generic
   linux-image-generic-lts-xenial
   expected: "0 packages can be updated. 0 updates are security updates."
   actual:   "11 packages can be updated.10 updates are security updates."

1 test, 1 failure

[+] Running test system/reboot
 ✓ [S][EV] System does not require a reboot (Ubuntu)

1 test, 0 failures
```

In this case, the tests show that the system has updates most of them security updates, ready to apply. The `updates` test failed, but the `reboot` test passed.

Now apply the `updates` tag to update and install upgrades.

```
$ run.playbook --tags updates -e packages_upgrade=yes

PLAY [Configure host "dimsdemo1.devops.develop"] ******************************
. . .
TASK [base : Check to see if update-manager is running on Ubuntu] *************
Wednesday 16 August 2017  13:06:29 -0700 (0:00:01.049)       0:00:05.392 ******
changed: [dimsdemo1.devops.develop]

TASK [base : Kill update_manager to avoid dpkg lock contention] ***************
Wednesday 16 August 2017  13:06:30 -0700 (0:00:01.239)       0:00:06.631 ******
skipping: [dimsdemo1.devops.develop]

TASK [base : Check to see if gpk-update-viewer is running on Ubuntu] **********
Wednesday 16 August 2017  13:06:31 -0700 (0:00:01.049)       0:00:07.681 ******
skipping: [dimsdemo1.devops.develop]

TASK [base : Kill gpk-update-viewer to avoid dpkg lock contention] ************
Wednesday 16 August 2017  13:06:32 -0700 (0:00:01.048)       0:00:08.729 ******
skipping: [dimsdemo1.devops.develop]

TASK [base : Make sure blacklisted packages are absent (Debian)] **************
Wednesday 16 August 2017  13:06:33 -0700 (0:00:01.084)       0:00:09.814 ******
ok: [dimsdemo1.devops.develop] => (item=[u'modemmanager', u'resolvconf',
u'sendmail', u'whoopsie', u'libwhoopsie0'])

TASK [base : Only "update_cache=yes" if >3600s since last update (Debian)] ****
Wednesday 16 August 2017  13:06:35 -0700 (0:00:02.015)       0:00:11.829 ******
ok: [dimsdemo1.devops.develop]

TASK [base : Make sure required APT packages are present (Debian)] ************
Wednesday 16 August 2017  13:06:37 -0700 (0:00:01.610)       0:00:13.440 ******
ok: [dimsdemo1.devops.develop] => (item=[u'apt-transport-https', u'bash-completion',
u'ca-certificates', u'cpanminus', u'curl', u'dconf-tools', u'git-core',
u'default-jdk', u'gitk', u'gnupg2', u'htop', u'hunspell', u'iptables-persistent',
u'ifstat', u'make', u'myrepos', u'netcat', u'nfs-common', u'chrony', u'ntpdate',
u'openssh-server', u'patch', u'perl', u'postfix', u'python', u'python-apt',
u'remake', u'rsync', u'rsyslog', u'sshfs', u'sshpass', u'strace', u'tree', u'vim',
u'xsltproc', u'chrony', u'nfs-kernel-server', u'smartmontools', u'unzip'])

TASK [base : Make upgraded packages present if we are explicitly upgrading] ***
```

(continues on next page)

(continued from previous page)

```
Wednesday 16 August 2017  13:06:38 -0700 (0:00:01.750)       0:00:15.190 ******
changed: [dimsdemo1.devops.develop]

TASK [base : Check proxy availability] *****************************************
Wednesday 16 August 2017  13:09:12 -0700 (0:02:33.389)       0:02:48.580 ******
. . .
PLAY RECAP *********************************************************************
dimsdemo1.devops.develop   : ok=72   changed=4    unreachable=0    failed=0

Wednesday 16 August 2017  13:10:28 -0700 (0:00:01.069)       0:04:04.737 ******
================================================================================
base : Make upgraded packages present if we are explicitly upgrading -- 153.39s
. . .
```

Note: The flag `-e packages_upgrade=yes` sets the variable `packages_upgrade` that must evaluate to true in order for packages to be updated in the role. This is to ensure that package updates are done in a controlled manner. Set this variable to something that Ansible evaluates as true on the command line, or somewhere in the host vars section of the inventory.

Now re-run the two tests.

```
$ test.runner --match "updates|reboot"
[+] Running test system/updates
 ✓ [S][EV] All APT packages are up to date (Ubuntu)

1 test, 0 failures

[+] Running test system/reboot
  [S][EV] System does not require a reboot (Ubuntu)
   (in test file system/reboot.bats, line 8)
     `@test "[S][EV] System does not require a reboot (Ubuntu)" {' failed
   linux-image-4.4.0-92-generic
   linux-base
   linux-base

1 test, 1 failure
```

This time the updates test passes, but notice that some of the updates require a reboot, so that test fails. This means that a reboot needs to be planned and executed carefully, to ensure minimal disruption to anything dependent on this system (e.g., running virtual machines on a development system).

Attention: A developer workstation or production VM host running virtual machines needs to have the virtual machines shut down or suspended prior to a reboot of the VM host in order to ensure the VMs or the VM host does not lose network interfaces that are using DHCP. The VM host may lose a `vboxnet` interface, a VM may lose an `eth` interface, or both.

- Vagrants are handled as part of the shutdown process when you use the `dims.shutdown` wrapper script. After reboot, use `dims.shutdown --resume` (optionally with `--group` to select specific Vagrants by name or group) to resume them.
- Virtualbox VMs that were created by hand are not yet supported by `dims.shutdown`. Use the `virtualbox` management GUI to cleanly shut down any running VMs (and again after reboot, to bring them back up.) If this is a remote VM host, use `remmina` and the VNC wrapper script described in Section

Validating VNC over SSH Tunnelling to run the `virtualbox` management GUI remotely.

Using Ansible ad-hoc mode, the checks can be performed on multiple hosts at once:

```
$ ansible -m shell -a 'test.runner --match reboot' trident
yellow.devops.develop | SUCCESS | rc=0 >>
# [+] Running test system/reboot
1..1
ok 1 [S][EV] System does not require a reboot (Debian)
#

purple.devops.develop | SUCCESS | rc=0 >>
# [+] Running test system/reboot
1..1
ok 1 [S][EV] System does not require a reboot (Debian)
#
```

As a convenience for the system administrator, a `cron` job is managed by the `base` role that runs a script named `dims.updatecheck` on a daily basis. The variables that control the `cron` job are defined in the `group_vars/all/dims.yml` file:

```
cronjobs:
  - name: 'dims.updatecheck'
    weekday: '*'
    hour: '6'
    minute: '0'
    user: 'ansible'
    job: '{{ dims_bin }}/dims.updatecheck'
```

The `base` role creates the following file:

```
$ cat /etc/cron.d/dims
#Ansible: dims.updatecheck
0 6 * * * ansible /opt/dims/bin/dims.updatecheck
```

When updates are available, or a reboot is required, email is sent to the `root` account. Make sure that email to this account is forwarded by setting the `postmaster` variable to a valid email address. An example of the message that will be sent is shown here:

```
To: dittrich@u.washington.edu
Subject: dims.updatecheck results from purple.ops.ectf (2017-09-01T23:06:02.
↪211268+00:00)
Message-Id: <20170901230603.9D3C3582@breathe.prisem.washington.edu>
Date: Fri,  1 Sep 2017 16:06:03 -0700 (PDT)
From: root@breathe.prisem.washington.edu (root)

----------------------------------------------------------------------

Host: purple.ops.ectf
Date: 2017-09-01T23:06:02.211268+00:00

This is a report of available package updates and/or required reboot
status.  The output of the bats tests that were run is included below.

If package updates are necessary, this can be accomplished by running
the Ansible playbook for purple.ops.ectf with the following options:
```

(continues on next page)

(continued from previous page)

```
   --tags updates -e packages_update=true

If a reboot is necessary, ensure that the host (and anyone using it)
is prepared for the reboot:

  o Ensure that all users of external services are aware of any
    potential outage of services provided by this host (or its
    (VMs).

  o Halt or suspend any VMs if this is a VM host (and be prepared
    to ensure they are restart after rebooting is complete.)
    (Use the "dims.shutdown" script to facilitate this. See
    documentation and/or "dims.shutdown --usage".)

  o Notify any active users to ensure no active development work
    is lost.

