DIMS Project Final Report

Release 1.0.1

David Dittrich

Dec 05, 2017

Contents

This document (version 1.0.1) is the Final Report for the *Distributed Incident Management System* (DIMS) project, DHS Contract HSHQDC-13-C-B0013.

CHAPTER 1

Introduction

This document presents the Final Report for the *Distributed Incident Management System* (DIMS) Project.

1.1 Project Overview

The DIMS Project was intended to build upon a long history of DHS sponsored efforts to support security information event sharing among State, Local, Territorial, and Tribal (SLTT) government entities in the Pacific Northwest region known as the Public Regional Information Security Event Management (PRISEM) project. The PI served as the Director of Research and Development for the PRISEM Project and managed its networking and hardware at the University of Washington from the project's inception in 2008.

For more on the background on the relationship between the PRISEM and DIMS projects, see DIMS Operational Concept Description v 2.9.0, DIMS System Requirements v 2.9.0, and DIMS Commercialization and Open Source Licensing Plan v 1.7.0.

The period of performance for the DIMS Project ran from August 16, 2014 to August 15, 2017. Over the period of performance, the people listed in Table *Project Participants* were involved in project:

Table 1.1: Project Participants

Name	Organization(s)	Role(s)
David Dittrich	Applied Physics Laboratory, University of Washington; Center for Data Science, University of Washington Tacoma	Principal Investigator
Linda Parsons	Next Century	Subcontract programmer
Scott Warner	Next Century	Subcontract programmer
Mickey Ross	Ross Consulting Services	Subcontract program management, administrative services
Eliot Lim	Applied Physics Laboratory, University of Washington; Center for Data Science, University of Washington Tacoma	System administration, hardware support
Stuart Maclean	Applied Physics Laboratory, University of Washington	Programmer
Megan Boggess	University of Washington Tacoma; Center for Data Science, University of Washington Tacoma	Graduate Student RA; Programming, system administration.
Jeremy Parks	Center for Data Science, University of Washington Tacoma	Programming, system administration
Jeremy Johnson	Critical Informatics	Subcontract programming, system administration
Katherine Carpenter	Critical Informatics	Subcontract program management, administrative services

1.2 The Value Proposition

This section discusses the *value proposition* for the products of the DIMS project.

1.2.1 The Need

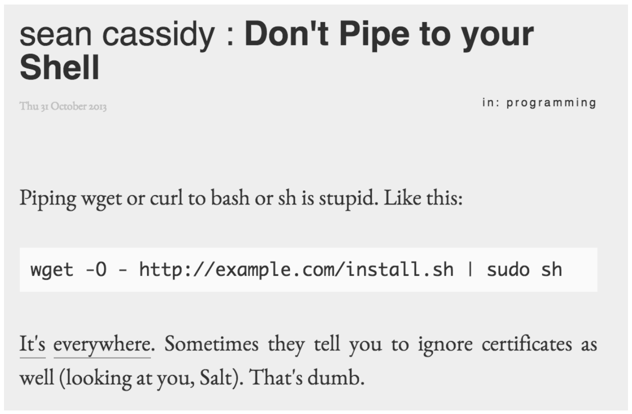
You can't have good system security without good system administration. Organizations need to have strong system administration skills in order to have a secure foundation for their operations. That 1/3 of attacks are due to mistakes and misconfigurations identified in Verizon's DBIR reflects a painful reality. And 100% of those breaches occurred in companies who employ humans.

Of course, all humans make mistakes, or miss things. Or they may not know better when trying to just figure out how to get their job done and blindly follow someone's lead, opening themselves and their organization up to a major security hole (as seen in Fig. 1.1 from Don't Pipe to your Shell).

Mistakes are easier to make in situations where it is difficult to see what is going on, or where someone is forced to deal with something new that they have never dealt with before and have little expertise. Paul Vixie has described the pain (in terms of operations cost and impact on security posture) that results from *complexity* in today's distributed systems and security products. *[Vix16]*

> *Increased complexity without corresponding increases in understanding would be a net loss to a buyer. [. . .]*
>
> *The TCO of new technology products and services, including security-related products and services, should be fudge-factored by at least 3X to account for the cost of reduced understanding. That extra 2X is a source of new spending: on training, on auditing, on staff growth and retention, on in-house integration.*

As knowledge and experience increase, the quality of work output increases and the errors and omissions decrease. Finding and procuring the talent necessary to operate at the highest level, however, is neither easy, fast, nor cheap.


sean cassidy : **Don't Pipe to your Shell**

Thu 31 October 2013

in: programming

Piping wget or curl to bash or sh is stupid. Like this:

```
wget -O - http://example.com/install.sh | sudo sh
```

It's everywhere. Sometimes they tell you to ignore certificates as well (looking at you, Salt). That's dumb.

Fig. 1.1: Piping insecure content directly into a privileged shell

This all raises the question, "What can our organization do bring the capabilities of multiple open source products into a functioning whole with the least amount of pain and best operating security outcome?"

1.2.2 Our Approach

Our approach is to provide a reference model for establishing a secure and maintainable distributed open source platform that enables secure software development and secure system operations. The DIMS team (now implementing the third iteration of some of the core elements) has experienced the pain of this process, which will reduce the cost for those who adopt our methodology.

The DIMS project brings together multiple free/libre open source software (FOSS) tools in a reference model designed to be built securely from the ground up. The two primary outcomes of this effort are:

1. An example platform for building a complex integrated open source system for computer security incident response released as open source software and documentation. These products provide a working and documented model platform (or DevOps infrastructure) that can facilitate the secure integration of open source components that (in and of themselves) are often hard to deploy, and often are so insecurely implemented that they are effectively wide open to the internet. This not only solves some of the infrastructure problems alluded to by the Linux Foundation, but also addresses Vixie's example of supporting organizations wanting to use open source security tools in concert to address their trusted information sharing and security operations needs.
2. Transitioning this platform into the public sector to support operational needs of SLTT government entities. DIMS project software products were included in a draft proposal for evaluation by the PISCES-NW not-for-profit organization for use in the Pacific Northwest. The last modification to the DIMS contract includes a pilot deployment for use by the United States Secret Service for their Electronic Crimes Task Force (ECTF) membership.

The DIMS System Requirements v 2.9.0 documents security practices and features that we have incorporated to the greatest extent possible, in a way that can be improved over time in a modular manner. The system automation and continuous integration/continuous deployment (CI/CD) features help in implementing and maintaining a secure system. (Red team application penetration testing will further improve the security of the system through feedback about weaknesses and deficiencies that crept in during development and deployment.)

Golden nugget

Over two decades of system administration and security operations experience underlies the architectural model that we have been researching, developing, implementing, and documenting. The barrier to entry is the amount of time and learning necessary to acquire this same expertise in order to be competitive.

1.2.3 Benefits per Cost

The value of the DIMS products and methodology comes from altering the cost equation described by Vixie, which can be expressed this way:

$$CustomerValue = \frac{CustomerBenefit}{cost(OpenSource) + cost(Implementation)}$$

The benefit to customers is maximized by the ability to construct and operate a secure incident response monitoring platform, expand it with additional open source tools as needed, saving a large part of the *2x* multiplier in implementation cost in system administration and operations overhead cited by Vixie. We enable this by helping make a less complex, more transparent, source controlled, and easier to secure open source platform than may otherwise be produced by someone leveraging multiple unfamiliar open source security tools from scratch. That means standing up a new server and adding new services so it can be reduced from taking hours or days per system to just a few minutes of effort. If that task has to be repeated dozens (or possibly hundreds) of times, the cost savings can be significant.

The DIMS team created and used a CI/CD model using Git, Jenkins CI, and Ansible for taking software source code, system automation instructions, software configuration, and documentation, to build a prototype for an open source software integration project. The resulting product can be used by an internal security operations group (or managed security service provider) to create an open source incident response capability. It also provides many of the elements called for in the CII Badge Program from the GitHub Security and Heroku Security policies.

Note: To see more detail about the full set of tools, techniques, and tasks that DIMS team members were expected to know or learn, see DIMS Job Descriptions v 2.9.1.

The impact of the effort expended in this project goes beyond implementing one set of open source service components for a single group. This model can be replicated widely and improved upon by others faced with the same set of challenges in developing an affordable and scalable incident response capability.

Note: Over the course of the project, we have learned of several other efforts to address a similar set of goals and have reached out (as time permitted) to find common ground and try to develop collaborative relationships that will have broad impact over time. This is expanded upon in Section Commercialization Plan.

1.2.4 Competition and Alternatives

The common way that organizations go about implementing open source products is by following whatever installation instructions may be provided by the authors. Avoiding the security problems illustrated by Fig. 1.1 involves searching the Internet to (hopefully) find some thread like Alternatives to piping the install script into your shell. #90 (from GitHub fisherman/fisherman, a "plugin manager for Fish," and no, we haven't heard of it before either.)

When it comes to the more difficult task of integrating multiple open source products into a functional distributed system, the research required to debug and solve a seemingly endless series of installation, configuration, and tuning problems.

1.2.5 Open Source Security Toolsets

Some of the open source security tools that an incident response team would want to consider implementing are covered in the following subsections.

Each of these systems is composed from several existing open source tools, combined with new open source scaffolding, glue, custom interfaces, and additional missing functionality that is necessary to achieve the resulting distributed system.

At the same time, each of these distributed open source systems relies upon their own chosen base operating system, libraries and languages, subordinate services (e.g., database, email transport agent, message bus, job scheduling, etc.) All too frequently, the choices made by each group are mutually exclusive, or left to the customer to work out on their own.

Note: To underscore Vixie's complexity and cost of implementation observation, Ubuntu 14.04 and Debian 7 have differences in how common services are configured that require debugging and custom configuration steps that vary between distributions, while the use of `systemd` for managing service daemons in Ubuntu 16.04 and Debian 8 are major impediments to migrating installation of all required components of these multi-service systems from Ubuntu 14.04 and Debian 7. Adding in RedHat Enterprise Linux, CentOS, or Fedora (all part of the same RedHat family) adds further complexity to the equation, which is a major reason why containerization is gaining popularity as a mechanism for isolating these dependency differences in a more manageable (but arguably less secure) fashion.

The Trident portal

The Trident portal is written in Go. Only Debian 7 (wheezy) is supported at this time, though Ubuntu 14.04 is on the list of future operating systems. Trident relies on PostreSQL for database, NGINX for web front end, and Postfix for email transport.

The Collective Intelligence Framework (CIF)

The Collective Intelligence Framework (CIF) is the primary offering from the CSIRT Gadgets Foundation. CIF is only supported on Ubuntu Linux. It is written in Perl and uses PostgreSQL, Apache2, BIND, Elasticsearch, ZeroMQ, and can support Kibana as an alternative interface to the indexed data in Elasticsearch.

A monolithic *EasyButton* installation script is available in the PlatformUbuntu section of the CIF wiki to automate the installation steps.

The Mozilla Defense Platform (MozDef)

The Mozilla Defense Platform (MozDef) was developed by Mozilla to replace a commercial SIEM product with open source alternatives. They report processing over 300 Million records per day with their internal deployment.

MozDef uses Ubuntu 14.04 as the base operating system. It has components for front-end user interface written in Javascript using Meteor, Node.js, and d3, and back-end data processing scripts written in Python using uWSGI, bottle.py, with MongoDB for a database, RabbitMQ for message bus, and NGINX for web app front end.

For installation, there is a demonstration `Dockerfile` for creating a monolithic Docker image with all of the MozDef components in it. (This is not the way Docker containers are intended to implement scalable microservices, but it does provide a very easy way to see a demonstration instance of MozDef). The manual instructions are more elaborate and must be followed carefully (including considering the admonitions related to security, e.g., "Configure your security group to open the ports you need. Keep in mind that it's probably a bad idea to have a public facing elasticsearch.")

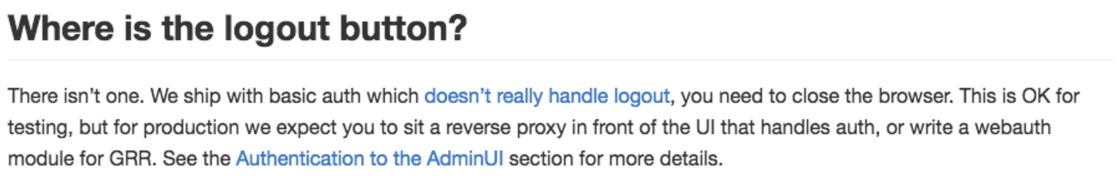
GRR Rapid Response

Another example of a system made up of multiple components, packaged together into a single easy-to-install package, is the GRR Rapid Response system, a "forensic framework focused on scalability enabling powerful analysis."

GRR runs on Ubuntu 16.04. To ease installation of the server components, the GRR team, like CIF and MozDef, provide both a monolithic installation script for a VM installation and a `Dockerfile` to run in a container. They also have packages for installing the client components on Windows, OS X, and Linux.

Attention: The GRR team chose to move to `systemd`, rather than continue to support the older `upstart`, `init.d`, or `supervisord` service daemon systems that are used by other products described in this section. This means you must support three (or four) different service daemon management mechanisms in order to incorporate all of the tools described here into a single integrated deployment.

GRR's documentation similarly includes admonitions about security and functionality that is left to the customer to implement. Take Fig. 1.2, a question from their FAQ as an example:



Where is the logout button?

There isn't one. We ship with basic auth which doesn't really handle logout, you need to close the browser. This is OK for testing, but for production we expect you to sit a reverse proxy in front of the UI that handles auth, or write a webauth module for GRR. See the Authentication to the AdminUI section for more details.

Fig. 1.2: Question about the logout button from GRR FAQ

1.2.6 Integrated Open Source Solutions

The DIMS project began in Q4 2013. In the second half of 2015 two very similar efforts were identified that use some of the same tools for the same reasons. Both validate the model being established by DIMS and the value proposition for adopters.

Summit Route Iterative Defense Architecture

An organization named Summit Route has described what they call the Iterative Defense Architecture (see Fig. 1.3) that is very similar in form and content to what the DIMS project has focused on producing.

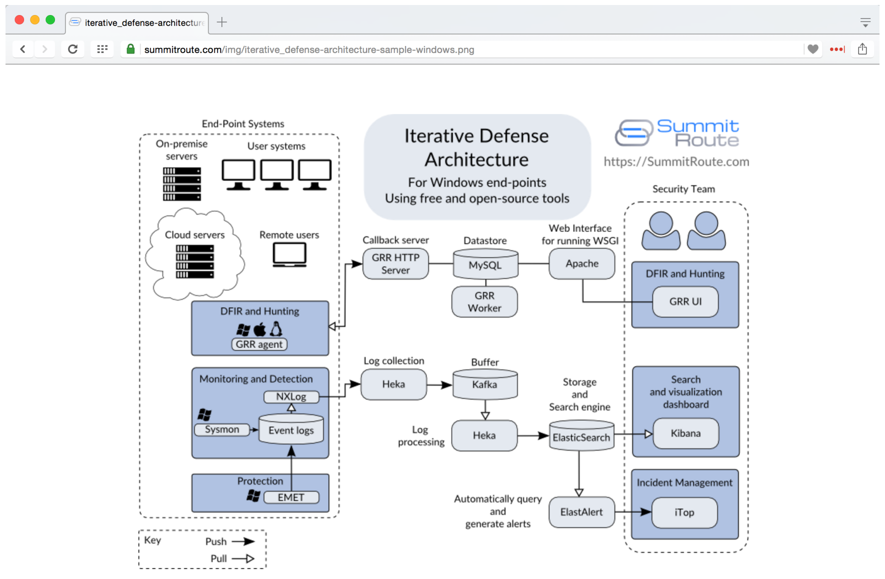


Fig. 1.3: Summit Route Integrated Defense Architecture

OpenCredo

A consultancy in the United Kingdom named *OpenCredo* is also working on a similar architecture to the DIMS project (see Fig. 1.4). Some of the specific components differ, but conceptually are the same and would meet the same requirements for the foundation (minus the dashboard, portal, etc.) that is specified in DIMS System Requirements v 2.9.0.

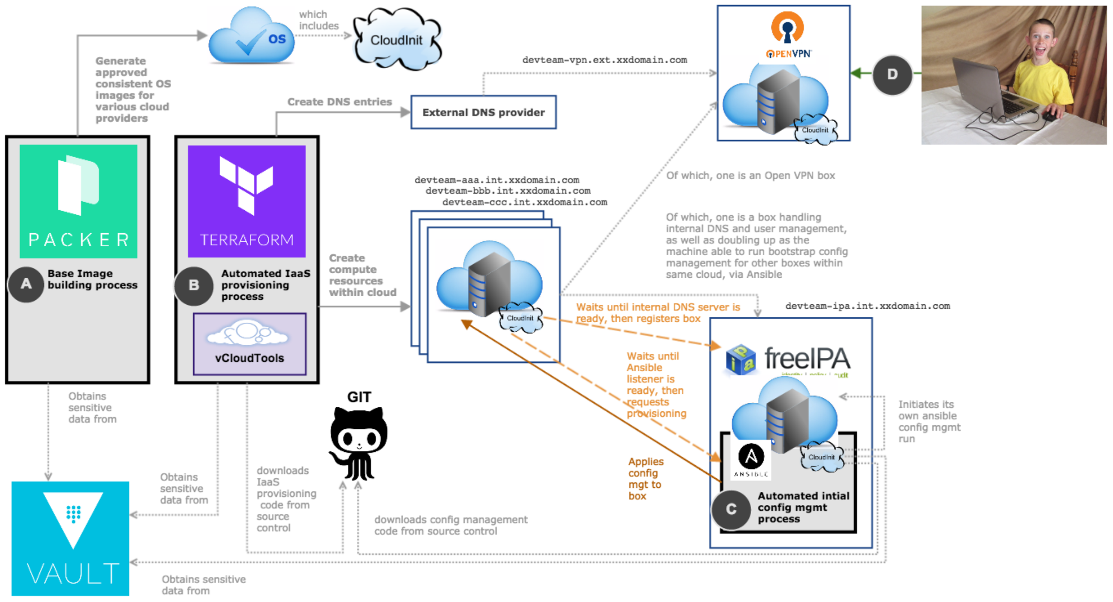

which includes
OS
CloudInit
Generate approved consistent OS images for various cloud providers
Create DNS entries
External DNS provider
devteam-vpn.ext.xxdomain.com
PACKER
A
Base Image building process
TERRAFORM
B
Automated IaaS provisioning process
vCloudTools
Create compute resources within cloud
devteam-aaa.int.xxdomain.com
devteam-bbb.int.xxdomain.com
devteam-ccc.int.xxdomain.com
CloudInit
OPENVPN
D
Of which, one is an Open VPN box
Of which, one is a box handling internal DNS and user management, as well as doubling up as the machine able to run bootstrap config management for other boxes within same cloud, via Ansible
Waits until internal DNS server is ready, then registers box
devteam-ipa.int.xxdomain.com
freeIPA
Waits until Ansible listener is ready, then requests provisioning
Initiates its own ansible config mgmt run
Applies config mgt to box
ANSIBLE
C
Automated intial config mgmt process
GIT
Obtains sensitive data from
Obtains sensitive data from
downloads IaaS provisioning code from source control
downloads config management code from source control
VAULT
Obtains sensitive data from

Fig. 1.4: OpenCredo core building blocks

The remainder of this report is divided into the following sections:

- Section *Referenced Documents* summarizes referenced documents (with links to those available online for convenience).
- Section *Outcomes* covers the value, expected outcomes, impacts, products, problems to be solved by, and benefits of this project.
- Section *Challenges Encountered* covers some of the technical challenges that were encountered over the course of the project.
- Section *Needed Enhancements* discusses needed enhancements and directions that follow-on projects could take, building from the state of released code, configuration, and documentation products.
- Section *Recommendations for Follow-on Projects* includes recommendations by the PI for consideration in planning follow-on projects, whether they use DIMS products or not, intended to help reduce friction in the software development process.
- Section *License* includes the open source software license under which DIMS products are to be released.

