Interview with Tom Hatch

RIK FARROW

Rik is the editor of *;login:*.
rik@usenix.org

Tom is the creator and principal architect of SaltStack. His years of experience as principal cloud architect for Beyond Oblivion, software engineer for Applied Signal Technology, and system admin for Backcountry.com provided real-world insight into requirements of the modern datacenter not met by existing tools. Tom's knowledge and hands-on experience with dozens of new and old infrastructure management technologies helped to establish the vision for Salt. Today, Tom is one of the most active contributors in the open source community. For his work on Salt, Tom received the Black Duck "Rookie of the Year" award in 2012 and was named to the GitHub Octoverse Top 10 list for the project with the highest number of unique contributors, rubbing shoulders with projects like Ruby on Rails and OpenStack.
thatch@saltstack.com

Rikki Endsley met Tom Hatch, CTO of SaltStack and creator of the Salt [1] open source project, during USENIX 2013 Federated ConferencesWeek. Rikki suggested I interview Tom and, when I found out that Salt uses ZeroMQ, my own curiosity was piqued.

ZeroMQ is one of a new breed of asynchronous messaging libraries. These libraries hide the complexity of developing your own queueing software. They also can use reliable multicast protocols, which can cut down on the amount of network traffic when many hosts will be receiving the same information. Message queueing also helps prevent overloading the server, a problem known as incast.

While Salt competes with more popular configuration management systems, like Puppet, Chef, and CFEngine, I thought that what makes it different also makes it important to take a deeper look at Salt.

Rik: There are lots of configuration management systems available today, but what caught my eye was the mention of Salt using ZeroMQ. Tell us about how Salt uses ZeroMQ and what the advantages of doing that are.

Tom: The big thing to remember about Salt is that it was first and foremost a remote execution system, and this is the reason I used ZeroMQ initially. ZeroMQ has been a great help with the configuration management system as well. The ability to have queued connections has allowed Salt to scale very well out of the box without modification and has been a foundation for Salt's flexibility.

The real benefit of ZeroMQ is that it has allowed us to do things where we have clean communication about anything in a deployment so that decisions can be made about configurations anywhere, which allows us to make a truly ad hoc distributed configuration management system. We are working on a UDP alternative to ZeroMQ to get past some of the limitations of ZeroMQ.

Rik: What specifically attracted you to using message queueing in the first place, instead of more traditional means of communication, like SSH over TCP?

Tom: Really, it was the PUB/SUB interface in ZeroMQ. PUB/SUB is very difficult to do with non-async systems and is very slow. ZeroMQ gave us an easy way to have a PUB/SUB interface that works well for remote execution. Beyond that, ZeroMQ has continually proven itself a great asset in communication with large groups of systems.

Salt does also ship with an optional SSH transport to have an "agentless" approach, but ZeroMQ is still substantially more scalable and faster.

To get more into the weeds, ZeroMQ's ability to queue connections and commands on the client side allows for very large numbers of systems to request information at once and all wait for the server end to be available rather than just bogging down the server. This is one of the key advantages in the Salt file server. This file server is built entirely on top of ZeroMQ from scratch and subsequently is capable of sending very large numbers of small files to large groups of systems with great efficiency.

Rik: You just mentioned the Salt file server. Is this how you store commands to be executed on clients? Or is this something else, like the general term for your server, which can interface with other CMDBs?

Tom: The Salt file server is just part of the Salt master. The master includes a fully functional file server and is a critical part of the CM system in Salt, because the salt minions (the agents) download the configuration management files from the master and compile them locally. The Salt file server can also be used to distribute large files like VM disk images. This is part of how Salt's cloud controller, salt-virt, works.

ZeroMQ is what makes serving files so scalable. It lets Salt minions download chunks of files and queue for their downloads, which keeps things humming and prevents overload on the master.

So the commands to be executed in the CM system are all stored in files (typically YAML files) and served via the Salt file system. Then the commands are compiled down to data structures on the minion and executed.

Salt has a lot of services in the master. For instance, the Pillar system is what allows the master to connect to external systems like CMDBs to get raw data which can be used when compiling the config management files into execution data.

Rik: I watched your presentation at UCMS '13 [2], and you mentioned the Pillar system. So Pillar can extract configuration information from existing systems, like Puppet or Chef, and then provide it to Salt minions? Do the minions use the original configuration agent, or are configuration actions performed natively by the minions?

Tom: Pillar is a system that allows for importing generic data, not just configuration data. It is minion-specific data that is generated on the master. So think of it more as a variable store that can be accessed from Salt's configuration management system.

Configuration management is performed natively in Salt by the minion. The great thing about Pillar is that it can be a top-level data store. Pillar makes it VERY easy to make generic configuration management formulas in Salt.

Let me try and sum up a few things since we are kind of jumping from topic to topic.

Salt is a remote execution platform that can execute generic commands on clients called minions. This generic command execution means that Salt is often used to orchestrate deployments that already use Puppet, and Wikipedia is a classic example.

But Salt has its own CM system which incorporates features not found in Puppet or Chef and ties directly into the remote execution system, allowing for many cross-communication routines to work—all over ZeroMQ.

Salt management is all about data, so the config management files that Salt uses, or formulas, can use Pillar (data generated on the master—optionally from external sources like CMDBs) or grain data (data generated on the minion, things like the OS version—kind of like Puppet's Facter [3]) as variables or to decide what routines should be executed.

So to answer your question, Pillar can pull data out of systems that Puppet uses, like Hiera, and reuse the data in its own configurations, or it can just call Puppet directly on the client. Or everything can be directly managed by Salt.

Rik: I find it interesting that Salt can handle data returned by the minion. Your UCMS presentation hinted at that. Getting configuration information back from the client, like Puppet's Facter, is certainly useful for configuration management. But because Salt is a remote execution system, I was wondering how it handles other information that might be returned by a client: for example, a failed compilation of a downloaded package because of missing library dependencies? How hard is it to set up Salt to handle other information, like my example, from a minion?

Tom: What you asked in the first question starts to hint at where Salt goes beyond configuration management. Since Salt is based on remote execution and events, any time something fails in a configuration management run, an event gets fired on the master. This means that the Salt Reactor can pick up the event and react to it. The reaction is arbitrary: it can reach out to the minion and try the install again, or it can react by destroying the virtual machine running the minion (assuming it is a VM, as an example) by communicating to the hypervisor, or to the cloud system, such as OpenStack, AWS, or any major cloud provider.

Also, since Salt's configuration management system is natively idempotent, it is easy to fire the configuration management to run again, either manually or via the Reactor.

References

[1] Salt: https://github.com/saltstack/salt.

[2] UCMS SaltStack talk: https://www.usenix.org/conference/ucms13/summit-program/presentation/Hatch.

[3] Puppet Facter: http://puppetlabs.com/facter.