-----------------------------------------------------------------------
test.runner --tap --match "updates|reboot"

# [+] Running test system/updates
1..1
not ok 1 [S][EV] All APT packages are up to date (Debian)
# (from function `assert' in file system/helpers.bash, line 18,
#  in test file system/updates.bats, line 12)
#   `assert "0 packages can be updated." bash -c "apt list --upgradable 2>/dev/null"'␣
↪failed
#
# WARNING: apt does not have a stable CLI interface yet. Use with caution in scripts.
#
# expected: "0 packages can be updated."
# actual:   "Listing...firefox-esr/oldstable 52.3.0esr-1~deb8u2 amd64 [upgradable fro
m: 52.2.0esr-1~deb8u1]gir1.2-soup-2.4/oldstable 2.48.0-1+deb8u1 amd64 [upgradable fro
m: 2.48.0-1]git/oldstable 1:2.1.4-2.1+deb8u4 amd64 [upgradable from: 1:2.1.4-2.1+deb8
u3]git-core/oldstable 1:2.1.4-2.1+deb8u4 all [upgradable from: 1:2.1.4-2.1+deb8u3]git
-man/oldstable 1:2.1.4-2.1+deb8u4 all [upgradable from: 1:2.1.4-2.1+deb8u3]gitk/oldst
able 1:2.1.4-2.1+deb8u4 all [upgradable from: 1:2.1.4-2.1+deb8u3]iceweasel/oldstable
52.3.0esr-1~deb8u2 all [upgradable from: 52.2.0esr-1~deb8u1]libdbd-pg-perl/jessie-pgd
g 3.6.2-1~pgdg80+1 amd64 [upgradable from: 3.4.2-1]libgd3/oldstable 2.1.0-5+deb8u10 a
md64 [upgradable from: 2.1.0-5+deb8u9]libpq5/jessie-pgdg 9.6.4-1.pgdg80+1 amd64 [upgr
adable from: 9.4.13-0+deb8u1]libsoup-gnome2.4-1/oldstable 2.48.0-1+deb8u1 amd64 [upgr
adable from: 2.48.0-1]libsoup2.4-1/oldstable 2.48.0-1+deb8u1 amd64 [upgradable from:
2.48.0-1]"
#
# [+] Running test system/reboot
1..1
ok 1 [S][EV] System does not require a reboot (Debian)
#

-----------------------------------------------------------------------
```

8.2 Renewing Letsencrypt Certificates

The imported role ansible-role-certbot that is being used for Letsencrypt support creates a `crontab` entry in the `ansible` account to automatically renew the certificate when it is about to expire. You can see the `crontab` entry using Ansible ad-hoc mode:

```
$ ansible -m shell -a 'crontab -l' trident
yellow.devops.develop | SUCCESS | rc=0 >>
#Ansible: Certbot automatic renewal.
20 5 * * * /opt/certbot/certbot-auto renew --quiet --no-self-upgrade

purple.devops.develop | SUCCESS | rc=0 >>
#Ansible: Certbot automatic renewal.
20 5 * * * /opt/certbot/certbot-auto renew --quiet --no-self-upgrade
```

You can always run this command whenever you want, again using Ansible ad-hoc mode:

```
$ ansible -m shell -a '/opt/certbot/certbot-auto renew --no-self-upgrade' trident
purple.devops.develop | SUCCESS | rc=0 >>
Requesting root privileges to run certbot...
  /home/ansible/.local/share/letsencrypt/bin/letsencrypt renew --no-self-upgrade

-------------------------------------------------------------------------------
Processing /etc/letsencrypt/renewal/breathe.prisem.washington.edu.conf
-------------------------------------------------------------------------------

The following certs are not due for renewal yet:
  /etc/letsencrypt/live/breathe.prisem.washington.edu/fullchain.pem (skipped)
No renewals were attempted.Saving debug log to /var/log/letsencrypt/letsencrypt.log
Cert not yet due for renewal

yellow.devops.develop | SUCCESS | rc=0 >>
Requesting root privileges to run certbot...
  /home/ansible/.local/share/letsencrypt/bin/letsencrypt renew --no-self-upgrade

-------------------------------------------------------------------------------
Processing /etc/letsencrypt/renewal/echoes.prisem.washington.edu.conf
-------------------------------------------------------------------------------

The following certs are not due for renewal yet:
  /etc/letsencrypt/live/echoes.prisem.washington.edu/fullchain.pem (skipped)
No renewals were attempted.Saving debug log to /var/log/letsencrypt/letsencrypt.log
Cert not yet due for renewal
```

8.3 Updating Secondary Components

The package update steps above perform what you could call a *first order* update process, that is, updating the packages for the major components of the operating system. Some of these components, however, themselves use plugins or other sub-components that require updating. This is most disruptive for major releases (e.g., going from PyCharm `2016.2` to `2016.3`, as shown in the next section).

A development system will have more of these components requiring secondary updates. Partly because of this reason, these type of components are pinned to a specific version. When updating the `ansible-dims-playbooks`, take note of the changes and check for required secondary updates.

Attention: You will sometimes need to communicate the need for these secondary updates to users of the system (e.g., to developers) because some tools like Vagrant and PyCharm keep plugins in users' accounts, not in system directories. As it is difficult to automate this process in a robust way, each user must take responsibility for updating their own plugins to avoid having their toolset go out-of-sync with other developers and cause random failures that are difficult to track down.

In this section, we cover updating Vagrant and PyCharm.

8.3.1 Updating Vagrant Plugins

Vagrant is used for development using Virtualbox virtual machines. It has a few plugins that were adopted (or at least experimentally used) during DIMS development.

After upgrading Vagrant to a new version, users can update their plugins with the following command:

```
$ vagrant plugin update
Updating installed plugins...
Fetching: vagrant-ignition-0.0.3.gem (100%)
Successfully uninstalled vagrant-ignition-0.0.1
Updated 'vagrant-ignition' to version '0.0.3'!
Updated 'vagrant-scp' to version '0.5.7'!
Updated 'vagrant-share' to version '1.1.9'!
Updated 'vagrant-triggers' to version '0.5.3'!
Updated 'vagrant-vbguest' to version '0.14.2'!
```

8.3.2 Updating PyCharm Community Edition

PyCharm is installed using Ansible. The normal workflow for updating a component like PyCharm is to test the new version to ensure it works properly, then update the variables for PyCharm in the Ansible `inventory` before exporting your old settings and then running the `pycharm` role for your development system.

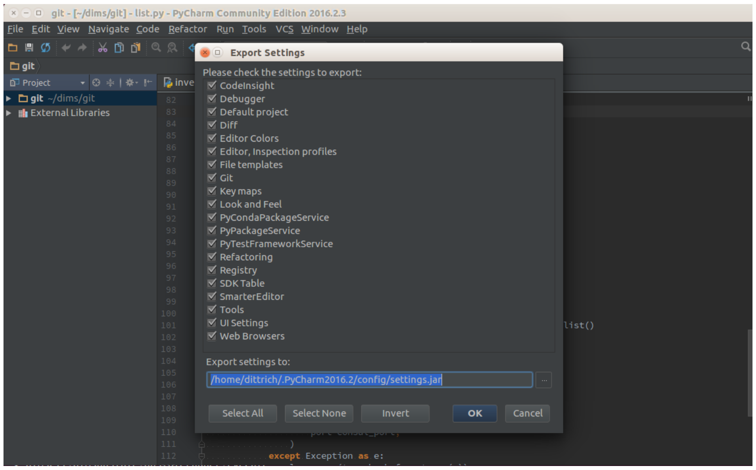
PyCharm keeps all of its state, including settings, breakpoints, indexes, in internal data stores in a directory specific to the version of PyCharm being used. For example, PyCharm 2016.2.3 files are kept in `$HOME/.PyCharm2016.2`. When updating to the release `2016.3.1`, the location changes to `$HOME/.PyCharmCE2016.3`. You need to run PyCharm `2016.2.3` to export your settings, then run the new PyCharm `2016.3.1` version to import them.

To export settings, run PyCharm `2016.2.3` and select **File>Export Settings...**. A dialog will pop up that allows you to select what to export and where to export it. You can use the defaults (pay attention to where the exported setting file is located, since you need to select it in the next step.) Select **Ok** to complete the export. See Figure *Exporting Settings from PyCharm 2016.2.3*.

After PyCharm has been updated, select **File>Import Settings...** and select the `.jar` file that was created in the previous step and then select **Ok**. Again, the defaults can be used for selecting the elements to import. See Figure *Importing Settings to PyCharm 2016.3.1*.

Once you have completed this process and are successfully using version `2016.3.1`, you can delete the old directory.

```
$ rm -rf ~/.PyCharm2016.2
```


git - [~/dims/git] - list.py - PyCharm Community Edition 2016.2.3

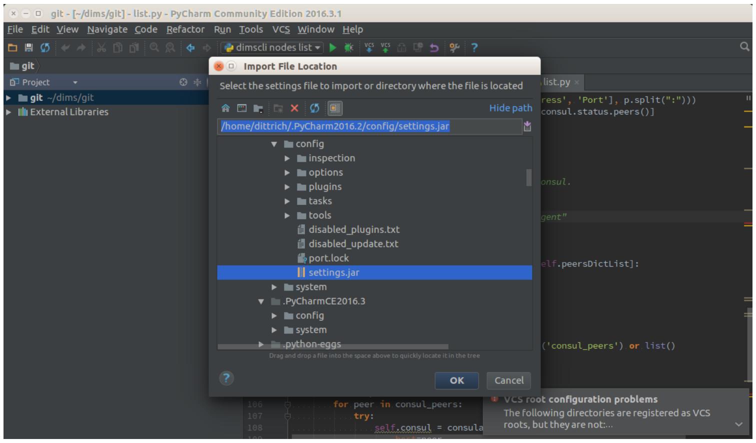
Export Settings

Please check the settings to export:

- ☑ CodeInsight
- ☑ Debugger
- ☑ Default project
- ☑ Diff
- ☑ Editor Colors
- ☑ Editor, Inspection profiles
- ☑ File templates
- ☑ Git
- ☑ Key maps
- ☑ Look and Feel
- ☑ PyCondaPackageService
- ☑ PyPackageService
- ☑ PyTestFrameworkService
- ☑ Refactoring
- ☑ Registry
- ☑ SDK Table
- ☑ SmarterEditor
- ☑ Tools
- ☑ UI Settings
- ☑ Web Browsers

Export settings to: /home/dittrich/.PyCharm2016.2/config/settings.jar

Fig. 1: Exporting Settings from PyCharm 2016.2.3


git - [~/dims/git] - list.py - PyCharm Community Edition 2016.3.1

Import File Location

Select the settings file to import or directory where the file is located

/home/dittrich/.PyCharm2016.2/config/settings.jar

