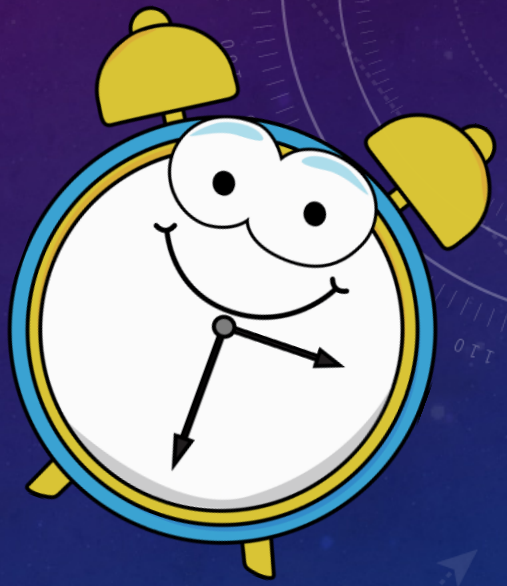
CONFIGURATION MANAGEMENT & ORCHESTRATION WITH

Anirban Saha

CloudOpen and LinuxCon Europe 2014, Dusseldorf

How we're gonna spend the next 2 hours



- Salt overview and features
- Get familiar with Salt components
- Learn how to configure Salt and write states, pillars, etc.
- Demonstrate a basic Salt environment
- Learn about Salt Cloud (using AWS EC2)
- Learn to use Salt orchestrate
- Create an application stack (load balancers, application servers, database servers) using all of the above

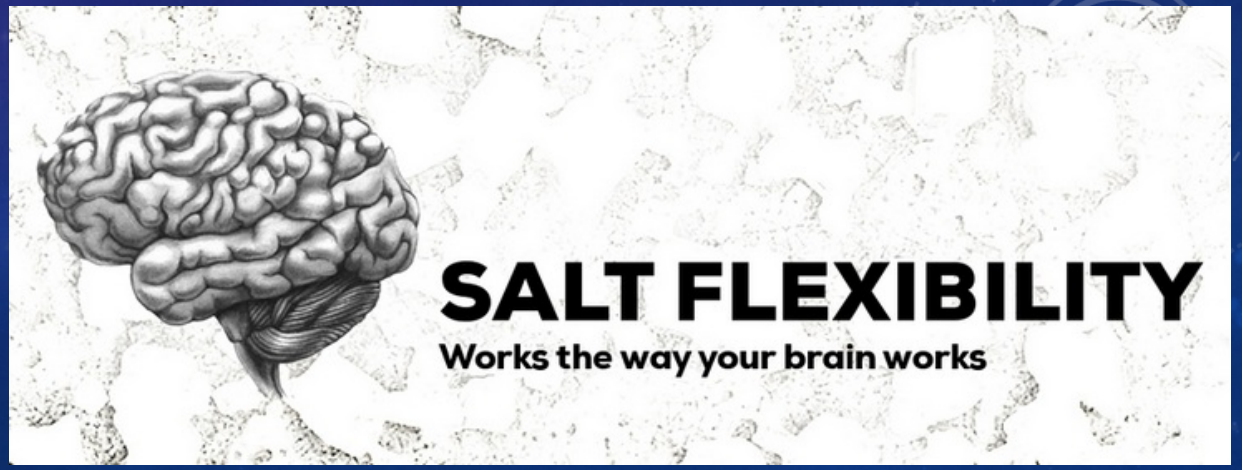


Photo Source : saltstack.org

SALT FEATURES

- Very simple and easy to set up and maintain
- Fast and parallel remote execution
- LOADS and LOADS of built-in modules
- Flexible and scalable

BASIC OVERVIEW

salt server node
(called salt-master)

ZeroMQ message broker

public key + AES encryption

msgpack - fast + light network traffic

salt client node
(called salt-minion)

package - salt-master
daemon - salt-master

TCP ports 4505 and 4506
need to be opened

package - salt-minion
daemon - salt-minion

No ports need to be
opened

SALT TERMINOLOGIES EXPLAINED

Salt Master - The host running the salt-master daemon and serving the configuration for the client nodes (minions)

Salt Minion – The host running the salt-minion daemon and sync'ing with the salt-master

States – Salt State system or SLS is the representation of a state that a system should be in and is represented in simple YAML format

Pillars – It is an interface for Salt to offer global values to be distributed to minions which can be referenced from state files (e.g. usernames, passwords, keys)

Environments – It is the method to organize state tree directories (e.g. production, staging, development, etc.)

