Welcome to

Using Ansible to run a small hosting infrastructure

Henrik Lund Kramshøj hlk@zencurity.dk

Slides are available as PDF, kramshoej@Github

Try searching for automate-it-2017.tex in the repo

slide are available as PDF kramshoej@Github

Goal and Agenda: Ansible and more

Our company is rolling out a new health infrastructure of connected clinics in Norway.

We are very few people running the systems, so we need to automate

... but automation bring other benefits.

Prerequisites: Python, SSH, SSH keys, sudo

Ansible introduction, what is this Ansible

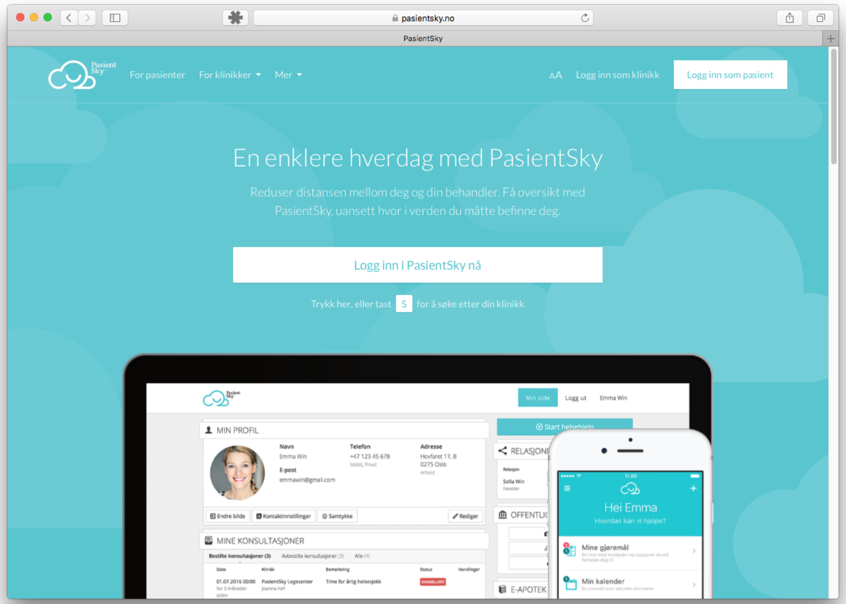
Ansible targets: Linux hosts, ESXi, network devices

Ansible examples, and workshop

Keywords: workshop, Ansible, YAML, automating boring stuff

For optimal fun, use your laptop, fetch it in next break!

