DIMS Ansible playbooks Documentation

Release 2.8.0

Dave Dittrich

Aug 02, 2017

Contents:

This document (version 2.8.0) describes the DIMS Ansible playbooks (`ansible-dims-playbooks` for short) repository contents.

CHAPTER 1

Introduction

This chapter documents the DIMS Ansible playbooks (`ansible-dims-playbooks` for short) repository.

This repository contains the Ansible playbooks and inventory for a development/test environment. This is conventionally known as a `local` deployment, as it comprises a baremetal host system intended to serve as an *Ansible control host*, with a series of virtual machines to provide services.

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.

- 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 *Ansible Best Practices and Related Documentation* (originally collected at <https://staff.washington.edu/dittrich/home/unix.html#ansible>).

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.

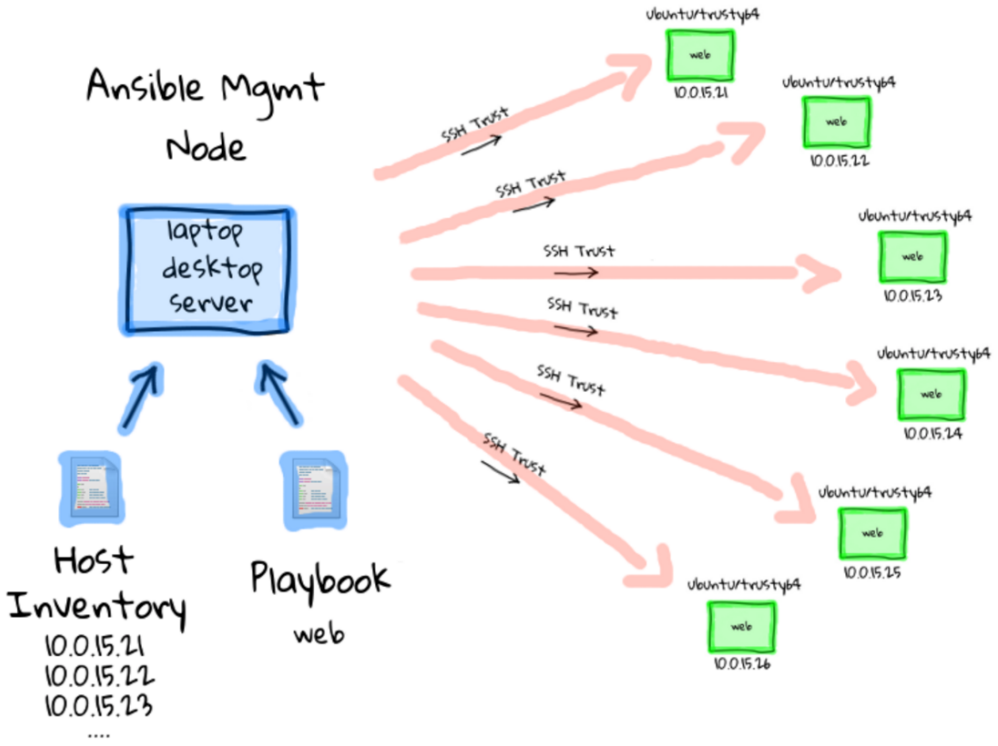

Fig. 2.1: Ansible Overview

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 consiciously 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!

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",
            "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",
        "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",
            "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@uw.ops-trust.net",
            "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",
            "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"
        },
        "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",
        "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",
            "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.

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
```

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

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 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 `ansbile-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
- 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

Bootstrapping a VM Host as an Ansible Controller

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: We are assuming that you have set up `/etc/ansible/ansible.cfg`, or a perhaps `~/.ansible.cfg`, 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.
  . . .

..
```

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

```
hosts (11):
  blue14.devops.local
  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
```

We now validate the temporary `bootstrap` group that defines the two hosts we are setting up.

```
$ export ANSIBLE_HOST_KEY_CHECKING=False
$ 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 to now use Ansible ad-hoc mode with 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 get the path to the private key, adding the `.pub` extension for 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,
    "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
}
```

Now remove the `--ask-pass` option to instead use the specified SSH private key to validate that standard remote access with Ansible will work.

```
$ ansible -m raw -a uptime  bootstrap
dellr510.devops.develop | SUCCESS | rc=0 >>
 22:33:44 up  3:49,  3 users,  load average: 1.14, 0.81, 0.99
Shared connection to 140.142.29.186 closed.
```

```
stirling.devops.develop | SUCCESS | rc=0 >>
 22:33:44 up  4:27,  3 users,  load average: 1.12, 1.10, 1.03
Shared connection to 140.142.29.161 closed.
```

CHAPTER 4

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.