Node Groups – This is a method to group client nodes as per their functions, locations, etc

Grains – It is an interface to fetch information about the underlying system of a minion

SALT BINARIES EXPLAINED

On the master :

salt – The most used binary to control and execute states on remote systems
salt-cp – Copies a local file to the remote minions
salt-key – Used to manage public keys for authenticating minions
salt-master – It is the daemon running on the salt master
salt-run – Used to execute convenience applications called runners but only on the master
salt-ssh – Allows salt routines to be executed using ssh only
salt-cloud – Used to provision cloud instances in public and private clouds

On the minion :

salt-minion – It is the daemon running on clients to receive commands from the master
salt-call – Used on the client nodes to fetch configurations from the master

SALT FILES EXPLAINED

Files on the master :

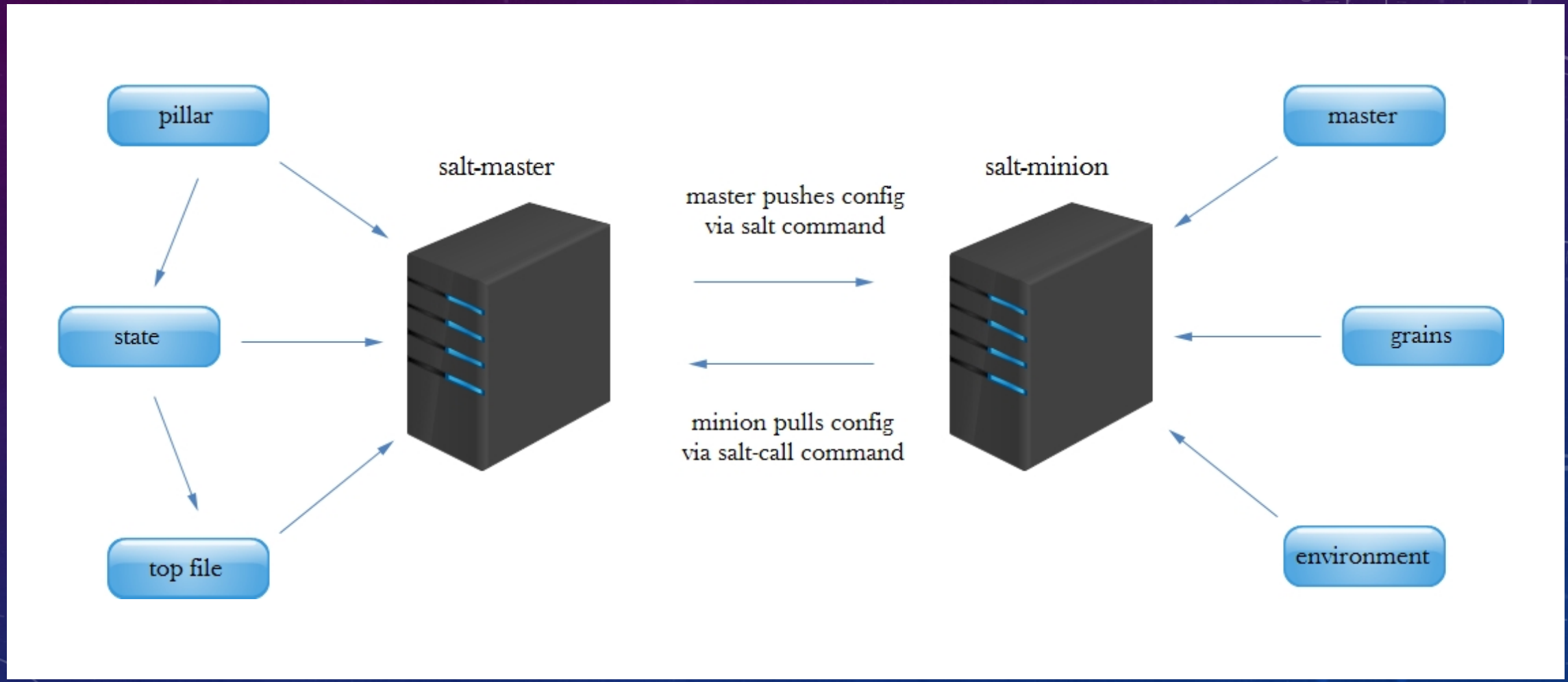
/etc/salt – Directory containing the main configuration files
/etc/salt/master – The salt master main configuration file defining all required resources for the master
/etc/salt/pki/master – Directory containing the master keys for authentication with minions and also the accepted or rejected minion keys
/var/log/salt/master – The main log file for the master daemon
/var/cache/salt/master – The cache directory for master cache data

SALT FILES EXPLAINED (CONTD.)