Pasientsky.no - the environment and services


pasientsky.no

PasientSky

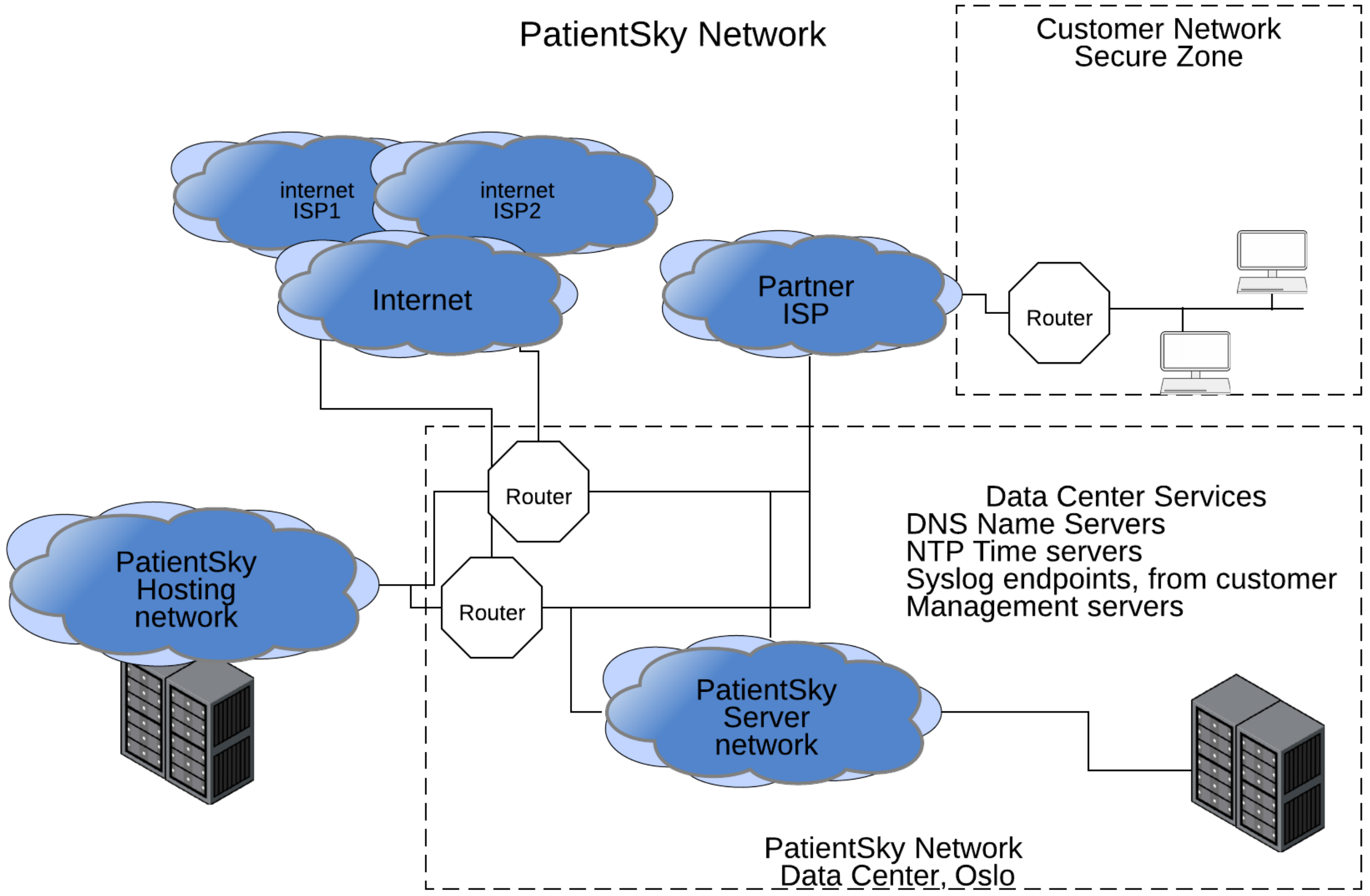
En enklere hverdag med PasientSky

Reduser distansen mellom deg og din behandler. Få oversikt med PasientSky, uansett hvor i verden du måtte befinne deg.

