The Merits of Using Ansible and Nagios Together

Michael Pobega (pobegam@sunyit.edu)

May 30, 2017

Abstract

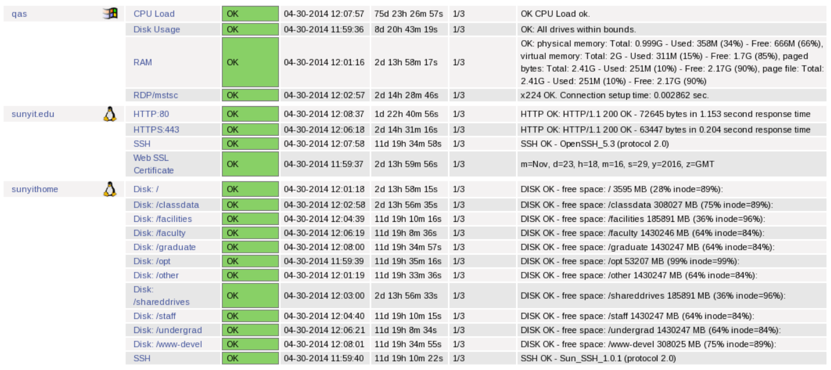
Overview of the use of Nagios and Ansible on the same machine and/or network, and how they can complement each other.

Contents

1 Nagios Retrospective

1.1 Deployment

This was the most daunting part of my entire Capstone. Deploying Nagios was a complete pain; a majority of the work was information gathering and firewall modifications.



qas	CPU Load	OK	04-30-2014 12:07:57	75d 23h 26m 57s	1/3	OK CPU Load ok.
	Disk Usage	OK	04-30-2014 11:59:36	8d 20h 43m 19s	1/3	OK: All drives within bounds.
	RAM	OK	04-30-2014 12:01:16	2d 13h 58m 17s	1/3	OK: physical memory: Total: 0.999G - Used: 358M (34%) - Free: 666M (66%), virtual memory: Total: 2G - Used: 311M (15%) - Free: 1.7G (85%), paged bytes: Total: 2.41G - Used: 251M (10%) - Free: 2.17G (90%), page file: Total: 2.41G - Used: 251M (10%) - Free: 2.17G (90%)
	RDP/mstsc	OK	04-30-2014 12:02:57	2d 14h 28m 46s	1/3	x224 OK. Connection setup time: 0.002862 sec.
sunyit.edu	HTTP:80	OK	04-30-2014 12:08:37	1d 22h 40m 56s	1/3	HTTP OK: HTTP/1.1 200 OK - 72645 bytes in 1.153 second response time
	HTTPS:443	OK	04-30-2014 12:06:18	2d 14h 31m 16s	1/3	HTTP OK: HTTP/1.1 200 OK - 63447 bytes in 0.204 second response time
	SSH	OK	04-30-2014 12:07:58	11d 19h 34m 58s	1/3	SSH OK - OpenSSH_5.3 (protocol 2.0)
	Web SSL Certificate	OK	04-30-2014 11:59:37	2d 13h 59m 56s	1/3	m=Nov, d=23, h=18, m=16, s=29, y=2016, z=GMT
sunyithome	Disk: /	OK	04-30-2014 12:01:18	2d 13h 58m 15s	1/3	DISK OK - free space: / 3595 MB (28% inode=89%):
	Disk: /classdata	OK	04-30-2014 12:02:58	2d 13h 56m 35s	1/3	DISK OK - free space: /classdata 308027 MB (75% inode=89%):
	Disk: /facilities	OK	04-30-2014 12:04:39	11d 19h 10m 16s	1/3	DISK OK - free space: /facilities 185891 MB (36% inode=96%):
	Disk: /faculty	OK	04-30-2014 12:06:19	11d 19h 8m 36s	1/3	DISK OK - free space: /faculty 1430246 MB (64% inode=84%):
	Disk: /graduate	OK	04-30-2014 12:08:00	11d 19h 34m 57s	1/3	DISK OK - free space: /graduate 1430247 MB (64% inode=84%):
	Disk: /opt	OK	04-30-2014 11:59:39	11d 19h 35m 16s	1/3	DISK OK - free space: /opt 53207 MB (99% inode=99%):
	Disk: /other	OK	04-30-2014 12:01:19	11d 19h 33m 36s	1/3	DISK OK - free space: /other 1430247 MB (64% inode=84%):
	Disk: /shareddrives	OK	04-30-2014 12:03:00	2d 13h 56m 33s	1/3	DISK OK - free space: /shareddrives 185891 MB (36% inode=96%):
	Disk: /staff	OK	04-30-2014 12:04:40	11d 19h 10m 15s	1/3	DISK OK - free space: /staff 1430247 MB (64% inode=84%):
	Disk: /undergrad	OK	04-30-2014 12:06:21	11d 19h 8m 34s	1/3	DISK OK - free space: /undergrad 1430247 MB (64% inode=84%):
	Disk: /www-devel	OK	04-30-2014 12:08:01	11d 19h 34m 55s	1/3	DISK OK - free space: /www-devel 308025 MB (75% inode=89%):
	SSH	OK	04-30-2014 11:59:40	11d 19h 10m 22s	1/3	SSH OK - Sun_SSH_1.0.1 (protocol 2.0)