Files on the minion :

/etc/salt – Directory containing the main configuration files
/etc/salt/minion – The salt minion main configuration file defining resources for the minion and also the salt master to contact
/etc/salt/grains – The file containing the grains for the minion in YAML format if not defined in the main configuration file
/etc/salt/pki/minion – Directory containing the minion keys for authentication with the master
/var/log/salt/minion – The main log file for the minion daemon
/var/cache/salt/minion – The cache directory for the minion cache data

PUTTING ALL COMPONENTS TOGETHER



MASTER CONFIGURATION

File - /etc/salt/master

Common options with default values (mostly remains commented and acts as default):
interface : 0.0.0.0 (interface to which to bind to)
publish_port: 4505 (the port on which to listen)
root_dir: / (this gets prepended to other files like log_file, pki_dir, cache_dir if set)
pki_dir: /etc/salt/pki/master (directory to hold master keys and minion keys already authenticated)
log_file: /var/log/salt/master (file containing log entries for master daemon)

Have to be set explicitly :
autosign_file: /etc/salt/autosign.conf (allows listed host's keys to be accepted automatically, takes options like *, *.domain.com, etc)
autoreject_file: /etc/salt/autoreject.conf (allows listed host's keys to be rejected automatically)

MASTER CONFIGURATION (CONTD.)

Important options :

file_roots (defines the directories which act as the configuration base for environments containing state files) e.g.

```
file_roots:
  base:
    - /opt/cloudopen/salt/base
  production:
    - /opt/cloudopen/salt/production
  staging:
    - /opt/cloudopen/salt/staging
  development:
    - /opt/cloudopen/salt/development
```

MASTER CONFIGURATION (CONTD.)

pillar_roots (defines the directories which act as the configuration base for environments containing pillar files) e.g.

```
pillar_roots:
  base:
    - /opt/cloudopen/salt/pillar/base
  production:
    - /opt/cloudopen/salt/pillar/production
  staging:
    - /opt/cloudopen/salt/pillar/staging
```

nodegroups (groups configured based on function types, location, etc. used for remote execution and synchronization) e.g.

```
nodegroups:
  proddb: 'G@server_type:db and G@environment:production'
  stgapp: 'G@server_type:app and G@environment:staging'
```

TIME FOR SOME

HANDS ON ACTION

HOW TO WRITE SALT CONFIGURATION

https://github.com/rosesnthornz/cloudopen-europe-2014.git

We'll start and cover the configuration of the following components and refer to the files in the repository (cloudopen-europe-2014/salt)

- Pillars
- States
- top.sls
- Nodegroups

CONFIGURING PILLARS

File: PILLAR_BASE/<environment>/<resource_name>/init.sls
e.g. /opt/salt/pillar/production/users/init.sls

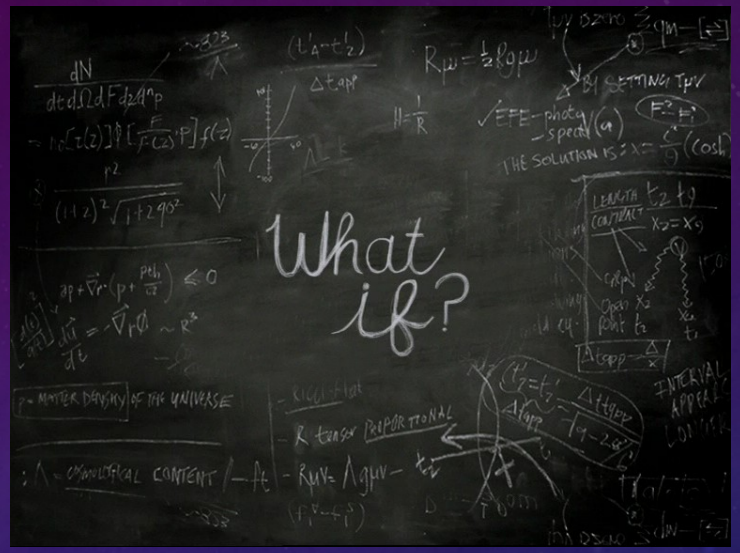
```
users:
  prod_user: prod-app
  prod_user_passwd: '<password_hash>'
```

Accessed in state files as :

```
{{ pillar['users']['prod_user'] }}
```

```
dev_user_list:
  optimus:
    uid: 7001
    passwd: '<password_hash>'
  bumblebee:
    uid: 7002
    passwd: '<password_hash>'
```