```
config
    inspection
    options
    plugins
    tasks
    tools
    disabled_plugins.txt
    disabled_update.txt
    port.lock
    settings.jar
system
.PyCharmCE2016.3
  config
  system
.python-eggs
```

Drag and drop a file into the space above to quickly locate it in the tree

VCS root configuration problems
The following directories are registered as VCS roots, but they are not:...

Fig. 2: Importing Settings to PyCharm 2016.3.1

CHAPTER 9

Backups and Restoration

A good part of ongoing system administration is producing backups of files and database content that is created after initial system setup and that cannot be replaced by simply running a playbook again. Things like copies of Git repositories and the content of the PostgreSQL database and ancillary files used by the Trident portal are two primary sets of data that you will want to backup, and possibly more importantly to restore in case of a drive failure or accidental deletion.

Built into the playbooks for Letsencrypt certificate installation as part of the `nginx` role, and Trident database tables as part of the `trident-core` role, are mechanisms for automatic restoration from backups. This is very handy for development and testing using Vagrant virtual machines, since these are typically destroyed and rebuilt regularly. Restoring from backups helps more quickly get back to a functional state that possibly would trigger certificate generation limits (in the case of `nginx` and Letsencrypt) or a lot of manual actions in a graphical user interface to set up user accounts (in the case of the Trident portal).

This section will go through these backup and restoration utilities and how to use them.

9.1 Backup Directories and Files

A directory is created on DIMS systems to be used for storing backup files. In production, these files would be copied to tape, to encrypted external storage (e.g., AWS buckets), to external removable hard drives, etc.

The location for storing these files is `/opt/dims/backups` (pointed to by the Ansible global variable `dims_backups`.) After following the steps outlined in Chapter *Bootstrapping DigitalOcean Droplets*, this directory will be empty:

```
$ tree /opt/dims/backups/
/opt/dims/backups/

0 directories, 0 files
```

After completing the steps in Chapter *Creating VMs*, there will be two Trident portals (one for development/testing, and the other for production use) that have initial content put in place by the `trident-configure` role.

Caution: The trident-configure role is not entirely idempotent in relation to a running system that is manipulated manually by users and administrators. That is to say, any configuration changes made through the tcli command line interface, or the Trident portal interface, are not directly reflected in the Ansible inventory used to bootstrap the Trident portals. That means that any changes made will be reverted by the trident-configure role to what the inventory says they should be (and any new trust groups or mailing lists created manually will not be put back by the trident-configure role, which is unaware they exist).

This is an area that needs further work to be completely idempotent for long-term production systems. In the mean time, be aware of these limitations and only make configuration changes by setting variables in the inventory files and create database backups so as to keep copies of database content.

9.2 Creating a Backup

The playbook playbooks/trident_backup.yml exists to easily perform the backup operation using ansible-playbook. The playbook is very simple, as seen here:

```
---

# This playbook supports stand-alone use of the trident_backup.yml
# task file to backup the database and ancillary files of a Trident
# portal installed using the D2 Ansible playbooks and inventory.
# The tasks have been separated to allow their use from within roles.

- name: Backup trident files
  hosts: '{{ host|default("trident") }}'
  gather_facts: true
  user: root
  vars_files:
    - "{{ playbooks_root }}/vars/global.yml"
  tasks:
    - include_tasks: '{{ tasks_path }}/trident_backup.yml'

# vim: ft=ansible :
```

By default, the playbook is applied to the trident group:

```
$ ansible --list-hosts trident
  hosts (2):
    yellow.devops.develop
    purple.devops.develop
$ ansible-playbook /opt/dims/git/ansible-dims-playbooks/playbooks/trident_backup.yml

PLAY [Backup trident files ] ***************************************************

TASK [include] *****************************************************************
Saturday 12 August 2017  20:50:29 -0700 (0:00:00.064)       0:00:00.064 *******
included: /opt/dims/git/ansible-dims-playbooks/tasks/trident_backup.yml for
yellow.devops.develop, purple.devops.develop

TASK [Define local trident backup directory] ***********************************
Saturday 12 August 2017  20:50:30 -0700 (0:00:01.199)       0:00:01.264 *******
ok: [yellow.devops.develop]
ok: [purple.devops.develop]
```

(continues on next page)

(continued from previous page)

```
TASK [debug] ******************************************************************
Saturday 12 August 2017  20:50:31 -0700 (0:00:01.129)       0:00:02.394 *******
ok: [yellow.devops.develop] => {
    "trident_backup_dir": "/opt/dims/backups/yellow.devops.develop"
}
ok: [purple.devops.develop] => {
    "trident_backup_dir": "/opt/dims/backups/purple.devops.develop"
}

TASK [Define backup_ts timestamp] *********************************************
Saturday 12 August 2017  20:50:32 -0700 (0:00:01.125)       0:00:03.520 *******
ok: [yellow.devops.develop]
ok: [purple.devops.develop]

TASK [Define trident_backup_file] *********************************************
Saturday 12 August 2017  20:50:34 -0700 (0:00:02.161)       0:00:05.681 *******
ok: [yellow.devops.develop]
ok: [purple.devops.develop]

TASK [Ensure local trident backup directory exists] ***************************
Saturday 12 August 2017  20:50:35 -0700 (0:00:01.162)       0:00:06.843 *******
changed: [purple.devops.develop -> 127.0.0.1]
changed: [yellow.devops.develop -> 127.0.0.1]

TASK [Create remote temporary directory] **************************************
Saturday 12 August 2017  20:50:37 -0700 (0:00:01.463)       0:00:08.307 *******
changed: [purple.devops.develop]
changed: [yellow.devops.develop]

TASK [Define _tmpdir variable] ************************************************
Saturday 12 August 2017  20:50:38 -0700 (0:00:01.635)       0:00:09.943 *******
ok: [yellow.devops.develop]
ok: [purple.devops.develop]

TASK [Create backup of postgresql database] ***********************************
Saturday 12 August 2017  20:50:40 -0700 (0:00:01.129)       0:00:11.072 *******
changed: [purple.devops.develop]
changed: [yellow.devops.develop]

TASK [Fetch trient backup file] ***********************************************
Saturday 12 August 2017  20:50:42 -0700 (0:00:02.076)       0:00:13.148 *******
changed: [purple.devops.develop]
changed: [yellow.devops.develop]

TASK [Set backup ownership] ***************************************************
Saturday 12 August 2017  20:50:43 -0700 (0:00:01.577)       0:00:14.726 *******
changed: [yellow.devops.develop -> 127.0.0.1]
changed: [purple.devops.develop -> 127.0.0.1]

TASK [Remove temporary directory] *********************************************
Saturday 12 August 2017  20:50:45 -0700 (0:00:01.317)       0:00:16.044 *******
changed: [yellow.devops.develop]
changed: [purple.devops.develop]

PLAY RECAP ********************************************************************
purple.devops.develop            : ok=12   changed=6    unreachable=0    failed=0
yellow.devops.develop            : ok=12   changed=6    unreachable=0    failed=0
```

(continues on next page)

(continued from previous page)

```

Saturday 12 August 2017  20:50:46 -0700 (0:00:01.344)       0:00:17.388 *******
===============================================================================
Define backup_ts timestamp ---------------------------------------------- 2.16s
Create backup of postgresql database ------------------------------------ 2.08s
Create remote temporary directory --------------------------------------- 1.64s
Fetch trident backup file ----------------------------------------------- 1.58s
Ensure local trident backup directory exists ---------------------------- 1.46s
Remove temporary directory ---------------------------------------------- 1.34s
Set backup ownership ---------------------------------------------------- 1.32s
include ----------------------------------------------------------------- 1.20s
Define postgresql_backup_file ------------------------------------------- 1.16s
Define local trident backup directory ----------------------------------- 1.13s
Define _tmpdir variable ------------------------------------------------- 1.13s
debug ------------------------------------------------------------------- 1.13s
```

The backups will now show up, each in their own host's directory tree:

```
$ tree /opt/dims/backups/
/opt/dims/backups/
+-- purple.devops.develop
│   +-- trident_2017-08-12T20:50:33PDT.tar.bz2
+-- yellow.devops.develop
    +-- trident_2017-08-12T20:50:33PDT.tar.bz2

2 directories, 2 files
```

There is a similar playbook for backing up the `/etc/letsencrypt` directory with all of its certificate registration and archive history data.

```
---

# This playbook supports stand-alone use of the letsencrypt_backup.yml
# task file to backup the letsencrypt certificate store directory.
# The tasks have been separated to allow their use from within roles.
#
# See the tasks/letsencrypt_backup.yml file for usage.

