Salt vs. Ansible (/post/salt-vs-ansible/)

A highly opinionated comparison between Salt and Ansible.

`Salt` (https://jensrantil.github.io/tags/salt) `Ansible` (https://jensrantil.github.io/tags/ansible) `provisioning` (https://jensrantil.github.io/tags/provisioning)

Salt vs. Ansible

Some time ago was set to evaluate configuration management systems. I've heard opinions from someone I respect that Puppet (http://puppetlabs.com) and Chef (http://www.getchef.com) were complicated beasts to get up and running, and since I am a Python guy I've generally always kept an eye on Ansible (http://www.ansible.com) and Salt (http://www.saltstack.com). Ruby is so far not my thing, but hey let's not start a flamewar about that!

Last year I got to spend a good 6 months working with Ansible for provisioning of servers. I became very familiar with the tool. In that project, Ansible was mostly chosen because it was easy to get started and had good documentation. The team I was working with was especially fond of the Best Practises (http://docs.ansible.com/playbooks_best_practices.html) documentation which got us started quickly and taught us what structure had worked previously.

A couple of weeks ago I went on a 10-day vacation in Japan, and for those who don't know me that usually equals me spending a fair amount of time of reading computer literature or documentation! Inbetween great sushi, Tokyo skylines, amazing skiing and a rough flue I found the Salt PDF-documentation (https://media.readthedocs.org/pdf/salt/latest/salt.pdf) to be great leisure!

I've also spent some time getting a small Salt setup up and running with their States (http://docs.saltstack.com/topics/tutorials/starting_states.html) system. Since I now feel like I have a pretty robust background in both systems, I feel obliged to make a highly opinionated comparison. Let's go:

Terminology

Both Salt and Ansible are originally built as execution engines. That is, they allow executing commands on one or more remote systems, in parallel if you want.

Ansible supports executing arbitrary command line commands on multiple machines. It also supports executing *modules*. An Ansible module (http://docs.ansible.com/modules.html) is basically a Python module written in a certain Ansible friendly way. Most standard Ansible modules are idempotent. This means you tell them the state you'd want your system to be in, and the module tries to make the system look like that.

Ansible also has the concept of a Playbook (http://docs.ansible.com/playbooks.html). A playbook is a file that defines a series of module executions for a set of hosts. A playbook can vary the hosts modules are executed on. This makes it possible to orchestrate multiple machines, such as take them out of a load balancers before upgrading an application.

Salt has two types of modules; execution modules (http://docs.saltstack.com/ref/modules/all/index.html) and state modules (http://docs.saltstack.com/ref/states/all/index.html). Execution modules are modules simply executes

something, it could be a command line execution, or downloading a file. A state module is more like an Ansible module, where the arguments define a state and the module tries to fulfill that end state. In general state modules are using execution modules to most of their work.

State modules are executed using the the `state` execution module. The state module also supports defining states in files, called SLS files. Which states to apply to which hosts is defined in a top.sls (http://docs.saltstack.com/ref/states/top.html) file.

Both playbooks and SLS files (usually) are written in YAML.

As a side-note I want to point out the a remote execution engine is highly useful for tasks such as doing inventory or kick off a certain operation on multiple machines.

Architecture

Salt is built around a *Salt master* and multiple *Salt minions* that are connecting to the master when they boot. Generally, commands are issued on the master command line. The master then dispatches those commands out to minions. Initially minions initiate a handshake consisting of a cryptographic key exchange and after that they have a persistent encrypted TCP connection. I could babble at length about how they are using the ZeroMQ (http://zeromq.org) library for communication, but let's just say that a Salt master can handle a _lot_ of minions without being overloaded thanks to ZeroMQ.

Because minions have a persistent connection to Salt master commands are fast to reach the minions. The minions also cache various data to make execution faster.

Ansible is masterless and it uses SSH as its primary communication layer. This means it is slower, but being masterless might make it slightly easier to run setup and test Ansible playbooks. Some claim it's also more secure because it doesn't require another server application. Read more about that under "Security" further down.

Ansible does support a ZeroMQ version, but it requires an initial SSH connection to setup. I tried it, and to be honest I didn't see that much of a speedup. I guess bigger improvements can be seen for larger playbooks and more hosts.