Figure 1: Nagios listing of hosts and services.

The first servers added were added from a list obtained from an old Nagios server, but the rest had to be gathered by hand. After weeks in VCenter gathering system information including hostnames, IPs, and services, the Nagios server was finally ready to be deployed.

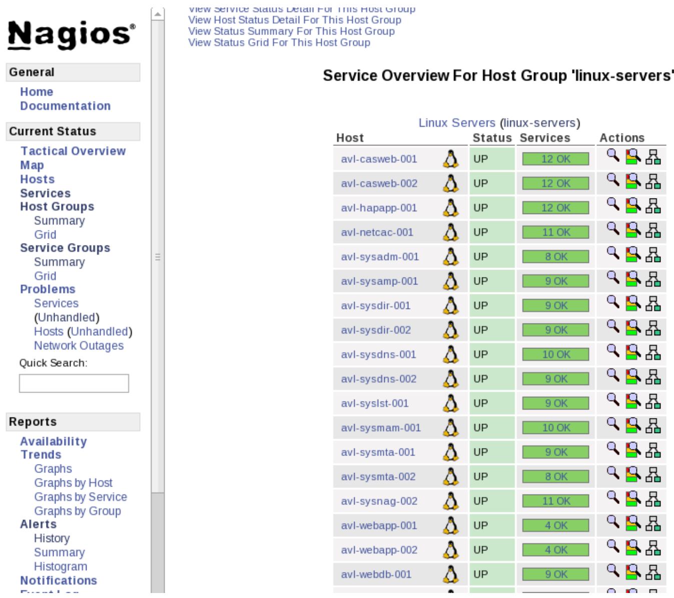
Hostgroups were created to easily manage batch commands for multiple hosts. These hostgroups include Linux, Windows, NRPE, NSClient++, Webservers and MySQL Cluster (which as of the time of writing was still being used, but is now deprecated).

1.2 Security

The server sits behind a firewall and is only accessible from the administrative subnet. There isn't much else to say about this, limiting access seems to be the best practice for securing a Nagios server.

1.3 Conclusion

This is a simple server and I don't have too much to say about it that hasn't already been said better by someone else. The majority of the work for this server was busy work, and besides keeping watch on Apache and Linux exploits the server is secured via firewall making it inaccessible by the outside world.