- name: Backup letsencrypt certificate store
  hosts: '{{ host|default("nginx") }}'
  gather_facts: true
  user: root
  vars_files:
    - "{{ playbooks_root }}/vars/global.yml"
  tasks:
    - block:
      - include_tasks: '{{ tasks_path }}/letsencrypt_backup.yml'
      become: true

# vim: ft=ansible :
```

The default for this playbook is the `nginx` group. If you do not have an `nginx` group, or want to select a different group, define the variable `host` on the command line:

```
$ ansible-playbook $PBR/playbooks/letsencrypt_backup.yml -e host=trident

```

(continues on next page)

(continued from previous page)

```
PLAY [Backup letsencrypt certificate store] ***********************************

TASK [include] ****************************************************************
Saturday 12 August 2017  22:55:26 -0700 (0:00:00.063)       0:00:00.063 *******
included: /opt/dims/git/ansible-dims-playbooks/tasks/letsencrypt_backup.yml
for yellow.devops.develop, purple.devops.develop

TASK [Define _default_backups_dir] ********************************************
Saturday 12 August 2017  22:55:27 -0700 (0:00:01.200)       0:00:01.264 *******
ok: [yellow.devops.develop]
ok: [purple.devops.develop]

TASK [Create temporary directory for cert backup] *****************************
Saturday 12 August 2017  22:55:28 -0700 (0:00:01.126)       0:00:02.391 *******
changed: [yellow.devops.develop]
changed: [purple.devops.develop]

TASK [Define _tmpdir variable] ************************************************
Saturday 12 August 2017  22:55:30 -0700 (0:00:01.655)       0:00:04.046 *******
ok: [yellow.devops.develop]
ok: [purple.devops.develop]

TASK [Define backup_ts timestamp] *********************************************
Saturday 12 August 2017  22:55:31 -0700 (0:00:01.123)       0:00:05.170 *******
ok: [yellow.devops.develop]
ok: [purple.devops.develop]

TASK [Define certbot_backup_file] *********************************************
Saturday 12 August 2017  22:55:33 -0700 (0:00:02.157)       0:00:07.328 *******
ok: [yellow.devops.develop]
ok: [purple.devops.develop]

TASK [Create backup of letsencrypt certificates] ******************************
Saturday 12 August 2017  22:55:34 -0700 (0:00:01.123)       0:00:08.452 *******
changed: [yellow.devops.develop]
changed: [purple.devops.develop]

TASK [Ensure local cert directory exists] *************************************
Saturday 12 August 2017  22:55:36 -0700 (0:00:01.600)       0:00:10.052 *******
ok: [purple.devops.develop -> 127.0.0.1]
ok: [yellow.devops.develop -> 127.0.0.1]

TASK [Fetch backup copy of letsencrypt directory] *****************************
Saturday 12 August 2017  22:55:38 -0700 (0:00:01.465)       0:00:11.517 *******
changed: [yellow.devops.develop]
changed: [purple.devops.develop]

TASK [Note success in backing up certs] ***************************************
Saturday 12 August 2017  22:55:39 -0700 (0:00:01.488)       0:00:13.006 *******
ok: [yellow.devops.develop]
ok: [purple.devops.develop]

TASK [Set backup ownership] ***************************************************
Saturday 12 August 2017  22:55:40 -0700 (0:00:01.138)       0:00:14.145 *******
changed: [yellow.devops.develop -> 127.0.0.1]
changed: [purple.devops.develop -> 127.0.0.1]
```

(continues on next page)

(continued from previous page)

```
TASK [Remove temporary directory] *********************************************
Saturday 12 August 2017  22:55:41 -0700 (0:00:01.321)       0:00:15.467 *******
changed: [yellow.devops.develop]
changed: [purple.devops.develop]

TASK [fail] *******************************************************************
Saturday 12 August 2017  22:55:43 -0700 (0:00:01.348)       0:00:16.816 *******
skipping: [yellow.devops.develop]
skipping: [purple.devops.develop]

PLAY RECAP ********************************************************************
purple.devops.develop           : ok=12   changed=5    unreachable=0    failed=0
yellow.devops.develop           : ok=12   changed=5    unreachable=0    failed=0

Saturday 12 August 2017  22:55:44 -0700 (0:00:01.103)       0:00:17.920 *******
===============================================================================
Define backup_ts timestamp ---------------------------------------------- 2.16s
Create temporary directory for cert backup ------------------------------ 1.66s
Create backup of letsencrypt certificates ------------------------------- 1.60s
Fetch backup copy of letsencrypt directory ------------------------------ 1.49s
Ensure local cert directory exists -------------------------------------- 1.47s
Remove temporary directory ---------------------------------------------- 1.35s
Set backup ownership ---------------------------------------------------- 1.32s
include ----------------------------------------------------------------- 1.20s
Note success in backing up certs ---------------------------------------- 1.14s
Define _default_backups_dir --------------------------------------------- 1.13s
Define certbot_backup_file ---------------------------------------------- 1.12s
Define _tmpdir variable ------------------------------------------------- 1.12s
fail -------------------------------------------------------------------- 1.10s
```

You will now have a backup of the Letsencrypt certificates for both `yellow` and `purple`:

```
$ tree /opt/dims/backups/
/opt/dims/backups/
+-- purple.devops.develop
|   +-- letsencrypt_2017-08-12T22:55:32PDT.tgz
|   +-- trident_2017-08-12T20:50:33PDT.pgdmp.bz2
+-- yellow.devops.develop
    +-- letsencrypt_2017-08-12T22:55:32PDT.tgz
    +-- trident_2017-08-12T20:50:33PDT.pgdmp.bz2

2 directories, 4 files
```

9.3 Restoring from a Backup

To restore the Trident PostgreSQL and ancillary files from a backup, use the playbook `playbooks/tridenr_restore.yml`. This playbook is similar to the backup playbook, however it has no default (you must specify the host or group you want to restore explicitly).

```
---

# This playbook supports stand-alone use of the trident_restore.yml
# task file to restore the database and ancillary files of a Trident
# portal installed using the D2 Ansible playbooks and inventory. The
```

(continues on next page)

(continued from previous page)

```
# tasks have been separated to allow their use from within roles.

- name: Restore trident files
  hosts: '{{ host|default("trident") }}'
  gather_facts: true
  user: root
  vars_files:
    - "{{ playbooks_root }}/vars/global.yml"
  tasks:
    - include_tasks: '{{ tasks_path }}/trident_restore.yml'

  handlers:
    - name: restart trident
      systemd:
        name: trident
        state: started

# vim: ft=ansible :
```

To invoke this task file from within the `trident-core` role, which will pre-populate the Trident PostgreSQL database from a backup rather than running `tsetup`, as well as restore ancillary files (e.g., PGP keys and keyrings) set the variable `trident_backup_restorefrom` to point to a specific backup file, or to `latest` to have the most recent backup be applied.

There is no restore playbook for Letsencrypt certificates, however if you define the variable `certbot_backup_restorefrom` to a specific backup file path, or to `latest`, it will be restored when the `nginx` role is next applied.

9.4 Scheduled Backups

The last section showed how to manually trigger backups of Trident's PostgreSQL database and ancillary files, or Letsencrypt certificates for NGINX, using playbooks.

These tasks can be saved in `crontab` files to schedule backups for whatever frequency is desired. This is not automated at this point in time.

9.5 Other System Backups

All other backup operations will need to be performed manually, or scheduled using `crontab` as discussed in the last section.

Note: If you do create a `crontab` entry to perform backups, note the size of the backups and prepare to also prune older backups so as to not fill your hard drive. This would be a nice feature to add when resources allow it.

CHAPTER 10

License

```
Apache 2.0 License
==================

Copyright (C) 2018 David Dittrich. All rights reserved.

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.


Berkeley Three Clause License
=============================

Copyright (C) 2014-2017 University of Washington. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors
```

(continues on next page)

(continued from previous page)

```
may be used to endorse or promote products derived from this software without
specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

CHAPTER 11

Appendices

11.1 Quick Steps to Restarting Squid Proxy Container

Downloading and installing several hundred packages per host while testing provisioning of multiple Vagrant virtual machines can take several hours to perform over a 1-5 Mbps network connection. Even a single Vagrant can take around 45 minutes to fully provision after a `vagrant destroy`. Since this task may need to be done over and over again, even for just one system, the process becomes very tedious and time consuming.

To minimize the number of remote downloads, a local proxy can help immensely. The DIMS project utilizes a `squid-deb-proxy` running in a Docker container on VM host systems to allow all of the local VMs to take advantage of a single cacheing proxy on the host. This significantly improves performance (cutting the time down to just a few minutes), but this comes at a cost in occassional instability due to the combination of `iptables` firewall rules that must contain a `DOCKER` chain for Docker, which attempts to keep the `squid-deb-proxy` container running across reboots of the VM host can result in the the container effectively "hanging" from time to time. This manifests as a random failure in an Ansible task that is trying to use the configured proxy (e.g., see the `python-virtualenv` build failure in Section *Using DIMS Bash functions in Bats tests*.)

A `bats` test exists to test the proxy:

```
$ test.runner integration/proxy
[+] Running test integration/proxy
  [S][EV] HTTP download test (using wget, w/proxy if configured)
   (in test file integration/proxy.bats, line 16)
     `[ ! -z "$(wget -q -O - http://http.us.debian.org/debian/dists/jessie/Release |␣
↪grep non-free/source/Release 2>/dev/null)" ]' failed
  [S][EV] HTTPS download test (using wget, w/proxy if configured)
   (in test file integration/proxy.bats, line 26)
     `[ ! -z "$(wget -q -O - https://packages.debian.org/jessie/amd64/0install/
↪filelist | grep 0install 2>/dev/null)" ]' failed

2 tests, 2 failures
```

This error will manifest itself sometimes when doing development work on Vagrants, as can be seen here:

```
$ cd /vm/run/purple
$ make up && make DIMS_ANSIBLE_ARGS="--tags base" reprovision-local
[+] Creating Vagrantfile
. . .
TASK [base : Only "update_cache=yes" if >3600s since last update (Debian)] ****
Wednesday 16 August 2017  16:55:35 -0700 (0:00:01.968)       0:00:48.823 ******
fatal: [purple.devops.local]: FAILED! => {
    "changed": false,
    "failed": true
}

MSG:

Failed to update apt cache.


RUNNING HANDLER [base : update timezone] **************************************
Wednesday 16 August 2017  16:56:18 -0700 (0:00:43.205)       0:01:32.028 ******

PLAY RECAP ********************************************************************
purple.devops.local        : ok=15   changed=7    unreachable=0    failed=1

Wednesday 16 August 2017  16:56:18 -0700 (0:00:00.000)       0:01:32.029 ******
===============================================================================
base : Only "update_cache=yes" if >3600s since last update (Debian) ---- 43.21s
. . .
make[1]: *** [provision] Error 2
make[1]: Leaving directory `/vm/run/purple'
make: *** [reprovision-local] Error 2
```

When it fails like this, it usually means that `iptables` must be restarted, followed by restarting the `docker` service. That usually is enough to fix the problem. If not, it may be necessary to also restart the `squid-deb-proxy` container.