Note: Some of the content of this report comes from other previously delivered project documents, or references found in *working* documents and/or the PI's (Dave Dittrich (@davedittrich)) home page (which served as a general project reference on a number of topics).

CHAPTER 2

Referenced Documents

1. HSHQDC-13-C-B0013, “From Local to Gobal Awareness: A Distributed Incident Management System,” Draft contract, Section C - Statement of Work
2. DIMS Ansible playbooks v 2.13.1
3. DIMS Operational Concept Description v 2.9.0
4. DIMS System Requirements v 2.9.0
5. DIMS Job Descriptions v 2.9.1
6. DIMS Commercialization and Open Source Licensing Plan v 1.7.0
7. https://staff.washington.edu/dittrich/home/

CHAPTER 3

Outcomes

3.1 Summary of Project Outcomes

As described in DIMS Operational Concept Description v 2.9.0, the DIMS project started out with two primary expected outcomes: (1) an example platform for building a complex integrated open source system for computer security incident response, and (2) to transition this platform into the public sector to support operational needs of SLTT government entities. The latest modification to the contract includes a pilot deployment for use by the United States Secret Service in addition to the open source release of source code and documentation.

This project successfully implemented a prototype for a deployable open source distributed system. As described in this section and Section *Introduction*, other projects have contemporaneously pursued a similar goal of producing a generally usable system comprised of open source components. The DIMS project includes some features not found in these other projects (e.g., the integrated `bats` tests, the breadth and depth of documentation, the features to support managing multiple simultaneous deployments with private configuration).

The outcome of the DIMS Project is by no means a production-ready commercially marketable system, but the open source products are competitive in many aspects with other projects created by larger teams of software engineering professionals at commercial software companies. Further refinement within an entity staffed and focused on bringing a product or service to the market could quickly get there, but what is public now is ahead of many of the open source code examples that one can find by searching the internet, and the list of resources in Section *Reference Links* far exceeds any similar collection that could be identified by the DIMS team. Integration of DIMS features with those of the other projects described herein to produce a full-featured and production capable system would be the ideal, though the source of funding for such an effort is unclear.

What follows are sections identifying some of the key high level achievements.

3.2 Ansible Playbooks

The most significant achievement of this project was the production and refinement of a set of Ansible playbooks, roles, and task files, capable of instantiating a small-scale distributed system comprised of Ubuntu 14, Ubuntu 16, Debian 8, and Container Linux by CoreOS systems, implementing an `etcd`, `consul`, and Docker Swarm cluster.

These playbooks share many similarities with those of some other publicly available projects that were developed contemporaneously with the DIMS Project, including the OpenCredo and Summit Route Iterative Defense Architecture projects mentioned in Section *Integrated Open Source Solutions*, as well as the Fedora Project's Ansible playbooks, Intel Corporation's Trusted Analytics Platform, and DebOps.

Note: The PI reached out to each of these groups listed (with the exception of the Debops group) and two other projects using Ansible for multi-system deployment, to see if there was any possibility to collaborate or take development of DIMS Ansible playbooks. The majority of these inquiries resulted in no response at all (despite multiple attempts in some cases.) One outreach resulted in a conversation that was a dead end, and another response suggested the chances of potential funding were low. Only one person engaged in multiple follow-on conversations, but no funding opportunities to support a collaboration could be identified.

Two other projects were identified in the final days of the DIMS Project while investigating options for using hashicorp/terraform and Digital Ocean (Cisco Systems's Mantl and Mesosphere's DC/OS). These projects appear to have significantly larger and more well-resourced development and marketing teams. The fact that so many similar projects exist does confirm the viability of the direction taken in the DIMS Project.

The Ansible playbooks created by the DIMS project differ from each of these other projects in several ways. One primary difference is the separation of customization and configuration from the playbooks repository to facilitate continued development and integration of new tools capable of being managed independently of the public `ansible-dims-playbooks` repository.

These playbooks include the following features:

- Support for Ubuntu 14.04, Ubuntu 16.04, Debian 8, Container Linux by CoreOS, and partial support for Mac OS X (Darwin) and RedHat Enterprise Linux.
- Installation and pre-configuration of Trident trust group management portal.
- Integrated multi-level testing using `bats`.
- Support for official SSL certificates using Letsencrypt and self-signed SSL certificates using https://github.com/uw-dims/ansible-role-ca
- Support for automated backup and restoration of Trident PostgreSQL database and Letsencrypt certificates.
- Support for version pinning of core subsystems across development and production hosts for improved distributed system stability.
- Support for automated system-wide checks for availability of package updates, application of updates, and detection of required reboot, with option for email notification.
- Support for isolated Vagrant Virtualbox virtual machines (including local copies of Ansible playbooks for testing branches and improved distributed system stability). This includes automated VM suspension upon host shutdown using, and multi-VM resumption after, using the `dims.shutdown` script.

3.3 Trident Portal

The project began using the original Perl-based *ops-trust* portal system using a hosted portal that was pre-configured. An initial Ansible playbook to deploy a local instance was produced, but the team continued to use the hosted server. In the final year of the project, Ansible playbook support for the new Trident portal (re-written in Go, with both a command line and web application graphical interface) was finally added and the ability to replicate the Trident portal was achieved. Features to support customization of the portal's graphical elements (banner image, default icon image for users who have not loaded their own photo, logo image, and CSS style sheet settings for font and web page colors) were added to support custom branding.

As mentioned in the previous section, along with the playbook for installing Trident the ability to backup and restore both the Trident database and the Letsencrypt SSL/TLS certificates was added. This allows easier development, testing, and training with the Trident portal by simplifying deployment of two portal servers at once (one for dev/test/training and the other for "production" use.) Combined with the re-written Jenkins build scripts, an improved mechanism for debugging and development of new Trident features is now possible. (Testing of these features with volunteers associated with the Trident portal in use by the ops-trust community is being discussed and will continue as an independent project after this project's end date.)

3.4 Pilot Deployment

A deployment of the https://github.com/uw-dims/ansible-dims-playbooks code on a stand-alone baremetal server hosting two virtual machines running instances of the Trident portal, customized and branded specifically for the U.S. Secret Service Electronic Crimes Task Force (ECTF) following Customizing a Private Deployment, was produced for use in a pilot project. Included are a Training Manual (https://trident-training-manual.readthedocs.io) and User Manual (https://trident-user-manual.readthedocs.io) focused on the Trident portal.

3.5 Continuous Integration/Continuous Deployment

Very early on, the project team established a set of Git source repositories that were focused on discrete component services or functionality. Splitting things up into discrete and focused repositories was done to establish a model of modularity (to help make it easier to add new open source tools over time) and to allow independent open source release of repositories. In all, over 40 discrete repositories were created (some now deprecated, but the majority providing functioning components addressing all of the requirements listed in the contract and detailed in the DIMS System Requirements v 2.9.0 document).

Next, a Jenkins CI server was set up and tied to the Git repositories using Git post-commit hooks that trigger *build* jobs for source code and documentation. Some build jobs then, in turn, trigger *deploy* jobs that push the built products onto the systems that use them (see Continuous Integration for more detail on this process).

Throughout this entire workflow, log entries are generated (using a program `logmon`) that publishes them on an AMQP channel where they can be monitored from the DIMS Dashboard, monitored from a terminal session using the same `logmon` program, or collected from the logging channel for indexed storage.

3.6 Install and Build Automation

System administrators are familiar with the steps of setting up a computer systems, be it a server or a desktop development workstation, by starting with an operating system installation ISO image, creating a bootable CD-ROM or USB drive, creating accounts for the system administrator and some users, selecting additional packages to install, and finally installing third-party open source tools as needed.

This is a relatively simple process, and works well if the number of servers and workstations is small, if the number of project members is small (and turnover in staff is low and the team does not grow), if the software being developed is limited in size and scope, and if things don't change very quickly. Developers can even set up their own workstations and manage them.

3.7 Integrated Tests

One of the requirements of the project was testing and validation of the system components. A great deal of effort was spent in writing comprehensive test plans and in performing two system-wide tests. After the experience of doing these test plans and tests, a decision was made to integrate the simplest set of tests as possible into the normal operation of the system. The Bats: Bash Automated Testing System was chosen for its simplicity. A structured mechanism for embedding tests into Ansible Playbook roles was developed, along with a script to facilitate running tests named (not surprisingly) `test.runner`. This testing methodology is described in Section Testing System Components of DIMS Ansible playbooks v 2.13.1.

Listing 3.1: Successful test run from command line

```
$ 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
```

Listing 3.2: Failed unit test in Ansible playbook

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

3.8 Python Virtualenv Encapsulation

A frequently experienced point of friction within the team had to do with differences in the tools being used by developers. One team member has `git` version `2.1` and the other has version `1.8` and can't access the repo the night before a deadline. One person has the `hub-flow` tools and the other does not, but they also don't know how to merge and push branches so their code is not available to the team. Someone installs a broken version of an internal tool and doesn't realize it when they try to test someone else's commits, so their test fails when it should succeed and nobody knows why it is happening.

As a means of isolating and encapsulating a Python based shell environment to facilitate development, testing, working on branches, and generally experimenting in a non-destructive manner, the use of a standardized Python virtual environment called `dimsenv` was implemented. This is a little heavier-weight use of the Python `virtualenv` mechanism, encapsulating more than just Python interpreter and `pip` installed packages.

The `python-virtualenv` role builds a specific version of Python, installs a specific set of version-pinned `pip` packages, and also adds a series of programs to the `bin/` directory so as to ensure the full set of commands that have been documented in the dimsdevguide are available and at the same revision level.

This not only saves time in setting up a development environment, but makes it more consistent across systems and between development team members. Things like testing new versions of Ansible is trivial. You just clone the `dimsenv` environment (which has all the development tools in it already), use `workon` to enable the new virtual environment, and `pip install ansible==$DESIRED_VERSION`. Then run the playbooks you want to test. It is easy to switch back and forth, allowing development and debugging of playbooks to be able to migrate to the latest version of Ansible more easily, while still being able to fall back to the standard to get back to a stable build environment. While this is an unconventional use of Python `virtualenv`, it works pretty well and saves lots of time.

3.9 DIMS Dashboard

A functional dashboard web application was developed using distributed system features provided by several VM compute servers over AMQP, with single-signon tied to Google authentication. This dashboard supported user stories defined in the dimssr with built-in test capabilities. This was the most production-ready and well-engineered components of the system.

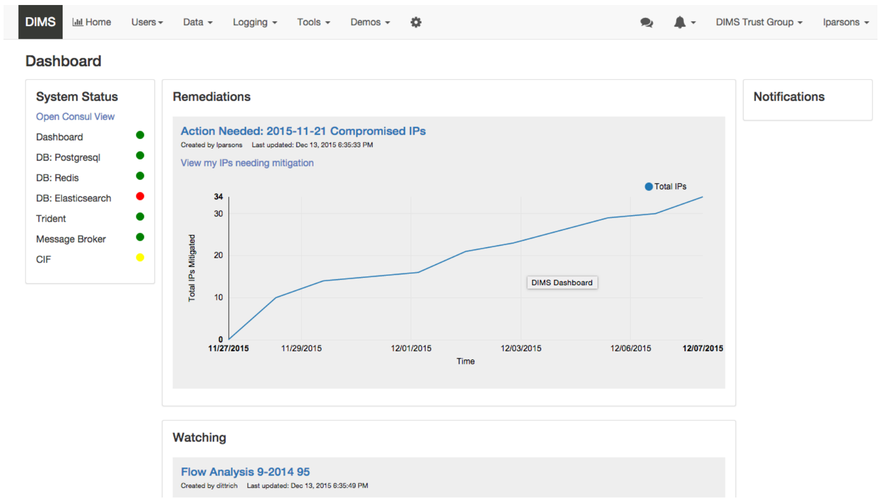


Fig. 3.1: DIMS Dashboard

3.10 Ingest of STIX Documents

Java bindings for STIX were produced to facilitate ingest of STIX version 1.1 documents into the DIMS system. (The current release of STIX is now version 2.0.)

3.11 Tupelo and Related Host Forensic Tools

A Java client/server application for manipulation of host file system disk images and related metadata named "Tupelo" was produced as part of an earlier National Science Foundation grant funded project. It was enhanced with inclusion of libraries for access TSK tools and manipulating virtual machine disk images, and integrated into the early DIMS development deployment.

3.12 Software Products and Documentation

The following table provides links to public source code repositories and documentation.

Table 3.1: Software Products and Documentation

Source repository	Documenation
https://github.com/uw-dims/ansible-dims-playbooks	https://ansible-dims-playbooks.readthedocs.io
https://github.com/uw-dims/device-files	(No additional documentation)
https://github.com/uw-dims/dims-ad	https://dims-ad.readthedocs.io
https://github.com/uw-dims/dims-adminguide	https://dims-adminguide.readthedocs.io
https://github.com/uw-dims/dims-dashboard	https://dims-dashboard.readthedocs.io
https://github.com/uw-dims/dims-devguide	https://dims-devguide.readthedocs.io
https://github.com/uw-dims/dims-jds	https://dims-jds.readthedocs.io
https://github.com/uw-dims/dims-ocd/	https://dims-ocd.readthedocs.io
https://github.com/uw-dims/dims-sr/	https://dims-sr.readthedocs.io
https://github.com/uw-dims/dims-swplan	https://dims-swplan.readthedocs.io
https://github.com/uw-dims/dims-training-manual	https://dims-training-manual.readthedocs.io
https://github.com/uw-dims/dims-tp/	https://dims-tp.readthedocs.io
https://github.com/uw-dims/dims-user-manual	https://dims-user-manual.readthedocs.io
https://github.com/uw-dims/fuse4j	(No additional documentation)
https://github.com/uw-dims/java-native-loader	(No additional documentation)
https://github.com/uw-dims/stix-java	(No additional documentation)
https://github.com/uw-dims/trident-training-manual	https://trident-training-manual.readthedocs.io
https://github.com/uw-dims/trident-user-manual	https://trident-user-manual.readthedocs.io
https://github.com/uw-dims/tsk4j	(No additional documentation)
https://github.com/uw-dims/tupelo	(No additional documentation)
https://github.com/uw-dims/xsdwalker	(No additional documentation)

CHAPTER 4

Challenges Encountered

This section describes many of the challenges that were encountered during the project's period of performance. Some where eventually overcome in the final months of the project, though others were not. Suggestions on dealing with some of these issues are found in Section *Needed Enhancements* and Section *Recommendations for Follow-on Projects*.

4.1 Understanding the Tools Being Used

It is a myth that humans only use 10% of their brain.[1] But it is common for programmers and system administrators to only learn a small portion of the features and capabilities of any given program or tool that they may need to use. The simple thing to do is to search StackExchange for system administration tasks, or StackOverflow for programming tasks, and simply copy/paste what someone posts there in order to quickly "solve" the problem at hand and move on to the next thing.

Taking such short-cuts, trying to avoid the investment of time required to learn the capabilities of a given tool can result in problems later on. Scripts may be written that restrict or limit the utility of the underlying programs they call, or that perform extra work that could be done in a more straight-forward or idiomatic way using advanced features of the underlying programs. At the very minimum, everyone sharing use of a common tool must have a sufficient baseline familiarization with the content of available documentation by skimming through it completely to be able to quickly get past a blocker.

When someone is tasked with solving a particular problem, given a set of requirements or use cases that should be satisfied, it is important that they take responsibility for studying the tool(s) being used to understand how best to perform the task at hand and to share their knowledge with other team members using the same tool. It does not work for the project lead to have to become the expert in every tool and micro-manage how team members do their work.

This problem was exacerbated in the DIMS Project due to the large number of new and rapidly changing tools and technologies that are necessary to assemble a system with the complexity defined in the project scope of work. Certain project team members had experience in specific programming languages, limiting their ability to contribute in some situations. Most had some familiarity with Unix system administration on their own development workstations, but were not able to branch out to unfamiliar Linux distributions. During this project every team member was pushed to

[1] All You Need to Know About the 10 Percent Brain Myth, in 60 Seconds, by Christian Jarrett, July 24, 2014.

learn new programming languages, new data storage mechanisms, new inter-process communication protocols, new software development tools and processes, new types of security event data, and new concepts in network and host forensics required to process security event data, threat intelligence streams, and malware artifact metadata.

4.2 Staffing Challenges

The document DIMS Job Descriptions v 2.9.1 was produced to list the full set of skills and experience required by those working on a project of this nature. As mentioned in the previous section, every member of the team was pushed beyond their technical limits and had to constantly research and learn new technologies and rapidly acquire new software engineering, network engineering, or system administration skills. Not everyone is capable of, or happy with, being pushed beyond their limits on a daily basis and some turnover within the project was directly related to this pressure to deliver.

Over the course of the project, the team size was typically 3-5 people, with most of the team working at less than 100% FTE. One contractor was 100% FTE for the majority of the project and had to step in to perform tasks that were not being performed by other team members.

The team was also partially virtual, with one, two, and sometimes three staff members working on the East Coast, while the rest of the team was on the West Coast (mostly in Seattle, WA, but at one point split between Seattle, Tacoma, and Bremerton, WA and one person in California.) Regular "scrum" meetings were held using online tools (at various times Adobe Connect, Skype, and Google Hangout were all used, to varying degrees of effectiveness or frustration.) This made the problem of trying to bring team members up to speed on new concepts and skills difficult, due to lack of physical presence and availability.

Another difficulty resulted from political issues as opposed to strictly technical issues. In order to meet the objectives of the contract, the team was being pushed far beyond their capabilities. Some people respond to this by putting in the extra time it takes to improve their skill set (either inside working hours, or seeing it as an investment in their professional career, doing some extra-curricular learning.) Others respond by pushing back, focusing their efforts on only those tasks they are comfortable with and no more, or otherwise not following the established development path. Some documentation was not produced as requested (in some cases the PI was able to make up for the deficit, but this was not possible when the PI did not write the software.)

One risk to a project that is hard to avoid is a dependency on external software products or standards that are outside of the control of the project (e.g., see *STIX Development Libraries*). Such situations can cause larger organizational weaknesses and personnel issues to surface that simply cannot be solved without commitment and full-throated support from higher up in the organization. The best that can sometimes be achieved is to learn from the situation, find a way to move on, and carry the lessons forward to do better in the future.

4.3 DNS Challenges

4.3.1 Naming Computers

Naming computers is not easy. One of the tenets of secure design is *separation of services*, which historically has driven system administrators to limit services to one service per computer. You have a DNS server that does DNS, a database server for data storage, a web server that provides HTTP/HTTPS web application services for browsers, an FTP file server that only serves static files, etc.