Trykk her, eller tast S for å søke etter din klinikk

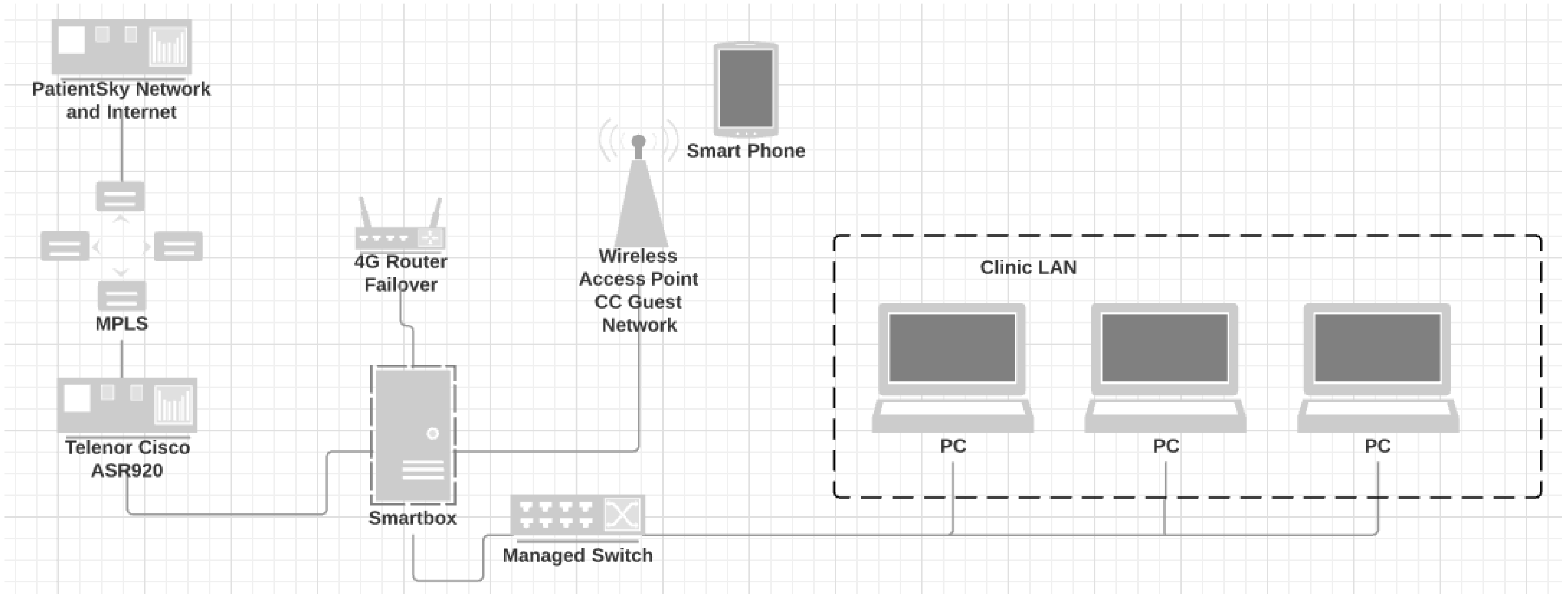
Connected Clinic from PasientSky provides modern and revolutionary solutions meeting the special communication needs in the health sector. A small and smart box provides quick and stable internet connection with integrated telephony and time book.

Overview



Most servers are Linux, percentage is OpenBSD, running on VMware ESXi

OpenBSD CPE: BGP, PF and service daemons



- Soekris Net6501-50 1 Ghz CPU, 1024 Mbyte DDR2-SDRAM, 4 x 1Gbit Ethernet
- OpenBSD operating system
- We install new Smartboxes every week

Important processes and components

- Setup hardware
- Connect cables

- Setup development environment
- Setup staging environment - like development
- Setup production environment - like staging
- Setup firewalls, security, LDAP servers
- Setup other surrounding infrastructure

Top parts hard to automate, bottom easier ☺

What is Ansible

AUTOMATION FOR EVERYONE

Ansible is designed around the way people work and the way people work together.

Ansible has thousands of users, hundreds of customers and over 2,400 community contributors.

750+ Ansible modules

`https://www.ansible.com/`

We have been using Ansible for about 2 years

and we dont use it *correctly*

Alternatives: (cfengine), Chef, Puppet or Salt

Ansible at Patientsky

- We use Ansible - mostly 1.9 on Mac
- We use Git repo on bitbucket - keeping it private
- Ansible used for OpenBSD version 6.0, but since hmm 5.5?
- Ansible used for Ubuntu Linux, multiple versions
- Really a standard setup
- Also a few select FreeBSD, some CentOS for testing OpenStack etc.
- We could actually use Ansible for Junos devices, hmm perhaps soon ☺

Who uses Ansible

- PatientSky Danmark Aps
- Bornhack - Thomas Tykling Rasmussen

How Ansible Works: inventory files

List your hosts in one or multiple text files:

```
[all:vars]
ansible_ssh_port=34443

[office]
fw-01 ansible_ssh_host=192.168.1.1 ansible_ssh_port=22
ansible_python_interpreter=/usr/local/bin/python

[infrastructure]
smtp-01     ansible_ssh_host=185.60.160.37 ansible_python_interpreter=/usr/local/bin/python
vpnmon-01   ansible_ssh_host=10.50.22.18
```

- Inventory files specify the hosts we work with
- Linux and OpenBSD servers shown here
- Real inventory for the site with development and staging approx 500 lines
- office and infradstructure are group names

How Ansible Works: ad hoc parallel execution

Using the inventory file you can run commands with Ansible:

```
ansible -m ping new-server
ansible -a "date" new-server
ansible -m shell -a "grep a /etc/something" new-server
```