Service Overview For Host Group 'linux-servers'

Linux Servers (linux-servers)

Host	Status	Services	Actions
avl-casweb-001	UP	12 OK	
avl-casweb-002	UP	12 OK	
avl-hapapp-001	UP	12 OK	
avl-netcac-001	UP	11 OK	
avl-sysadm-001	UP	8 OK	
avl-sysamp-001	UP	9 OK	
avl-sysdir-001	UP	9 OK	
avl-sysdir-002	UP	9 OK	
avl-sysdns-001	UP	10 OK	
avl-sysdns-002	UP	9 OK	
avl-syslst-001	UP	9 OK	
avl-sysmam-001	UP	10 OK	
avl-sysmta-001	UP	9 OK	
avl-sysmta-002	UP	8 OK	
avl-sysnag-002	UP	11 OK	
avl-webapp-001	UP	4 OK	
avl-webapp-002	UP	4 OK	
avl-webdb-001	UP	9 OK	

Figure 2: Linux hostgroup view inside of Nagios.

2 Ansible Retrospective

2.1 Deployment

Deployment of Ansible on a network is near trivial. Because Ansible uses SSH as it's connection method it doesn't require any additional software installed on any of the clients, meaning deployment is as simple as *'yum install ansible'*. This is probably Ansible's biggest strength over other equally (or even moreso) robust solutions such as Puppet, Chef and Salt.

The client nodes need to have the ansible account present and accessible via SSH. The solution I've deployed uses SSH keys to connect and the local ansible accounts have randomized passwords (just to lower the possible attack vectors by one). Currently the template used to create new VMs has the Ansible account already present, but for any legacy machines that don't have the account already it can be created easily; just add an ansible user/group and copy the public SSH key.

2.2 Security

Initially in my first draft of this paper this paragraph talked about how it was great that Ansible uses standardized packages such as OpenSSH and OpenSSL to handle it's connection and encryption, but it's since been rewritten in response to the Heartbleed Bug.

Using standard libraries and packages such as OpenSSH and OpenSSL offer huge security benefits; they get patched constantly by upstream/disto maintainers, they are guaranteed to be thoroughly tested, and come standard with most *nix systems. The other side of the coin, however, is that they are also bigger targets. Bugs and exploits may very well exist and not be publicized, such as the Heartbleed vulnerability that existed in the OpenSSL code for two years (an eternity in terms of security).

Another big portion of security has to do with how accessible the administrator wants the servers to be. On the one hand, the ansible account can be setup to have full sudo access with NOPASSWD on every system; this allows password-less use of the *–sudo* flag when calling Ansible. On the other hand if the *ansible* user account is ever cracked or broken in to the user will have full root access to the system, which could quickly turn into a network-wide problem if the hacker is crafty enough.

One way around this is to only enable the ansible account to have NOPASSWD access to specific binaries on the system. The issue with this approach is that each server's *sudoers* file would have to be modified by hand to add or remove permissions, which could end up being a huge undertaking.

```
ansible ALL=NOPASSWD: /sbin/shutdown, /sbin/reboot, /usr/bin/yum
```

/etc/sudoers: limited permissions

Another workaround is to specify the root passwords at runtime when calling Ansible, but this is time consuming if all of the servers have different passwords.

All in all, Ansible is a very secure choice for automation. Gaining access to the Ansible server doesn't automatically give access to the nodes due to it's use of SSH host keys, unlike other automation suites such as Salt, where breaking into the master would encompass a full network compromise. On the topic of Salt it uses it's own AES implementation which has been proven in the past to not be the most secure implementation; another +1 for using standard SSH as a transport.