In such simple deployments, naming a computer based on the service it provides seems to make sense and to be simple. Until you decide it makes more sense to combine some related services on that one computer, at which point one of two things happens:

1. The computer's name now only matches one of the two services, and it becomes harder to know what computer name to use when trying to connect to the second service. ("The Git repos are on `git`, and Jira is on `jira`. We put Jenkins on one of those two servers, but was it on `git` or on `jira`?).
2. The service is put on another computer (possibly a virtual machine) and the computer's name now matches the service. But now there is also another computer host to manage, with `iptables` rules, accounts and passwords allowing administrator access, the need to copy in SSH keys, etc. As more computers are added, management and use gets harder and harder.

Part of this problem is handled by adopting a policy of *not* naming computers after services, but instead using more generic host names (like colors like `red` and `orange`, or generic names like `node01` through `node09`). Those host names are then mapped with DNS *A* records (and associated *PTR* records to properly reverse-map the IP to name) and using CNAME entries that create aliases in DNS name space, allowing URLs to be formed with the service name as part of the DNS name. (E.g., `trident.devops.local` may map to `yellow.devops.local` via a CNAME).

The drawback to this is that the administration of A records, PTR records, and CNAMES is more difficult than simple `/etc/hosts` entries, and requires a deeper understanding of DNS internals by all involved. The final implementation of DIMS Ansible playbooks generates DNS host name mappings using Jinja templating to generalize creating DNS entries.

Another problem that must be dealt with when placing multiple services on the same system is TCP port mappings. You can only have one service listening to port `80/tcp`, port `443/tcp`, etc. That requires that services like Trident, a web application service, etc., all have their own unique high-numbered service ports (e.g., `8080/tcp` for Trident, `8000/tcp` for the web application service, `8500/tcp` for Consul's UI, etc.) But now how do you remember which port to use to get to which service on which host? Adopting a prefix with the service's name and using a CNAME that aliases the host allows an easier to remember mechanism to reach services, though at the cost of complexity in NGINX reverse proxy configuration. You can now access Trident using `https://trident.devops.local/trident` and `https://consul.devops.local/consul` to get to the Consul UI. What is more, using multiple DNS records for each Consul node in a cluster allows for round-robin access to distribute the connections across cluster nodes:

```
$ dig consul.devops.local +short
192.168.56.23
192.168.56.21
192.168.56.22
```

4.3.2 Separating DNS Name Spaces

Adding to the complexity of DNS and host naming is the situation of multi-homed hosts. Most people are accustomed to one computer with one or two interfaces (like a laptop with either a wired Ethernet interface, or a WiFi interface, only one of which is active at any given time). That means the computer always has just one active IP address, and since laptops are usually used for connecting as a client to remote services, they don't even need to have a DNS name!

Layered, segmented networks that involve external firewalling, Virtual Private Network (VPN) access to multi-segmented Virtual Local Area Network (VLAN) switched or virtual machine network environments cause problems when it comes to host naming and DNS naming.

The early implementation of DIMS DNS used a single DNS namespace, with multiple names per host that were arbitrarily chosen with some hosts having four or more names using A records, some in the `prisem.washington.edu` namespace, even though they only existed in the internal DNS server and not in the external authoritative name servers.

For example, a DNS name like `jira.prisem.washington.edu` would exist in the internal server, mapping to an IP address in the `140.142.29.0/14` network block. Doing `dig @128.95.120.1 jira.prisem.washington.edu` (an official UW name server) or `dig @8.8.8.8 jira.prisem.washington.edu` (one of Google's name servers) would fail to get an IP address, but making the request of the internal server would work. Since Jira was running behind a reverse proxy, however, the host that was actually running the Jira server was not the one using the address on the `140.142.29.0/24` network block, so a second DNS name `jira-int.prisem.washington.edu` (also non-existent externally) would map to the internal IP address, which was only accessible over a VPN. This resulted in a huge amount of confusion. Which host was actually running Jira? What port? What order for DNS servers has to exist to ensure the request goes to the internal DNS server first, not the external DNS servers that don't know the answer?

The proper way that multi-homed network namespace management is handled is through the use of *Split horizon* (or *split-brain*) DNS. This requires multiple DNS servers, multiple DNS zones, and careful mapping of the IP addresses and DNS names for each of the zones, as necessary to route packets properly through the correct interface. Again, this requires a much deeper understanding of DNS than is common.

4.3.3 Handling Dynamic Addressing on Mobile Devices

Yet one more issue that complicates connectivity is the use of mobile devices like laptops, which must use a VPN to connect to access-controlled hosts behind firewalls. If split-horizon DNS is used, with one DNS server behind the VPN such that it is only accessible when the VPN is connected, the mobile device may experience significant delays in DNS requests that cannot be sent to the unavailable DNS server. This requires complicated dynamic DNS resolver configuration that is difficult to set up and to debug without expertise in advanced network configuration on the operating system being used (in this case, Mac OS X and Ubuntu Linux were the two predominant operating systems on laptops.)

One of the ramifications of mobile devices using Ubuntu Linux is the role of `NetworkManager`, a notoriously problematic service in terms of network configuration management. It is very difficult to take control of services like `dnsmasq` for split-horizon DNS, or use VPNs (especially multiple VPNs, as was implemented in this project from the start), without running into conflicts with `NetworkManager`.

The DIMS project started using the Consul service as a means of registering the IP address of a client using a VPN, such that the current address and accessibility status is available using Consul's DNS service. As Consul was going to be used for service health monitoring as well, this seemed like a good choice. One downside is further complexity in DNS handling, however, since not all hosts in the deployment were configured to run Consul using Ansible playbooks.

4.4 Distributed Systems Challenges

There are several challenges to building even a small-scale distributed system compromising multiple operating systems on multiple network segments with multiple layers of baremetal, virtual machine, and/or containerization.

4.4.1 Physical Distribution

One of the core challenges when building distributed systems results from using separate DNS host names and physically separate data centers and/or logically separated subnets.

At the start of the DIMS project, hardware was physically located in two server rooms in two separate buildings operated by the Applied Physics Laboratory, with staff being located on a separate floor in one of the buildings. What is more, some staff had computers using static IP addresses with direct access to the internet, while others used dynamic IP addresses behind a separate APL "logical firewall" device. This meant use of four separate IP address ranges on four subnets behind two different firewalls. Other hardware was located in the main UW Data Center in the UW Tower building on a fifth network. Add to this one hypervisor on a system in the APL server room and another in the UW Tower, each with a separate OpenVPN server, with the necessity to route traffic between virtual machines on the two hypervisors. (Both hypervisors, by the way, were different and ran on two different operating systems.)

On multiple occasions, hardware had to be moved from one location to another (which meant changing IP addresses on both bare-metal hosts and virtual machines, changing routes, and changing VPNs.) The last time hardware was moved, in order to consolidate it all into one data center, the entire system became unstable and non-functional.

One of the machines being moved served as the hypervisor for approximately a dozen virtual machines making up the core of the DIMS development environment. At least three previous attempts were made to task team members with documenting the "as-built" configuration of all of these components, their IP addresses and routes, and mechanisms for remote control, in order to plan for the configuration changes needed to perform the move. Each previous time a move had been planned it had to be put off because higher priority tasks needed to be addressed and/or team members had left the project before they had completed the tasks necessary for migration. When the hardware finally had to be hastily moved due to the impending extended leave of a key participant, the hastily performed move caused the entire DIMS network to become non-functional and the PI and two team members spent the next five days working to get the system functional and stable again. This process revealed that the configuration of the DIMS systems was significantly below the quality level previously assumed. System configuration settings were not adequately documented, were almost entirely hand-crafted (as opposed to being under Ansible configuration control as was specified), used two different hypervisors (KVM and Virtualbox) on two different operating systems (RedHat Enterprise Linux 6 and Debian) and the networking relied heavily on something known as Project 172 private address routing combined with internal virtual networks that were administered by just one former team member using remote desktop services and/or X11 forwarding from a workstation that was no longer available as an option to use. The instability and outages caused by this long-delayed (yet required) hardware move set the team back significantly and had ripple effects on other deadlines and events that could not be adjusted or canceled.

4.4.2 Stability

Due to the inherent inter-relationships between subcomponents in a distributed system, stability of the overall system is a constant challenge. Not only are relocations of hardware like those described in an earlier Section a contributor to instability, but so are software changes. As the DIMS project is using open source operating systems and tools that may be updated on as frequent as a monthly basis, often resulting in parts of the system "breaking" when an update happens.

As the entire distributed system was not put under Ansible control from the start, and "as-built" documentation was lacking in several areas, some architectural changes resulted in critical system components breaking, with no clear way to fix them. This could lead to days of running `tcpdump` and `strace`, watching `syslog` log file output, and poking at servers (after clearing the browser cache frequently to eliminate problems due to erroneous cached content) in order to diagnose the problem, reverse engineer the solution, and meticulously put all of the related configuration files under Ansible control. This was complicated by the fact that the team members who set up some of these systems were no longer on the project and could not assist in the cleanup.

One of the solutions that was attempted was to use Docker containers for internal microservices. The hope was to avoid some of the complexities of out-of-date libraries, version incompatibilities in programs, and differences in operating systems. The project team looked at several ways to deploy Docker containers in a clusterized environment and chose to use CoreOS (now called "Container Linux by CoreOS"). While this allowed clusterization using `etcd`, `consul`, and eventually Docker Swarm mode, it also resulted in a trade-off between leaving the three servers running CoreOS for clustering stable (and thus drifting apart in versions from the regularly updated development hosts running Ubuntu 14 and Debian 8), or dealing with changes to configuration files that had to be ported to Vagrant Virtualbox "box" files and the bare-metal cluster at the same time. As these systems were not easily controlled with Ansible at first, this caused a lot of frustration that was never fully eliminated. As the baremetal servers were re-purposed for pilot deployment work, the central cluster services degraded and took some formerly working services with them.

4.5 Software Engineering Challenges

The software engineering skill levels and experience of the team members varied widely, as did their individual coding styles, language preferences, and debugging abilities. This resulted in several points of friction (both technically and politically) over time. It also made it difficult to rely on documented requirements and white board sessions to provide sufficient direction for programmers to independently produce "production" quality system components. A project of this scope requires more direct interaction between the PI (who knows the user requirements and what needs to be built to meet them) and individual team members (who are tasked with building those components). This requires a greater level of institutional support and commitment, or a more highly-skilled and experienced engineering team, than was available.

4.5.1 Using Agile

Problems with achieving and maintaining a cadence with Agile/Scrum and software releases, were exacerbated by the issues of team member physical distribution, time zone differences and work schedule differences. All team members were new to using Git, which has a steep learning curve to begin with. Differences in versions across workstations caused problems in sharing code using Git. Getting everyone to adopt common processes and tools proved to be difficult. The most prevalent model for branching, described by Vincent Driessen's "A successful Git branching model" was chosen as the right model to follow. Getting all team members to learn it, and follow it, was not entirely achieved. (A diagram of the model is shown in Figure *Vincent Driessen Git branching model*).

The dimsdevguide was produced, with sections laying out things like policy (Development and Core Tool Policy) and guidance on using Git (Source Code Management with Git).

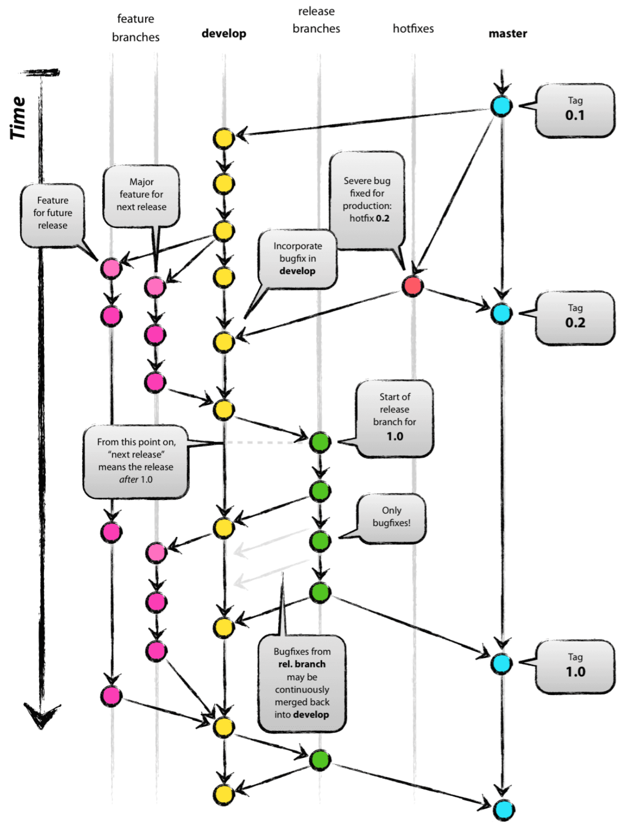

feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release *after* 1.0
Only bugfixes!
Tag 1.0
Bugfixes from rel. branch may be continuously merged back into develop

Fig. 4.1: Vincent Driessen Git branching model

What tended to happen over and over was a situation where a large number of disruptive changes and bugfixes would all be placed onto a single long-lived feature branch (sometimes going for weeks at a time) before merging them back into the `develop` branch, let alone released to the `master` branch. In order to test successfully (and sometimes just be be able to have a stable build at all) would require multiple repositories all being on the same feature branch. The worst case was that one part of the system would only work on one feature branch and another part would only work on a different feature branch, creating an impasse where a full build of multiple VMs would not work. This caused repeated states of instability and high stress leading up to demonstrations.

It wasn't until Q2 2017 that stability was achieved on the `master` branch, regular merges from feature branches to `develop` and vice-versa kept both `develop` and feature branches stable, and hotfix branches used more diligently to improve `master` and `develop` branches without losing these fixes on long-lived feature branches. In retrospect, "release early, release often" and "build from master" to validate merged features should be the mantra. (This process was adopted leading up to the pilot deployment, which was built almost exclusively from the public `master` branch of https://github.com/uw-dims/ansible-dims-playbooks).

4.5.2 Challenges Related to Abstraction

Related to the *Distributed Systems Challenges* are challenges related to abstraction. Abstraction presents challenges in many ways.

1. The presence of an abstraction layer in code and service connections may create opacity (i.e., things behave like a *black box* and either work or fail, with little feedback). This requires greater expertise in debugging.
2. Abstraction in service oriented architecture requires a greater level of expertise in configuring and debugging systems in that it is necessary for someone to be able to understand and control the system at any level in the abstracted stack, or to be able to jump up and down the application stack in order to diagnose and debug the system when something does not work. If someone is only capable of understanding the highest layer in the abstracted stack and something does not work, they must rely on someone else who has expertise at the lower layers in order to debug and fix any problems. (This is related to the issue of *opacity* in the system, or the *black box* effect).
3. The lack of an abstraction layer requires more direct connections between *caller* and *callee* in programs, or between *connector* and *connectee* in TCP/IP socket connections. This directness seems simple at first, but in the face of a large number of connections or calls, it becomes very difficult to add each new connection, to make changes, or to debug when one of a large number of similar looking connections fails.
4. The lack of an abstraction layer also makes it harder to support versioning of APIs, since more direct calls are being made and things like changes in function names or changes in IP addresses, DNS names, or TCP/IP ports.

One place where abstraction comes in handy is providing a standard application programming interface (API) that takes a simple set of parameters in a function call, but hides the underlying details of where data is obtained prior to being returned to the caller in a single data structure. The Trident portal holds a limited set of attributes about a user, but some programs integrated into DIMS need more attributes. That means one of two things must happen:

1. Trident must be modified to support the extra attributes that are needed, or
2. An abstraction layer is added that allows one call to Trident to get the attributes it holds, and a second call to a DIMS database component to get the extra attributes, combining them into one data structure that is returned to the caller. This is illustrated in Figure *user-attributes.jpg*.

4.5.3 Backward Compatibility

In Section *Stability*, the problem of version drift between like components in a distributed system was discussed. The right answer is to put everything under Ansible control from the very start and to handle subtle variations in how things are installed and configured by using the minimum necessary "glue scripting" so as to stay in sync with versions across all subsystems. This is a difficult task that takes expertise that was not commonly available across all team members.

Backward compatibility issues also arose with one of the core components the DIMS project was using: the Trident portal. Open source projects (DIMS included) move forward and change things at whatever cadence they can follow. Sometimes this means some fairly significant changes will happen quickly, requiring some effort to keep up. This results in a challenge: stay on the cutting edge by focusing effort as soon as changes are made, or try to maintain some stability by pinning to older versions that are working?

In order to keep stability in the development environment to make forward progress on a number of fronts, the Trident version was pinned to `1.3.8`. The pilot deployment, however, would need to be done using a newer version (at the

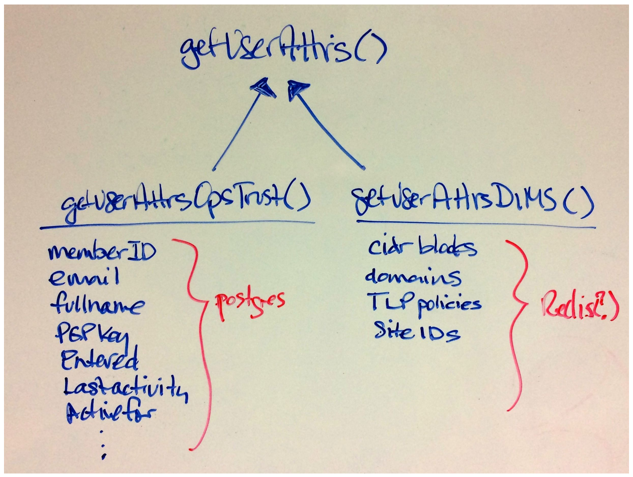


Fig. 4.2: user-attributes.jpg

time `1.4.2`, currently `1.4.5`). There were at least two significant changes made between the `1.3.8` and `1.4.2` versions: The CSS style sheets used by the Trident portal GUI went from two files to one file, changing names at the same time, and there were some incompatible changes to the command set for the `tcli` command line interface that was used by Ansible to install and configure Trident. These changes required some reverse engineering of the changes by extracting files from the two packages and differencing everything in order to then use conditional logic and dictionaries to quickly switch between version `1.3.8` and `1.4.2` in order to keep a stable working demo and simultaneously prepare for the pilot deployment. (A variation of this technique is illustrated in the code block *Excerpt of client.py showing version support*). This diverted a significant amount of energy for a period of time that pushed other tasks to the background.

4.6 External Dependencies and Pacing

One of the most laudable goals of this project was the use of open source tools to be integrated into an affordable distributed system capable of scaling to the degree needed to handle millions of security events per day. The flip side of this is that every one of the open source tools that come from outside entities are produced on someone else's whim (including pace of release, quality of testing, rate of disruptive changes in code, time available to respond to interactions, etc.)

For example, keeping up with the pace and direction of change in STIX core development, and difficulties in maintaining development momentum within the project team, limited this avenue and it could not be sustained. (See *STIX Development Libraries*.) Other challenges listed in this section caused the pace internal to our team to be much slower than desired, resulting in difficulty in our reaching out and interacting with developers of the Trident portal. The friction within the project slowed some of our internal development, requiring that we play "catch-up" late in the project and not being able to provide as much input as we had hoped to their developers towards features we needed.

4.7 Testing

The contract included requirements for adherence to a specific software design standard and for two iterations of producing a full-system test plan and test report. The prime contractor organization had no previous experience with these standards and no formal in-house training or resources to support production of the test plan or test report. The sub-contractor providing project management assistance procured a software testing subject matter expert with experience at a large aerospace company. The initial plan developed by this expert (while perhaps typical for a large project in a large organization with specialized staff dedicated to testing) went far beyond what the DIMS Project's staffing and budget resources could support to manage the test planning, execution, and reporting, not to mention the cost of the commercial testing tools being recommended.