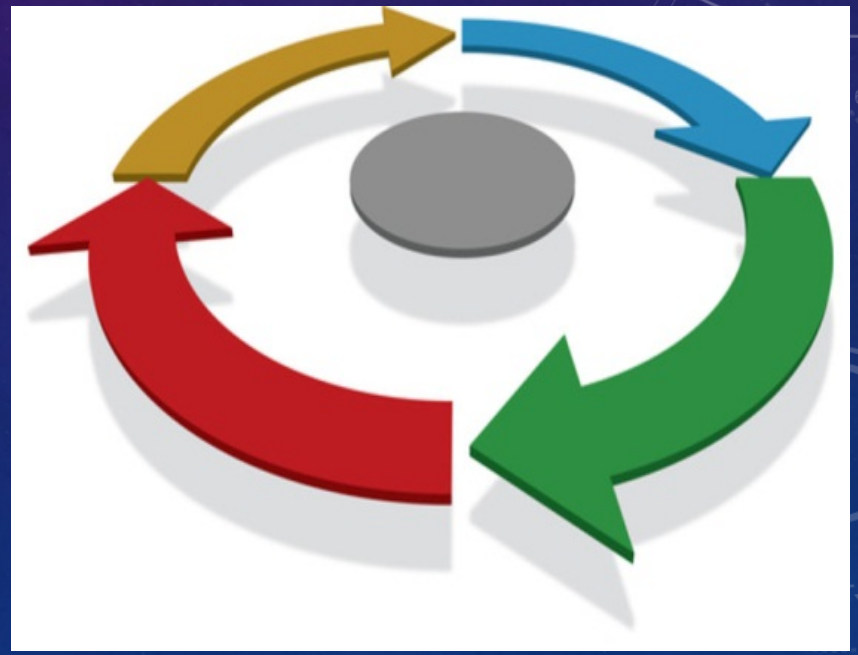
Conditionals???




```
users:
  {% if grains['server_type'] == 'app' %}
    dev_user: dev-app
  {% elif grains['server_type'] == 'web' %}
    dev_user: dev-web
  {% endif %}
```

Iterations???

```
{% for user, details in pillar['dev_user_list'].iteritems() %}
{{ user }}:
  user.present:
    - home: /home/{{ user }}
    - uid: {{ details['uid'] }}
    - password: {{ details['passwd'] }}
    - shell: /bin/bash
    - gid: 5002
{% endfor %}
```



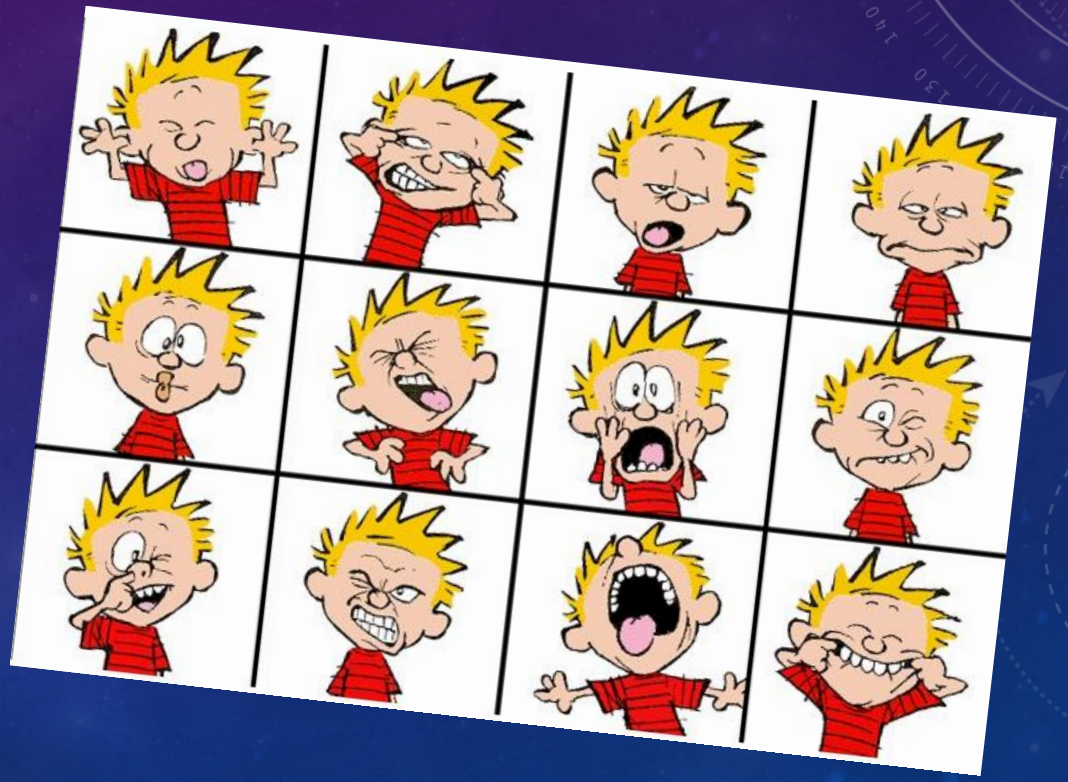
Python functions???

```
users:
  {% if grains['host'].startswith('co') %}
    event: cloudopen
  {% elif grains['host'].startswith('lc') %}
    event: linuxcon
  {% endif %}
```