2.3 Everyday Use (and misuse)

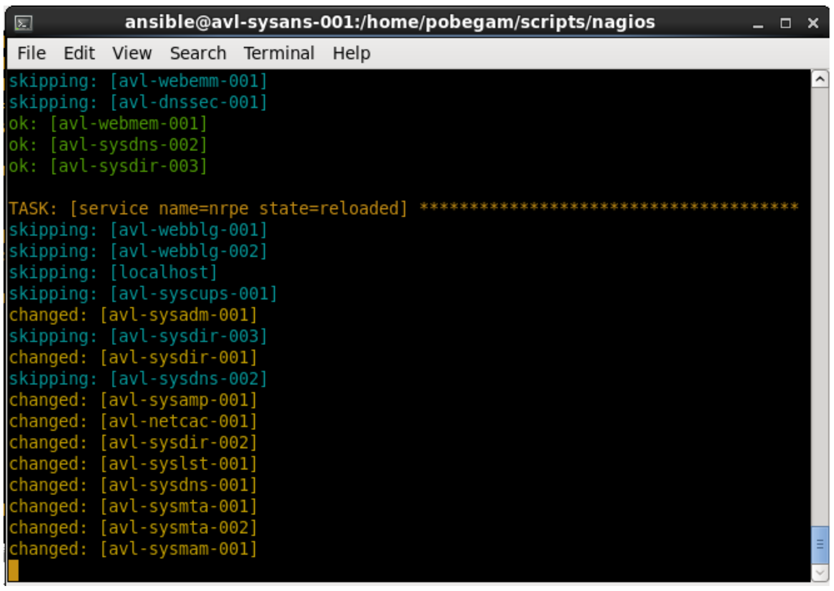
In my time playing with Ansible I've used it to automate a lot of things; I've also made some rookie mistakes in the process of learning Ansible.

Originally I didn't realize how powerful Ansible modules are or that Ansible could iterate over commands, and had HUGE playbooks of one line commands (which I won't include for the reader's sanity). *ansible-doc* has become my go-to command for the past few months even moreso than *man*; you can find documentation for all of the Ansible modules listed there, which makes writing one-liners or code during an internet outage a breeze.

By the way, Ansible iteration looks like this:

```
   - name: yum update stuff (this step takes a while)
     yum: name={{ item }} state=latest
     with_items:
      - openssl
      - openssl-devel
      - rng-tools
      - keepalived
```

Iteration/multiple arguments in an Ansible Playbook


ansible@avl-sysans-001:/home/pobegam/scripts/nagios
File Edit View Search Terminal Help

```
skipping: [avl-webemm-001]
skipping: [avl-dnssec-001]
ok: [avl-webmem-001]
ok: [avl-sysdns-002]
ok: [avl-sysdir-003]

TASK: [service name=nrpe state=reloaded] ***************************************
skipping: [avl-webblg-001]
skipping: [avl-webblg-002]
skipping: [localhost]
skipping: [avl-syscups-001]
changed: [avl-sysadm-001]
skipping: [avl-sysdir-003]
changed: [avl-sysdir-001]
skipping: [avl-sysdns-002]
changed: [avl-sysamp-001]
changed: [avl-netcac-001]
changed: [avl-sysdir-002]
changed: [avl-syslst-001]
changed: [avl-sysdns-001]
changed: [avl-sysmta-001]
changed: [avl-sysmta-002]
changed: [avl-sysmam-001]
```

Figure 3: Ansible doing it's thing.

I've used Ansible for multiple things (which I will list but are documented elsewhere):

- DNSSEC Deployment for sunyct.edu
- Deploying DNS failover
- Distributing and enabling scripts and cronjobs to enable functionality between servers and ServiceDesk
- Enabling remote syslogging
- Deploying memcache for a Wordpress server
- Installing NRPE (both from yum and from source at separate times) on multiple servers
- Distributing and updating custom commands and scripts for servers monitored by NRPE