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 `$GIT/dims-ci-utils/cookiecutter`. When testing or using DIMS cookiecutters, make sure to have an updated `dims-ci-utils` repo.

The directory `$GIT/dims-ci-utils/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.

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.

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
|   |       +- UW-logo-16x16.ico
|   |       +- UW-logo-32x32.ico
|   |       +- UW-logo.ico
|   |       +- UW-logo.png
|   +- README.rst
|   +- VERSION
+- hooks
    +- post_gen_project.sh

7 directories, 18 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/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
|       +- UW-logo-16x16.ico
|       +- UW-logo-32x32.ico
|       +- UW-logo.ico
|       +- UW-logo.png
+- VERSION

4 directories, 10 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/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:

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/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.

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: "dittrich@u.washington.edu"
  project_name: "DIMS Ansible Playbooks"
  project_slug: "ansible-dims-playbooks"
  project_short_description: "Ansible Playbooks for DIMS System Configuration"
  project_copyright_name: "University of Washington"
```

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]: Megan Boggess
email []: mboggess@uw.edu
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]: 2015-10-29
project_version [1.0.0]:
project_slug [test-repo]:
project_copyright_name [University of Washington]:
project_copyright_date [2014-2015]:
$ cd ~/test-repo
$ ls
docs  VERSION
$ tree -a
.
+- .bumpversion.cfg
+- docs
|   +- build
|   |   +- .gitignore
|   +- Makefile
|   +- source
|       +- conf.py
|       +- images
|       +- index.rst
|       +- license.rst
|       +- license.txt
|       +- static
|       |   +- .gitignore
|       +- templates
|       |   +- .gitignore
|       +- UW-logo-16x16.ico
|       +- UW-logo-32x32.ico
|       +- UW-logo.ico
|       +- UW-logo.png
+- VERSION

3 directories, 14 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
    |       +- UW-logo-16x16.ico
    |       +- UW-logo-32x32.ico
    |       +- UW-logo.ico
    |       +- UW-logo.png
    +- README.rst
    +- VERSION

6 directories, 12 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:

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

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

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
```

Populating the Private Configuration Repository

Start creating your local customization repository using the cookiecutter template discussed in the *The dims-new-repo Cookiecutter* 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: "dittrich@u.washington.edu"
  project_name: "Deployment \"devtest\" private configuration"
  project_slug: "private-devtest"
  project_short_description: "Ansible playbooks private content for \"devtest\"␣
↪deployment"
  project_copyright_name: "University of Washington"
```

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
    |   +- .gitignore
    |   +- Makefile
    |   +- source
    |       +- conf.py
    |       +- index.rst
    |       +- introduction.rst
    |       +- license.rst
    |       +- license.txt
    |       +- _static
    |       |   +- .gitignore
    |       +- _templates
    |       |   +- .gitignore
    |       +- UW-logo-16x16.ico
    |       +- UW-logo-32x32.ico
    |       +- UW-logo.ico
    |       +- UW-logo.png
    +- DO_NOT_PUBLISH_THIS_REPO
    +- hooks
    |   +- post-checkout
    |   +- post-merge
    +- Makefile
    +- README.rst
    +- VERSION

    5 directories, 20 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'
'/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'
'/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'
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':
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.sectf.yml:# File: host_vars/blue14.devops.local.yml
blue16.devops.sectf.yml:# File: host_vars/blue16.devops.local.yml
green.devops.sectf.yml:# File: host_vars/green.devops.local.yml
hub.devops.sectf.yml:# File: host_vars/hub.devops.local.yml
node01.devops.sectf.yml:# File: host_vars/node01.devops.local.yml
node02.devops.sectf.yml:# File: host_vars/node02.devops.local.yml
node03.devops.sectf.yml:# File: host_vars/node03.devops.local.yml
orange.devops.sectf.yml:# file: host_vars/orange.devops.local
orange.devops.sectf.yml:jenkins_url_external: 'http://orange.devops.local:8080'
purple.devops.sectf.yml:# File: host_vars/purple.devops.local.yml
red.devops.sectf.yml:# File: host_vars/red.devops.local.yml
vmhost.devops.sectf.yml:  'local': 'vboxnet3'
yellow.devops.sectf.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/sectf/' $F; done
$ cd -
/home/dittrich/dims/git
```

CHAPTER 5

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.