To keep track of machines Ansible recommends you have an *inventory* file that basically contains a list of hosts, grouped by groups, possibly with attributes added to either a group or a single host. You can have multiple inventory files, say, one for staging and one for production.

Salt also supports using SSH instead of ZeroMQ using Salt SSH (http://docs.saltstack.com/topics/ssh/). But beware, it's alpha software (and I haven't tried it)…

Community

For both projects I've had inquiries both on IRC and on mailing list. I've also contributed patches to them both - Python code and documentation patches. Here's a summary of my experience:

Ansible: Quick friendly response on IRC. The rest of the project seems to be less of a community effort and more a one-man-show lead by Michael DeHaan. I'm sorry to say, but I enjoy communities that are more open to improvements and has a friendlier tone. Ansible improvement issues are closed before fixed, which I feel like is hiding issues under the rug. Good news all inquiries has been getting a response.

Salt has so far proven to be a more welcoming community. IRC responses has mostly been quick and friendly. Sometimes I've had to resort to the mailing list. I've had a couple of mailing list issues that has taken ~4 days to get a response on, but it seems like most threads will get a follow-up response eventually.

My clear impression is that Salt has a more mature community in terms of tone, welcomeness and collaboration. I'm probably stepping on a lot of people's toes saying this, but hey - this is an oppinionated review!

Speed

While you might think that speed is not important when you have a few servers, I believe you are wrong. Being able to iterate fast is always important. Period. A slow startup slows you down. If something takes more than 30 seconds to compile I end up on Twitter, which means compiling just took 120 seconds. It's the same with deployment.

Ansible is always using SSH for initiating connections. This is slow. Its ZeroMQ implementation (mentioned earlier) does help, but initialization is still slow. Salt uses ZeroMQ by default, and it is _fast_.

Like said earlier, Salt has a permanent minion processes. This enables Salt to cache files for faster reexecution of things.

Code structure

My biggest pet peeve with Ansible modules is that they can't be imported (because they are executing code on import (https://github.com/JensRantil/ansible/blob/devel/library/files/copy#L189)). This means there's some magic involved when testing modules because you can't import a module like any other. I don't like magic. I prefer pure simple code. This is more of the Salt style.

Using less magic also means that it is more clear how to write test suites for Salt modules. Salt is thouroughly tested. It also made me happy to see that Salt comes with a three (http://docs.saltstack.com/topics/development/tests/) chapters (http://docs.saltstack.com/topics/tests/integration.html) on testing (http://docs.saltstack.com/topics/development/tests/unit.html), including the fact that they encourage mocking to enable testing of infrastructure if you don't have, say, a MySQL instance.

The above said, Ansible has a pretty clean code in general. I managed to navigate it pretty quickly. However, improving code structure (https://groups.google.com/d/msg/ansible-project/mpRFULSiIQw/jIIQdOSubnUJ) was clearly not of interest to the "community".

Both Ansible and Salt have regular installables via (https://pypi.python.org/pypi/ansible) PyPi (https://pypi.python.org/pypi/salt).

Vagrant

While talking about testing… DevOps people loves Vagrant. Until recently I had not worked with it. Vagrant comes with provisioning modules both for Salt and Ansible. This makes it a breeze to get up and running with a master+minion in Vagrant, or executing a playbook on startup.

Orchestration

Both Ansible and Salt supports orchestration. I'd say orchestration rules generally are easier to get an overview of in Ansible. Basically, a playbook is split up in groups of tasks, where each group matches to a set of hosts (or a hostgroup). Each group is executed chronologically according to order. The same comes for the executions order of tasks.

Salt supports events (http://docs.saltstack.com/topics/event/index.html) and reactors (http://docs.saltstack.com/topics/reactor/) to those events. This means a Salt execution can trigger things on another machine. Salt's execution engine also enables things such as monitoring and it's

going to be really interesting to see what comes of that in the future. For basic orchestration you can also use Overstate (http://docs.saltstack.com/ref/states/overstate.html) to set up various roles in a cluster in a special order.

Ansible wins here because of its simplicity. Salt wins in features because of it being able to react continuously to cluster changes.

Both Salt and Ansible also supports executing tasks over a window of machines. This is useful to make sure a service is always available through for example an upgrade.

Security

Ansible uses SSH for transport. SSH is a battle tested protocol. As long as the SSH server is correctly configured (with a good random number generator), I believe most people would assume an SSH client is secure.

