Salt - Conductor Framework Grains

Overview

This page describes the list of salt grains that are created and used throughout the salt ecosystem when provisioning and managing vm instances with the SaltStack conductor runner. This runner uses salt-cloud to provisioning instance in the cloud with specific configurations (role based etc..) with lots of hooks to provide and end to end delivery of product, systems and services.

The conductor framework consists of multiple aspects. Conductor runner is the central automation tool, but there is a specific Pillar and State data model that goes hand in hand with the conductor to provide an enterprise scalable framework for Saltstack.

The model and the runner together create and consume quite a few salt grains specific to the implementation. Salt itself creates a lot of grains on each minion that is registered to the master. Salt-cloud also creates grains when used to provision new instances.

Grains are a very important part of the Salt ecosystem and are used in many ways when targeting minions.

The conductor creates and uses grains on all instances that it will create and uses them for targeting in numerous situations like finding cluster members, identifying cloud, cluster and provision id's, orchestration hooks etc...

Each product or technology specific salt state may use grains to learn what tasks to do. This is a recommendation in the Conductor framework.

A minion can have almost unlimited grains of any type (string, int, list, dictionary). This page documents the static and dynamic salt grains that will be applied to minions created within the conductor tool with a brief explanation. This is not intended to be a deep dive on the topic of Saltstack grains and the extend to what they can be used beyond how the conductor is using them in the framework.

Using grains in states

As noted above, the conductor framework will create many grains on new minions depending on the role and provisioning type. Some of the these conductor specific grains are "baked in" to the tooling, but there are also any number of custom grains that can be set on instances based on the product group/role specific configuration per environment.

Grains are not only powerful for targeting, by they are also useful when developing salt states within the framework. States can refer to grains for many things such as verification checks to make sure the state is appropriate for the minion based on its roles, gathering information required to complete the full setup such as cluster configuration and much more.

Example salt states using grains to determine what action should occur.

using grains to validate the role

```
{% set role = 'devops.consul-server' %}
{% set service_installed = salt['service.available']('consul-server') %}
{% if service_installed == True %}
  {% set iscomposite = salt['grains.get']('composite.role', None) %}
  {% if (grains['role']|lower == role) or
        ((not iscomposite == None) and (role in
grains['composite.role']) ) %}
    {% set consulconfig_path =
salt['pillar.get']('devops.role:consul-server:config-location', None) %}

    {% if not consulconfig_path == None %}
update.config stop consul-server service:
  service.dead:
    - names:
      - consul-server

re-generate local {{role}} config:
  module.run:
    - name: common_consul.create_devops_server_config
    - config: {{ consulconfig_path }}
    - env: {{ env }}

update.config startup {{role}} service:
  service.running:
    - names:
      - consul-server

    {% endif %}
  {% endif %}
{% endif %}
```

There is actually a lot of logic going on in the previous example. First, some jinja variables are being set, a service available check is being done, the verification of role grains, checks for a local minion file, then runs through a few actions. One of the actions is calling a custom execution module to do the actual file manipulation.

Jinja is used quite extensively throughout the conductor framework as far as state design. It is the default templating language used by Saltstack.

This next examples is basic example of enumerating the role grains from state being applied to a minion. Reference *salt://test/verify_grains.sls*

Example using grains in a salt state

using grains to get cluster specific information

```
{% set role = 'salty.nifi' %}
{% set _var = None %}
{% if (grains['role']|lower == role) or
        ((not iscomposite == None) and (role in
grains['composite.role']) ) %}

  {% set _var = grains['cluster.members'] %}

show cluster grains {{role}}:
  cmd.run:
    - output_loglevel: quiet
    - name: |
        echo {{grains['cluster.members']}}

  {% if  _var is iterable and var is not string %}

{{role}} has correct grain type:
  cmd.run:
    - name: |
        echo cluster.members is a List

  {% endif %}

{% endif %}
```