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
```

```
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
```

```
@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
```

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 base/templates/tests
base/templates/tests
+- component
+- helpers.bash.j2
+- integration
```

```
+- README.txt
+- system
|   +- deprecated.bats.j2
|   +- dims-accounts.bats.j2
|   +- dims-accounts-sudo.bats.j2
|   +- dims-base.bats.j2
|   +- dns.bats.j2
|   +- proxy.bats.j2
|   +- sudo
|   |   +- sudo-iptables.bats.j2
|   +- user
|       +- vpn.bats.j2
+- unit
    +- dims-filters.bats.j2

6 directories, 11 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
 |   +- helpers.bash -> /opt/dims/tests.d/helpers.bash
 +- system
 |   +- deprecated.bats
 |   +- dims-accounts.bats
 |   +- dims-accounts-sudo.bats
 |   +- dims-base.bats
 |   +- dims-ci-utils.bats
 |   +- dns.bats
 |   +- helpers.bash -> /opt/dims/tests.d/helpers.bash
 |   +- iptables-sudo.bats
 |   +- proxy.bats
 |   +- user
 |       +- helpers.bash -> /opt/dims/tests.d/helpers.bash
 |       +- vpn.bats
 +- unit
     +- bats-helpers.bats
     +- dims-filters.bats
     +- dims-functions.bats
     +- helpers.bash -> /opt/dims/tests.d/helpers.bash

 5 directories, 18 files
```

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

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

1 directories, 3 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
|   +- helpers.bash -> /opt/dims/tests.d/helpers.bash
+- system
|   +- deprecated.bats
|   +- dims-accounts.bats
|   +- dims-accounts-sudo.bats
|   +- dims-base.bats
|   +- dims-ci-utils.bats
|   +- dns.bats
|   +- docker-consul.bats
|   +- docker-core.bats
|   +- docker-network.bats
|   +- helpers.bash -> /opt/dims/tests.d/helpers.bash
|   +- iptables-sudo.bats
|   +- proxy.bats
|   +- user
|       +- helpers.bash -> /opt/dims/tests.d/helpers.bash
|       +- vpn.bats
+- unit
    +- bats-helpers.bats
    +- dims-filters.bats
    +- dims-functions.bats
    +- helpers.bash -> /opt/dims/tests.d/helpers.bash

5 directories, 21 files
```

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

```
TASK [base : Make links to helper functions present] ***************************
Wednesday 21 December 2016  20:04:15 -0800 (0:00:00.811)       0:02:55.421 ****
ok: [red.devops.local] => (item=[u'component', u'helpers.bash'])
ok: [red.devops.local] => (item=[u'integration', u'helpers.bash'])
ok: [red.devops.local] => (item=[u'system', u'helpers.bash'])
ok: [red.devops.local] => (item=[u'system/user', u'helpers.bash'])
ok: [red.devops.local] => (item=[u'unit', u'helpers.bash'])

msg: All items completed
ok: [red.devops.local] => (item=[u'user', u'helpers.bash'])

TASK [base : Identify bats test templates] *************************************
Wednesday 21 December 2016  20:04:19 -0800 (0:00:04.511)       0:02:59.933 ****
ok: [red.devops.local]

TASK [base : Make defined bats tests present] **********************************
Wednesday 21 December 2016  20:04:19 -0800 (0:00:00.054)       0:02:59.987 ****
changed: [red.devops.local] => (item=./system/dims-base.bats.j2)
changed: [red.devops.local] => (item=./system/dims-accounts-sudo.bats.j2)
changed: [red.devops.local] => (item=./system/dns.bats.j2)
changed: [red.devops.local] => (item=./system/deprecated.bats.j2)
changed: [red.devops.local] => (item=./system/proxy.bats.j2)
changed: [red.devops.local] => (item=./system/dims-accounts.bats.j2)
```

```
changed: [red.devops.local] => (item=./system/iptables-sudo.bats.j2)
changed: [red.devops.local] => (item=./system/user/vpn.bats.j2)
changed: [red.devops.local] => (item=./user/user-account.bats.j2)
changed: [red.devops.local] => (item=./user/user-deprecated.bats.j2)
changed: [red.devops.local] => (item=./unit/bats-helpers.bats.j2)
changed: [red.devops.local] => (item=./unit/dims-filters.bats.j2)
changed: [red.devops.local] => (item=./unit/dims_functions.bats.j2)

msg: All items completed
```

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.

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)
  -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
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
```

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

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"
# 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).