CONFIGURING STATES

File : SALT_BASE/<environment>/<state_name>/init.sls (the state file containing the configuration)
Files and templates : SALT_BASE/<environment>/<state_name>/files (the directory containing templates and files)
e.g. : /opt/salt/staging/mysql/init.sls

```
ruby-1.9.3:
  rvm.installed:
    - default: True

rails:
  gem.installed:
    - require:
      - rvm: ruby-1.9.3
      - cmd: set_ruby
```



CONFIGURING STATES (CONTD.)

```
/etc/my.cnf:
  file.managed:
    - source: salt://mysql/files/my.cnf
    - user: root
    - group: root
    - mode: 644
    - template: jinja
    - context:
      mysql_ip: {{ salt['network.ip_addrs']('eth0')[0] }}
```

(very important)

CONFIGURING TOP.SLS

File : SALT_BASE/<environment>/top.sls (the main file containing the definition of which state to assign to which hosts)
File : PILLAR_BASE/<environment>/top.sls (also applicable for pillars, as pillars have to be assigned to hosts)
e.g. : /opt/salt/production/top.sls

```
production:                          (the environment to which the top file belongs to)
  '*':                               (the global match wildcard, matching all hosts)
    - groups                         (list of states which have been configured and will be assigned to the matched hosts)
    - users

  'proddb':                          (name of configured nodegroup)
    - match: nodegroup               (very important, without this it'll try to match hostname)
    - mysql

  'server_type: app':                (grain based matching for states)
    - match: grain
    - tomcat

  'appserver,webserver':             (list based matching for states)
    - match: list
    - ntp
```

CONFIGURING NODEGROUPS

File - /etc/salt/master

```
nodegroups:
  us-dbservers: 'G@server_type:db and G@location:us-west-1'
  prodapp: 'G@server_type:app and G@environment:production'
  stgweb: 'G@server_type:web and G@environment:staging'
  puppetservers: 'L@*,*.clolab.com,clopadm01.clolab.com'
```

The first three lines are grain based nodegroup configuration

The last line is list based nodegroup configuration

CONFIGURING MINION

File - /etc/salt/minion

```
master: salt-master
environment: production
```

File - /etc/salt/grains

```
location: us-west-1
server_type: app
environment: production
```

APPLYING STATES & TARGETING MINIONS

From the master to the minion – push mechanism :

```
# salt '*' test.ping                                          (global match wildcard)
# salt –G 'server_type:app and env:prod' state.highstate      (grain based match)
# salt –C 'server_type:web and clo*' state.sls nginx          (compound match)
# salt –L 'appserver,dbserver,webserver' state.sls ntp       (list based match)
# salt –N stgdb cmd.run 'ps –ef | grep mysql'                (nodegroup based match)
```

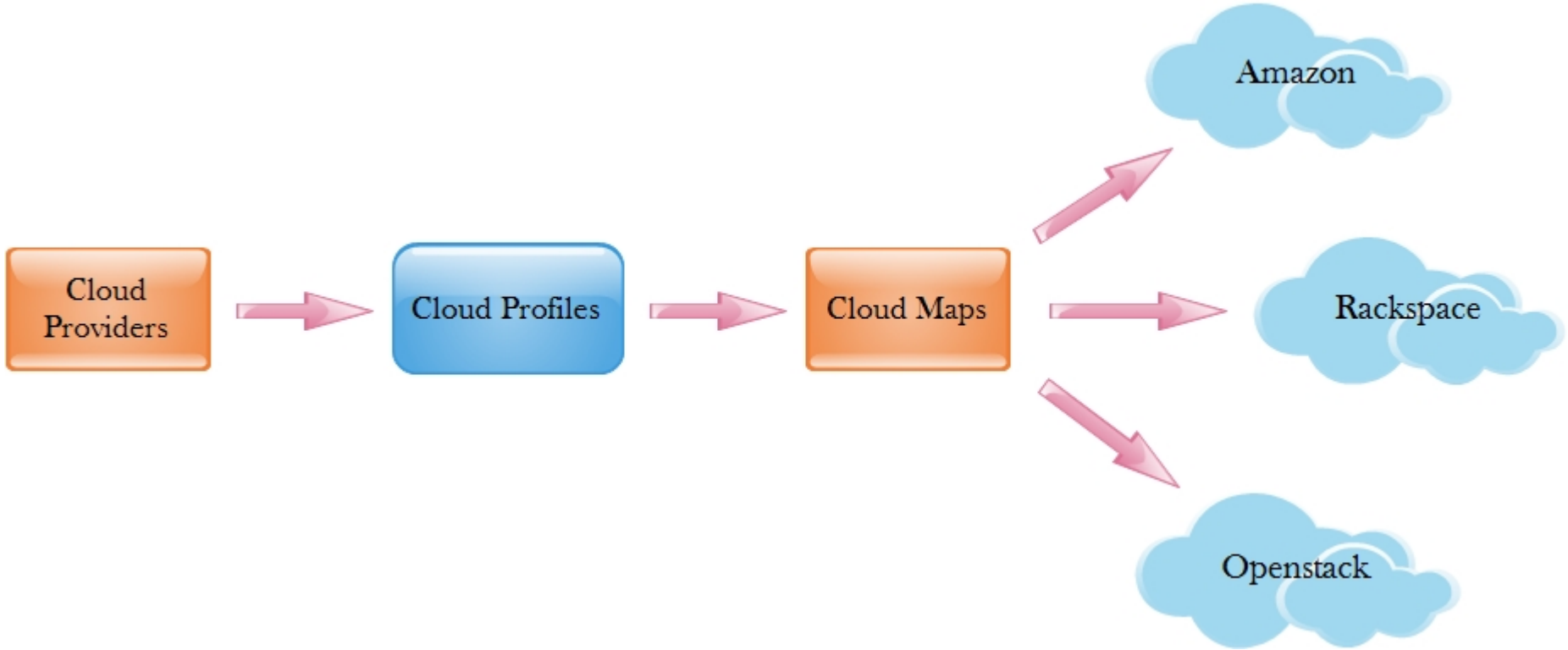
From the minion by calling the master – pull mechanism :

```
# salt-call state.highstate                                  (salt call from the minion)
```