This is the output when the above state is applied to a minion from the salt master

verify grain salt state output

```
salty-nifi-01.clid-1.us-east-1a.test.foobar.com:
----------
          ID: show cluster grains salty.nifi
    Function: cmd.run
        Name: echo ['salty-nifi-02.clid-1.us-east-1a.test.foobar.com',
'salty-nifi-01.clid-1.us-east-1a.test.foobar.com',
'salty-nifi-dummy-01.clid-1.us-east-1a.test.foobar.com',
'salty-nifimgr-01.clid-1.us-east-1a.test.foobar.com']

      Result: True
     Comment: Command "echo
['salty-nifi-02.clid-1.us-east-1a.test.foobar.com',
'salty-nifi-01.clid-1.us-east-1a.test.foobar.com',
'salty-nifi-dummy-01.clid-1.us-east-1a.test.foobar.com',
```

```
'salty-nifimgr-01.clid-1.us-east-1a.test.foobar.com']
              " run
     Started: 17:57:59.359968
    Duration: 11.672 ms
     Changes:
              ----------
              pid:
                  22545
              retcode:
                  0
              stderr:
              stdout:
                  [salty-nifi-02.clid-1.us-east-1a.test.foobar.com,
salty-nifi-01.clid-1.us-east-1a.test.foobar.com,
salty-nifi-dummy-01.clid-1.us-east-1a.test.foobar.com,
salty-nifimgr-01.clid-1.us-east-1a.test.foobar.com]
----------
          ID: salty.nifi has correct grain type
    Function: cmd.run
        Name: echo cluster.members is a List

      Result: True
     Comment: Command "echo cluster.members is a List
              " run
     Started: 17:57:59.371940
    Duration: 8.541 ms
     Changes:
              ----------
              pid:
                  22546
              retcode:
                  0
              stderr:
              stdout:
                  cluster.members is a List

Summary for salty-nifi-01.clid-1.us-east-1a.test.foobar.com
------------
Succeeded: 2 (changed=2)
Failed:    0
```

```
------------
Total states run:     2
Total run time:  20.213 ms
```

Example using grains in an orchestrate state

This state would run on the salt master since it is an Orchestration State (a server side state that is applied using the Orchestrate Runner in Saltstack). *This example was taken from one of the test orchestration state hooks.*

orchestration state using salt grains

```
{% set minion_target = salt['pillar.get']('target-minion', None) %}
{% set role_target = salt['pillar.get']('target-role', 'salty.nifi') %}
{% set cpid = salt['pillar.get']('cpid', None) %}
{% set resizing = salt['pillar.get']('resizing', False) %}

{% if not minion_target == None and not role_target == None %}

{{role_target}} pre startup orchestration test all:
  {% set target = "'( G@role:" + role_target + " or G@composite.role:*"
+ role_target + "* )'" %}
  salt.state:
    - tgt: {{ target }}
    - tgt_type: compound
    - sls: salty.nifi.pre_orch_test

  {% if resizing == False %}
{{role_target}} pre startup orchestration primary only:
  {% set target = "'G@internal.role:manager and ( G@role:" + role_target
+ " or G@composite.role:*" + role_target + "* )  and ( L@" +
minion_target + " )'" %}
  salt.state:
    - tgt: {{ target }}
    - tgt_type: compound
    - sls: salty.nifi.pre_orch_test

{{role_target}} pre startup orchestration dummy only:
  {% set target = "'G@internal.role:dummy and ( G@role:" + role_target +
" or G@composite.role:*" + role_target + "* )  and ( L@" + minion_target
+ " )'" %}
  salt.state:
    - tgt: {{ target }}
    - tgt_type: compound
    - sls: salty.nifi.pre_orch_test

  {% endif %}

{{role_target}} pre startup orchestration secondary only:
  {% set target = "'G@internal.role:secondary and ( G@role:" +
role_target + " or G@composite.role:*" + role_target + "* )  and ( L@" +
minion_target + " )'" %}
  salt.state:
    - tgt: {{ target }}
    - tgt_type: compound
    - sls: salty.nifi.pre_orch_test

{% endif %}
```

The above example shows how to target specific nodes of a cluster (aka internal roles), and assuring that the tasks/actions in the state would only get applied to the minions that match all of product.group, role and internal.role grains.

Sample output of grains created on a new instance

This examples shows the output of running this salt command on the salt master

```
salt '*' grains.items
```

The output shows the full list of grains configured on the minion. You can see grains are very 'granular' and thus provide a great way to target machines for state or other actions.