The PI identified MIL-STD-498, described at A forgotten military standard that saves weeks of work (by providing free project management templates). A simpler and more manageable plan was developed following the MIL-STD-498 Software Test Plan (STP.html), along with the Software Test Report (STR.html). Even with this simpler plan, the initial test consumed the majority of the project effort for several weeks leading up to the deliverable deadline.

Prior to the second system-wide test cycle, the PI spent time towards automating production of the Test Report from machine-parsable inputs. The second test took less effort than the first, but the amount of manual effort was still large and one team member did not produce any input for the test report until the week after the report was delivered to the sponsor, despite numerous requests in the weeks leading up to the deadline.

CHAPTER 5

Needed Enhancements

"*Perfection is finally attained not when there is no longer anything to add, but when there is no longer anything to take away.*"

Antoine de Saint Exupéry, Terre des Hommes (1939)

There are several areas in the DIMS development architecture that are more complex than is desirable, where things are difficult to use and bugs often get in the way. As the quote above attests, it is a challenge to make things simple, elegant, and robust. Given limited resources, time deadlines, and pressures to get things working, some components of the DIMS project have attained the success of a prototype and are now in need of reimplementation as a cleaner, tighter, next major version. This section describes some of those components and thoughts on what to do next.

5.1 Packer and Vagrant Workflow Process

When the DIMS project began, and the team first began to work with multiple Linux distributions (to follow guidance of each open source tool producer's specified supported platform requirements), the decision was made to use Packer for creating virtual machine images from distribution disk image files (colloquially known as **ISOs**, short for ISO 9660 format read-only boot images.).

To facilitate applying generic configuration settings and package choices to multiple Linux distribution boot images, some helper `Makefile` rules files were created, allowing the dependency chain to be defined such that Unix `make` can then optimize the production of products. You don't want to have to perform every step in a lengthy process (that involves downloading over a Gigabyte of package files) every time you want to create a new Vagrant virtual machine.

This process pipeline eventually included a Jenkins server that would trigger `ansible-playbook` execution to implement a complete continuous integration/continuous deployment environment. This process looked like that depicted in Figure *Packer/Vagrant Workflow*.

The options at the time were to use something like Chef, Puppet, Heat, or Terraform. The choice had been made to use Ansible for system configuration automation, which the team did not see as being compatible with Chef and Puppet, and programs like Heat and Terraform were designed for much more larger and more complicated multi-region cloud service deployments. We wanted DIMS deployments to fit in a single server rack using a small external network footprint, since the PRISEM project on which DIMS was to be built was built that way.

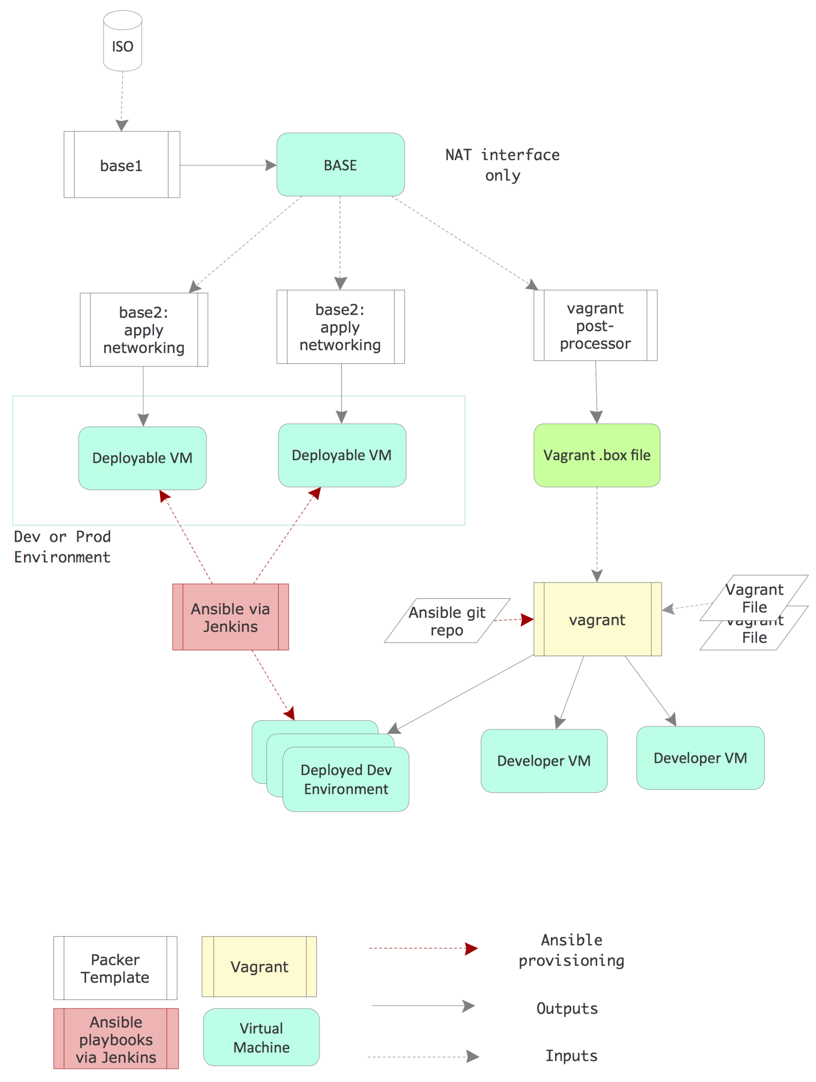


Fig. 5.1: Packer/Vagrant Workflow

In September of 2015, well into the DIMS project, Hashicorp came out with "otto" and "nomad".[1] These looked promising, but were immature and looked costly to implement. In August 2016, Hashicorp announced they were decommissioning and abandoning "otto".[2] There is still a need for a tool like this, but we continued to use the tools we had developed despite their limitations. Continued simplification of these tools and integration with Ansible through use of the inventory and templating scripts, Packer `.json` files, and `Vagrantfile` configuration files would help smooth things out.

In the long term, a solution that falls within the gap between a single server rack with custom `Makefile` and scripts and something as complex as OpenStack or AWS CloudFormation is desired. This could be Packer and Terraform with custom provisioners. Experiments using Packer to create Amazon instances was successfully performed and a prototype of Terraform to provision Digital Ocean droplets has been initiated and is anticipated to be completed after the project is completed for use in subsequent follow on projects using the DIMS software products.)

5.2 Normalization of Kickstart

Along with a standardized and simplified virtual machine instance build process, a related simplified bare-metal boot capability is needed for more efficient deployment of servers. Debian has a mechanism known as Kickstart that allows pre-configuration of steps needed to perform an unattended ("hands-off") installation of the operating system at boot time. This mechanism is used in DIMS as part of the Packer workflow, and as part of the customized USB thumb drive installer. It can also be made to work by Virtualbox (or other hypervisors, for that matter) directly.

- The Packer workflow uses inline commands to perform some initial system setup steps necessary to then use Ansible for the remainder of the system configuration.
- The Vagrant workflow for Ubuntu and Debian uses some inline commands in the `Vagrantfile` for pre-Ansible customization, and some external scripts.
- The Virtualbox Box file preparation for CoreOS uses external scripts to prepare CoreOS for Ansible, and other `Vagrantfile` inline commands for boot-time customization.
- The automated USB installer workflow uses the Kickstart `pressed.cfg` file for what preparatory steps Kickstart is capable of performing, and a secondary pre-boot customization script that is downloaded and executed at install time for other pre-Ansible customization.
- Manual creation of virtual machine guests or baremetal systems using the default Ubuntu or Debian installer (without using Kickstart) requires manual steps be performed to prepare the system for Ansible control.

The problem is, each of these workflows was created by separate team members at different times, much of this without coordination or integration. Multiple attempts were made to task team members with identifying all of the above and reducing or refactoring the steps into a coherent and consistent set of commonly-used scripts resulted. This resulted in some degree of simplification and integration, but there is much work remaining to be done here.

Rather than having multiple incompatible inline shell mechanisms (which are the easiest to implement, but least compatible means of accomplishing the same tasks), a cleaner way to handle this situation is to reduce the steps required in `preseed` steps to the bare minimum necessary to enable external Ansible control. Then these simpler `pressed` steps can be included as necessary by each tool during Kickstart or the install-time tasks can be performed in Bash shell scripts that can be called by each tool. This makes all of the install-time steps consistent, configurable using Ansible, and shared across tools. The remaining steps can then be turned into Ansible playbooks that can be applied post-boot, again in a completely consistent manner.

[1] https://www.hashicorp.com/blog/otto/
[2] https://www.hashicorp.com/blog/decommissioning-otto/

5.3 Configuration Management Database

At the start of the project, a combination of variables stored in files that could be exported through the shell's environment into scripts, `Makefile` rules, and Ansible vars files, was used. These mechanisms were not fully integrated and it was difficult to switch between different sets of variables to support multiple simultaneous deployments. For this reason, the team clung to a single deployment for far too long.

In terms of Ansible, the use of the simplistic and limited INI style inventory, with group and host variable files, was easy to learn, but proved difficult to manage for multiple deployments and for this reason its use held the project back for a long time.

Having multiple deployments was always a project objective, but how to achieve it using free and open source tools was not obvious to the team. It was clear that a configuration management database was needed that supported a more object-oriented "inheritance" style mechanism of defining variables that would more easily accommodate managing multiple simultaneous deployments.

The need here is for a system that behaves something like the way OpenStack supports a CLI for getting and setting variables in concert with a "cloud" configuration file to control high-level storage locations that allow a single interface to operate across multiple configuration databases. Ideally, this database would serve as what is called a "single point of truth" or "single source of truth" about not only hardware in a data center (e.g., servers and network equipment, rack slot allocations, switch ports, VLANs), but also configuration specifics that would drive Ansible playbooks for configuration and templating of scripts that run on the systems. A lot of research was done, but nothing seemed to be a good fit. Commercial tools like Ansible Tower[3] may solve this problem, but that was neither in the project's budget, nor did that conform with the objective of using only free and open source software tools. Other solutions were similarly focused on enterprise-level deployments and were not suitable for our use.

The tools that seem to exist are all focused on large-scale cloud deployments for massively-scaled, multi-datacenter deployments using a federated model. Trying to add them to the mix would be too costly and divert too much attention from other critical elements of system integration. What is needed by projects like this is a mechanism for many small-scale, single-datacenter deployments that are configured locally, but pull much of their code from the public repositories on GitHub.

The solution that was settled upon in the DIMS project was a combination of most variables being defaulted in roles with a separate "private" directory tree for each deployment that holds customization details in the form of Ansible YAML style inventory files and local customized files and templates that playbooks in the public `ansible-dims-playbooks` repository use before looking for generic equivalents in the public repository. This allowed the ability to operate multiple deployments in parallel with the public repository with less hassle, though this is still not the ideal solution.

5.4 Continued Reimplementation and Integration of Services

Due to some of the issues listed in Section *Challenges Encountered*, several of the sub-systems in the original development deployment that were never fully under Ansible control and had been hand-configured became unstable and failed. The DIMS dashboard web application, the Tupelo server, the Jenkins server, were all built on older Ubuntu 12.04 LTS and a Linux appliance virtual machine that was one of the first servers installed. As these base operating systems were manually created and managed, and the person who had originally set them up was no longer working on the project, rebuilding them would prove difficult and took lower priority to completion of other tasks. Some of the service, such as the dashboard web app, where also constructed using older Ansible playbooks that did not conform with the newer standards used for later playbooks and would similarly take extra time to be fully brought up to current standards. These tasks were added to Jira tickets, along with rebuilding all of the other central components (e.g., the Jenkins build server that failed when accidentally upgraded to a version with non-backward compatible features).

[3] A couple months before the DIMS project end of period of performance, RedHat released the Ansible Tower product in open source form as the AWX Project. There was no time to fully learn how to use and evaluate this product, though it appears it would be relatively easy to add it to the `ansible-dims-playbooks` as a role and deploy it along with other system components.

In the final months of the project, effort was put into re-implementing as many of the original (version 1) deployment services as possible. The RabbitMQ service, Jenkins with Git+SSH and Nginx file service, and Trident portal were all reimplemented and replicated on a new server. The Tupelo, PRISEM RPC services, and Lemon LDAP (for single-signon service) server roles remain to be re-implemented and updated from their original Ansible roles and the hand-crafted Jira system implementations. The DIMS Dashboard, Redis server, and ELK stack Ansible roles (which were all working in prototype form in year 2, prior to moving the project to UW Tacoma) should be easy to port after that, but it is likely that the Javascript Dashboard and Java Tupelo code are now out of date and will require experienced Javascript and Java programmers to bring them up to current coding standards.

5.5 Secrets as a Service

In the first year of the project, many secrets (passwords, non-public sensitive sample data, private keys, and SSL/TLS certificates) were committed to source code at worst, or passed around manually. This is neither a secure way to deal with these secrets, nor does it scale well. Ansible Vault and use of a separated private directory were prototyped as mechanisms to deal with the storing of shared secrets, but passwords were not entirely eliminated in favor of a ubiquitous single-signon mechanism. (Single-signon was implemented for Jira, Jenkins, and the DIMS Dashboard server, but no farther.) Trident uses a Javascript Web Token (JWT, pronounced "jot"). LDAP and JWT tokens could be extended, a service like FreeIPA or HashiCorp Vault (both used in the system illustration in Figure *OpenCredo core building blocks*), or Docker's built-in secrets management feature (see Introducing Docker Secret Management and Manage sensitive data with Docker secrets) could be used. There are many tradeoffs and service integration issues in this area that make this a non-trivial problem for all open source projects of this scope.

5.6 Testing and Test Automation

Section *Testing* describes effort put in by the PI to automate system-wide testing. This primarily centered on post-processing output of BATS tests and JSON files created using a simple user interface that collected information related to the tests as described in the DIMS Commercialization and Open Source Licensing Plan v 1.7.0. Another team member created scripts in Jira that produced these same JSON files describing the output of manual tests managed using Jira, reducing the amount of effort to perform and report on user interface tests. The final product was a structured set of RST files that could be processed with Sphinx to produce the test report in HTML, PDF, and epub formats. Such test automation decreased effort required to perform test cycles and supported automated production of reports with very little need for manual input.

The larger vision here was to scale test production and reporting by orchestrating the process using Ansible. For example, an Ansible playbook could invoke the `test.runner` script (see Running Bats Tests Using the DIMS test.runner) on every system, placing output into a known file, which can then be retrieved using the `fetch` module into a hierarchical directory structure based on the system names. The contents of this directory tree can then be turned into separate RST files and an `index.rst` file generated that is then rendered using Sphinx.

Further automation of the testing process along these lines would decrease the cost and disruption to regular development efforts, allowing more frequent testing and decreasing the effort spent on resolving failed tests. (This task was on the to-do list, but had to take a lower priority to other more important tasks.)

CHAPTER 6

Recommendations for Follow-on Projects

This section includes recommendations for consideration in planning follow-on projects, whether they use the DIMS Project software products or not, intended to help reduce friction in the software development process. Any group wanting to form a regional information sharing collaboration building on the DIMS software base, or a small open source development project wanting to use the platform for secure software development, will want to consider these suggestions.

During the time of this project, we encountered all of the typical problems that a team would have in the lifecycle of designing, deploying, and maintaining a small-scale (on the order of dozens of server components) distributed system. In order to have isolated development, test, and production systems, the difficulty factor goes up. To perform multiple production deployments and update code over time further increases the difficulty factor. Eventually, the lack of automation becomes a limiting factor at best, or leads to an extremely unstable, fragile, and insecure final product at worst.

The benefit to those who chose to follow our lead will be a faster and smoother journey than we experienced during the DIMS project period of performance. All of the hurdles, mistakes, struggles, and ultimately the many successes and achievements in distributed system engineering were not easily found in the open source community. The DIMS System Requirements v 2.9.0 documents security practices and features that we have attempted to incorporate to the greatest extent possible, in a way that can be improved over time in a modular manner. The system automation and continuous integration/continuous deployment features help in implementing and maintaining a secure system. (Red team application penetration testing will further improve the security of the system through feedback about weaknesses and deficiencies that crept in during development and deployment.)

6.1 Focus on System Build Automation

From the first days of the project, the PI constantly told the team to *not* build things by hand, since that does not scale and cannot be replicated. We didn't need one hand-built system, we needed multiple systems for development, testing, production, and anyone wanting to use the DIMS system needed to be able to quickly and easily stand up a system. This can only be accomplished using stable and well-documented build automation.

Ansible was chosen early on as what looked like the most promising system build automation tool, so in this sense saying "focus on system build automation" means "focus on mastering Ansible." Anyone wanting to build on the project's successes, and avoid some of its challenges, must ensure that *all team members* involved in development or system administration master using Ansible. That can't be stressed enough, since any system that is not under Ansible control is at risk of instability and very costly effort to fix or replace should something happen to it. Any host that is fully under Ansible control can be quickly rebuilt, quickly reconfigured, and much more easily debugged and diagnosed.

Rather than use SSH to log into hosts, whenever possible use `ansible` ad-hoc mode. The ability to invoke modules directly using `ansible` not only helps learn how the module works, but it also allows very powerful manipulation of any or all hosts in the inventory at once. This makes debugging, configuring, cleaning up, or any other task you need to perform, much easier and more uniform. Avoiding logging into hosts and remotely using the command line shell also helps focus on controlling the configuration and using the automation rather than hand-crafting uncontrolled system changes.

Using `ansible-playbook` with custom playbooks for complex tasks, or by invoking a master playbook that includes all other playbooks, facilitates performing an action across any or all hosts to keep things better in sync. It documents the steps in a way that they can immediately be performed, rather than documenting in English prose with code blocks that need to be cut/pasted and edited manually to apply them when needed. The friction caused by manual configuration of hosts is one of the biggest impediments to building a complex and scalable system.

Last, but not least, by putting **all** configuration files under Ansible control from the start enables much easier change management and incremental configuration adjustment with less disruption than with manual system administration. It is far easier to search Git history to figure out what changed, or search a few directory trees to locate where a particular variable is set or configuration file was customized. The `ansible_managed` line in configuration files on the end systems tells you precisely which file was used by Ansible to create the current file, allowing you to make changes and commit them to the Git repository to maintain history and maintain control. Editing a file on the end host and introducing an error, or accidentally deleting the file, make recovery difficult, while reliably re-applying a role is simple and easy.

6.2 Standardize Operating Systems

As much as possible, standardize on a small and manageable number of base operating systems and versions, and strive to keep up with the most recent release (and perhaps one previous release) to avoid supporting too many disparate features and one-off workarounds. Every major or minor version difference (e.g., 12.04.4 vs. 14.04.4 for Ubuntu Linux) or distribution difference (e.g., Fedora vs. RedHat Enterprise Linux vs. CentOS) can have implications for computability of sub-components, be they programs, libraries, or add-ons and utilities.

While this recommendation sounds simple, it is not. This task is made difficult by the choices of supported base operating system(s) made by each of the open source security tools you want to integrate. Great care needs to be taken in making the decisions of which operating systems to support, balanced with available expertise in the team for dealing with required debugging and configuration management tasks.