Think you had a lot already. . .?

Haaaahhh. . .I'm not done yet. . .

SALT CLOUD WORKFLOW



SALT CLOUD – CLOUD PROVIDERS

File : /etc/salt/cloud.providers
Directory : /etc/salt/cloud.providers.d/

```
cloudopen_ec2_us_west_2:

  ssh_interface: private_ips

  id: <aws_access_key>
  key: '<aws_secret_key>'

  keyname: common
  private_key: /etc/salt/key/common.pem

  location: us-west-2
  availability_zone: us-west-2b

  size: t2.micro
  del_root_vol_on_destroy: True
  ssh_username: ec2-user
  rename_on_destroy: True
  provider: ec2
```

(provider name, to be used in profiles)

(network interface for master to use while communicating with minion)

(key name created in AWS for instances)
(the private key to use for instance launch)

(username to use for logging in)

SALT CLOUD – CLOUD PROFILES

File : /etc/salt/cloud.profiles
Directory : /etc/salt/cloud.profiles.d/

```
cloudopen_ec2_prod_common:
```
(profile name, used in extending profiles or in cloud maps)

```
  minion:
    master: salt-master.domain.com
    environment: production
```
(minion configuration to be set in the minions)

```
  image: ami-721b7b42
  del_root_vol_on_destroy: True
```

```
  script: bootstrap-salt
  sync_after_install: all
```
(post install script to be run in the minions after launch)
(synchronize grains, modules, etc with the master)

SALT CLOUD – CLOUD PROFILES (CONTD.)

```
cloudopen_ec2_prod_db:                                  (profile name, extends other profiles, used in cloud maps)

  provider: cloudopen_ec2_us_west_2                     (provider name, configured in /etc/salt/cloud.providers)

  network_interfaces:                                   (only configurable in VPCs)
    - DeviceIndex: 0
      PrivateIpAddresses:
        - Primary: True
      AssociatePublicIpAddress: False
      SubnetId: subnet-4236c535
      SecurityGroupId:
        - sg-5b35a03e

  grains:                                               (grains to be set in the minions)
    server_type: db
    environment: production

  tag: {'Environment': 'Production', 'Role': 'Database'}    (AWS tags to be set in the instances)

  extends: cloudopen_ec2_prod_common                    (profile name, extended in other profiles)
```

SALT CLOUD – CLOUD MAPS

File : /etc/salt/cloud.map

```
cloudopen_ec2_prod_db:
   - colcpdb01.domain.com
   - colcpdb02.domain.com

cloudopen_ec2_prod_app:
   - colcpapp01.domain.com
   - colcpapp02.domain.com
   - colcpapp03.domain.com
   - colcpapp04.domain.com
```

(cloud profile name, configured in /etc/salt/cloud.profiles)

Launching the stack :

```
# salt-cloud –m /etc/salt/cloud.map –P
```

[-P for parallel launch]

Querying the stack :

```
# salt-cloud –m /etc/salt/cloud.map –Q
```

Terminating the stack :

```
# salt-cloud –m /etc/salt/cloud.map –d
```

Orchestration?