Note: The cause of this the recreation of the `DOCKER` chain, which removes the rules added by Docker, when restarting just the `iptables-persistent` service as can be seen here:

```
$ sudo iptables -nvL | grep "Chain DOCKER"
Chain DOCKER (2 references)
Chain DOCKER-ISOLATION (1 references)
$ sudo iptables-persistent restart
sudo: iptables-persistent: command not found
$ sudo service iptables-persistent restart
 * Loading iptables rules...
 *  IPv4...
 *  IPv6...
   ...done.
$ sudo iptables -nvL | grep "Chain DOCKER"
Chain DOCKER (0 references)
```

Restarting the `docker` service will restore the rules for containers that Docker is keeping running across restarts.

```
$ sudo service docker restart
docker stop/waiting
docker start/running, process 18276
$ sudo iptables -nvL | grep "Chain DOCKER"
```

(continues on next page)

(continued from previous page)

```
Chain DOCKER (2 references)
Chain DOCKER-ISOLATION (1 references)
```

The solution for this is to notify a special handler that conditionally restarts the docker service after restarting iptables in order to re-establish the proper firewall rules. The handler is shown here:

```
 - name: conditional restart docker
   service: name=docker state=restarted
   when: hostvars[inventory_hostname].ansible_docker0 is defined
```

Use of the handler (from roles/base/tasks/main.yml) is shown here:

```
 - name: iptables v4 rules (Debian)
   template:
     src: '{{ item }}'
     dest: /etc/iptables/rules.v4
     owner: '{{ root_user }}'
     group: '{{ root_group }}'
     mode: 0o600
     validate: '/sbin/iptables-restore --test %s'
   with_first_found:
     - files:
         - '{{ iptables_rules }}'
         - rules.v4.{{ inventory_hostname }}.j2
         - rules.v4.category-{{ category }}.j2
         - rules.v4.deployment-{{ deployment }}.j2
         - rules.v4.j2
       paths:
         - '{{ dims_private }}/roles/{{ role_name }}/templates/iptables/'
         - iptables/
   notify:
     - "restart iptables ({{ ansible_distribution }}/{{ ansible_distribution_release }
↪})"
     - "conditional restart docker"
   become: yes
   when: ansible_os_family == "Debian"
   tags: [ base, config, iptables ]
```

A tag iptables exists to allow regeneration of the iptables rules and perform the proper restarting sequence, which should be used instead of just restarting the iptables-persistent service manually. Use ansible-playbook instead (e.g., run.playbook --tags iptables) after making changes to variables that affect iptables rules.

```
$ cd $GIT/dims-dockerfiles/dockerfiles/squid-deb-proxy

$ for S in iptables-persistent docker; do sudo service $S restart; done
 * Loading iptables rules...
 *  IPv4...
 *  IPv6...
   ...done.
docker stop/waiting
docker start/running, process 22065

$ make rm
docker stop dims.squid-deb-proxy
```

(continues on next page)

(continued from previous page)

```
test.runner -dims.squid-deb-proxy
docker rm dims.squid-deb-proxy
-dims.squid-deb-proxy

$ make daemon
docker run \
          --name dims.squid-deb-proxy \
          --restart unless-stopped \
          -v /vm/cache/apt:/cachedir -p 127.0.0.1:8000:8000 squid-deb-proxy:0.7 2>&1 >
↪/dev/null &
2017/07/22 19:31:29| strtokFile: /etc/squid-deb-proxy/autogenerated/pkg-blacklist-
↪regexp.acl not found
2017/07/22 19:31:29| Warning: empty ACL: acl blockedpkgs urlpath_regex "/etc/squid-
↪deb-proxy/autogenerated/pkg-blacklist-regexp.acl"
```

The test should now succeed:

```
$ test.runner --level '*' --match proxy
[+] Running test integration/proxy
 ✓ [S][EV] HTTP download test (using wget, w/proxy if configured)
 ✓ [S][EV] HTTPS download test (using wget, w/proxy if configured)

2 tests, 0 failures
```

11.2 Recovering From Operating System Corruption

Part of the reason for using a Python virtual environment for development is to encapsulate the development Python and its libraries from the system Python and its libraries, in case a failed upgrade breaks Python. Since Python is a primary dependency of Ansible, a broken system Python is a Very Bad Thing. ™

For example, the following change was attempted to try to upgrade `pip` packages during application of the base role. Here are the changes:

```
$ git diff
diff --git a/roles/base/tasks/main.yml b/roles/base/tasks/main.yml
index 3ce57d8..182e7d8 100644
--- a/roles/base/tasks/main.yml
+++ b/roles/base/tasks/main.yml
@@ -717,7 +717,7 @@
 - name: Ensure pip installed for system python
   apt:
     name: '{{ item }}'
-    state: installed
+    state: latest
   with_items:
     - python-pip
   become: yes
@@ -725,7 +725,7 @@
   tags: [ base, config ]

 - name: Ensure required system python packages present
-  shell: 'pip install {{ item }}'
+  shell: 'pip install -U {{ item }}'
   with_items:
```

(continues on next page)

(continued from previous page)

```
     - urllib3
     - pyOpenSSL
```

Applying the `base` role against two systems resulted in a series of error messages.

```
$ ansible-playbook master.yml --limit trident --tags base

. . .

PLAY [Configure host "purple.devops.local"] **********************************

. . .

TASK [base : Ensure required system python packages present] ******************
Thursday 17 August 2017  10:36:13 -0700 (0:00:01.879)       0:02:22.637 *******
changed: [purple.devops.local] => (item=urllib3)
failed: [purple.devops.local] (item=pyOpenSSL) => {
    "changed": true,
    "cmd": "pip install -U pyOpenSSL",
    "delta": "0:00:07.516760",
    "end": "2017-08-17 10:36:24.256121",
    "failed": true,
    "item": "pyOpenSSL",
    "rc": 1,
    "start": "2017-08-17 10:36:16.739361"
}

STDOUT:

Downloading/unpacking pyOpenSSL from https://pypi.python.org/packages/41/bd/
↪751560b317222ba6b6d2e7663a990ac36465aaa026621c6057db130e2faf/pyOpenSSL-17.2.0-py2.
↪py3-none-any.whl#md5=0f8a4b784b6
81231f03edc8dd28612df
Downloading/unpacking six>=1.5.2 from https://pypi.python.org/packages/c8/0a/
↪b6723e1bc4c516cb687841499455a8505b44607ab535be01091c0f24f079/six-1.10.0-py2.py3-
↪none-any.whl#md5=3ab558cf5d4f7a72
611d59a81a315dc8 (from pyOpenSSL)
  Downloading six-1.10.0-py2.py3-none-any.whl
Downloading/unpacking cryptography>=1.9 (from pyOpenSSL)
  Running setup.py (path:/tmp/pip-build-FCbUwT/cryptography/setup.py) egg_info for␣
↪package cryptography

    no previously-included directories found matching 'docs/_build'
    warning: no previously-included files matching '*' found under directory 'vectors'
Downloading/unpacking idna>=2.1 (from cryptography>=1.9->pyOpenSSL)
Downloading/unpacking asn1crypto>=0.21.0 (from cryptography>=1.9->pyOpenSSL)
Downloading/unpacking enum34 (from cryptography>=1.9->pyOpenSSL)
  Downloading enum34-1.1.6-py2-none-any.whl
Downloading/unpacking ipaddress (from cryptography>=1.9->pyOpenSSL)
  Downloading ipaddress-1.0.18-py2-none-any.whl
Downloading/unpacking cffi>=1.7 (from cryptography>=1.9->pyOpenSSL)
  Running setup.py (path:/tmp/pip-build-FCbUwT/cffi/setup.py) egg_info for package␣
↪cffi

Downloading/unpacking pycparser from https://pypi.python.org/packages/8c/2d/
↪aad7f16146f4197a11f8e91fb81df177adcc2073d36a17b1491fd09df6ed/pycparser-2.18.tar.gz
↪#md5=72370da54358202a60130e223d4
```

(continues on next page)