Using a configuration management program like Ansible helps by expressing installation steps using different Ansible modules or plays, though it does require engineering discipline to deal with complexity (above and beyond what a Bash script would entail, for example) and to ensure the right plays work the right way on the right operating system. This could mean maintaining a large set of group variables (one for each alternative operating system), using variables in inclusion directives to select from those alternatives, and/or using "Ansible facts" derived at run time with logic (e.g., `when:  ansible_os_family == "Debian"` as a conditional in a playbook) Developing Ansible playbooks in a modular way that can easily accommodate generalized support for multiple operating systems (e.g., using a "plug-in" style model) is a more sophisticated way of writing playbooks that requires a greater level of expertise of those writing the playbooks. Such expertise, or institutional support for employee training to achieve it, are not always available.

6.3 Standardize on Virtual Machine Hypervisor

Attempting to use different hypervisors on different operating systems vastly increases the friction in moving virtual machine resources from host to host, from network to network, and from manual to automated creation. While it is often possible to export a VM image, convert it to another format, and import that VM back into another hypervisor, these steps require additional planning, time and effort, and data transfer and storage resources that add friction to the development process.

Start with a preferred hypervisor to support and take the time to migrate legacy virtual machines to that preferred hypervisor, rather than attempting to support part of the system with one hypervisor and the rest with another. If it becomes necessary to support additional hypervisors, require replication of the *entire* system of systems in a separate deployment (i.e., fully independent, not sharing any resources in a way that couples the heterogeneous systems) to ensure that tests can be performed to validate that all software works identically using the alternate hypervisor.

The `vncserver` role[1] was created to make it easier to remotely manage long-running virtual machines using a GUI hypervisor control program. Using CLI tools is also necessary, however, to more easily script operations so they can be parallelized using Ansible ad-hoc mode, or scheduled with `cron` or other background service managers.

6.4 Manage Static Config Files Differently than User-controlled Files

Managing files in `/etc` is different than `$USER/.gitconfig`. Let users customize things, and add (merge) group content rather than wholesale replacing files based on templates. Blindly installing configuration files is not idempotent, and causes regression problems for users when an Ansible playbook or role wipes out changes a user has made and takes the configuration file back to an initial state.

There are several ways to do this, some more complicated than others. One of the easiest ways is to start with a generic file that has very little need for customization and will run on all systems, which in turn uses a *drop-in* inclusion mechanism to in turn support inclusion of two types of files:

1. Adding operating-system specific additions that are selected by some variable, such as output of `uname -s` as a component of the file name, or:
2. Allowing users to control their own customizations by including a file with some string like `local` in its name.
3. Supporting the ability for users to place their account configuration files in a personal Git repository that can be cloned and pulled to development systems so as to make the configurations consistent across hosts.

6.5 Robust, Flexible, and Replicable Build Environment

Some of the DIMS tools were initially prototyped using the Unix `make` utility and `Makefile` rules files. The `make` utility is nice in that it supports dependency chaining. Things don't need to be rebuilt if the constituent files used to build them have not changed. This works great for source code, since programs are all static files (e.g., `.c` and `.h` files for C programs) that can easily have timestamps checked to see if they require recompiling to create new libraries or executable files. It is a little more difficult when a script is produced from a template, which is produced from a complex set of inventory files, host variable files, group variable files, and command line variable definitions as is supported by Ansible. In that case, the `Makefile` model is harder to use, especially for those who are not experts in how `make` works and may not have the skills required to efficiently debug it with `remake` or other low-level process tracing tools.

Tools like Jenkins or Rundeck provide a similar kind of dependency chaining mechanism which may be preferable to `make`, provided that programmers carefully use variables and templating to produce the build jobs such that they can

[1] https://github.com/uw-dims/ansible-dims-playbooks/blob/master/roles/vncserver/tasks/main.yml

be deployed to development, testing, staging, and production environments without having to manually change hard-coded paths, etc. This level of generality may be difficult to set up, but is necessary to be able to scale and replicate the build environment. This may sound like a "nice to have" thing, but when cloning the system for deployment requires manually copying build artifacts out of the one-and-only development build server, manually setting up a mechanism allowing virtual machines to access the files, and manually keeping it up to date as things change, the "must have" nature makes itself painfully obvious.

6.6 Avoid Painting Yourself into a Corner with Versions

From the start, build everything to support at least two operating system release versions (the current release and one release back, or `N` and `N-1`) and try to move as quickly as possible to the current release to avoid getting locked in to older systems. This process is made easier if everyone writing scripts and configuration files follows a "no hard-coded values" rule for things like version numbers, hashes of distribution media for integrity checking, file names of ISO installation disk images, etc.

If all of the required attributes of an operating system release (e.g., version major and minor number, CPU architecture type, ISO download URL, SHA256 hash of ISO, etc.) were referenced with variables and those variables used consistently throughout the OS build and Ansible deployment and configuration process, alternating between the two is a simple matter of alternating between two sets of variable definitions. This is where dictionaries (also known as "maps") come in handy, allowing a single key (e.g., "ubuntu-14.04.5") to serve as an index to obtain all of the constituent variables in a consistent way. If the Packer build process, the Kickstart install process, and the Ansible playbooks, all define these attributes in different ways, it becomes very difficult to upgrade versions.

Since operating systems are incrementally improving over time, the build environment **must** take this into consideration to keep you from getting painted into a metaphorical corner and finding it difficult to get out (without spending a lot of time that should otherwise be directed to more productive tasks). Requiring support for version `N` and `N-1` simultaneously not only provides a mechanism for testing package and configuration updates across versions, but means that it will be much simpler when version `N+1` is released to upgrade, test and plan a system-wide migration to the new OS release.

Similarly, source code and system configuration (e.g., Ansible playbooks) should also support versioning. An example of how to do this is found in the GitHub source repository for openstack/python-openstackclient. The source code for client.py (starting at line 24 in client.py, and highlighted in the following excerpted code block) shows how this is done by defining the `DEFAULT_API_VERSION` (which can be changed via the `--os_identity_api_version` command line option), and mappings of the option strings to directory names found in the directory of openstack/python-openstackclient and to module names.

Listing 6.1: Excerpt of `client.py` showing version support

```
DEFAULT_API_VERSION = '3'
API_VERSION_OPTION = 'os_identity_api_version'
API_NAME = 'identity'
API_VERSIONS = {
    '2.0': 'openstackclient.identity.client.IdentityClientv2',
    '2': 'openstackclient.identity.client.IdentityClientv2',
    '3': 'keystoneclient.v3.client.Client',
}

# Translate our API version to auth plugin version prefix
AUTH_VERSIONS = {
    '2.0': 'v2',
    '2': 'v2',
    '3': 'v3',
}
```

Of course this requires greater engineering discipline when programming, but had this technique been known and used from the start of the project it would have resulted in a much more organized and structured source directory tree that can support deprecation of old code, transition and migration to new versions, as well as clean deletion of obsolete code when the time comes. Using this mechanism of uniformly handling version support is much more modular than using conditional constructs within programs, or mixing old and new files in a single directory without any clear way to delineate or separate these files.

6.7 Budget for System Maintenance

To paraphrase a joke in the programming world: "You have a problem. You decide to solve your problem using free and open source software tools and operating systems. Now you have two problems." Sure, its a joke, but that makes it no less true.

Trying to compose a system using open source parts that are constantly changing requires constantly dealing with testing upgrades, updating version numbers in Ansible playbook files, applying patches, debugging regression problems, debugging version inconsistencies between systems, and updating documentation. The more software subsystems and packages that are used, the greater the frequency of changes that must be dealt with. Assume that somewhere between 25% to 50% of the project working time will be spent dealing with these issues.

The automation provided by Ansible, and the integration of unit and system tests (see Testing System Components) helps immensely with identifying what may be misconfigured, broken, or missing. Be disciplined about adding new tests. Regularly running tests saves time in the long run. Make sure that all team members learn to use these tools, as well as spend time learning debugging techniques (see Debugging with Ansible and Vagrant).

6.8 Testing

To avoid the issues described in Section *Testing*, follow-on projects are strongly advised to use these same MIL-STD-498 documents (leveraging the Sphinx version of the templates used by the DIMS Project, listed in Section *Software Products and Documentation*) and the simpler BATS mechanism to write tests to produce machine-parsable output.

We found that when BATS tests were added to Ansible playbooks, and executed using the `test.runner` script after provisioning Vagrant virtual machines, it was very easy to identify bugs and problems in provisioning scripts. Friction in the development process was significantly reduced as a result. This same mechanism can be extended to support the system-wide test and reporting process. (See Section *Testing and Test Automation*).

CHAPTER 7

License

CHAPTER 8

Appendices

8.1 STIX Development Libraries

Subject: Re: [cti-users] Re: Announcing python-stix2

```
Sender: <cti-users@lists.oasis-open.org>
Delivered-To: mailing list cti-users@lists.oasis-open.org
Received: from lists.oasis-open.org (oasis-open.org [66.179.20.138])
    by lists.oasis-open.org (Postfix) with ESMTP id B1A965818580;
    Wed, 15 Mar 2017 09:48:17 -0700 (PDT)
Message-ID: <58C96F55.4030104@apl.washington.edu>
Subject: Re: [cti-users] Re: Announcing python-stix2
Date: Wed, 15 Mar 2017 09:44:05 -0700
From: Stuart Maclean <stuart@apl.washington.edu>
To: "Back, Greg" <gback@mitre.org>, Eyal Paz <eyalp@checkpoint.com>,
        "cti@lists.oasis-open.org" <cti@lists.oasis-open.org>,
        "cti-users@lists.oasis-open.org" <cti-users@lists.oasis-open.org>
References: <AAD76FB4-D5FE-41FA-9BDD-1EE6BBE4062D@mitre.org>
        <2956157C13955D458A922DDE028FEE7E0209A8F61D@IL-EX10.ad.checkpoint.com>
        <D3938F78-9B27-48B2-87BA-E57803DEBA81@mitre.org>
In-Reply-To: <D3938F78-9B27-48B2-87BA-E57803DEBA81@mitre.org>

On 03/15/2017 09:22 AM, Back, Greg wrote:
>
> Hi Eyal,
>
> MITRE isn't working on anything, and I'm not personally aware of
> anyone else who is either. But if someone is, hopefully they're on one
> of these lists and will respond.
>
> Greg

Greg, all,

I have followed the STIX development over the past couple of years, and
```

```
even developed a Java library for STIX manipulation, see
https://github.com/uw-dims/stix-java. That was when STIX 1.1 was current.

I watched with some dismay as the XML schema way of doing things in STIX
1.1 was dismantled in favor of JSON for STIX 2.  While XML schemas are
complicated, they are very precise, and a document can be checked
against a schema to assert its validity.  Further, the richness of tools
for XML schemas, notably the JAXB/xjc tools for Java, gives developers a
huge leg-up in implementing an API for STIX manipulation.  I just
cranked the JAXB handle over the .xsd file set, and hey presto I had a
set of Java classes with which to start my API.

Going out on a limb here, I also think that Java developers LIKE more
discipline that Python developers.  Static typing vs duck typing?  The
XML schema way of representing information is disciplined, and leads to
fewer 'You meant X? I thought you meant Y' gotchas at runtime.
Extrapolating, I think the JSON way of data representation is less
disciplined than the XML way, hence the natural inclination of Python
developers to prefer JSON.

Looking at the STIX 2 specs, I admire effort the authors must have put
in.  But from a library builder's point of view, there being no
machine-ingestable docs (ie the xsd files in STIX 1.1), I am back to
square one (if I have missed any machine-readable docs, I apologize).

So, in a nutshell, my own feeling is that there will be no Java-language
STIX 2 manipulation tools, a sad fact, and I sincerely hope I am wrong
in this respect.  I do know that I won't add to my STIX 1.1 Java effort.

Stuart

----
```

8.2 Reference Links

Note: Over the life of the project, the PI lead the effort to research new technologies and guide the project team in learning these technologies in order to leverage them to acheive the integration goals of the project. The best references were organized and maintained as part of the PI's home page and all team members where urged to look there first and to let the PI know when they find new information that is not already there. These references are included here to help guide anyone else following this same path.

8.2.1 Containerization, Virtualization, "Microservice Architectures"

- Microservices, Wikipedia
- Microservices: a definition of this new architecture, my Martin Fowler
- Operating-system-level virtualization (a.k.a., "containers"), Wikipedia
- **Linux Containers**
  - Container Overview, from CoreOS
- When to use containers (and when not to), by Steven Vaughan-Nichols, May 15, 2017

- The Twelve-Factor App (*"The twelve-factor methodology can be applied to apps written in any programming language, and which use any combination of backing services (database, queue, memory cache, etc)."*)

Docker

Caution: Docker and all the related entities in the Docker ecosystem are changing **constantly** (and at a fast pace) so many of the resources below can become outdated or obsolete in a matter of months. Docker Releases occur regularly, so pay attention to dates and version numbers in the resources below when considering implementing something, but also look for useful tips and tricks that may still be relevant over time. Also, you may want to Subscribe to Docker Weekly Newsletter to maintain your own situational awareness of updates and high-quality articles, blog posts, and videos that can help you keep up.

Docker intro

- Docker ("Build, ship, and run any application, anywhere.")
- Docker: Build, Ship and Run Any App, Anywhere, by Martijn Dwars, Wiebe van Geest, Rik Nijessen and Rick Wieman
- Docker Reference Architecture: Designing Scalable, Portable Docker Container Networks, by Mark Church, Docker web site, October 18, 2016
- Docker Command Line
- Jeff Lindsay on Best Practices for Working with Containers and Docker
- GitHub docker/docs/sources/articles/dockerfile_best-practices.md (Best practices for writing Dockerfiles)
- Cooking with Docker: useful tips for beginners, by Blair Hudson
- Docker traps (and how to avoid them), by Mr. Blue Coat
- What is Docker?
- Introduction to Docker, Solomon Hykes (dotCloud founder and CTO of Docker), Twitter University
- 5 Reasons to Start Using Docker, by David Bolton, June 1, 2015
- GitHub prakhar1989/docker-curriculum ("A comprehensive tutorial on getting started with Docker! http://prakhar.me/docker-curriculum/")
- Play with Docker classroom, Docker web site

Tips and Tricks for More Advanced Docker Use

- What's eating my disk? Docker System Commands explained, by Nils De Moor, April 12, 2017
- Docker Container Anti Patterns, by Arun Gupta, October 30, 2016
- What's New In Docker 1.12, by Emmet O'Grady, August 2, 2016
- Docker Tips And Tricks: Some tips about running and building with docker, by Luís Armando Bianchin, February 15, 2016
- Flexible Docker entrypoint scripts, by camptocamp SA, March 22, 2016
- Deploying public keys in Docker containers, by camptocamp SA, March 22, 2016

- When and why should I use apt-get update?, stackexchange, May 19, 2014
- Best practices for writing Dockerfiles, Docker.com web site
- Create a base image, Docker.com web site
- Building good docker images, by Jonathan Berghnoff, October 3, 2014
- Where are my containers? Dockerized service discovery with Consul, by Jose Luis Ordiales, January 23, 2015
- Automatic container registration with Consul and Registrator, by Jose Luis Ordiales, February 3, 2015
- Trapping signals in Docker containers, by Grigoriy Chudnov, April 7, 2015
- Docker Inspect Template Magic, by Adrian Mouat
- Wrangling Grafana and InfluxDB into a Docker image, by Si Beaumont, February 17, 2016
- An Example of Docker Multi Host Clustering Part 1, by Joshua Davis, January 20, 2016
- An Example of Docker Multi-Host Networking with Hadoop - Part 2, by Joshua Davis, February 10, 2016
- An Example of Docker Multi-Host Networking With Hadoop Part 3 - Validating your Swarm Cluster, by Joshua Davis, February 18, 2016
- See also: *dockercleanup*

Books and guides (and a mindmap!)

- The Docker Ecosystem (mindmap by Krishnan Subramanian)
- **The Docker Book, by James Turnbull**
  - Sample Chapter
  - Code examples
- **O'Reilly Docker cookbook, by Sebastien Goasguen**
  - GitHub how2dock/docbook (Sample code and Vagrant files for O'Reilly Docker cookbook)
- **Welcome to the Docker User Guide**
  - Understanding Docker
  - Network Configuration
  - **Docker Compose**
    * See also: How to use Docker Compose to run complex multi container apps on your Raspberry Pi and the RaspberryPi section of *Small form-factor hardware systems*.
- How to Use Docker on OS X: The Missing Guide, by Chris Jones

Why Docker?

