Makina States Documentation

Release 1.0

Mathieu Le Marec Pasquet, Régis Leroy & Makina Corpus folks

September 27, 2016

Contents

1 Usage **3**
1.1 Makina-States usage 3

2 Reference **39**
2.1 Reference 39

3 Indices and tables **203**

Python Module Index **205**

Please note that the documentation is far from complete, more over on the usage front.

Please have more a look on the reference chapters.

CHAPTER 1

Usage

1.1 Makina-States usage

1.1.1 About

Makina-States is:

- a consistent collection of SaltStack formulaes and salt modules (execution, states, grains, runners, etc)
- a consistent way to deploy various projects with salt
- a collection of shell scripts

The idea is to have a coherent provision and orchestration lifecycle, leveraging saltstack for the implementation:

- Infrastructure will be orchestrated via makina-states
- Projects will be delivered as docker containers prebacked via salt
- A clear separation of concerns with the sysadmin stuff of the developper stuff has to be done whenever possible to hide uneccessary complexity.

This is why we have two parts called **mastersalt** and **salt** but this is totally optionnal and up to you to use them separatly (default) or in a mixed mode:

- **the infrastructure tasks** called **mastersalt** (/srv/mastersalt)
  - making a machine up & running
  - installing the base configuration upon
  - configuring the firewall
  - etc
- **the projects tasks** called **salt** (/srv/salt)
  - installing consumed services like a reverse proxy, a database server
  - installing the application
  - install project maintainance related stuff like restart crons, or databases backups
  - etc

1.1.2 Setup & usage

Installation & basic usage

Briefing

For now, use Ubuntu >= 14.04.. Makina-States can be ported to any linux based OS, but we here use ubuntu server and this is the only supported system for now. It can be used in any flavor, lxc, docker, baremetal, kvm, etc.

To install our base salt installation, you have to choose between 3 main mode of operations:

The *regular modes* light mode via boot-salt.sh:

- The regular preset modes manages the system configuration from end to end, from the system, to makina-states, including the saltstack/salt installation itself.
- The special `scratch` mode manages only the saltstack + makina-states configuration by default, and it's up to you to apply any other state

The *light mode* light mode via install_makina_states.sh:

- Use makina-states where salt is already installed and where it's install has not to have to be done via makina-states itself.

Reminder

- Makina-states is based on "nodetypes presets" that are prebundled collections of makina-states states to apply to the system.
- On those nodetypes, we may manage "controllers", aka the salt daemons.
- On those nodetypes, we may configure "localsettings" like vim, git, & basepackages or network configurations. If any other preset than **scratch** has been activated, many localsettings will be applied (see mc_states/modules/localsettings.py:registry)
- After all of the previous steps, we may configure services like sshd, crond, or databases. If we are on the **scratch** mode, no services are configured by default.
- Eventually, we may by able to install projects via mc_project. A project is just a classical code repository which has a ".salt" folder commited with enougth information on how to deploy it.

Details

just run the **_scrits/boot-salt.sh** script as **root**,

Please read next paragraphs before running any command.

- In most cases, all our production installs run 2 instances of salt: **mastersalt** and **salt** which can be be in **asterless** or **remote** mode. In this mode, certains states are only reachable from the mastersalt daemons (the low levels which will can break the system and have to be done via sysadmin).
- In some case, you can install only the **salt** side, and both mastersalt & salt configurations will be available for use in this mode.
- As a sole developer, You will nearly never have to handle much with the **mastersalt** part unless you are going to be very low-level.
- All the behavior of the script can be controlled via environment variables or command line arguments switches.

- That's why you will need to tell which daemons you want (minion/master) and on what kind of environment you are installing on (the nodetype).
- You'll also have to set the **minion id**. The default choice for **–minion-id** is the current machine hostname. You should keep this naming scheme unless you have a good reason to change it.
- Default salt install is **masterless (standalone)**.
- Default mastersalt install is **remote (connected)**.
- Your choice for `\-\-nodetype` is certainly one of:
  - **scratch** manages by default only the salt installation and configuration. You ll want to activate this mode if you want to apply explicitly your states without relying of default nodetypes configuration.
  - **server (default)** matches a baremetal server, and manage it from end to end (base packages, network, locales, sshd, crond, logrotate, etc, by default)
  - **vm** matches a VM (not baremetal), this is mostly like **server**.
  - **lxccontainer** matches a VM (not baremetal), this is mostly like **server**.
  - **laptop** is like server but also install packages for working on a developement machine (prebacking a laptop for a dev)
  - **dockercontainer** matches a VM (not baremetal), this is mostly like **server**, but install & preconfigure circus to manage daemons.
  - **devhost** is suitable for a development machine enabling states to act on that, by example installation of a test local-loop mailer.
  - **vagrantvm** is suitable to flag vagrant boxes and is a subtype of devhost
- For configuring all salt daemons, you have some extra parameters (here are the environment variables, but you have also command line switches to set them
  - **–salt-master-dns**; hostname (FQDN) of the linked master
  - **–salt-master-port**: port of the linked master
  - **–mastersalt**: is the mastersalt hostname (FQDN) to link to
  - **–mastersalt-master-port**: overrides the port for the distant mastersalt server which is 4606 usually (read the script)

Regular modes (via boot-salt.sh)

boot-salt.sh will try to remember how you configured makina-states on each run. It stores configs in :

- /etc/salt/makina-states & if available /etc/mastersalt
- /etc/makina-states

Indeedn while running, the script try to find enougth information (nodetype, salt installs, branch), and will automaticly guess & store the parameters by itself.

In other words, you will just have to type **boot-salt.sh**, and verify settings the next time you ll use it.

REMEMBER THAT FOR NOW YOU HAVE TO USE UBUNTU >= 14.04.

Download Get the script:

```
wget http://raw.github.com/makinacorpus/makina-states/master/_scripts/boot-salt.sh
```

Short overview:

```
./boot-salt.sh --help
```

Detailed overview:

```
./boot-salt.sh --long-help
```

CLI Exemples If you want to install only a minion which will be connected to a remote mastersalt master:

```
./boot-salt.sh --mastersalt <MASTERSALT_FQDN> \
    [--mastersaltsalt-master-port "PORT OF MASTER  IF NOT 4506"]
```

If you want to install salt on a bare server, without mastersalt:

```
./boot-salt.sh --no-mastersalt
```

If you want to install salt on a machine flaggued as a devhost (server + dev mode):

```
./boot-salt.sh --n devhost
```

If you want to install and test test mastersalt system locally to your box:

```
./boot-salt.sh --mastersalt-master --mastersalt $(hostname -f)
```

If you want to manage from end to end your server, select also the `laptop` preset nodetype:

```
./boot-salt.sh --mastersalt <MASTERSALT_FQDN> \
    [--mastersaltsalt-master-port "PORT OF MASTER  IF NOT 4506"] -n laptop
```

To skip the automatic code update/upgrade:

```
./boot-salt.sh -S
```

To switch on a makina-states branch, like the **stable** branch in production:

```
./boot-salt.sh -b stable
```

Upgrade Upgrade will:

- Run predefined & scheduled upgrade code
- Uupdate makina-states repositories in /srv/salt & /srv/makina-states
- Update core repositories (like salt code source in /srv/makina-states/src/salt)
- Redo the daemon configuration if necessary
- Redo the daemon association if necessary
- Do the highstates (salt and masterone if any)

```
boot-salt.sh -C --upgrade
```

Light mode (via install_makina_states.sh) This is mainly needed to integrate Makina-States within a pre-existing salt infrastructure (via install_makina_states.sh).

Basically makina states contains:

- a python egg
- a lot of custom salt modules of different types (execution, grains, states, cloud, etc.)
- a collection of formulaes

To enable it into your salt infrastructure:

- You have to put it in your salt_root to activate the formulaes:
- You have to install python dependencies (see the script) and the mc_states python package (included in makina-states)
- You have to link all custom salt modules to your salt root and synchronnise your minions caches.

We provide a convenient helper for this purpose called **_scripts/install_makina_states.sh**:

```
wget \
 http://raw.github.com/makinacorpus/makina-states/master/_scripts/install_makina_states.sh
export SALT_ROOT="/srv/salt" # whereever it is
./install_makina_states.sh
```

The script can safely be recalled after each makina-states "git pull" to relink the updated modules.

Activating another nodetype preset after installation If you installed the **scratch** preset and want to switch to another preset:

```
[master]salt-call [--local] state.sls makina-states.nodetypes.<your_new_preset>
```

If you installed a preset and want to switch to another preset:

- edit **/etc/makina-states/nodetype** and put your new preset
- edit **/etc/*/makina-states/nodetypes.yaml** and set to false your old preset
- Finally, run:

```
    [master]salt-call [--local] state.sls makina-states.nodetypes.<your_new_preset>
```

Configuration

Makina states is a consistent collection of specialized states (or formulaes) that you can apply to your system to install and configure it.

Our saltstack formulaes (the **sls** files) leverage the use of saltstack execution modules (called makina-states registries) to expose aggregated variables from different configuration sources where the main ones are the grains and the pillarand the local makina-states registries (msgpack or yaml files inside /etc/*salt).

To install something, you can either:

- set a key/value in pillar or grains (eg: **makina-states.services.http.nginx: true**) and play the highstate
- Call directly a specific state: **salt-call state.sls makina-states.services.http.nginx**. Remember that playing a state may register it and will be called again in further highstates.
- Include directly a specific state in a **include:** sls statement from a custom state of yours.

As soon as those states are run, they will set a flag on the machine having the side effect to register them to be `auto replayed` on next highstates, implicitly.

In other word, in the future highstates, they will even run even if we have not included them explicitly.

To sum up, we have made all of those states reacting and setting tags for the system to be totally dynamic. All the things you need to do is to set in pillar or in grains the appropriate values for your machine to be configured.

4 levels are available to make install a `makina state` formula.

- Direct inclusion via the 'include:' statement:

```
foo.sls:
    include:
      - include makina-states.services.http.nginx
```

- Install directly the state via a salt/salt-call state.sls:

```
salt-call state.sls makina-states.services.http.nginx
```

- Relevant grain configuration slug:

```
salt-call --local grains.setval makina-states.services.http.nginx true
salt-call state.highstate
```

- Relevant pillar configuration slug for the highstate to pick up the newly registred state:

```
/srv/pillar/foo.sls
makina-states.services.http.nginx: true in an included  pillar file
```

You can indeed imagine there is a lot of variables that can be modified to apply the configuration to a minion. To find what to do, we invite you to just read the states and the documentation that seem to be relevant to your needs. And to know what flag to modify, find the "python registry" and check the param to modify, it's always the same dance.

Best practise

Be careful with grains

- Grains can be particulary insecures because they are freely set on the client (minion) side,
- Grains are are slow to update.
- You have to pay attention that states and pillar accesses chained by this inheritance are only limited to the scope you want and do not expose too many sensitive information because a minion setted a particular grain.

Writing rules Some rules to write makina-states states:

- When you create a new state, include it in its respective registry.
- Avoid to write an absolute path, use localsettings.locations.PATH_PREFIX or another registry variable.
- Try to isolate the settings and make a subregistry to regroup them.
- Never ever use short form of states (states without names, use states unique IDs)

  DO:

```
foo-foo:
    cmd.run
        name: /foo/foo
```

  DONT:

```
    /foo/foo
        cmd.run: []
```

- Please use as much as possible require(_in)/watch(_in) to ensure your configuration slugs will be correctly ordered during execution.
- If your states are getting being or need scheduling, please add separate hooks (mc.proxy.hook states) files (see developer documentation).

Use a lxc based makina-states environment

In makina-states, our LXC images are known to use a network bridge called `lxcbr1`.

They use the `10.5/16` network and `10.5.0.1` as the default gateway.

For this to work, you have plenty of solutions.

Install LXC makina-states on ubuntu with upstart

Install lxc Official doc: https://help.ubuntu.com/lts/serverguide/lxc.html

First install LXC

```
sudo apt-get install lxc bridge-utils
```

Prepare network connectivity In makina-states, our images are known to use a network bridge called `lxcbr1`.

They use the `10.5/16` network and `10.5.0.1` as the default gateway.

For this to work, you have plenty of solutions.

Network bridge The first thing you ll have to do is to persist the network bridge. For this, on ubuntu, the simpliest thing is to inspire ourselves from the default lxc-net configuration and create the following configuration file

First, create **as root** this upstart job `/etc/init/lxc-net-makina.conf` & helpers:

```
for i in /etc/init/lxc-net-makina\
         /usr/bin/magicbridge.sh /etc/reset-net-bridges;do
 curl --silent \
  "https://raw.githubusercontent.com/makinacorpus/makina-states/stable/files${i}" \
  > "${i}"
done
chmod 644 /etc/init/lxc-net-makina
chmod 755 /usr/bin/magicbridge.sh /etc/reset-net-bridges
cp /usr/bin/magicbridge.sh /usr/bin/lxc-net-makina.sh
```

Don't forget that you can read the upstart job but basically, it creates the bridge and then masquerade the outband traffic.

Then reload it with:

```
service lxc-net-makina restart
```

You will see you newly created bridge with:

```
# ip addr show dev lxcbr1
5: lxcbr1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether fe:16:a7:12:b3:3e brd ff:ff:ff:ff:ff:ff
    inet 10.5.0.1/16 brd 10.5.255.255 scope global lxcbr1
       valid_lft forever preferred_lft forever
    inet6 fe80::c9f:baff:fe43:a2ef/64 scope link
       valid_lft forever preferred_lft forever
# ifconfig lxcbr1
lxcbr1    Link encap:Ethernet  HWaddr fe:16:a7:12:b3:3e
          inet adr:10.5.0.1  Bcast:10.5.255.255  Masque:255.255.0.0
          adr inet6: fe80::c9f:baff:fe43:a2ef/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Packets reçus:2567376 erreurs:0 :0 overruns:0 frame:0
          TX packets:5204695 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:0
          Octets reçus:135360650 (135.3 MB) Octets transmis:1160735414 (1.1 GB)
```

Activate kernel forwarding Please follow *Network route forwarding*.

Install the image on ubuntu You can now read and proceed with the following section, *Install the base LXC container*.

Note about firewalling on ubuntu Last but not least, if you use a firewall, and we hope you do so, please refer to the firewalling section for further configuration. Please read *Network firewalling and masquerating a makina-states LXC based image*.

On ubuntu you may be using:

- *ufw*
- *shorewall*

Install a new container

- Refer to *Create a container*

Conclusion (ubuntu) Well done, you may now enjoy your new container You may want to continue with:

- *Projects for developers*

Install LXC makina-states on a systemd based host (debian/archlinux/fedora)

This applies to:

- debian >= jessie
- archlinux
- fedora

Install lxc on ArchLinux install LXC & tools

```
sudo pacman -S deboostrap lxc bridge-utils netctl yum
```

debootstrap and yum are from AUR.

Install lxc on debian install LXC

```
sudo apt-get install deboostrap lxc bridge-utils
```

Install lxc on Fedora install LXC

```
sudo yum install deboostrap lxc bridge-utils lxc-extra
service lxc restart
```

Prepare network connectivity In makina-states, our images are known to use a network bridge called `lxcbr1`.

They use the `10.5/16` network and `10.5.0.1` as the default gateway.

For this to work, you have plenty of solutions.

Network bridge The first thing you ll have to do is to persist the network bridge. For this, on fedora, the simpliest thing is to inspire ourselves from the default ubuntu lxc-net configuration and create the following configuration file

First, create **as root** this systemd Unit `/etc/systemd/system/lxc-net-makina` & helpers:

```
for i in /etc/systemd/system/lxc-net-makina.service \
         /usr/bin/magicbridge.sh /etc/reset-net-bridges;do
    curl --silent \
     "https://raw.githubusercontent.com/makinacorpus/makina-states/stable/files${i}" \
     > "${i}"
done
chmod 644 /etc/systemd/system/lxc-net-makina.service
chmod 755 /usr/bin/magicbridge.sh /etc/reset-net-bridges
cp /usr/bin/magicbridge.sh /usr/bin/lxc-net-makina.sh
```

Don't forget that you can read the systemd job but basically, it creates the bridge and then masquerade the outband traffic.

Then reload it with:

```
systemctl enable lxc-net-makina
service lxc-net-makina restart
```

You will see you newly created bridge with:

```
# ip addr show dev lxcbr1
5: lxcbr1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether fe:16:a7:12:b3:3e brd ff:ff:ff:ff:ff:ff
    inet 10.5.0.1/16 brd 10.5.255.255 scope global lxcbr1
       valid_lft forever preferred_lft forever
    inet6 fe80::c9f:baff:fe43:a2ef/64 scope link
       valid_lft forever preferred_lft forever
# ifconfig lxcbr1
lxcbr1    Link encap:Ethernet  HWaddr fe:16:a7:12:b3:3e
          inet adr:10.5.0.1  Bcast:10.5.255.255  Masque:255.255.0.0
          adr inet6: fe80::c9f:baff:fe43:a2ef/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Packets reçus:2567376 erreurs:0 :0 overruns:0 frame:0
          TX packets:5204695 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:0
          Octets reçus:135360650 (135.3 MB) Octets transmis:1160735414 (1.1 GB)
```

Activate kernel forwarding Please follow *Network route forwarding*.

Install the image on a systemD based host You can now read and proceed with the following section, *Install the base LXC container*.

Note about firewalling on a systemD based host Last but not least, if you use a firewall, and we hope you do so, please refer to the firewalling section for further configuration. Please read *Network firewalling and masquerating a makina-states LXC based image* .

On fedora you may be using:

- *firewalld*
- *shorewall*

Install a new container

- Refer to *Create a container*

Conclusion (systemD based host) Well done, you may now enjoy your new container

You may want to continue with:

- *Projects for developers*

Use makina-states in docker

Install a makina-states docker environement

Intro and history Cluster based on LXC/kvm and mastersalt-pillar was the first thing we had, This allowed us to have a git/push/deploy to environment workflow. It's not what you ll have to use to spawn a kubernetes based cluster, as we want things to be a lot more immutable.

We still reuse bits from the past, but we ll input the settings differently as it was a bit too hard from end users to use that.

Idea is to deploy pre-backed containers onto production and do not do heavy configuration at runtime. In other words, we just edit some configuration file to wire the container to server the app request, but we do not reconfigure it from end to end.

Basic development installation

Install docker

- If you system is not supported, you can try to run it, but it just untested. you need at least docker, with aufs support.
- If you do not run ubuntu, run it intro your virtualisation software (Virtualbox, parallell, etc)
- For ubuntu, you best bet is to use **something >= Ubuntu 14.04** with a **recent kernel extras image** (**>=3.19** from enablement stack). Verify with

```
uname -ar
```

- At this time of writing, you can upgrade your kernel by issuing the following command

```
apt-get install linux-image-extra-3.19.0-33-generic # vivid / trusty
```

- Install lxc-utils & docker by reading your distribution guidelines for that purpose
  - Eg on ubuntu:

```
apt-get install lxc docker rsync
```

- Replace docker by makina-corpus version (1.9+), We have modified it to allow to use a custom apparmor profile instead of inject it's own and broken one.
  - The sources are here@github.
  - We provide a prebuilt binary for linux.:

```
cp /usr/bin/docker /usr/bin/docker.dist
curl -L --insecure -s \
    https://github.com/makinacorpus/docker/releases/download/mc_1/docker \
    -o /usr/bin/docker
# or wget \
#  https://github.com/makinacorpus/docker/releases/download/mc_1/docker \
#  -O /usr/bin/docker
chmod +x /usr/bin/docker
```

- If you are on Ubuntu or any system protected by **apparmor**, you ll have to tweak your apparmor installation. If you are not configuring your system via makina-states, you can however bring back the profile quite easily

Configure apparmor

```
mkdir -pv /etc/apparmor.d/abstractions/lxc
curl -L --insecure -s https://raw.githubusercontent.com/makinacorpus/makina-states/master/files/etc/a
curl -L --insecure -s https://raw.githubusercontent.com/makinacorpus/makina-states/master/files/etc/a
cd /tmp
curl -L --insecure -s https://raw.githubusercontent.com/makinacorpus/makina-states/master/files/etc/a
curl -L --insecure -s https://raw.githubusercontent.com/makinacorpus/makina-states/master/files/etc/a

cd /
patch -Np2 < /tmp/usr.sbin.ntpd.patch
patch -Np2 < /tmp/usr.sbin.ntpd.perms.patch
service apparmor restart
```

Install the base image Clone makina-states, even if not installing it on you host

```
mkdir /srv/mastersalt && cd /srv/mastersalt
git clone http://github.com/makinacorpus/makina-states.git
```

Create the base makinacorpus/makina-states image

```
cd /srv/mastersalt/makina-states
./docker/build-scratch.sh
# at the end of the script, this will output the base image tag
```

Makina-States based docker Images

Contents

- *Makina-States based docker Images*
  - *Rules*
    - *Run time*
    - *Build time*
  - *layout inside the Image*
  - *Initialise your dev environment*
    - *Download and initialize the layout*
    - *OPTIONNAL: Generate a a certificate with a custom authority for testing purposes*
    - *Register the certificate to the host openssl configuration*
    - *Configure the image via the salt PILLAR*
  - *Build & Run*
    - *DNS configuration*
  - *List of example images*

Rules

- Docker images share a common layout, and inherit from the `makina-states` base images from docker-hub.
- Applications are deployed into those containers via *mc_project*.
- Those images use **mc_project** in `remote_less` mode and should not rely on a full system running, we are in a docker. For long living processes, use circus.
- Runtime include an `initial pre-re-configure step` before launching the app and the entry point lives into `$project_root/bin/launch.sh`
  - Ideally, there is a `mc_launcher.py` saltstack module to orchestrate the whole reconfigure step.
- Images include at least 2 mountpoints for the `logs` and the `data` folders.

Run time The app is launched an managed via a `bin/launch.sh` (Example) script, which should ideally:

- Replace the default pillar by the **configuration/pillar.sls** if it existsa. This is the only thing we need to do before launching a salt module script that does the rest.
- Execute a salt **mc_launcher.py** (Example) module which runs our app after maybe having reconfigured it.
  - allow inbound ssh connections for allowed keys
  - reconfigure (ideally by exec'ing a subset of the sls in **.salt**) the container to serve the app (eg: update domain to server, ip of the database, registration to autodiscovery service)
  - spawn a circus daemon at the end of the configuration.
  - The module should have at least implements this interface:

```
def sshconfig(name=PROJECT):
    '''code to allow ssh_keys to connect'''
    pass
def reconfigure(name=PROJECT):
    '''code to reconfigure the app to serve requests
      in this specific context'''
    pass
def launch(name=PROJECT, ssh_config=False, re_configure=False):
    if ssh_config:
        ssh_config(name=name)
```

```
            if re_configure
                re_configure(name=name)
            # code to launch the app in foreground
```

- Indeed, the app is lightly reconfigured via salt and may be given an overriden pillar file via a filesystem volume to help to reconfigure it. **Think to rename the pillar configuration key along with the name of your project** See mc_project configuration pillar file
- Volumes and files that need to be prepolulated should be filled by the launcher if and only if it is not already data placed into them.
- A Control-C or quit signal must inhibit any launched process more or less gracefully

Build time

- We configure the image through a regular *mc_project* based saltstack project.
- All the processes inside the container must be managed if possible via circus
- POSIX Acls are now to be avoided at all cost to avoid export/import problems as tar is used to exchange images, the extended attributes are lost in the middle

layout inside the Image This is of course an example but it reflects what we need to respect:

```
/srv/salt/custom.sls      <- custom pillar
/srv/projects/<project>
   |
   |- project/ <- application code
   |      |- Dockerfile    <- Each app needs to have a basic Dockerfile
   |      |- bin/launch.sh <- launcher that:
   |      |                   - copy $data/configuration/pillar.sls -> $pillar/init.sls
   |      |                   - reconfigure (via salt) the app
   |      |                   - launch the app in foreground
   |      |- .salt         <- deployment and reconfigure code (mc_project based)
   |      |- .salt/100_dirs_and_prerequisites.sls
   |      |- .salt/200_reconfigure.sls
   |      |- .salt/300_nginx.sls
   |      |- .salt/400_circus.sls
   |      |- .salt/_modules/mc_launcher.py
   |                 code that is used to reconfigure the image
   |                 at launch time (via launch.sh)
   |
   |- pillar/  <- salt extra pillar that overrides PILLAR.sample (itself
   |              overriden by data/configuration/pillar.sls)
   |
   |- data/                   <- exposed through a docker volume
         |- data/             <- persistent data root
         |- configuration/    <- deploy time pillar that is used at reconfigure
                                  time (startup of a pre-built image)
```

Initialise your dev environment We separate the project codebase from any persistent data that is needed to be created along any container. Those folders will be mounted inside the running container as docker volumes.

- one dedicated for the clone of the codebase: **${PROJECT}**
- one dedicated for the persistent data & configuration: **${DATA}**
- a subdirectory of data is exposed as a docker volume: **${VOLUME}**

If you run a prebuilt image, you may not need the project codebase folder.

By convention, the name of the persistant data holding directory is the name of the clone folder suffixed by `_data`. Eg if you clone your project inside `~/project`, the data folder will be `~/project_data`. The data folder can't and must not be inside the project folder as we drastically play with unix permissions to enforce proper security and the two of those folders do not have at all the same policies. The special folder **project_data/volume** is mounted as a docker voume inside the container at the project data directory location. We refer it as **${VOLUME}**.

You need to add a volume that will contains those subdirs:

${PROJECT}/ git clone of this repository, the project code inside the container. this folder contains a '.salt' folder which describe how to install & configure this project. (/srv/projects/<name>/project)

${PROJECT}/Dockerfile Dockerfile to build your app

${PROJECT}/.salt mc_project configuration to configure your app

${DATA}/volume/ aka ${VOLUME} mounted as the persistent data folder inside the container (/srv/projects/<name>/data)

${DATA}/volume/configuration directory holding configuration bits for the running container that need to be edited or accessible from the host & the user

${DATA}/volume/data persistent data

Inside of the data volume, we also differentiate in term of permissions the configuration from the datas (later is more laxist). For the configuration directories, after the image has been launched, you ll certainly need to gain root privileges to re-edit any files in those subdirs.

Project_data in details:

${VOLUME}/ssh/*.pub ssh public keys to allow to connect as root

${VOLUME}/configuration contains the configuration

${VOLUME}/configuration/pillar.sls configuration file (saltstack pillar) for the container

${VOLUME}/data/ top data dir

Download and initialize the layout

```
export REPO_URL="http://git/orga/repo.git"
export PROJECT="${WORKSPACE}/myproject" # where you want to put the code
export DATA="${PROJECT}_data"           # where you want to put the data
export VOLUME="${DATA}/volume"          # where you want to put the docker volume
mkdir -p "${DATA}" "${VOLUME}"
git clone "${REPO_URL}" "${PROJECT}"
```

OPTIONNAL: Generate a a certificate with a custom authority for testing purposes This script will generate a CA and sign a wildcard certificate for CN="${DOMAIN}" with it .. code-block:: bash

gen_password() { < /dev/urandom tr -dc _A-Z-a-z-0-9 | head -c${1:-64};echo; } DATA="${DATA:-$(pwd)}" CA_PATH="${CA_PATH:-${DATA}/ca}" C="${C:-FR}" L="${L:-Paris}" ST="${ST:-IleDeFrance}" CA="${CA:-"dockerca"}" EXPIRY="${EXPIRY:-$((365*100))}" DOMAIN="${DOMAIN:-"registryh.docker.tld"}" mkdir -p "${CA_PATH}" cd "${CA_PATH}" CA_PASSWD="$(cat ca_passwd 2>/dev/null)" DOMAIN_PASSWD="$(cat "${DOMAIN}_passwd" 2>/dev/null)" CA_PASSWD="${CA_PASSWD:-$(gen_password)}" DOMAIN_PASSWD="${DOMAIN_PASSWD:-$(gen_password)}" echo "$CA_PASSWD" > ca_passwd echo "$DOMAIN_PASSWD" > "${DOMAIN}_passwd" if ! test -e ca_key.pem;then

openssl genrsa -des3 -passout file:ca_passwd -out sca_key.pem openssl rsa -in sca_key.pem -passin file:ca_passwd -out ca_key.pem

fi if ! test -e ca.crt;then

**openssl req -new -x509 -days ${EXPIRY} -key ca_key.pem -out ca.crt** -subj "/C=${C}/ST=${ST}/L=${L}/O=${CA}/CN=${CA}/"

fi if ! test -e "${DOMAIN}_key.pem";then

openssl genrsa -des3 -passout "file:${DOMAIN}_passwd" -out "s${DOMAIN}_key.pem" openssl rsa -in "s${DOMAIN}_key.pem" -passin "file:${DOMAIN}_passwd" -out "${DOMAIN}_key.pem"

fi if ! test -e "${DOMAIN}.crt";then

openssl req -new -key "${DOMAIN}_key.pem" -out "${DOMAIN}.csr" -subj "/C=${C}/ST=${ST}/L=${L}/O=${CA}/CN=*.${DOMAIN}/"

**openssl x509 -CAcreateserial -req -days ${EXPIRY} -in ${DOMAIN}.csr** -CA ca.crt -CAkey ca_key.pem -out "${DOMAIN}.crt"

fi cat "${DOMAIN}.crt" "ca.crt" > "${DOMAIN}.bundle.crt" cat "${DOMAIN}.crt" "ca.crt" "${DOMAIN}_key.pem" > "${DOMAIN}.full.crt" chmod 644 *crt* chmod 640 *key* *full.crt *_passwd

Register the certificate to the host openssl configuration

```
cat | sudo sh << EOF
cp "${DATA}/ca/${domain}.bundle.crt /usr/local/share/ca-certificates\
&& update-ca-certificates
EOF
```

Configure the image via the salt PILLAR

You need then to fill the pillar to reconfigure your container at running time.

- setup a domain to serve for the registry (the virtualhost name)
- (opt) the SSL certificate informations

```
mkdir -p "${VOLUME}/configuration"
cp .salt/PILLAR.sample "${VOLUME}/configuration/pillar.sls"
sed -re "s/makina-projects.projectname/makina-projects.registry/g"\
  -i "${VOLUME}/configuration/pillar.sls"
$EDITOR "${VOLUME}/configuration/pillar.sls" # Adapt to your needs
```

Build & Run **Be sure to have completed the initial configuration (SSL, PILLAR) before launching the container.** You may not need to **build** the image, you can directly download it from the docker-hub.

```
docker pull <orga>/<image>
# or docker build -t <orga>/<image> .
```

Run

```
docker run -ti -v "${VOLUME}":/srv/projects/<project>/data <orga>/<image>
```

DNS configuration When your container is running and you want to access it locally, in development mode,
 just inspect and register it in your /etc/hosts file can avoid you tedious setup

Assuming that you configured the container to respond to **${DOMAIN}**.

```
IP=$(sudo docker inspect -f '{{ .NetworkSettings.IPAddress }}' <YOUR_CONTAINER_ID>)
cat | sudo sh << EOF
sed -i -re "/${DOMAIN}/d" /etc/hosts
echo $IP ${DOMAIN}>>/etc/hosts
EOF
```

List of example images

- docker registry

Mastersalt documentation

Installation of a cluster based on mastersalt

Briefing Most of mastersalt part is configured via a file: `/srv/mastersalt/database.sls`. This is a simple YAML (+jinja) file to describe your infra in a very consive format.

This file is then read by the `mc_pillar` execution module which is called from the `mc_pillar` ext_pillar module and assemble pieces of information through PILLAR entries

IOW, The ext pillar will setup the pillar for mastersalt to help to manage a whole infractructure:

This covers those parts:

- CA/SSL certificates generation (on master side)
- CA/SSL certificates delivery (on minions)
- Supervision (icinga)
- Authorising SSH access to boxes and configure SSH servers
- Manage auto upgrades via unattended
- Managing PAM & NSSconfiguration
- backups (client & master) (based burp)
- DNS (bind)
- LDAP (openldap)
- Baremetal and VM network configuration (this include setuping ip faiover aliases on baremetal servers)
- Firewall configuration (ms_iptables (simple iptables frontend configured via json))
- repositories managment (APT)
- locales managment
- Cloud Controller orchestration
  - Manage dns entries
  - Reverse proxies (haproxy & firewalld (http(s)/ssh/snmp)
  - Spawning VMS (kvm, lxc) and managing their lifecycle
- Kernel sysctl managment
- Configure base machine configurations (editors, base packages & so on)
- etc.

The yaml files doesnt exist at first, you have to create it. You can get a sample from https://github.com/makinacorpus/makina-states/blob/stable/files/database.sls and adapt it to your needs. An empty file is generated for you on first install.

Install a mastersalt master Ensure that your local box FQDN is correctly configured by issuing:

hostname -f

You should have something like that in your /etc/hosts:

```
127.0.0.1 mastersaltmaster.foo.net mastersaltmaster localhost
```

Then, you can proceed by bootstrapping mastersalt

```
mkdir -p /srv/mastersalt
apt-get install -y curl git
git clone https://github.com/makinacorpus/makina-states.git /srv/mastersalt/makina-states
/srv/mastersalt/makina-states/_scripts/boot-salt.sh --mastersalt-master --mastersalt $(hostname -f)
```

After installation you can begin to edit **/srv/mastersalt-pillar/database.sls** and bring up the rest of your new saltstack based infra, piece after piece !:

```
vim /srv/mastersalt-pillar/database.sls
```

WARNING The mastersalt binaries are prefixed with 'master' like: 'mastersalt', 'mastersalt-call', 'mastersalt-run', 'mastersalt-key'.

1.1.3 Projects for developers

Project management

Contents

- *Project management*
  - *Intro*
  - *Specifications*
    - *Initialization*
  - *Deploying, two ways of doing things*
    - *mc_project.init_project: initialize the layout*
    - *mc_project.deploy, the main entry point*
    - *Directly on the remote server, by hand*
      - *VARIANT: Deploy by hand, on a vagrant VM*
    - *Deploy with git instructions*
      - *Reminder*
      - *Deploy*
    - *Sumup*
    - *Configuration pillar & variables*
  - *What's happen when there is a deploy ?*
  - *Filesystem considerations*
  - *SaltStack integration*
  - *Related topics*

Intro

This page is the most important thing you ll have to read about makina-states as a **developer consumer**, take the time it needs and deserves.

Never be afraid to go read makina-states code, it will show you how to configure and extend it. It is simple python and yaml.

See python exemples:

- the modules (saltstack doc about modules)
- the states (saltstack doc about states)
- the runners (saltstack doc about runners)

See formulaes exemples:

- saltstack doc about states formulaes
- saltstack doc about states formulaes2
- the localsettings
- the services

Specifications

See the original *specification*, and specially the *layout*, the *install* procedure, and the *fixperms* procedure.

A good sumup of the spec is as follow, but please read it once...

- There is a separate repo distributed along the project named **pillar** to store configuration variables, passwords and so on.
- Projects are deployed via instructions based on saltstack which are contained into the **.salt** folder inside the codebase.

The deployment includes global phases in this order:

- archive `archive.sls`
- sync code from remotes if there are remotes
- sync/install custom salt modules (exec, states, etc) from the codebase if any
- fixperms (`fixperms.sls`)
- install (`install.sls`)
- fixperms
- rollback (`rollback.sls`)if error

Some of those phases can be edited via the user, and some other not (install, & sync steps).

That will explain that in your **.salt** folder, you have at least `install.sls`, `fixperms.sls`, `rollback.sls`, and for old projects `notify.sls`.

All other sls found at **toplevel** which are not those ones are executed in lexicographical order (alphanum) and the convention is to name them `\d\d\d_NAME.sls`

The `PILLAR.sample` file contains default configuration variable for your project and helps you to know what variable to override in your custom pillar.

Initialization

- a project in corpus / makina-states is a git repository checkout which contains the code and a well known saltstack based procedure to deploy it from end to end in the **.salt** folder.
- By default the project procedure is done via a masterless salt call.
- The first thing to do is to create a **nest** from such a project, **IF IT IS NOT ALREADY DONE** (just ls /srv/projects to check):

```
salt-call --local mc_project.deploy <project_name> # dont be long, dont use - & _
```

- This empty structure respects the aforementioned corpus reactor anatomy, and is just an useless helloword project which should look like:

```
/srv/projects/<project_name>
    |
    |- pillar/init.sls: override values in PILLAR.sample and define
    |                   any other arbitrary pillar DATA.
    |
    |- data/: anything which is persisted to disk must live here
    |         from drupal sites/default/files, python eggs, buildouts parts,
    |         gems cache, sqlite files, static files, docroots, etc.
    |
    |- project/ <- a checkout or your project
    |   |-  .git
    |   |-  codebase
    |   |-  .salt
    |     |- _modules : custom salt python exec modules
    |     |- _states  : custom salt python states modules
    |     |- _runners : custom salt python runners modules
    |     |- _sdb     : custom salt python sdb modules
    |     |- _...
    |     |
    |     |- PILLAR.sample
    |     |- task_foo.sls
    |     |- 00_deploy.sls
    |
    [ If "remote_less" is False (default)
    |- git/project.git: bare git repos synchronnized (bi-directional)
    |                   with project/ used by git push style deployment
    |- git/pillar.git:  bare git repos synchronnized (bi-directional)
                        with pillar/ used by git push style deployment
```

- What you want to do is to replace the `project` folder by your repo. This one contains your code, as asual, plus the **.salt** folder,
- **WELL Understand** what is :
  - a salt SLS , it is the nerve of the war.
  - the Pillar of salt.
- **be ware**, on the production server the `.git/config` is linked with the makina-states machinery and you cannot replace it blindly, you must use *Deploy with git instructions* to do it.
- Ensure to to have at least in your project git folder:
  - `.salt/PILLAR.sample`: configuration default values to use in SLSes
  - `.salt/archive.sls`: archive step
  - `.salt/fixperms.sls`: fixperm step

  - `.salt/rollback.sls`: rollback step

- You can then add as many SLSes as you want, and the ones directly in **.salt** will be executed in alphabetical order except the ones beginning with **task_** (task_foo.sls). Indeed the ones beginning with **task_** are different beasts and are intended to be either included by your other slses to factor code out or to be executed manually via the `mc_project.run_task` command.
- You can and must have a look for inspiration on *Projects list*

Deploying, two ways of doing things

To build and deploy your project we provide two styles of doing style that should be appropriate for most use cases.