SALT ORCHESTRATION WITH ORCHESTRATE

File : SALT_BASE/<ENVIRONMENT>/orchestration/app_stack.sls
e.g. /opt/cloudopen/salt-cloud/production/orchestration/app_stack.sls

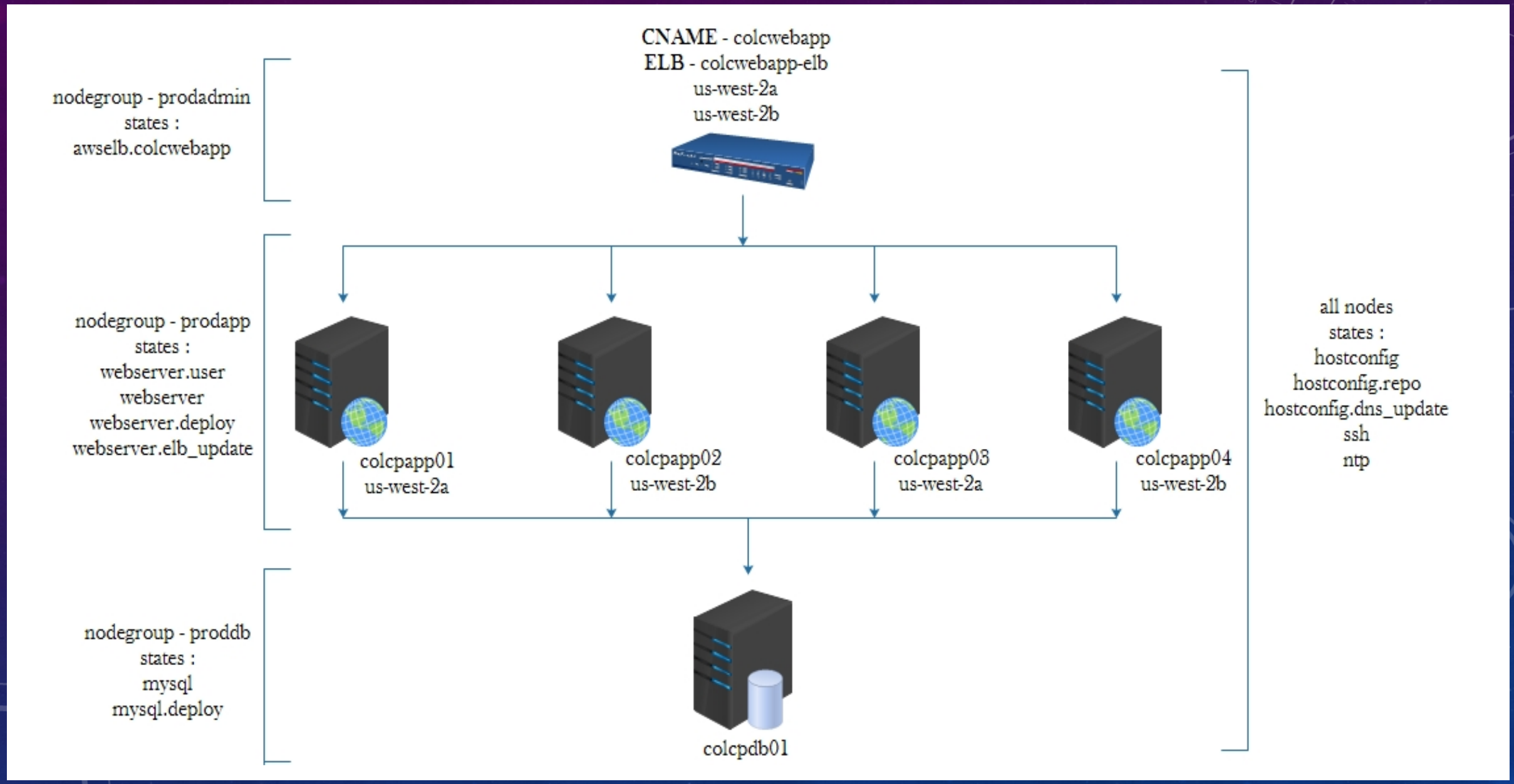
```
webserver_deploy:                          (custom name of definition)
  salt.state:
    - tgt: 'prodapp'                       (targeting minions)
    - tgt_type: nodegroup                  (target type)
    - highstate: True
    - require:                             (dependency on other definitions,
      - salt: dbserver_deploy              e.g. webservers launch only after  dbservers)

dbserver_deploy:
  salt.state:
    - tgt: 'server_type:db'
    - tgt_type: grain
    - highstate: True
```

Running the orchestration command :

```
# salt-run state.orchestrate orchestration.app_stack saltenv=production
```