- Running commands on multiple servers is easy now
- This alone has value, you can start
- Checking settings on servers
- Making changes to servers

How Ansible Works: Playbooks

The benefit comes with tasks - do something:

```
- hosts: smartbox-*
  become: yes
  tasks:
  - name: Create a template pf.conf
    template:
      src=pf/pf.conf.j2
      dest=/etc/pf.conf owner=root group=wheel mode=0600
   notify:
      - reload pf
    tags:
      - firewall
      - pf.conf
```

- Specify the end result, more than the steps, also restarts daemons
- Use the modules from `https://docs.ansible.com/ansible/modules_by_category.html`
- Jinja templates - ooooooh so great!

How Ansible Works: typical execution

```
ansible-playbook -i hosts.odn1 -K infrastructure-firewalls.yml -t pf.conf --check --diff

ansible-playbook -i hosts.odn1 -K infrastructure-firewalls.yml -t pf.conf

ansible-playbook -i hosts.odn1 -K infrastructure-nagios.yml -t config-only

ansible-playbook -i smartboxes -K create-pf-conf.yml -l smartbox-xxx-01
```

- Pro tip: check before you push out changes to production networks ☺
- Diff will show the changes about to be made

How Ansible Works: atypical execution / gotchas

```
ansible -i ../smartboxes.osl1 --become --ask-become-pass -m shell
-a "pfctl -s rules" -l smartbox01

ansible -i ../smartboxes.osl1 --become --ask-become-pass -m shell
-a "nmap -sP 185.161.1xx.123-124 2> /dev/null| grep done" all
```

- Sometimes you need a trick or persistence
- Ansible moving from *sudo* to *become*
- The normal -K did not work, but the above does for ad hoc commands