The common workflow is:

- use `mc_project.init_project` to create the structure to host your project
- use `mc_project.report` to verify things are in place
- git push/or edit then push the pillar `/srv/projects/<project>/pillar` to configure the project
- git push/or edit then push the code inside `/srv/projects/<project>/project`
- launch the deploy
- Wash, Rince, Repeat

mc_project.init_project: initialize the layout The following command is the nerve of the war:

```
salt-call \
    --local -lall \
    mc_project.init_project $project [remote_less=false/true]
```

- `--local -lall` instructs to run in masterless mode and extra verbosity
- `mc_project.init_project $project` instructs to create the layout of the name `$project` project living into `/srv/projects/$project/project`
- (opt) `remote_less` instructs to deploy with or without the git repos that allow users to use (or not) a **git push to prod to deploy** workflow.
  - If `remote_less=true`, the git repos wont be created, and you wont be able to push to git remotes to deploy your project (you ll have to do it directly on the server, by the *hand procedure*.
  - If `remote_less=false`, you ll also be able to use the *push to prod feature*.

mc_project.deploy, the main entry point The following command is the nerve of the war:

```
salt-call \
    --local -lall \
    mc_project.deploy $project\
     [only=step2[,step1]] \
     [only_steps=step2[,step1]]
```

- `--local -lall` instructs to run in masterless mode and extra verbosity
- `mc_project.deploy $project` instructs to deploy the name `$project` project living into `/srv/projects/$project/project`

- (opt) `only` instructs to execute only the named global phases, and when deploying directly onto a machine, you will certainly have to use `only=install,fixperms,sync_modules` to avoid the archive/sync/rollback steps.
- (opt) `only_steps` instruct to execute only a specific or multiple specific sls from the **.salt** folder during the **install** phase.

Directly on the remote server, by hand Either directly from the deployment host as root:

Initialise the layout (only the first time)

```
ssh root@remoteserver
export project="foo"
salt-call --local -ldebug mc_project.init_project $project
```

Edit the pillar

```
ssh root@remoteserver
export project="foo"
cd /srv/projects/$project
# maybe you want to edit before pillar deploy
$ÊDITOR pillar/init.sls
cd pillar;git commit -m foo;git push;cd ..
```

Update the project code base from git

```
ssh root@remoteserver
export project="foo"
cd /srv/projects/$project/project
# if not already done, add your project repo remote
git remote add g https://github.com/o/myproject.git
# in any cases, update your code
git fetch --all
git reset --hard remotes/g/<the branch to deploy>
git push --force origin HEAD:master
```

Launch deploy

```
ssh root@remoteserver
# launch the deployment
export project="foo"
salt-call --local -ldebug \
    mc_project.deploy $project \
    only=install,fixperms,sync_modules
# or to deploy only a specific sls
salt-call --local -ldebug \
    mc_project.deploy $project \
    only=install,fixperms,sync_modules only_steps=000_foo.sls
git push o HEAD:<master> # replace master by the branch you want to push
                         # back onto your forge
```

VARIANT: Deploy by hand, on a vagrant VM We generally setup environments based on makina-states/vms that we share amongst our developers.

In development, our best practises are not to pull from our private git repositories directly from inside the VM.

The HOST on which the virtualbox is running, is on the contrary controlled by the developer and it's more safe to pull/push the code from here.

To sum up, any **git push/pull** operation has to be done **from the localhost** and not the vm.

In other words, the HOST can access any of the VM files with the help of a shared **sshfs** mountpoint `./VM`. And the HOST can also access the outside repositories. So the host in the interface that will push code inside the VM.

This setup involves using the `remote_less` feature of `mc_project` where we do not deploy via a `git push` nor use `archive/rollback` mechanims.

Initialise/launch a makina-states/vms box (this will take some time, specially the first time)

```
git clone https://github.com/makinacorpus/vms
cd vms
./manage.sh init
```

Open one console connected to the VM as **root**

```
./manage.sh ssh
sudo su # (default password: vagrant)
```

Initialise the layout (only the first time)

```
ssh root@remoteserver
export project="foo"
salt-call --local -ldebug mc_project.init_project $project remote_less=true
```

Edit the pillar

```
cd /srv/projects/$project/pillar
$EDITOR init.sls
git commit -am up
```

Open a second shell, on your local machine (**not on the VM**) where you ll update the project code base from git.

```
export project="foo"
cd vms/VM/srv/projects/$project/project
# if not already done, add your project repo remote
git remote add o https://github.com/o/myproject.git
# in any cases, update your code
git fetch --all
git reset --hard remotes/o/<the branch to deploy>
```

On the former shell ssh-connected to the vagrant box, launch deploy

```
salt-call --local -ldebug mc_project.deploy $project only=install,fixperms,sync_modules
# or to deploy only a specific sls
salt-call --local -ldebug \
    mc_project.deploy $project \
    only=install,fixperms,sync_modules only_steps=000_foo.sls
```

When you want to commit your changes, return to the second shell, on your local machine

```
export project="foo"
cd vms/VM/srv/$project/project
git push o HEAD:<master> # replace master by the branch you want to push
                         # back onto your forge
```

Deploy with git instructions

Reminder

- **WARNING**: you can use it only if you provisionned your project with attached remotes (the default)
- Remember use the remotes inside `/srv/projects/<project>/git` and not directly the working copies
- If you push on the pillar, it does not trigger a deploy
- If you push on the project, it triggers the full deploy procedure including archive/sync/rollback.
- To get useful push informations, on the remote server to deploy to, just do

```
salt-call --local -lall mc_project.report
```

Deploy The following lines edit the pillar, and push it, this does not trigger a deploy

```
cd $WORKSPACE/myproject
git clone host:/srv/projects/project/git/pillar.git
$EDITOR pillar/init.sls
cd pillar;git commit -am up;git push;cd ..
```

The following lines prepare a clone of your project codebase to be able to be deployed onto production or staging servers

```
cd $WORKSPACE/myproject
git clone git@github.com/makinacorpus/myawsomeproject.git
git remote add prod /srv/projects/project/git/project.git
git fetch --all
```

To trigger a remote deployment, now you can do:

```
git push [--force] prod <mybranch>:master
eg: git push [--force] prod <mybranch>:master
eg: git push [--force] prod awsome_feature:master
```

- **REMINDER:**
  - DONT MESS WITH THE **ORIGIN** REMOTE when your are connected to your server in any of the `pillar` or `project` directory..
  - The `<branchname>:master` is really important as everything in the production git repositories is wired on the `master` branch. You can push any branch you want from your original repository, but in production, there is only **master**.

Sumup To sum all that up, when beginning project you will:

- Initialize if not done a project structure with `salt-call --local mc_project.init_project project`
- If you do not want git remotes, you can alternativly use `salt-call --local mc_project.init_project project remote_less=true`
- add a **.salt** folder alongside your project codebase (in it's git repo).
- deploy it, either by:
  - git push your **pillar** files to `host:/srv/projects/<project>/git/pillar.git`
  - git push your **project code** to `host:/srv/projects/<project>/git/project.git` (this last push triggers a deploy on the remote server)

  - Your can use `--force` as the deploy system only await the `.salt` folder. As long as the folder is present of the working copy you are sending, the deploy system will be happy.
- or connected to the remote host to deploy onto
  - edit/commit/push in `host:/srv/projects/<project>/pillar`
  - edit/commit/push/push to force in `host:/srv/projects/<project>`
  - Launch the `salt-call --local mc_project.deploy <name> only=install,fixperms,sync_modules` dance
- Wash, Rince, Repeat

Configuration pillar & variables We provide in **mc_project** a powerfull mecanism to define default variables used in your deployments. hat you can safely override in the salt pillar files. This means that you can set some default values for, eg a domain name or a password, and input the production values that you won't commit along side your project codebase.

- Default values have to be stored inside the **PILLAR.sample** file.
- Some of those variables, the one at the first level are mostly read only and setup by makina-states itself. The most important are:
  - `name`: project name
  - `user`: the system user of your project
  - `group`: the system group of your project
  - `data`: top level free variables mapping
  - `project_root`: project root absolute path
  - `data_root`: persistent folder absolute path
  - `default_env`: environment (staging/prod/dev)
  - `pillar_root`: absolute path to the pillar
  - `fqdn`: machine FQDN
- The only variables that you can edit at the first level are:
  - **remote_less**: is this project using git remotes for triggering deployments
  - **default_env**: environement (valid values are staging/dev/prod)
  - **env_defaults**: indexed by **env** dict that overloads data (pillar will still have the priority)
  - **os_defaults**: indexed by **os** dict that overloads data (pillar will still have the priority)
- The other variables, members of the **data** sub entry are free for you to add/edit.
- Any thing in the pillar `pillar/init.sls` overloads what is in `project/.salt/PILLAR.sample`.

You can get and consult the result of the configuration assemblage like this:

```
salt-call --local -ldebug mc_project.get_configuration <project_name>
```

- Remember that projects have a name, and the pillar key to configure and overload your project configuration is based on this key.

  If your project is name **foo**, you ll have to use **makina-projects.foo** in place of **makina-projects.example**.

Example

in `project/.salt/PILLAR.sample`, you have:

```
makina-projects.projectname:
  data:
    start_cmd: 'myprog'
```

in `pillar/init.sls`, you have:

```
makina-projects.foo:
   data:
     start_cmd: 'myprog2'
```

- In your states files, you can access the configuration via the magic `opts.ms_project` variable.
- In your modules or file templates, you can access the configuration via `salt['mc_project.get_configuration'(name)`.
- A tip for loading the configuration from a template is doing something like that:

```
# project/.salt/00_deploy.sls
{% set cfg = opts.ms_project %}
toto:
  file.managed:
      - name: "source://makina-projects/{{cfg.name}}/files/etc/foo"
      - target: /etc/foo
      - user {{cfg.user}}
      - group {{cfg.user}}
      - defaults:
          project: {{cfg.name}}

# project/.salt/files/etc/foo
{% set cfg = opts.ms_project %}
My Super Template of {{cfg.name}} will run {{cfg.data.start_cmd}}
```

What's happen when there is a deploy ?

- When you do a git push, you have the full procedure, see *spec doc*
- When you use `only=install,fixperms,sync_modules` it only do some the *install* & *fixperms* procedures.

Filesystem considerations

We use POSIX Acls in various places on your project folders. At first, it feels a bit complicated, but it will enable you to smoothlessly edit your files or run your programs with appropriate users without loosing security.

SaltStack integration

As you know in makina-states, there are 2 concurrent salt installs, one for **salt**, the one that you use, and one for **mastersalt** for the devil ops. In makina-states, we use by default:

- a virtualenv inside `/salt-venv/salt`
- salt from a fork installed inside `/salt-venv/salt/src/salt`

- the salt file root resides, as usual, in `/srv/salt`
- the salt pillar root resides, as usual, in `/srv/pillar`
- the salt configuration root resides, as usual, in `/etc/salt`

As you see, the project layout seems not integration on those following folders, but in fact, the project initialisation routines made symlinks to integrate it which look like:

```
/srv/salt/makina-projects/<project_name>>  -> /srv/projects/<project_name>/project$/.salt
/srv/pillar/makina-projects/<project_name> -> /srv/projects/<project_name>/pillar
```

- The pillar is auto included in the **pillar top** (`/srv/pîllar/top.sls`).
- The project salt files are not and **must not** be included in the salt **top** for further highstates unless you know what you are doing.

You can unlink your project from salt with:

```
salt-call --local -ldebug mc_project.unlink <project_name>
```

You can link project from salt with:

```
salt-call --local -ldebug mc_project.link <project_name>
```

Related topics

You can refer to *mc_project_2 / project settings regitry APIV2*

Projects list

More generally, a research link: Projects As the above lists are far from exhaustive.

Projects Exemples

- zope
- django
- drupal
- flask
- php
- staticwww

Helpers & resource projects

- pgsql
- mysql
- elasticsearch
- solr
- osmdb

- rabbitmq
- mongodb

App deployements exemples

- tilestream
- odoo
- gitlab
- seafile
- redmine
- fusiondirectory
- vaultier
- jenkins
- jenkins-slave
- dockerregistry
- svn

1.1.4 Additionnal docs

Misc documentation

Journal of installing makina-states old an old uncentralamnaged box

get a recent git If you do not have a git >= 1.8:

```
apt-get build-dep git-core
wget https://git-core.googlecode.com/files/git-1.9.0.tar.gz
tar xzvf git*z
cd git*
make configure && ./configure && make
cd /usr/bin && mkdir oldgit && mv git* oldgit
cd - && make install
```

recent ssl

```
export CFLAGS="-fPIC"
wget http://www.openssl.org/source/openssl-1.0.1g.tar.gz
tar xzvf openssl-1.0.1g.tar.gz
cd openssl-1.0.1g
./config --prefix=/usr/local shared && make depend && make && make install && ldconfig
unset CFLAGS
```

get a recent python If you do not have a python >= 2.7:

```
export CFLAGS="-I/usr/local/include" LDFLAGS="-L/usr/local/lib"
apt-get build-dep python2.5
ln -s /usr/local/ /usr/local/ssl
wget https://www.python.org/ftp/python/2.7.6/Python-2.7.6.tgz --no-check-certificate
tar xzvf Python-2.7.6.tgz
cd Python-2.7.6
./configure CFLAGS="-I/usr/local/include" LDFLAGS="-L/usr/local/lib"  --prefix=/usr/local/ --disable-
mv -f /usr/local/bin/python /usr/local/bin/python.old
ldconfig
wget https://bitbucket.org/pypa/setuptools/raw/bootstrap/ez_setup.py -O - | python2.7
/usr/local/bin/easy_install -U virtualenv
```

ZMQ If you do not have libzmq >= 4

```
wget http://download.zeromq.org/zeromq-4.0.4.tar.gz
tar xzvf zeromq-4.0.4.tar.gz
cd zeromq-4.0.4
./configure --with-pgm --prefix=/usr/local && make && make install && ldconfig
```

YAML

```
wget http://pyyaml.org/download/libyaml/yaml-0.1.5.tar.gz
tar xzvf yaml-0.1.5.tar.gz
cd yaml-0.1.5
./configure --prefix=/usr/local && make && make install && ldconfig
```

Bootstrap makina-states with care

```
cd /srv
mkdir salt
cd salt
git clone git@github.com:makinacorpus/makina-states.git
cd makina-states
./_scripts/boot-salt.sh -b stable -m <MINION_ID>
```

Troubleshooting

```
Generated script '/srv/salt/makina-states/bin/buildout'.
Launching buildout for salt initialisation
Traceback (most recent call last):
  File "bin/buildout", line 17, in <module>
    import zc.buildout.buildout
  File "/srv/salt/makina-states/eggs/zc.buildout-1.7.1-py2.7.egg/zc/buildout/buildout.py", line 40, i
    import zc.buildout.download
  File "/srv/salt/makina-states/eggs/zc.buildout-1.7.1-py2.7.egg/zc/buildout/download.py", line 20, i
    from zc..buildout.easy_install import realpath
  File "/srv/salt/makina-states/eggs/zc.buildout-1.7.1-py2.7.egg/zc/buildout/easy_install.py", line 1
    import setuptools.package_index
  File "/usr/local/lib/python2.7/dist-packages/distribute-0.6.24-py2.7.egg/setuptools/package_index.p
    sys.version[:3], require('distribute')[0].version
  File "build/bdist.linux-x86_64/egg/pkg_resources.py", line 728, in require
    supplied, ``sys.path`` is used.
  File "build/bdist.linux-x86_64/egg/pkg_resources.py", line 626, in resolve
    ``VersionConflict`` instance.
```

```
pkg_resources.DistributionNotFound: distribute
Failed buildout
```

Update your system setuptools install to match latest setuptools (distribute + setuptools fork reunion):

```
sudo easy_install -U setuptools
```

Network route forwarding

There is a 'systctl' option controlling weither a datagram can be sent or not (http://en.wikipedia.org/wiki/IP_forwarding). You have to enable it for LXC to work. Another thing will be to make it persist to further reboots.

Create `/etc/sysctl.d/99_custom.conf`

```
net.ipv4.ip_forward = 1
```

And reload it with:

```
sysctl --system
```

Then ensure that it is enabled with:

```
sysctl net.ipv4.ip_forward
```

You may want to continue with:

- *Install the image on ubuntu*
- *Install the image on a systemD based host*

lxc makina-states documentation

Network firewalling and masquerating a makina-states LXC based image

This part is optionnal, and is relevant only if you use a firewall.

To ensure internet connectivity, you ll have to `masquerade` the 10.5/16 network to ensure the big internet dialog.

Although if you created the system-D or upstart job and it does that for you, if you use an aditionnal firewall, you ll have also to double the configuration in it.

For this, you have plenty of options depending of what firewalling software you are using.

The big picture The network which will have at the end will look like:

```
88.86.85.96   88.5.5.5
NET  ---------ROUTER-------- YourHOST --- LXCBR1 (10.5.0.1)
               10.6.1.25      10.6.1.1            |
                                                  |--- LXC1 (10.5.0.6)
                                                  |
                                                  |--- LXC2 (10.5.0.7)
                                                  |
                                                  |--- LXC3 (10.5.0.8)
```

The 10.6/16 network is your home or work network, which can be anything like 192.168.x.x or other **rfc1918** network.

- The first thing will be to allow the traffic to jump from **lxcbr1** to the segment in **10.6/0**. This is done by enabling the kernel ip forwarding option, see *Network route forwarding*.
- The second thing will be to remap the traffic from **10.5** from coming from your internal machine, this is what we call **IPV4 NAT**. We do this with the **masquerading** stuff.
- And the third thing will be to **allow** the firewall to let pass the **lxc traffic**.

Step2 and Step3 are often done with your firewall software.

shorewall

- Ensure to apply *Network route forwarding*.

You ll have to allow ip forwarding in `/etc/shorewall/shorewall.conf`:

```
IP_FORWARDING=Yes
```

You ll have to add a `masq` in `/etc/shorewall/masq`:

```
br0 lxcbr1  -   -   -   -   -
```

Replace br0 with your primary interface like `eth0` or `em1`?

You ll have to create a **lxc** zone in `/etc/shorewall/zones`:

```
lxc ipv4  -   -   -
```

You ll have then to attach lxcbr1 to a **lxc** zone in `/etc/shorewall/interfaces`:

```
lxc lxcbr1  routeback,bridge,tcpflags,nosmurfs
```

Then, you ll mark this lxc zone as trusted in `/etc/shorewall/policy`, Make sure that the lxc rules are prior to any blocking rules.:

```
lxc net ACCEPT  -   -
$FW lxc ACCEPT  -   -
```

Then reload shorewall:

```
shorewall safe-restart
```

You may want to continue with:

- *Note about firewalling on ubuntu*
- *Note about firewalling on a systemD based host*

firewalld

- Ensure to apply *Network route forwarding*.

Identify and allow lxc traffic

```
firewall-cmd --add-zone=lxc --permanent
firewall-cmd --permanent --zone=lxc --add-interface=lxcbr1
firewall-cmd --permanent --zone=lxc --set-target=ACCEPT
```

Masquerade public ip Masquerade whatever what are your public zones with one or more of the following commands:

```
firewall-cmd --permanent --zone=home --add-masquerade
firewall-cmd --permanent --zone=external --add-masquerade
firewall-cmd --permanent --zone=public --add-masquerade
```

You may want to continue with:

- *Note about firewalling on ubuntu*
- *Note about firewalling on a systemD based host*

ufw

- Ensure to apply *Network route forwarding*.
- create or edit `/etc/default/ufw` and add/update **DEFAULT_FORWARD_POLICY**

```
DEFAULT_FORWARD_POLICY="ACCEPT"
```

- Create or edit `/etc/ufw/before.rules` and add or adapt

```
*nat
:POSTROUTING ACCEPT [0:0]
-A POSTROUTING -s 10.5/16 -o eth0 -j MASQUERADE
# don't delete the 'COMMIT' line or these nat table rules won't be processed
COMMIT
```

- You will have to add here any network you are bridging from the lxcbr1 bridge (and by default we use 10.5/16).

You may want to continue with:

- *Note about firewalling on ubuntu*
- *Note about firewalling on a systemD based host*

iptables This means that you manage your firewall manually, you are on your own baby, just allow the traffic from and to lxcbr1 (10.5/16) and masquerade it.

You may want to continue with:

- *Note about firewalling on ubuntu*
- *Note about firewalling on a systemD based host*

Install the base LXC container

First ensure that there is plenty of space on `/var/lib/lxc`. Plently as at least **10GB**.

If you do not have enought place on the partition, you may have another mounted partition or one that you can create which will provide that extra space. Follow the next chapter to add this extra space.

On other cases, you can directly jump to *Install base LXC Image*.

Make room for space (optional)

Mount your origin partition When you have this partition, first ensure that it is mounted, and it has a relevant entry on your `/etc/fstab` for it to be mounted at boot time.

For example, Imagine that you have mounted your data partition in `/home`, you will have to have or to add one like which looks like one on those following entries in your `/etc/fstab` file:

```
# Origin target fstype fsopts * *
UUID=19710386-5ed2-4b6c-b289-628adac75e5b /home ext4 defaults 0 0
# those lines are equivalent
# /dev/sdc4                               /home ext4 default 0 0
```

If you want to use `UUIDS`, you can find the uuid belonging to one partition like this:

```
ls /dev/disk/by-uuid/
```

Of course, **xfs** and **ext4** can be any supported filesystem except fat or ntfs.

The options **defaults** can be different on your installation, this is probably not a problem.

The filesystem can by encrypted, it will obisouly be slowier but it will work.

Map your original partition to `/var/lib/lxc` Then, you ll have to map this extra big partition to `/var/lib/lxc`. Wwe will use that as a `bind mound` (a powerfull symlink like redirection).

We will call `/home`, this extra big partition mountpoint.

Remapping a directory from this partition is as simple as:

- Creating the directory on your partition holding the new mountpoint:

```
    mkdir -p /home/var/lib/lxc
```

- adding a new entry at the bottom of your fstab:

```
    /home/var/lib/lxc /var/lib/lxc none bind,defaults,exec 0 0
```

You will obviously replace `/home` by the mountpoint location of your extra big data partititon. You will obviously replace `/home/var/lib/lxc` by the directory you had created previously.

You can then activate this configuration it with:

```
mount /var/lib/lxc
```

And verify the extra space on it with:

```
df -f /var/lib/lxc
```

Which should be the same that:

```
df -h /home/var/lib/lxc
```

As it is now on your fstab, it will survive to reboots.

- That means that `/home/var/lib/lxc` is mounted in place of `/var/lib/lxc`
- Anything which is written to or accessed from `/var/lib/lxc` will instead be written in `/home/var/lib/lxc`.

Install base LXC Image Download and install the lxc container is simplified through a python script.

Download a copy of makina-states, which contains helpers:

```
git clone https://github.com/makinacorpus/makina-states.git -b stable
```

You can do that by issuing as root those following commands:

```
cd makina-states
git checkout stable
# either sudo, or use a root shell
sudo ./_scripts/restore_lxc_image.py
```

This will download and install your image in `/var/lib/lxc`.

Finish installation If you were following the general installation procedure for the LXC procedure, you may go back to the general docuementation by following one of those following links:

- *Note about firewalling on ubuntu*
- *Note about firewalling on a systemD based host*

Create a container

Briefing As soon as you have the base template or any template derived from it, you can spawn a new container based on those template.

The filesystem from this base container will be copied, and all further modifications will only exist on your new container.

There are some steps involved in cloning a makina-states based container:

- clone the LXC container
- reset the lxc bare informations like the mac, and the ip
- reset the SSH and salt information inside the new container
- maybe mark the salt installations (salt & mastersalt) as masterless

For this we created a simple helper freeing you from this hassle.

Spawn the container You can use it the following way:

```
cd makina-states
sudo ./_scripts/spawn_container.py --name=<name> [ ... optional opts ]
```

Examples:

```
# get random non allocated it
sudo ./_scripts/spawn_container.py --name=<name>
sudo ./_scripts/spawn_container.py --ip=<new_ip in 10.5 range> --name=<name>
```

The name of the container will become it's **minion_id**.

A good idea is to name containers with a FQDN like:

```
sudo ./_scripts/spawn_container.py [--mac=xx:xx:xx:xx:xx:xx] [--ip=10.5.0.3] --name=myproject.lxc.lo
```

The ip and mac will be generated from non allocated ports, if they are not specified, you can then edit the LXC config to get them:

```
vim /var/lib/lxc/<container_name>/config
```

They are the value of:

- `lxc.network.ipv4`
- `lxc.network.hwaddr`
- You may use sudo to allow automaticly the sudoer user ssh keys to connect into the container via ssh

Network configuration To access services on your container, you may want to edit your **HOST** `/etc/hosts` and add those types of redirections:

```
10.5.0.3 myproject.lxc.local www.foo.com
```

Where:

- **10.5.0.3** is the ip of your container
- **myproject.lxc.local** and **www.foo.com** would be domains that you want to access inside your lxc.

Think that any entry in your **/etc/hosts** will shadow any real DNS information, and that can be harmful if you use real DNS informations like **www.google.com**, just remember if you shadows a real name, that your LXC will be used in place of the real internet service.

SSH connection All the keys from your host root's ssh folder and user folder if the container was spawned using sudo are allowed to connect as **root** via ssh on the container.

Connect to your container these ways:

- ssh roo@10.5.0.3
- ssh root@@myproject.lxc.local (if you added something in your `/etc/hosts`

Edit files of the container from your host You can use **sshfs** to easily edit the files inside your container from your host.

Just issue:

```
mkdir myhost
sshfs roo@10.5.0.3:/ myhost
```

This will mount the **/** of your container in myhost, and you ll can edit the files as if you were directly on the host. Don't forget that you are connected as root user, and files created will be owned by root and respect UMASK and acls settings from your container.

To umount:

```
fusermount -u myhost
```

If your shell seems blocked:

```
killall -9 sshfs
fusermount -u myhost
```

if the fusermount failed:

```
sudo umount -f /home/user/myhost
```

Conclusion You may now want to continue with

- *Conclusion (ubuntu)*
- *Conclusion (systemD based host)*

Writing new states

Write a new makina-state service

We will take the circus implementation as exemple.

Integration of service needs:

- A *salt execution module* to store special service parameters and configuration
- An entry in mc_services.py:registry method to autoinstall it when needed
- A line incorporating the module parameters in mc_services.py:settings

Taking circus as an example:

- service settings
- service autoload
- place for formula states

CHAPTER 2

Reference

2.1 Reference

2.1.1 Implementation design & details

RFC: corpus, embedded project configuration with salt

The origin

Nowoday no one of the P/I/S/aaS existing platforms fit our needs and habits. No matter of the gret quality of docker, heroku or openshift, they did'nt make it for us. And, really, those software are great, they inspired corpus a lot ! Please note also, that in the long run we certainly and surely integrate those as plain **corpus** drivers to install our projects on !

For exemple, this is not a critisism at all, but that's why we were not enougth to choose one of those platforms (amongst all of the others):

'heroku'_

non free

docker

- Not enougth stable yet, networking integration is really a problem here. It is doable, but just complicated and out of scope.
- do not implement all of our needs, it is more tied to the 'guest' part (see next paragraphs)
- But ! Will certainly replace the LXC guests driver in the near future.

openshift Tied to SElinux and RedHat (we are more on the Debian front ;)). However, its great design inspired a lot of the corpus one.

openstack Irrelevant and not so incompatible for the PaaS platform, but again we didn't want locking, openstack would lock us in the first place. We want an agnostic PaaS Platform.

The needs

That's why we created a set of tools to build the best flexible PaaS platform ever. That's why we call that not a PaaS platform but a Glue PaaS Platform :=).

- Indeed, what we want is more of a CloudController + ToolBox + Dashboards + API.

- This one will be in charge of making projects install any kind of compute nodes running any kind of VMs smoothly and flawlessly.
- Those projects will never ever be installed directly on compute nodes but rather be isolated.
  - They will be isolated primarly by isolation-level virtualisation systems (LXC, docker, VServer)
  - Bue they must also be installable on plain VMs (KVM, Xen) or directly baremetal.
- We don't want any PaaS platform to suffer from some sort of lockin.
- We prefer a generic deployment solution that scale, and better AUTOSCALE !
- All the glue making the configuration must be centralized and automatically orchestrated.
- This solution must not be tied to a specific tenant (baremetal, EC2) nor a guest driver type (LXC, docker, XEN).
- Corrolary, the **low level** daily tasks consists in managment of:
  - network (< OSI L3)
  - DNS
  - Mail
  - operationnal supervision
  - Security, IDS & Firewalling
  - storage
  - user management
  - baremetal machines
  - hybrid clouds
  - public clouds
  - VMs
  - containers (vserver, LXC, docker)
  - operationnal supervision
- Eventually, on top of that orchestrate projects on that infrastructure
  - installation
  - continenous delivery
  - intelligent test reports, deployment reports, statistic, delivery & supervision dashboards
  - backups
  - autoscale

Here is for now the pieces or technologies we use or are planning or already using to achieve all of those goals:

- Developer environments
  - makina-corpus/vms + makina-corpus/makina-states + saltstack/salt
- Bare metal machines provision
  - makina-states + saltstack
  - Ubuntu server
- VMs (guests)

  - Ubuntu based lxc-utils LXC containers + makina-states + saltstack
- DNS:
  - makina-states + bind: local cache dns servers & for the moment dns master for all zones
  - **FUTURE** makina-states + powerdns: dynamic managment of all DNS zones
- Filesystem Backup
  - burp
- Database backup
  - db_smart_backup
- Network:
  - ceph, openvswitch
- Logs, stats:
  - now: icinga2 / pnp for nagios
  - future: icinga2 / logstash / kibana
- Mail
  - postfix
- User managment (directory)
  - Fusion directory + openldap
- Security
  - at least shorewall & fail2ban
- CloudController
  - saltstack
  - makina-states + makina-states/mastersalt
- projects installation, upgrades & contineous delivery
  - makina-states (mc_project, *Intro*)
- autoscale
  - makina-states

The whole idea

The basic parts of corpus PaaS platform:

- The cloud controller
- The cloud controller client applications
- The compute nodes
  - Where are hosted guests
    - Where projects run on
  - The developer environments which are just a special kind of compute nodes

The first thing we will have is a classical makina-states installation in mastersalt mode. We then will have salt cloud as a cloud controller to control compute nodes via **makina-states.services.cloud.{lxc, saltify, ...}** (lxc or saltify) Those compute nodes will install guests. Those guests will eventually run the final projects pushed by users.

Hence an api and web interface to the controller we can:

- Add one or more ssh key to link to the host
- Request to link a new compute node
- Request to initialize a new compute node
- List compute nodes with their metadata (ip, dns, available slots, guest type)
- Get compute ndoos/container/vms base informations (ssh ip / port, username, pasword, dns names)
- Link more dns to the box
- Manage (add or free) the local storage.
- Destroy a container
- Unlink a compute node

The users will just have either: - Push the new code to deploy - Connect via ssh to do extra manual stuff if any including a manual deployment

Permission accesses

- We will use an ldap server to perform authentication

The different environment platforms

We also want to distinguish at least those 3 environments, so 3 ways for you to deploy at least.

dev The developper environments (laptop)

staging the stagings and any other QA platform

prod the production platform

Objectives

The layout and projects implementation must allow us to

- Automaticly rollback any unsucessful deployment
- In production and staging, archive application content from N last deployments
- Make the development environment easily editable
- Make the staging environment a production battletest server
- Production can deploy from non complex builds, and the less possible dependant of external services

For this, we inspired ouselves a lot from openshift and heroku (custom buildpacks) models.

Actual layout

Overview of the project source code repositories A project will have at least 2 local git repositories:

```
/srv/projects/myproject/git/project.git/
  A repository where lives its sourcecode and deployment recipes
/srv/projects/myproject/git/pillar.git/
  A repository where lives its pillar
```

This repository master branch consequently has the minimal following structure:

```
master
    |- what/ever/files/you/want
    |- .salt -> the salt deployment structure
    |- .salt/PILLAR.sample     -> default pillar used in the project, this
    |                             file will be loaded inside your
    |                             configuration
    |- .salt/rollback.sls      -> rollback code run in case of problems
    |- .salt/archive.sls       -> pre save code which is run upon a deploy
    |                             trigger
    |- .salt/fixperms.sls      -> reset permissions script run at the end of
    |                           deployment
    |- .salt/_modules          -> custom salt modules to add to local salt
    |       /_runners             install
    |       /_outputters
    |       /_states
    |       /_pillars
    |       /_renderers
    |
    |- .salt/00_DEPLOYMENT.sls -> all other slses will be executed in order
                                  and are to be provided by th users.
```

- A private repository with restricted access with any configuration data needed to deploy the application on the PAAS platform. This is in our case the project pillar tree:

```
    pillar master
       |- init.sls the pillar configuration
```

As anyways, you ll push changes to the PAAS platform, no matter what you push, the PAAS platform will construct according to the pushed code :). So you can even git push -f if you want to force things.

Overview of the paas local directories

/srv/projects/myproject/project/ The local clone of the project branch from where we run in all modes. In other words, this is where the application runtimes files are. In application speaking

- **django/python ala pip:** the virtualenv & root of runtime generated configuration files
- **zope:** this will the root where the bin/instance will be lauched and where the buildout.cfg is
- **php webapps:** this will be your document root + all resources
- **nodejs:** etc, this will be where nginx search for static files and where the nodejs app resides.

/srv/projects/myproject/pillar The project specific states pillar tree local clone.

/srv/projects/myproject/data/ Where must live any persistent data

/srv/pillar/makina-projects/myproject -> /srv/projects/myproject/pillar pillar symlink for salt integration

/srv/salt/makina-projects/myproject -> /srv/projects/myproject/.salt/<env> state tree project symlink for salt integration

/srv/salt/{_modules,runners,outputters,states,pilalrs,renderers}/*py -> /srv/projects/myproject/.salt/<typ>/mod.py custom salt python execution modules

The deployment procedure is as simple a running meta slses which in turn call your project ones contained in a subfolder of the **.salt** directory during the **install** phase.

The **.salt** directory will contain SLSs executed in lexicographical order. You will have to take exemple on another projects inside **makina-states/projects** or write your states. Those slses are in charge to install your project.

- The **persistent configuration directories**

  /etc static global configuration (/etc)

- **The persistent data directories** If you want to deploy something inside, make a new archive in the release directory with a dump or a copy of one of those files/directories.

  /var Global data directories (data & logs) (/var) Minus the package manager cache related directories

  /srv/projects/project/data

  - Specific application datas (/srv/projects/project/data)
    - Datafs and logs in zope world
    - drupal thumbnails
    - mongodb documentroot
    - ...

- **Networkly speaking**, to enable switch of one container to another we have some solutions but in any case, **no ports** must be **directly** wired to the container. **Never EVER**.

Either:

- Make the host receive the inbound traffic data and redirect (NAT) it to the underlying container
- Make a proxy container receive all dedicated traffic and then this specific container will redirect the traffic to the real underlying production container.

Procedures

Those procedure will be implemented by either:

- Manual user operations or commands
- Git hooks
- salt execution modules
- jinja macros (collection of saltstack states)

All procedures are tied to a **default** sls inside the **.salt** project folder and can per se be overriden.

Project initialization/sync procedure

- Initiate the project specific user
- Initiate the ssh keys if any
- Initiate the pillar and project bare git repositories inside the git folder

- Clone local copies inside the project, pillar and salt directories
- If the salt folder does not exists, create it
- If any of default slses procedures are not yet present, create them
- Wire the pillar configuration inside the pillar root
- Wire the pillar init.sls file to the global pillar top file
- Wire the salt configuration inside the salt root
- Echo the git remotes to push the new deployement on.
- Wire any salt modules in .salt/{_modules,runners,etc}

Project archive procedure

- If size is low, we enlarge the container
- run the pre archive hooks
- archive the **project** directory in an **archive/deployed** subdirectory
- run the post archive hooks (make extra dumps or persistent data copies)
- run the archives rotation job

Project Release-sync procedure

- Be sure to sync the last git deploy hook from makina-states
- Fetch the last commits inside the **deploy** directory

Project install procedure We run all slses in the project **.salt** directory which is not tied to any default procedure.

Project fixperms procedure

- Set & **reset (enforce)** needed user accesses to the filesystem

Rollback procedure

- Only run if something have gone wrong
- We move the failed **project** directory in the deployment **archives/<UUID>/project.failed** sub directory
- We sync back the previous deployment code to the **project** directory
- We execute the rollback hook (user can input database dumps reload)

Workflows

Full procedure

- project **deployment** is triggered
- project **archive** procedure
- project **initialization/sync** procedure
- project **release-sync** procedure

- project **fixperms** procedure
- project **install** procedure
- project **fixperms** procedure (yes again)
- In error: **rollback** procedure

IMPLEMENTATION: How a project is built and deployed

For now, at makinacorpus, we think this way:

- Installing somewhere a mastersalt master controlling compute nodes and only accessible by **ops**.
- Installing elsewhere at least one compute node which will receive project nodes (containers):
  - linked to this mastersalt as a mastersalt minion
  - a salt minion linked to a salt master which is probably local and controlled by **project members aka devs**, by default these salt minion and salt master services are toggled off and the salt-call should be runned **masterless** (salt-call –local)

Initialisation of a cloud controller

Complex, contact @makinacorpus.

This incude:

- Setting up the dns master & slaves for the cloud controlled zone.
- Setting up the cloud database
- Setting up at least one compute node to deploy projects
- Deploying vms

Request of a compute node or a container

- Edit the mastersalt database file to include your compute node and vms configuration.
- Run any appropriate mastersalt runners to deploy & operate your compute nodes and vms

Initialisation of a compute node

This will in order:

- auth user
- check infos to attach a node via salt cloud
- Register DNS in the dns master for thie compute node and its related vms
- generate a new ssh key pair
- install the guest_type base system (eg: makina-states.services.virt.lxc)
- Generate root credentials and store them in grains on mastersalt
- Configure the basic container pillar on mastersalt
  - root credentials

  - dns
  - firewall rules
  - defaultenv (dev, prod, preprod)
  - compute mode override if any (default_env inside /srv/salt/custom.sls)
- Run the mastersalt highstate.

Initialisation of a project - container environment

This will in order:

- auth user
- Create a new container on endpoint with those root credentials
- Create the layout
- use the desired salt cloud driver to attach the distant host as a new minion
- install the key pair to access the box as root
- Generate root credentials and store them in grains on mastersalt
- Configure the basic container pillar on mastersalt
  - root credentials
  - dns
  - firewall rules
- Run the mastersalt highstate

Initialisation of a project

- We run the initalization/sync project procedure
- Send a mail to sysadmins, or a bot, and initial igniter with the infos of the new platform access
  - basic http & https url access
  - ssh accces
  - root credentials
- User create the project
- Project directories are initialised

upgrade of a project The code is not pull by production server it will be pushed with git to the environment ssh endpoint:

- Triggered either by an automatted bot (jenkins)
- By the user itself, hence he as enougth access

In either way, the trigger is a git push.