- Sailing Past Dependency Hell With Docker, by Alex Johnson, July 2, 2015

Architecture

- Docker vs. VMs? Combining Both for Cloud Portability Nirvana, by Thorsten von Eicken, September 2, 2014
- Understand the architecture (v1.10), Docker web site
- Higher Order Infrastructure - Microservices on the Docker Swarm, by Nicola Paolucci, GOTO 2016

Docker in Production

- Why Docker is Not Yet Succeeding Widely in Production, by Simon Hørup Eskildsen, July 2015
- The Realities of Docker in Production, by Ben Ball, March 31, 2015
- Docker in Production, by Jérôme Petazzoni, September 22, 2014
- **Docker deployments, by Paul Showalter & Karl Matthias, New Relic, September 10, 2014**
  - GitHub 6si/shipwright (The right way to build, tag and ship Docker containers.)
  - GitHub newrelic/centurion (A mass deployment tool for Docker fleets)
  - GitHub newrelic/check_docker (A Go Nagios check for Docker)
  - GitHub newrelic/go_nagios (Go lang package for writing Nagios checks)

Docker and Development

- **How To Cook Microservices (with Ruby spices) is a continuously updated collection of** notes, insights and ideas about building software platforms empowered by microservices architecture with Ruby language
- Docker for Developers, by Jérôme Petazzoni, November 25, 2014
- Tips for running Docker in development, by Philip Kallberg, May 27, 2015
- GitHub wsargent/docker-cheat-sheet (Docker Cheat Sheet)
- Test, Develop, Build, Stage with Docker, by Simone Di Maulo, May 2, 2015
- A Docker Dev Environment in 24 Hours! (Part 1 of 2), by John Fiedler, October 31, 2013
- **A Docker Dev Environment in 24 Hours! (Part 2 of 2), by John Fiedler, November 5, 2013**
  - GitHub relateiq/docker_public (Instant RelateIQ Development Environment - code from parts 1 and 2)
- Top 10 Open-Source Docker Developer Tools, by Lucas Carlson, March 5, 2014
- Executable Images - How to Dockerize Your Development Machine, by Quinten Krijger, August 29, 2015
- Development Environments with Vagrant, Docker, and Supervisord, by Tyler H.T. Cipriani, May 25, 2014

Docker and Security

- Docker Security, Docker web site
- Docker Ramps Up Container Security, by Jack M. Germain, May 13, 2016
- **Understanding and Hardening Linux Containers (PDF), NCC Group whitepaper by Aaron Grattafiori, April 20, 2016**

  - Docker and High Security Microservices: A Summary of Aaron Grattafiori's DockerCon 2016 Talk, by Daniel Bryant on August 14, 2016
- Containing and Attack with Linux Containers (PDF), Shmoocon 2016 presentation by Jay Beale, InGuardians, January 17, 2016
- Using Linux Containers to Jail Programs, by Jay Beale, Raleigh B-Sides, October 9, 2015
- GitHub docker/docker-bench-security (The Docker Bench for Security is a script that checks for all the automatable tests included in the CIS Docker 1.6 Benchmark. https://dockerbench.com)
- Introducing a *Super* Privileged Container Concept, by rhatdan, November 6, 2014
- Are Docker containers really secure?, by Daniel J Walsh, July 22, 2014
- Bringing new security features to Docker, by Daniel J Walsh, September 3, 2014
- Someone said that 30% of the images on the Docker Registry contain vulnerabilities, by jpetazzo, May 27, 2015
- Exploring Docker Volumes for Phases of Development, by Alan Kent, May 31, 2015
- Understanding Docker Security and Best Practices, Docker blog, May 5, 2015
- Container Security: Just The Good Parts, by Trevor Jay, April 29, 2015
- Docker SELinux Experimentation with Reduced Pain, by zwischenzugs, April 29, 2015
- **The sad state of sysadmin in the age of containers, by Erich Shubert [pointing out some extremely poor security practices**
  - reddit.com/r/programming comments thread
- Docker security gets thumbs-up despite containers' rapid rise, by Toby Wolpe, ZDNet News, January 12, 2015
- Docker security in the future, by Daniel J Walsh, March 19, 2015
- Docker Security: Best Practices for your Vessel and Containers, by Michael Boelen, January 22, 2015
- GitHub GDSSecurity/Docker-Secure-Deployment-Guidelines (Deployment checklist for securely deploying Docker)
- CoreOS vs. Project Atomic: A Review, Major Hayden, May 13, 2014
- Analysis of Docker Security, by Than Bui
- Storage Concepts in Docker: Persistent Storage, by Mark Lamourine, October 10, 2014 [Has a good discussion of SELinux labeling of directories for use by Docker containers]
- Docker's just a bit dodgy, but ready for rollout says Gartner, by Simon Sharwood
- Snappy Security
- Launch secure LXC containers on Fedora 20 using SELinux and sVirt, by Major Hayden
- See also the SElinux, Apparmor, GRsecurity section

Orchestration

- **Docker Orchestration workshop - Jérôme Petazzoni, Feb 21, 2016**
  - Part 1 (1:21:09)
  - Part 2 (0:59:37)
  - Parts 3 and 4 (3:20:54)
  - GitHub jpetazzo/orchestration-workshop (Slides and examples for the workshop)

- **Docker Orchestration & Metrics, Tiffany Jernigan, Seattle Docker Meetup, December 2016**
  - Slides from Tiffany's presentation
- Docker 1.12 Swarm Mode Deep Dive Part 2: Orchestration, YouTube by Andrea Luzzardi, July 28, 2016
- Create a swarm cluster with Docker 1.12 swarm mode, by Luc, July 5, 2016
- ContainerPilot and the Autopilot Pattern, by Tim Gross, Container Summit Austin, July 19, 2016
- 3 Node Swarm Cluster in 30 seconds (Docker 1.12), by John Zaccone, July 29, 2016
- **Apollo**
  - Capgemini Apollo: An Open Source Microservice and Big Data Platform, by Daniel Bryant, June 6, 2015
  - GitHub Capgemini/Apollo, An open-source platform for cloud native applications based on Apache Mesos and Docker. http://capgemini.github.io/devops/apollo/")
  - How Apollo Uses Weave and Weave Scope, by Graham Taylor, June 30, 2015
- Atlassian Orchestration with Docker: multi-host support for the win!, by Nicola Paolucci, December 16, 2015
- DockerCon EU: Keynote on Orchestration (Docker Machine, Swarm, Compose)
- Docker 101: Dockerizing Your Infrastructure, by Stanley Lewis
- Docker and Maestro for fun, development and profit, by Maxime Petazzoni
- Docker Containers and Kubernetes with Brian Dorsey, YouTube, December 23, 2014
- CoreOS Meetup: etcd and Kubernetes, YouTube, March 19, 2015
- Docker orchestration with maestro-ng, by heisel
- **GitHub signalfuse/maestro-ng (Orchestration of Docker-based, multi-host environments http://www.signalfuse.com)**
  - GitHub signalfx/maestro-base (Base Docker image for Maestro-enabled components http://www.signalfuse.com)
  - GitHub WIZARD-CXY/docker-cassandra (Docker image for Cassandra (Maestro orchestration))
  - GitHub iantruslove/docker-elasticsearch (Docker image for ElasticSearch (Maestro orchestration))
  - GitHub signalfx/docker-zookeeper (Docker image for ZooKeeper (Maestro orchestration) http://www.signalfuse.com)
  - GitHub signalfx/docker-kafka (Docker image for Kafka (Maestro orchestration) http://www.signalfuse.com)
- (See also A production ready Docker workflow. Part 3: Orchestration tools)
- Getting Started with CoreOS and Docker using Vagrant, by Luke Bond
- Deploying Docker Containers on a Vagrant CoreOS Cluster with fleet, by Luke Bond
- Vessel automates the setup & use of dockerized development environments

Using a private registry

- Docker Registry, Docker web site
- Deploying a registry server, Docker web site
- Running Secured Docker Registry 2.0, by Jaroslav Holub, April 28, 2015

- How To Set Up a Private Docker Registry on Ubuntu 14.04, by Nik van der Ploeg, October 15, 2014
- How to Secure Your Private Docker Registry, by Alex Welch, January 2, 2015 (goes with previous post)
- Setup Your Own Docker Registry on CoreOS, Vultr, May 7, 2015
- Setting Up Docker Private Registry, by beingasysadmin, January 14, 2015
- Docker: How to Use Your Own Private Registry, Twitter University, November 12, 2013

Configuration management and service discovery

- Demystifying Service Discovery under Docker Engine 1.12.0, by ajeetraina, July 27, 2016
- Consul ("Service discovery and configuration made easy. Distributed, highly available, and datacenter aware.")
- Consul Part 1: Service discovery, the easy way, YouTube video by OpsForce, July 17, 2016
- Consul Part 2: Service health and templates, YouTube video by OpsForce, August 3, 2016
- Configuration management with Consul, by Michael de Jong, October 26, 2015
- **Configuration management from Git to Consul, by Ryan Breen, May 8, 2015**
  - GitHub Cimpress-MCP/git2consul ("Mirrors the contents of a git repository into Consul KVs.")
  - GitHub Cimpress-MCP/fsconsul ("Write Consul K/Vs to the filesystem.")
  - GitHUb Cimpress-MCP/gosecret ("A Go library for encrypting and decrypting slices of byte arrays.")
- Distributed Configuration Management and Dark Launching Using Consul, by Bill Monkman, November 26, 2014 (Has examples of creating configuration management kv store)
- GitHub kelseyhightower/confd ("Manage local application configuration files using templates and data from etcd or consul")
- An Introduction to Using Consul, a Service Discovery System, on Ubuntu 14.04 (Part 1 of 3), by Justin Ellingwood, August 15, 2014
- How to Configure Consul in a Production Environment on Ubuntu 14.04 (Part 2 of 3), by Justin Ellingwood, August 15, 2014
- How To Secure Consul with TLS Encryption on Ubuntu 14.04 (Part 3 of 3), by Justin Ellingwood, August 15, 2014
- Understanding Modern Service Discovery with Docker, by Jeff Lindsay
- Consul Service Discovery with Docker, by Jeff Lindsay
- Automatic Docker Service Announcement with Registrator, by Jeff Lindsay
- A High Available Docker Container Platform Using CoreOS And Consul, by Mark van Holsteijn
- GitHub democracyworks/consul-coreos ("Bootstraps a Consul cluster on CoreOS using fleet and etcd")
- Simulating service discovery with Docker and etcd, by Aaditya Talwai
- GitHub Cimpress-MCP/git2consul ("Mirrors the contents of a git repository into Consul KVs.")
- GitHub ianbytchek/docker-coreos-ansible-toolbox ("Using Ansible with CoreOS? Use CoreOS Ansible toolbox!")

Networking and Docker containers

- **Docker native networking**
  - Docker Stacks and Attachable networks, by Alex Ellis, February 17, 2017
  - Docker 1.12 Swarm Mode Deep Dive Part 1: Topology, YouTube by Andrea Luzzardi, July 28, 2016
  - Splendors and Miseries of Docker Network, by Aleksandr Tarasov, November 16, 2015
  - Add Docker Machine to Swarm cluster after creation, stackoverflow thread, January 4, 2016
  - Docker Networking takes a step in the right direction, Docker blog, April 30, 2015
- **Weave**
  - GitHub weaveworks/weave ("Simple, resilient multi-host Docker networking http://weave.works")
  - Multi-host Docker deployment with Swarm and Compose using Weave 0.11, by errordeveloper, May 27, 2015
  - Networking Docker Containers with Weave on CoreOS, Weave web site
  - Using Weave Scope Standalone to Visualize and Monitor Docker Containers, Weave web site
  - Using Docker Machine and Swarm with Weave 0.10, by errordeveloper, May 6, 2015
  - How to set up networking between Docker containers, by Dan Nanni, March 20, 2015
  - **Using Docker Machine with Weave 0.10, by Ilya Dmitrichenko, April 22, 2015**
    - GitHub infrabricks/powerstrip-demo (This is a TLS powerstrip weave demo, installed with docker-machine!)
  - **Elasticsearch, Weave and Docker, by errordeveloper, January 20, 2015**
    - GitHub errordeveloper/weave-demos/hello-apps/elasticsearch-js/scripts/run_elasticsearch_2_clusters.sh
    - **Curator: Tending your time-series indices, by Aaron Mildenstein, January 20, 2014**
      - GitHub elastic/curator (Curator: Tending your Elasticsearch indices")
    - Logstash how to remove old logs, by Andriy Podanenko
  - Deploying and migrating an Elasticsearch-Logstash-Kibana stack using Docker Part 1, by Ryan Wallner, January 12, 2016
  - Deploying and migrating an Elasticsearch-Logstash-Kibana stack using Docker Part 2, by Ryan Wallner, January 12, 2016
  - I just created a Cassandra cluster that spans 3 different network domains, by using 2 simple shell commands. How cool is that?, by Yaron Rosenbaum, October 8, 2014
  - Running a Weave Network on CoreOS, by errordeveloper, October 28, 2014
  - How to Network Docker Containers with Weave, by Benjamin Ball, September 14, 2014
  - Getting Started with Weave and Docker on CoreOS (GitHub fintanr/weavegs)
  - Docker, Weave, Raspberry Pi and a bit of Networked Cloud Computing!, by Alexander Grendel, December 9, 2014
- **Pipework**
  - GitHub jpetazzo/pipework (Software-Defined Networking tools for LXC (LinuX Containers))
  - Advanced Docker Networking with Pipework, by Sam Leathers

- **Flannel**
  - CoreOS + Layer Meetup 10/6/14: Brian "Redbeard" Harrington - Warming Up With Flannel, YouTube, October 7, 2014
  - Docker Networking - CoreOS Flannel, Sreenivas Makam, January 18, 2015
- **OpenVPN**
  - How To Run OpenVPN in a Docker Container on Ubuntu 14.04, by Kyle Manna, February 2, 2015
  - Start an OpenVPN Container as a systemd managed service under CoreOS, by c-garcia, March 15, 2015
  - OpenVPN in a container, by Ed Vielmetti, November 20, 2015
  - Securing wifi traffic with OpenVPN and Docker, by Christopher Bunn, April 03, 2015
  - **Docker + Joyent + OpenVPN = Bliss, by Jérôme Petazzoni, September 10, 2013**
    - GitHub jpetazzo/dockvpn ("Recipe to build an OpenVPN image for Docker")
- Integrating Proxy With Docker Swarm (Tour Around Docker 1.12 Series), by Viktor Farcic, August 1, 2016 [Good networking diagrams]
- Docker networking overview, by Filip Verloy, February 17, 2016
- Docker Swarm and experimental multihost networking with docker-machine and boot2docker, by IIkka Anttonen, July 15, 2015
- Three Solutions to Bi-directional Linking Problem in Docker Compose, by Abdelrahman Hosny, July 1, 2015
- Docker Networking Meetup - Intro to Weave/Flannel, by Dhananjay 'DJ' Sampath, January 23, 2015
- 5 ways Docker is fixing its networking woes, by Serdar Yegulalp, October 20, 2014
- Docker networking - IP per container, one /24 per pod(worker), by Vicente De Luca, April 30, 2015
- See also: Network Configuration

Persistent storage in Docker containers

- Docker: Storage Patterns for Persistence, by Karim Vaes, February 11, 2016
- Comprehensive Overview of Storage Scalability in Docker, by Jeremy Eder, September 30, 2014
- Storage Concepts in Docker: Shared Storage and the VOLUME directive, by Mark Lamourine, October 7, 2014
- Storage Concepts in Docker: Persistent Storage, by Mark Lamourine, October 10, 2014 [Has a good discussion of SELinux labeling of directories for use by Docker containers]
- Share disks through NFS on a CoreOS cluster?, stackexchange post, November 25, 2014
- Enabling and Mounting NFS on CoreOS, by Scott Lowe, February 20, 2015
- Exploring Docker Volumes for Phases of Development, by Alan Kent, May 31, 2015
- Quickly build arbitrary size Hadoop Cluster based on Docker, by KiwenLau, May 29, 2015
- See also: Hadoop section of Databases and database tools

Persistent services within Docker containers

- Automatically start containers, Docker web site
- Introducing dumb-init, an init system for Docker containers, by Chris K., Jan 6, 2016
- Docker All The Things: Nginx And Supervisor, by Matthew McKeen, December 14, 2013
- Running node and nginx in docker with supervisor, stackoverflow post, December 2, 2014
- Creating a Docker Container to run PHP, NGINX and Hip Hop VM (HHVM), July 15, 2014
- Roll your own Docker registry with Docker, Compose, Supervisor, and Nginx, by Philipp Wintermantel, March 18, 2014

Clustering Docker containers

- Using Docker Stack And Compose YAML Files To Deploy Swarm Services, by Viktor Farcic, January 23, 2017
- Running a Small Docker Swarm Cluster, Scott Lowe, March 6, 2015
- Docker Machine, Compose & Swarm, by MediaGlasses
- Clustering Using Docker Swarm 0.2.0, by Arun Gupta
- A quick overview of Docker Swarm, by Olivier Robert, February 2, 2016

Logging/monitoring activity of containers

- Collecting All Docker Logs with Fluentd, by Kiyoto Tamura, July 7, 2015
- Automating Docker Logging: ElasticSearch, Logstash, Kibana, and Logspout, by Nathan LeClaire, April 27, 2015
- Running High Performance and Fault Tolerant Elasticsearch Clusters on Docker, sematext group, November 24, 2015
- Scalable Docker Monitoring with Fluentd, Elasticsearch and Kibana 4, by manu, November 21, 2014
- Performance tuning ELK stack, by Josh Reichardt, March 30, 2015
- syslog logging driver for Docker, by Mark Wolfe, May 3, 2015
- **Real-time monitoring of Hadoop clusters, by Attila Kanto, October 7, 2014**
  - Apache Hadoop 2.6.0 on Docker, by Janos Matyas, September 15, 2014

Cleaning up (or keeping Docker images as small as possible)

- Reducing the size of Docker Images, by Richard Woudenberg, March 18, 2015
- Making Debian Docker Images Smaller, by Dave Beckett, April 18, 2015
- How I shrunk a Docker image by 98.8% – featuring fanotify, by Jean-Tiare LE BIGOT, April 25, 2015
- Low on disk space, cleaning up old Docker containers, by johnarce, May 20, 2014
- Squashing Docker Images, by Jason Wilder, August 19, 2014
- Optimizing Docker Images, by Brian DeHamer, July 28, 2014
- Create The Smallest Possible Docker Container, by Adriaan de Jonge, July 4, 2014

- How To Create The Smallest Possible Docker Container Of Any Image, by Mark van Holsteijn, June 30, 2015

Docker and Continuous Integration

- Docker Jenkins, by Yuri Kushch, February 17, 2017
- Easy CD-CI with Jenkins, Docker Swarm and Docker secrets, by KrazY CoCoon Lab, March 6, 2017
- Fully automated development environment with docker-compose, by Andrew Orsich, February 22, 2017
- Continuous Integration, Delivery or Deployment with Jenkins, Docker and Ansible by Viktor Farcic, February 11, 2015
- Deployment Pipeline using Docker, Jenkins, Java and Couchbase, by Arun Gupta, September 9, 2016
- **Scaling to Infinity with Docker Swarm, Docker Compose and Consul, by Viktor Farcic, July 2, 2015**
  - (Part 1/4) - A Taste of What is to Come
  - (Part 2/4) - Manually Deploying Services
  - (Part 3/4) - Blue-Green Deployment, Automation and Self-Healing Procedure
  - (Part 4/4) - Scaling Individual Services
- **Putting Jenkins in a Docker Container, by Maxfield F. Stewart, Riot Games Engineering blog**
  - GitHub maxfields2000/dockerjenkins_tutorial ("A repository for items learned in my Getting Started with Jenkins and Docker tutorial series")
- Delivering eBay's CI Solution with Apache Mesos – Part I, by The eBay PaaS Team, April 4, 2014
- Delivering eBay's CI Solution with Apache Mesos – Part II, by The eBay PaaS Team, May 12, 2014
- DockerCon Video: Delivering eBay's CI Solution with Apache Mesos & Docker, by Victor Coisne, June 23, 2014
- Disaster-proofing slaves with Docker Swarm and the CloudBees Jenkins Platform, by Tracy Kennedy, June 19, 2015
- Continuous Integration and Delivery with Docker, by Jaroslav Holub, May 28, 2015
- Jenkins, Docker Hub (Official Jenkins Docker Image)
- Running Jenkins in Docker Containers, by Peter Sellers, February 11, 2015
- Import Jenkins Configuration to a dockerized jenkins, by Allan Espinoza, January 25, 2015
- Docker in Docker with Jenkins and Supervisord, by Johan Haleby, March 14, 2015
- Jenkins in a Docker container, by by Thomas Einwaller, September 1, 2014

Miscellaneous Docker related articles and blog posts

- Consul Template for transparent load balancing of containers, by Jose Luis Ordiales, April 1, 2015
- Orchestrating your containers with CoreOS, an introduction, by Jose Luis Ordiales, July 12, 2015
- 3 hours to Docker fundamentals: Jumpstart your Docker knowledge, by Aater Suleman, October 13, 2014
- Docker Registry richhaase/bigtop-hadoop
- Automated docker ambassadors with CoreOS + registrator + ambassadord

- Understanding user file ownership in docker: how to avoid changing permissions of linked volumes, stackoverflow
- GitHub signalfx/docker-java (Java Docker API Client)
- A production ready Docker workflow (part 1), by Luis Elizondo
- A production ready Docker workflow. Part 2: The Storage Problem, by Luis Elizondo
- A production ready Docker workflow. Part 3: Orchestration tools, by Luis Elizondo
- Your very own server with Docker, by Erwyn
- A Not Very Short Introduction to Docker, by Anders Janmyr
- Deploying NGINX and NGINX Plus with Docker, by Rick Nelson
- Make Docker Machine Do Anything With Our Experimental Extensions, by kcoleman, September 26, 2015
- Consul + Consul-Template with Docker-Compose, by Neil Ni, September 14, 2015
- Virtualenv the Docker way, by Justyna Ilczuk, Nov 16, 2014
- GitHub daniel-illi/docker-haproxy-letsencrypt (“Dockerized HAProxy with Let’s Encrypt integration”)
- GitHub BradJonesLLC/docker-nginx-letsencrypt (“Nginx container with Let’s Encrypt auto-renew “)