(continued from previous page)

```
88136 (from cffi>=1.7->cryptography>=1.9->pyOpenSSL)
  Running setup.py (path:/tmp/pip-build-FCbUwT/pycparser/setup.py) egg_info for␣
↪package pycparser

    warning: no previously-included files matching 'yacctab.*' found under directory
↪'tests'
    warning: no previously-included files matching 'lextab.*' found under directory
↪'tests'
    warning: no previously-included files matching 'yacctab.*' found under directory
↪'examples'
    warning: no previously-included files matching 'lextab.*' found under directory
↪'examples'
Installing collected packages: pyOpenSSL, six, cryptography, idna, asn1crypto, enum34,
↪ ipaddress, cffi, pycparser
  Found existing installation: pyOpenSSL 0.14
    Not uninstalling pyOpenSSL at /usr/lib/python2.7/dist-packages, owned by OS
  Found existing installation: six 1.8.0
    Not uninstalling six at /usr/lib/python2.7/dist-packages, owned by OS
  Found existing installation: cryptography 0.6.1
    Not uninstalling cryptography at /usr/lib/python2.7/dist-packages, owned by OS
  Running setup.py install for cryptography

    Installed /tmp/pip-build-FCbUwT/cryptography/cffi-1.10.0-py2.7-linux-x86_64.egg
    Traceback (most recent call last):
      File "<string>", line 1, in <module>
      File "/tmp/pip-build-FCbUwT/cryptography/setup.py", line 312, in <module>
        **keywords_with_side_effects(sys.argv)
      File "/usr/lib/python2.7/distutils/core.py", line 111, in setup
        _setup_distribution = dist = klass(attrs)
      File "/usr/lib/python2.7/dist-packages/setuptools/dist.py", line 266, in __init_
↪_
        _Distribution.__init__(self,attrs)
      File "/usr/lib/python2.7/distutils/dist.py", line 287, in __init__
        self.finalize_options()
      File "/usr/lib/python2.7/dist-packages/setuptools/dist.py", line 301, in␣
↪finalize_options
        ep.load()(self, ep.name, value)
      File "/usr/lib/python2.7/dist-packages/pkg_resources.py", line 2190, in load
        ['__name__'])
    ImportError: No module named setuptools_ext
    Complete output from command /usr/bin/python -c "import setuptools, tokenize;__
↪file__='/tmp/pip-build-FCbUwT/cryptography/setup.py';exec(compile(getattr(tokenize,
↪'open', open)(__file__)
.read().replace('\r\n', '\n'), __file__, 'exec'))" install --record /tmp/pip-qKjzie-
↪record/install-record.txt --single-version-externally-managed --compile:


Installed /tmp/pip-build-FCbUwT/cryptography/cffi-1.10.0-py2.7-linux-x86_64.egg

Traceback (most recent call last):

  File "<string>", line 1, in <module>

  File "/tmp/pip-build-FCbUwT/cryptography/setup.py", line 312, in <module>

    **keywords_with_side_effects(sys.argv)
```

(continues on next page)

(continued from previous page)

```
  File "/usr/lib/python2.7/distutils/core.py", line 111, in setup

    _setup_distribution = dist = klass(attrs)

  File "/usr/lib/python2.7/dist-packages/setuptools/dist.py", line 266, in __init__

    _Distribution.__init__(self,attrs)

  File "/usr/lib/python2.7/distutils/dist.py", line 287, in __init__

    self.finalize_options()

  File "/usr/lib/python2.7/dist-packages/setuptools/dist.py", line 301, in finalize_
↪options

    ep.load()(self, ep.name, value)

  File "/usr/lib/python2.7/dist-packages/pkg_resources.py", line 2190, in load

    ['__name__'])

ImportError: No module named setuptools_ext

----------------------------------------
  Can't roll back cryptography; was not uninstalled
Cleaning up...
Command /usr/bin/python -c "import setuptools, tokenize;__file__='/tmp/pip-build-
↪FCbUwT/cryptography/setup.py';exec(compile(getattr(tokenize, 'open', open)(__file__
↪).read().replace('\r\n', '
\n'), __file__, 'exec'))" install --record /tmp/pip-qKjzie-record/install-record.txt -
↪-single-version-externally-managed --compile failed with error code 1 in /tmp/pip-
↪build-FCbUwT/cryptogra
phy
Storing debug log for failure in /root/.pip/pip.log

. . .

PLAY RECAP *********************************************************************
purple.devops.local        : ok=60   changed=35   unreachable=0    failed=1

Thursday 17 August 2017  10:36:29 -0700 (0:00:00.001)       0:02:38.799 *******
===============================================================================
base : Ensure required system python packages present ------------------ 16.16s
base : Ensure dims (system-level) subdirectories exist ----------------- 15.85s
base : Only "update_cache=yes" if >3600s since last update (Debian) ----- 5.65s
base : conditional restart docker --------------------------------------- 5.60s
base : Make sure required APT packages are present (Debian) ------------- 2.14s
base : Clean up dnsmasq build artifacts --------------------------------- 2.09s
base : Make sure blacklisted packages are absent (Debian) --------------- 2.03s
base : Check to see if https_proxy is working --------------------------- 1.99s
base : Log start of 'base' role ----------------------------------------- 1.95s
base : Make backports present for APT on Debian jessie ------------------ 1.89s
base : Ensure pip installed for system python --------------------------- 1.88s
base : Only "update_cache=yes" if >3600s since last update -------------- 1.85s
base : Make dbus-1 development libraries present ------------------------ 1.85s
base : iptables v4 rules (Debian) --------------------------------------- 1.84s
base : iptables v6 rules (Debian) --------------------------------------- 1.84s
```

(continues on next page)

(continued from previous page)

```
base : Make full dnsmasq package present (Debian, not Trusty) ----------- 1.82s
base : Create base /etc/hosts file (Debian, RedHat, CoreOS) ------------- 1.64s
base : Make /etc/rsyslog.d/49-consolidation.conf present ---------------- 1.63s
base : Make dnsmasq configuration present on Debian --------------------- 1.60s
base : Ensure DIMS system shell init hook is present (Debian, CoreOS) --- 1.56s
```

The `base` role is supposed to ensure the operating system has the fundamental settings and pre-requisites necessary for all other DIMS roles, so applying that role should *hopefully* fix things, right?

```
$ ansible-playbook master.yml --limit trident --tags base

. . .

PLAY [Configure host "purple.devops.local"] ***********************************

. . .

TASK [base : Make sure blacklisted packages are absent (Debian)] **************
Thursday 17 August 2017  11:05:08 -0700 (0:00:01.049)       0:00:30.456 *******
...ignoring
An exception occurred during task execution. To see the full traceback, use
-vvv. The error was: AttributeError: 'FFI' object has no attribute 'new_allocator'
failed: [purple.devops.local] (item=[u'modemmanager', u'resolvconf', u'sendmail']) =>
↪{
    "failed": true,
    "item": [
        "modemmanager",
        "resolvconf",
        "sendmail"
    ],
    "module_stderr": "Traceback (most recent call last):\n  File \"/tmp/ansible_
↪ehzfMx/
    ansible_module_apt.py\", line 239, in <module>\n    from ansible.module_utils.
↪urls import fetch_url\n
File \"/tmp/ansible_ehzfMx/ansible_modlib.zip/ansible/module_utils/urls.py\", line␣
↪153,
in <module>\n  File \"/usr/local/lib/python2.7/dist-packages/urllib3/contrib/
↪pyopenssl.py\", line 46,
in <module>\n    import OpenSSL.SSL\n  File \"/usr/local/lib/python2.7/dist-packages/
↪OpenSSL/__init__.py\",
line 8, in <module>\n    from OpenSSL import rand, crypto, SSL\n  File \"/usr/local/
↪lib/
python2.7/dist-packages/OpenSSL/rand.py\", line 10, in <module>\n    from OpenSSL._
↪util
import (\n  File \"/usr/local/lib/python2.7/dist-packages/OpenSSL/_util.py\", line 18,
↪ in
<module>\n    no_zero_allocator = ffi.new_allocator(should_clear_after_alloc=False)\n
AttributeError: 'FFI' object has no attribute 'new_allocator'\n",
    "module_stdout": "",
    "rc": 1
}

MSG:

MODULE FAILURE
```

(continues on next page)

(continued from previous page)

```
TASK [base : Only "update_cache=yes" if >3600s since last update (Debian)] ****
Thursday 17 August 2017  11:05:10 -0700 (0:00:01.729)       0:00:32.186 *******
An exception occurred during task execution. To see the full traceback, use -vvv.
The error was: AttributeError: 'FFI' object has no attribute 'new_allocator'
fatal: [purple.devops.local]: FAILED! => {
    "changed": false,
    "failed": true,
    "module_stderr": "Traceback (most recent call last):\n  File \"/tmp/ansible_
 ↪ganqlZ/
    ansible_module_apt.py\", line 239, in <module>\n    from ansible.module_utils.
 ↪urls import fetch_url\n
File \"/tmp/ansible_ganqlZ/ansible_modlib.zip/ansible/module_utils/urls.py\", line␣
 ↪153, in
<module>\n  File \"/usr/local/lib/python2.7/dist-packages/urllib3/contrib/pyopenssl.
 ↪py\", line 46,
in <module>\n    import OpenSSL.SSL\n  File \"/usr/local/lib/python2.7/dist-packages/
OpenSSL/__init__.py\", line 8, in <module>\n    from OpenSSL import rand, crypto,␣
 ↪SSL\n
File \"/usr/local/lib/python2.7/dist-packages/OpenSSL/rand.py\", line 10, in <module>
 ↪\n
from OpenSSL._util import (\n  File \"/usr/local/lib/python2.7/dist-packages/OpenSSL/_
 ↪util.py\",
line 18, in <module>\n    no_zero_allocator = ffi.new_allocator(should_clear_after_
 ↪alloc=False)\n
AttributeError: 'FFI' object has no attribute 'new_allocator'\n",
    "module_stdout": "",
    "rc": 1
}

MSG:

MODULE FAILURE


RUNNING HANDLER [base : update timezone] **************************************
Thursday 17 August 2017  11:05:11 -0700 (0:00:01.530)       0:00:33.716 *******

PLAY RECAP ********************************************************************
purple.devops.local        : ok=14   changed=7    unreachable=0    failed=1

Thursday 17 August 2017  11:05:11 -0700 (0:00:00.001)       0:00:33.717 *******
===============================================================================
base : Log start of 'base' role ----------------------------------------- 1.88s
base : Make sure blacklisted packages are absent (Debian) --------------- 1.73s
base : Create base /etc/hosts file (Debian, RedHat, CoreOS) ------------- 1.55s
base : Only "update_cache=yes" if >3600s since last update (Debian) ----- 1.53s
base : Set timezone variables (Debian) ---------------------------------- 1.53s
base : iptables v6 rules (Debian) --------------------------------------- 1.48s
base : iptables v4 rules (Debian) --------------------------------------- 1.48s
base : Ensure getaddrinfo configuration is present (Debian) ------------- 1.48s
base : Check to see if dims.logger exists yet --------------------------- 1.31s
base : Set domainname (Debian, CoreOS) ---------------------------------- 1.17s
base : Check to see if gpk-update-viewer is running on Ubuntu ----------- 1.16s
base : Set hostname (runtime) (Debian, CoreOS) -------------------------- 1.16s
base : Make /etc/hostname present (Debian, CoreOS) ---------------------- 1.16s
base : Disable IPv6 in kernel on non-CoreOS ----------------------------- 1.16s
```