The nerve of the war: jinja macros and states, and execution modules Project states writing is done by layering a set of saltstacl **sls** files in a certain order. Those will ensure an automatic deployment from end to end. The salt states and macros will abuse of execution modules to gather informations but also act on the underlying system.

The project common data structure

Overview

- to factorize the configuration code but also keep track of specific settings, those macros will use a common data mapping structure which is good to store defaults but override in a common manner variables via pillar.
- all those macros will take as input this **configuration** data structure which is a mapping containing all variables and metadata about your project.
- this common data mapping is not copied over but passed always as a reference, this mean that you can change settings in a macro and see those changes in later macros.

The project configuration registry execution module helper The base execution module used for project management is *mc_project / project settings regitry switcher and common functions*

It will call under the hood the latest **API** version of the mc_project module.

eg: `mc_project_2.*`

This will define methods for:

- Crafting the base **configuration** data structure
- initialising the project filesystem layout, pillar and downloading the base sourcecode for deployment (salt branch)
- deploying and upgrading an already installed project.
- Setting a project configuration

If there are too many changes in a project layout, obviously a new project API module should be created and registered for the others to keep stability.

APIV2

The project execution module interface (APIV2) Note that there two parts in the module:

- One set of methods are the one you are most likely to use handle local deployment
- One another set of methods is able to handle remote deployments over ssh. The only requirement for the other host is that makina-states should be installed first and ssh access should be configured previously to any deploy call. The requirement was to have only a basic ssh access, that why we did not go for a RAET or 0Mq salt deployment structure here.

See *mc_project_2 / project settings regitry APIV2*

The project sls interface (APIV2) Each project must define a set of common sls which will be the interfaced and orchestred by the project execution module. Theses sls follow the aforementionned procedures style.

The important thing to now remember is that those special sls files cannot be run without the project runner execution module

Indeed, we inject in those sls contextes a special **cfg** variable which is the project configuration and without we can't deploy correctly.

- We have two sets of sls to consider
  - **The set of sls providen by a makina-states installer** this is specified at project creation and stored in configuration for further reference

  - **The set of sls providen by the project itself in the .salt directory this is where the user will customize it's deployment steps**.

The installer set is then included by default at the first generation of the user installer set at the creation of the project.

Project initialisation & installation

- Refer to *Intro*
- Some installers example: *Projects list*

Design

Base Filesystem Layout

Please respect the following layout when you write states

Salt

- **/salt-venv**: salt virtualenv
- **/srv/salt**: salt file root
- **/srv/salt/makina-states**: makina-states clone
- **/srv/pillar**: Pillar
- **/var/log/salt**: log files
- **/var/cache/salt**: salt cache files
- **/var/run/salt**: salt run files
- **/etc/salt**: salt configuration files
- **/etc/makina-states/branch**: current used makina-states branch
- **/etc/makina-states/nodetype**: configuration of the current nodetype
- **/etc/salt/makina-states**: local registries

Mastersalt mode

- **/mastersalt-venv**: mastersalt virtualenv
- **/srv/mastersalt**: mastersalt file root
- **/srv/mastersalt/makina-states**: makina-states clone
- **/srv/mastersalt-pillar**: Pillar
- **/var/log/mastersalt**: log files
- **/var/cache/mastersalt**: mastersalt cache files
- **/var/run/mastersalt**: mastersalt run files
- **/etc/mastersalt**: mastersalt configuration files
- **/etc/mastersalt/makina-states**: local registries

Misc locations

- **/srv/apps**: Third party & non system packaged application
- **/srv/backups**: root for file based backups or database dumps

Projects integration

- **/srv/projects/<project_name>/salt**:
- **/srv/projects/<project_name>/pillar**:
- **/srv/projects/<project_name>/project**:
- **/srv/pillar/makina-projects/<project_name>/salt**: Symlink to the pillar project directory
- **/srv/salt/makina-projects/<project_name>/salt**: Symlink to the salt project directory

Main states kind segregation

Overview In makina states we have organised our states in the following way:

controllers states related to the base salt infrastructure (layout, deamons, permissions)

nodetypes states related only to the hardware, the machine type or virtualisation facillity

localsettings states related to the base machine configuration minus all that can looks like a service or a deamon (eg: directories creations, hosts managment, pam, vim but not ssh)

services states related to services configuration (eg: ssh, docker, apache, backups scripts & crons)

projects Set of macros to be externallly included project consumers as the base to setup their projects using makina-states bricks

cloud set of related stuff linked to mastersalt salt cloud integration

For all those kinds, we have execution modules, sub-execution modules and formulaes containers. to leverage and factorize all variables and have well placed macros.

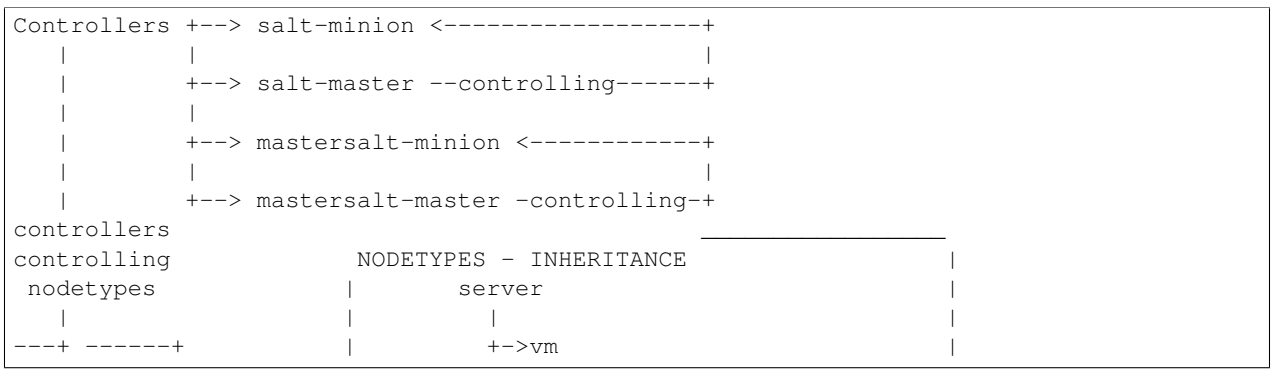
For exemple, php users will certainly have to deal the following files:

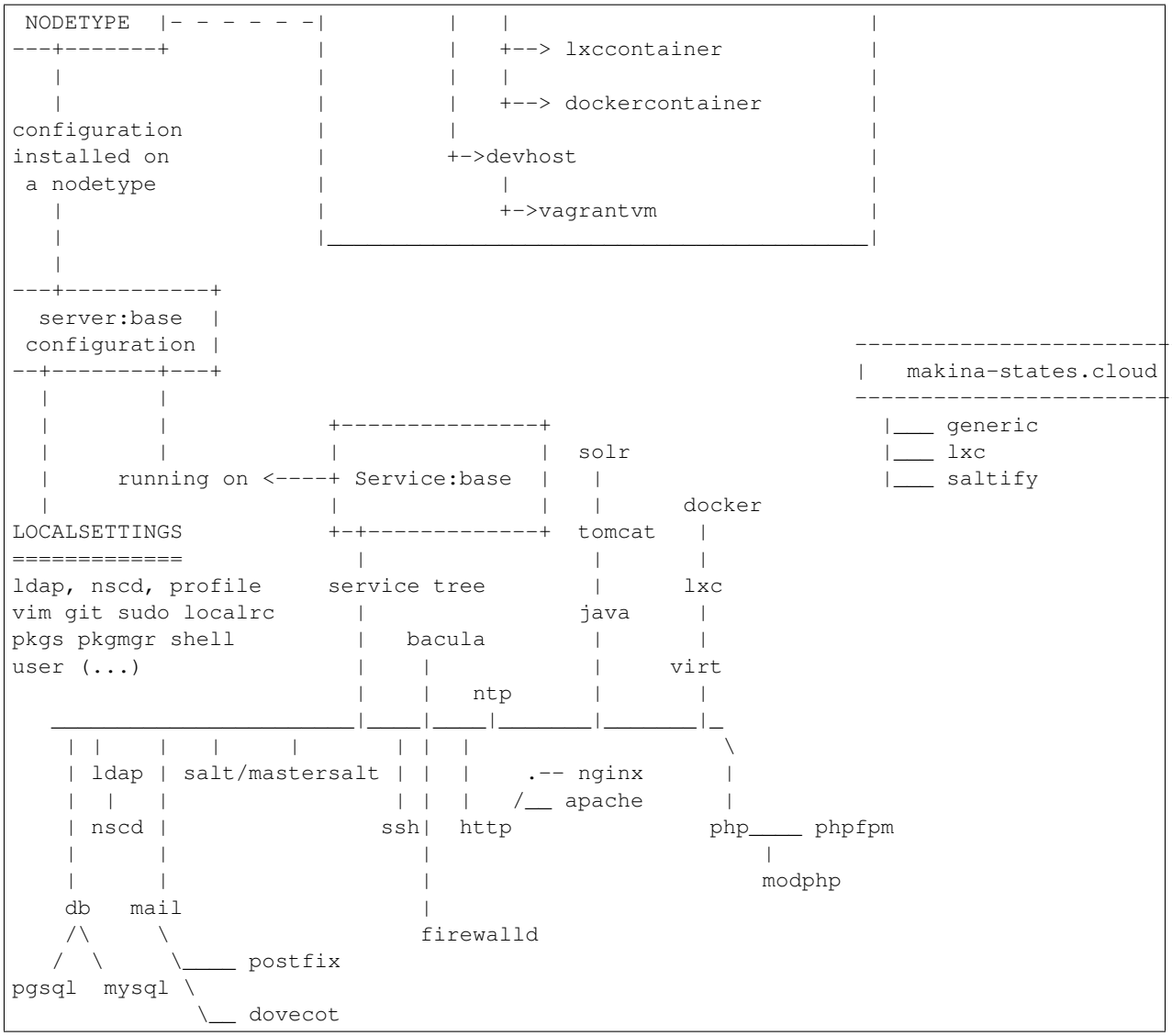
mc_states/modules/services.py registries

mc_states/services/{http,php}/*.sls states files (formulaes)

/srv/pillar/foo.sls custom settings

Wrap-up Tree of different configuration flavors inheritance:





Registries

Registries types For each kind of configurations, we use salt execution modules as settings storages to store metadata, configuration settings and state inclusion registry. We have "global registries", inner subregistries and "sub inner registries" like "services", "services.settings" and "php.settings" registries.

Metadata registry & inclusion registries are only are mandatory for global sub registries.

Those registries are simple python dictionnaries.

The modules are many of the mc_states.mc_* modules (mc_services, mc_nodetypes, mc_php, etc)

Metadata Registry which stock a name (eg: nodetypes, localsettings, services, controllers) and the global registries to load before this registry.

Example:

```
{
    'name': 'controllers',
    'bases': ['localsettings'],
}
```

Settings Registry which stock arbitrary configuration settings related to the current registry. Eg:

- apache: registry.worker -> the mpm worker
- localsettings: locations -> common path prefixes for various loations (/usr/bin, salt root)

Example:

```
{
    'port': 1234,
    'password': 'foo',
}
```

Registry This registry will contain the **states prefix** and **grains prefix** to construct autoinclusion slses files. It contains also the 'default inclusion status' for any linked states in the **defaults** key. This registry will load then from configuration the state of inclusion of any included/configured sub state and feed some shortcuts like **actives**, **is** (**has** is a copy/synonym of **is**) and **unactivated** settings.

For example, if you have php disabled as a default, but have installed or enabled it via the pillar, the registry will load it as 'active' thus triggering the autoinclusion of 'php' during an highstate. This registry may also be queried from any state field to known if a state is active (eg: querying if we are on a devhost by looking if devhost is in the nodetypes registry.

This registry lool like:

```
{
    'states_pref': 'makina-states.services',
    'grains_pref': 'makina-states.services',
    'actives': [{'php.fpm':{'active': True} }],
     'unactivated' [{'php.modphp':{'active': False} }],
     'defaults': {
        'php.fpm':{'active': True} },
        'php.modphp':{'active': False},
     }
     'is' : {'modphp': False, 'phpfpm': True},
     'has' : {'modphp': False, 'phpfpm': True},
}
```

The default general order of inclusion is as follow:

- Local settings
- Controllers
- Nodes Types
- Services

Formulaes

A formulae is a set of states to install and configure a particular subset of a machine, from provision, to setup the editors up to configure an httpd server. All formulaes containing states to deliver your applications and services follow the same organisation rules that we previously have seen in registries:

- nodetypes: preset for a type of machine
- controllers: salt/mastersalt
- localsettings: base conf (network, editors, languages, compilators, etc)
- services: databases, servers (sshd), etc.

Examples:

- makina-states.nodetypes.dockercontainer
- makina-states.controllers.salt_master
- makina-states.localsettings.vim
- makina-states.localsettings.python
- makina-states.services.http.apache
- makina-states.services.http.nginx
- makina-states.services.base.openssh.server
- makina-states.services.proxy.haproxy

Inside those states files, the idea is to configure the states via the underlying registries shared via salt execuction modules.

Hooks

Hooks in the makina-states meaning have two functions:

- Provide a robust orchestration mecanism
- Provide a way to skip a full subset of states by non inclusion of the real states files during different execution modes (standalone, full).

Those hooks are just states which use the **mc_proxy.hook** function wich does nothing more forward the changes of acendant of the states to the descendant of this state. For example in:

```
one:
    cmd.run:
        - name: /bin/true

foo:
    mc_hook.proxy:
        - require:
            - cmd: one

two
    mc_hook.proxy:
        - require:
            - mc_proxy: foo
```

As one will return a change, foo will also trigger in turn a change, thus two will see foo as 'changed'.

You can also easily skip a full subset of states, imagining the three following files:

- **a.sls**:

```
    include:
        - hooks
```

```
    one:
        cmd.run:
            - name: /bin/true
            - watch_in
                - cmd: one
```

- **hooks.sls**:

```
    foo:
       mc_hook.proxy: []
```

- **c.sls**:

```
    include:
        - hooks
        {{ if something }}
        - a
        {{ endif }}

    two
        mc_hook.proxy:
            - require:
                - mc_proxy: foo
```

You just discoved the magic of hooks to make custom execution modes without having to clutter your states files with custom require/includes, you just have to subscribe to isolated hooks, and in the real states files, to also subscribe to those hooks. In the listeners, you just have to conditonnaly include the real state file for the job to be correctly executed with the other states.

Tests & Quality Assurance

This will run:

- unit tests
- linters
- install all states

See travis test helper as an entry point to launch the tests on your environment.

Basically, it use nosetests via a custom salt module: mc_test

2.1.2 States & modules reference

Custom states modules

Execution modules

Registries

mc_bootstraps / bootstraps related registry

`mc_states.modules.mc_bootstraps.`**`metadata`**`()`
: metadata registry for bootstraps

`mc_states.modules.mc_bootstraps.`**`registry`**`()`
: registry registry for bootstraps

mc_states.modules.mc_bootstraps.**settings**()
settings registry for bootstraps

mc_nodetypes / nodetypes registry

mc_states.modules.mc_nodetypes.**get_makina_grains**()
Expose real time grains

mc_states.modules.mc_nodetypes.**has_system_services_manager**()
Does the actual host has a system level services manager aka a PIDEINS

mc_states.modules.mc_nodetypes.**metadata**()
nodetypes metadata registry

mc_states.modules.mc_nodetypes.**registry**()
nodetypes registry registry

mc_states.modules.mc_nodetypes.**settings**()
nodetypes settings registry

mc_controllers / controllers related variables

mc_states.modules.mc_controllers.**allow_lowlevel_states**()
Do we allow low level states

in dual stack saltstack installs, only allow low level states on the mastersalt side in other cases, without the presence of the conf flag this will return 'unkown' also ensuring that in this case we can apply the states (no complete makina-states installs)

mc_states.modules.mc_controllers.**metadata**()
controllers metadata registry

mc_states.modules.mc_controllers.**registry**()
controllers registry registry

mc_states.modules.mc_controllers.**settings**()
controllers settings registry

mc_localsettings / localsettings variables

mc_states.modules.mc_localsettings.**get_pillar_fqdn**(*sls*, *template*)
if template name is none, it is a directly accessed sls (rendered from a string), here we can guess the name from sls sls does not have '.' it is a directly

mc_states.modules.mc_localsettings.**metadata**()
metadata registry for localsettings

mc_states.modules.mc_localsettings.**registry**(*ttl=900*)
registry registry for localsettings

mc_states.modules.mc_localsettings.**settings**(*ttl=900*)
settings registry for localsettings

OBSOLETE, PLEASE USE LOCAL REGISTRIES MOST MAKINA STATES CORE IS ALREADY MIGRATED

WILL DISAPPEAR IN A NEAR FUTURE

mc_services / servives registries & functions

mc_states.modules.mc_services.**get_service_enabled_state**(*pm=None*, *enable_toggle=None*)
Return if a service should be enabled at boot or not depending on the service function.

if service func is 'running' -> enabled if service func is 'dead' -> disaabled

mc_states.modules.mc_services.**get_service_function**(*pm=None*, *enable_toggle=None*, *has_system_services_manager=None*, *activate_function='service.running'*, *deactivate_function='service.dead'*)

Return the appropriate service function for the activated process manager For the API conveniance, we reflect back any service.function. If pm is system, or manually forced, we want to return the appropriate service function (one of: activate_function/deactivate_function). We take care to check not to return any function when we are in a docker hence we do not have any system level function, nor the access to the service module

pm the processes manager (one of none|forced|system|circus|supervisor|service.{running,dead}) system or forced means to return a function between activate or deactivate

enable_toggle choose if we are in system mode either to return the 'activate' or 'deactivate' function, by default we return the activate function

activate_function salt module function to activate a service (api compatible with service.*)

deactivate_function salt module function to deactivate a service (api compatible with service.*)

has_system_services_manager overrides the makina-states detection of the presence of a system services manager

mc_states.modules.mc_services.**settings**()

Global services registry

RemoteExecution/PaaS/ProjectManagement modules

mc_project / project settings regitry switcher and common functions see *mc_project_2 / project settings regitry APIV2*

mc_states.modules.mc_project.**get_common_vars**(**args*, ***kwargs*)

Retro compat, wrapper to get_configuration

mc_project_2 / project settings regitry APIV2 This is a Corpus Paas reactor, deploy your projects with style, salt style.

This can either:

- Deploy locally a project
- Deploy remotely a project over ssh (the remote host must have makina-states installed)

mc_states.modules.mc_project_2.**clean_salt_git_commit**(*directory*, *commit=True*, ***kw*)

mc_states.modules.mc_project_2.**deploy**(*name*, **args*, ***kwargs*)

Deploy a project

Only run install step:

```
salt-call --local -ldebug mc_project.deploy <name> only=install
```

Run only one or certain install step:

```
    salt-call --local -ldebug mc_project.deploy\
            <name> only=install only_steps=00_foo
    salt-call --local -ldebug mc_project.deploy\
            <name> only=install only_steps=00_foo,02_bar
```

Only run install & fixperms step:

```
    salt-call --local -ldebug mc_project.deploy <n> only=install,fixperms
```

Deploy entirely (this is what is run whithin the git hook):

```
    salt-call --local -ldebug mc_project.deploy <name>
```

Skip a particular step:

```
    salt-call --local -ldebug mc_project.deploy <name>\
            skip_release_sync=True skip_archive=True skip_notify=True
```

mc_states.modules.mc_project_2.**get_configuration**(*name*, **args*, ***kwargs*)

Return a configuration data structure needed data for the project API macros and configurations functions project API 2

name name of the project

remote_less Does the project use local remotes (via git hooks) for users to push code inside remotes and have the local working copy synchronized with those remotes before deploy. Default to False, set to True to not use local remotes If the project directory .git folder exists, and there is no local remote created, the local remotes feature will also be disabled

fqdn fqdn of the box

minion_id minion_id of the box

default_env environnemt to run into (may be dev|prod, better to set a grain see bellow)

project_root where to install the project,

git_root root dir for git repositories

user system project user

groups system project user groups, first group is main

defaults arbitrary data mapping for this project to use in states. It will be accessible throught the get_configuration().data var

env_defaults per environment (eg: prod|dev) specific defaults data to override or merge inside the defaults one

os_defaults per os (eg: Ubuntu/Debian) specific defaults data to override or merge inside the defaults one

only_install Only run the install step (make the others skipped)

skip_archive Skip the archive step

skip_release_sync Skip the release_sync step

skip_install Skip the install phase

skip_rollback Skip the rollback step if any

skip_notify Skip the notify step if any

Any other kwarg is added to the data dict.

Internal variables reference

pillar_root pillar local dir

salt_root salt local dir

archives_root archives directory

data_root persistent data root

project_git_root project local git dir

pillar_git_root pillar local git dir

nodata do not compute data

data The final mapping where all defaults will be mangled. If you want to add extra parameters in the configuration, you d better have to add them to defaults.

force_reload if the project configuration is already present in the context, reload it anyway

sls_includes includes to add to the project top includes statement

no_default_includes Do not add salt_minon & other bases sls like ssh to default includes

rollback FLAG: do we rollback at the end of all processes

You can override the non read only default variables by pillar/grain like:

```
salt grain.setval makina-projects.foo.url 'http://goo/goo.git
salt grain.setval makina-projects.foo.default_env prod
```

You can override the non read only default arbitrary attached defaults by pillar/grain like:

```
/srv/projects/foo/pillar/init.sls:

makina-projects.foo.data.conf_port = 1234
```

mc_states.modules.mc_project_2.**get_configuration_item**(*project*, *item=<object object>*, ***kw*)

Return an item, maybe filtered or all the config

in the form:

```
{key: <itemKey or 'cfg'>, item: VALUE, cfg: <Whole config dict>}
```

CLI examples:

```
salt-call --local                mc_project_2.get_configuration_item eee git_deploy_hook
salt-call --local                mc_project_2.get_configuration_item eee
```

mc_states.modules.mc_project_2.**get_project**(*name*, **args*, ***kwargs*)

Alias of get_configuration for convenience

mc_states.modules.mc_project_2.**init_pillar_dir**(*directory*, *init_data=None*, *user=None*, *group=None*, *project=None*, *commit_all=False*, *do_push=False*, *remote_less=False*, ***kw*)

Initialize a basic versionned pillar directory

directory directory to execute into

user user to exec commands as and for new files

group group for new files

commit_all do we do a final git add/commit

do_push do we do a final push

project project name

init_data configuration options in the corpus format (see get_configuration)

mc_states.modules.mc_project_2.**init_project**(*name*, **args*, ***kwargs*)
See common args to feed the neccessary variables to set a project You will need at least:

- A name
- A type
- The pillar git repository url
- The project & salt git repository url

mc_states.modules.mc_project_2.**init_remote_structure**(*host*, *project*, ***kw*)
Initialize a remote project structure over ssh

CLI Examples:

```
salt-call --local mc_project.init_remote_structure host.fr <project>
```

mc_states.modules.mc_project_2.**init_repo**(*working_copy*, *user=None*, *group=None*, *ret=None*, *bare=True*, *init_salt=False*, *init_pillar=False*, *init_data=None*, *project=None*, *remote_host=None*, *cfg=None*, *remote_less=False*, *api_version='2'*)
Initialize an empty git repository, either bare or a working copy

This can be either:

- a basic corpus-salt base project template
- a pillar repo (containing an init.sls)
- an enmpty directory (containing an empty .empty file just for git repo init)

CLI Examples:

```
salt-call --local mc_project.init_repo /foo
salt-call --local mc_project.init_repo /foo bare=true
salt-call --local mc_project.init_repo /foo init_salt=True
salt-call --local mc_project.init_repo /foo init_pillar=True
salt-call --local mc_project.init_repo /foo bare=true init_salt=True
salt-call --local mc_project.init_repo /foo bare=true init_pillar=True
```

mc_states.modules.mc_project_2.**init_salt_dir**(*directory*, *user=None*, *group=None*, *commit_all=False*, *do_push=False*, *project=None*, *init_data=None*, *remote_less=False*, ***kw*)
Initialize a basic corpus project directory

directory directory to execute into

user user to exec commands as and for new files

group group for new files

commit_all do we do a final git add/commit

do_push do we do a final push

project project name

init_data configuration options in the corpus format (see get_configuration)

mc_states.modules.mc_project_2.**init_ssh_user_keys**(*user*, *failhard=False*, *ret=None*)
Copy root keys from root to a user to allow user to share the same key than root to clone distant repos. This is useful in vms (local PaaS vm)

mc_states.modules.mc_project_2.**install**(*name*, *only_steps=None*, *task_mode=False*, **args*, ***kwargs*)
You can filter steps to run with only_steps All sls in .salt which are not special or tasks (beginning with tasks will be executed)

eg:

```
salt mc_project.deploy only=install onlystep=00_foo,001_bar
salt mc_project.deploy only=install onlystep=00_foo
```

mc_states.modules.mc_project_2.**link**(*names*, **args*, ***kwargs*)
Add the link wired in salt folders (pillar & salt) & register the pillar in pillar top

name list of project(s) separated by commas

mc_states.modules.mc_project_2.**link_into_root**(*name*, *ret*, *link*, *target*, *do_link=True*)
Link a salt managed directory into salt root

This takes care of not leaving a dangling symlink

mc_states.modules.mc_project_2.**link_pillar**(*names*, **args*, ***kwargs*)
Add the link wired in pillar folder & register the pillar in pillar top

name list of project(s) separated by commas

mc_states.modules.mc_project_2.**link_salt**(*names*, **args*, ***kwargs*)
Link a salt managed directory into salt root This takes care of not leaving a dangling symlink

mc_states.modules.mc_project_2.**load_sample**(*cfg*, **args*, ***kwargs*)
Load the project PILLAR.sample back to the cfg configuration dict

mc_states.modules.mc_project_2.**orchestrate**(*host*, *project*, *init=True*, *init_project=None*, *init_pillar=None*, *init_remote=True*, *sync=True*, *sync_project=None*, *sync_pillar=None*, *refresh=True*, *refresh_pillar=None*, *refresh_project=None*, *deploy=True*, *pillar_origin=None*, *pillar_rev=None*, *origin=None*, *rev=None*, *pre_init_hook=None*, *post_init_hook=None*, *pre_init_remote_hook=None*, *post_init_remote_hook=None*, *pre_sync_hook=None*, *deploy_fun='mc_project.remote_deploy'*, *deploy_kwarg=None*, *deploy_args=None*, *post_sync_hook=None*, *pre_hook=None*, *post_hook=None*, *only=None*, *only_steps=None*, ***kw*)
Orchestrate a project deployment, remotely

Note:

a project is composed by it's code & deployment recipe (.salt) and it's pillar.

This:

•Run the pre init hook if any

•initiliazes & prepare the code locally

- •Run the post init hook if any
- •Run the pre init remote hook if any
- •May initiliazes the remote project structure
- •Run the post init remote hook if any
- •Run the pre sync hook if any
- •Sync the code to remote buffer directories
- •Sync the remote deployment directories (pillar & project)
- •Run the post sync hook if any
- •Run the pre deploy hook if any
- •Run the deployment procedure (salt-call mc_project.deploy dance)
- •Run the post deploy hook if any

The deployed code will at first be initialized:

project code is initialized as either:

- from an empty structure (shell helloworld)
- from **git**:

  origin git url

  rev git remote tag/branch/changeset
- from an empty structure (shell helloworld)

pillar code is initialized as either:

- from an empty structure (shell helloworld)
- from **git**:

  pillar_origin git url

  pillar_rev git remote tag/branch/changeset

If the urls were not specified, but the git repositories present in the local directories are valid, they will be deployed as-is.

If the git repositories have a valid remote, the sync step will use it, even if the "origin/pillar_origin" were not specified explicitly

host host where to deploy

project project to deploy onto (name)

init/init_project/init_pillar do we do the full init step do we do the init_project step (overrides init). do we do the init_pillar step (overrides init).

init_remote do we do the init_remote step

origin/pillar_origin url of the pillar to deploy if any (if None: empty pillar)

rev/pillar_rev changeset if the project/pillar is from a git url (master)

sync/sync_project/sync_pillar do we do the full sync step. do we do the sync_project step (overrides sync). do we do the sync_pillar step (overrides sync)

refresh/refresh_project/refresh_pillar do we update the code prior to sync. do we do the refresh_project step (overrides refresh_project). do we do the refresh_project step (overrides refresh)

deploy do we do the deploy step

deploy_args any extra deploy/run_task positional argument

deploy_kwarg any extra deploy/run_task extra argument

pre_init_hook/post_init_hook/ pre_init_remote_hook/post_init_remote_hook/ pre_sync_hook/post_sync_hook/ pre_hook/
deployment hook (see above lifecycle explaination & hook spec)

only/only_steps mc_project.deploy deploy limits arguments(if any)

An hook is a salt function, in any module with the following signature:

```
def hook(host, project, opts, **kw)::
    print("hourray")
```

Making in a execution module, /srv/salt/_module/foo.py:

```
def hook(host, project, opts, **kw)::
    print("Called on {0}".format(host))
```

Can be called like this:

```
salt-call --local mc_project.orchestrate h proj init_hook=foo.hook
```

CLI Examples:

```
salt-call --local mc_project.orchestrate host.fr <project>\
        origin="https://github.com/makinacorpus/corpus-pgsql.git"
salt-call --local mc_project.orchestrate host.fr <project>\
        origin="https://github.com/makinacorpus/corpus-pgsql.git"\
        rev=stable\
        pillar_origin="https://github.com/mak/corpus-pillar.git"\
        rev=stable
salt-call --local mc_project.orchestrate host.fr <project>\
        origin="https://github.com/makinacorpus/corpus-pgsql.git"
salt-call --local mc_project.orchestrate host.fr <project>\
        deploy_args=['a'] \
        origin="https://github.com/makinacorpus/corpus-pgsql.git"
salt-call --local mc_project.orchestrate host.fr <project>\
        deploy_kwarg={'myparam': 'a'} \
        origin="https://github.com/makinacorpus/corpus-pgsql.git"
```

mc_states.modules.mc_project_2.**orchestrate_task**(*host*, *project*, *task*, *init=True*, *init_project=None*, *init_pillar=None*, *init_remote=True*, *sync=True*, *sync_project=None*, *sync_pillar=None*, *refresh=True*, *refresh_pillar=None*, *refresh_project=None*, *deploy=True*, *deploy_args=None*, *deploy_kwarg=None*, *pillar_origin=None*, *pillar_rev=None*, *origin=None*, *rev=None*, *pre_init_hook=None*, *post_init_hook=None*, *pre_init_remote_hook=None*, *post_init_remote_hook=None*, *pre_sync_hook=None*, *post_sync_hook=None*, *pre_hook=None*, *post_hook=None*, *only=None*, *only_steps=None*, ***kw*)

Orchestrate a project through mc_project._orchestrate and use mc_project.remote_task <TASK NAME>

CLI Examples:

```
salt-call --local mc_project.orchestrate_task host.fr <project> task_make_users \
        origin="https://github.com/makinacorpus/corpus-pgsql.git"
salt-call --local mc_project.orchestrate host.fr <project> make_users\
        origin="https://github.com/makinacorpus/corpus-pgsql.git" task_make_users\
        rev=stable\
        pillar_origin="https://github.com/mak/corpus-pillar.git" task_make_users\
        rev=stable
salt-call --local mc_project.orchestrate host.fr <project> task_make_users\
        origin="https://github.com/makinacorpus/corpus-pgsql.git"
salt-call --local mc_project.orchestrate host.fr <project> task_make_users\
        deploy_args=['a'] \
        origin="https://github.com/makinacorpus/corpus-pgsql.git"
salt-call --local mc_project.orchestrate host.fr <project> task_make_users\
        deploy_kwarg={'myparam': 'a'} \
        origin="https://github.com/makinacorpus/corpus-pgsql.git"
```

mc_states.modules.mc_project_2.**push_changesets_in**(*directory*, *remote='origin'*, *branch='master:master'*, *opts=''*, ***kw*)

Thin wrapper to git push

directory directory where to act on

user user to push as

branch branch part of the git command

remote remote to push to

opts 'origin', opts to give

CLI Examples:

```
salt-call --local  mc_project.push_changesets_in /foo
salt-call --local  mc_project.push_changesets_in /foo opts="-f"
salt-call --local  mc_project.push_changesets_in /foo opts="-f"
```

mc_states.modules.mc_project_2.**remote_deploy**(*host*, *project*, **args*, ***kw*)
Run a deployment task, remotely

host host to deploy onto

project project to deploy

deploy_kwarg deploy kwargs

deploy_args deploy arguments

deploy_only_steps/only_steps set only_steps kwarg (shortcut)

deploy_only/only set only kwarg (shortcut)

salt_function salt deploy function (one of: mc_project.deploy/mc_project.run_task)

CLI Examples:

```
salt-call --local mc_project.remote_deploy                host.fr <project>
salt-call --local mc_project.remote_deploy                host.fr <project> only=install,fixper
salt-call --local mc_project.remote_deploy                host.fr <project> only=install,fixper
```

mc_states.modules.mc_project_2.**remote_project_hook**(*hook*, *host*, *project*, *opts*, ***kw*)
Execute an hook

An hook is a salt function, in any module with the following signature:

```
def hook(host, project, opts, **kw)::
    print("hourray")
```

mc_states.modules.mc_project_2.**remote_run_task**(*host*, *project*, *task=None*, **args*, ***kw*)
Run a task from the .salt directory, remotely

Task can be either given as a kwarg or the first positional argument.

CLI Examples:

```
salt-call --local mc_project.remote_run_task                host.fr <project> task=task_hellowo
salt-call --local mc_project.remote_run_task                host.fr <project> task_helloworld.s
salt-call --local mc_project.remote_run_task                host.fr <project> task_helloworld.s
```

mc_states.modules.mc_project_2.**remote_task**(*host*, *project*, **args*, ***kw*)
Alias to remote_run_task

CLI Examples:

```
salt-call --local mc_project.remote_task                host.fr <project> task=task_helloworld.
salt-call --local mc_project.remote_task                host.fr <project> task_helloworld.sls
salt-call --local mc_project.remote_task                host.fr <project> task_helloworld.sls a
```

mc_states.modules.mc_project_2.**remove_path**(*path*)
Remove a path.

mc_states.modules.mc_project_2.**report**()
Get connection details & projects report

CLI Examples:

```
salt-call --local mc_project.report
```

mc_states.modules.mc_project_2.**rollback**(*name*, **args*, ***kwargs*)
Run the rollback corpus step

mc_states.modules.mc_project_2.**rotate_archives**(*name*, **args*, ***kwargs*)
Run the rotate_archives corpus step

mc_states.modules.mc_project_2.**run_task**(*name*, *only_steps*, **args*, ***kwargs*)
Run one or more tasks inside a project context.

You can filter steps to run with only_steps

All sls in `.salt` which are a task (all files beginning with task_ will be searched and the one matching only_steps (string or list) will be executed)

mc_states.modules.mc_project_2.**set_configuration**(*name*, *cfg=None*, **args*, ***kwargs*)
set or update a local (grains) project configuration

mc_states.modules.mc_project_2.**sync_git_directory**(*directory*, *origin=None*, *rev=None*, *sync_remote='sync'*, *refresh=False*, *local_branch=None*, ***kw*)
Agressively sync a git working copy with its remote

directory directory where to act

origin remote origin <url>

rev changeset to deploy

local_branch local branch to set to

sync_remote name of the remote

refresh if the working copy exists, force the git pull dance

CLI Examples:

```
salt-call --local \
        mc_project.sync_git_directory /foo origin=git://foo rev=develop
```

mc_states.modules.mc_project_2.**sync_hooks_for_all**(**args*, ***kwargs*)
Get connection details & projects report

mc_states.modules.mc_project_2.**sync_modules**(*name*, **args*, ***kwargs*)
Install custom modules to the salt modules directory in the global module dir,

Note: we install two symlinks to provide room for overlapping modules eg: project1/project/.salt/_modules/foo.py:bar can be called:

```
salt-call foo.bar()
salt-call foo_foo.bar()
```

mc_states.modules.mc_project_2.**sync_remote_working_copy**(*host*, *directory*, *remote_directory=None*, *remote_local_copy=None*, *lremote=None*, ***kw*)
CLI examples:

```
salt-call --local mc_project.sync_remote_working_copy\
        a.fr /srv/remote-projects/a.fr
salt-call --local mc_project.sync_remote_working_copy\
        a.fr /srv/remote-projects/a.fr /otherdir
```

mc_states.modules.mc_project_2.**sync_working_copy**(*wc*, *rev=None*, *ret=None*, *origin=None*, *reset=False*, ***kw*)
Synchronnze a directory with it's git remote

directory directory to execute into

reset force sync with remote

user user to exec commands as and for new files

rev force rev to reset to

origin origin to sync with

mc_states.modules.mc_project_2.**uncache_project**(*name*)
Uncache a configuration after an initial load either by it's name or by its structure

mc_states.modules.mc_project_2.**unlink**(*names*, **args*, ***kwargs*)
Remove the link wired in salt folders (pillar & salt)

name list of project(s) separated by commas

mc_states.modules.mc_project_2.**unlink_pillar**(*names*, **args*, ***kwargs*)
Remove the link wired in pillar folder & unregister the pillar in pillar top

name list of project(s) separated by commas

mc_states.modules.mc_project_2.**unlink_salt**(*names*, **args*, ***kwargs*)
Remove the link wired in salt folder & unregister the pillar in pillar top

name list of project(s) separated by commas

mc_states.modules.mc_project_2.**working_copy_in_initial_state**(*wc*, ***kw*)
Test if a directory is at the first git commit from this system

wc where to execute

user user to act with

mc_remote / remote execution functions The following functions are related to do remote executions over ssh transport. This for both raw commands and local salt executions. Those functions are just variations from salt.utils.cloud (which i (kiorky) also helped to wrote for some parts, like... the remote exec ones :))

This have nice features like:

- handling interactive passwords
- using ssh gateways
- transfering files using different fallback methods
- running salt-call over remote (even masterless) salt installations

If this module gains salt core, there are some small makina-states deps:

- mc_states.renderers.lyaml
- mc_states.modules.mc_utils.magicstring (which needs chardet)

mc_states.modules.mc_remote.**delete_remote**(*host*, *filepath*, *mode='-f'*, *level='info'*, ***kw*)
Delete a remote file (or directory)

mc_states.modules.mc_remote.**highstate**(*host*, *outputter='json'*, *transformer='highstate'*, *strip_out=True*, ***kw*)
Run an highstate on an host and fails on error kwargs are forwarded to ssh helper functions !

host host to connect onto

ssh_user/user (first win) user to connect as (default: root)

ssh_port/port (first win) Port to connect onto (default: 22)

sls sls to execute

CLI Examples:

```
salt-call --local mc_remote.highstate foo.net
```

mc_states.modules.mc_remote.**interactive_ssh**(*cmd*, ***kw*)

Establish a ssh connection layer, executes a command, interact.

