Salt Scheduler Exercises

Setup

Checkout Git repo:
https://github.com/garethgreenaway/salt-scheduler-tutorial

cd salt-scheduler-tutorial

vagrant up

vagrant ssh

All Salt services are stopped and disabled, we want to run them manually so we can see the actions being logged. If comfortable start tmux otherwise create multiple ssh connections using vagrant ssh.

As root, using sudo, start the salt-master, the salt-minion, and the salt-event runner in different windows or tmux sessions:

sudo salt-master -l debug
sudo salt-minion -l debug
sudo salt-run state.event pretty=True

The minion will attempt to connect to the master and wait until it's key has been signed. Switching to another window or tmux session, run the following to see pending key requests:

sudo salt-key -L

You should see a key for the minion named salt in the “Unaccepted keys” section. Let’s accept that key. Run the following:

sudo salt-key -yA

To accept the key for the minion. Running “sudo salt-key -L” again and you’ll see the minion is now listed in the “Accepted Keys” section.

Let’s verify that we can run commands against our minion. Running the following should return a simple “True” response from our single minion:

*sudo salt * test.ping*

And running the following should return the version of Salt that our minion is running:

*sudo salt * test.version*

Using the Scheduler

Let’s try some scheduling. In the /vagrant directory there is a file called schedule.conf. Let’s copy that over to the salt minion configuration directory.

sudo cp -v /vagrant/schedule.conf /etc/salt/minion.d

And stop the Salt minion in the window or tmux session it is running in and start it up again so the schedule configuration is loaded.

In the windows or tmux session where the minion is running we should now start to see some log messages showing our scheduled

job running as well as some events from that job where we have the event runner running.

Lets go ahead and clear out the scheduler by using the purge function in the schedule module. Running the following will clear everything out.

sudo *salt * schedule.purge*

Then double checking the output from:

sudo *salt * schedule.list*

Should show that the schedule for our minion is now empty.

We can easily bring the previous schedule items back from the saved configuration file using the reload function in the schedule module. Run the following command to reload the schedule which is saved:

*sudo salt * schedule.reload*

Then double checking the output from:

sudo *salt * schedule.list*

Should show that the schedule for our minion has been restored.

Let’s look at adding some schedule items in using the execution module.

Using the schedule execution module

Before we do that let's clear out the schedule by removing the schedule configuration file we copied over before and restart the salt minion in the tmux or window it's running in.

rm /etc/salt/minion.d/schedule.conf

As we saw in the examples easier, we can easily add items into the scheduler using the schedule module. Let's go ahead and add a job in named "job1" that runs the test.version function and wil execue every 20 seconds.

*salt * schedule.add job1 function='test.version' seconds=20*

Watching both the salt minion logs and the event runner, we should see the call the module as well as when our job starts to run.

Let's add a second job, called job2:

*salt * schedule.add job2 function='test.ping' seconds=30*

This one will run test.ping and will run every 30 seconds. Which we should now also see in the logs and the event runner.

Let's try out the various disable functions. We'll start by disabling the entire schedule.

*salt * schedule.disable*

Once this runs, we should not see any additional jobs being run the Salt scheduler. Let's enable it.

*salt * schedule.enable*

Now let's disable a single job, let's disable job1.

*salt * schedule.disable_job job1*

Now in the logs and the event runner, we should only see information about job2 running.

We have job1 and job2 in place, job1 is currently disabled and we want to change something about job2. Let's start with something simple and change how often it runs. For this task, we'll use the modify function in the schedule module. Running the following command will change job2 from running every 30 seconds to every 3600 seconds or one hour:

*salt * schedule.modify_job job2 seconds=3600*

On occasion we'll want to run the jobs in the Salt schedule manually, to accomplish this we'll use the run_job function. Using the command below will cause job2 to run.

*salt * schedule.run_job job2*

Now that we have our jobs setup how we want them it will be hefpful to be able to save them so that the next time we restart our Salt minion or the server that Salt is running on the schedules are the same. For this we can use the save function in the schedule module. The following command will cause the current contents of the Salt scheduler to be saved to disk on the minion.

*salt * schedule.save*

Using the schedule state module

The last thing we'll look at using to interact with the Salt scheduler is using the schedule state modules. The execution modules work very well but using the Salt state system brings many advantages over the execution modules. Managing schedule items through the state systems allows you to use the built in templating systems. Inside the /srv/salt directory there are several state files, we'll look at the *schedule.sls* file first.

It's a fairly simple state file that is calling the present function in the schedule state module. We're passing in two arguments, *function* and *seconds*. This will add a schedule item to the scheduler which will run the test.version function every 10 seconds.

Let's go ahead and apply this state file to add the schedule item to the scheduler. Before we do that let's clear any jobs left around from the last examples.

sudo *salt * schedule.purge*

And now apply the state file by calling the apply function in the state module and giving it the name of the state file we wish to run.

sudo *salt * state.apply schedule*

We should now begin to see in our Salt minion logs and the event runner, the job running.

One of the useful features we saw earlier when adding items to the scheduler is the use of metadata. This can be particularly useful when looking for a job that has already run. Looking back at our list of available state files, there is one named *schedule_metadata.sls* which looks almost identical to the previous one except it has an additional parameter called metadata that has as it's argument a dictionary. Running the state.apply function again but with this new state file, we'll see the previously added schedule item is updated to include this new parameter.

sudo *salt * state.apply schedule_metadata*

Watching the logs and the event runner, we should now see the metadata information when the job is run.

Using an additional Salt runner, we can now search for previous runs of this job that included our metadata information. Running the following:

sudo salt-run jobs.list_jobs search_metadata="{'foo': 'bar'}"

Will display all the previous jobs that include that metadata. There are additional arguments that we could provide to this Salt runner to narrow down the search by either the function we're executing or the Salt minion that the job is running on.

The other very useful feature of Salt that we saw earlier was using the returner system with the Salt scheduler. By default all jobs are sent back to the Salt master but they are not preserved if the master is stopped and started again. By using a returner we can send all job data to an external system and ensure we can refer back to it later.

Looking back at our available state files we have one last one to look at, *schedule_returner.sls*. Inside this state file we'll see that this one builds on the previous one with an additional parameter included that specifies a returner. In this case we'll be using the *rawfile_json* returner which will use a local file on the Salt master and for each event write a line containing that event data in a json format.

Running the state.apply function again but with this new state file, we'll see the previously added schedule item is updated to include this new parameter.

sudo *salt* * *state.apply schedule_returner*

Watching the logs and the event runner, we'll the job running and if we check in the directory /var/log/salt/ we should see a file named events. Looking in that file we'll see the job return data in json format.

We've just scratched the surface of what is possible with the schedule within Salt. I encourage everyone to play around with the various options as well as the additional time elements.

BONUS

As a bonus exercise, please explore the *skip* and *postpone* functionality. Modify one of the existing schedule state files, create a new one, or use the schedule module to add a new job to the schedule running on a frequency that allows time to query, skip/postpone, and monitor (eg. every 60 seconds). Then query the scheduler for the next available execution time, using the *show_next_fire_time* function and pass that value along to the *skip* function. Additionally take the value returned from *show_next_fire_time* and pass it along to the postpone_job function with a new desired run time.