Ansible User's Guide to Saltstack

What Ansible and Saltstack have in common

Both tools support similar core features and serve the same use cases:

- Are hybrid (imperative/declarative) configuration management for mutable infrastructure
- Use YAML and Jinja2 with similar syntax

How Ansible and Saltstack differ

Most differences stem from their differing architectures: while Ansible is agentless, Saltstack follows the master-minion by default (but also supports master only, minion only as well).

Initial setup

For Ansible nothing needs to be done on the bastion where playbooks will be run to configure hosts. Once playbooks have been written all that needs to be done is to run the playbooks.

For Saltstack the master needs to be installed and configured (default config may suffice). After that the minion needs to be installed. When a Salt minion starts, by default it searches for a master that resolves to the `salt` hostname on the network. If found, the minion initiates the handshake and key authentication process with the Salt master. Master can be configured to accept keys automatically or manually.

Executing commands

With Ansible a simple example command to check OS version of hosts looks like this:

Bash

```bash
ansible servers -m setup -i ansible_hosts -a
'filter=ansible_memfree_mb,ansible_memtotal_mb'
```

For Saltstack the same would be:

Bash

```bash
# '*' means that all minions will be targeted with the command
salt '*' status.meminfo
```

Defining tasks for multiple sets of hosts

Ansible has the concept of playbooks where tasks are defined. The equivalent for Saltstack are state formulas.

In Ansible, a play is a set (one or more) of tasks to execute on a hostgroup. A run is a set of plays that are run. Ansible has some best practices on how to organize Ansible plays and playbooks, however there does not exist a set structure. For comparisons sake I assume the following (very common) structure: there is one run which runs multiple plays on multiple hostgroups, each play defines a number of playbooks to run from roles.

For Saltstack there is the top file which is similar to an Ansible play as it describes a list hostgroups and a list of formulas to run on the hostgroups.

A top file example:

YAML

```yaml
base:          # Apply SLS files from the directory root for the 'base' environment
  'web*':      # All minions with a minion_id that begins with 'web'
    - apache   # Apply the state file named 'apache.sls'
```

The equivalent play for Ansible would be:

YAML

```yaml
- name: PLAYBOOK FOR GROUP 'WEB'
  hosts: web       # All hosts in 'web' hostgroup

  roles:
  - role: apache  # Apply the playbook from 'apache' role
```

Writing tasks

The tasks in the aforementioned apache role in Ansible would have the following structure:

Text Only

```
roles/
    apache/               # this hierarchy represents a "role"
        tasks/            #
            main.yml      #  <-- tasks file can include smaller files if
warranted
        templates/        #  <-- files for use with the template resource
            httpd.conf.j2   #  <------- templates end in .j2
        files/            #
            bar.txt       #  <-- files for use with the copy resource
        vars/             #
            main.yml      #  <-- variables associated with this role
        defaults/         #
            main.yml      #  <-- default lower priority variables for this
role
```

The `roles/apache/tasks/main.yml` aka the role playbook would look something like this:

YAML

```yaml
- name: Install apache
  ansible.builtin.package:
    name:
      - httpd
    state: installed

- name: Template apache config
  ansible.builtin.template:
    src: httpd.conf.j2
    dest: /etc/httpd/conf/httpd.conf

- name: Copy apache config
  ansible.builtin.copy:
    src: bar.txt
    dest: /etc/httpd/conf/
```

For Saltstack the state formula for apache would have the following structure:

Text Only

```
```

```
formulas/
    apache/
        init.sls
        httpd.conf.j2
        bar.txt
```

The `init.sls` aka the state formula would look something like this:

YAML

```yaml
install apache:
  pkg.installed:
    - name: httpd

run apache service:
  service.running:
    - name: httpd

/etc/httpd/conf/httpd.conf:
  file.managed:
    - source: salt://apache/http.conf.j2
    - user: root
    - group: root
    - mode: 644
    - template: jinja
    - defaults:
        custom_var: "default value"
        other_var: 123

/etc/httpd/conf/bar.txt:
  file.managed:
    - source: salt://apache/http.conf
    - user: root
    - group: root
    - mode: 644
```

Worth noting that Salt also allows (and even encourages) to use templating in formulas while Ansible does not allow templating in playbooks. For example we could add some variables to the apache config task from before depending on the OS of the minion:

YAML

```
/etc/httpd/conf/httpd.conf:
  file.managed:
    - source: salt://apache/http.conf.j2
    - user: root
    - group: root
    - mode: 644
    - template: jinja
    - defaults:
        custom_var: "default value"
        other_var: 123
{% if grains['os'] == 'Ubuntu' %}
    - context:
        custom_var: "override"
{% endif %}
```

Managing variables

Ansible has many (22 in fact) different places for variables with a hierarchy of variable precedence. In essence variables can be defined almost anywhere.