This session can can be password interactive and we will by default a password challenge and forward the result to the user.

The session control behavior can be controller by subclassing the AbstractSshSession class, see 'ssh_interaction_class':

cmd the full ssh command to wrap the execution from

eg:

```
ssh foo.com "ls /"
```

ssh_quote do we force the script to be quoted

ssh_display_ssh_output do we send output on the console

ssh_output_loglevel log level

ssh_password_retries how many retries do we try for the password

ssh_interaction_class _AbstractSshSession subclass(not instance) (_SSHPasswordChallenger by default)

timeout timeout for command execution

mc_states.modules.mc_remote.**local_mastersalt_call**(**a*, ***kw*)

Execute mastersalt-call locally, maybe in another shell see salt-call

mc_states.modules.mc_remote.**local_salt_call**(**a*, ***kw*)

Execute salt-call locally, in another shell see salt-call

mc_states.modules.mc_remote.**mastersalt_call**(**a*, ***kw*)

Execute mastersalt-call remotely see salt-call

mc_states.modules.mc_remote.**run**(*host*, *script*, ***kw*)

Wrapper to ssh but get only stdout

kwargs are forwarded to ssh helper functions !

returning only the code exist status

mc_states.modules.mc_remote.**salt_call**(*host*, *fun=None*, *arg=None*, *kwarg=None*, *outputter='json'*, *transformer=None*, *unparse=True*, *loglevel='info'*, *salt_call_bin='salt-call'*, *masterless=None*, *minion_id='local'*, *salt_call_script=None*, *strip_out=None*, *hard_failure=False*, *remote=None*, *use_vt=None*, *new_shell=None*, *ttl=0*, **args*, ***kwargs*)

Executes a salt_call call remotely via ssh

host host to execute on

ssh_user/user (first win) user to connect as (default: root)

ssh_port/port (first win) Port to connect onto (default: 22)

fun saltcall function to call

arg args for the saltcall function

kwarg kwargs for the saltcall function

outputter outputter for the saltcall return

transformer outputter used to unparse the value returned from the call (the "2nd pass")

unparse unserialise the return and tries to split out the local result from regular salt call

loglevel loglevel to use

vt_loglevel loglevel to use for vt

salt_call_bin binary to run

masterless do we run masterless (–local)

salt_call_script override default salt call wrapper shell script

remote use salt locally (you must set host to None !

new_shell if we execute locally, new_shell can be set to false to execute directly the salt function instead of calling a new shell to call the function, this can be used in conjunction with ttl to cache results easily, eg

```
salt-call --local -lall mc_remote.salt_call fun=test.ping \
   host=127.0.0.1 new_shell=False ttl=60

salt-call --local mc_remote.mastersalt_call fun=mc_cl.settings \
   host=127.0.0.1 new_shell=False ttl=60
```

use_vt When ran locally, use use_vt to stream output

args are appended to arg as arguments to the called salt function

kwargs They are forwarded to ssh helper functions !

ttl Use filecache based execution (the result will be cached for X seconds). If the cache is not expired, the result will be used and the function wont be executed.

CLI Examples:

```
salt-call --local mc_remote.salt_call \
        foo.net test.ping

salt-call --local mc_remote.salt_call \
        foo.net cmd.run \
        'for i in $(seq 5);do echo $i;sleep 1;done' \
        kwarg='{use_vt: True, python_shell: True, user: ubuntu}' \
        ssh_gateway=127.0.0.1 port=40007

salt-call --local mc_remote.salt_call \
        foo.net cmd.run \
        'for i in $(seq 2);do echo $i;sleep 1;done;echo é' \
        kwarg='{use_vt: True, python_shell: True, user: ubuntu}' \
        outputter=yaml

salt-call --local mc_remote.salt_call \
        foo.net cmd.run \
        'for i in $(seq 2);do echo $i;sleep 1;done;echo é' \
        kwarg='{use_vt: True, python_shell: True, user: ubuntu}' \
        unparse=False outputter=yaml
```

mc_states.modules.mc_remote.**sls_**(*host*, *sls*, *outputter='json'*, *transformer='highstate'*, *strip_out=True*, ***kw*)

Run a state file on an host and fails on error kwargs are forwarded to ssh helper functions !

host host to connect onto

ssh_user/user (first win) user to connect as (default: root)

ssh_port/port (first win) Port to connect onto (default: 22)

sls sls to execute

CLI Examples:

```
salt-call --local mc_remote.sls foo.net mysls
```

mc_states.modules.mc_remote.**ssh**(*host*, *script*, ***kwargs*)

Executes a script command remotly via ssh Attention, if you use a gateway, only key auth is possible on the gw Please also look ssh_kwargs

host host to execute the script on

ssh_user/user (first win) user to connect as (default: root)

ssh_port/port (first win) Port to connect onto (default: 22)

tmpdir tempfile to upload script to (default to /tmp, this mountpoint must not have the -noexec mount flag)

script script or command to execute:

- if the script contains multiple lines We put the content in a temporary file, as-is before uploading it
- If the script is a filepath we upload it as-is
- In other cases we wrap it in a simple shell wrapper before uploading

vt_loglevel loglevel to use for vt

Even in case of a command, it will be wrapped before execution to ease shell quoting

Any extra keywords parameters will by forwarded to:

_get_ssh_args (see doc) to mangle connection details

interactive_ssh(& ssh_interaction_class) (see doc) to interact during ssh session

CLI Examples:

```
salt-call mc_remote.ssh foo.net ssh_gateway=127.0.0.1 port=40007 \
        "cat /etc/hostname" user=mytest password=secret

salt-call mc_remote.ssh foo.net \
        "cat /etc/hostname"
```

mc_states.modules.mc_remote.**ssh_kwargs**(*first_argument_kwargs=None*, ***kw*)

Lookup & sanitize input in kwargs to have only one value for various & well known SSH connection parameters & other related to mc_remote.

The resulting structure is an well known interface dict usable and used in all this module functions.

All kwargs will be lookup in the form ssh_param and then param.

Eg: ssh_user, if no value, user, if not value, default.

This supports for now:

user root

tty use ssh -t (true)

port 22

key_filename ~/id_{rsa,dsa} if existing else ~/.ssh/id_rsa

gateway_key ~/id_{rsa,dsa} if existing else ~/.ssh/id_rsa

gateway gateway host/ip if any

gateway_user root

gateway_port 22

password clear password if any (None)

password_retries how many do we retry a failed password (3)

known_hosts_file known_hosts_file if any (/dev/null)

host_key_checking toggle host checking (no)

makedirs When transfering files or the shell wrapper, are we allowed to create the conttainer(s) (false)

quote quote commands (false)

progress display ssh transfer stats (false)

show_running_cmd flag for extra logs (true)

no_error_log Do not report execution failure in logs

display_ssh_output flag to stream output streams (true)

mc_states.modules.mc_remote.**ssh_retcode**(*host*, *script*, ***kw*)

Wrapper to ssh

kwargs are forwarded to ssh helper functions !

returning only the code exist status

mc_states.modules.mc_remote.**ssh_transfer_dir**(*host*, *orig*, *dest=None*, ***kwargs*)

Transfer directories to an host via ssh layer Please also look ssh_kwargs

This will try then fallback on next transfer method. In order we try:

- rsync
- scp
- sftp

host host to tranfer to

ssh_user/user (first win) user to connect as

ssh_port/port (first win) Port to connect onto

orig filepath to transfer

dest where to upload, defaults to orig

makedirs create parents if any

vt_loglevel vt loglevel

progress activate transfer progress

Any extra keywords parameters will by forwarded to:

_get_ssh_args (see doc) to mangle connection details

interactive_ssh(& ssh_interaction_class) (see doc) to interact during ssh session

mc_states.modules.mc_remote.**ssh_transfer_file**(*host*, *orig*, *dest=None*, ***kwargs*)

Transfer files to an host via ssh layer Please also look ssh_kwargs

This will try then fallback on next transfer method. In order we try:

- rsync
- scp
- sftp
- gzip piped to dest host gunzip
- cat piped to dest host uncat

host host to tranfer to

ssh_user/user (first win) user to connect as

ssh_port/port (first win) Port to connect onto

orig filepath to transfer

dest where to upload, defaults to orig

makedirs create parents if any

vt_loglevel vt loglevel

progress activate transfers statsv

display_content_on_error show script on error

Any extra keywords parameters will by forwarded to:

_get_ssh_args (see doc) to mangle connection details

interactive_ssh(& ssh_interaction_class) (see doc) to interact during ssh session

mc_states.modules.mc_remote.**yamldump_arg**(*arg*, *default_flow_style=True*, *line_break='\n'*, *strip=True*)

yaml.safe_dump the arg This is the counterpart of salt.utils.args.yamlify_arg

mc_dnshelpers The functions else of settings Must be executed on dns master side

This needs those extra pillar settings to configure mc_provider (api settings)

mc_states.modules.mc_dnshelpers.**register_dns_masters**(*only_domains=None*, *only_providers=None*)

Use registrar apis to switch the nameservers to the ones we manage on mastersalt

only_domains list of domains to act on, if empty all managed domains will be checked

only_providers limit action to one or more providers (gandi, ovh)

CLI Examples:

```
mastersalt-call mc_dns.register_dns_masters only_providers=ovh
mastersalt-call mc_dns.register_dns_masters only_providers=gandi
mastersalt-call mc_dns.register_dns_masters foo.net
```

mc_states.modules.mc_dnshelpers.**settings**(*ttl=900*)

Share the mc_dns.settings for shortness of configuration items

mc_djutils / django helpers

mc_states.modules.mc_djutils.**backup_database**(*db_host=None*, *db_name=None*, *cfg='project'*)

backup database before some critical operations

mc_states.modules.mc_djutils.**restore_from**(*db_host=None*, *db_user=None*, *db_pass=None*, *db_name=None*, *media=None*, *dump=None*, *orig_db_host=None*, *orig_db_name=None*, *orig_media=None*, *orig_media_host=None*, *skip_setup=False*, *skip_db=False*, *skip_media=False*, *dev=None*, *drop=True*, *post_hook=None*, *cfg='project'*)

Restore local django from a distant one Connect to host, get the pg backup connect to another host, load the pg backup sync medias run a post hook if any

cfg name of the makina-states project to get configuration from

mc_states.modules.mc_djutils.**setup_database**(*db_host=None*, *db_user=None*, *db_pass=None*, *db_name=None*, *drop=False*, *cfg='project'*)

Connect to a host with ssh

create a pg user <maybe drop the db> create a db owned by the user

Subregistries

mc_dns / dns settings The main settings in there is "default_dnses" which are the dns for resolution As soon as you install a dns service on the behalf of makina-states

mc_mvn / mvn registry

mc_states.modules.mc_mvn.**settings**()

mvn

mc_firewalld / firewalld functions

mc_states.modules.mc_firewalld.**add_aliased_interfaces**(*data=None*)

Mark aliases of interfaces to belong to the same interface of the attached interface, by default

mc_states.modules.mc_firewalld.**add_natted_networks**(*data=None*)

Add nat rules if possible on non public zones

mc_states.modules.mc_firewalld.**complete_rich_rules**(*rules=None*, *rule=None*, *family=None*, *destinations=None*, *icmp_block=None*, *masquerade=None*, *sources=None*, *protocols=None*, *ports=None*, *forward_ports=None*, *services=None*, *enrule=None*, *audit=None*, *log=None*, *log_prefix=None*, *log_level=None*, *limit=None*, *action=None*)

Subroutine of the rich rule helper

mc_states.modules.mc_firewalld.**rich_rules**(*family='ipv4'*, *sources=None*, *source=None*, *destinations=None*, *destination=None*, *services=None*, *service=None*, *ports=None*, *port=None*, *audit=None*, *log=None*, *log_prefix=None*, *log_level=None*, *forward_ports=None*, *forward_port=None*, *limit=None*, *icmp_block=None*, *masquerade=False*, *action='accept'*, *public_ips=None*, *protocols=None*)

Helper to generate rich rules compatibles with firewalld

see firewalld.richlanguage(5) (man)

public_ips firewalld does make special forward rules relying on packet marking whenever they macth the port and the source/dest. This means that it will remap all traffic from an aforementioned rule matching this port on the public zone to be reentrant and goes to the destination of the rule instead of the real destination. Thus to correctly NAT services without limiting outgoing traffic on the same ports from network branches withing the NAT, we limit the exposure of the target rules to the public facing ips of the underlying host. This also means, that logically, as nearly always with natted services, network access points from within the NAT cant access the services as if they would be outside of the NAT. Technically speaking, in case of forward ports, we only apply the rule on the public facing address (aka addresses of interfaces which are linked to the public zones) by limitating the rule scopes to those destinations.

public_ips is indeed a list of ips, which if empty or None will be computed from host network informations.

To disable destinations restrictions, you can set public_ips to False.

```
rich_rules(forward_port={'port': '12', addr='1.2.3.4'}, port='22')
rich_rules(forward_port={'port': '12', addr='1.2.3.4'}, port='22'},
           destination='x.x.x.x')
rich_rules(masquerade=True, source='x.x.x.x', dest='x.x.x.x')
rich_rules(source='address="1.2.3.4"', port='22', action='drop')
rich_rules(destination='address="1.2.3.4"',
           port='22',
           audit=True,
           action='drop')
rich_rules(destination='address="1.2.3.4"',
           port={'port': '22', 'protocol': 'tcp'})
rich_rules(destination='address="1.2.3.4"', port='22',
           audit=True, action='drop')
```

mc_states.modules.mc_firewalld.**search_aliased_interfaces**(*data=None*)

Add public interfaces as candidates for aliased zones to support common IP Failover scenarii

mc_states.modules.mc_firewalld.**settings**()

firewalld settings

makina-states.services.firewalld.enabled set to true to activate firewalld

DESIGN all services & forwardport & ports & etc are setted via rich rules to allow fine-graines selections of source and destination variations.

GLOBAL SETTINGS

permissive_mode force all traffic to be accepted

allow_local force all traffic from rfc1918 to be accepted

public_interfaces internet faced interfaces

internal_interfaces interfaces wired to internal network with no much restriction

public_services services to allow

restricted_services services to block

<XXX>-direct direct rules

<XXX>-passthrough direct/passthrough rules (not implemented yet)

services list of services to deine

zones mapping of zones definitions

PER ZONE SETTINGS

You can configure zone settings via via entries in the zone pillar

default_policy enforce policy, attention in firewalld world, everything is dropped if no match, so no need to force reject. Its even harmful as it wont cut any further rich rules to have a change to apply !

interfaces interfaces to add to the zone

XXX-rules rich rules

For exmeple, to Add some rich rules in pillar to a zone, all `makina-states.services.firewall.firewalld.zones.public.rules<id>` are merged

```
makina-states.services.firewall.firewalld.zones.public.rules-foo:
  {% for i in salt['mc_firewalld.rich_rules'](
    port=22, action='drop'
  )- {{i}} {% endfor %}
  {% for i in salt['mc_firewalld.rich_rules'](
    forward_port={'port': 1122, 'to_addr': '1.2.3.4', 'to_port'=22}
  )- {{i}} {% endfor %}
makina-states.services.firewall.firewalld.zones.public.rules-bar:
  - "rule service name="ftp" log limit value="1/m" audit accept"
  {% for i in salt['mc_firewalld.rich_rules'](
    port=22, destinations=['127.0.0.1'],  action='drop'
  )- {{i}} {% endfor %}
  {% for i in salt['mc_firewalld.rich_rules'](
      port=22, destinations=['not address="127.0.0.2"'],  action='drop'
  )- {{i}} {% endfor %}
```

Whitelist some services:

```
    makina-states.services.firewall.firewalld.public_services-append:
        - smtp
```

Change whitelisted services:

```
    makina-states.services.firewall.firewalld.public_services: [http]
```

Define a new service:

```
    makina-states.services.firewall.firewalld.services.foo:
        port: [{protocol: tcp, port: 2222}]
```

NOTE

DO NOT ACTIVATE MASQUERADING, IT IS TOO MUCH CATCHY PLEASE USE APPROPRIATE RESTRICTIVES RICH MASQUERADE RULES

mc_dbus / dbus functions
mc_states.modules.mc_dbus.**settings**()
dbus settings

mc_apparmor / apparmor functions
mc_states.modules.mc_apparmor.**settings**()
apparmor settings

mc_apache / apache httpd functions If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_apache
```

Check the mc_apache states for details.

Do not forget to sync salt cache:

```
salt-call state.sls makina-states.controllers.salt
```

mc_states.modules.mc_apache.**a2dismod**(*module*)
Runs a2dismod for the given module.

This will only be functional on Debian-based operating systems (Ubuntu, Mint, etc).

module string, module name

CLI Examples:

```
salt '*' mc_apache.a2dismod autoindex
```

mc_states.modules.mc_apache.**a2enmod**(*module*)
Runs a2enmod for the given module.

This will only be functional on Debian-based operating systems (Ubuntu, Mint, etc).

module string, module name

CLI Examples:

```
salt '*' mc_apache.a2enmod autoindex
```

mc_states.modules.mc_apache.**check_version**(*version*)
Ensures the installed apache version matches all the given version number

For a given 2.2 version 2.2.x current version would return an OK state

version The apache version

CLI Examples:

```
salt '*' mc_apache.check_version 2.4
```

mc_states.modules.mc_apache.**get_version**()
Ensures the installed apache version matches all the given version number

For a given 2.2 version 2.2.x current version would return an OK state

version The apache version

CLI Examples:

```
salt '*' mc_apache.get_version
```

mc_states.modules.mc_apache.**settings**()

Registry for apache related settings

heses settings are loaded from defaults + grains + pillar. pache Fine Settings

httpd_user apache system user

mpm mpm to use

mpm-packages system related packaged to install the desired mpm

version targeted apache version to switch the configuration for

log_level httpd log level

fastcgi_params mappings of specific params for mod_fastcgi Please look the module code and the apache documentation if you are not happy with defaults

fastcgi_enabled internal setting

fastcgi_socket_directory internal setting

serveradmin_mail default server admin email

Timeout The number of seconds before receives and sends time out. default is 300 (5min), 1 or 2 min should be enough for any client request (so 60 or 120). beware of DOS!

KeepAlive bool: are KeepAlive requests allowed

MaxKeepAliveRequests: maximum number of allowed KeepAlive requests (compare with MaxClients)

KeepAliveTimeout: How many seconds should we keep Keepalive conn open (say something between 3 and 5 usually, be careful for DOS!)

log_level

log level, allowed values are debug, info, notice, warn, error, crit, alert, emerg

serveradmin_mail default webmaster mail (used on error pages)

mpm prefork

StartServers number of server processes to start

MinSpareServers minimum number of server processes which are kept spare

MaxSpareServers &maximum number of server processes which are kept spare

MaxRequestsPerChild maximum number of requests a server process serves set 0 to disable process recylcing

MaxClients (alias MaxRequestWorkers): maximum number of server processes allowed to start

mpm worker

StartServers initial number of server processes to start

MinSpareThreads minimum number of worker threads which are kept spare

MaxSpareThreads maximum number of worker threads which are kept spare

ThreadLimit ThreadsPerChild can be changed to this maximum value during a graceful restart. ThreadLimit can only be changed by stopping and starting Apache.

ThreadsPerChild constant number of worker threads in each server process

MaxRequestsPerChild

(alias MaxConnectionsPerChild): maximum number of requests a server process serves set 0 to disable process recylcing

MaxClients (alias MaxRequestWorkers): maximum number of threads

mpm event all workers settings are used

AsyncRequestWorkerFactor max of concurrent conn is:

```
(AsyncRequestWorkerFactor + 1) * MaxRequestWorkers
```

mc_states.modules.mc_apache.**vhost_settings**(*domain*, *doc_root*, ***kwargs*)

Used by apache macro

vh_top_source source (jinja) of the file.managed for the virtualhost template. (empty by default) this will be included at the global conf level

vh_template_source source (jinja) of the file.managed for the virtualhost template. this will be the vhost definitions which in turn include the vhost defs

vh_content_source source (jinja) of the file.managed for the virtualhost template. this will be included at the vhost level

serveradmin_mail data that may be used on error page default is webmaster@<site-name>

number Virtualhost priority number (for apache), without a default VH the first one became the default virtualhost

redirect_aliases True by default, make a special Virtualhost with all server_aliases, all redirecting with a 301 to the site name, better for SEO. But you may need real server_aliases for static parallel file servers, for example, then set that to True.

allow_htaccess False by default, if your project use .htaccess files, then prey for your soul, eat some shit, kill yourself and set that to True

vhost_basename: basename of file in /etc/apache2/sites-{enabled,available}

log_level log level

ssl_interface/interface interface of the namevirtualhost (like in "*:80"), default is "*"

ssl_port/port port of the namevirtualhost (like in "*:80"), default is "80" and "443" for ssl version

mc_autoupgrade / packages autoupgrade

mc_states.modules.mc_autoupgrade.**settings**()

autoupgrade registry

mc_bind / named/bind functions For the documentation on usage, please look *BIND/NAMED integration*.

mc_states.modules.mc_bind.**cached_zone_headers**()

Store a cached but much small memory footprint version of all zones data for quickier access to construct views in main bind configuration. Those keys are enabled:

- views
- server_type
- fqdn
- masters
- slaves

- fpath
- template
- source

mc_states.modules.mc_bind.**get_view**(*view*)

Get the mapping describing a bind view

zones light mapping containing zone headers to feed the views configuration file. See cached_zone_headers

match_clients [any]

recursion no

additional_from_auth no

additional_from_cache no

mc_states.modules.mc_bind.**get_zone**(*zone*)

Get the mapping describing a bind zone

views list of views to enable this zone in

serial zone serial

server_type one of master/slave

ttl TTL of soa

fqdn zone FQDN

soa_ns zone main nameserver (ns.{fqdn})

soa_contact soa contact (sysadmin.{fqdn})

refresh refresh(300)

retry retry(60)

expire expire ()

minimum minimum (300)

notify notify (true in mastermode and false if slave)

rrs records for the zone in mastermode. This list list of records of the zone is in bind syntax

slaves list of slaves to allow transfer to in master mode

masters list of master to get zones from in slave mode

allow_transfer list of transfer items

allow_query list of query items

allow_update list of update items

mc_states.modules.mc_bind.**settings**()

Named settings

Without further configuration, this will setup a caching name server. With a little effort, you can easily turn this server in a powerful and flexible nameserver.

For the documentation on usage, please look *BIND/NAMED integration*.

pkgs pkg to install for a named install

config master config file path

local_config local master config file path

options_config options config file path

default_zones_config default zone config file path

dnssec do we use dnssec (not implemented now)

named_directory var directory

user user for named service (root)

group group for named service (named)

service_name service name

mode configuration files mode ('640')

views List of managed view names

zones List of managed zones names

serial 2014030501

slaves default dns server slaves if any

ttl 300

refresh 300

retry 60

expire 2419200

minimum 299

rndc_conf path to rndc configuration

rndc_key path to rndc key

servers_config_template salt://makina-states/files/etc/bind/named.conf.servers

key_config_template {{settingsnsalt://makina-states/files/etc/bind/named.conf.key

bind_config_template salt://makina-states/files/etc/bind/named.conf

local_config_template salt://makina-states/files/etc/bind/named.conf.local

options_config_template salt://makina-states/files/etc/bind/named.conf.options'

logging_zones_config_template salt://makina-states/files/etc/bind/named.conf.logging

default_zones_config_template salt://makina-states/files/etc/bind/named.conf.default-zones

zone_template salt://makina-states/files/etc/bind/pri_zone.zone

loglevel

default error

general error

database error

config error

security error

resolver error

xfer_in nfo

xfer_out info

notify error

client error

unmatched error

queries error

network error

update info

dispatch error

lame_servers error

mc_burp / burp functions burp settings

mc_states.modules.mc_burp.**settings**()

burp settings server generates its ca server generate locally the client configuration and push it via rsync to clients

prefix: makina-states.services.backups.burp

Server opts (prefix: makina-states.services.backups.burp.server_conf)

- fqdn
- port
- status_port
- client_port
- client_status_port
- directory
- clientconfdir
- pidfile
- hardlinked_archive
- working_dir_recovery_method
- max_children
- max_status_children
- umask
- syslog
- stdout
- client_can_delete
- client_can_force_backup
- client_can_list
- client_can_restore
- client_can_verify
- version_warn

- •keep
- •timer_script
- •timer_arg
- •ca_conf
- •ca_name
- •ca_server_name
- •ca_burp_ca
- •ssl_cert
- •ssl_key
- •ssl_key_password
- •ssl_dhfile
- •notify_failure_script
- •notify_failure_arg
- •server_script_pre
- •server_script_pre_arg
- •server_script_pre_notify
- •server_script_post
- •server_script_post_arg
- •server_script_post_run_on_fail
- •server_script_post_notify
- •restore_client

client opts (client_common and client.<fqdn>)

- (prefix: makina-states.services.backups.burp.client_common)
- (prefix: makina-states.services.backups.burp.clients.<fqdn>)
- dedup_group
- mode
- port
- pidfile
- syslog
- stdout
- progress_counter
- ratelimit
- network_timeout
- autoupgrade_dir
- autoupgrade_os
- server_can_restore

- cross_filesystem
- cross_all_filesystems
- encryption_password
- ca_burp_ca
- ca_csr_dir
- ssl_cert
- ssl_key
- ssl_ciphers
- backup_script_pre
- backup_script_post
- restore_script_pre
- restore_script_post
- include
- exclude
- exclude_ext
- exclude_regex
- exclude_fs
- min_file_size
- max_file_size
- nobackup
- read_fifo
- read_all_fifos
- read_blockdev
- read_all_blockdevs
- exclude_comp
- cron_periodicity: automaticly spray all around the hour
- cron_cmd
- restore_client
- ssh_port (default 22)
- ssh_user (default root)

For each client you can define a ssh gateway (eg: for VMs with a private ip)

Extra params are:

- ssh_gateway: ip[:port] (default: 22)
- ssh_gateway_user
- ssh_gateway_key

mc_casperjs / casperjs/npm registry

mc_states.modules.mc_casperjs.**settings**()

casperjs

mc_circus / circus functions

mc_states.modules.mc_circus.**settings**()

circus settings

location installation directory

mc_cloud_compute_node / cloud compute node related functions

mc_states.modules.mc_cloud_compute_node.**cleanup_allocated_ips**(*target*)

Maintenance routine to cleanup ips when ip exhaution arrises

mc_states.modules.mc_cloud_compute_node.**cleanup_ports_mapping**(*target*, *kind='ssh'*)

This is a maintenance routine which can be called to cleanup ports when range exhaustion is incoming

mc_states.modules.mc_cloud_compute_node.**default_settings**()

Default compute node settings

target target minion id

expose/expose_limited expose configuration to other nodes, see mc_cloud.ext_pillar

all_vms contain compute node vms on compute node contain all vms on controller A mapping indexed by vm minion ids and containing some info:

```
{vm name: virt type}
```

all_targets a mapping indexed by target minions ids containing either only the compute node on compute node all the compute nodes on controller

vms vms on compute node or empty dict

targets a mapping indexed by target minions ids containing also vms and supported vt

reverse_proxies mapping of reverse proxies info

domains list of domains served by host

vts a list of supported virt types (lxc)

has global configuration toggle

firewall global firewall toggle

port_range_start from where we start to enable ssh NAT ports. Default to 40000.

port_range_end

from where we end to enable ssh NAT ports. Default to 60000.

Basically the compute node needs to:

- setup reverse proxying
- setup it's local internal addressing dns to point to private ips
- everything else that's local to the compute node

The computes nodes are often created implicitly by registration of vms on specific drivers like LXC but you can get_cloud_confr some manually.

makina-states.cloud.compute_node.settings.targets.devhost11.local: {}

To add or modify a value, use the mc_utils.default habitual way of modifying the default dict.

mc_states.modules.mc_cloud_compute_node.**del_conf_for_target**(*target*, *setting*)
Register a specific setting for a specific target

mc_states.modules.mc_cloud_compute_node.**ext_pillar**(*id_*, *prefixed=True*, *ttl=32140800*, **args*, ***kw*)
compute node extpillar

mc_states.modules.mc_cloud_compute_node.**feed_http_reverse_proxy_for_target**(*target_data*)
Get reverse proxy information mapping for a specicific target This return a useful mappings of infos to reverse proxy http and ssh services with haproxy

mc_states.modules.mc_cloud_compute_node.**feed_network_mappings_for_target**(*target_data*, *kinds=None*)
Network mappings are in the form:

This is the form of the APPCONTAINER SPEC mixin ACI/ACE:

```
[
    {
        'name': 'default',
        'hostPort': <int>,
        'to_addr': <Local IPV4 Address of vm>,
        'port': <int>,
        'count': <int> (opt),
        'protocol': 'tcp' / 'udp'
    }
]
```

mc_states.modules.mc_cloud_compute_node.**find_ip_for_vm**(*vm*, *default=None*, *network='10.5.0.0'*, *netmask='16'*, *target=None*)
Search for:

- an ip already allocated
- an random available ip in the range

To get and maybe allocate an ip for a vm call

find_ip_for_vm(target, vmname)

For force/set an ip use:

```
set_allocated_ip(target, vmname, '1.2.3.4')
```

mc_states.modules.mc_cloud_compute_node.**find_mac_for_vm**(*vm*, *default=None*, *target=None*)
Generate and assign a mac addess to a specific vm on a specific host

mc_states.modules.mc_cloud_compute_node.**find_password_for_vm**(*vm*, *default=None*, *pwlen=32*, *target=None*)
Return the vm password after creating it the first time

mc_states.modules.mc_cloud_compute_node.**get_all_vts**(*supported=None*)
Get makina-states.cloud VTS

- supported=None: all
- supported=True: supported vt
- supported=False: unsupported vt

mc_states.modules.mc_cloud_compute_node.**get_allocated_ips**(*target*)
Get the allocated ips for a specific target

mc_states.modules.mc_cloud_compute_node.**get_conf_for_target**(*target*, *setting*, *default=None*)
get the stored specific setting for a specific target

mc_states.modules.mc_cloud_compute_node.**get_conf_for_vm**(*vm*, *setting*, *default=None*, *target=None*)
.

mc_states.modules.mc_cloud_compute_node.**get_targets**(*vt=None*, *ttl=32140800*)
Return all vms indexed by targets

mc_states.modules.mc_cloud_compute_node.**get_vms**(*vt=None*, *vm=None*, *ttl=32140800*)
Returns vms indexed by name

mc_states.modules.mc_cloud_compute_node.**get_vms_for_target**(*target*, *vt=None*)
Return all vms for a target

mc_states.modules.mc_cloud_compute_node.**get_vms_per_type**(*target*)
Return all vms indexed by vt for a special target

mc_states.modules.mc_cloud_compute_node.**get_vts**(*supported=True*)
Alias to get_all_vts At the difference of supported is True by default

mc_states.modules.mc_cloud_compute_node.**remove_allocated_ip**(*target=None*, *ip=None*, *vm=None*)
Remove any ip from the allocated IP registry. If vm is specified, it must also match a vm which is allocated to this ip.

target target to act onto (opt if vm is given)

ip specifically specify the ip to delete

vm if given, delete any ip belonging to this vm name

mc_states.modules.mc_cloud_compute_node.**set_allocated_ip**(*vm*, *ip*, *target=None*)
Allocate an ip for a vm on a compute node for a specific vt

```
>>> set_allocated_ip('foo.bar.net', 'mybm.bar.net', '2.2.3.4')
```

mc_states.modules.mc_cloud_compute_node.**set_conf_for_target**(*target*, *setting*, *value*)
Register a specific setting for a specific target

mc_states.modules.mc_cloud_compute_node.**settings**(*ttl=120*)
compute node related settings

mc_states.modules.mc_cloud_compute_node.**target_for_vm**(*vm*, *target=None*, *ttl=32140800*)
Get target for a vm

mc_states.modules.mc_cloud_compute_node.**vt_for_vm**(*vm*, *target=None*)
Get VT for a vm

mc_cloud_controller / cloud related variables

- This contains generate settings around salt_cloud
- This contains also all targets to be driven using the saltify driver

- LXC driver profile and containers settings are in *mc_cloud_lxc / lxc registry for compute nodes*.

mc_states.modules.mc_cloud_controller.**settings**(*ttl=60*)
compute node related settings

mc_cloud_images / cloud images release managment & delivery Please also have a look at the runner.

mc_states.modules.mc_cloud_images.**build_docker_from_rootfs**(*image*, *rootfs*, *force=False*, ***kwargs*)
All kwargs are forwarded to mc_cloud_images.rebuild_ms_docker

image image to build

rootfs lxc ROOTFS to package as a docker container

force if lxc tarfile is present, redo it anyway

mc_states.modules.mc_cloud_images.**default_settings**()
cloudcontroller images templates settings

/

sftp_user user to use to publish imgs via sftp

git_url makina-states fork (git url (ssh based)) to publish new images to

lxc specific lxc images settings

images mapping of images informations

cron_sync activate the img synchronnizer

cron_hour hour for the img synchronnizer

cron_minute minute for the img synchronnizer

mc_states.modules.mc_cloud_images.**ext_pillar**(*id_*, *prefixed=True*, *ttl=32140800*, **args*, ***kw*)
Images extpillar

mc_states.modules.mc_cloud_images.**pack_docker**(*image*, *destination_image=None*, *acls=None*, *tmpdir='/var/lib/docker'*)
Pack a docker image up to 1 or 2 layers, whith POSIX Acls support

First layer is the whole image Second layer contains the acls, restored.

Procedure:

•Import an image from exporting this container

•Run a container from this new image

•Restore acls

•Reexport / import

•Update the docker image tag

mc_states.modules.mc_cloud_images.**rebuild_ms_docker**(*image*, *destination_image=None*, ***kwargs*)
Rebuild & pack an image, in one row

image docker image to build from

destination_image docker image to tag upon successfu build (default to image)

acls if True: save/restore acls

refresh if True: run the rebuild dance

pack if True: pack the resulting image

Any kwargs will be forwarded to refresh_ms_docker.

mc_states.modules.mc_cloud_images.**settings**(*ttl=60*)
Images registry

mc_states.modules.mc_cloud_images.**sf_release**(*images=None*, *flavors=None*, *sync=True*)
Upload a prebuild makina-states layout in different flavors for various distributions to sourceforge.

For now this includes:

- lxc container based on Ubuntu LTS
- current ubuntu LTS based tarball containing the minimum vital to bring back to like makina-states without rebuilding it totally from scratch. This contains a slimed version of the containere files ffrom /salt-venv /srv/*salt /etc/*salt /var/log/*salt /var/cache/*salt /var/lib/*salt /usr/bin/*salt

this is used in makina-states.cloud.lxc as a base for other containers.

pillar/grain parameters: see mc_cloud_images.settings & mc_cloud_images.complete_images to set appropriate parameters for git, sourceforce, & etc urls & users

Do a release:

```
mastersalt-call -all mc_lxc.sf_release makina-states-trusty\
    [flavor=[lxc/standalone]] sync=True|False
```

mc_cloud_lxc / lxc registry for compute nodes
mc_states.modules.mc_cloud_lxc.**is_lxc**()
Return true if we find a system or grain flag that explicitly shows us we are in a LXC context
mc_states.modules.mc_cloud_lxc.**vm_extpillar**(*vm*, *data*, **args*, ***kw*)
Get per LXC container specific settings

mc_states.modules.mc_cloud_lxc.**vt_default_settings**(*cloudSettings*, *imgSettings*)
Default lxc container settings

clone_from default image

image LXC template to use 'ubuntu'

profile default profile to use. see saltstack definition of container profiles

network_profile default net profile to use. see saltstack definition of container networking profiles

bridge we install via states a bridge in 10.5/16 lxcbr1) 'lxcbr1'

backing (lvm, overlayfs, dir, brtfs) 'lvm'

vgname 'data'

fstab list of fstab entries

lvname 'data'

lxc_conf []

lxc_conf_unset []

vms List of containers ids classified by host ids:

```
    (Mapping of {hostid: [vmid]})
```

The settings are not stored here for obvious performance reasons

mc_states.modules.mc_cloud_lxc.**vt_extpillar**(*target*, *data*, **args*, ***kw*)
LXC extpillar

mc_cloud / cloud registries & functions

mc_states.modules.mc_cloud.**default_settings**()
makina-states cloud global configuration options

master The default master to link to into salt cloud profile

master_port The default master port to link to into salt cloud profile

mode (salt or mastersal (default)t)

pvdir salt cloud providers directory

pfdir salt cloud profile directory

bootsalt_branch bootsalt branch to use (default: master or prod if prod)

bootsalt_args makina-states bootsalt args in salt mode

bootsalt_mastersalt_args makina-states bootsalt args in mastersalt mode

keep_tmp keep tmp files

ssh_gateway (all the gw params are opt.) ssh gateway info

ssh_gateway_port ssh gateway info

ssh_gateway_user ssh gateway info

ssh_gateway_key ssh gateway info

ssh_gateway_password ssh gateway info

is mapping with various informations

controller is this minion a cloud controller

compute_node is this minion a cloud compute node

vm is this minion a cloud operating vm

mc_states.modules.mc_cloud.**ext_pillar**(*id_*, *prefixed=True*, *ttl=32140800*, **args*, ***kw*)
Makina-states cloud extpillar

NOTE This ext pillar is responsible for taking care of exposing other nodes configuration to a particular node if we have configured this via the expose/expose_limited settings

expose/exposed list of vm or compute nodes which will have full access to the vm infos in via ext_pillar

- expose mean give access to other vm conf
- exposed mean take acces on other vm conf

expose_limited/exposed_limited dict of vm or compute nodes which will have access to the vm infos via the ext_pillar Here sensitive info may be filtered, for now, nothing is implememted and we give full for now.

Levels are(only full is really implemented):

full full access to conf

light all conf but passwords or sensitive

network network conf only

example of pilar conf settings in database.yaml:

cloud_vm_attrs:

myvm:

expose_limited: other_vm0: full other_vm1: network other_vm2: password mynode: full

cloud_cn_attrs:

mynode:

expose_limited: other_vm1: full other_vm2: network other_node: password

This can be accessed client side via mc_cloud.settings/expositions

mastersalt-call mc_cloud.settings -> expositions / vms or cns

mc_states.modules.mc_cloud.**extpillar_settings**(*id_=None*, *ttl=32140800*, **args*, ***kw*)
return the cloud global configuation opts['id'] should resolve to mastersalt

mc_states.modules.mc_cloud.**gather_expositions**(*ttl=32140800*)
Merge expositions amongst CN & VM settings as a vm can also be a compute node itself

mc_states.modules.mc_cloud.**settings**(*ttl=60*)
Global cloud configuration

mc_cloud_saltify / cloud related variables

- This contains generate settings around cloud_saltify
- This contains also all targets to be driven using the saltify driver
- LXC driver profile and containers settings are in *mc_lxc / lxc registry*.

mc_states.modules.mc_cloud_saltify.**default_settings**(*cloudSettings*)
Except targets, we take all the default from *mc_cloud / cloud registries & functions*

bootsalt_args args to give to bootsalt (default to cloudcontroller configured value)

bootsalt_mastersalt_args args to give to bootsalt (default to cloudcontroller configured value)

mode salt mode (salt/mastersalt) (default to cloudcontroller configured value)

master salt master fqdn to rattach to (default to cloudcontroller configured value)

master_port salt master port to rattach to (default to cloudcontroller configured value)

bootsalt_branch default bootsalt_branch to use (default to cloudcontroller configured value)

id fqdn/minionid of the host to saltify

ssh_gateway (all the gw params are opt.) ssh gateway info

ssh_gateway_port ssh gateway info

ssh_gateway_user ssh gateway info

ssh_gateway_key ssh gateway info and default hosts key to ssh in

ssh_gateway_password ssh gateway password

ssh_keyfile use the ssh key (private) instead of using password base authentication

name name of the host if it does not match id (do not use...)

ip eventual ip if dns is not yet accessible

mode mastersalt or salt, keep at any price mastersalt or your are on your own

ssh_username user name to connect as to provision the box

password password to use (leave empty for key)

no_sudo_password disable sudo password handling (default: False). If the guest system disable sudo password asking, set this parameter to true

sudo_password sudo_password (leave empty to default to password)

sudo do we use sudo (bool)

targets List of minionid Targets where to bootstrap salt using the saltcloud saltify driver (something accessible via ssh)

mc_states.modules.mc_cloud_saltify.**target_extpillar**(*name*, *c_data=None*, *ttl=32140800*)

Settings for bootstrapping a target using saltcloud saltify driver (something accessible via ssh) mappings in the form:

mc_dumper / Some useful wrappers to dump/load values

mc_states.modules.mc_dumper.**cyaml_dump**(**args*, ***kw*)

Encode a value in yaml with raw c Yaml dumper

mc_states.modules.mc_dumper.**cyaml_load**(**args*, ***kw*)

Load a value encoded in yaml

The first positional argument is either a yaml value or a path filename.

mc_states.modules.mc_dumper.**iyaml_dump**(*data*, *flow=None*, **args*, ***kw*)

load a value in yaml

mc_states.modules.mc_dumper.**json_dump**(*data*, *pretty=False*, **args*, ***kw*)

encode a string in json

mc_states.modules.mc_dumper.**json_load**(*data*, **args*, ***kw*)

load a json string

mc_states.modules.mc_dumper.**msgpack_dump**(*data*, **args*, ***kw*)

encode a value with msgpack

mc_states.modules.mc_dumper.**msgpack_load**(*data*, **args*, ***kw*)

loade a msgpacked value

mc_states.modules.mc_dumper.**old_yaml_dump**(*data*, *flow=None*, *nonewline=None*, **args*, ***kw*)

encode a value with yaml

DO NOT TOUCH TO NONEWLINE=True as default (RETROCOMPAT WITH STATES)

mc_states.modules.mc_dumper.**yaml_dump**(*data*, *flow=None*, *nonewline=None*, **args*, ***kw*)

encode a value with yaml using saltstack yaml dumper

mc_states.modules.mc_dumper.**yaml_load**(**args*, ***kw*)

Wrapper to cyaml_load

mc_states.modules.mc_dumper.**yencode**(*string*, **args*, ***kw*)

wrapper to `yencode()`

mc_dbsmartbackup / db_smart_backup functions

mc_states.modules.mc_dbsmartbackup.**settings**()

Configuration registry for dbsmartbackup (https://github.com/kiorky/db_smart_backup)

cron_activated toggle on/off the nightly cron

cron_hour which hour of the day do we run the script

cron_minute which minute of the day do we run the script

backup_path_prefix root level dir for the backup storage

dbexclude exclude some databases from the backup

dbnames select databases to backup (default: all)

disable_mail to disable the summary email

global_backup do we also store the global objects and privileges (default: 1)

owner owner of the files

group group of the backup files

keep_days how many days to keep

keep_lasts how many 'last' backups to leep

keep_logs how many logs to keep

keep_monthes how many monthes to keep

keep_weeks': '8', how many weeks to keep

mail summary email recipient

mysqldump_autocommit do we use mysqldump autocommit

mysqldump_completeinserts

mysqldump_debug

mysqldump_locktables

mysqldump_noroutines

mysqldump_no_single_transaction

mysql_password mysql root password

mysql_sock_paths list of directories where to find mysql socket

mysql_use_ssl do we use ssl to connect to mysql

mongodb_path path to mongodb

mongodb_user mongodb admin

mongodb_password mongodb admin password

servername servername tied to those backups

mc_dhcpd / dhcpd registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_dhcpd
```

mc_states.modules.mc_dhcpd.**settings**()
dhcpd registry

mc_env / env registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_env
```

mc_states.modules.mc_env.**settings**()
env registry

default_env Environment defaults (one of: dev/prod/preprod)

mc_etckeeper
mc_states.modules.mc_etckeeper.**settings**()
etckeeper

mc_etherpad / etherpad functions
mc_states.modules.mc_etherpad.**settings**()
Etherpad settings

version Change which version of etherpad is installed.

location Change the directory in which circus is installed.

apikey The secret used to encrypt transmissions.

title The title of the server.

ip Ip on which the server will bind.

port Port the server will listen for.

dbType Type of the database.

dbSettings Settings of the database.

requireSession Require session setting.

editOnly Edit only setting.

admin Create an admin or not.

adminPassword Admin's password.

mc_fail2ban / fail2ban functions
mc_states.modules.mc_fail2ban.**settings**()
fail2ban settings

location conf dir

destemail: destination mail for alerts(root@fqdn})

loglevel

3.

logtarget (/var/log/fail2ban.log)

mail_from (fail2ban@makina-corpus.com)

mail_to (root)

mail_enabled (false)

mail_host (localhost)

mail_port

25.

mail_user (foo)

mail_password (bar)

mail_localtime (true)

mail_subject ([Fail2Ban] <section>: Banned <ip>)

mail_message

**(Hi,
 The IP <ip> has just been banned by Fail2Ban'** after <failures> attempts against <section>.
' Regards,
 Fail2Ban)

socket (/var/run/fail2ban/fail2ban.sock)

backend (polling)

bantime

86400.

maxretry

10.

ssh_maxretry ({maxretry})

protocol (tcp)

mta (sendmail)

banaction (iptables or shorewall if activated)

ignoreip ([127.0.0.1])

postfix_enabled (false)

wuftpd_enabled (false)

vsftpd_enabled (false)

proftpd_enabled (false)

pureftpd_enabled (false)

ssh_enabled (true)

recidive_enabled (false)

asterisk_tcp_enabled (false)

asterisk_udp_enabled (false)

named_refused_tcp_enabled (false)

mc_grub / grub registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_grub
```

mc_states.modules.mc_grub.**settings**()
grub registry

mc_haproxy / haproxy functions
mc_states.modules.mc_haproxy.**settings**()
haproxy settings

mc_icinga / icinga functions The first level of subdictionaries is for distinguish configuration files. There is one subdictionary per configuration file. The key used for subdictionary correspond to the name of the file but the "." is replaced with a "_"

The subdictionary "modules" contains a subsubdictionary for each module. In each module subdictionary, there is a subdictionary per file. The key "enabled" in each module dictionary is for enabling or disabling the module.

The "nginx" and "uwsgi" sub-dictionaries are given to macros in **kwargs parameter.

The key "package" is for listing packages installed between pre-install and post-install hooks

The keys "has_pgsql" and "has_mysql" determine if a local postgresql or mysql instance must be installed. The default value is computed from default database parameters If the connection is made through a unix pipe or with the localhost hostname, the booleans are set to True.

mc_states.modules.mc_icinga.**clean_global_variables**()
Function to remove global variables # TODO: find how to call this function

mc_states.modules.mc_icinga.**edit_configuration_object_settings**(*filen*, *attr*, *value*, *auto_host*, *definition*, ***kwargs*)
Settings for edit_configuration_object macro

mc_states.modules.mc_icinga.**get_settings_for_object**(*target=None*, *obj=None*, *attr=None*)
expand the subdictionaries which are not cached in mc_icinga.settings.objects

mc_states.modules.mc_icinga.**objects**()
icinga objects settings

this dictionary is the subdictionary of icinga.settings.objects but because of it is too big, we can't put it in the cache

dictionary to configure objects

directory directory in which objects will be stored. All the files in this directory are removed when salt is executed

objects_definitions dictionary to store objects configuration like commands, contacts, timeperiods, ... each subdictionary is given to configuration_add_object macro as **kwargs parameter

purge_definitions list of files which will be deleted. It is used to delete a host or specific service the file paths are given relative to directory specified above (in the "directory" key) each element in the list is given to configuration_remove_object macro as **kwargs parameter

autoconfigured_hosts_definitions dictionary to store hosts auto configurations ; each subdictionary is given to configuration_add_auto_host macro as **kwargs parameter

mc_states.modules.mc_icinga.**remove_configuration_object**(*filen=None*, *get=False*, ***kwargs*)
Add the file in the file's list to be removed

mc_icinga_web / icinga_web functions when it is written that "each key will be treated as a parameter", if you add keys in dictionary, they will be added in the configuration file with

```
<ae:paremeter name="{{key}}">{{value}}</ae:parameter>
```

when it is written that "foo must be replace with foo value", you can add as many sub-dictionaries as you want. All sub-dictionaries must have the same structure. It is to avoid a list but in templates it is treated as if there was a list of dictionaries.

I have prefered

```
'foo': {
    'n1': {
        'param1': "v1",
    },
    'n2': {
        'param1': "v2",
    },
},
```

instead of

```
'foo': [
    {'name': "n1",
     'param1': "v1",
    },
    {'name': "n2",
     'param1': "v2",
    },
]
```

When a real list is kept, It is precised below. Generally it is when the content is not structures but simple values.

The template of xml configuration files use a lot of loops in order to add content easily but it is not the case with the ini files where directives are limited and always the same.

The "nginx" and "phpfpm" sub-dictionaries are given to macros in **kwargs parameter. If you add a key in, you can access in the nginx configuration template or in phpfpm configuration template. In ngin x subdictionary, the "icinga_cgi" and "icinga_web" keys store values used to fill the template.

Otherwise, the first level of subdictionaries is for distinguish configuration files. There is one subdictionary per configuration file. The key used for subdictionary correspond to the name of the file but the "." is replaced with a "_"

I have not found dtd/xsd files in order to verify grammar of xml files.

Only the hashed password and the salt for the root account for icinga-web interface are stored in settings. The hashed password is not computed automatically from an other value with the clear password (settings dictionary doesn't contains the clear password)

I should add a state to compute the hash from clear password but I don't have successfully done this.

The keys "has_pgsql" and "has_mysql" determine if a local postgresql or mysql instance must be installed. The default value is computed from default database parameters If the connection is made through a unix pipe or with the localhost hostname, the booleans are set to True.

The default parameters for icinga_ido database connection are get from icinga settings.

In the templates, I didn't perform a lot of check. For example if a value must be set only if a other directive has a precise value, I didn't add a if statement. It is possible to create invalid configuration files.

If in a list, each value must be unique, I tried to have the elements of the list as dictionary keys.

For optional values which don't have a default value, I didn't set them in the default dictionary but in the templates, I have done:

```
{% if data.get('foo', None) %}
foo={{data.foo}}
{% endif %}
```

Theses optional values corresponds to commented keys in the default dictionary.

mc_states.modules.mc_icinga_web.**settings**()

icinga_web settings

location installation directory

package list of packages to install icinga-web

configuration_directory directory where configuration files are located

has_pgsql install and configure a postgresql service in order to store icinga-web data (no ido2db data)

has_mysql install and configure a mysql service in order to store icinga-web data (no ido2db data)

modules

nagvis

enable enable the nagvis module which add link to nagvis in icinga-web

cronks_xml dictionary to store the cronks. The content is added in cronks.xml. The structure is the same that 'cronks_xml' subdictionary.

pnp4nagios

enable enable the pnp4nagios module which add links to graphs in icinga-web

package package to install for pnp4nagios integration

cronks_extensions-templates dictionary in which, each key is the name of an extension template and the content of the dictionary contains the values to fill the template each 'key': "value" produce a "<parameter name={{key}}>{{value}}</parameter>". The key "parameter" or "parameter_*" produce a "<parameter></parameter>" tag. Each subdictionary add sub parameters tags.

root_account Dictionary to store root account information. It is the account created on first installation of icinga_web

login login for root login on web interface

hashed_password password for root login on web interface

salt salt used to hash the password

databases dictionary to store databases connections parameters

ido2db dictionary to store ido2db database connection parameters

type type of sgbd used "pgsql" or "mysql"

host host used for connection

port port used for connection

user user used for connection

password password used for connection

name database name

prefix prefix used in table's names

web dictionary to store icinga-web database connection parameters

type type of sgbd used “pgsql” or “mysql”

host host used for connection

port port used for connection

user user used for connection

password password used for connection

name database name

nginx dictionary to store values of nginx configuration

domain name of virtualhost created to serve webpages

doc_root root location of virtualhost

vh_content_source template file for nginx content file

vh_top_source template file for nginx top file

icinga_web dictionary to store values used in templates given in vh_content_source and vh_top_source

web_directory location under which webpages of icinga-web will be available

images_dir directory where images used by icinga-web are stored

styles_dir directory where css used by icinga-web are stored

bpaddon_dir directory where bpaddon scripts are located

ext3_dir directory where ext3 scripts are located

fastcgi_pass socket used to contact fastcgi server in order to interpret php files

phpfpm dictionary to store values of phpfpm configuration

open_basedir paths to add to open_basedir

extensions_package additional packages to install (such as php5-pgsql or php5-mysql for php database connection)

doc_root root location for php-fpm

session_auto_start must be 0 to run icinga-web

mc_java / java registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_java
```

mc_states.modules.mc_java.**settings**()

java registry

default_jdk_ver default JDK version

mc_kernel / kernel registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_kernel
```

mc_states.modules.mc_kernel.**settings**()

kernel registry

Systcl values for

tcp_wmem 4096 65536 16777216

tcp_rmem 4096 87380 16777216

rwmemmax 16716777216 77216

ip_local_port_range 1025 65535

tcp_max_sync_backlog 20480

tcp_fin_timeout 15

net_core_somaxconn 4096

netdev_max_backlog 4096

no_metrics_save 1

ulimit 64000

tcp_congestion_control cubic

tcp_max_tw_buckets 2000000

tcp_tw_recycle 0

vm_min_free_kbytes int(((grains['mem_total']/64)*1024*1024)/1000)

tcp_syn_retries 2

tcp_tw_reuse 1

tcp_timestamps 0

vm_swappiness 1

mc_locales / locales registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_locales
```

mc_states.modules.mc_locales.**settings**()

locales registry

locales locales to use

default_locale Default locale

mc_locations
mc_states.modules.mc_locations.**settings**(*cached=True*)
locations

mc_logrotate / logrotate registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_logrotate
```

mc_states.modules.mc_logrotate.**settings**()
logrotate registry

days days rotatation

mc_lxc / lxc registry This module contains settings for lxc and helper functions

mc_states.modules.mc_lxc.**is_lxc**()
in case of a container, we have the container name in cgroups else, it is equal to /

in lxc: ['11:name=systemd:/user/1000.user/1.session', '10:hugetlb:/thisname', '9:perf_event:/thisname', '8:blkio:/thisname', '7:freezer:/thisname', '6:devices:/thisname', '5:memory:/thisname', '4:cpuacct:/thisname', '3:cpu:/thisname', '2:cpuset:/thisname']

in host: ['11:name=systemd:/', '10:hugetlb:/', '9:perf_event:/', '8:blkio:/', '7:freezer:/', '6:devices:/', '5:memory:/', '4:cpuacct:/', '3:cpu:/', '2:cpuset:/']

mc_states.modules.mc_lxc.**settings**()
Lxc registry

virt defaults (makina-states.services.virt.lxc) is_lxc

containers Mapping of containers defintions classified by host

mc_memcached / memcached registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_memcached
```

mc_states.modules.mc_memcached.**settings**()
memcached registry

mc_mongodb / mongodb functions
mc_states.modules.mc_mongodb.**settings**()
mongodb settings

location installation directory

mc_mumble / mumble registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_mumble
```

mc_states.modules.mc_mumble.**settings**()
mumble registry

mc_mysql / mysql functions

mc_states.modules.mc_mysql.**settings**(***kwargs*)
mysql settings

isPercona TDB

isOracle TDB

isMariaDB TDB

number_of_table_indicator number of tables contained in the database

port TBD

user TBD

group TBD

root_passwd autogenerated root password (can be overriden before db install)

sharedir TBD

users TBD

datadir TBD

tmpdir TBD

etcdir TBD

logdir TBD

basedir TBD

sockdir TBD

conn_host / conn_user / conn_pass autogenerated Connection settings, user/pass/host used by salt to manage users, grants and database creations

character_set default character set on CREATE DATABASE (use utf8' not 'utf-8')

collate default collate on CREATE DATABASE

noDNS Avoid name resolution on connections checks, must-have. This is the skip-name-resolv option

memory_usage_percent / available_mem the macro will compute magiccaly the settings to fit this percentage of full memory on the host. So by default it's 50% of all RAM on a dev envirronment and 85% for a production one where the MySQl server should be alone on a server. Then all others settings parameters in the 'tunning' key could be set to False to let the macro fill the gaps. If you set somtehing other than False for one of theses settings it will be used instead of the value computed by the macro, check the macro for details and comments on all theses parameters. Note the "_M" means Mo, so for 2Go of innodb_buffer_pool_size use 2024, that is 2024Mo. Tweak the 'number_of_table_indicator' to adjust some settings automatically

from that, for example several Drupal instances using a lot of fields could manage several hundreds of tables. <=== IMPORTANT

If you want to fine tune the mysql server, read the method to know non documented parameters to override, the code is well documented but spared from configuration from end users.

mc_nagvis / nagvis functions You can add your own key/values in backends, rotations and actions subdictionaries.

mc_states.modules.mc_nagvis.**add_geomap_settings**(*name*, *hosts*, ***kwargs*)
Settings for the add_geomap macro

mc_states.modules.mc_nagvis.**add_map_settings**(*name*, *_global*, *objects*, *keys_mapping*, ***kwargs*)
Settings for the add_map macro

mc_states.modules.mc_nagvis.**settings**()
nagvis settings

location installation directory

package list of packages to install navgis

configuration_directory directory where configuration files are located

root_account dictionary to store root account information. It is the account with the userId=1 in the sqlite database

login login for root login on web interface

hashed_password password for root login on web interface

salt salt used to hash the password

default_password the password inserted when nagvis is installed. it is to check that the password was not previously modified

nginx

dictionary to store values of nginx configuration

domain name of virtualhost created to serve webpages

doc_root root location of virtualhost

vh_content_source template file for nginx content file

vh_top_source template file for nginx top file

nagvis dictionary to store values used in templates given in vh_content_source and vh_top_source

web_directory location under which webpages of nagvis will be available

fastcgi_pass socket used to contact fastcgi server in order to interpret php files

phpfpm dictionary to store values of phpfpm configuration

open_basedir paths to add to open_basedir

doc_root root location for php-fpm

session_auto_start must be 0 to run nagvis

global_php dictionary to store values used in global.php

AUTH_PASSWORD_SALT salt used for password. We notice the salt used is the same for all passwords which is a security weakness.

nagvis_ini_php dictionary to store values used in nagvis_ini_php each subdictionary represents an ini section if a key is not present, the directive will not be added in the configuration file

global dictionary to store values of global section in nagvis_ini_php

authmodule name of authentication module

authorisationmodule name of authorisation module

file_group group used to launch nagvis script

file_mode default file mode for temporary files

geomap_server url of geomap server

http_proxy http proxy

http_proxy_auth auth for http proxy

paths dictionary to store values of paths section in nagvis_ini_php

base location of nagvis installation

htmlbase location of php files. It should be the same value that nginx.nagvis.web_directory

defaults dictionary to store values of defaults section in nagvis_ini_php

backend default backend

automap dictionary to store values of automap section in nagvis_ini_php

defaultparams default parameters

defaultroot default root

graphvizpath location of graphviz binary

backends dictionary to store values of backends section in nagvis_ini_php each subdictionary corresponds to a "backend_foo" section

foo dictionary to store values of foo backend. foo must be replaced with the name of the backend the keys and values expected depends on the backend type

backendtype type of backend

rotations dictionary to store values of rotations section in nagvis_ini_php each subdictionary corresponds to a "rotation_foo" section

foo dictionary to store values of foo rotation. foo must be replaced with the name of the rotation

maps list of maps which are in the rotation

interval interval

actions dictionary to store values of actions section in nagvis_ini_php each subdictionary corresponds to a "action_foo" section

foo dictionary to store values of foo action. foo must be replaced with the name of the action

action_type type of action

obj_type "host" or "service" or "host,service"

condition condition to apply the action

domain domain

username username

mc_network / network registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_network
```

mc_states.modules.mc_network.**default_net**()
Function to be used on a running system (opposed to settings) Use by default a bridge (with main interface as the only first port) or the main interface as the link with internet

mc_states.modules.mc_network.**ext_ip**()
Return the external IP address

mc_states.modules.mc_network.**get_broadcast**(*dn*, *ip*)
Get a server broadcase ipsubnet.255

mc_states.modules.mc_network.**get_dnss**(*dn*, *ip*)
Get server dnss

mc_states.modules.mc_network.**get_fo_broadcast**(*dn*, *ip*)
Get broadcast for an ip failover

mc_states.modules.mc_network.**get_fo_netmask**(*dn*, *ip*)
Get netmask for an ip failover

mc_states.modules.mc_network.**get_gateway**(*dn*, *ip*)
Get a server gateway default to ipsubnet.254 except for online where the gw == ipsubnet.1

mc_states.modules.mc_network.**get_netmask**(*dn*, *ip*)
Get a server netmask default to ipsubnet.255

mc_states.modules.mc_network.**ns_whois**(*name*, *ttl=86400*, *cache=True*, *whois_ttl=2592000*)
Make a whois request and return data For evident performance questons, We cache whois data for one month!

mc_states.modules.mc_network.**settings**()
network registry

networkManaged Do we manage the network configuration

interfaces Dict of configuration for network interfaces

main_ip main server ip

hostname main hostname

domain main domain

devhost_ip devhost ip

mc_states.modules.mc_network.**whois_data**(*ip*, *ttl=86400*, *whois_ttl=2592000*)
Make a whois request and return data For evident performance questons, We cache whois data for one month!

mc_nginx / nginx registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_nginx
```

mc_states.modules.mc_nginx.**settings**()

nginx registry

is_reverse_proxied is nginx itself is reverse proxified (true in cloudcontroller mode)

reverse_proxy_addresses authorized reverse proxied addresses

use_real_ip do we use real ip module

use_naxsi configure & use naxsi

use_naxsi_secrules use sec rules in naxsi

naxsi_ui_pass pass for naxsi leaning mode ui

naxsi_ui_host host for naxsi leaning mode ui

naxsi_ui_intercept_port intercept_port for naxsi leaning mode ui

naxsi_ui_extract_port extract port for naxsi leaning mode ui

use_naxsi_learning put naxsi in learning mode

naxsi_denied_url uri for naxsi refused cnx

real_ip_header which http header to search for real ip

reverse_proxy_addresses control real ip addresses (list of values). Default to network gateway (in cloud controllermode, haproxy is there)

logformat default log format

logformats custom log formats mapping

allowed_hosts default allowed hosts

ulimit default ulimit for the workers

open_file_cache raw setting for nginx (see nginx documentation)

open_file_cache_valid raw setting for nginx (see nginx documentation)

open_file_cache_min_uses raw setting for nginx (see nginx documentation)

open_file_cache_errors raw setting for nginx (see nginx documentation)

epoll do we use epoll (true on linux)

default_type raw setting for nginx (see nginx documentation)

worker_processes nb workers, default to nb of cpus

worker_connections raw setting for nginx (see nginx documentation)

multi_accept raw setting for nginx (see nginx documentation)

ssl_cacert ssl_cacert content if any

ssl_redirect unconditionnal www -> ssl redirect

ssl_protocols SSLv3 TLSv1 TLSv1.1 TLSv1.2

ssl_cacert_first False

ssl_session_cache shared:SSL:10m

ssl_session_timeout 10m

ssl_cipher IGH:!aNULL:!MD5

user nginx user

server_names_hash_bucket_size raw setting for nginx (see nginx documentation)

loglevel nginx error loglevel (crit)

logdir nginx logdir (/var/log/nginx)

access_log '{logdir}/access.log

ldap_cache: use ldap auth plugin cache (True)

sendfile raw setting for nginx (see nginx documentation)

tcp_nodelay raw setting for nginx (see nginx documentation)

tcp_nopush raw setting for nginx (see nginx documentation)

reset_timedout_connection raw setting for nginx (see nginx documentation)

client_body_timeout raw setting for nginx (see nginx documentation)

send_timeout raw setting for nginx (see nginx documentation)

keepalive_requests raw setting for nginx (see nginx documentation)

keepalive_timeout raw setting for nginx (see nginx documentation)

types_hash_max_size raw setting for nginx (see nginx documentation)

server_tokens raw setting for nginx (see nginx documentation)

server_name_in_redirect raw setting for nginx (see nginx documentation)

error_log '{logdir}/error.log'

gzip enabling gzip

redirect_aliases do we redirect server aliases to main domain

port http port (80)

ssh_port https port (443)

default_domains default domains to server ['localhost']

docdir /usr/share/doc/nginx

doc_root /usr/share/nginx/www

vhost_default_template salt://makina-states/files/etc/nginx/sites-available/vhost.conf

vhost_wrapper_template Default template for vhosts salt://makina-states/files/etc/nginx/sites-available/vhost.conf

vhost_default_content Default content template for the DEFAULT DOMAIN vhost salt://makina-states/files/etc/nginx/sites-available/default.conf'

vhost_top_template default template to include in vhost top salt://makina-states/files/etc/nginx/sites-available/vhost.top.conf,

vhost_content_template default template for vhost content salt://makina-states/files/etc/nginx/sites-available/vhost.content.conf

virtualhosts Mapping containing all defined virtualhosts

rotate days to rotate log

default_vhost set to false to disable default vhost

`mc_states.modules.mc_nginx.`**`vhost_settings`**(*domain*, *doc_root*, ***kwargs*)

Settings for the nginx macro

ssl_cert ssl_cert content if any

ssl_key ssl_key content if any

mc_nodejs / nodejs/npm registry

`mc_states.modules.mc_nodejs.settings()`

nodejs

mc_ntp / ntp registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_ntp
```

`mc_states.modules.mc_ntp.settings()`

ntp

servers server entries without the fudge headers

fudge fudge entries without the fudge headers

default_all allow query from ext (firewalled in ms)

block_ext block all outbound queryies

default restrict set to false to generate a no<flag>

ignore False

limited False

lowpriotrap False

kod False

peer True

trap False

serve True

trust True

modify False

query False

restrict list of restrict entries without the leading restrict

mc_pgsql / Postgresql related functions

`mc_states.modules.mc_pgsql.settings()`

Postgresql settings registry

pgDbs mapping of postgresql databases with their settings

postgresqlUsers mapping of postgresql users with their settings

user system user

version default postgres version

defaultPgVersion default postgres version

versions activated postgresql version

postgis mapping of supported postgres per postgis version

postgis_db name of the postis template

pg_hba List of pg_hba entries

pg_conf settings for postgresql.conf 'default' is looked up by default but you can override settings for a specific version inside a subversion part eg:

```
makina-states.services.db.postgresql.pg_conf.9.1:
    port: 5433


listen
    list of hosts to listen on
port
    value for port
unix_socket_directories
    value for unix_socket_directories
extras
    free dict with key/values pairs,
    for strings you must quote them::

        makina-states.services.db.postgresql.pg_conf.extras:
          # no quote
          enable_tidscan: 'on'
          # quote
          log_line_prefix: "'%t '"
```

mc_states.modules.mc_pgsql.**wrapper**(*wrappy*)

Wrap a postgresql salt module function to set automaticly the socket or port to use. This is debian specific for now

mc_phantomjs / phantomjs/npm registry

mc_states.modules.mc_phantomjs.**settings**()

phantomjs

mc_php / php registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_php
```

mc_states.modules.mc_php.**composer_command**(*command=None*, *cwd=None*, *args=None*, *composer=None*)

Run a composer command. Result of the command is in the 'msg' key of the returnded dictionnary

composer full path to composer, defaulting to '/usr/local/bin/composer'

command the command you want in composer.

args string of command arguments, optionnal

mc_states.modules.mc_php.**fpmpool_settings**(*domain*, *doc_root*, ***kw*)

Generate options to be given for the pool configuration generation Some on the main options:

session_cookie_domain Special cookie domain string for cookies (totally optionnal)

listen Custom listen string for php fpm listen directive For example, if you do not want to use the default sockets scheme

pool_name force the fpm pool name (useful for multiple projects to use the same pool)

chroot Do we run in a fpm chrooted env. (certainly defaults to true in current layout)

active True by default, set to False to disable the Virtualhost even if it will be generated.

mc_states.modules.mc_php.**settings**()

This is called from mc_services, loading all PHP default settings

Settings are merged with grains and pillar via mc_utils.defaults

```
    ------------------------------------------------
    -------- MODULE ZEND OPCACHE -------------------------
    replacement for APC!
    @see for details of options:
    https://raw.github.com/zendtech/ZendOptimizerPlus/master/README
```

```
    -------- MODULE APC --------------------------------
    WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING
    WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING
    WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING
    WARNING : APC is somewhat deprecated and Zend opcache is the replacment
    So the default behavior will not be to install it!!!
```

APC General shared settings

shm_segments seems to perform better with only one shared segment but if you cannot upgrade this segment size, then create several setting ignored in mmap mode (so chances are this will always be 1)

shm_size so here the segment size, but you may need to allow it in your OS for 128M Put in /etc/sysctl.conf

- kernel.shmmax=134217728
- kernel.shmall=2097152

default in most OS is 32M

mmap_file_mask If compiled with MMAP support by using –enable-mmap this is the mktemp-style file_mask to pass to the mmap module for determining whether your mmap'ed memory region is going to be file-backed or shared memory backed

APC

Per virtualhost/php-fpm pool:

enabled enabling apc

rfc1867 allow progress upload bars

APC

include_once_override Optimisation of include/require_once calls

canonicalize transform paths in absolute ones (no effect if apc.stat is not 0), files from stream wrappers (extended includes) won't be cached if this is activated as they cannot be used with php's realpath()

stat In production set it to False, then file changes won't be observed before apache or php-fpm is restarted, significant boost, else file time is stated at each access (needed at True in dev)

stat_ctime avoid problems with rsync or svn not modifying mtime but only ctime so if you're in production set this to False, like for the previous one

num_files_hint indication on number of files (ZF=1300, nude Drupal 7=1000)

user_entries_hint indication on the number of cache variables

APC: cache lifetime managmenent

ttl time (s) we can stay on the cache even when the cache is full

Cache full count that means Garbage Collector is never inactivating theses datas before this time is over

>0 old data could stay in the cache while new data wants to come, if no data is deprecated

7200 entries older than 2 hours will be thrown to make some place

0 emptying full cache when full

user_ttl same as above, for user cache

gc_ttl this one is the same but you should note this prevents Garbage collecting after each source change.

APC

filters could be used to prevent some caching on specific files but it's better to cache often used files, isn't it? At least in production

max_file_size factory default to 1M, files bigger than that won't be cached

APC: various things

write_lock if True only one process caching a same file (better than apc.slam_defense)

file_update_protection "2" prevents caching half written files (by cp for example) by waiting x seconds for new files caching. set it to 0 if using only rsync or mv

lazy_functions early versions of APC only optimisations from Facebook, adding a lazy loding capabilities, so you can parse a lot of files and only used things are cached NEED TO BE TESTED: DANGEROUS!!

lazy_classes same as above

MODULE XDEBUG:

```
    php_admin_value[xdebug.default_enable] =   xdebug_default_enable ;
    ; http://xdebug.org/docs/all_settings#collect_params (0|1|2|3|4)
    php_admin_value[xdebug.collect_params] = xdebug_collect_params  0;
    php_admin_value[xdebug.profiler_enable] = xdebug_profiler_enable ;
    php_admin_value[
        xdebug.profiler_enable_trigger
    ] = xdebug_profiler_enable_trigger  0;
    php_admin_value[
        xdebug.profiler_output_name
    ] = xdebug_profiler_output_name  /cachegrind.out.%p;
```

mc_pkgs / pkgs registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_pkgs
```

mc_states.modules.mc_pkgs.**settings**()
pkgs registry

installmode install or update mode (install/latest)

keyserver default GPG server

dist current system dist

lts_dist current distributaion stable release

force_apt_ipv4 force apt to use ipv4

force_apt_ipv6 force apt to use ipv6

apt

ubuntu

dist dist version

comps enabled comps

mirror mirror

last last unstable release

lts stable release

debian

stable stable release

dist current dist

comps enabled comps

mirror mirror

mc_pnp4nagios / pnp4nagios functions There are many way to configure pnp4nagios. I have chosen to configure pnp4nagios "Bulk Mode with NPCD and npcdmod" (http://docs.pnp4nagios.org/pnp-0.6/config#bulk_mode_with_npcd_and_npcdmod) because of the lightness of the configuration

`mc_states.modules.mc_pnp4nagios.`**`settings`**`()`

pnp4nagios settings

location installation directory

package list of packages to install icinga-web

configuration_directory directory where configuration files are located

nginx dictionary to store values of nginx configuration

domain name of virtualhost created to serve webpages

doc_root root location of virtualhost

vh_content_source template file for nginx content file

vh_top_source template file for nginx top file

pnp4nagios dictionary to store values used in templates given in vh_content_source and vh_top_source

web_directory location under which webpages of pnp4nagios will be available

fastcgi_pass socket used to contact fastcgi server in order to interpret php files

realm message displayed for digest authentication

htpasswd_file location of file storing users password or url for ldap authent

htdoc_dir root location for web_directory

phpfpm dictionary to store values of phpfpm configuration

open_basedir paths to add to open_basedir

extensions_package additional packages to install (such as php5-pgsql or php5-mysql for php database connection)

doc_root root location for php-fpm

session_auto_start must be 0 to run icinga-web

npcd_cfg dictionary to store configuration of npcd.cfg file

config_php dictionary to store configuration of config.php file

conf subdictionary to store the values of the $conf[] php array variable

views subdictionary to store the values of the $views[] php array variable

each subdictionary under views corresponds to a subarray. The key of subdictionaries are the values for "title" array key

rra_cfg dictionary to store the configuration of rra.cfg file

RRA_STEP value for RRA_STEP

steps list of strings where each string is a line in rra.cfg file

mc_postfix / postfix functions

`mc_states.modules.mc_postfix.`**`settings`**`()`

postfix settings

mode

custom custom mode, specific explictly all your options

relay satellite mode

localdeliveryonly mails are only delivered locally, no email are sent on the network (default)

catchall redirect all mail to this user if set if localdelivery is enabled, all mail are redirected to one user user is root, guest or nobody if catchall is False, no catchall will happen

use_tls do we use tls ('yes')

inet_protocols which protocol to enable ('ipv4')

check_policy_service content filtering service (None)

conf_dir main configuration directory (/etc) (locs['conf_dir'])

mailname this server address to use in recipien/exp. (grains['fqdn'])

cert ssl certificate content including all the chain of certification Will use mc_ssl based on mailname instead

cert_key ssl certificate key Will use mc_ssl based on mailname instead

inet_interfaces where to bind('all')

mailbox_size_limit size max of the mailbox(0)

auth enable smtp auth

mynetworks list of hosts/nets to add to mynetworks

relay_domains Mapping {relaydomain: action}

transports list of mappings {transport:'', 'nexthop': ''} default transport is '*'. This can be used to make a satellite

virtual_map dict of key/value pair to feed the virtual table

mc_provider / provider functions Useful functions to locate a particular host or setting

class mc_states.modules.mc_provider.**ClientNotActivated**
.

mc_states.modules.mc_provider.**settings**()
provider settings

is booleans

online are we on an online host

ovh are we on an ovh host

sys are we on an soyoustart host

have_rpn online specific: do we have rpn

mc_psad / psad functions
mc_states.modules.mc_psad.**settings**()
psad settings

alertdest (root@fqdn)

hostname (fqdn)

mc_pureftpd / pureftpd functions
mc_states.modules.mc_pureftpd.**settings**()
pureftpd settings

Daemon lauch parameters (makina-states.services.ftp.pureftpdefaults)

Virtualchroot TDB

InetdMode TDB

UploadUid TDB

UploadGid TDB

UploadScript TDB

Pureftp configuration (makina-states.services.ftp.pureftp)

AllowAnonymousFXP TDB

AllowDotFiles TDB

AllowUserFXP TDB

AltLog TDB

AnonymousBandwidth TDB

AnonymousCanCreateDirs TDB

AnonymousCantUpload TDB

AnonymousOnly TDB

AnonymousRatio TDB

AntiWarez TDB

AutoRename TDB

Bind TDB

BrokenClientsCompatibility TDB

CallUploadScript TDB

ChrootEveryone TDB

ClientCharset TDB

Daemonize TDB

DisplayDotFiles TDB

DontResolve TDB

FSCharset TDB

IPV4Only TDB

IPV6Only TDB

KeepAllFiles TDB

LimitRecursion TDB

LogPID TDB

MaxClientsNumber TDB

MaxClientsPerIP TDB

MaxDiskUsage TDB

MinUID TDB

NATmode TDB

NoAnonymous TDB

NoChmod TDB

NoRename TDB

NoTruncate TDB

Quota TDB

SyslogFacility TDB

TLS TDB

TrustedGID TDB

TrustedIP TDB

Umask TDB

UserBandwidth TDB

UserRatio TDB

VerboseLog TDB

PassiveIP TDB

PassivePortRange TDB

PAMAuthentication TDB

UnixAuthentication TDB

PureDB TDB

MySQLConfigFile TDB

ExtAuth TDB

LDAPConfigFile TDB

PGSQLConfigFile TDB

mc_python / python registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_python
```

mc_states.modules.mc_python.**settings**()
python registry

versions python versions

alt_versions different versions from system(internal var)

version current version

mc_rabbitmq / rabbitmq functions
mc_states.modules.mc_rabbitmq.**settings**()
rabbitmq settings

location installation directory

mc_rdiff-backup / rdiff-backup functions
mc_states.modules.mc_rdiffbackup.**settings**()
rdiff-backup settings

mc_redis / redis functions
mc_states.modules.mc_redis.**settings**()
redis settings

location installation directory

mc_rsyslog / rsyslog functions
mc_states.modules.mc_rsyslog.**settings**()
rsyslog settings

spool spool directory

user syslog user

group syslog group

admin_group admin group

listen_addr listen address

- 0.0.0.0 on baremetal
- 127.0.0.1 on vms

Yes syslog is opened to world on baremetal, but we filter it using the restriction feature of our shorewall installation, see *mc_shorewall / shorewall functions*, so please install also shorewall ! By default on baremetal it will accept only localhost traffic.