Ansible can also easily connect as multiple non-root users to a single host. If you are extremelly picky about having processes running as `root` you should evaluate Ansible. That said, Ansible supports using `sudo` to execute its modules as `root` . If you don't want to connect over SSH as `root` , that is.

Salt uses its "own" AES implementation and key handling. By "own" I want to make a point out that it uses the PyCrypto (https://www.dlitz.net/software/pycrypto/) package for this. There has (http://www.cvedetails.com/vulnerability-list/vendor_id-12943/product_id-26420/version_id-155046/Saltstack-Salt-0.17.0.html) been security issues with Salt, but at the same time I think the architecture is so simple that security is fairly easy to maintain.

What's also important to note is that Salt runs its master and minions as `root` by default. This can be changed, but obviously it can be hard to install Debian packages etc. if you are not root. As for the master, you can configure it to allow the `salt` command as non-root. I highly recommend that.

Sensitive data

All sensitive data being rolled out will at some point need to reside on the provisioning machine. If the provisioning machine is a sysadmin's machine, which nowadays usually are laptops, you risk having that data stolen.

After deep and long thinking about this I believe the authoritarian master approach is a better one. This means sensitive data can be enforced to only reside on one locked down place (with encrypted backups, of course). Salt lets you store security credentials in "Pillars". Sure, a master intrusion would be devastating, but you only need to secure one machine. Not all developer machines in cafés, on trains and airports.

Obviously, Ansible users do have the option to always execute their playbooks from a secured machine that holds the sensitive data. But is this what people do?

Auditability

When talking about security I also think auditability is important. Here, Salt wins big. Every execution Salt does is stored (http://docs.saltstack.com/topics/jobs/index.html) for X number of days on the master. This makes it easy to both debug, but also see if there's been anything fishy going on.

Deployment

Ansible is definitely easier here. No deployment is needed. Sure, Salt supports SSH but the documentation mostly assumes ZeroMQ. But hey, SSH is slow anyway…

A nice thing about provisioning minions is that they are the ones connecting to the master. This makes it quick and easy to bootstrap a bunch of new machines quickly. The minion-connects architecture is also useful if you'd like to use something like Amazon's autoscaling features. Each autoscaled instance will automatically become a minion.

The Salt bootstrap script (https://github.com/saltstack/salt-bootstrap) is incredibly useful for bootstrapping and makes it a breeze. It handles a bunch of different distributions and is well documented (http://salt.readthedocs.org/en/latest/topics/tutorials/salt_bootstrap.html).

Learning curve

Ansible wins here. It's easier to get started and comprehend. Mostly because nothing else is needed than cloning the Ansible GIT repo, setting a couple of environment variables and starting to execute your playbooks.

Salt can run in masterless mode (http://docs.saltstack.com/topics/tutorials/quickstart.html). This makes it easier to get it up and running. However, for production (and stability) I recommend getting an actual master up and running.

Generally, Salt comes with more bells and whistles but the cost is that the learning curve is steeper. Salt is highly modular (http://docs.saltstack.com/topics/development/modular_systems.html). This is great in terms of code structure but requires more parts to be understood to fully grok Salt.

Upgrading

Upgrading Salt depends on how it was installed. For Debian based distributions there is an `apt` repository holding the latest Debian packages. So upgrading is simply `apt-get upgrade`. For Ubuntu, there is a PPA. Both repositories are actively maintained. The latest `2014.1.0` release that came out recently had its Debian/Ubuntu packages packaged within a week (and that was long!).

Upgrading Ansible is even simpler; You simply execute `git fetch && git checkout <tag>`. That's it.

Documentation

As of documentation, both projects have all information you need to get up and running, developing modules and configure setups. Ansible has historically had a better structure of its documentation than Salt. That said, there has been great (https://github.com/saltstack/salt/issues/10526) effort (https://github.com/saltstack/salt/pull/10792) to structure the Salt documentation recently. I contributed a lot of issues to that goal and most of them have been fixed.

Conclusion

To me, Ansible was a great introduction to automated server configuration and deployment. It was easy to get up and running and has great documentation.

Moving forward, the scalability, speed and architecture of Salt has it going for it. For cloud deployments I find the Salt architecture to be a better fit. I would not hesitate to use Salt in the future.

All this said, you should give both projects a spin before making your decision. They're fairly quick to set up and test.