The grains specific to Conductor are explained with comments.

```
salty-nifi-01.clid-1.us-east-1a.test.foobar.com:            <--- Minion
ID. This is also set as the hostname grain (as well as the machine
hostname)
    ----------
    SSDs:
        - xvda
        - xvdb
        - xvdf
        - xvdg
        - xvdh
        - xvdi
        - xvdj
    biosreleasedate:
        08/24/2006
    biosversion:
        4.2.amazon
    cloud.nifi.cluster.id:                                  <--- A
unique environment scoped product cluster grain
        1
    cluster.location:                                       <---
Cloud (AWS) region the instance is residing
        us-east-1a
    cluster.members:                                        <---
List type grain containing all member nodes of this cluster
        - salty-nifi-02.clid-1.us-east-1a.test.foobar.com
        - salty-nifi-01.clid-1.us-east-1a.test.foobar.com
        - salty-nifi-dummy-01.clid-1.us-east-1a.test.foobar.com
        - salty-nifimgr-01.clid-1.us-east-1a.test.foobar.com
    cluster.members.ip:                                     <---
List type grain containing all member nodes IP address of this cluster
        - 10.123.20.169
        - 10.123.20.151
        - 10.123.20.144
        - 10.123.20.233
    common.post.startup.orchestration:                      <---
```

```
this instance (role) has a common (env scoped) post startup
orchestration state hook
        - orch.common.post-startup                                  <---
represents the salt state to run salt://orch/common/post-startup
    cpid:
        8907168268531761081
    cpu_flags:
        - fpu
        - vme
        - de
        - pse
        - tsc
        - msr
        - pae
        - mce
        - cx8
        - apic
        - sep
        - mtrr
        - pge
        - mca
        - cmov
        - pat
        - pse36
        - clflush
        - mmx
        - fxsr
        - sse
        - sse2
        - ht
        - syscall
        - nx
        - rdtscp
        - lm
        - constant_tsc
        - rep_good
        - nopl
        - xtopology
        - eagerfpu
        - pni
        - pclmulqdq
        - ssse3
        - fma
        - cx16
        - pcid
        - sse4_1
        - sse4_2
        - x2apic
        - movbe
        - popcnt
```

```
        - tsc_deadline_timer
        - aes
        - xsave
        - avx
        - f16c
        - rdrand
        - hypervisor
        - lahf_lm
        - abm
        - invpcid_single
        - fsgsbase
        - bmi1
        - avx2
        - smep
        - bmi2
        - erms
        - invpcid
        - xsaveopt
    cpu_model:
        Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz
    cpuarch:
        x86_64
    disks:
    dns:
        ----------
        domain:
        ip4_nameservers:
            - 10.120.20.10
            - 10.123.20.10
        ip6_nameservers:
        nameservers:
            - 10.120.20.10
            - 10.123.20.10
        options:
        search:
            - elizalive.com
            - ec2.internal
        sortlist:
    domain:
        clid-1.us-east-1a.test.foobar.com
    fqdn:
        salty-nifi-01.clid-1.us-east-1a.test.foobar.com
    fqdn_ip4:
        - 10.123.20.151
    fqdn_ip6:
        - fe80::cf2:f0ff:fe73:a64e
    gid:
        0
    gpus:
    group.post.startup.orchestration:                       <--- this
```

```
instance (role) has a product group scoped post startup orchestration
state hook
        - orch.salty.post-startup
    group.pre.provision.orchestration:                        <--- this
instance (role) has a product group scoped pre provision orchestration
state hook
        - orch.salty.pre-provision
    group.pre.startup.orchestration:                          <--- this
instance (role) has a product group scoped pre startup orchestration
state hook
        - orch.salty.pre-startup                              <---
represents the salt state to run salt://orch/salty/pre-startup
    groupname:
        root
    host:
        salty-nifi-01
    hwaddr_interfaces:
        ----------
        eth0:
            0e:f2:f0:73:a6:4e
        lo:
            00:00:00:00:00:00
    id:
        salty-nifi-01.clid-1.us-east-1a.test.foobar.com      <---
conductor will set the Name aws tag to the hostname local var. Salt sets
to id
    init:
        systemd
    internal.role:                                            <--- the
cluster internal role
        secondary
    ip4_interfaces:
        ----------
        eth0:
            - 10.123.20.151
        lo:
            - 127.0.0.1
    ip6_interfaces:
        ----------
        eth0:
            - fe80::cf2:f0ff:fe73:a64e