udp_port udp port (514)

mc_rvm / rvm registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_rvm
```

mc_states.modules.mc_rvm.**settings**()
rvm registry

rvm_url rvm download url

rubies Activated rubies

rvm_user RVM user

rvm_group RVM group

mc_salt / salt related helpers
mc_states.modules.mc_salt.**settings**()
Registry of settings decriving salt installation

Please read the code to be sure to understand it before changing parameters as it can brick your installation. That's why most of this stuff will be underdocumented at first sight.

mc_screen / screen registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_screen
```

mc_states.modules.mc_screen.**settings**()
screen registry

mc_shorewall / shorewall functions
mc_states.modules.mc_shorewall.**settings**()
shorewall settings

makina-states.services.shorewall.enabled: activate shorewall

It will also assemble pillar slugs to make powerfull firewalls by parsing all ***-makina-shorewall** pillar entries to load the special shorewall structure:

All entries are merged in the lexicograpical order

makina-states.services.firewall.shorewall:

interfaces TBD

rules TBD

params TBD

policies TBD

zones TBD

masqs TBD

proxyarp TBD

nat TBD

mc_snmpd / snmpd functions snmpd module

mc_states.modules.mc_snmpd.**settings**()

snmpd settings

SNMPDRUN yes

MIBS /usr/share/mibs

SNMPDOPTS -Lsd -Lf /dev/null -p /var/run/snmpd.pid

TRAPDRUN no

TRAPDOPTS -Lsd -p /var/run/snmptrapd.pid

agentAddress udp:161,udp6:[::1]:161

default_user user

default_key sup3rs3cret

default_password s3cret

default_enc_type DES

default_password_enc_type SHA

mc_ssh / OpenSSH related functions

mc_states.modules.mc_ssh.**settings**()

Open ssh registry

prefixes:

- makina-states.services.ssh.server

settings.AuthorizedKeysFile List of authorized key filepaths

settings.ChallengeResponseAuthentication do we authorize ChallengeResponseAuthentication

settings.X11Forwarding do we authorize X11Forwarding

settings.PrintMotd do we print motd

settings.UsePrivilegeSeparation UsePrivilegeSeparation mode

settings.Banner path to the banner

settings.UsePAM do we use pam authentication

sshgroup

named of the allowed group or users allowed to connect via ssh

AllowUsers List of users allowed to connect via ssh

AllowGroups List of users allowed to connect via ssh

- makina-states.services.ssh.client

  StrictHostKeyChecking to be documented

  UserKnownHostsFile to be documented

  AddressFamily to be documented

  ConnectTimeout to be documented

  SendEnv to be documented

  HashKnownHosts to be documented

  GSSAPIAuthentication to be documented

  GSSAPIDelegateCredentials to be documented

mc_ssl / ssl registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_ssl
```

mc_states.modules.mc_ssl.**ca_ssl_certs**(*domains*, ***kwargs*)

Wrapper to ssl_certs to also return the cacert information Return a triple (ert, key, ca) if ca is none: ca==''

mc_states.modules.mc_ssl.**domain_match**(*domain*, *cert_domain*, *wildcard_match=False*)

Test if a domain exactly match other domain the other domain can be a wildcard, and this only match top level wildcards as per openssl spec

```
>>> from mc_states.modules.mc_ssl import domain_match
>>> domain_match('a.com', 'a.com')
True
>>> domain_match('a.a.com', '*.a.com')
True
>>> domain_match('a.a.a.com', '*.a.com')
False
>>> domain_match('aaa.a.com', '*.a.com')
True
>>> domain_match('a', '*')
False
>>> domain_match('a.a', '*.a')
False
>>> domain_match('a.com', '*.a.com')
True
>>> domain_match('a.com', '*.a.a.com')
False
```

mc_states.modules.mc_ssl.**domain_match_wildcard**(*domain*, *wildcard_or_domain*)

Test if a common name matches a wild card

```
>>> from mc_states.modules.mc_ssl \
...     import domain_match_wildcard as match_wildcard
>>> match_wildcard('foo.dom.net', '*.foo.dom.net')
True
>>> match_wildcard('www.foo.dom.net', '*.foo.dom.net')
```

```
    True
    >>> match_wildcard('foo.dom.net', 'foo.dom.net')
    True
    >>> match_wildcard('www.foo.dom.net', 'foo.dom.net')
    True
    >>> match_wildcard('dom.net', '*.dom.net')
    True
    >>> match_wildcard('www.dom.net', '*.dom.net')
    True
    >>> match_wildcard('dom.net', 'dom.net')
    True
    >>> match_wildcard('www.dom.net', 'dom.net')
    True
```

mc_states.modules.mc_ssl.**get_cert_for**(*domain*, *gen=False*, *domain_csr_data=None*)
Generate or return certificate for domain

The certificates are stored inside <pillar_root>/cloud-controller/ssl

Search order precedence:

- ./custom/<subdomain>.<domain>.<tld>
- wildcard certificate: ./custom/*.<domain>.<tld>
- signed by the controller: ./<cloudctlr>/certs/*.<domain>.<tld>
- signed by the controller: ./<cloudctlr>/certs/<sub>.<domain>.<tld>

mc_states.modules.mc_ssl.**get_configured_cert**(*domain*, *gen=False*, *ttl=60*)
Return any configured ssl cert for domain or the wildward domain matching the precise domain. It will prefer to use any real signed certificate over a self signed certificate

mc_states.modules.mc_ssl.**get_custom_cert_for**(*domain*)
Seach for certificate and key file inside pillar folder

pillarroot/cloud-controller/ssl/custom:

<domain>.key contain private ,key

<domain>.crt contain cert

<domain>.auth.crt contain auth chain

<domain>.bundle.crt <generated if not present> contain cert + auth chain

<domain>.full.crt <generated if not present> contain cert + auth chain + key

mc_states.modules.mc_ssl.**get_installed_cert_for**(*domain*)
Seach for certificate and key file inside pillar folder

pillarroot/cloud-controller/ssl/custom:

<domain>.key contain private ,key

<domain>.crt contain cert

<domain>.auth.crt contain auth chain

<domain>.bundle.crt <generated if not present> contain cert + auth chain

<domain>.full.crt <generated if not present> contain cert + auth chain + key

mc_states.modules.mc_ssl.**get_selfsigned_cert_for**(*domain*, *gen=False*, *domain_csr_data=None*)
Generate or return certificate for domain

The certificates are stored inside <pillar_root>/cloud-controller/ssl

Search precedence:

- ./custom/<subdomain>.<domain>.<tld>
- wildcard certificate: ./custom/*.<domain>.<tld>
- selfsigned: ./selfsigned/certs/*.<domain>.<tld>
- selfsigned: ./selfsigned/certs/<subdomain>.<domain>.<tld>

mc_states.modules.mc_ssl.**load_certs**(*path*)
Load certificates from a directory (certs must be suffixed with .crt) return 2 dictionnary:

- one contains certs with common name as indexes
- one contains certs with subjectaltnames as indexes

mc_states.modules.mc_ssl.**load_selfsigned_certs**(*path*)
Load certificates from a directory (certs must be suffixed with .crt) return 2 dictionnary:

- one contains certs with common name as indexes
- one contains certs with subjectaltnames as indexes

mc_states.modules.mc_ssl.**search_matching_certificate**(*domain*, *as_text=False*, *selfsigned=True*)
Search in the pillar certificate directory the certificate belonging to a particular domain

mc_states.modules.mc_ssl.**search_matching_selfsigned_certificate**(*domain*, *gen=False*, *as_text=False*)
Search in the pillar certificate directory the certificate belonging to a particular domain

mc_states.modules.mc_ssl.**selfsigned_last**(*ctuple*)
Certificate tuple containing in first element the text of the PEM certificate

mc_states.modules.mc_ssl.**selfsigned_ssl_certs**(*domains*, *gen=False*, *as_text=False*)
Maybe Generate and Return SSL certificate and key paths for domain Certicates are generated inside pillar/cloudcontroller/<minionid>. this generates a signed certificate with a generated certificate authority with the name of the current minion.

mc_states.modules.mc_ssl.**settings**()
ssl registry

country country

st st

l l

o organization

cn common name

email mail

certs mapping of COMMON_NAME: (cert_text, key_text, cacert_chain_txt)

- cert_text and cacert_chain_txt contain x509 certs, concatenated
- chain_txt is an empty string if selfsigned or not found
- key we will validated to be a valid private key
- all certs will be validated to be x509 certs

mc_states.modules.mc_ssl.**ssl_certs**(*domains*, ***kw*)
Maybe Generate and Return SSL certificate and key paths for domain Certicates are generated inside pillar/cloudcontroller/<minionid>. this generates a signed certificate with a generated certificate authority with the name of the current minion.

Return a xtuple (cert, key) Cert can contain multiple certs (full chain of certification)

mc_states.modules.mc_ssl.**ssl_chain**(*common_name*, *cert_string*)
Extract the cerfificate and auth chain for a certificate file or string containing one or multiple certificates

Return a tuble:

- The certificate maching the common name If not found, assume the first of the given certs
- The rest of certificates as the ssl chain authentication

mc_states.modules.mc_ssl.**ssl_infos**(*cert_text*, ***kw*)
Get some infos out of a PEM certificates kw can contain default values

issuer cert issuer

subject cert subject

mc_states.modules.mc_ssl.**ssl_key**(*cert_string*)
Extract valid ssl keys from a string or a file & return the first

mc_states.modules.mc_ssl.**ssl_keys**(*cert_string*)
Extract valid ssl keys from a string or a file

mc_supervisor / supervisor functions
mc_states.modules.mc_supervisor.**settings**()
supervisor settings

location installation directory

mc_timezone / timezone registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_timezone
```

mc_states.modules.mc_timezone.**settings**()
timezone registry

tz timezone

mc_tomcat / tomcat functions
mc_states.modules.mc_tomcat.**settings**()
tomcat settings

jdk_ver jdk version to use (will install packages)

java_opts java opts to give to tomcat start

java_home JAVA_HOME of the jdk to use

users mapping of users, roles & password:

```
        {
            'admin': {
              'password': 'admin',
              'roles': ['admin', 'manager'],
            }
        }
```

shutdown_port default shutdown port (8005)

tomcat_user tomcat system user

tomcat_group tomcat system group

address default address to listen on

port default http port (8080)

ssl_port default ssl port (8443)

ajp_port default ajp port (8009)

defaultHost default hostname (localhost)

welcome_files list of files to serve as index (index.{htm,html,jsp})

loglevel_console log level console (FINE)

loglevel_1catalina_org log level for defaults vhosts (FINE)

loglevel_2localhost_org log level for defaults vhosts (FINE)

loglevel_Catalina_localhost_level loglevel for catalina section (INFO)

mc_ulogd / ulogd functions

mc_states.modules.mc_ulogd.**settings**()
ulogd settings

mc_updatedb / updatedb functions

mc_states.modules.mc_updatedb.**settings**()
updatedb settings

mc_usergroup / usergroup registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_usergroup
```

mc_states.modules.mc_usergroup.**get_default_sysadmins**()

mc_states.modules.mc_usergroup.**get_home**(*user*, *home=None*)

mc_states.modules.mc_usergroup.**settings**()
usergroup registry

filesystem.group Group of the special editor group

filesystem.groupId Gid of the special editor group

makina-states.localsettings.users System configured users

makina-states.localsettings.admin.sudoers sudoers (project members)

makina-states.localsettings.defaultSysadmins Priviliegied local users accounts (sysadmin, ubuntu, vagrant)

makina-states.localsettings.admin.sysadmins_keys sysadmins's ssh key to drop inside privilegied accounts

makina-states.localsettings.admin.sysadmin_password sysadmin password

makina-states.localsettings.admin.root_password root password

makina-states.localsettings.admin.absent_keys list of mappings to feed ssh_auth.absent_keys in order to remove ssh keys entries from all managed users

mc_uwsgi / uwsgi functions

mc_states.modules.mc_uwsgi.**config_settings**(*config_name*, *config_file*, *enabled*, ***kwargs*)

Settings for the uwsgi macro

mc_states.modules.mc_uwsgi.**settings**()

uwsgi settings

location installation directory

package list of packages to install uwsgi

configuration directory directory where configuration files are located

run_at_startup "yes" or "no"

verbose "yes" or "no"

print_confnames_in_initd_script_output "yes" or "no"

inherited_config inherited config to fill missing uwsgi parameters

mc_www / www registry If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_modules
```

Documentation of this module is available with:

```
salt '*' sys.doc mc_www
```

mc_states.modules.mc_www.**settings**()

www registry

fastcgi_socket_directory fastcgi socket directory

Other modules

mc_macros / macros helpers

mc_states.modules.mc_macros.**autoinclude**(*reg*, *additional_includes=None*)

Helper to autoload & (un)register services in a top file

mc_states.modules.mc_macros.**construct_registry_configuration**(*name*, *defaults=None*)

Helper to factorise registry mappings

mc_states.modules.mc_macros.**filecache_fun**(*func*, *args=None*, *kwargs=None*, *registry='disk_cache'*, *prefix=None*, *ttl=1*)

Execute a function and store the result in a filebased cache

func func to execute

args positional args to func

kwargs kwargs to func

registry name of the file inside /etc/makina-states

prefix cache key

ttl if 0: do not use cache

mc_states.modules.mc_macros.**get_registry**(*registry_configuration*)

Mangle a registry of activated/unactived states to be run as part of the automatic highstate inclusion.

```
{
    'kind': 'foo',
    'bases': ['localsettings'],
    'defaults': {
       'mastersalt_minion': {'active': False},
       'mastersalt_master': {'active': False},
       'salt_minion': {'active': False},
       'salt_master': {'active': True}
      }
    }
}
```

Will activate the 'makina-states.controllers.salt_master' and deactivate all other states to be automaticly run

EG, for automatic activation of firewalld, lookup in Configs for this key (pillar, grains, reg):

```
makina-states.services.is.firewall.firewalld: true
makina-states.services.firewall.firewalld: true
```

Idea why for the dict containing 'active', i did not choosed a simple boolean is to support other data in the near future.

We return here a registry in the form:

```
{
    'kind': 'foo',
    'bases': ['localsettings'],
    'states_pref': 'makina-states.foo',
    'grains_pref': 'makina-states.foo',
    'activated': {'salt_master': {'active': True}},
    'unactivated': {
       'mastersalt_minion': {'active': False},
       'mastersalt_master': {'active': False},
       'salt_minion': {'active': False},
    },
    'defaults': {
       'mastersalt_minion': {'active': False},
       'mastersalt_master': {'active': False},
       'salt_minion': {'active': False},
       'salt_master': {'active': True}
      }
    }
}
```

mc_states.modules.mc_macros.**is_active**(*registry*, *name*)

Is the queried service active in the registry

mc_states.modules.mc_macros.**is_item_active**(*registry_name*, *item*, *default_status=False*, *grains_pref=None*, *force=False*)

Look in pillar/grains/localconfig for registry activation status

mc_states.modules.mc_macros.**pack_dump_local_registry**(*registry*)
encode in a file using msgpack backend

mc_states.modules.mc_macros.**register**(*kind*, *slss*, *data=None*, *suf=''*)
Register a/some service(s) in the local registry

mc_states.modules.mc_macros.**unregister**(*kind*, *slss*, *data=None*, *suf=''*)
Unregister a/some service(s) in the local registry

mc_states.modules.mc_macros.**update_local_registry**(*registry_name*, *params*, *registry_format='yaml'*)
Alias to update_local_registry

mc_states.modules.mc_macros.**update_registry_params**(*registry_name*, *params*, *registry_format='yaml'*)
Update the desired local registry

mc_utils / Some usefull small tools

mc_states.modules.mc_utils.**assert_good_grains**(*grains*)
' no time to search/debug why, but sometimes grains dict is empty depending on the call context grains loading bug retriggered (i fixed once, do not remember where, FU SALT ...

mc_states.modules.mc_utils.**cache_check**(**args*, ***kw*)
Wrapper for `invalidate_memoize_cache()` to set __opts__

mc_states.modules.mc_utils.**copy_dictupdate**(*dict1*, *dict2*)
Similar to dictupdate but with deepcopy of two merged dicts first.

mc_states.modules.mc_utils.**cyaml_dump**(**args*, ***kw*)
Retro compat to `mc_states.modules.mc_dump.cyaml_dump()`

mc_states.modules.mc_utils.**cyaml_load**(**args*, ***kw*)
Retro compat to `mc_states.modules.mc_dump.cyaml_load()`

mc_states.modules.mc_utils.**defaults**(*prefix*, *datadict*, *ignored_keys=None*, *overridden=None*, *noresolve=False*, *firstcall=True*)
Magic defaults settings configuration getter

- Get the "prefix" value from the configuration (pillar/grain)
- Then overrides or append to it with the corresponding key in the given "datadict" if value is a dict or a list.
  - If we get from pillar/grains/local from the curent key in the form: "{prefix}-overrides: it overrides totally the original value.
  - if the datadict contains a key "{prefix}-append and the value is a list, it appends to the original value
- **If the datadict contains a key "{prefix}":**
  - If a list: override to the list the default list in conf
  - Elif a dict: update the default dictionnary with the one in conf
  - Else take that as a value if the value is not a mapping or a list

mc_states.modules.mc_utils.**dictupdate**(*dict1*, *dict2*)
Merge two dictionnaries recursively

test:

```
salt '*' mc_utils.dictupdate '{foobar:
            {toto: tata, toto2: tata2},titi: tutu}'
            '{bar: toto, foobar: {toto2: arg, toto3: arg2}}'
----------
```

```
    bar:
        toto
    foobar:
        ----------
        toto:
            tata
        toto2:
            arg
        toto3:
            arg2
    titi:
        tutu
```

mc_states.modules.mc_utils.**file_read**(*fic*)
read the content a file

mc_states.modules.mc_utils.**format_resolve**(*value*, *original_dict=None*, *this_call=0*, *topdb=False*, ***kwargs*)
Resolve a dict of formatted strings, mappings & list to a valued dict Please also read the associated test:

```
    {"a": ["{b}", "{c}", "{e}"],
     "b": 1,
     "c": "{d}",
     "d": "{b}",
     "e": "{d}",
    }

    ====>
    {"a": ["1", "1", "{e}"],
     "b": 1,
     "c": "{d}",
     "d": "{b}",
     "e": "{d}",
    }
```

mc_states.modules.mc_utils.**generate_stored_password**(*key*, *length=None*, *force=False*, *value=None*)
Generate and store a password. At soon as one is stored with a specific key, it will never be renegerated unless you set force to true.

mc_states.modules.mc_utils.**get**(*key*, *default=''*, *local_registry=None*, *registry_format='pack'*, *delimiter=<class 'DEFAULT_TARGET_DELIM'>*)
Same as 'config.get' but with different retrieval order.

This routine traverses these data stores in this order:

- Local minion config (opts)
- Minion's pillar
- **Dict:**
  - passed in local_registry argument
  - or automaticly loaded global registries
- Minion's grains
- Master config

CLI Example:

```
    salt '*' mc_utils.get pkg:apache
```

mc_states.modules.mc_utils.**get_local_cache**(**args*)
Wrapper for `get_local_cache()`

mc_states.modules.mc_utils.**get_mc_server**(**args*, ***kw*)
Wrapper for `get_local_cache()`

mc_states.modules.mc_utils.**get_uniq_keys_for**(*prefix*)
Return keys for prefix:

- if prefix is in conf
- All other keys of depth + 1

With makina.foo prefix:

- returns makina.foo
- returns makina.foo.1
- dont returns makina.foo.1.1
- dont returns makina
- dont returns makina.other

mc_states.modules.mc_utils.**hash**(*string*, *typ='md5'*, *func='hexdigest'*)
Return the hash of a string CLI Examples:

```
    salt-call --local mc_utils.hash foo
    salt-call --local mc_utils.hash foo md5
    salt-call --local mc_utils.hash foo sha1
    salt-call --local mc_utils.hash foo sha224
    salt-call --local mc_utils.hash foo sha256
    salt-call --local mc_utils.hash foo sha384
    salt-call --local mc_utils.hash foo sha512
```

mc_states.modules.mc_utils.**invalidate_memoize_cache**(**args*, ***kw*)
Wrapper for `invalidate_memoize_cache()` to set __opts__

mc_states.modules.mc_utils.**is_a_bool**(*value*)
is the value a bool

mc_states.modules.mc_utils.**is_a_complex**(*value*)
is the value a complex

mc_states.modules.mc_utils.**is_a_dict**(*value*)
is the value a dict

mc_states.modules.mc_utils.**is_a_float**(*value*)
is the value a float

mc_states.modules.mc_utils.**is_a_int**(*value*)
is the value an int

mc_states.modules.mc_utils.**is_a_list**(*value*)
is the value a list

mc_states.modules.mc_utils.**is_a_long**(*value*)
is the value a long

mc_states.modules.mc_utils.**is_a_number**(*value*)
is the value a number

mc_states.modules.mc_utils.**is_a_set**(*value*)
is the value a set

mc_states.modules.mc_utils.**is_a_str**(*value*)
is the value a stirng

mc_states.modules.mc_utils.**is_a_tuple**(*value*)
is the value a tuple

mc_states.modules.mc_utils.**is_iter**(*value*)
is the value iterable (list, set, dict tuple)

mc_states.modules.mc_utils.**iyaml_dump**(**args*, ***kw*)
Retro compat to `mc_states.modules.mc_dump.iyaml_dump()`

mc_states.modules.mc_utils.**json_dump**(**args*, ***kw*)
Retro compat to `mc_states.modules.mc_dump.old_json_dump()`

mc_states.modules.mc_utils.**json_load**(**args*, ***kw*)
Retro compat to `mc_states.modules.mc_dump.json_load()`

mc_states.modules.mc_utils.**list_cache_keys**(**args*, ***kw*)
Wrapper for `list_cache_keys()` to set __opts__

mc_states.modules.mc_utils.**local_minion_id**(*force=False*)
search in running config root then in well known config mastersalt root then in well known config salt root then use regular salt function

mc_states.modules.mc_utils.**magicstring**(*thestr*)
Convert any string to UTF-8 ENCODED one

mc_states.modules.mc_utils.**manage_file**(*name*, ***kwargs*)
Easier wrapper to file.manage_file

mc_states.modules.mc_utils.**memoize_cache**(**args*, ***kw*)
Wrapper for `memoize_cache()` to set __opts__

CLI Examples:

```
mastersalt-call -lall mc_pillar.memoize_cache test.ping
```

mc_states.modules.mc_utils.**msgpack_dump**(**args*, ***kw*)
Retro compat to `mc_states.modules.mc_dump.old_msgpack_dump()`

mc_states.modules.mc_utils.**msgpack_load**(**args*, ***kw*)
Retro compat to `mc_states.modules.mc_dump.msgpack_load()`

mc_states.modules.mc_utils.**msr**()
get salt root from either pillar or opts (minion or master)

mc_states.modules.mc_utils.**nyaml_dump**(**args*, ***kw*)
Retro compat to `mc_states.modules.mc_dump.yaml_dump()`

mc_states.modules.mc_utils.**old_yaml_dump**(**args*, ***kw*)
Retro compat to `mc_states.modules.mc_dump.old_yaml_dump()`

mc_states.modules.mc_utils.**output**(*mapping*, *raw=False*, *outputter='highstate'*)
This return a formatted output

mc_states.modules.mc_utils.**purge_memoize_cache**(**args*, ***kw*)
Wrapper for `invalidate_memoize_cache()` to set __opts__

mc_states.modules.mc_utils.**register_memcache_first**(*pattern*)
Wrapper for `invalidate_memoize_cache()` to set __opts__

mc_states.modules.mc_utils.**remove_cache_entry**(**args*, ***kw*)
Wrapper for `remove_cache_entry()` to set __opts__

mc_states.modules.mc_utils.**remove_entry**(**args*, ***kw*)
Wrapper for `remove_cache_entry()` to set __opts__

mc_states.modules.mc_utils.**salt_root**()
get salt root from either pillar or opts (minion or master)

mc_states.modules.mc_utils.**test_cache**(*ttl=120*)
.

mc_states.modules.mc_utils.**traverse_dict**(*data*, *key*, *delimiter=<class 'DEFAULT_TARGET_DELIM'>*)
Handle the fact to traverse dicts with '.' as it was an old default and makina-states relies a lot on it

This restore the old behavior of something that can be traversed

makina-states.foo:

bar: c: true

can be traversed with makina-states.foo.bar.c

mc_states.modules.mc_utils.**uncached_get**(*key*, *default=''*, *local_registry=None*, *registry_format='pack'*, *delimiter=<class 'DEFAULT_TARGET_DELIM'>*)
Same as 'config.get' but with different retrieval order.

This routine traverses these data stores in this order:

- Local minion config (opts)
- Minion's pillar
- **Dict:**
  - passed in local_registry argument
  - or automaticly loaded global registries
- Minion's grains
- Master config

CLI Example:

```
salt '*' mc_utils.get pkg:apache
```

mc_states.modules.mc_utils.**unix_crypt**(*passwd*)
Encrypt the stringed password in the unix crypt format (/etc/shadow)

mc_states.modules.mc_utils.**update_no_list**(*dest*, *upd*, *recursive_update=True*)
Recursive version of the default dict.update

Merges upd recursively into dest But instead of merging lists, it overrides them from target dict

mc_states.modules.mc_utils.**yaml_dump**(**args*, ***kw*)
Retro compat to `mc_states.modules.mc_dump.old_yaml_dump()`

mc_states.modules.mc_utils.**yaml_load**(**args*, ***kw*)
Retro compat to `mc_states.modules.mc_dump.yaml_load()`

mc_states.modules.mc_utils.**yencode**(**args*, ***kw*)
Retro compat to `mc_states.modules.mc_dump.yencode()`

States modules

mc_apache / apache states If you alter this module and want to test it, do not forget to deploy it on minion using:

```
salt '*' saltutil.sync_states
```

If you use this state as a template for a new custom state do not forget to use to get this module included in salt modules.

To comment

```
apache-main-conf:
  makina-states.apache.deployed:
    - version: 2.2
    - log_level: debug
```

Or using the "names:" directive, you can put several names for the same IP. (Do not try one name with space-separated values).

```
server1:
  host.present:
    - ip: 192.168.0.42
    - names:
      - server1
      - florida
```

mc_states.states.mc_apache.**deployed**(*name*, **args*, ***kwargs*)
DEPRECATED

mc_states.states.mc_apache.**exclude_module**(*name*, *modules*)
Soft disable one or mode apache modules

name ignored

modules list or comma separated list of modules

mc_states.states.mc_apache.**include_module**(*name*, *modules*)
Soft enable one or mode apache modules

name ignored

modules list or comma separated list of modules

mc_git / Interaction with Git repositories The difference or to be more precise, the only addition of using git.latest is that we do a merge –ff-only when pulling in case of errors to be a bit more fairful on updates

```
https://github.com/saltstack/salt.git:
  mc_git.latest:
    - rev: develop
    - target: /tmp/salt
```

mc_states.states.mc_git.**latest**(**args*, ***kwargs*)
Compat wrapper

mc_states.states.mc_git.**old_latest**(*name*, *rev=None*, *target=None*, *runas=None*, *user=None*, *force=None*, *force_checkout=False*, *submodules=False*, *mirror=False*, *bare=False*, *remote_name='origin'*, *always_fetch=False*, *identity=None*, *onlyif=False*, *unless=False*, *firstrun=True*)
Make sure the repository is cloned to the given directory and is up to date Thin wrapper to git.latest that also makes a git merge –only-ff to merge unharmful commits without failling hard

name Address of the remote repository as passed to "git clone"

rev The remote branch, tag, or revision ID to checkout after clone / before update

target Name of the target directory where repository is about to be cloned

runas Name of the user performing repository management operations

Deprecated since version 0.17.0.

user Name of the user performing repository management operations

New in version 0.17.0.

force Force git to clone into pre-existing directories (deletes contents)

force_checkout Force a checkout even if there might be overwritten changes (Default: False)

submodules Update submodules on clone or branch change (Default: False)

mirror True if the repository is to be a mirror of the remote repository. This implies bare, and thus is incompatible with rev.

bare True if the repository is to be a bare clone of the remote repository. This is incompatible with rev, as nothing will be checked out.

remote_name defines a different remote name. For the first clone the given name is set to the default remote, else it is just a additional remote. (Default: 'origin')

always_fetch If a tag or branch name is used as the rev a fetch will not occur until the tag or branch name changes. Setting this to true will force a fetch to occur. Only applies when rev is set. (Default: False)

identity A path to a private key to use over SSH

onlyif A command to run as a check, run the named command only if the command passed to the `onlyif` option returns true

unless A command to run as a check, only run the named command if the command passed to the `unless` option returns false

mc_postgres_database / Wrapper to automaticly set the rigth pgsql to attack

`mc_states.states.mc_postgres_database.`**`absent`**(*name*, **args*, ***kw*)

Absent wrapper

`mc_states.states.mc_postgres_database.`**`present`**(*name*, **args*, ***kw*)

Present wrapper

mc_postgres_extension / Wrapper to automaticly set the rigth pgsql to attack

`mc_states.states.mc_postgres_extension.`**`absent`**(*name*, **args*, ***kw*)

Absent wrapper

`mc_states.states.mc_postgres_extension.`**`present`**(*name*, **args*, ***kw*)

Present wrapper

mc_postgres_group

mc_posrgres_group / Wrapper to automaticly set the rigth pgsql to attack

`mc_states.states.mc_postgres_group.`**`absent`**(*name*, **args*, ***kw*)

Absent wrapper

`mc_states.states.mc_postgres_group.`**`present`**(*name*, **args*, ***kw*)

Present wrapper

mc_postgres_user / Wrapper to automaticly set the rigth pgsql to attack

mc_states.states.mc_postgres_user.**absent**(*name*, **args*, ***kw*)

Absent wrapper

mc_states.states.mc_postgres_user.**present**(*name*, **args*, ***kw*)

Present wrapper

mc_proxy / Dummy state generation

mc_states.states.mc_proxy.**hook**(*name*, *changes=None*, ***kw*)

State that will always return ret, use that for orchestration purpose

name name of dummy state

mc_states.states.mc_proxy.**mod_watch**(*name*, ***kwargs*)

Execute a dummy state in case of watcher changes

mc_registry / local registries Makina-States local registries management

mc_states.states.mc_registry.**update**(*name*, *params*, ***kw*)

Active or deactive a param in the named makina-states loccalregistry

bacula / Management of bacula File Daemon Configuration Configure Bacula file daemon to allow connections from a particular Bacula director, set password credentials, as well as the file daemon name and port that it runs on. Configure the messages that get returned to the director.

```
/etc/bacula/bacula-fd.conf:
  bacula:
    - fdconfig
    - dirname: bacula-dir
    - dirpasswd: test1234
    - fdname: bacula-fd
    - fdport: 9102
    - messages: bacula-dir = all, !skipped, !restored
```

mc_states.states.bacula.**fdconfig**(*name*, *dirname=None*, *dirpasswd=None*, *fdname=None*, *fdport=None*, *messages=None*)

Configure a bacula file daemon

dirname The name of the director that is allowed to connect to the file daemon.

dirpasswd The password that the director must use to successfully connect to the file daemon.

fdname The name of the file daemon

fdport The port that the file daemon should run on

messages Define how and what messages to send to a director.

makina-states grains modules

Makina custom grains

makina.upstart true if using upstart

makina.lxc true if inside an lxc container

makina.docker true if inside a docker container

makina.devhost_num devhost num if any

mc_states.grains.makina_grains.**get_makina_grains**()

Runners modules

mc_api Convenient functions to use a salt infra as an api Internal module used as api.

mc_states.runners.mc_api.**apply_sls**(*slss*, *concurrent=True*, **a*, ***kwargs*)
args

slss one or list of sls

output output to stdout

ret mc_state results dict

salt_target target

sls_kw useful to give pillar:

```
(**{sls_kw: {pillar: {1:2}}})
```

mc_states.runners.mc_api.**cli**(**args*, ***kwargs*)
Correctly forward salt globals to a regular python module

mc_states.runners.mc_api.**destroy**(*id_*, *sshport=22*, *sshhost=None*, *destroy=False*, *remove_key=True*, ***kwargs*)
Alias to remove

mc_states.runners.mc_api.**remove**(*id_*, *sshport=22*, *sshhost=None*, *destroy=False*, *remove_key=True*, ***kwargs*)
Remove salt linking to a host (common code) This requires ssh access

mc_lxc Jobs for lxc managment

mc_states.runners.mc_lxc.**sync_images**(*only=None*, *force=False*, *output=True*, *force_output=False*, *__salt__from_exec=None*)
Sync the 'makina-states' image to all configured LXC hosts minions

WARNING it checks .ms_version inside the rootfs of the LXC if this one didnt change, images wont be synced

Configuration:

mc_lxc / lxc registry settings:

images_root master filesystem root to lxc containers

images list of image to sync to lxc minions

containers all minion targets will be synced with that list of images

mc_cloud_controller runner

mc_states.runners.mc_cloud_controller.**configure_node**(*target*, *no_saltify=True*, *no_vms=True*, *no_controller=True*, **a*, ***kw*)
Shortcut to only configure a compute node

mc_states.runners.mc_cloud_controller.**configure_vm**(*target*, *no_controller=True*, *no_compute_nodes=True*, *no_saltify=True*, **a*, ***kw*)
Shortcut to only configure vm

mc_states.runners.mc_cloud_controller.**deploy**(*output=True*, *ret=None*)
Prepare cloud controller configuration can also apply per virtualization type configuration

mc_states.runners.mc_cloud_controller.**dns_conf**(*output=True*, *ret=None*)
Prepare cloud controller dns (BIND) server

mc_states.runners.mc_cloud_controller.**exists**(*name*)
return true if the 'target' is already provisionned

mc_states.runners.mc_cloud_controller.**orchestrate**(*only=None*, *only_vms=None*, *skip=None*, *skip_vms=None*, *no_controller=False*, *no_dns_conf=False*, *no_configure=False*, *no_saltify=False*, *no_provision=False*, *no_vms=False*, *no_compute_nodes=False*, *no_post_provision=False*, *no_vms_post_provision=False*, *output=True*, *refresh=True*, *ret=None*)
install controller, compute node, vms & run postdeploy

no_configure skip configuring the cloud controller

skip list of compute nodes to skip

skip_vms list of vm to skip

only explicit list of compute nodes to deploy

only_vms explicit list of vm to deploy

no_provision skip compute node & vm provision

no_compute_nodes skip configuration of compute nodes

no_vms do not provision vms

no_post_provision do not post provision compute nodes

no_vms_post_provision do not post provision vms

mc_states.runners.mc_cloud_controller.**post_configure**(*output=True*)
post configuration

mc_states.runners.mc_cloud_controller.**prepare_controller**(*no_saltify=True*, *no_compute_nodes=True*, *no_vms=True*, **a*, ***kw*)
Shortcut to prepare the controller

(DNS; VT orchestration)

mc_states.runners.mc_cloud_controller.**remove**(*node_name*, *destroy=False*, *remove_key=True*, *only_stop=False*, ***kwargs*)
Remove a node NOTE: only lxc vms are supported for now Actually this consists in:

•disabling crons

•disabling services

•unlinking the key from mastersalt

mc_states.runners.mc_cloud_controller.**report**(**a*, ***kw*)
Alias to mc_cloud_compute_node.report

mc_states.runners.mc_cloud_controller.**run_vt_hook**(*hook_name*, *ret=None*, *target=None*, *vts=None*, *output=True*, **args*, ***kwargs*)

Run an hook for a special vt on a controller, or a compute node or a vm

mc_states.runners.mc_cloud_controller.**saltify_node**(*target*, *no_controller=True*, *no_compute_nodes=True*, *no_vms=True*, **a*, ***kw*)

Shortcut to only saltify something

mc_cloud_compute_node runner

mc_states.runners.mc_cloud_compute_node.**configure_firewall**(*target*, *ret=None*, *output=True*)

shorewall configuration

mc_states.runners.mc_cloud_compute_node.**configure_host**(*target*, *ret=None*, *output=True*)

shorewall configuration

mc_states.runners.mc_cloud_compute_node.**configure_hostsfile**(*target*, *ret=None*, *output=True*)

local dns configuration

mc_states.runners.mc_cloud_compute_node.**configure_network**(*target*, *ret=None*, *output=True*)

install network configuration

mc_states.runners.mc_cloud_compute_node.**configure_prevt**(*target*, *ret=None*, *output=True*)

install all prevt steps

mc_states.runners.mc_cloud_compute_node.**configure_reverse_proxy**(*target*, *ret=None*, *output=True*)

haproxy configuration

mc_states.runners.mc_cloud_compute_node.**configure_sshkeys**(*target*, *ret=None*, *output=True*)

drop the compute node ssh key

mc_states.runners.mc_cloud_compute_node.**configure_sslcerts**(*target*, *ret=None*, *output=True*)

deploy SSL certificates on compute node

mc_states.runners.mc_cloud_compute_node.**deploy**(*target*, *output=True*, *ret=None*, *hooks=True*, *pre=True*, *post=True*)