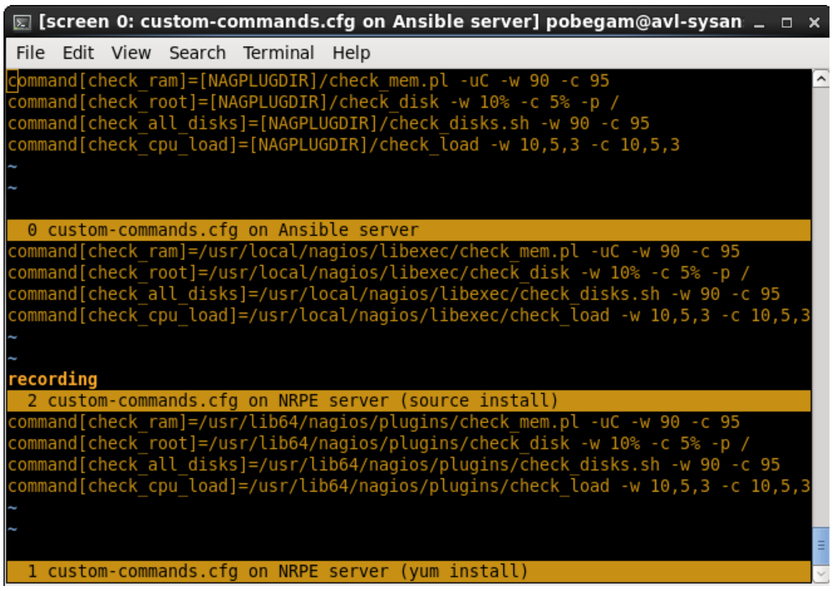
Ansible has also been invoked on the command line manually to do multiple things, such as scan for and patch known Linux exploits/rootkits, including Heartbleed and the recent __UMBREON__ rootkit.

2.4 Conclusion

Ansible is a FANTASTIC tool that I will be using in the future. It has taught me a lot about automation and the mindset involved, and it's given me a great toolset to bring with me into the world of Sysadmin'ing.

3 Ansible and Nagios

Ansible and Nagios are two tools with a very Unixy mindset; they each do one thing and they do it well, while relying on other tools to fill in the blanks. Nagios monitors and alerts and does those tasks amazingly well, but automating anything requires a separate tool; in this case, Ansible.


[screen 0: custom-commands.cfg on Ansible server] pobegam@avl-sysan
File Edit View Search Terminal Help

```
command[check_ram]=[NAGPLUGDIR]/check_mem.pl -uC -w 90 -c 95
command[check_root]=[NAGPLUGDIR]/check_disk -w 10% -c 5% -p /
command[check_all_disks]=[NAGPLUGDIR]/check_disks.sh -w 90 -c 95
command[check_cpu_load]=[NAGPLUGDIR]/check_load -w 10,5,3 -c 10,5,3
~
~
  0 custom-commands.cfg on Ansible server
command[check_ram]=/usr/local/nagios/libexec/check_mem.pl -uC -w 90 -c 95
command[check_root]=/usr/local/nagios/libexec/check_disk -w 10% -c 5% -p /
command[check_all_disks]=/usr/local/nagios/libexec/check_disks.sh -w 90 -c 95
command[check_cpu_load]=/usr/local/nagios/libexec/check_load -w 10,5,3 -c 10,5,3
~
~
recording
  2 custom-commands.cfg on NRPE server (source install)
command[check_ram]=/usr/lib64/nagios/plugins/check_mem.pl -uC -w 90 -c 95
command[check_root]=/usr/lib64/nagios/plugins/check_disk -w 10% -c 5% -p /
command[check_all_disks]=/usr/lib64/nagios/plugins/check_disks.sh -w 90 -c 95
command[check_cpu_load]=/usr/lib64/nagios/plugins/check_load -w 10,5,3 -c 10,5,3
~
~
  1 custom-commands.cfg on NRPE server (yum install)
```

Figure 4: custom-commands.png on three different servers

Ansible was used to deploy and administer the servers that Nagios monitored. NRPE installs were automated and can now be rapidly redeployed, while NSClient++ (Windows' NRPE) must be installed manually on each server, which has proven to be a time consuming task.

Another issue we ran into was using arguments with NRPE. Our legacy servers have NRPE installed with arguments disabled at compile time, so using NRPE arguments is out of the question. This means that commands must be statically defined in a local file on each server. Ansible makes updating these files (and uploading the shell scripts that are run) very simple – it's just a matter of identifying the folders the files need to be put into and copying them.