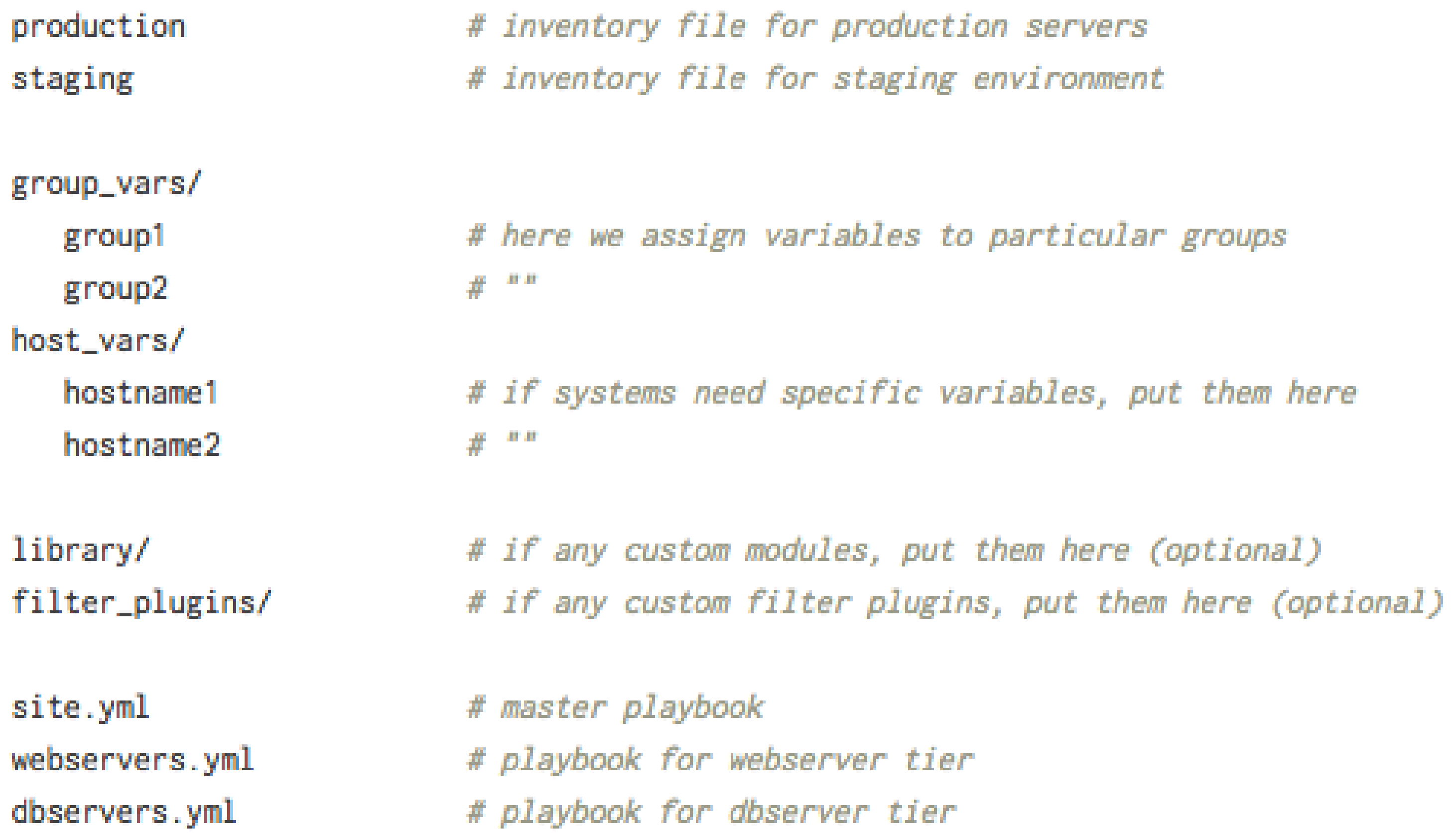
Stop: discussion benefits of Ansible

Do we even need to run the same command on multiple servers?

What are the benefits of Ansible?

- Central configuration management - git repo
- Same playbook - different inventory file, what happens
- Already using Ansible, tell us why and how

Structure of Ansible repos



```
production                # inventory file for production servers
staging                   # inventory file for staging environment

group_vars/
   group1                 # here we assign variables to particular groups
   group2                 # ""
host_vars/
   hostname1              # if systems need specific variables, put them here
   hostname2              # ""

library/                  # if any custom modules, put them here (optional)
filter_plugins/           # if any custom filter plugins, put them here (optional)

site.yml                  # master playbook
webservers.yml            # playbook for webserver tier
dbservers.yml             # playbook for dbserver tier
```

Recommended dir layout - partial, from:
`https://docs.ansible.com/ansible/playbooks_best_practices.html`

Our setup

- Central configuration management - git repo
- Same playbook - different inventory file, staging and production run from SAME tasks
- Have playbooks per server type: front end firewalls, back end firewalls, etc.
- Some 130 playbooks in main production server config repo
- Another 9 files with total 1200 lines (sort -u = 530 unique lines) for a small ISP-like setup, firewalls, network security monitoring, network management systems, name servers and a few services

```
infrastructure-firewall-mda.yml infrastructure-nameserver.yml
infrastructure-firewall1.yml infrastructure-netflow-syslog.yml
infrastructure-firmwareserver.yml infrastructure-nms.yml*
infrastructure-ipam.yml* infrastructure-nsm-master.yml*
infrastructure-nagios.yml*
```

Life of a server

- Create VM
- Network install - with pxeboot
- Standard settings: hostname, LDAP, SSH, timezone, ...
- Configure this server: application installation, settings, etc.
- Configure monitoring: like Smokeping

Up and running with Ansible

Prequisites for Ansible - you need a Linux machine:

- python language - Ansible uses this
- ssh keys - remote login without passwords
- Sudo - allow regular users to do superuser tasks
- Recommended tool: `ssh-copy-id` for getting your key on new server
- Recommended Change: `sshd_config` - no passwords allowed, no bruteforce
- Recommended to use: jump hosts/ProxyCommand in `ssh_config`
- Highly recommended: Git and/or github for version control

Official docs:
`https://docs.ansible.com/ansible/intro_installation.html`

Installation options

Options:

1. use your laptop, easy if you run Mac or Linux
2. use my virtual jump host you can run commands - but editing files troublesome
3. install a local virtual machine, like Kali Linux and use graphical editor

BTW Windows users need to checkout:
`https://docs.ansible.com/ansible/intro_windows.html`
”Starting in version 1.7, Ansible also contains support for managing Windows machines. This uses native PowerShell remoting, rather than SSH.”