(continues on next page)

(continued from previous page)

```
debug : include ---------------------------------------------------------- 1.07s
base : iptables v4 rules (CoreOS) ---------------------------------------- 1.06s
base : iptables v6 rules (CoreOS) ---------------------------------------- 1.06s
debug : debug ------------------------------------------------------------ 1.05s
debug : debug ------------------------------------------------------------ 1.05s
debug : debug ------------------------------------------------------------ 1.05s
```

Since Debian `apt` is a Python program, it requires Python to install packages. The Python packages are corrupted, so Python will not work properly. This creates a deadlock condition. There is another way to install Python packages, however, so it can be used via Ansible ad-hoc mode:

```
$ ansible -m shell --become -a 'easy_install -U cffi' trident
yellow.devops.local | SUCCESS | rc=0 >>
Searching for cffi
Reading https://pypi.python.org/simple/cffi/
Best match: cffi 1.10.0
Downloading https://pypi.python.org/packages/5b/b9/
↪790f8eafcdab455bcd3bd908161f802c9ce5adbf702a83aa7712fcc345b7/cffi-1.10.0.tar.gz
↪#md5=2b5fa41182ed0edaf929a789e602a070
Processing cffi-1.10.0.tar.gz
Writing /tmp/easy_install-RmOJBU/cffi-1.10.0/setup.cfg
Running cffi-1.10.0/setup.py -q bdist_egg --dist-dir /tmp/easy_install-RmOJBU/cffi-1.
↪10.0/egg-dist-tmp-lNCOck
compiling '_configtest.c':
__thread int some_threadlocal_variable_42;

x86_64-linux-gnu-gcc -pthread -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-prototypes -fno-
↪strict-aliasing -D_FORTIFY_SOURCE=2 -g -fstack-protector-strong -Wformat -
↪Werror=format-security -fPIC -c
 _configtest.c -o _configtest.o
success!
removing: _configtest.c _configtest.o
compiling '_configtest.c':
int main(void) { __sync_synchronize(); return 0; }

x86_64-linux-gnu-gcc -pthread -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-prototypes -fno-
↪strict-aliasing -D_FORTIFY_SOURCE=2 -g -fstack-protector-strong -Wformat -
↪Werror=format-security -fPIC -c
 _configtest.c -o _configtest.o
x86_64-linux-gnu-gcc -pthread _configtest.o -o _configtest
success!
removing: _configtest.c _configtest.o _configtest
Adding cffi 1.10.0 to easy-install.pth file

Installed /usr/local/lib/python2.7/dist-packages/cffi-1.10.0-py2.7-linux-x86_64.egg
Processing dependencies for cffi
Finished processing dependencies for cffi

purple.devops.local | SUCCESS | rc=0 >>
Searching for cffi
Reading https://pypi.python.org/simple/cffi/
Best match: cffi 1.10.0
Downloading https://pypi.python.org/packages/5b/b9/
↪790f8eafcdab455bcd3bd908161f802c9ce5adbf702a83aa7712fcc345b7/cffi-1.10.0.tar.gz
↪#md5=2b5fa41182ed0edaf929a789e602a070
Processing cffi-1.10.0.tar.gz
```

(continues on next page)

(continued from previous page)

```
Writing /tmp/easy_install-fuS4hd/cffi-1.10.0/setup.cfg
Running cffi-1.10.0/setup.py -q bdist_egg --dist-dir /tmp/easy_install-fuS4hd/cffi-1.
 ↪10.0/egg-dist-tmp-nOgko4
compiling '_configtest.c':
__thread int some_threadlocal_variable_42;

x86_64-linux-gnu-gcc -pthread -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-prototypes -fno-
 ↪strict-aliasing -D_FORTIFY_SOURCE=2 -g -fstack-protector-strong -Wformat -
 ↪Werror=format-security -fPIC -c
 _configtest.c -o _configtest.o
success!
removing: _configtest.c _configtest.o
compiling '_configtest.c':
int main(void) { __sync_synchronize(); return 0; }

x86_64-linux-gnu-gcc -pthread -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-prototypes -fno-
 ↪strict-aliasing -D_FORTIFY_SOURCE=2 -g -fstack-protector-strong -Wformat -
 ↪Werror=format-security -fPIC -c
 _configtest.c -o _configtest.o
x86_64-linux-gnu-gcc -pthread _configtest.o -o _configtest
success!
removing: _configtest.c _configtest.o _configtest
Adding cffi 1.10.0 to easy-install.pth file

Installed /usr/local/lib/python2.7/dist-packages/cffi-1.10.0-py2.7-linux-x86_64.egg
Processing dependencies for cffi
Finished processing dependencies for cffi
```

Now we can back out the addition of the `-U` flag that caused the corruption and apply the base role to the two hosts using the `master.yml` playbook.

```
$ ansible-playbook master.yml --limit trident --tags base

. . .

PLAY [Configure host "purple.devops.local"] ***********************************

. . .

PLAY [Configure host "yellow.devops.local"] ***********************************

. . .

PLAY RECAP ********************************************************************
purple.devops.local        : ok=136  changed=29   unreachable=0    failed=0
yellow.devops.local        : ok=139  changed=53   unreachable=0    failed=0

Thursday 17 August 2017  11:20:08 -0700 (0:00:01.175)       0:10:03.307 *******
===============================================================================
base : Make defined bats tests present --------------------------------- 29.18s
base : Make defined bats tests present --------------------------------- 28.95s
base : Ensure dims (system-level) subdirectories exist ----------------- 15.89s
base : Ensure dims (system-level) subdirectories exist ----------------- 15.84s
base : Ensure required system python packages present ------------------- 8.81s
base : Make sure common (non-templated) BASH scripts are present -------- 8.79s
base : Make sure common (non-templated) BASH scripts are present -------- 8.74s
```

(continues on next page)

(continued from previous page)

```
base : Ensure required system python packages present ------------------- 8.71s
base : Make subdirectories for test categories present ------------------ 6.84s
base : Make links to helper functions present --------------------------- 6.83s
base : Make subdirectories for test categories present ------------------ 6.83s
base : Make links to helper functions present --------------------------- 6.81s
base : Ensure bashrc additions are present ------------------------------ 4.63s
base : Ensure bashrc additions are present ------------------------------ 4.59s
base : Only "update_cache=yes" if >3600s since last update (Debian) ----- 4.45s
base : Make sure common (non-templated) Python scripts are present ------ 3.77s
base : Make sure common (non-templated) Python scripts are present ------ 3.77s
base : conditional restart docker --------------------------------------- 3.17s
base : Make sure common (templated) scripts are present ----------------- 2.96s
base : Make sure common (templated) scripts are present ----------------- 2.94s
```

In this case, the systems are now back to a functional state and the disruptive change backed out. Were these Vagrants, the problem of a broken system is lessened, so testing should *always* be done first on throw-away VMs. But on those occassions where something goes wrong on “production” hosts, Ansible ad-hoc mode is a powerful debugging and corrective capability.

11.3 Advanced Ansible Tasks or Jinja Templating

This section includes some advanced uses of Ansible task declaration and/or Jinja templating that may be difficult to learn from Ansible documentation or other sources. Some useful resources that were identified during the DIMS Project are listed in Section bestpractices.

11.3.1 Multi-line `fail` or `debug` Output

There are times when it is necessary to produce a long message in a `fail` or `debug` play. An answer to the stackoverflow post In YAML, how do I break a string over multiple lines? includes multiple ways to do this. Here is one of them in action in the `virtualbox` role:

```
---

# File: roles/virtualbox/tasks/main.yml

# Note: You can't just run 'vboxmanage list runningvms' to get
# a list of running VMs, unless running as the user that started
# the VMs. (Virtualbox keeps state on a per-user basis.)
# Instead, we are looking for running processes.

- name: Look for running VM guests
  shell: "ps -Ao user,pid,cmd|egrep '^USER|virtualbox/VBox.* --startvm'|grep -v '␣
↪egrep '"
  register: _vbox_result
  delegate_to: '{{ inventory_hostname }}'
  tags: [ virtualbox, packages ]

- name: Register number of running VM guests
  set_fact:
    _running_vms: '{{ ((_vbox_result.stdout_lines|length) - 1) }}'
  when: _vbox_result is defined
  tags: [ virtualbox, packages ]
```

(continues on next page)

