Date: 04.06.2025

SaltStack Research: Concepts, Installation, Configuration

Contents

1. Introduction

Salt is:

- **A configuration management system**. Salt is capable of maintaining remote nodes in defined states.
- **A distributed remote execution system** used to execute commands and query data on remote nodes, either individual or by using an arbitrary selection criteria.

It enables better, faster, and more malleable remote execution. Salt accomplishes this by handling large loads of information, and thousands of individual servers quickly through a simple and manageable interface.

2. Features

1. **Simple** to set up and maintain
2. **Parallel execution** of commands across remote systems
3. **Built on proven tech** like ZeroMQ, AMQ, AES, etc.
4. **Python client interface** using modules or API
5. **Fast, Flexible and Scalable**
6. **Open source** under Apache 2.0 license

3. Key Concepts

1. **Remote Management System** – Uses the Remote Execution Engine which creates a high-speed, secure and bi-directional communication net for a group of systems to execute multiple commands across thousands of remote systems in seconds with a single execution.
2. **Configuration Management** – Uses Salt States which is a fast, flexible and easy-to-use configuration management system that stores all configuration/data inside an easily understood data structure (usually YAML with Jinja templates).
3. **Automation and Orchestration** – Uses a master-minion model that enables push and pull execution. Thus making the Salt's configuration management architecture, event-driven and self-healing. Salt can also operate in agent-based or agentless mode

4. Salt System Structure

Salt functions on a master-minion topology.

i. **Salt Master**
   a. A server running the salt-master service.
   b. The master provides a cohesive platform for orchestration and automation between managed systems.

ii. **Salt Minion**
   a. Any system or device managed by Salt.
   b. Either running the salt-minion service or be agentless using salt-ssh or salt-proxy.
   c. A minion running the service may execute commands without a master in stand-alone mode.

iii. **Salt Grain**
   a. Salt comes with an interface to derive information about the underlying system called the grains interface (presents Salt with grains of information).
   b. Grains are collected for the operating system, domain name, IP address, kernel, OS type, memory, and many other system properties. Custom grain data creation is also possible.
   c. Grain data is relatively static. However, grain data is refreshed when system information changes (such as network settings) or when a new value is assigned to a custom grain.

iv. **Open event system (event bus)**
   a. The event system is used for inter-process communication between the Salt Master and Salt Minions.
      In the event system:
      i. Events are seen by both the master and minions.
      ii. Events can be monitored and evaluated by both.
   b. All minions see jobs and results by subscribing to events published on the event system
   c. Salt uses a pluggable event system with two layers:
      i. ZeroMQ (0MQ) - The current default socket-level library providing a flexible transport layer.
      ii. Tornado - Full TCP-based transport layer event system.

v. **Salt States**
   a. A fast, flexible and easy-to-use configuration management system.
   b. It stores all configuration/data inside an easily understood data structure (usually YAML with Jinja templating).

vi. **Salt Pillar**

   a. Salt's pillar feature takes data defined on the Salt Master and distributes it to minions as needed.
   b. Salt pillar brings data into the cluster from the opposite direction as grains. While grains are data generated from the minion, the pillar is data generated from the master.
   c. Information transferred using pillar has a dictionary generated for the targeted minion and encrypted with that minion's key for secure data transfer.
   d. Pillar data is encrypted on a per-minion basis, which makes it useful for storing sensitive data specific to a particular minion.

vii. **Beacons**

   a. The beacon system is a monitoring tool that can listen for a variety of system processes on Salt Minions.
   b. Beacons can trigger reactors which can then help implement a change or troubleshoot an issue.
   c. Beacons are used for a variety of purposes, including:
      i. Automated reporting
      ii. Error log delivery
      iii. Microservice monitoring
      iv. User shell activity
      v. Resource monitoring

viii. **Reactors**

   a. When coupled with reactors, beacons can create automated responses using pre-written remediation states to infrastructure and application issues.
   b. Reactors can be applied in a variety of scenarios:
      i. Infrastructure scaling
      ii. Notifying administrators
      iii. Restarting failed applications
      iv. Automatic rollback

ix. **Runners**

   a. Salt runners are convenience applications executed with the salt-run command.
   b. Salt runners work similarly to Salt execution modules.
   c. However, they execute on the Salt Master instead of the Salt Minions.