Saltstack has grains, variables that come *from* minions, and pillars, variables that go *to* minions as well as variables defined in formulas. In that sense, managing variables becomes a lot more explicit and readable when compared to Ansible.

Merging lists in Ansible:

YAML

```yaml
- name: Install apache
  ansible.builtin.package:
    name:
      - "{{ item }}"
    state: installed
  with_items: "{{ pkgs | combine(group_pkgs, list_merge='append_rp') }}"
```

Merging lists in Saltstack:

YAML

```yaml
install packages:
  pkg.installed:
    - name: {{ pillar['pkgs']['group_pkgs'] }}
```

Event based tasks

Ansible does not have event based tasks, however a developer preview exists for event driven Ansible features

Saltstack has reactors, which can be used to run formulas when defined events occur. This makes managing a dynamic set of hosts easy. For example master will run configuration formulas when a minion wakes up from powersave:

YAML

```yaml
reactor:                            # Master config section "reactor"
  - 'salt/minion/*/start':          # Match tag "salt/minion/*/start"
    - /srv/reactor/start.sls        # Things to do when a minion starts
    - /srv/reactor/monitor.sls      # Other things to do
```

Reusability of tasks

In Ansible it is possible to separate roles into separate repositories and include them in playbooks by linking the repositories. It is also possible to separate multiple components (like multiple roles) into Ansible collections and then to include them in playbooks. With collections it is possible to specify which roles and components to include.

With Saltstack it is possible to separate formulas and include them in the master configuration from the repository. It is possible to specify which formulas and paths to include and where to place them.

Conclusion

Points against Ansible and for Salt

First, configuration management often turns into smart variable handling as your actual configurations get more generic over time. Thus handling variables becomes really important. Saltstack is more more explicit ways in which it handles variables: they're defined in a pillar. Contrast this with the variable precedence for Ansible (there's a hierarchy of 22 different variable locations). Usually with Ansible an admin will usually only have to think about 4 different variable locations (command line, hostgroup, host and role variables), but the requirement to always needing to consider and build playbooks around variable precedence makes for less readable and harder to maintain repos.

Secondly, Saltstack uses master-agent architecture over Ansible's much much easier to get started with agentless approach. However, as a result, it tends not to scale as well as

Saltstack because you have to open an SSH connection to every machine you want to manage. Saltstack, by contrast, uses ZeroMQ to communicate with minions: minions listen for instructions on one ZeroMQ port on the master (4505) and post back results on another (4506). It's blazing fast and it scales really well. Works even when SSH goes down or the machines don't expose ssh (e.g. locked down machines that do not need SSH). Worth noting that in terms of actually writing tasks for minions, they are very similar and both use YAML and Jinja, so in that sense they are both as easy to onboard new people for writing new tasks. The learning curve increase for Saltstack comes from the initial master-minion setup, but even that is not a hugely complicated job all things considered.

Thirdly, master-agent architecture of Salt enables true event driven management with reactors which is especially good for dynamic host inventories (with hosts that often power on and off like gray nodes). So we could much easily define scenarios for minion wakeups.

For notifications, Ansible has easier integration for sending emails however if we split the runs (by hostgroup) then that means an email for each hostgroup. Aggregating errors will become a problem that needs a custom script. For Salt aggregating results and more granular control over notifications needs more work, but is a lot more flexible and does not need custom scripts.

Points for Ansible and against Salt

The support for Ansible is a lot better, there are more people using it in both enterprise and open source. Community is also a lot bigger. Having such a big user base behind them gives Ansible much stable and reliable updates compared to Salt where looking at the best practices even from 3 years ago differ from the ones recommended today.

Even if on paper SaltStack seems a lot better in many aspects on paper, in reality there are quite a few worrying problems. Like the git filesystem. It is possible to use `gitfs` alongside the default `rootfs` to automatically fetch Salt formulas, pillars etc from remote git repos, however `gitfs` is unfortunately buggy and full of python version mismatch errors among other things (Check the warnings in the `gitfs` documentation). Basic things like version controlling config management should be reliable and it begs the question of what else might be buggy in Salt.

Even variable handling gets messy when moving away from pillar only setups where you have to merge pillars with defaults and use map files in formulas which one might have to do to keep SaltStack performant when managing a large number of minions. Here's where the Ansible variable precedence actually comes in handy, because you can just define variables in the playbook and not have to worry about merging them.

Further resources for Saltstack

- Salt In 10 Minutes Walkthrough
- Salt User Guide
- Salt Documentation
- SaltStack Configuration Management Best Practices