Install Ansible on Ubuntu Linux clients

- Mac OS X `brew install ansible19`
- Ubuntu Linux try something like:

```
$ sudo apt-get install software-properties-common
$ sudo apt-add-repository ppa:ansible/ansible
$ sudo apt-get update
$ sudo apt-get install ansible openssh-client
```

- Other platforms - sorry google it

Yes, I expect OpenSSH client is also installed :-D

Install python on servers

- Ubuntu server: `apt install python openssh-server`
- OpenBSD: `pkg_add python`
  Requires `PKG_PATH` set, see below

OpenBSD package path can be set in `/root/.profile`

```
PKG_PATH=ftp://mirror.one.com/pub/OpenBSD/`uname -r`/packages/`uname -m`
PKG_PATH=https://stable.mtier.org/updates/$(uname -r)/$(arch -s):$PKG_PATH
export PKG_PATH
```

Yes, I expect OpenSSH server is also installed :-D

Create OpenSSH compatible private / public key pair

```
hlk@generic:~$ ssh-keygen -f .ssh/kramse
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in .ssh/kramse.
Your public key has been saved in .ssh/kramse.pub.
The key fingerprint is:
SHA256:chCjaP6BHaoPy/EMDlP6xKAP4aGAX2mknGA/ZoAzU3o hlk@generic
The key's randomart image is:
+---[RSA 2048]----+
|   .   o         |
|.o . . o         |
|BoE + .          |
|oXoB o .         |
|=.O=* . S        |
|=O++.. o         |
|X++ .            |
|+X=              |
|.o+o             |
+----[SHA256]-----+
```

`/home/hlk/.ssh/kramse.pub` is the public key in this example

SSH utility ssh-copy-id

You need to copy your SSH public key to the server to use SSH+Ansible:

```
hlk@kunoichi:hlk$ ssh-copy-id -i .ssh/kramse hlk@10.0.42.147
/usr/local/bin/ssh-copy-id: INFO: Source of key(s) to be installed: ".ssh/kramse.pub"
The authenticity of host '10.0.42.147 (10.0.42.147)' can't be established.
ECDSA key fingerprint is SHA256:DP6jqadDWEJW/3FYPY84cpTKmEW7XoQ4zDNf/RdTu6M.
Are you sure you want to continue connecting (yes/no)? yes
/usr/local/bin/ssh-copy-id: INFO: attempting to log in with the new key(s),
to filter out any that are already installed
/usr/local/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you
are prompted now it is to install the new keys
hlk@10.0.42.147's password:

Number of key(s) added:        1

Now try logging into the machine, with:   "ssh -o 'IdentitiesOnly yes' 'hlk@10.0.42.147'"
and check to make sure that only the key(s) you wanted were added.
```

This is the best tool for the job!

Exercise: trying Ansible

Create inventory file, and then:

```
ansible -m ping new-server
ansible -a "date" new-server
ansible -m shell -a "grep a /etc/something" new-server
```

- Lets try running Ansible!
- Hopefully there is a small getting started repo to clone from Github ☺
- Install software: ansible and openssh-client
- Generate SSH key pair
- Server to use should be shown on the whiteboard (or similar)
- You can override user with `ansible -u manager`
  very useful if you are bringing up a server from PXE boot using user `manager`
- Trouble? Try running with `-vvv`, try manual ssh, is Python ready?