Ansible solved all of these issues in a very elegant fashion. It checks what kind of server it is dealing with:

```
  - name: check if /etc/nrpe.d/ exists
    stat: path=/etc/nrpe.d/
    register: nrpe_exists
  - name: check if /etc/init.d/nrpe exists
    stat: path=/etc/init.d/nrpe
    register: nrpe_initd
  - name: check if /etc/xinetd.d/nrpe exists
    stat: path=/etc/xinetd.d/nrpe
    register: nrpe_xinetd
```

Discovering information about NRPE servers

Then it copies the file to the proper directory based on the previous tests:

```
- name: Uploading $REMOTE$:/etc/nrpe.d/custom-commands.cfg for yum
    install
  copy: src=/srv/nfs/nrpe/custom-commands.cfg
      dest=/etc/nrpe.d/custom-commands.cfg owner=nrpe group=nrpe
      mode=750 force=yes
  when: nrpe_exists.stat.exists and nrpe_initd.stat.exists
- name: Uploading $REMOTE$:/etc/nrpe.d/custom-commands.cfg for src
    install
  copy: src=/srv/nfs/nrpe/custom-commands.cfg
      dest=/etc/nrpe.d/custom-commands.cfg owner=nagios group=nagios
      mode=750 force=yes
  when: nrpe_exists.stat.exists and nrpe_xinetd.stat.exists
```

Copying files to NRPE servers

And uses sed to replace the text *[NAGPLUGDIR]* with the proper directory for that system.

```
- name: Updating $REMOTE$:/etc/nrpe.d/custom-commands.cfg for yum
    install
  command: sed -i "s#\[NAGPLUGDIR\]#/usr/lib64/nagios/plugins#g"
      /etc/nrpe.d/custom-commands.cfg
  when: nrpe_exists.stat.exists and nrpe_initd.stat.exists
- name: Updating $REMOTE$:/etc/nrpe.d/custom-commands.cfg for src
    install
  command: sed -i "s#\[NAGPLUGDIR\]#/usr/local/nagios/libexec#g"
      /etc/nrpe.d/custom-commands.cfg
  when: nrpe_exists.stat.exists and nrpe_xinetd.stat.exists
```

Modifying custom-commands.cfg

Then it copies over all of the custom scripts from the Ansible server and places them into the Nagios plugin directory, and finally restarts the daemon (nrpe or xinetd based on the install method):

```
- name: Copy over custom NRPE executables for yum install
  copy: src=/srv/nfs/nrpe/libexec/ dest=/usr/lib64/nagios/plugins/
      owner=nrpe group=nrpe mode=750
  when: nrpe_exists.stat.exists and nrpe_initd.stat.exists
- name: Copy over custom NRPE executables for src install
  copy: src=/srv/nfs/nrpe/libexec/ dest=/usr/local/nagios/libexec/
      owner=nagios group=nagios mode=750
  when: nrpe_exists.stat.exists and nrpe_xinetd.stat.exists

- service: name=nrpe state=reloaded
  when: nrpe_initd.stat.exists
- service: name=xinetd state=reloaded
  when: nrpe_xinetd.stat.exists
```

This playbook has completely trivialized NRPE deployment. Initially, when I wanted to create a new command or modify a previously existing one I had to do it server by server. After Ansible, though, the process is as simple as creating the script on the Ansible server, running the Playbook, and modifying the relevant hosts/hostgroups in Nagios.

Keep in mind this playbook only represents one use of Ansible and Nagios in conjunction; Anything Nagios needs that requires changes on the client side would be wholly trivialized with Ansible.

Between this, the simplified NRPE deployment and all of the one liners used to manage groups of servers simultaneously, Ansible has proven itself to be a completely irreplacable tool when used in conjuction with Nagios. It fills a lot of the voids that Nagios doesn't (by definition Nagios is only a monitoring tool), and allows an administrator to make sweeping changes to monitored hosts with one simple command.