Prepare cloud controller configuration can also apply per virtualization type configuration

mc_states.runners.mc_cloud_compute_node.**filter_compute_nodes**(*nodes*, *skip*, *only*)

filter compute nodes to run on

mc_states.runners.mc_cloud_compute_node.**install_vts**(*target*, *ret=None*, *output=True*)

install all virtual types to be ready to host vms

mc_states.runners.mc_cloud_compute_node.**orchestrate**(*skip=None*, *skip_vms=None*, *only=None*, *only_vms=None*, *no_compute_nodes=False*, *no_provision=False*, *no_post_provision=False*, *no_vms_post_provision=False*, *no_vms=False*, *output=True*, *refresh=True*, *ret=None*)

Orchestrate the whole cloud deployment. In this order:

- provision compute nodes minus the skipped one and limiting to the 'only' if any
- provision vms minus the skipped one and limiting to the 'only_compute_vm' if any. If the vms are to be hosted on a failed host, they will be skipped
- post provision compute nodes
- post provision vms

skip list or comma separated string of compute node to skip (will skip contained vms too)

only list or comma separated string of compute node If set, it will only provision those compute nodes and contained vms

no_provision do not run the compute nodes provision

no_post_provision do not run the compute nodes post provision

no_compute_nodes skip configuration of compute nodes

skip_vms list or comma separated string of vms to skip

only_vms list or comma separated string of vms. If set, it will only provision those vms

no_vms do not run the vm provision

no_vms_post_provision do not run the vms post provision

mc_states.runners.mc_cloud_compute_node.**post_deploy**(*target*, *ret=None*, *output=True*)
Prepare cloud controller configuration can also apply per virtualization type configuration

mc_states.runners.mc_cloud_compute_node.**post_provision_compute_nodes**(*skip=None*, *only=None*, *output=True*, *refresh=False*, *ret=None*)
post provision all compute nodes

mc_states.runners.mc_cloud_compute_node.**provision_compute_nodes**(*skip=None*, *only=None*, *no_compute_nodes=False*, *output=True*, *refresh=True*, *ret=None*)
provision compute nodes

skip list or comma separated string of compute node to skip (will skip contained vms too)

only list or comma separated string of compute node If set, it will only provision those compute nodes and contained vms

mc_states.runners.mc_cloud_compute_node.**reconfigure_front**(*target*, *ret=None*, *output=True*)
Small hook to reconfigure the reverse proxy part of a compute node, meaned to be used via the CLI

mc_states.runners.mc_cloud_compute_node.**report**(*targets*, *ret=None*, *refresh=False*, *output=True*)
Parse all reachable compute nodes and vms and regenerate the local configuration registries concerning cloud deployment

mc_states.runners.mc_cloud_compute_node.**run_vt_hook**(*hook_name*, *target*, *ret=None*, *vts=None*, *output=True*, **args*, ***kwargs*)
Difference with cloud controller bare one is that here we have the target argument mandatory

mc_states.runners.mc_cloud_compute_node.**upgrade_vts**(*target*, *ret=None*, *output=True*)
upgrade all virtual types to be ready to host vms

mc_cloud_vm runner

mc_states.runners.mc_cloud_vm.**destroy**(*vm*, ***kwargs*)
Alias to remove

mc_states.runners.mc_cloud_vm.**orchestrate**(*compute_node=None*, *skip=None*, *only=None*, *output=True*, *refresh=False*, *ret=None*)
install all compute node vms

```
mastersalt-run mc_cloud_vm.orchestrate t.dom.fr
mastersalt-run mc_cloud_vm.orchestrate t.dom.fr only=['foo.domain.tld']
mastersalt-run mc_cloud_vm.orchestrate t.dom.fr skip=['foo.domain.tld']
```

mc_states.runners.mc_cloud_vm.**post_provision**(*vm*, *ret=None*, *output=True*)
post provision a vm

compute_node where to act

vt virtual type

vm vm to spawn

steps list or comma separated list of steps Default:

```
['ping', 'post_provision_hook']
```

```
mastersalt-run -lall mc_cloud_vm.post_provision foo.domain.tld
```

mc_states.runners.mc_cloud_vm.**post_provision_vms**(*cn*, *skip=None*, *only=None*, *ret=None*, *output=True*, *refresh=False*)
post provision all or selected compute node vms

```
mastersalt-run -lall mc_cloud_vm.post_provision_vms host1.domain.tld
mastersalt-run -lall mc_cloud_vm.post_provision_vms                host1.domain.tld only=['foo.
mastersalt-run -lall mc_cloud_vm.post_provision_vms                host1.domain.tld skip=['foo2
```

mc_states.runners.mc_cloud_vm.**provision**(*vm*, *steps=None*, *ret=None*, *output=True*)
provision a vm

compute_node where to act

vt virtual type

vm vm to spawn

steps list or comma separated list of steps

```
mastersalt-run -lall mc_cloud_vm.provision foo.domain.tld
```

mc_states.runners.mc_cloud_vm.**provision_vms**(*compute_node=None*, *skip=None*, *only=None*, *ret=None*, *output=True*, *refresh=False*)
Provision all or selected vms on a compute node

```
mastersalt-run -lall mc_cloud_vm.provision_vms host1.domain.tld
mastersalt-run -lall mc_cloud_vm.provision_vms                  host1.domain.tld only=['foo.domai
mastersalt-run -lall mc_cloud_vm.provision_vms                  only=['foo.domain.tld']
mastersalt-run -lall mc_cloud_vm.provision_vms                  host1.domain.tld skip=['foo2.doma
```

mc_states.runners.mc_cloud_vm.**step**(*vm*, *step*, *ret=None*, *output=True*)
Execute a step on a VM node

mc_states.runners.mc_cloud_vm.**vm_initial_highstate**(*vm*, *ret=None*, *output=True*)
Run the initial highstate, this step will run only once and will further check for the existence of <saltroot>/makina-states/.initial_hs file

compute_node where to act

vm vm to run highstate on

```
mastersalt-run -lall mc_cloud_vm.vm_initial_highstate foo.domain.tld
```

mc_states.runners.mc_cloud_vm.**vm_markers**(*vm*, *ret=None*, *output=True*)
install markers at / of the vm for proxified access

compute_node where to act

vm vm to install grains into

mc_states.runners.mc_cloud_vm.**vm_ping**(*vm*, *ret=None*, *output=True*)
ping a specific vm on a specific compute node

vm vm to ping

```
mastersalt-run -lall mc_cloud_vm.vm_ping foo.domain.tld
```

mc_states.runners.mc_cloud_vm.**vm_preprovision**(*vm*, *ret=None*, *output=True*)
Run the preprovision:

For performance reasons, this is a merge of steps

- markers
- sshkeys

```
mastersalt-run -lall mc_cloud_vm.vm_preprovision foo.domain.tld
```

mc_states.runners.mc_cloud_vm.**vm_sshkeys**(*vm*, *ret=None*, *output=True*)
Install controller ssh keys for user too on this specific vm

compute_node where to act

vm vm to install keys into

```
mastersalt-run -lall mc_cloud_vm.vm_sshkeys foo.domain.tld
```

mc_cloud_lxc runner

exception mc_states.runners.mc_cloud_lxc.**MoveError**
.

mc_states.runners.mc_cloud_lxc.**configure_images**(*target*, *ret=None*, *output=True*)
configure all images templates

mc_states.runners.mc_cloud_lxc.**configure_install_vt**(*target*, *ret=None*, *output=True*)
install lxc

mc_states.runners.mc_cloud_lxc.**destroy**(*vm*, ***kwargs*)
Alias to remove

mc_states.runners.mc_cloud_lxc.**install_vt**(*target*, *output=True*)
install & configure lxc

mc_states.runners.mc_cloud_lxc.**post_deploy_controller**(*output=True*)
Prepare cloud controller LXC configuration

mc_states.runners.mc_cloud_lxc.**post_post_deploy_compute_node**(*target*, *output=True*)
post deployment hook for controller

mc_states.runners.mc_cloud_lxc.**remove**(*vm*, *destroy=False*, *only_stop=False*, ***kwargs*)
Remove a container

mc_states.runners.mc_cloud_lxc.**sync_images**(*target*, *output=True*, *ret=None*)
sync images on target

mc_states.runners.mc_cloud_lxc.**upgrade_vt**(*target*, *ret=None*, *output=True*)
Upgrade LXC hosts This will reboot all containers upon lxc upgrade Containers are marked as being rebooted, and unmarked as soon as this script unmark explicitly them to be done.

mc_states.runners.mc_cloud_lxc.**vm_hostsfile**(*vm*, *ret=None*, *output=True*)
manage vm /etc/hosts to add link to host

```
mastersalt-run -lall mc_cloud_lxc.vm_hostsfile foo.domain.tld
```

mc_states.runners.mc_cloud_lxc.**vm_initial_setup**(*vm*, *ret=None*, *output=True*)
set initial password at least

```
mastersalt-run -lall mc_cloud_lxc.vm_initial_setup foo.domain.tld
```

mc_states.runners.mc_cloud_lxc.**vm_preprovision**(*vm*, *ret=None*, *output=True*)
Shortcut to run all preprovision steps like initial_setup or hostfiles

```
mastersalt-run -lall mc_cloud_lxc.vm_grains foo.domain.tld
```

mc_states.runners.mc_cloud_lxc.**vm_reconfigure**(*vm*, *ret=None*, *output=True*, *rootfs=None*, *force=False*)
Reconfigure the vm if neccessary

```
mastersalt-run -lall mc_cloud_lxc.vm_reconfigure_net foo.domain.tld
```

mc_states.runners.mc_cloud_lxc.**vm_spawn**(*vm*, *ret=None*, *output=True*, *force=False*)
spawn the vm

```
mastersalt-run -lall mc_cloud_lxc.vm_spawn foo.domain.tld
```

mc_cloud_kvm runner

mc_states.runners.mc_cloud_kvm.**configure_install_vt**(*target*, *ret=None*, *output=True*)
install kvm

mc_states.runners.mc_cloud_kvm.**install_vt**(*target*, *output=True*)
install & configure kvm

mc_states.runners.mc_cloud_kvm.**post_deploy_controller**(*output=True*)
Prepare cloud controller KVM configuration

mc_states.runners.mc_cloud_kvm.**post_post_deploy_compute_node**(*target*, *output=True*)
post deployment hook for controller

mc_states.runners.mc_cloud_kvm.**sync_images**(*target*, *output=True*, *ret=None*)
sync images on target NOT IMPLEMENTED

mc_states.runners.mc_cloud_kvm.**upgrade_vt**(*target*, *ret=None*, *output=True*)
Upgrade KVM hosts NOT IMPLEMENTED

mc_cloud_saltify runner

mc_states.runners.mc_cloud_saltify.**filter_compute_nodes**(*nodes*, *skip*, *only*)
filter compute nodes to run on

mc_states.runners.mc_cloud_saltify.**orchestrate**(*only=None*, *skip=None*, *ret=None*, *output=True*, *refresh=False*)
Parse saltify settings to saltify all targets

output display output

only specify explicitly which hosts to provision among all avalaible ones

skip hosts to skip

refresh refresh pillar

mc_states.runners.mc_cloud_saltify.**saltify**(*name*, *output=True*, *ret=None*)
Saltify a specific target

Api modules

Convenient functions to use a salt infra as an api

class mc_states.saltapi.**ImgStepError**
.

class mc_states.saltapi.**NoRegistryLoaderFound**
.

exception mc_states.saltapi.**PortConflictError**
.

class mc_states.saltapi.**RenderError**(*msg*, *original=None*, *ret=None*, **args*, ***kwargs*)
.

class mc_states.saltapi.**SSHCommandFailed**(*message*, *exec_ret=<object object>*)
.

class mc_states.saltapi.**SSHCommandFinished**(*message*, *exec_ret=<object object>*)
.

class mc_states.saltapi.**SSHCommandTimeout**(*message*, *exec_ret=<object object>*)
.

class mc_states.saltapi.**SSHExecError**(*message*, *exec_ret=<object object>*)
.

class mc_states.saltapi.**SSHInterruptError**(*message*, *exec_ret=<object object>*)
.

class mc_states.saltapi.**SSHLoginError**(*message*, *exec_ret=<object object>*)
.

class mc_states.saltapi.**SSHTimeoutError**(*message*, *exec_ret=<object object>*)
.

class mc_states.saltapi.**SSHTransferFailed**(*message*, *exec_ret=<object object>*)
.

class mc_states.saltapi.**SSHVtError**(*message*, *exec_ret=<object object>*)
.

class mc_states.saltapi.**SaltCallFailure**(*message*, *exec_ret=<object object>*)
.

class mc_states.saltapi.**TransformError**(*msg*, *original=None*, *ret=None*, **args*, ***kwargs*)
.

mc_states.saltapi.**client**(*fun*, **args*, ***kw*)
Execute a salt function on a specific minion using the salt api. This will set automatic timeouts for well known functions. This will also call well known api calls for a specific time.

Special kwargs:

salt_cfgdir alternative configuration file directory

salt_cfg alternative configuration file

salt_target target to exec things on

salt_timeout timeout for jobs

salt_job_poll poll interval to wait for job finish result

salt_ttl cache ttl either 2 seconds or see __CACHED_FUNS preselector

mc_states.saltapi.**concat_res_or_rets**(*ret*, *cret=None*, *result_keys=None*, *output_keys=None*, *dict_keys=None*, *omit=None*)
Convenient and magical way to merge 2 structures or strings for usage in salt functions.

concatenate string with string join them (separated with a newline)

concatenate string with dict: append all output keys from dict in the string separated by a new line and prefixed by the output key identfier

concatenate dict with string: concatenate (with newlineà) the string in all output keys

concatenate dict with dict: merge corresponding keys in an intelligent way:

- result from ret is setted to false if cret's one is setted to False
- merge output keys (separate with newline)
- merge dict keys by updating or creating the corresponding key in ret from cret

```
>>> from collections import OrderedDict
>>> from mc_states.saltapi import concat_res_or_rets
>>> concat_res_or_rets({}, {'result': False})
{'result': False}
>>> concat_res_or_rets({'result': True}, {'result': False})
{'result': False}
>>> concat_res_or_rets('oo', {'stdout': 'a', 'stderr': 'b'})
'oo\nSTDOUT: a\nSTDERR: b'
>>> concat_res_or_rets('a', 'b')
'a\nb'
>>> concat_res_or_rets(OrderedDict([('stdout', 'a'), ('stderr', 'b')]),
...                    'de')
OrderedDict([('stdout', 'a\nde'), ('stderr', 'b'), ('output', 'de')])
>>> concat_res_or_rets(OrderedDict([('stdout', 'a'), ('stderr', 'b')]),
...                    {'stdout': 'c', 'stderr': 'd'})
OrderedDict([('stdout', 'a\nc'), ('stderr', 'b\nd')])
>>> concat_res_or_rets({'changes': {1: 2, 3: 4, 5: 6}},
```

```
    ...                     {'changes': {1: 3, 3: 4}})
    {'changes': {1: 3, 3: 4, 5: 6}}
```

mc_states.saltapi.**get_local_client**(*cfgdir=None*, *cfg=None*, *conn=None*, ***kwargs*)
Get a local client

cfgdir/cfg args given to localclient

Utilities functions (deprecated location)

2.1.3 Formulaes & macros helpers reference

Makina-states Formulaes

Localsettings

Those formulaes will configure basic configuration on your host that is/are not tied to a service

For example; writing something in /etc is a good catch for a localsettings states

We let the user have a word on the final local settings which are activated This can be customized by putting keys either in pillar or in grains in the form: 'makina-states.localsettings.<statename>'

EG: to disable the default vim configuration, either set a grain or a pillar value:

```
makina-states.localsettings.vim: False
```

casper configuration see *mc_casperjs / casperjs/npm registry*

Install casper.js

You can then use the macro:

```
{% import "makina-states/localsettings/casperjs/init.sls" as casperjs with context %}
{{ casperjs.install('1.9.7', 'sha1_hash') }}
```

phantomjs configuration see *mc_phantomjs / phantomjs/npm registry*

Install phantom.js

You can then use the macro to install a specific version of phantomjs in /srv/app/phantomjs/<ver>:

```
{% import "makina-states/localsettings/phantomjs/init.sls" as phantomjs with context %}
{{ phantomjs.install('1.1-beta3', 'sha1_hash') }}
```

updatedb configuration see *mc_updatedb / updatedb functions*

timezone configuration Configure the local machine timezone

etckeeper configuration Configure and install etckeeper to keep track of /etc modifications.

After each package installation or highstate, we save /etc in a local git.

Git configuration Basic configuration to:

- put a basic git config in **/etc/gitconfig**
- configure the makina-corpus gitorious server ssh alias for each configured user including root.

It will uses the **makina-states.localsettings.users** state registry configuration items (all the makina-states.localsettings.users items) as the users where are dropped the ssh config slugs.

/etc/hosts managment Configure /etc/hosts entries based on configuration setings:

Eg having in pillar .. code-block:: yaml

toto-makina-hosts:

- ip: 10.0.0.8 hosts: foo.company.com foo
- ip: 10.0.0.3 hosts: bar.company.com bar

others-makina-hosts:

- ip: 192.168.1.52.1 hosts: foobar.foo.com foobar
- ip: 192.168.1.52.2 hosts: toto.foo.com toto2.foo.com toto3.foo.com toto
- ip: 10.0.0.4 hosts: alias alias.foo.com

All theses entries will be entered inside a block identified by .. code-block:: yaml

#– start salt managed zone – PLEASE, DO NOT EDIT (here) #– end salt managed zone –

It's your job to ensure theses IP will not be used on other entries in this file.

If you want to add some data in this block without using the pillar you can also use a file.accumulated state and push content in an accumulator while targeting /etc/hosts file with filename entry, this way .. code-block:: yaml

this-is-my-custom-state

file.accumulated:

- filename: /etc/hosts
- name: hosts-accumulator-makina-hosts-entries
- text: "here your text"
- require_in: - file: makina-etc-host-vm-management

JDK configuration Oracle JDK configuration and installation on Debian like systems using webupd8team repos.

- It installs JDK6 & JDK7.
- It links the **jdkDefaultver** as default jdk

Exposed pillar settings

makina-states.localsettings.java.default_jdk_ver default jdk version (6 or 7)

Exposed hooks

makina-states-jdk_begin before jdk install

makina-states-jdk_last after jdk install

PamLdap configuration

- install ldap base packages
- integrate pam with LDAP via nslcd, nss-ldapd pam-ldapd.

Pillar sample:

```
makina-states.localsettings.ldap:
  ldap_uri: ldap://ldap.foo.net/
  ldap_base: dc=company,dc=org
  ldap_passwd: ou=People,dc=company,dc=org?sub
  ldap_shadow: ou=People,dc=company,dc=org?sub
  ldap_group: ou=Group,dc=company,dc=org?sub
  ldap_cacert: /etc/ssl/cacerts/cacert.pem
  enabled: True
  nslcd:
    ssl: start_tls
```

Exposed settings:

makina-states.localsettings.ldap.enabled true/false: activate pamldap wiring

makina-states.localsettings.ldap.ldap_uri ldaps://localhost:636/

makina-states.localsettings.ldap.ldap_base dc=company,dc=org

makina-states.localsettings.ldap.ldap_passwd ou=People,dc=company,dc=org?sub

makina-states.localsettings.ldap.ldap_shadow ou=People,dc=company,dc=org?sub

makina-states.localsettings.ldap.ldap_group ou=Group,dc=company,dc=org?sub

makina-states.localsettings.ldap.ldap_cacert /etc/ssl/cacerts/cacert.pem (opt)

makina-states.localsettings.ldap.nslcd.ldap_ver None

makina-states.localsettings.ldap.nslcd.scope sub

makina-states.localsettings.ldap.nslcd.user nslcd

makina-states.localsettings.ldap.nslcd.group nslcd

makina-states.localsettings.ldap.nslcd.ssl start_tls, # ssl, off, start_tls

makina-states.localsettings.ldap.nslcd.tls_reqcert allow

makina-states.localsettings.ldap.nslcd.tls_cacert None

makina-states.localsettings.ldap.nslcd.bind_dn None

makina-states.localsettings.ldap.nslcd.bind_pw None

makina-states.localsettings.ldap.nslcd.rootpwmoddn None

makina-states.localsettings.ldap.nslcd.rootpwmodpw None

makina-states.localsettings.ldap.nslcd.bind_timelimit 30

makina-states.localsettings.ldap.nslcd.timelimit 30

makina-states.localsettings.ldap.nslcd.idle_timelimit 3600

makina-states.localsettings.ldap.nslcd.reconnect_sleeptime 1

makina-states.localsettings.ldap.nslcd.reconnect_retrytime 10

Locales managment Manage and configure system locales and their use

You can override default locales by adding them in pillar ths way:

Exposed settings:

> **makina-states.localsettings.locales.locales** list of locales to construct (default: fr variants)
>
> **makina-states.localsettings.locales.locale** default shell locale (default: fr)

rc.local managment manage /etc/rc.local via helper scripts in **/etc/rc.local.d** goal is to launch tricky services on the end of init processes. Eg launch the firewall only after lxc interfaces are up and so on.

Idea is that you just have to drop executable files inside /etc/rc.local.d and they will be executed (lexicographical order) at boot.

Network configuration Configure machine physical network based on pillar information. This state will only apply if you set to true the config value (grain or pillar): **makina-states.localsettings.network_managed** The template is shared with the lxc state, please also look at it.

Exposed settings:

> **makina-states.localsettings.network.managed** default: False

It will look for extra pillar entries suffixed in **-makina-network** as follow

```
makina-states.localsettings.network.interfaces.{{ifname}}:
  auto: opt (default: True)
  mode: opt (default: dhcp or static if address)
  address: opt
  netmask: opt
  gateway: opt
  dnsservers: opt
  pre-up: opt
    - list of rules non prefixed with post-up
  post-up: opt
    - list of rules non prefixed with post-up
  pre-down: opt
    - list of rules non prefixed with pre-down
  post-down: opt
    - list of rules non prefixed with pre-down
```

EG

```
makina-states.localsettings.network.managed : true
# manually configured interface
makina-states.localsettings.network.interfaces.eth0:
  address: 8.1.5.4
  netmask: 255.255.255.0
  gateway: 8.1.5.1
  dnsservers: 8.8.8.8
makina-states.localsettings.network.interfaces.em1: {} # -> dhcp based interface
makina-states.localsettings.network.interfaces.eth0-ipv6:
  ifname: eth0
  address: 2002:42D0:8:2202::1
  netmask: 64
  gateway:
  dnsservers: 127.0.0.1 212.126.32.92 8.8.8.8 4.4.4.4
  post-up:
    - /sbin/ip -f inet6 route add 2002:4120:2:2Fff:ff:ff:ff:ff dev eth0
```

```
    - /sbin/ip -f inet6 route add default via 2002:4120:2:2Fff:ff:ff:ff:ff
  pre-down:
    - /sbin/ip -f inet6 route del default via 2002:4120:2:2Fff:ff:ff:ff:ff
    - /sbin/ip -f inet6 route del 2002:4120:2:2Fff:ff:ff:ff:ff dev eth0
```

with explicit order

```
makina-states.localsettings.network.ointerfaces
    - em1: {} # -> dhcp based interface
    - eth0-ipv6:
          ifname: eth0
          address: 2002:42D0:8:2202::1
          netmask: 64
          gateway:
          dnsservers: 127.0.0.1 212.126.32.92 8.8.8.8 4.4.4.4
          post-up:
            - /sbin/ip -f inet6 route add 2002:4120:2:2Fff:ff:ff:ff:ff dev eth0
            - /sbin/ip -f inet6 route add default via 2002:4120:2:2Fff:ff:ff:ff:ff
          pre-down:
            - /sbin/ip -f inet6 route del default via 2002:4120:2:2Fff:ff:ff:ff:ff
            - /sbin/ip -f inet6 route del 2002:4120:2:2Fff:ff:ff:ff:ff dev eth0
    - eth0:
          address: 8.1.5.4
          netmask: 255.255.255.0
          gateway: 8.1.5.1
          dnsservers: 8.8.8.8
```

Nodejs configuration Install Node.js and allow the installation of Node.js packages through npm.

You can use the grain/pillar following setting to select the npm packages:

Exposed settings:

makina-states.localsettings.npm.packages LIST (default: [])

makina-states.localsettings.npm.versions LIST (default: [])

You can include version for packages, eg:

```
makina-states.localsettings.npm.packages: ['grunt@0.6']
```

There is a macro available to specify the version of node.js you want to use. Be sure to have them installed first:

```
makina-states.localsettings.npm.versions: ['0.8.26']
```

You can then use the macro:

```
{% import "makina-states/localsettings/nodejs-standalone.sls" as nodejs with context %}
{{ nodejs.npmInstall('less', '0.8.26') }}
```

nscd configuration Installs and manage a basic nssswitch.conf plus the nscd daemon.

Package manager configuration

Debian systems Manage official apt mirrors, for thirdparty repositories, you may have better to write a file in /etc/apt/sources.list.d/foo.list or better: use the salt pkgrepo.installed state.

Exposed settings:

makina-states.apt.ubuntu.mirror debian mirror to use

makina-states.apt.debian.mirror debian mirror to use

makina-states.apt.ubuntu.comps main (defaults: main restricted universe multiverse) defaults comps to install on ubuntu

makina-states.apt.debian.comps main (defaults: main contrib non-free) defaults comps to install on debian

Default packages managment Manage collections of packages to install by default on all boxes.

Python configuration Help to manage several python versions at once

For ubuntu users, install the deadsnakes repository.

Exposed settings:

makina-states.localsettings.python.versions LIST (default: ["2.4", "2.5", "2.6"]), pythons to install

eg:

```
salt-call grains.setval makina-states.localsettings.python.versions '["2.6"]'
```

dotdeb repository configuration For debian distributions, configure the dotdeb repository inside apt

RVM (ruby) Setup rvm environment inside **/usr/local/rvm**.

The rvm group is **rvm**.

Exposed settings:

makina-states.localsettings.rvm.url url to download the installer

makina-states.localsettings.rvm.rubies rubies to install as bases

makina-states.localsettings.rvm.user base rvm user

makina-states.localsettings.rvm.group base rvm user group

Shell configuration Make as ubuntu & others do: Configure /etc/profile.d as a direcotry container for a collection of shell scripts to sourced when a user login and execute his shell profile.

Sudoers managment Install sudo and a basic /etc/sudoers files which includes all files inside /etc/sudoers.d

It makes also all members of **sudo** or **admin** groups, sudoers

sysctl managment See *mc_kernel / kernel registry*

This configures various settings to tune the linux kernel for maximum performance.

Hooks

sysctl-post-hook executed after setting sysctls

System Users & SSH accces configuration See also ssh service documentation

Basic configuration to create users based on pillar configuration.

The following states use those data mappings:

- makina-states.localsettings.vim
- makina-states.localsettings.users
- makina-states.localsettings.git
- makina-states.service.base.ssh

Pass generation

```
>>> import crypt;print crypt.crypt('secret', '$6$SALTsalt$')
```

SSH To allow users to connect as root we define in pillar an entry which ties # ssh keys container in the 'keys' mapping to the near by 'users' mapping. See makina-states.services.base.ssh.

```
makina-states.localsettings.users.toto: []
makina-states.localsettings.users.root:
    home: /users/root (opt)
    admin: true (opt)
    ssh_keys: ['kiorky.pub', 'salt://foo.pub']
```

- salt://files/ssh/kiorky.pub && salt://foo.pub will be authorized in root's authorized ssh keys
- This will also create root as an admin if not existing
- This will also create a standard user named 'toto'
- As you guessed, if you do not specify an url, the keys are looked in salt://files/ssh.

It will uses the **makina-states.localsettings.users** state registry configuration items.

Other settings:

makina-states.localsettings.admin.sudoers sudoers list

makina-states.localsettings.admin.sysadmins_keys ssh keyfiles to drop from saltmaster

makina-states.localsettings.admin.sysadmin_password global sysadmin password hash

makina-states.localsettings.admin.root_password root password hash (default to sysadmin if unset

makina-states.localsettings.admin.absent_ssh_keys ssh keyfiles mappings to disable from auth (all users)

```
makina-states.localsettings.admin.sudoers: [joe]
makina-states.localsettings.admin.password: s3cret
makina-states.localsettings.admin.absent_ssh_keys:
    AAAAB3NzaC1yc2EAAAABIwAAAQEA6NF8iallvQVp22WDkTkyrtvp9eWW6A8YVr: {}
    AAAAB3NzaC1yc2EAAAABIwAAAQEA6NF8iallvQVp22WDkTkyrtvp9eWW6A8YVr:
        enc: ssh-rsa
makina-states.localsettings.admin.sysadmin_keys:
    - foo.pub
    - salt://foofoo.pub
```

If no root or sysadmin password, no changes to the system You have also a macro providen in this state to easily create users.

VIM editor configuration Basic vimrc handling for both global and configured users.

GNU Screen configuration Basic screenrc handling

Nodetypes formulaes

Devhost nodetype This state will regiter and mark this minion as a **devhost**.

The devhost has the special meaning to marking the machine as a development box.

Currently, we associate some behaviors on development boxes like making postfix & dovecot (mail) to be completly local. We also will tweak some settings like in apache, php or mysql states when we are in dev.

The idea when you have to test if you are on a development box is to test for devhost in the registry:

```
{{ salt['mc_nodetypes.registry']().is.devhost }}
```

Docker container nodetype Inherit from lxc container nodetype.

LXC container nodetype LXC container marker

This will install some stuff to run nicely within an LXC container, specially on ubuntu where we patch upstart jobs and init system to behave correctly when containerized.

Server nodetype Baremetal server type, this is the basetype for all other nodetypes. This will trigger the installation of **makina-states.localsettings**.

In standalone mode, we do not include localsettings.

TravisCI nodetype Mark this machine as a travis node. Useful to skip some part of your states when you are limited by travis limitations like no setting sysctl and no supporting posix acls.

To test if you are on a travis box:

```
{{ salt['mc_nodetypes.registry']().is.travis }}
```

Vagrant VM nodetype Will trigger some extra setup when we are on a vagrant box. It already inherit the setup from devhost.

On a vagrant box, we include the docker and lxc states, we also automaticly add some scripts to manage freespace (zerofree & system-cleanup)

We disable plymouth on ubuntu.

We also manage an **/etc/devhosts** file which is then merged via the provision script up to the host **/etc/hosts** file to provide access via name to the machine services.

Thus you can do **http://devhost[NUM].local** in your browser.

VM Nodetype This state will regiter and mark this minion as a **virtual machine**.

Controllers

Hooks

Hooks for salt orchestration For salt orchestration we provide the following hooks:

dummy-pre-salt-checkouts before we do any code update on base salt repositories

dummy-salt-layout before any base file or directory creation

dummy-pre-salt-service-restart before salt is restarted after its reconfiguration

dummy-post-salt-service-restart after salt is restarted after its reconfiguration

Hooks for mastersalt orchestration For mastersalt orchestration we provide the following hooks:

dummy-pre-mastersalt-checkouts before we do any code update on base mastersalt repositories

dummy-mastersalt-layout before any base file or directory creation

dummy-pre-mastersalt-service-restart before mastersalt is restarted after its reconfiguration

dummy-post-mastersalt-service-restart after mastersalt is restarted after its reconfiguration

Controllers

Salt To configure all salt daemons including at least a minion and certainly a master, we have three states files

salt install the base salt filesystem layout and files

salt_master configure a salt master daemon

salt_minion configure a salt minion daemon

Many of the makina-states components can select a branch (see mc_salt.settings module) Eg for makina-states:

```
makina-states.salt.makina-states.rev: apiv2
```

All those states files have a **-standalone** variant that let us redo a light reconfiguration upon highstates to take less time but with enought configuration to let us assume that the installation is sufficiently correct.

All those formulaes are thin wrappers to the salt_macro.

MasterSalt To configure all mastersalt daemons including at least a minion and certainly a master, we have three states files

mastersalt install the base mastersalt filesystem layout and files

mastersalt_master configure a mastersalt master daemon

mastersalt_minion configure a mastersalt minion daemon

All those states files have a **-standalone** variant that let us redo a light reconfiguration upon highstates to take less time but with enought configuration to let us assume that the installation is sufficiently correct.

All those formulaes are thin wrappers to the salt_macro.

Services

cache services

memcached

memcached

Backup services

astrailsafe obsolete and not maintained

burp configuration see *mc_burp / burp functions*

Configure a burp server.

Use the makina-states.services.backup.burp.client to :

- install burp binary
- install the cron

Use the makina-states.services.backup.burp.server to :

- install burp binary on server
- configure server
- generate client backup configuration files part
- push & dispatch & restart burp services on clients
- the burp server node must access the client via ssh as root via a key without password.
- Server & Clients must have **rsync**.
- The backuped machines must access the burp server on **4971** port.
- We offer a way to access clients via a ssh gateway

Burp uses a 'check for backup timer', each client will have a cron that ask the server to know if it is time for the client to be backuped, and in this case, the backup starts from the client.

This is why there is a parameter 'cron_periodicity' which is by default ran each 20 minutes and sprayed all over one hour between clients for the load not to be high on backup server.

The number of simultaneous backups is controlled via the 'max_children' server parameter.

bacula-fd configuration

db_smart_backup configuration Configure and install https://github.com/kiorky/db_smart_backup. This will run it from cron one time a day.

rdiff backup

backup users specificities

Base services

Hooks

Hooks for ssh orchestration (keys)

Services

ntp configuration

ssh configuration

Collaboration services

Etherpad configuration Etherpad allows you to edit documents collaboratively in real-time, much like a live multi-player editor that runs in your browser.

It uses circus to monitor the process.

Pillar Pillar values start with makina-states.services.collab.etherpad:

Value	Description	Default
version	Change which version of etherpad is installed.	'1.3.0'
location	Edit the directory in which circus is installed.	locs['apps_dir'] + '/etherpad'
apikey	The secret used to encrypt transmissions.	'SECRET-API-KEY-PLS-CHANGE-ME'
title	The title of the server.	'Etherpad'
ip	Ip on which the server will bind.	'0.0.0.0'
port	Port the server will listen for.	'9001'
dbType	Type of the database.	'dirty'
dbSettings	Settings of the database.	'{"filename": "var/dirty.db"}'
requireSession	Require session setting.	'true'
editOnly	Edit only setting.	'false'
admin	Create an admin or not.	False
adminPassword	Admin's password.	'admin'

Databases services

Services

Mongodb configuration

MySQL configuration

Postgresql configuration This states file aims to configure and manage postgresql clusters throught their respective unix sockets.

This is common wrappers around **postgresql** states/modules to arrange with how we manage postgresql clusters (multi versions, layout), that's why we, again, created our own custom states.

You have:

postgresql_db a macro to define databases with a default group as owner

postgresql_user a macro to define an user, his privileges and groups

postgresql_group a macro to define a group

postgresql_ext a macro to install a single pgsql extension

postgresql_exts a macro to install pgsql extensions

postgresql.conf configuration

You can override the postgresql.conf by either:

- attaching to the accumulator (see below)
- write a file in $CONF_PREFIX/<filename>.conf (except for any default setting in postgresql.conf)
- editing or overriding the 'pg_conf.<ver>' setting in the pgsql settings a (list of dicts), see the mc_states.modules.mc_pgsql module)

PG_HBA configuration

You can override the pg_hba.conf by either:

- attaching to the accumulator (see below)
- editing or overriding the 'pg_hba' setting in the pgsql settings a (list of dicts), see the mc_states.modules.mc_pgsql module)

Example from pillar

```
makina.services.postgresql.pg_hba  [ {'type': 'local',
                                     'database': 'foo',
                                     'user': foo', address,
                                     'foo', 'method': 'md5'} ]
makina.services.postgresql.pg_hba-overrides: [ {...} ]
```

Example to use the pg_hba block

```
append-to-pg-hba-{-accumulator:
  file.accumulated:
    - name: pghba-accumulator
    - require_in:
      - file: append-to-pg-hba-block
    - filename: /etc/postgresql/9.3/main/pg_hba.conf
    - text: '# Example from salt !'
```

Configuration via pillar example You can define via pillar the default user to run psql command as:

```
makina-states.services.postgresql.user: foo (default: postgres)
```

You can also define in pillar databases and users respecting naming convention: By default the owner of the database is a group with the same name suffixed with _owner for the user to be added to. We assign then users to this group

Define a database and its owner as follow (see salt.states.postgres_database.present)

```
bar-makina-postgresql:
  name: foo (opt, default; 'bar')
  encoding: foo (opt, default; utf8)
  template: foo (opt, default; template0)
  tablespace: foo (opt, default; pg_default)
```

This will create a 'bar' database owned by the group **bar_owners**

Define a user a follow (see salt.states.postgres_user.present)

```
bar-makina-services-postgresql-user:
  password: h4x
  groups: bar_owners (opt, default: [])
  encrypted: True (opt, default: True)
  superuser: True (opt, default: False)
  createdb: True (opt, default: False)
  replication: True (opt, default: False)
```

This will create a bar user with 'h4x' password and in group 'bar-owners' (the one of the precedent database)

eg:

```
mydb-makina-postgresql: {}
mydb-makina-services-postgresql-user:
  password: ckan-password
  superuser: True
  groups:
    - mydb_owners
```

Macro usage examples You can use them in your own states as follow

```
include:
  - makina-states.services.db.postgresql
{% import "makina-states/services/db/postgresql/init.sls" as pgsql with context %}
{% set db_name = dbdata['db_name'] %}
{% set db_tablespace = dbdata['db_tablespace'] %}
{% set db_user = dbdata['db_user'] %}
{% set db_password = dbdata['db_password'] %}
{{ pgsql.postgresql_db(db_name, tablespace=db_tablespace) }}
{{ pgsql.postgresql_user(db_user,
                         db_password,
                         groups=['{0}_owners'.format(db_name)]) }}
```

Remember that states should not contain any secret password or user. So here for example dbdata would be coming from a default macro loading pillar data.