Success looks something like this

```
$ ssh-keygen -f .ssh/kramse
... generates key pair and saves public key in .ssh/kramse.pub
$ ssh-copy-id -i .ssh/kramse manager@10.0.42.147
... asks for password "henrik42" and installs key

$ ansible -i hosts.sitename -u manager -m ping server01
server01 | success >> {
    "changed": false,
    "ping": "pong"
}
```

Exercise: try fetching facts

```
ansible -i hosts.odn1 -u manager -m setup server01 | grep hostname
        "ansible_hostname": "odn1-fw-odn1-01",
```

- Facts are fetched by default from servers
- Can be fetched / investigated using the setup module
- Returns JSON
- Try saving the output in a file and look at it:
  ```
  ansible -i hosts.odn1 -u manager -m setup server01 > facts.txt
  less facts.txt
  ```

Exercise: try adding you own user

```
---
- hosts: all:!*openbsd*
  become: true
  serial: 10
tasks:
  - group: name=hlk state=present
  - user: name=hlk shell=/bin/bash group=sudo
```

- Copy above or edit the create-user.yml
- Replace ”hlk” with your username, the one you want
- Run this task / playbook so your own user is created
  `ansible-playbook -i hosts.sitename -K -u manager create-user.yml`
- Dont forget to install your key on this user!
- Try running multiple times, and try adding check and diff: `ansible-playbook -i hosts.sitename`
- Congratulations: you are now running Ansible!

Important notes about tasks

> Ansible Tasks are **idempotent**. Without a lot of extra coding, bash scripts are usually not safety run again and again. Ansible uses ”Facts”, which is system and environment information it gathers (”context”) before running Tasks.
>
> Ansible uses these facts to check state and see if it needs to change anything in order to get **the desired outcome**. This makes it safe to run Ansible Tasks against a server over and over again.

Also not describing what to do, but what you want the result to be!

Quote from:
`https://serversforhackers.com/an-ansible-tutorial`

Group vars basics

```
---
# file: group_vars/all

location_name : "mydatacenter"
country_code : "dk"
city : "Copenhagen"
timezone : "Europe/Copenhagen"
```

- Group variables get loaded automatically
- Can be used for site specific things, Odense or Oslo for us
- Host vars work the same way, we prefer groups
- Note: secrets can use Ansible vault, `https://docs.ansible.com/ansible/playbooks_vault.html`

Group vars - grouping hosts dynamically

```
  # talk to all hosts just so we can learn about them,
  # and save dynamic group os_OpenBSD etc.
  - group_by: key=os_{{ ansible_os_family }}
    tags:
        - always
```

with `group_vars` files:

```
group_vars/os_Debian:service_sshd: ssh
group_vars/os_FreeBSD:service_sshd: sshd
group_vars/os_OpenBSD:service_sshd: sshd
```

Then the handler script can use:

```
  - name: restart sshd
    service: name={{ service_sshd }} state=restarted
```

Exercise: which operating system

```
tasks/common.yml
tasks/common-ubuntu.yml - include: common.yml
tasks/common-openbsd.yml - include: common.yml
tasks/common-freebsd.yml - include: common.yml
```

- Service ssh vs sshd was just an example, we can add more to these files
- Different operating systems have small differences
- Ansible handles this quite nicely

Templates

Go back to example with packet filter config

```
  - name: Create a template pf.conf
    template:
        src=roles/common/files/openbsd/default-pf.conf.j2
        dest=/etc/pf.conf owner=root group=wheel mode=0600
```

Pro tip: always use template for text files, sooner or later you need to insert vars

```
{% if inventory_hostname_short == "fw-01" or inventory_hostname_short == "fw-02" %}
# Allow something to something
pass quick from 10.1.2.3 to 10.1.0.0/22
{% endif %}
```

These files can use lots of crazy, and crazy useful features

Template can include files

In some devices we may want to include another file, custom firewall rules:

```
{% include "custom-pf/" + inventory_hostname ignore missing %}
```

This works very nicely, if the file is there, include it.