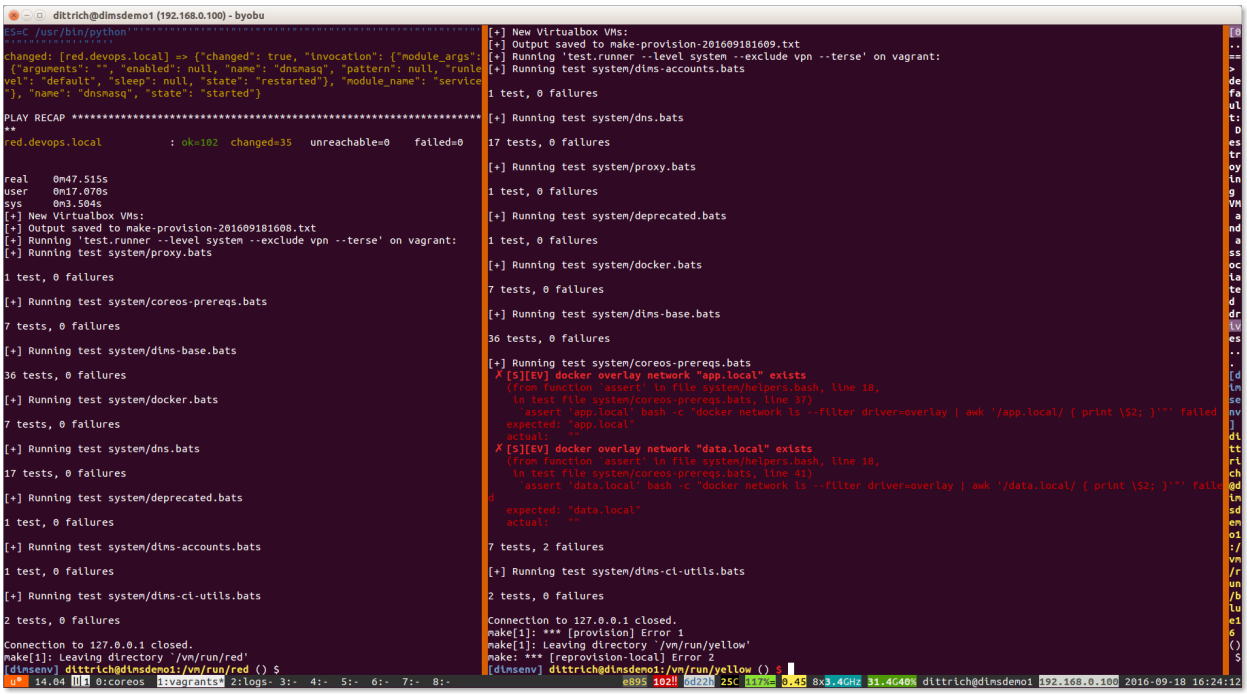

Fig. 5.1: Using `test.runner` in Vagrant Provisioning

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
```

```
load helpers

function setup() {
    source $DIMS/bin/dims_functions.sh
    touch --reference=/bin/ls /tmp/bats.ls-marker
    for name in A B; do
        mkdir -p /tmp/bats.tmp/${name}.dir
        touch /tmp/bats.tmp/${name}.txt
    done
}

function teardown() {
    rm -f /tmp/bats.ls-marker
    rm -rf /tmp/bats.tmp
}

@test "[U][EV] say() strips whitespace properly" {
    assert '[+] unce, tice, fee times a madie...' say 'unce, tice,  fee    times a madie...      '
}

@test "[U][EV] say_raw() does not strip whitespace" {
    assert '[+] unce, tice,  fee   times a madie...      ' say_raw 'unce, tice,  fee  times a madie...      '
}

# This test needs to directly source dims_functions in bash command string because of multi-command structure.
@test "[U][EV] add_on_exit() saves and get_on_exit() returns content properly" {
    assert "'([0]=\"cat /dev/null\")'" bash -c ". $DIMS/bin/dims_functions.sh; touch /tmp/foo; add_on_exit cat /dev/null; get_on_exit"
}