(continued from previous page)

```
- block:
  - fail: msg='Could not determine number of running VMs'
    when: _vbox_result is not defined or _running_vms is not defined
    tags: [ virtualbox, packages ]

  rescue:
  - name: Register failure
    set_fact:
      _running_vms: -1
    tags: [ virtualbox, packages ]

- fail:
    msg: |
      Found {{ _running_vms }} running Virtualbox VM{{ (_running_vms|int ==␣
↪1)|ternary("","s") }}.
      Virtualbox cannot be updated while VMs are running.
      Please halt or suspend {{ (_running_vms|int == 1)|ternary("this VM","these VMs
↪") }} and apply this role again.
      {% raw %}{% endraw %}
      {% for line in _vbox_result.stdout_lines|default([]) %}
      {{ line }}
      {% endfor %}
  when: _running_vms is defined and _running_vms|int >= 1
  tags: [ virtualbox, packages ]

- import_tasks: virtualbox.yml
  when: _running_vms is defined and _running_vms|int == 0
  tags: [ virtualbox, packages ]

# vim: ft=ansible :
```

When this code is triggered, the output is now clean and clear about what to do.

```
TASK [virtualbox : fail]␣
↪*******************************************************************
task path: /home/dittrich/dims/git/ansible-dims-playbooks/roles/virtualbox/tasks/main.
↪yml:33
Wednesday 06 September 2017  12:45:38 -0700 (0:00:01.046)       0:00:51.117 ***
fatal: [dimsdemo1.devops.develop]: FAILED! => {
    "changed": false,
    "failed": true
}

MSG:

Found 1 running Virtualbox VM.
Virtualbox cannot be updated while VMs are running.
Please halt or suspend this VM and apply this role again.

USER       PID CMD
dittrich 15289 /usr/lib/virtualbox/VBoxHeadless --comment orange_default_
↪1504485887221_79778 --startvm 62e20c31-7c2c-417a-a5ab-3a056aa81e2d --vrde config
```

11.4 Leveraging the Terraform State File

Terraform maintains state in a file named `terraform.tfstate` (and a backup file `terraform.tfstate.backup`) in the home directory where Terraform was initialized. While the `terraform.tfstate` file is a JSON object that can be manipulated using programs like jq, the proper way to exploit this state is to use `terraform output --json`.

11.4.1 Introduction to `jq`

To better understand how to manipulate the contents of the `terraform.tfstate` file with `jq`, we will start out by directly manipulating the file so we don't have to *also* struggle with defining Terraform `output` variables.

Note: See Reshaping JSON with jq for examples of how to use `jq`.

Using the filter `.` with `jq` will show the entire structure. Here are the first 10 lines in a `terraform.tfstate` file

```
$ jq -r '.' terraform.tfstate | head
{
  "version": 3,
  "terraform_version": "0.11.1",
  "serial": 7,
  "lineage": "755c781e-407c-41e2-9f10-edd0b80bcc9f",
  "modules": [
    {
      "path": [
        "root"
      ],
```

Note: To more easily read the JSON, you can pipe the output through `pygmentize` to colorize it, then `less -R` to preserve the ANSI colorization codes. The command line to use is:

```
$ jq -r '.' terraform.tfstate | pygmentize | less -R
```

By choosing a specific field for the filter, `jq` will print just that field.

```
$ jq -r '.lineage' terraform.tfstate
755c781e-407c-41e2-9f10-edd0b80bcc9f
```

Adding `[]` to a field that is an array produces a list, and piping filters with a `|` allows additional filtering to be applied to narrow the results. Functions like `select()` can be used to extract a specific field from a list element that is a dictionary, allowing selection of just specific members. In the next example, the nested structures named `resources` within the structure `modules` are evaluated, selecting only those where the `type` field is `digitalocean_record` (i.e., DNS records).

```
$ jq -r '.modules[] | .resources[] | select(.type | test("digitalocean_record"))' terraform.tfstate
```

The first record is highlighted in the output here. Within the record are two fields (`.primary.attributes.fqdn` and `.primary.attributes.value`) that are needed to help build `/etc/hosts` style DNS mappings, or to generate a YAML inventory file.

```
{
  "type": "digitalocean_record",
  "depends_on": [
    "digitalocean_domain.default",
    "digitalocean_droplet.blue"
  ],
  "primary": {
    "id": "XXXXXXXX",
    "attributes": {
      "domain": "example.com",
      "fqdn": "blue.example.com",
      "id": "XXXXXXXX",
      "name": "blue",
      "port": "0",
      "priority": "0",
      "ttl": "360",
      "type": "A",
      "value": "XXX.XXX.XXX.XX",
      "weight": "0"
    },
    "meta": {},
    "tainted": false
  },
  "deposed": [],
  "provider": "provider.digitalocean"
}
{
  "type": "digitalocean_record",
  "depends_on": [
    "digitalocean_domain.default",
    "digitalocean_droplet.orange"
  ],
  "primary": {
    "id": "XXXXXXXX",
    "attributes": {
      "domain": "example.com",
      "fqdn": "orange.example.com",
      "id": "XXXXXXXX",
      "name": "orange",
      "port": "0",
      "priority": "0",
      "ttl": "360",
      "type": "A",
      "value": "XXX.XXX.XXX.XXX",
      "weight": "0"
    },
    "meta": {},
    "tainted": false
  },
  "deposed": [],
  "provider": "provider.digitalocean"
}
```

By adding another pipe step to create an list item with just these two fields, and adding the -c option to create a single-line JSON object.

```
$ jq -c '.modules[] | .resources[] | select(.type | test("digitalocean_record")) | [ .primary.attributes.fqdn, .primary.attributes.value ]' terraform.tfstate
```

```
["blue.example.com","XXX.XXX.XXX.XX"]
["orange.example.com","XXX.XXX.XXX.XXX"]
```

These can be further converted into formats parseable by Unix shell programs like `awk`, etc., using the filters `@csv` or `@sh`:

```
$ jq -r '.modules[] | .resources[] | select(.type | test("digitalocean_record")) | [ .
↪primary.attributes.name, .primary.attributes.fqdn, .primary.attributes.value ]| @csv
↪' terraform.tfstate
"blue","blue.example.com","XXX.XXX.XXX.XX"
"orange","orange.example.com","XXX.XXX.XXX.XXX"
$ jq -r '.modules[] | .resources[] | select(.type | test("digitalocean_record")) | [ .
↪primary.attributes.name, .primary.attributes.fqdn, .primary.attributes.value ]| @sh
↪' terraform.tfstate
'blue' 'blue.example.com' 'XXX.XXX.XXX.XX'"
'blue' 'orange.example.com' 'XXX.XXX.XXX.XXX'"
```

11.4.2 Processing `terraform output --json`

While processing the `terraform.tfstate` file directly is possible, the proper way to use Terraform state is to create **output** variables and expose them using `terraform output`:

```
$ terraform output
blue = {
  blue.example.com = XXX.XX.XXX.XXX
}
orange = {
  orange.example.com = XXX.XX.XXX.XXX
}
```

This output could be processed with `awk`, but we want to use `jq` instead to more directly process the output using JSON. To get JSON output, add the `--json` flag:

```
$ terraform output --json
{
    "blue": {
        "sensitive": false,
        "type": "map",
        "value": {
            "blue.example.com": "XXX.XX.XXX.XXX"
        }
    },
    "orange": {
        "sensitive": false,
        "type": "map",
        "value": {
            "orange.example.com": "XXX.XX.XXX.XXX"
        }
    }
}
```

To get to clean single-line, multi-colum output, we need to use `to_entries[]` to turn the dictionaries into key/value pairs, nested two levels deep in this case.

```
$ terraform output --json | jq -r 'to_entries[] | [ .key, (.value.value|to_entries[]|␣
↪.key, .value) ]|@sh'
'blue' 'blue.example.com' 'XXX.XX.XXX.XXX'
'orange' 'orange.example.com' 'XXX.XX.XXX.XXX'
```

Putting all of this together with a much simpler `awk` script, a YAML inventory file can be produced as shown in the script `files/common-scripts/terraform.inventory.generate.sh`.

```
#!/bin/bash
#
# vim: set ts=4 sw=4 tw=0 et :
#
# Copyright (C) 2018, David Dittrich. All rights reserved.
#
#  Licensed under the Apache License, Version 2.0 (the "License");
#  you may not use this file except in compliance with the License.
#  You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
#  Unless required by applicable law or agreed to in writing, software
#  distributed under the License is distributed on an "AS IS" BASIS,
#  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#  See the License for the specific language governing permissions and
#  limitations under the License.
#
# This script was inspired by https://gist.github.com/leucos/6f8d93de3493431acd29

cat <<EOD
---
# This is a generated inventory file produced by $0.
# DO NOT EDIT THIS FILE.

do:
  hosts:
EOD

# Create inventory for running droplets from terraform state file
terraform output --json | jq -r 'to_entries[] | [ .key, (.value.value|to_entries[]| .
↪key, .value) ]|@sh' |
awk '{
    printf "    %s:\n", $1
    printf "      dims_fqdn: %s\n", $2
    printf "      ansible_host: %s\n", $3
     }'

exit 0
```

```
---
# This is a generated inventory file produced by /Users/dittrich/dims/git/ansible-
↪dims-playbooks/files/common-scripts/terraform.inventory.generate.sh.
# DO NOT EDIT THIS FILE.

do:
  hosts:
    'blue':
      ansible_host: 'XXX.XXX.XXX.XX'
```

(continues on next page)

(continued from previous page)

```
      ansible_fqdn: 'blue.example.com'
    'orange':
      ansible_host: 'XXX.XXX.XXX.XXX'
      ansible_fqdn: 'orange.example.com'
```

This inventory file can then be used by Ansible to perform ad-hoc tasks or run playbooks.

```
$ make ping
ansible -i ../../environments/do/inventory \
                 \
                -m ping do
orange | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
blue | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
```

Section author: Dave Dittrich dave.dittrich@gmail.com