Exercise: check with lineinfile

Try checking SSH `sshd_config` settings using check and diff

- Edit the file tasks/common.yml
- Uncomment the task with `UseDNS`
- Run this task with `--check --diff`
- Yes, you may run this - and change the line
- Only the first one will see: Changed, others see OK

Use the documentation:
`https://docs.ansible.com/ansible/lineinfile_module.html`

Exercise: play with lineinfile

Try doing changes to your .profile using lineinfile

- Copy or edit the edit-hosts.yml, see tasks/common* for examples
- Run this task so you add a hostname to the /etc/hosts
- Advanced users: copy a file like /etc/services to `$HOME` and try modifying that
- Common problem: the line gets added on every run
- Sometimes we delete the line first, and then add - on every run
- Personally I often prefer having the complete file as a template in repo
  `https://xkcd.com/1171/`

Use the documentation:
`https://docs.ansible.com/ansible/lineinfile_module.html`

Ansible roles

sorry, we dont use Ansible in the correct way

So forget everything you learnt until now ☺

... not really, but make sure to visit:
`https://docs.ansible.com/ansible/playbooks_best_practices.html`

Exercise Investigate the existing playbooks

Ideas for this exercise:

- Make more changes to the playbooks, read them, execute
- Feel free to install: Nginx server01, Apache server02, Tomcat server03 etc.
- Research your favourite application server, does a playbook / role exist?
- Consider the steps that would be needed to deploy Ansible
  Which blockers do you see? Mostly technical problems or human resistance?

This was an introduction and hopefully enough to get you started

Special cases: Templates and the groups

We will now work through a couple of advanced template examples from our production:

- Control statements if/endif, else etc. NRPE example with default part and specials
- Firewall differences between dev and prod, things that are not ready yet
- Smokeping loops over the group vars smartboxes
- The smartboxes custom files, with neat trick if file does not exist

Adopting a server

- Copy files from server: like relevant ones from /etc/
- Create basic playbook(s) to copy back to server
- Generalize by making it templates and moving stuff to `group_vars`
- List packages, dpkg -l on Ubuntu/Debian
- Try installing packages on new server until it matches
- The whole process takes very little time, and you now have config backup!

Use the check and diff a lot ☺

Bad stuff with Ansible

- Worst, slow speed - solved by running specific tags, but annoying
- Nasty problem with Notify on Macs - did not notify and restart services!

Other problems when using Ansible

- Log rotate - easy to install and configure a lot, and forget this
- Requires monitoring, especially if you have many servers *duuuuh*
- Central logging, also recommended for other reasons
- Not running complete playbooks, gets outdated
- If you copy a playbook, which one is the most recent one to run?

Conclusion

Automation is cool - use it

Extras

I have brought a Cisco IOS device, anyone want to try it?

`https://docs.ansible.com/ansible/ios_config_module.html`

I may have brought a Juniper Junos device

`https://docs.ansible.com/ansible/junos_config_module.html`

OpenSSH client config with jump host

My recommended SSH client settings, put in `$HOME/.ssh/config`:

```
Host *
    ServerAliveInterval=30
    ServerAliveCountMax=30
    NoHostAuthenticationForLocalhost yes
    HashKnownHosts yes
    UseRoaming no

Host jump-01
  Hostname 10.1.2.3
  Port 12345678

Host fw-site-01 10.1.2.5
  User hlk
  Port 34
  Hostname 10.1.2.5
  ProxyCommand ssh -q -a -x jump-01 -W %h:%p
```

I configure fw using both hostname and IP,
then I can use name, and any program using IP get this config

OpenBSD python install

I still use Python 2.7, also I recommend running the ln commands:

```
# pkg_add python
quirks-2.241 signed on 2016-07-26T16:56:10Z
quirks-2.241: ok
Ambiguous: choose package for python
a        0: <None>
         1: python-2.7.12
         2: python-3.4.5
         3: python-3.5.2
Your choice: 1
python-2.7.12:bzip2-1.0.6p8: ok
python-2.7.12:libffi-3.2.1p2: ok
python-2.7.12:libiconv-1.14p3: ok
python-2.7.12:gettext-0.19.7: ok
python-2.7.12: ok
--- +python-2.7.12 -------------------
If you want to use this package as your default system python, as root
create symbolic links like so (overwriting any previous default):
 ln -sf /usr/local/bin/python2.7 /usr/local/bin/python
 ln -sf /usr/local/bin/python2.7-2to3 /usr/local/bin/2to3
 ln -sf /usr/local/bin/python2.7-config /usr/local/bin/python-config
 ln -sf /usr/local/bin/pydoc2.7  /usr/local/bin/pydoc
```

Copy paste the ln commands, so they make the links