A Salt runner can be a simple client call or a complex application.

5. Salt Configurations

Although the standard Salt configuration model is the master-minion model, minions do not necessarily have to be managed by a master. Salt provides additional options for managing minions:

Masterclass	Enables use of Salt's configuration management for a single machine without calling out to a Salt master on another machine.
Salt Cloud	Provisions and manages systems on cloud hosts or hypervisors. It uses the Saltify drive to install Salt on existing machines (virtual or bare metal).
Proxy Minions	Send and receive commands from minions that, for whatever reason, can't run the standard salt-minion service.
Agentless	Use SSH to run Salt commands on a minion without installing an agent.

6. Salt Installation

1. Check the system requirements to ensure platform compatibility for Salt and open the required network ports. Also ensure you have correct permissions to install packages on the targeted nodes.
2. Install the salt-master service on the node that will manage other nodes by sending commands. Then install the salt-minion service on the other nodes.
    a. Linux-based OS – Use the bootstrap script or manually install Salt using the instruction for each OS.
    b. Windows or MacOS – Download and run the installer file for that system.
3. Configure the Salt minions to add the DNS/hostname or IP address of the Salt Master they will connect to. Add additional configurations to master and minions as needed
4. Start the service on the master, then the minions.
5. Accept the minion keys after the minion connects.
6. Verify that the installation was successful by sending a test ping.
7. Install third-party Python dependencies needed for specific modules.

7. Configuring Salt

Salt configuration is very simple. The configuration files will be installed to /etc/salt and are named after the respective components, /etc/salt/master, and /etc/salt/minion.

i. **Configuring Master**

By default the Salt master listens on ports 4505 and 4506 on all interfaces (0.0.0.0).
To bind Salt to a specific IP, redefine the "interface" directive in the master configuration file, typically /etc/salt/master, as follows:

```
- #interface: 0.0.0.0
+ interface: 10.0.0.1
```

After updating the configuration file, restart the Salt master

ii. **Configuring Minions**

By default a Salt Minion will try to connect to the DNS name "salt"; if the Minion is able to resolve that name correctly, no configuration is needed.
If the DNS name "salt" does not resolve to point to the correct location of the Master, redefine the "master" directive in the minion configuration file, typically /etc/salt/minion, as follows:

```
- #master: salt
+ master: 10.0.0.1
```

After updating the configuration file, restart the Salt minion.

iii. **Proxy Configuration**

A proxy minion emulates the behaviour of a regular minion and inherits their options. Similarly, the configuration file is /etc/salt/proxy and the proxy tries to connect to the DNS name "salt". In addition to the regular minion options, there are several proxy-specific.

8. Running Salt Services

1. Start the master in the foreground:
   `salt-master` (to daemonize the process, pass the -d flag)
2. Start the minion in the foreground:
   `salt-minion` (to daemonize the process, pass the -d flag)

9. Salt Key Exchange

For security, Salt uses key-based authentication
Two types of keys are used in Salt:

- **RSA**
  - Backbone to Salt's authentication and encryption model.
  - All Salt daemons run with unique RSA keys.
  - Generated by minions and master when they start for the first time and then used for PKI-based authentication.
  - These keys are used by minions to authenticate the AES key to master.
  - Each minion presents a public key to the Salt master. The key is then examined, compared, and explicitly accepted by an administrator.
- **AES**
  - The master also sends a rotating AES key that is used to encrypt and decrypt messages sent by the Salt master.
  - The returned AES key is encrypted using the public key initially sent by the Salt minion, and can therefore be decrypted only by the same Salt minion.

The AES key is rotated in either of these conditions:

- Every 24 hours on the master
- When the master is restarted
- When a minion key is deleted.

The key rotation allows the master to lock out minions that are not authenticated and it allows system-wide communication encryption.

10. Managing keys

Most minion management is handled through a client called salt-key that runs on the Salt master. This client is used for managing which minions are available to a master.
You can use the salt-key command to interface with the authentication system to accept, reject, and otherwise manage keys.