        lo:
            - ::1
    ip_interfaces:
        ----------
        eth0:
            - 10.123.20.151
            - fe80::cf2:f0ff:fe73:a64e
        lo:
            - 127.0.0.1
```

```
        - ::1
    ipv4:
        - 10.123.20.151
        - 127.0.0.1
    ipv6:
        - ::1
        - fe80::cf2:f0ff:fe73:a64e
    kernel:
        Linux
    kernelrelease:
        3.10.0-693.11.6.el7.x86_64
    locale_info:
        ----------
        defaultencoding:
            UTF-8
        defaultlanguage:
            en_US
        detectedencoding:
            UTF-8
    localhost:                                                  <---
conductor sets this to the Name tag value
        salty-nifi-01.clid-1.us-east-1a.test.foobar.com
    lsb_distrib_codename:
        CentOS Linux 7 (Core)
    lsb_distrib_id:
        CentOS Linux
    machine_id:
        609bbd29e32a4898e604f49bff82a88c
    manufacturer:
        Xen
    master:
        10.123.20.125
    mdadm:
    mem_total:
        1837
    node.index:                                                 <---
conductor creates this based on how many existing same name pattern
instances in the env
        1
    node_location:                                              <--- Cloud
(AWS) region the instance is residing
        us-east-1a
    nodename:
        ip-10-123-20-151.elizalive.com
    num_cpus:
        1
    num_gpus:
        0
    os:
        CentOS
```

```
    os_family:
        RedHat
    osarch:
        x86_64
    oscodename:
        CentOS Linux 7 (Core)
    osfinger:
        CentOS Linux-7
    osfullname:
        CentOS Linux
    osmajorrelease:
        7
    osrelease:
        7.4.1708
    osrelease_info:
        - 7
        - 4
        - 1708
    path:
        /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/bin:/sbin
    pid:
        11203
    pillar.environment:                                     <---
conductor sets this for convenience. The minion config is where salt
needs it to be
        test
    product.group:                                          <---
conductor sets this for every instance. VERY IMPORTANT GRAIN!
        salty
    productname:
        HVM domU
    ps:
        ps -efHww
    pythonexecutable:
        /bin/python2
    pythonpath:
        - /usr/bin
        - /usr/lib64/python27.zip
        - /usr/lib64/python2.7
        - /usr/lib64/python2.7/plat-linux2
        - /usr/lib64/python2.7/lib-tk
        - /usr/lib64/python2.7/lib-old
        - /usr/lib64/python2.7/lib-dynload
        - /usr/lib64/python2.7/site-packages
        - /usr/lib/python2.7/site-packages
    pythonversion:
        - 2
        - 7
        - 5
        - final
```

```
        - 0
    role:                                                   <--- role
grain set to format productgroup.product
        salty.nifi
    role.base:                                              <--- if
product is shared/common, this grain will also be set
        nifi
    role.id:                                                <---
conductor uses this. Usually same as the role grain
        salty.nifi
    role.post.startup.orchestration:                        <--- this
instance (role) has a role scoped post startup orchestration state hook
        - orch.salty.post.nifi
    role.pre.provision.orchestration:                       <--- this
instance (role) has a role scoped pre provision orchestration state hook
        - orch.salty.pre-prov.nifi
    role.pre.startup.orchestration:                         <--- this
instance (role) has a role scoped pre startup orchestration state hook
        - orch.salty.pre.nifi                               <---
represents the salt state to run salt://orch/salty/pre/nifi.sls
    saltpath:
        /usr/lib/python2.7/site-packages/salt
    saltversion:
        2017.7.5
    saltversioninfo:
        - 2017
        - 7
        - 5
        - 0
    salty.nifi.cluster.id:                                  <---
unique product group role scoped cluster ID, also scoped per environment
        1
    selinux:
        ----------
        enabled:
            True
        enforced:
            Enforcing
    serialnumber:
        ec2db74d-4aee-8724-e6e1-1bdda61b5f74
    server_id:
        1716389426
    shell:
        /bin/sh
    systemd:
        ----------
        features:
            +PAM +AUDIT +SELINUX +IMA -APPARMOR +SMACK +SYSVINIT +UTMP
+LIBCRYPTSETUP +GCRYPT +GNUTLS +ACL +XZ +LZ4 -SECCOMP +BLKID +ELFUTILS
+KMOD +IDN
```

```
    version:
        219
uid:
    0
username:
    root
uuid:
    ec2db74d-4aee-8724-e6e1-1bdda61b5f74
virtual:
    xen
virtual_subtype:
```

```
        Xen PV DomU
    zmqversion:
        4.1.4
```