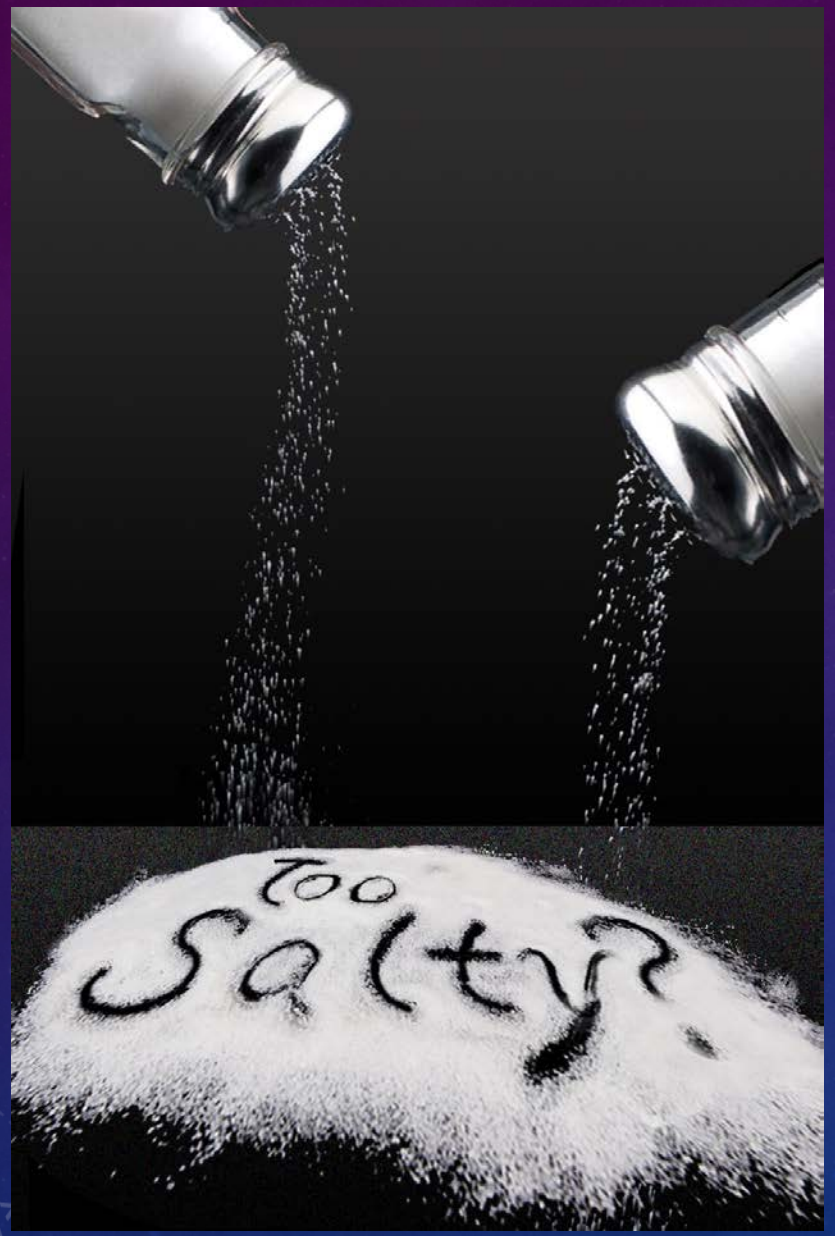
Notice the dependency?

APP STACK DEMONSTRATION



APP STACK DEMONSTRATION OBJECTIVES

- The stack is going to be a farm of web hosts, all of them connecting to a DB server, and all of the web hosts under an ELB with a CNAME pointing to the ELB
- First the ELB is launched, configured and the CNAME is created pointing to the ELB DNS name
- Next the DB server is launched, configured, the DBs created, the security steps executed and the tables populated in the DB
- The web hosts are then launched fetching the website from a git repository and then the web hosts register themselves with the ELB
- All the steps are interdependent, i.e. the DB host won't launch without the ELB and the web hosts won't launch without the DB host



Just a little more. . .

I promise. . .

Some more awesome Salt features…

- **Event System** — Used to fire events between Salt instances using the ZeroMQ interface, carrying tag data for filtering of events and a dict data structure containing information about the event
- **Reactor System** — Uses the Salt event system to associate event tags with reaction files which uses high data to define reactions to be executed
- **Salt Virt** — Cloud controller capability, supports core cloud operations such as virtual machine deployment, VM migration, network profiling, VM inspection, integration with Salt
- **Salt SSH** — Salt capability through the SSH protocol, bypassing the need for the Salt minion to be running on the remote system

CONTACT :

Anirban Saha
Email : sahaanirban1988@gmail.com
Twitter : @rosesnthornz
Skype : anirban.saha.88