Operating systems for running Docker

CoreOS

- **CoreOS**
  - Container Overview, CoreOS web site
  - CoreOS Clustering, CoreOS web site
  - CoreOS Cluster Architectures, CoreOS web site
  - CoreOS Overview and Current Status, Slideshare by Sreenivas Makam, April 17, 2016
  - etcd, CoreOS web site
  - Running CoreOS on Vagrant, CoreOS web site
  - **Booting CoreOS via PXE, CoreOS web site**
    - GitHub kelseyhightower/coreos-ipxe-server (“CoreOS iPXE server”)
  - Customizing Docker, CoreOS web site
  - Deploying a Service Using fleet CoreOS web site
  - **Installing CoreOS to Disk, CoreOS web site**
    - GitHub nyarla/coreos-live-iso/makeiso.sh
  - Ignition: Better Machine Configuration (CoreOS Fest 2015), May 27, 2015 (has some details of boot sequence that are hard to learn)
  - **Lessons Learned From Building Platforms on Top of CoreOS (CoreOS Fest 2015), May 27, 2015**
    - 10 Lessons Learned Using CoreOS, by Gabriel Monroy (slides for CoreOS Fest 2015 talk)
- **Pay attention to these bug reports (and what is said in them)**

  - Network settings should be set in oem cloud-config.yml (coreos/bugs #11, April 28, 2014)
  - custom iptables - boot order, (coreos/bugs #58, June 26, 2014)
  - docker-tcp.socket fails: Socket service docker.service already active, refusing. (coreos/coreos-vagrant bug #172, September 30, 2014)
  - How does docker-tcp.socket actually enable Docker's remote API on CoreOS?, superuser post, January 5, 2015
- How can I customize bashrc, bash_profile or profile on a CoreOS installation?, Stackoverflow post by Richard, Jun 2 2015
- GitHub https://github.com/coreos/coreos-overlay/tree/master/app-shells/bash/files/
- CoreOS: Orchestrating the Fleet, YouTube video by Brian Waldon, July 8, 2014
- **Digital Ocean tutorial series: Getting Started with CoreOS**
  - How To Troubleshoot Common Issues with your CoreOS Servers, (part 8 of 9), September 18, 2015
  - How To Secure Your CoreOS Cluster with TLS/SSL and Firewall Rules, (part 9 of 9), December 7, 2015

OpenNode OS

- OpenNode OS ("Lightweight bare-metal cloud OS combining Linux Containers and KVM full virtualization options into payload optimized solution.")
- NodeFabric Host Image ("NodeFabric delivers hyperconverged database and storage solution for highly available, self-healing and load-balanced cloud services")

RancherOS

- RancherOS (A minimalist distribution of Linux designed from the ground up to run Docker containers.)
- Announcing RancherOS

Project Atomic

- Project Atomic (Trusted Distributions, Atomic Updates)

Ubuntu "Snappy"

- **Snappy Ubuntu Core**
  - SNAPPY UBUNTU CORE image for Raspberry Pi
- It's a Snap!
- Snappy Security

ArchLinux

- ArchLinux

8.2.2 Configuration management and automated provisioning

- Collins ("Infrastructure management for engineers")
- **GitHub coreos/etcd (A highly-available key value store for shared configuration and service discovery)**
    - Brandon Philips Explains etcd, by Phil Whelan
    - CoreOS: etcd 2.0, by Brandon Philips
- The Marriage of Ansible and Docker, by Zuletzt geändert von Unbekannter Benutzer, June 5, 2015
- docker, ansible and vagrant, by Wojtek Oledzki, January 1, 2015
- Ubuntu Cloud-Init Technology, YouTube video by ubuntucloud, December 8, 2010
- The Assimilation Project
- Ansible and Docker, by Ash Wilson
- **OpenStack**
    - **Building HA Clusters with Ansible and Openstack, by Remy van Elst, July 25, 2014**
        - GitHub RaymiiOrg/ansible (ansible/openstack-example)
    - Automating Openstack with cloud init run a script on VM's first boot, by Remy van Elst, March 11, 2015
    - Provisioning IaaS Clouds with Dynamic Ansible Inventories and OpenStack Metadata, by Lukas Pustina
    - GitHub ewindisch/dockenstack (OpenStack Devstack on Docker)
    - OpenStack: Docker wiki section from OpenStack
    - OpenStack is overkill for Docker: New tooling is necessary for effectively managing Docker at scale, by Matt Asay, August 10, 2015
    - Up and Running with Docker Machine and OpenStack, by Spencer Smith, Solinea blog, July 8, 2015
    - OpenStackClient and OpenStack Python SDK, youtube video by Dean Troyer, October 27, 2015
    - **Life Without DevStack: OpenStack Development With OSA, by Miguel Grinberg**
        - (Demonstration video)
- Agile Configuration Management – Intermezzo by Marcus Philip, Diabol
- Top 5 Open Source Linux Server Provisioning Software, by NIXCRAFT

8.2.3 Virtualbox

- Virtual Box Headless Cheatsheet: Headless Virtual Machine Install/Import and setup, by Tim Arneaud, October 9, 2012

8.2.4 Packer

- **Packer**
    - Command Line: Build
    - Debugging Packer Builds
- GitHub boxcutter/ubuntu ("Virtual machine templates for Ubuntu")

- GitHub tylert/packer-build ("Packer Automated VM Image and Vagrant Box Builds")
- Packer: In 10 minutes, from zero to bootable VirtualBox Ubuntu 12.04, by @kappataumu, September 8, 2013

8.2.5 Vagrant

- Vagrant
- How To Use Vagrant To Create Small Virtual Test Lab on a Linux / OS X / MS-Windows
- How to Create and Share a Vagrant Base Box, by George Fekete, July 17, 2014
- Vagrantfile Explained: Setting Up and Provisioning with Shell, by George Fekete, July 19, 2014
- Multi-Machine environment documentation from Hashicorp
- Using Vagrant and Ansible, Ansible documentation
- Change Insecure Key To My Own Key On Vagrant, by ermaker, November 18, 2015
- GitHub AAFC-MBB/vagrant-specify7 ("Package to launch a Specify7 instance in a Vagrant VM")
- IP lookup for Vagrant private networking, by wamonite, July 12, 2014
- Elegant virtualization with Vagrant, by zenonharley, July 21, 2015

8.2.6 Ansible

- **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
    * Jinja2 Template Inheritence
    * [jinja2] Help with blocks, Reddit /r/Ansible post by by FlowLabel
  - **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

8.2.7 Storing Secrets for Development and Configuration

- Managing Secrets In Docker Swarm Clusters, by Viktor Farcic, February 23, 2017
- Docker Compose v3.1 file format now supports Docker 1.13.1 Secret Management, by ajeetraina, February 15 2017
- Secrets at Scale: Automated Bootstrapping of Secrets and Identity in the Cloud, by Ian Haken, USENIX, January 30, 2017
- Ask HN: In a microservice architecture, how do you handle managing secrets?, HackerNews post, January 9, 2016
- **Variable separation using Ansible**
    - Variable File Separation, Ansible documentation
    - How to open source provisioning script yet keep secrets secret? Is a two repository approach recommended?, self.ansible post by sovietmudkipz
- Ansible Vault
    - Safely storing Ansible playbook secrets, On Web Security blog, June 23, 2015
    - Best Practices: Variables and Vaults, Ansible web site
    - Ansible: Using Vault, video tutorial by ServersForHackers, Feb 16, 2015
    - Managing Secrets with Ansible Vault – The Missing Guide (Part 1 of 2), by Dan Tehranian, July 24, 2015
    - Managing Secrets with Ansible Vault – The Missing Guide (Part 2 of 2), by Dan Tehranian, July 24, 2015
    - Ansible: How to encrypt some variables in an inventory file in a separate vault file?, Stackoverflow post by Adam Matan, May 13 2015
    - GitHub Gist tristanfisher/Ansible-Vault how-to.md ("A short tutorial on how to use Vault in your Ansible workflow. Ansible-vault allows you to more safely store sensitive information in a source code repository or on disk.")
    - Encrypting Login Credentials in Ansible Vault, by Dan Tehranian, August 17, 2015
- Hashicorp Vault
    - Introduction to Vault, YouTube video by Seth Vargo, December 24, 2015
    - Vault: A tool for managing secrets, by Mitchell Hashimoto, April 28, 2015
    - 2 Key-Value Stores compared - Keywhiz (Square) & Vault (Hashicorp), YouTube video, June 12, 2015
    - GitHub hashicorp/vault ("A tool for managing secrets. http://vaultproject.io")
    - Storing Secrets at Scale with HashiCorp's Vault: Q&A with Armon Dadgar, by Daniel Bryant, September 9, 2015

  - GitHub jhaals/ansible-vault ("ansible lookup plugin for secrets stored in Vault by HashiCorp")
  - Managing all your secrets with Vault - Review and Walkthrough, by Martin Rusev, January 29, 2016

- **Consul**
  - Crypt: Store and retrieve encrypted configuration parameters from etcd or consul

8.2.8 Terraform

- GitHub hashicorp/terraform ("Terraform is a tool for building, changing, and combining infrastructure safely and efficiently. https://www.terraform.io/")
- Seth Vargo on Hashicorp Terraform, YouTube video by Seth Vargo, February 17, 2015
- Terraform Deploying Consul Cluster in One Command, Vimeo video from HashiCorp
- How To Use Terraform with DigitalOcean, by Mitchell Anicas, September 25, 2017
- Getting Started with Terraform for Digitalocean, by Eric Wright, January 9, 2017
- Terraform Infrastructure Design Patterns, by Bart Spaans, September 14, 2015
- Using Vagrant, Docker, & Terraform to streamline your development & demo environments, YouTube video by Nikhil Vaze, October 18, 2015
- Orchestrating Docker with Terraform and Consul by Mitchell Hashimoto, YouTube video, January 7, 2015
- Guide to automating a multi-tiered application securely on AWS with Docker and Terraform, by Greg Osuri
- ansible stuffs - how to securely get SSH fingerprints/public keys from newly created AWS instances, by Bill Cawthra, September 8, 2016

8.2.9 Nomad

- Nomad, by Armon Dadgar, September 28, 2015
- Introduction to Nomad, nomadproject.io web site
- GitHub hashicorp/nomad ("A Distributed, Highly Available, Datacenter-Aware Scheduler https://www.nomadproject.io/")

8.2.10 Otto

- Otto: Development and Deployment Made Easy ottoproject.io web site
- First look at HashiCorp's Otto, YouTube video by Mat Schaffer, September 29, 2015
- Mitchell Hashimoto - Introducing Nomad and Otto, YouTube video, October 27, 2015
- Otto: Getting Started, ottoproject.io web site
- GitHub hashicorp/otto ("Development and deployment made easy. https://ottoproject.io")

8.2.11 Automated distributed system deployment options using Ansible

- ansible-dims-playbooks (ansible-dims-playbooks Documenations)
- Mantl (*Mantl Documentation*)_

- DC/OS
- Intel Trusted Analytics Platform
- Debops
- The Fedora Project ansible playbook/files/etc repository for fedora infrastructure

8.2.12 Using `systemd` and `upstart` for services

- Demystifying systemd: A Practical Guide, by Ben Breard and Lennart Poettering, Red Hat Summit, April 14-17, 2014 (PDF file)
- systemd for Administrators, eBook by psankar
- Systemd Essentials: Working with Services, Units, and the Journal, Justin Ellingwood, DigitalOcean, April 20, 2015
- How To Use Systemctl to Manage Systemd Services and Units, Justin Ellingwood, DigitalOcean, February 1, 2015
- Understanding Systemd Units and Unit Files, Justin Ellingwood, DigitalOcean, February 17, 2015
- How to debug Systemd problems, Fedora Project web site
- Auditing systemd: solving failed units with systemctl, by Michael Boelen, November 24, 2014
- systemd.service — Service unit configuration, Freedesktop.org man page
- Investigating systemd errors, ArchLinux web page
- SystemdForUpstartUsers, Ubuntu web site
- Gist damncabbage/unicorn.conf ("I have the most terrible Upstart config for Unicorn. Support start, stop, restart and reload (rolling restart).")
- How can I tell upstart to restart a service when a different service is restarted?, by Marius Gedminas, AskUbuntu post, February 13, 2013
- **Logging**
  - Reading the System Log, CoreOS web site
  - How To Use Journalctl to View and Manipulate Systemd Logs, Justin Ellingwood, DigitalOcean, February 5, 2015
  - GitHub systemd/journal2gelf ("Ships new systemd journal entries to a remote destination in Graylog Extended Log Format (GELF)")
  - Sending CoreOS Logs to Loggly, by Garland Kan, October 27, 2015
  - Platform logging, Deis
  - Responsive infrastructure - How to setup rsyslog, elasticsearch and kibana on CoreOS and AWS (2), by Simon Dittlman, April 3, 2015
  - A UDP syslog client in Python — for Windows and UNIX, by Christian Stigen Larsen, December 3, 2013

8.2.13 Small form-factor hardware systems

- **Raspberry Pi**
  - Raspberry Pi Wiki Hub

- RPi SD cards
- **How to Flash an SD Card for Raspberry Pi, by Johnny Winters**
  - GitHub hypriot/flash ("Command line script to flash SD card images for the Raspberry Pi")
- **Docker Pirates ARMed with explosive stuff (hypriot blog)**
  - Getting started with Docker on your Raspberry Pi, hypriot blog, October 13, 2015
  - Let Docker Swarm all over your Raspberry Pi Cluster, by Stefan, July 3, 2015
  - Heavily ARMed after major upgrade: Raspberry Pi with Docker 1.5.0, March 3, 2015
  - How to use Docker Compose to run complex multi container apps on your Raspberry Pi, April 6, 2015
- **Using a Raspberry Pi as a PXE boot server**
  - Raspberry Pi as a PXE, TFTP, NFS, proxy DHCP server, by Adam Niedzwiedzki, February 7, 2014
  - Network booting machines with a PXE server running in a Docker container, by Jérôme Petazzoni, December 7, 2013
  - Raspberry Pi as PXE Server, by Cody Bunch, August 19, 2014
  - How To: Setup a PXE Boot Server on Debian Part 2, May 9, 2011
- Raspbian Image with Docker 1.5.0 Released for Raspberry Pi Boards, by cnxsoft
- Docker on Raspberry Pi in 4 Simple Steps, by resin.io
- Visualize your Raspberry Pi containers with Portainer or UI for Docker, by Stefan, October 31, 2016
- Cloudy with a chance of Raspberries, by Matt Williams
- Swarming Raspberry Pi - Part 1, by Matt Williams
- **Let's build a PicoCluster for Docker Swarm, Hypriot blog, Mar 23, 2016**
  - 5 Node PicoCluster (Raspberry PI), PicoCluster web site
  - DIY 5 Node Cluster of Raspberry Pi 3s, by Nick Smith, April 2016
- OpenVPN with 2 factor authentication on the Raspberry Pi, by Mark, Coder36 blog, 15 December 2014

- **UP**
  - UP Board
- **Beaglebone Black**
  - Using Docker on Beaglebone Black, by Mark Fink
  - Snappy Ubuntu for Devices - The Year of the Linux Countertop!
- **CuBox-i**
  - CuBox-i Mini Computer for XBMC/Kodi player, Android TV Box and Linux
  - ArchLinux is one of the available operating systems from Solid Run
- **Clustering small form-factor devices**
  - #Building ARM cluster Part 1: Collecting , wiring and powering devices, by Mateusz Kaczanowski
  - #Building ARM cluster Part 2: Create and write system image with goback!, by Mateusz Kaczanowski

  - #Building ARM cluster Part 3: Docker, fleet, etcd. Distribute containers!, by Mateusz Kaczanowski

8.2.14 Miscellaneous Distributed System Construction

- GitHub uw-dims/ansible-dims-playbooks ("DIMS Ansible playbooks")
- GitHub trustedanalytics ("Components for Trusted Analytics Platform")
- DebOps
- The Open Home Lab Stack, by Mighty Womble, October 1, 2017
- Boot my (secure)->(gov) cloud, by Nicki Watt, August 10, 2015

8.2.15 Scripting in `bash`

- Basic grammar rules of Bash, BashHackersWiki
- **Commandlinefu.com**
  - Find Duplicate Files (based on size first, then MD5 hash)
  - Recursively remove all empty directories
  - **GitHub google/styleguide ("Style guides for Google-originated open-source projects")**
    - Shell Style Guide
- **Command line option parsing**
  - GitHub kward/shflags (Automatically exported from https://code.google.com/p/shflags)
  - Easy Bash Scripting With Shflags, by Steve Francia, July 8, 2011
  - Using getopts in bash shell script to get long and short command line options, Stackoverflow post
- **Advanced Bash scripting**
  - How "Exit Traps" Can Make Your Bash Scripts Way More Robust And Reliable, by Aaron Maxwell
  - I set variables in a loop that's in a pipeline. Why do they disappear after the loop terminates? Or, why can't I pipe data to read?, BasFAQ/024
  - The Ultimate Bash Array Tutorial with 15 Examples, by Sasikala on June 3, 2010
  - BashGuide/Arrays
- **Debugging Bash scripts**
  - Debug your shell scripts with bashdb, by Ben Martin, November 24, 2008
  - Debugging a script, Bash Hackers Wiki
  - Why does my shell script choke on whitespace or other special characters?, StackExchange post by Gilles, May 24 2014

8.2.16 Postfix

- Postfix Configuration Parameters, postfix web site
- Postfix Standard Configuration Examples, postfix web site
- Basic settings in the Postfix main.cf file, Rackspace web site, December 29, 2015

- Setting up a mail server using Postfix in 5 minutes, by Rudd-O
- Question about DNSmasq and MX records, FreeBSD forums post by danaeckel, March 12, 2013

8.2.17 Dell Edge Server Stuff

- Problem with PERC H710 Raid Controller
- Building the LSI 9285-8e Driver for Openfiler 2.99, windowsmasher blog
- Dell Driver Details: Linux SLES10 SP4 Driver 00.00.05.39-1 for PERC H310 / H710 / H710P / H810 Controllers.