@test "[U][EV] get_hostname() returns hostname" {
    assert "$(hostname)" get_hostname
}

@test "[U][EV] is_fqdn host.category.deployment returns success" {
    is_fqdn host.category.deployment
}

@test "[U][EV] is_fqdn host.subdomain.category.deployment returns success" {
    is_fqdn host.subdomain.category.deployment
}

@test "[U][EV] is_fqdn 12345 returns failure" {
    ! is_fqdn 12345
}

@test "[U][EV] parse_fqdn host.category.deployment returns 'host category deployment'" {
    assert "host category deployment" parse_fqdn host.category.deployment
}

@test "[U][EV] get_deployment_from_fqdn host.category.deployment returns 'deployment'" {
    assert "deployment" get_deployment_from_fqdn host.category.deployment
}
```

```
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
. . .
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,
#  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
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...

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.

CHAPTER 6

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.

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.

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"
}
```

```
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'

# 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"
        }
    ]
}
```

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",
                    "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",
        "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 => {
    "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"
}
$ 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!

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'),
                    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 7

Upgrading and Updating Components

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

Updating PyCharm Community Edition

Now that we have seen an example of setting variables at the host level that override group variables, and validating the values of those variables at run time, we will see how an example of upgrading the application.

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*.

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.

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
```

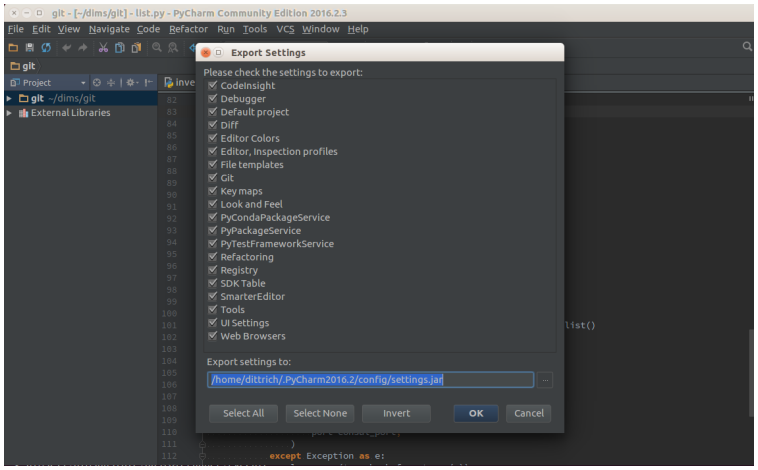

Fig. 7.1: Exporting Settings from PyCharm 2016.2.3

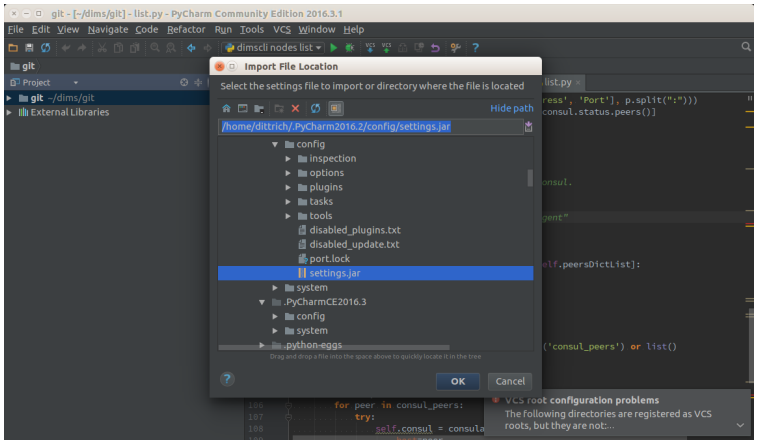

Fig. 7.2: Importing Settings to PyCharm 2016.3.1

Identifying When Rebooting is Needed

A `bats` system test exists to check to see if any packages were installed that require a reboot. This test is templated to tailor it for supported operating systems. Use the `test.runner` script to execute just the `reboot` test:

```
$ test.runner --match reboot
[+] 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-4.4.0-87-generic
   linux-base
   linux-base

1 test, 1 failure
```

CHAPTER 8

License

```
Berkeley Three Clause License
=============================

Copyright (c) 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
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.
```

Section author: Dave Dittrich dittrich@u.washington.edu