Flags to manage keys:

Flag	Description
-a <minion ID>	Accepts a specific minion’s key. This flag needs to be followed by an argument that includes the ID of the minion key that you want to accept.
-A	Accept all the keys.
-d <minion ID>`	Deletes a specific minion’s key. This flag needs to be followed by an argument that includes the ID of the minion key that you want to delete.
-L	List all minion IDs.

11. Using Jinja with Salt

By default, SLS files are rendered as Jinja templates and then parsed as YAML documents. Since the state system only processes raw data, the SLS files can be any structured format that is supported by a Python renderer library.

i. **Using Renderer Pipes**

Render pipes allow the output of render engines to be piped into each other, similar to Unix pipes.

To pipe the output of the Jinja renderer into the YAML renderer, place the following shebang on the first line of the SLS file:

```
#!jinja|yaml
```

ii. **Using Jinja Renderer**

Jinja is used to template SLS files when more flow control is needed.

By default, Salt uses the Jinja templating language to manage programmatic control over the YAML files.

Jinja Delimiters	Definition
{% … %}	Define a Jinja statement
{%- … -%}	Define a Jinja statement, but remove whitespace from beginning and end of line
{{ … }}	Print a Jinja expression or call a Salt execution directly at the desired location of the file
{{- … -}}	Jinja expression removing whitespace from beginning and end of line
{# … #}	Jinja comments - not included in output after being rendered
{#- … -#}	Jinja comments removing whitespace from beginning and end of line

12. Pros and Cons

The technology underlying Salt and SaltStack Enterprise has strong and weak points depending on the user's skills, as well as the deployment upon which the user will act.

Pros:

- Best for highly scalable and resilient workplaces.
- Easy to install and set up.
- YAML syntax across all the scripting tasks and Python support lowers the learning curve for beginners.
- Most economical automation solutions for enterprises as it is Open-Source.

Cons:

- Difficult to manage and check documentation
- Not a good solution for OS other than Linux
- SaltStack Enterprise’s GUI is not feature rich
- Target states cannot be checked in a specific order, thus diminishing the ability to program dependencies between systems.

13. SaltStack powered applications

Salt powers VMware by Broadcom's Tanzu Salt (previously Aria Automation Config / vRealize Automation SaltStack Config / SaltStack Enterprise), and can be found under the hood of products from Juniper, Cisco, Cloudflare, Nutanix, SUSE, and Tieto, to name a few.

Tanzu Salt is not a standalone program and is available via Tanzu Platform.

Product Links:

Tanzu Salt: VMware Tanzu Salt
Tanzu Platform: VMware Tanzu Platform

14. VMWare Aria Automation

VMWare Aria Automation documentation: VMware Aria Automation 8.18
Install VMWare Aria Automation via the VMware Aria Suite Easy Installer

Installation steps:

1. Log in to the Broadcom Support Portal and from the My Dashboard view, select VMware Cloud Foundation.
    a. Go to My Downloads and select VMware Aria UniversalVMware Aria Universal Enterprise. Click Subscription.
    b. From the list of primary downloads that appears, select the View Group link on the line for VMware Aria Suite Lifecycle.
    c. Click the cloud icon to download the binary for VMware Aria Automation Easy Installer.
2. After you download the file, mount the lcm-installer.iso file.
3. Browse to the folder vrlcm-ui-installer inside the CD-ROM.
4. The folder contains three subfolders for three operating systems. Based on your operating system, browse to the corresponding operating system folder inside the vrlcm-ui-installer folder.
5. Click the installer file in the folder.

OS	File Path
Windows	lcm-installer\vrlcm-ui-installer\win32
Linux	a. Login to Linux VM. b. Run apt-get install p7zip-full. c. Run 7z x vra-lcm-installer.iso. d. Run chmod +x vrlcm-ui-installer/lin64/installer e. Run apt install libnss3 (required only if the libnss3 component is not installed.) f. Run vrlcm-ui-installer/lin64/installer.
Mac	vrlcm-ui-installer/mac/Installer

6. The VMware Aria Suite LifecycleVMware Aria Automation Easy Installer UI is specific to the operating system. Ensure that you are using the valid UI folder path to run the installer.