8.2.18 Software Engineering Best Practices

- **GitHub linuxfoundation/cii-best-practices-badge ("Core Infrastructure Initiative Best Practices Badge https://bestpractic**
    - Best Practices Criteria for Free/Libre and Open Source Software (FLOSS) (version 0.5.0)
    - Security
    - Implementation
- High Cohesion, Loose Coupling, the Bojan's Blog, April 8, 2015
- **Security for developers, Mediawiki**
    - Security for developers/Architecture, Mediawiki

8.2.19 Programming in Python

Learning Python

- Learn Python the Hard Way
- Python.org (official online documentation and tutorials)
- **Video tutorials**
    - GitHub s16h/py-must-watch ("Must-watch videos about Python: Inspired by js-must-watch. Create pull requests to add more awesome links :-)")
    - Transforming Code into Beautiful, Idiomatic Python, YouTube video by Raymond Hettinger, March 20, 2013
    - Python's Class Development Toolkit, YouTube video by Raymond Hettinger, March 20, 2013
    - **Raymond Hettinger - Beyond PEP 8 – Best practices for beautiful intelligible code, PyCon 2015**
        - Slides can be found at: https://speakerdeck.com/pyconslides/transforming-code-into-beautiful-idiomatic-python-by-raymond-hettinger-1
        - Some of the code can be found here: http://msoucy.me/2015/05/pycon-2015-beyond-pep8/
    - Raymond Hettinger - Super considered super!, PyCon 2015
- Dive Into Object-oriented Python, by Leonardo Giordani, April 17, 2016
- 30 Python Language Features and Tricks You May Not Know About, by Sahand Saba, May 19, 2014

General Python references

- Python syntax and semantics, Wikipedia
- **The Hitchhiker's Guide to Python**
  - *Code Style*
  - Structuring Your Project
- Open Sourcing a Python Project the Right Way, by Jeff Knupp, August 16, 2013
- Cookiecutter: Project Templates Made Easy, by Daniel Roy Greenfeld, August 17, 2013
- Hacking Secret Cyphers with Python
- Logilab Python Tools
- pip
- Useful Python Functions and Features
- Code Academy - Learn to code for free
- Check IO - Learn Python by playing a game
- Anaconda (Completely free enterprise-ready Python distribution for large-scale data processing, predictive analytics, and scientific computing)

Advanced Python Programming

- How to Create A New Python Module (and deploy it using pip), by liranbh, January 2017
- A guide to Python's function decorators, by Ayman Farhat, January 2014
- Intro to some Python dependencies of OpenStack, by Run for the Hills
- Easy, clean, reliable Python 2/3 compatibility, by Python Charmers Pty Ltd
- Easy Signal Handling for Python Daemons, Jeff Hull, August 30, 2012
- The Top Mistakes Developers Make When Using Python for Big Data Analytics, by Karolina Alexiou
- Don't Slurp: How to Read Files in Python, by mssaxm, September 27, 2013 by mssaxm
- Python: Multiprocessing large files, by Nick, March 29, 2012
- Dynaconf - Let your settings to be Dynamic, by Bruno Rocha, February 14, 2016
- **Microservices with Python, RabbitMQ and Nameko, by Bruno Rocha, March 4, 2016**
  - nameko
- How to use findall, finditer, split, sub, and subn in Regular Expressions, by mamikon, 2013
- **Pandas**
  - Multi-Processing With Pandas, by Gouthaman Balaraman, November 19, 2014
  - Pandas: Data Analysis with Python, by Carsten Schnober
  - Exploring U.S. Traffic Fatality Data (using Python and Pandas), by Ben Van Dyke, December 16, 2015
- Adventures in Optimizing Text Processing (Lessons learned while post-processing 1.75 billion lines of Hadoop output), by Michael Richman, January 21, 2014
- Git revision numbers for setuptools packages, by Christopher Berner, February 26, 2012

- A template for Python packages, including a sound setup.py and Makefile for tagging releases by hcarvalhoalves
- How To Quickly Compute The Mandelbrot Set In Python, by Jean Francois Puget, December 28, 2015
- A Globals Class pattern for Python, by Steve Ferg, October 20, 2010
- GitHub Yelp/undebt ("A fast, straightforward, reliable tool for performing massive, automated code refactoring")
- Create, use, and remove temporary files securely, Openstack web site

Command line programs in Python

- Writing Awesome Command-Line Programs in Python, YouTube video from EuroPython 2014, July 24, 2014
- GitHub openstack/cliff ("Command Line Interface Formulation Framework http://openstack.org")
- **[openstack-dev] [oslo][openstackclient] transferring maintenance of cliff to the OpenStack client team, by Doug Hellmann**
    - **Oslo (OpenStack Common Libraries)**
        - GitHub OpenStack Oslo repos
    - cmd2
    - Python Apps the Right Way: entry points and scripts, by Chris Warrick, September 15, 2014
- worked examples of argparse and python logging, Gist by olooney
- Accessing value from either os.environ or argparse, Stackexchange post

Python IDEs, shells, and debuggers

- **PyCharm**
    - PyCharm Overview, YouTube video by JetBrains, May 5, 2014
    - 9 Reasons You Should be Using PyCharm, by Michael Kennedy, November 19, 2015
    - How to Get Started with PyCharm and Have a Productive Python IDE, by Pedro Kroger
    - Productive Python with PyCharm, by Paul Everitt, 2015
    - Install PyCharm on Ubuntu Linux, by Akbar S. Ahmed, February 10, 2015
    - Debug Support in Interactive Python Console, YouTube video by JetBrains, May 16, 2014
    - Configure PyCharm to use virtualenv, by Akbar S. Ahmed, February 10, 2015
    - An Epic Review of PyCharm 3 from a Vim User's Perspective, by Andrew Brookins
    - How do not waste time in PyCharm, YouTube vide by Paqmind, October 18, 2014
- IPython
- full: April 2014 Meetup - Matt Boehm and Python debuggers, DCPython tutorial session

Python virtual environments

- virtualenv source code
- Virtual Environments, Python Guide docs

- Python Power Tools: virtualenv video by Tuts+ Code
- Python Power Tools: virtualenvwrapper video by Tuts+ Code
- Python: pyenv, pyvenv, virtualenv – What's the Difference?, by Abu Ashraf Masnun, April 10, 2016
- **Reverse-engineering Ian Bicking's brain: inside pip and virtualenv, presented by Carl Meyer at PyCon 2011, Atlanta**
    - Slides of Carl Meyer's presentation
- Configure PyCharm to use virtualenv, by Akbar S. Ahmed, February 10, 2015
- Development Environment in Python - Part 1: Virtualenv and Pip, by Michael Herman
- Development Environment in Python - Part 2: Git and Github, by Michael Herman
- Running IPython cleanly inside a virtualenv
- Automatically load a virtualenv when running a script, stackoverflow, May 15, 2014
- Virtualenv's bin/activate is Doing It Wrong, datagrok / gist:2199506, July 3, 2014
- Virtualenv the Docker way, by Justyna Ilczuk, Nov 16, 2014

8.2.20 Testing and Test Automation

- Ned Batchelder: Getting Started Testing - PyCon 2014, PyCon 2014
- **Robot Framework**
    - How To Configure PyCharm for Robot Framework, YouTube video by KWoodScreencast, July 2014
    - Robot Framework Tutorial 2- First Script
- **GitHub sstephenson/bats ("Bats: Bash Automated Testing System")**
    - Testing Bash scripts with Bats, by Spike Grobstein, September 7, 2013
    - How to use Bats to test your command line tools, engineyard, April 29, 2014
    - GitHub duggan/pontoon/test/bats
    - GitHub jenkinsci/docker/tests
    - More Projects using Bats

8.2.21 Sphinx + reST

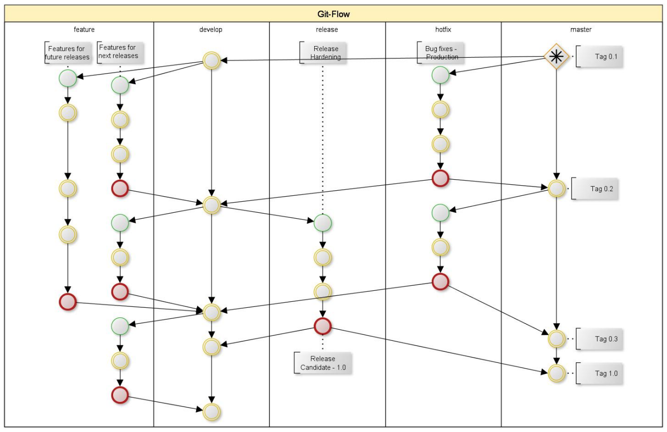
- **Sphinx**
    - Sphinx Style Guide
    - Sphinx Tutorial v0.1 talk by Brandon Rhodes from PyCon 2013
    - Documenting Your Project Using Sphinx
    - Easy and beautiful documentation with Sphinx
- **Restructured Text (reST) and Sphinx CheatSheet**
    - The reStructuredText_ Cheat Sheet: Syntax Reminders
    - Tables
- reStructuredText tool support

- Quick reStructuredText
- ReadTheDocs
- Don't be afraid to commit
- Sphinx Autodoc Tutorial for Dummies
- How to generate sphinx documentation for python code running in an embedded system
- Problems with StructuredText
- Automatically Generating Documentation for All Python Package Contents, stackoverflow.com
- **sphinx.ext.intersphinx – Link to other projects' documentation**
  - GitHug Gist shimizukawa/conf.py (Sphinx: Link to outer non-sphinx document by using intersphinx. https://groups.google.com/d/topic/sphinx-users/P_FolrZVoNg/discussion)
- Python Continuous Documentation
- sphinxcontrib-bibtex
- sphinx-contrib at Bitbucket
- StackOverflow thread: Using Sphinx to write personal websites and blogs
- ABlog for Sphinx
- **sphinx.ext.graphviz – Add Graphviz graphs**
  - Google search for 'graphviz dot tutorial'

8.2.22 Git

- Git
- Git tips from the trenches, February 1, 2014
- Learn Git Branching (really cool animated tutorial, visualizing Git concepts)
- Git Interactive Rebase, Squash, Amend and Other Ways of Rewriting History, by Tute Costa, November 3, 2014
- Git: Using topic branches and interactive rebasing effectively, by kumar303
- Git - When to Merge vs. When to Rebase, by Derek Gourlay, February 15, 2016
- **Viewing Git history**
  - 2.3 Git Basics - Viewing the Commit History, the Git Book
  - gitk
  - GitHub esc/git-big-picture
  - Visualizing branch topology in git, Stackoverflow post by Benjol, December 3, 2009
- **Git merging and conflict resolution**
  - kdiff3
  - **meld**
    - Git merging using Meld, Stackoverflow thread, June 21, 2012
    - Three way git merging with meld, by Lukáš Zapletal, September 17, 2012
    - Painless Merge Conflict Resolution in Git, by Kevin Wu Won, September 14, 2010

- **Oops. I didn't mean to do that… (Ugh!)**
  - How to undo (almost) anything with Git, GitHub blog post by jaw6, June 8, 2015
  - Undo Almost Anything with Git webinar, YouTube video by Peter Bell and Michael Smith, February 11, 2014
  - Undoing Changes, Git tutorial, attlasian.com
- **GitHub for hosting your Git repositories**
  - Fork A Repo (GitHub Help)
  - Forking Projects, (GitHub Guides)
  - Syncing a fork (GitHub Help)
  - Pushing to a remote (GitHub Help)
- **git-flow utilities to follow the Vincent Dreisen branching workflow**
  - Getting Started – Git-Flow, yakiloo



- hub (Git front end for GitHub)
- **HubFlow (GitFlow for GitHub)**
  - Using GitFlow with GitHub (Note: At this time, this workflow is designed for developers who belong to the same GitHub organisation. Although you are welcome to use it for opensource projects, it is designed for companies like DataSift who are using private repos on GitHub.)
  - Versioning
  - GitHub Flow Like a Pro with these 13 Git Aliases
  - Development flow with git flow, github, by Vysakh Sreenivasan
  - GitFlow: safely merge develop changes to a feature branch, stackoverflow thread
- Comparing Workflows, Atlassian Tutorials
- **bumpversion (can flexibly increment version numbers and commit/tag in Git)**
  - bumpversion screencast showing bumpversion in action
  - A Python Versioning Workflow With Bumpversion
  - PEP 440 – Version Identification and Dependency Specification (Python versioning specification that addresses several limitations of the previous attempt at a standardized approach to versioning, as described in PEP 345 and PEP 386.)

- **Git aliases and advanced tricks**
  - Includes in your git config, by Aurélien's room, December 5, 2015
  - Using git's include for private configuration information (like github tokens)
  - Protect Your Private Data Using Gitconfig Include Directive, Basetta.Bz, January 28, 2013
  - One weird trick for powerful Git aliases, by Nicola Paolucci, October 3, 2014
  - GitHubGist mwhite/git-aliases.md (The Ultimate Git Alias Setup)
  - Must Have Git Aliases: Advanced Examples, by @durdn
  - GitHubGist mcxiaoke/.gitconfig
  - What are your favorite Git aliases?, Reddit `r/git` post,
  - SCM Breeze (GitHub Gist *ndbroadbent/scm_breeze* ("Adds numbered shortcuts to the output git status, and much more http://madebynathan.com/2011/10/18/git-shortcuts-like-youve-never-seen-before/"))
  - **Encrypting some/all of a Git repository**
    * encrypted repositories?, Gmane thread
    * GitHub AGWA/git-crypt ("Transparent file encryption in git https://www.agwa.name/projects/git-crypt/")
    * GitHub Gist shadowhand/encrypted-git-repo.md ("Transparent Git Encryption")
  - 7.14 Git Tools - Credential Storage, Git book
- **Mac OS X and Windows issues with Git**
  - Git on Mac OS X: Don't ignore case!, by Howard Lewis Ship, July 28, 2010
  - How do I commit case-sensitive only filename changes in Git?, Stackoverflow, July 16, 2013
  - Case sensitivity in Git, Stackoverflow, January 18, 2012 (has solution for Mac OS X users)
- version.py Python script for getting version number from Git to use in `setup.py`
- **Advanced tasks (DANGER)**
  - How to delete files permanently from your local and remote git repositories, by Anoopjohn, February 20, 2014
  - GitHub aaronzirbes/shrink-git-repo.sh ("This script will help you remove large files from your git repo history and shrink the size of your repository.")
  - How to Shrink a Git Repository, by Steve Lorek, May 11, 2012

8.2.23 Agile/Scrum

- Agile methodology
- Scrum methodology
- Learn Scrum in 7 Minutes!, by techexcel, (YouTube video)
- Scrum and Kanban, Kanbantool,
- Introduction to Scrum - CollabNet Scrum Training Part 1, YouTube
- Jira Agile
- **Humor about Agile/Scrum**

  - I Need Agile Methodology, YouTube [Play on the hilarious iPhone vs. Evo video. "NSFW because of language (though that's probably relative to your work environment)."]
  - Another perspective on SCRUM, YouTube

8.2.24 Continuous Integration and DevOps

- Continuous Integration
- **Jenkins CI**
  - Pipeline as Code with Jenkins, Jenkins web sit
  - GitHub Organization Folder Plugin, Jenkins Plugin web site
  - Pipeline Stage View Plugin, Jenkins Plugin web site
  - JENKINS_HOME directory
  - Distributed builds
  - Automatically Generating Jenkins Jobs, by Jim Graf, March 11, 2016
  - Simple Continuous Integration / Deployment With Jenkins, by Randall Degges
  - 7 Habits of Highly Effective Jenkins Users, by Andrew Beyer, July 28, 2011
  - A Back Up And Restore Recipe for Jenkins Metadata, by chris, March 16, 2015
  - How To Deploy Jenkins Completely Pre-Configured - Automating Jenkins, by Emmet O'Grady, October 11, 2016
  - Jenkins2 Pipeline jobs using Groovy code in Jenkinsfile, by Wilson Mar, August 3, 2016
  - Creating Jenkins pipelines with Ansible, part 1, by Joel Wilsson, Augusst 6, 2016
  - Creating Jenkins pipelines with Ansible, part 2, by Joel Wilsson, August 10, 2016
- GoCD ("Simplify Continuous Delivery")
- What is DevOps?
- Implementing Continuous Delivery: Jenkins or Bamboo?, by Francis Adanza, April 12, 2016

8.2.25 Kanban method

- Kanban (Development), Wikipedia
- Kanban Applied to Software Development: from Agile to Lean, by Kenji Hiranabe, January 14, 2008
- Kanban vs. Scrum: Kanban is NOT for Software Development, but Scrum is!, by charlesbradley, February 4, 2013
- How To Get Started With Kanban In Software Development, by Derick Bailey, August 5, 2009
- Adventures in Lean, by Derick Bailey, November 19, 2008

8.2.26 Software Engineering Project Management

- 44 engineering management lessons, by Slava Akhmechet, October 3, 2014

8.2.27 Messaging using AMQP

- AMQP and Beyond: Messaging by Extending RabbitMQ, by Tony Garnock-Jones
- kombu - AMQP Messaging Framework for Python, Version: 1.0.5
- Postage - a RabbitMQ-based Component Python Library
- Federated Queues, RabbitMQ web site
- Distributed RabbitMQ brokers, RabbitMQ web site
- Alvaro Videla - Building a Distributed Data Ingestion System with RabbitMQ, YouTube, Jul 16, 2014
- Distributed log aggregation with RabbitMQ Federation, by Alvaro Videla, December 17, 2013
- Routing Topologies for Performance and Scalability with RabbitMQ, by Helena Edelson, April 1, 2011
- Configuring SSL for RabbitMQ, by Richard, January 27, 2013
- Securing the RabbitMQ Management Console with SSL, by Richard, January 27, 2013

8.2.28 Software licensing and Open Source

- tl;dr Legal quick references
  - Apache 2.0 License
  - BSD 3-Clause License (Revised)
  - Apache 2.0 and BSD 3-Clause Licenses (Merged)
- How to Make Money from Open Source, by pieterh, September 18, 2012
- How To Capture an Open Source Project, by pieterh, October 21, 2013
- OSI Hopes To Decrease Number of Licenses, Slashdot (Bruce Parens, creator of the Open Source definition, hopes to get back to just three Open Source licenses from which to chose, rather than over 30)
- Understanding Open Source Software, by Red Hat’s Mark Webbink, Esq., GROKLAW
- BSD and GPL Licensing, wikipedia
- The Open Source Initiative has a list of open source licenses
- Open Source Licensing: Links and Resources, wasabisystems.com
- GPL v3 drafting process (Current Release: January 16, 2006: Next Release: Anticipated May or June 2006)
- MySQL’s “dual-license” model
- Trademark and OSS, Software Pluralism, University of Washington Law School
- Derivative Works, Software Pluralism, University of Washington Law School
- Patent Risks, Software Pluralism, University of Washington Law School
- Dual-licensing model
  - Dual Licensing in Open Source Software Industry, by Mikko Välimä, January, 2003
  - Dual Licensing: Having Your Cake and Eating It Too, by Philip H. Albert, LinuxInsider, November 16, 2004
  - Dual Licensing, OSS Watch
  - MySL AB FLOSS Exception

- Open Source licenses in the courts
  - Litigation in Open Source, by Jason B. Wacha, Vice President of Corporate Affairs and General Counsel, MontaVista Software, Inc., August 8, 2004
  - Linksys routers caught up in open source dispute
  - Is Linksys shirking the GPL? (Maybe not.), O'Reilly OnLAMP.com
  - Linux's Hit Men, by Daniel Lyons, Forbes.com, October 14, 2003 [Discusses the problems of Cisco Systems' purchase of Linksys (manufacturer of home routers), which had GPL'd code on the Broadcom chips it used.]
  - GPL: IT'S THE LAW, by Nancy Cohen

8.2.29 Miscellaneous

- MIL-STD-498
  - A forgotten military standard that saves weeks of work (by providing free project management templates), by Kristof Kovacs
  - MIL-STD-498 Software Development and Documentation (all pdf files)
  - http://search.cpan.org/~softdia/Docs-US_DOD-STD2167A/
- How HipChat Stores And Indexes Billions Of Messages Using ElasticSearch And Redis
- Kuyruk 0.20.3
- Celery: Distributed Task Queue
- Kronuz / celery
- Tuple Space
- Useful one-line scripts for sed,

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

Bibliography

[Vix16] Paul Vixie. Magical Thinking in Internet Security. https://www.farsightsecurity.com/Blog/20160428-vixie-magicalthinking/, April 2016.