Exposed hooks The hooks are defined in makina-states.services.db.postgresql-hooks. {ver} is one of the installed postgresql versions (eg: 9.3) {db} is a database name

makina-postgresql-pre-base before postgresql installation

makina-postgresql-post-base after postgresql installation

{ver}-makina-postgresql-pre-create-group before installing a group role in a speicific postgresql version

{ver}-makina-postgresql-post-create-group before installing a group role in a speicific postgresql version

{ver}-makina-postgresql-pre-create-db before databases installation

{ver}-{db}-makina-postgresql-database-post-hook specific database post creation hook

{ver}-{db}-makina-postgresql-database-endpost-hook specific database post creation hook before creating another database

{ver}-makina-postgresql-post-create-db adter databases installation

{ver}-makina-postgresql-pre-create-user before installing an user role in a speicific postgresql version

{ver}-makina-postgresql-post-create-user after installing an user role in a speicific postgresql version

makina-postgresql-post-inst final hook

Redis configuration

DNS services

BIND/NAMED integration

WARNING Bind is used as a cache dns server only right now, we have not finnished the zone management. So the docuementation about pillar zone management, etc is not fully implemented yet (and even deactivated to avoid misconfiguration

Generalities

- On everything else than containers, we:
  - activate bind at least as a cache dns
  - Remove dnsmasq as caching dns
- We separate logs in logical log files
- Me manage the dns zones inside bind views
- you must install bind tools prior to run (or run twice) to have all the tool neccessary to genrate tsig infos
- The default view is named **net**
- We can manage
- **Idea is**
  - You defines zone with rrs
  - You define views
  - for each zone, you feed the views list to link to those views

Hooks

bind-pre-install prefore pkg install

bind-post-install after pkg install

bind-pre-conf before touching to any conf file

bind-post-conf after configuration

bind-pre-restart before service restart

bind-post-restart after service restart

bind-pre-reload before service reload

bind-post-reload after service reload

Registry For the documentation on usage, please look *mc_bind / named/bind functions*.

Defaults SOA settings

makina-states.services.dns.bind.ttl ttl for SOA record

makina-states.services.dns.bind.serial zone serial

makina-states.services.dns.bind.refresh zone refresh time

makina-states.services.dns.bind.retry zone retry time

makina-states.services.dns.bind.expire zone expire time

makina-states.services.dns.bind.minimum zone minimum

makina-states.services.dns.bind.notify is notify activated in named conf (**True/False**)

makina-states.services.dns.bind.server_type is it a **master** or **slave** zone

makina-states.services.dns.bind.masters For slave zones, list of masters. This is mandatory

Configured in pillar Zones The scheme to name a new zone is

```
makina-states.services.dns.bind.<zonekind>.<zonename>:
    setting1: value
```

You can override default settings on a per zone basis. Please look at implementation to know all switchs, but here are the fields inside a zone mapping:

name optjonnal fqdn of the host, default to the <id> part in the pillar string. This is the SOA name.

template If true, we will use a template to generate the zone file, see the defaults templates.

source

- alternative template file if template if True
- Otherwise, plain text source file for zone

views the views to put the zone in, default to [net]

ttl ttl for SOA record

serial zone serial

refresh zone refresh time

retry zone retry time

expire zone expire time

expire zone expire time

minimum zone minimum

notify is notify activated in named conf (**True/False**)

server_type is it a **master** or **slave** zone

salves For master zones, list of slave servers. This is optionnal

masters For slave zones, list of masters. This is mandatory

Defaults templates settings

makina-states.services.dns.bind.zone_template Template to generate zones

makina-states.services.dns.bind.reverse_template Template to generate reverse zones

makina-states.services.dns.bind.sec_zone_template Template to generate slave zones

makina-states.services.dns.bind.sec_reverse_template Template to generate reverse slave zones

Define a new acl An acl is in the form

```
makina-states.services.dns.bind.acls.<name>:
    clients: []
```

Exemple:

```
makina-states.services.dns.bind.acls.sec1:
    clients: ['!1.2.4.3']
```

Edit the client for the default 'local' acl which has recursion enabled

```
makina-states.services.dns.bind.acls.local.clients:
   clients: ['192.168/16', '127.0.0.1', '::1',]
```

Define a new server entry A server is in the form

```
makina-states.services.dns.bind.servers.<name>:
    keys: []
```

Exemple:

```
makina-states.services.dns.bind.servers.18.2.5.6:
    keys: ['sec1-key']
```

Define a new key A key is in the form

```
makina-states.services.dns.bind.keys.<name>:
  algorithm: hmac-md5 (default to this)
  secret: '<secure data>'
```

Exemple:

```
makina-states.services.dns.bind.keys.loc1:
  secret: 'aaaqsfsqfqsdfqsdfqsdfgeZA=='
```

RNDC configuration The configuration is automatic.

Bits are in:

- /etc/rndc.conf
- /etc/rndc.key

- /etc/bind.conf.key

Define a new view A view is in the form The linking between zones and view is done as a per view basis. See *Configured in pillar Zones*.

```
makina-states.services.dns.bind.views.<name>:
  match_clients: []
  recursion: no
  additional_from_cach: no
  additional_from_auth no
```

Exemple:

```
makina-states.services.dns.bind.views.intranet;
  match_clients: ['10.0.0.0/16']
  recursion: yes
  additional_from_cach: no
  additional_from_auth: no
```

Manage a zone directly from a file, no generation

```
makina-states.services.dns.bind.zones:
  template: false
  source: salt:///srv/salt/myzone

makina-states.services.dns.bind.zones.foo.net:
  serial: 2
  rrs:
    - '@ IN A 1.2.4.4'
makina-states.services.dns.bind.zones.foo.loc
  views: [intranet]
  serial: 2
  fqdn: foo.net
  rrs:
    - '@ IN A 192.168.4.4'
```

Save for reverse zone except the id would be the ip bits.

Manage a slave zone

```
makina-states.services.dns.bind.slave_zones.foo.net:
```

Save for reverse zone except the id would be the ip bits.

An example or a master/slave scenario on a shared pillar:

```
{% set masterip = '1.2.3.5' %}
{% set slave1ip = '1.2.3.4' %}
{% set slave1ip_tsig = salt['mc_bind.tsig_for'](slave1ip) %}
makina-states.services.dns.bind.keys.{{slave1ip}}:
  algorithm: HMAC-SHA512
  secret: "{{slave1ip_tsig}}"
```

On the master pillar:

```
makina-states.services.dns.bind: true
include:
    - common
```

```
makina-states.services.dns.bind.zones.toto.loc:
  allow_transfer: ['key "{{slave1ip}}"']
  serial: 4
  rrs:
    - '@ IN A 1.2.4.4'
    - 'ns IN A 1.2.4.4'
    - 'mx IN A 1.2.4.4'
    - '@ IN MX 10 mx.foo.net.'
    - '@ IN NS ns.foo.net.'
makina-states.services.dns.bind.servers.{{slave1ip}}:
  keys: ["{{slave1ip}}"]
```

This will enable the master to sign data sent to slave1

On the pillar slave targeted pillar, now:

```
makina-states.services.dns.bind: true
include:
    - common
makina-states.services.dns.bind.servers.{{masterip}}:
  keys: ["{{slave1ip_tsig}}"]
makina-states.services.dns.bind.zones.toto.loc:
  server_type: slave
  masters: ["{{masterip}}"]
```

dhcpd integration

Firewall services

Shorewall configuration Configure shorewall, see the following documentation + salt *mc_shorewall / shorewall functions* module to know which option to configure in pillar.

The firewall is able to mostly autoconfigire itsel, even for rpn, lxc & docker; you should just only have rules or params to add !

For tricky part, you can fallback on configuration hints via pillar/grains.

By default we configure a firewall enabling ssh, mail and and http(s) services only.

There are variables to easily restrict access by ip.

Default Rules We create defaults rules for you:

- fw -> all defined zones: allowed
- allow docker zone from/to all internal subnets
- allow lxc zone to {docker, fw} but no inter lxc
- drop all other traffic by default
- enable smtp, dns, ssh, http
- disable invalid, ping & ftp

You can either:

- disable all the default rules
- disable or enable the traffic controlled by one of those network flows

Enable/disable default rules Allowed by default:

makina-states.services.firewall.shorewall.no_default_rules true/false (false)

makina-states.services.firewall.shorewall.no_dns true/false (false)

makina-states.services.firewall.shorewall.no_web true/false (false)

makina-states.services.firewall.shorewall.no_ssh true/false (false)

makina-states.services.firewall.shorewall.no_ping true/false (false)

makina-states.services.firewall.shorewall.no_mastersalt true/false (false)

makina-states.services.firewall.shorewall.no_ntp true/false (false)

makina-states.services.firewall.shorewall.no_burp true/false (false)

makina-states.services.firewall.shorewall.no_ldap true/false (false)

makina-states.services.firewall.shorewall.no_mumble true/false (false)

Restricted to localhost by default:

makina-states.services.firewall.shorewall.no_syslog true/false (false)

Blocked by default:

makina-states.services.firewall.shorewall.no_salt true/false (true)

makina-states.services.firewall.shorewall.no_invalid true/false (true)

makina-states.services.firewall.shorewall.no_snmp true/false (true)

makina-states.services.firewall.shorewall.no_postgresql true/false (true)

makina-states.services.firewall.shorewall.no_mysql true/false (true)

makina-states.services.firewall.shorewall.no_ftp true/false (true)

Restrict access for some services Just configure a **RESTRICTED_SERVICE** parameter ! Supported params are:

- **RESTRICTED_SSH**: for restricting ssh access
- **RESTRICTED_SNMP**: for restricting snmp access
- **RESTRICTED_PING**: for restricting snmp access
- **RESTRICTED_FTP**: for restricting ftp access
- **RESTRICTED_POSTGRESQL**: for restricting postgres access
- **RESTRICTED_MYSQL**: for restricting mysql access
- **RESTRICTED_SYSLOG** for restricting syslog access
- **RESTRICTED_NTP** for restricting syslog access
- **RESTRICTED_MUMBLE** for restricting syslog access
- **RESTRICTED_LDAP** for restricting syslog access
- **RESTRICTED_BURP** for restricting syslog access

```
makina-states.services.firewall.shorewall.params.RESTRICTED_SSH: "<src_def>"
```

EG:

```
makina-states.services.firewall.shorewall.params:
  IP_FOO: "12.232.243.200"
  IP_COMPANY: "12.23.9.8,2.24.3.18,1.24.19.4"
  IP_SUPERVISION: "19.14.1.0,1.11.3.26"
  RESTRICTED_SSH: "net:$IP_FOO,$IP_COMPANY,$IP_SUPERVISION"
  RESTRICTED_SNMP: "net:$IP_SUPERVISION"
  RESTRICTED_PING: "net:$IP_SUPERVISION"
```

Firewalling lxc containers default policy:

- lxc -> dck: auth
- dck -> lxc: auth
- fw -> lxc: auth
- lxc -> net: auth

Dedibox RPN firewalling default policy:

- rpn -> all: drop
- fw -> rpn: auth

Firewalling docker containers default policy:

- dck -> net: auth
- dck -> dck: auth
- lxc -> dck: auth
- dck -> lxc: auth

Disable firewall even if installed Disable shorewall service to start in config (pillar, grain)

```
makina-states.services.shorewall.enabled: True | False
```

Defining shorewall interfaces

```
makina-states.services.firewall.interfaces:
  shorewall-zone-name:
    - interface: phyname
      options: shorewall interface options (man shorewall-interfaces)
```

Eg:

```
makina-states.services.firewall.interfaces:
  net:
    - interface: eth0
      options: tcpflags,dhcp,nosmurfs,routefilter,logmartians,sourceroute=0
```

Masquerade configuration

```
makina-states.services.firewall.shorewall.masqs:
  masq: (man shorewall-masq)
    interface-comment:
      interface: ifname
      source: (opt)
      address: (opt)
      proto: (opt)
      ports: (opt)
      ipsec: (opt)
      mark: (opt)
```

EG:

```
makina-states.services.firewall.shorewall.masq:
  lxc:
    interface: eth0
    source: lxcbr0
```

Params configuration Please note:

- All paramsare automaticly prefixed with **SALT_**
- All params are **sorted** lexicographically after the loading
- You needif you reference params to use the **SALT_** prefix, we wont replace params automatically.

```
makina-states.services.firewall.shorewall.params:
  param: value
```

EG:

```
makina-states.services.firewall.shorewall.params:
  thishostguest: 10.0.3.2
  00_cd: 10.0.3.2
  00_ab: 10.0.3.2
  a: 1
```

mapping afterloading:

```
makina-states.services.firewall.shorewall.params:
  SALT_00_ab: 10.0.3.2
  SALT_00_cd: 10.0.3.2
  SALT_a: 1
  SALT_thishostguest: 10.0.3.2
```

Zones configuration

```
makina-states.services.firewall.shorewall.zones:
  NAME: (man shorewall-zones)
    type: zone type
    options: (opt)
    in: (opt)
    out: (opt)
    in_options: (opt)
    out_options: (opt)
```

EG:

```
makina-states.services.firewall.shorewall.zones:
  zones:
    fw:  {type: firewall}
    net: {type: ipv4}
    lxc: {type: ipv4}
```

Policy configuration

```
makina-states.services.firewall.shorewall.policies: (list of dict):
  - source: shorewall zone (man shorewall-policies)
    dest: shorewall zone
    policy: policy
    loglevel: 'loglevel (opt)'
    limit: 'limit:burst (opt)'
```

EG:

```
makina-states.services.firewall.shorewall.policies:
  policy:
    - {source: $FW, dest: net, policy: ACCEPT,}
    - {source: rpn, dest: all, policy: DROP, loglevel: info}
    - {source: all, dest: all, policy: REJECT, loglevel: info}
```

Rules configuration

```
makina-states.services.firewall.shorewall.rules: (list of dict):
   - section: new (default) : established | related | all (opt)
     action: action todo
     source: source addr     (man shorewall-rules)
     dest: dest addr
     proto: (opt)
     dport: (opt)
     sport: (opt)
     odest: (opt)
     rate: (opt)
     user: (opt)
     mark: (opt)
     connlimit: (opt)
     time: (opt)
     headers: (opt)
     switch: (opt)
```

EG:

```
makina-states.services.firewall.shorewall.rules:
  - {section: established, action: 'Invalid(DROP)', source: net, dest: all}
  - {action: Invalid(DROP), source: net, dest: all}
  - {action: DNS(ACCEPT),   source: all, dest: all}
  - {action: SSH(ACCEPT),   source: all, dest: all}
  - {action: Ping(ACCEPT),  source: all, dest: all}
  - {action: Ping(DROP),    source: net, dest: $FW}
  - {comment: 'thishostguest lxc'}
  - {action: DNAT, source: net, dest: 'lxc:${thishostguest}:80', proto: tcp, dport: 8082}
  - {comment: 'dhcp in lxc'}
  - {action: ACCEPT, source: lxc, dest: fw , proto: udp, dport: '67:68'}
  - {action: ACCEPT, source: fw , dest: lxc, proto: udp, dport: '67:68'}
  - {comment: 'salt'}
  - {action: ACCEPT, source: all, dest: fw, proto: 'tcp,udp', dport: '4506,4505'}
```

```
  - {comment: 'relay smtp from lxc and drop from net'}
  - {action: Invalid(DROP), source: net, dest: all, proto: 'tcp,udp', dport: 25}
  - {action: ACCEPT       , source: lxc, dest: fw , proto: 'tcp,udp', dport: 25}
```

Default options a lot of options has been duplicated and parsed the same way to have two keys to facilitate default behavior for firewall + minus variations without having to deal with macros.

Be aware that we use those 'defaults' to apply/append/update (no override) also the default firewall configuration if you have not disabled the autoconfiguration.

Supported defaults:

- rules (default_rules)
- zones (default_zones)
- interface: (default_interfaces)
- masqs (default_masqs)
- params (default_params)
- policies (default_policies)

Example:

firewallcommon.sls:

```
makina-states.services.firewall.shorewall.default_rules:
    - {action: Invalid(DROP), source: net, dest: all}
```

firewall1.sls:

```
makina-states.services.firewall.shorewall.rules:
    - {action: WEB(ACCEPT), source: net, dest: all}
```

firewall2.sls:

```
makina-states.services.firewall.shorewall.rules:
    - {action: SSH(ACCEPT), source: net, dest: all}
```

Don't Repeat Yourself Tips and tricks Use jinja macros !

EG:

/srv/pillar/firewall-common.sls

```
{% macro params %}
    ip1: X.X.X.X
{% endmacro %}
```

/srv/pillar/minionfirewall.sls

```
{% import 'firewall-common.sls' as c with context %}
makina-states.services.firewall.shorewall.params:
    {{c.params()}}
    ip2: Y.Y.Y.Y
```

fail2ban configuration Configure fail2ban, see the salt *mc_fail2ban / fail2ban functions* module to know which option to configure in pillar.

Pillar key: **makina-states.services.firewall.fail2ban**

Eg:

```
makina-states.services.firewall.fail2ban: true
makina-states.services.firewall.fail2ban.maxretry: 10
```

psad configuration Configure psad, see the salt *mc_psad / psad functions* module to know which option to configure in pillar.

Pillar key: **makina-states.services.firewall.psad**

Settings key: **makina-states.services.firewall.psad**

Eg:

```
makina-states.services.firewall.psad: true
makina-states.services.firewall.psad.alertdest: sysadmin@goo.com
makina-states.services.firewall.psad.hostname: bar.goo.com
```

Ftp services

Hooks

FTPd orchestration hooks

Services

pureftpd configuration

Geographical information system services

Postgis configuration Install postgis packages. Then install a database named **postgis** using makina-states.services.db.postgresql macros.

Activation

```
makina-states.services.gis.postgis: true
```

Use a database with posgis template

```
{%- import "makina-states/services/db/postgresql.sls" as pgsql with context %}
{{- pgsql.postgresql_db(common.data.db.name, template="postgis") }}
{{- pgsql.postgresql_user(common.data['db']['user'],
                          password=common.data['db']['password'],
                          db=common.data['db']['name'],) }}
```

Qgis mapserver configuration

Http services

Hooks

Orchestration hooks for apache installation and configuration

Apache

Core

Apache httpd

Apache modules

apache_proxy

apache mod_fastcgi

apache mod_fcgid

Nginx

Nginx

- This integrates configure and tune the fast http nginx server.
- Please have a loop to *mc_nginx / nginx registry* to know all configuration options.
- We have take a spetial care to try to tune it some a good production usage start and you should really have a look to the generated configuration files to kow if it fits with your setup.
- In our own particular setup, nginx is served by a frontal haproxy reverse proxy. We for know use the xforwardedfor header but are panning to use the haproxy protocol as soon as it will be battletested.
- In other words, in such a setup we automaticly setup the realip module to log the real client infos

Please note that we offer a spetial macro to generate virtualhosts and manage their activation. Look at here.

Sites are enabled and deactivated a la debian, with the /etc/nginx/sites-{available/deactivated} directories.

```
{% import "makina-states/services/http/nginx/vhosts.sls" as nvh with context %}
vhostbody:
  file.managed:
      - name: /srv/salt/body.domain.com
      - source: ''
      - contents: |
                  redirect_permanent: google.fr
{{nvh.virtualhost('domain.com',
                   vhost_content_template='salt://body.domain.com') }}
```

You can also register new sites in pillar to avoid calling manually the macro. pillar example:

```
makina-states.services.http.nginx.virtualhosts.example.com:
    active: False
    small_name: example
    documentRoot: /srv/foo/bar/www
makina-states.services.http.nginx.virtualhosts.example.foo.com:
    active: False
    port: 8080
    server_aliases:
      - bar.foo.com
```

Note that the best way to make a VH is not the pillar, but loading the macro as we do here and use virtualhost()) call in a state. Then use the pillar to alter your default parameters given to this call

Java services

Solr4 configuration

Tomcat configuration By default, we will use the java6 oracle jdk jvm.

For example:

```
makina-states.services.java.tomcat7:
     java_home': /usr/lib/jvm/java-6-oracle
 users:
   admin:
     password: {{ password }}
     roles': ['admin', 'manager']
```

AVAILABLE DEFAULT SETTINGS You can override default settings in pillar via the mc_states.tomcat module, please look its relative doc.

CUSTOM CONFIGURATION BLOCKS Thanks to file.blockreplace + file.accumulated, you can also add on the fly raw tomcat configuration blocks. Custom configuration blocks have been wired on those files:

- server.xml
- context.xml
- web.xml
- logging.properties
- {{ locs.conf_dir }}/default/tomcat7 (practical to add JAVA_OPTS addition from other sls (eg adding solr datadir & home properties)
- catalina.properties

See at the end of this state file for the appropriate blockreplace to use in your case and where those block are located in the aforementioned files. What you will need is just to make a file.accumulated requirin the appropriate file.blockreplace ID to add your configuration block.

Exposed state orchestration hooks Hooks (in order):

tomcat-pre-install-hook before tomcat installation

tomcat-post-install-hook after tomcat installation

tomcat-pre-restart-hook before tomcat restart (after install)

tomcat-pre-blocks-hook: before applying block replaces in configuration files

tomcat-post-restart-hook after tomcat restart

Mail services

Imap

Dovecot

SMTP

Postfix Configure postfix, see the salt *mc_postfix / postfix functions* module to know which option to configure in pillar. There are shortcut modes to ease the configuration, please see bellow

Focus on how you can configure postfix (under the hood, we feed hastables):

- makina-states.services.postfix.sasl_passwd: /etc/postfix/sasl_passwd mappings
- makina-states.services.postfix.transport /etc/postfix/transports (useful for relay wildcards)
- makina-states.services.postfix.relay_domains /etc/postfix/relay domains (hosts)
- makina-states.services.postfix.mydestinations /etc/postfix/mydestinations
- makina-states.services.postfix.virtual_map: /etc/postfix/virtual_map (better way of rewriting aliases)

Each of those file has a .local counterpart which lets you put inside manual configuration. eg: /etc/postfix/transports.local

Relay mode

- This has no local delivery.
- This will forward all emails to a relay host.
- You can use authenticated smtp.

Example pillar use

```
makina-states.services.mail.postfix: true

makina-states.services.mail.postfix.mode: relay
makina-states.services.mail.postfix.transport:
  - nexthop: relay:[mx.f.com]

makina-states.services.mail.postfix.auth: true
makina-states.services.mail.postfix.sasl_passwd:
  - entry '[mx.f.com]'
    user: xxx@f.com
    password: xx

makina-states.services.mail.postfix.virtual_map:
  /root@.*/: sysadmin@f.com
  /postmaster@.*/: sysadmin@f.com
  /abuse@.*/: abuse@f.com
```

Local Delivery mode

- This will store all outbound mails emitted by local network interfaces locally into the nobody mailbox (/var/spool/nobody/ configurated recipient mailbox (configured in pillar)

Example pillar use

```
makina-states.services.mail.postfix: true
makina-states.services.mail.postfix.mode: localdeliveryonly
```

Redirect Delivery mode

- This will redirect all outbound mails emitted by local network to a specific address

Example pillar use

```
makina-states.services.mail.postfix: true
makina-states.services.mail.postfix.mode: redirect
makina-states.services.mail.postfix.local_dest: kiorky@gmail.com
```

If you need to identify yourself to the outbound smtp:

```
makina-states.services.mail.postfix: true
makina-states.services.mail.postfix.mode: redirect
makina-states.services.mail.postfix.auth: true
makina-states.services.mail.postfix.local_dest: kiorky@gmail.com
makina-states.services.mail.postfix.sasl_passwd:
  - entry '[smtp.gmail.com']
    user: kiorky@gmail.com
    password: **
```

Custom mode

- This will let you configure each option of postfix explicitly, no defaults from relay or localdelivery will be applied.

Example pillar use

```
makina-states.services.mail.postfix: true
makina-states.services.mail.postfix.mode: custom
```

Exposed hooks

- postfix-pre-install-hook
- postfix-post-install-hook
- postfix-pre-conf-hook
- postfix-post-conf-hook
- postfix-pre-restart-hook
- postfix-post-restart-hook

Monitoring services

Circus configuration Circus is a Python program which can be used to monitor and control processes and sockets.

Exposed Hooks:

circus-pre-restart before circusd restart

circus-post-restart after circusd restart

Pillar value start with makina-states.services.monitoring.circus:

location Edit the directory in which circus is installed. (**locs['apps_dir'] + '/etherpad'**)

SNMPD configuration See *mc_supervisor / supervisor functions* for configuration options.

Exposed Hooks:

- supervisor-pre-install
- supervisor-post-install
- supervisor-pre-configuration
- supervisor-post-configuration
- supervisor-pre-restart
- supervisor-post-restart

Pillar value start with **makina-states.services.monitoring.supervisor**.

SNMPD configuration we provide only snmpv3 configuration and totally disable snmpv2 for security considerations.

See *mc_snmpd / snmpd functions* for configuration options.

Exposed Hooks:

- snmpd-pre-install
- snmpd-post-install
- snmpd-pre-configuration
- snmpd-post-configuration
- snmpd-pre-restart
- snmpd-post-restart

Pillar value start with **makina-states.services.monitoring.snmpd**.

Client configuration For now we:

- install various nagios plugins inside /root/admin_scripts/nagios

Icinga configuration See *mc_icinga / icinga functions* for configuration options.

About packaging The icinga module can provide:

- configuration of icinga core
- configuration of database for ido2db
- configuration of ido2db daemon
- configuration of uwsgi in order to serve cgi
- configuration of nginx to serve cgi through uwsgi
- configuration of icinga (add/remove objects configuration)

The icinga_web module can provide:

- configuration of icinga web
- configuration of database for icinga web
- configuration of php-fpm
- configuration of nginx to serve php webpages through php-fpm

icinga_web module can configure its nginx virtualhost to serve cgi but icinga_web module doesn't configure uwsgi and doesn't install any file related to cgi.

The reason is to keep icinga and icinga_web independants. icinga-web and icinga can be installed on two differents hosts but CGI files require to have icinga-core installed on the host.

icinga_web module depends on icinga module only if cgi is enabled in icinga-web virtualhost (makina-states.services.monitoring.icinga_web.nginx.icinga_cgi.enabled is set to True).

In the same way, icinga_web doesn't depends on pnp4nagios even if the module is enabled. You have to install pnp4nagios separately.

The mysql configuration doesn't work.

The architecture of service folder looks like to :

init.sls is the file which includes the others

hooks.sls defines some states for schedule

macros.jinja is empty because I don't have used macros for icinga and icinga_web

prerequisites.sls defines states which install packages

configuration.sls defines states which write configuration and init scripts

services.sls defines states which start the service

pgsql.sls the states which install and configure postgresql if this dependance is needed. The states are called before ones in prerequisites.sls

mysql.sls the states which install and configure mysql if this dependance is needed. The states are called before ones in prerequisites.sls

nginx.sls the states which install and configure nginx and phpfpm if theses dependances are needed. The states are called before ones in prerequisites.sls

For "pgsql.sls", "mysql.sls" and "nginx.sls", I don't have separated the states between prerequisites, configuration and services.

The architecture between Icinga, Icinga-web and nagvis looks like to:

```
icinga---ido2db--------><-postgresql-><---icinga-web
     |
     +---mklivestatus---><-----------------nagvis
     |
     +---npcdmod-------><-npcd-><-rrd-><--pnp4nagios
```

Please note that icinga service offers four special macros to generate configurations. Theses macros are described below.

Configuration In configuration.sls, the general configuration of icinga and the objects configuration are made For the objects configuration, some macros are called automatically with the content defined in:

```
makina-states.services.monitoring.icinga.objects
```

This dictionary architecture looks like:

directory directory in which configurations files will be written (default: /etc/icinga/objects/salt_generated)

objects_definition dictionary in which each subdictionary is given to "configuration_add_object" macro as **kwargs. It is used to define objects like contacts or timeperiods or commands

autoconfigured_hosts_definitions dictionary in which each subdictionary is given to "configuration_add_auto_host" macro as **kwargs. It is used to define hosts and hostgroups and add services associated to them.

purge_definitions a list of files or directory which can be deleted. Each element of the list is given to "configuration_remove_object" macro as **kwargs)

The "purge_definitions" list is used to remove hosts. because a call to "configuration_add_auto_host" macro with all services disabled will remove only the services and not the host itself.

Macros

configuration_add_object This macro was written in order to add an object in the icinga configuration

```
{% import "makina-states/services/monitoring/icinga/init.sls" as icinga with context %}
{{ icinga.configuration_add_object(type, file, attrs, definition, **kwargs) }}
```

with

type the type of added object

file the filename of the added object

attrs a dictionary in which each key corresponds to a directive

definition the name used to identify the definition. It is the name used by configuration_edit_object. If none, configuration_edit_object will not work for this definition

The default directory where configuration files are located is:

```
/etc/icinga/objects/salt_generated/
```

The directory can be modified in the "makina-states.services.monitoring.icinga.objects" dictionary

You can change the configuration directory using **kwargs parameter

A call with:

```
{{ icinga.configuration_add_object(
                                type='host',
                                file='hosts/hostname1.cfg',
                                attrs={
                                        'host_name': "hostname1",
                                        'use': "generic-host",
                                    },
                               ) }}
```

Generates the file in /etc/icinga/objects/salt_generated/host/hostname1.cfg containing:

```
define host {
 use=generic-host
 host_name=hostname1
}
```

The services are managed in the same way:

```
{{ icinga.configuration_add_object(
                                type='service',
                                file='services/SSH',
                                attrs={
                                        'use': "generic-service",
                                        'service_description': "SSH",
                                    },
                               ) }}
```

That generates the file /etc/icinga/objects/salt_generated/service/SSH.cfg containing:

```
define service {
 use=generic-service
 service_description=SSH
}
```

configuration_remove_object This macro was written in order to remove an object in the icinga configuration

```
{% import "makina-states/services/monitoring/icinga/init.sls" as icinga with context %}
{{ icinga.configuration_remove_object(file, **kwargs) }}
```

with

file the filename of the added object

The default directory where configuration files are located is:

```
/etc/icinga/objects/salt_generated/
```

The directory can be modified in the "makina-states.services.monitoring.icinga.objects" dictionary

configuration_edit_object This macro was written because some values in object configuration depends on the rest of the configuration.

For example, you can have:

```
host_name=host1,host2,host3
```

in a service definition

But when you call the configuration_add_object, you don't know what hosts will be listed in this directive.

```
{% import "makina-states/services/monitoring/icinga/init.sls" as icinga with context %}
{{ icinga.configuration_edit_object(type, file, attr, value, auto_host, definition, **kwargs) }}
```

with

type the type of edited object

file the name of the edited object

attr the directive for which a value must be added

value the value added

auto_host true if the file is a file created with configuration_add_auto_host macro

definition the definition to edit in the file

The "file" argument value is relative to "makina-states.services.monitoring.icinga.objects.directory" (default: /etc/icinga/objects/salt_generated/)

The old values of the attr directive are not removed.

If you call:

```
{{ icinga.configuration_edit_object(type='service',
                                    file='SSH.cfg',
                                    attr='host_name',
                                    value='hostname1') }}
```

the previous service definition becomes:

```
define service {
 use=generic-service
 service_description=SSH
 host_name=hostname1
}
```

If you recall the macro with a different value:

```
{{ icinga.configuration_edit_object(type='service',
                                    file='SSH.cfg',
                                    attr='host_name',
                                    value='hostname2') }}
```

the previous service definition becomes:

```
define service {
 use=generic-service
 service_description=SSH
 host_name=hostname1,hostname2
}
```

when auto_host is set to true, the value for definition argument are:

- definition='host': or definition='hostgroup' the attribute will be added in the host/hostgroup definition
- definition=service: the attribute will be added in service definition. service have to be in the service list and have to be enabled
- definition=service-name: the attribute will be added in service loop definition.

Limits Currently, the macro doesn't edit the icinga.cfg file in order to add the directory in the list of "cfg_dir" You should think to make a coherent configuration.

By default, the /etc/icinga/objects is present in "cfg_dir".

No checks are done. You can generate invalid values for any directives. You can set non-existent directives too.

configuration_add_auto_host This macro is designed to add an host and associated services

```
{% import "makina-states/services/monitoring/icinga/init.sls" as icinga with context %}
{% icinga.configuration_add_auto_host(hostname,
                                      hostgroup=False,
                                      attrs={},
                                      ssh_user='root',
                                      ssh_addr='',
                                      ssh_port=22,
                                      ssh_timeout=30,
                                      backup_burp_age=False,
                                      backup_rdiff=False,
                                      beam_process=False,
                                      celeryd_process=False,
                                      cron=False,
                                      ddos=false,
                                      debian_updates=False,
                                      dns_association_hostname=False,
                                      dns_association=False,
                                      dns_reverse_association=False,
                                      disk_space=False,
                                      disk_space_root=False,
                                      disk_space_var=False,
                                      disk_space_srv=False,
                                      disk_space_tmp=False,
                                      disk_space_data=False,
                                      disk_space_mnt_data=False,
                                      disk_space_home=False,
                                      disk_space_var_lxc=False,
                                      disk_space_var_makina=False,
                                      disk_space_var_mysql=False,
                                      disk_space_var_www=False,
                                      disk_space_backups=False,
                                      disk_space_backups_guidtz=False,
                                      disk_space_var_backups_bluemind=False,
                                      disk_space_var_spool_cyrus=False,
                                      disk_space_nmd_www=False,
                                      drbd=False,
                                      epmd_process=False,
                                      erp_files=False,
                                      fail2ban=False,
                                      gunicorn_process=False,
                                      haproxy=False,
                                      ircbot_process=False,
                                      load_avg=False,
                                      mail_cyrus_imap_connections=False,
                                      mail_imap=False,
                                      mail_imap_ssl=False,
                                      mail_pop=False,
                                      mail_pop_ssl=False,
                                      mail_pop_test_account=False,
                                      mail_server_queues=False,
```

```
                                    mail_smtp=False,
                                    megaraid_sas=False,
                                    memory=False,
                                    memory_hyperviseur=False,
                                    mysql_process=False,
                                    network=False,
                                    ntp_peers=False,
                                    ntp_time=False,
                                    only_one_nagios_running=False,
                                    postgres_port=False,
                                    postgres_process=False,
                                    prebill_sending=False,
                                    raid=False,
                                    sas=False,
                                    snmpd_memory_control=False,
                                    solr=False,
                                    ssh=False,
                                    supervisord_status=False,
                                    swap=False,
                                    tiles_generator_access=False,
                                    ware_raid=False,
                                    web_apache_status=False,
                                    web_openid=False,
                                    web=False,
                                    services_attrs={}
                                   ) %}
```

with

hostname the hostname of the added host

hostgroup if true, a hostgroup will be added instead of a simple host (because it is possible to add services for a hostgroup)

attrs a dictionary in which each key corresponds to a directive in the host definition

ssh_user user to connect the host (it is used by check_by_ssh command)

ssh_addr address used to do the ssh connection in order to perform check_by_ssh. this address is not the hostname address becasue we can use a ssh gateway

ssh_port ssh_port

[service] a boolean to indicate that the service [service] has to be added

services_attrs a dictionary to override the default values for each service definition and to ad additional values. The keys begining with "cmdarg_" are the check command arguments. Each subdictionary corresponds to a service.

Some services use an additional subdictionary because they can be defined several times. It is the case of

- dns_association
- dns_reverse_assocation
- disk_space
- network
- solr
- web_openid
- web

For theses services, you may complete the services_attrs dictionary by adding a subsubdictionary (the dictionary associatio to 'a_service' key here):

```
service_attrs: {
    'dns_association': {
        'a_service': {
            'cmdarg_hostname': "www.example.net",
        }
    }
}
```

You can add several dns_association, disk_space, network, solr, web_openid, web

For others services, the directives are not in a subsubdctionary but directly in the subdictionary:

```
service_attrs: {
    'raid': {
        'check_command': "check",
    }
}
```

You have to insert in services_attrs only the non default values.

Note: The directive "host_name" will not be taken into account. The value will be replaced with the value of "hostname" macro argument

The host is added in /etc/icinga/objects/salt_generated/<hostname>/host.cfg The services are added in this directory too (for ssh it will be /etc/icinga/objects/salt_generated/<hostname>/ssh.cfg)

The services are defined specially for the host. There is no:

```
define service {
    host_name host1,host2
}
```

The commands definitions are located in objects/objects_defintions subdictionary in mc_icinga.py They are installed with a state in configuration.sls.

All the commands objects are created even if no service use them.

All the complexity is in "mc_icinga.add_auto_configuration_host_settings" function (see *mc_icinga / icinga functions*)

The macro only adds the host (or hostgroup) by calling "configuration_add_object" and browses the services.

if the service is enabled: a state adds the service configuration file by calling the "configuration_add_object" macro

if the service is disabled: a state removes the service configuration file by calling the "configuration_remove_object" macro

For each host, a state is executed for each service even if all the services are disabled. The execution takes about 30 minutes for 128 hosts and 50 services (the macro configuration_add_object is called 939 times and configuration_remove_object is called 6158 times (yes, it doesn't correspond to 50*128 because there are the commands definitions, contacts, ...)).

The speed can be improved by removing the "watch_in" directive in the "configuration_remove_object" macro (because this macro is called a lot of time).

Without this directive. the execution takes about 10 minutes for 128 hosts and 50 services but the configuration files are removed after the restart of icinga.

I don't have find how to fix this problem. I used a "order: 1" directive but in this case the states are executed before prerequisite (which is less problematic than when the execution was after the restart of icinga. The files are deleted

before the creation of new files. If a file is in "purge_definitions" dictionary and is created in another macro call. The file will be deleted and recreated in a next state)

Another idea is to delete several configuration files with only one state.

Off topic: I have used "order" directive in configuration_add_object macro too. (the execution time seems to be the same without any directive like order or watch_in and with a order directive)

The execution is 629.685 secondes for 128 hosts, 50 services and "order" instead of "watch_in" in "configuration_add_object" and "configuration_remove_object"

Without the "order" directive in configuration_add_object macro but with a "watch_in" directive, the execution is 820.238 secondes.

The difference is 190.553 for 939 "watch_in" (the 939 call of "configuration_add_object" macro). So a "watch_in" directive take 0.203 secondes

With 6158 "watch_in" for "configuration_remove_object", it is (0.203*6158) 1249.654 secondes (about 20 minutes).

I have supposed "watch_in" execution time constant.

With 50 services per hosts (ignore services_loop which can increase the number of services): The host autoconfiguration macro need about 10.353 secondes to execute "watch_in" directives in one call.

With about 360 hosts the excessive execution time approach the entire hour.

The issue was resolved by decreasing the number of states: there is only one state to create each host. The services for the host are in the same file.

This decrease the number of states and the call to configuration_remove_object is useless to delete old services because the file with services of the hosts is naturally edited.

The execution time decrease to 1 minute about for 128 hosts but with 1000 hosts it ran out of memory. However, it is perhaps a bad idea to have all services in a same file because the files becomes long.

The memory problem was solved by moving the "object" subdiction so that it is not cached. Only a list of hosts is cached is "object" subdictionary.

The function "get_settings_for_object" is designed to get non cached values.

An other modification is that the macro doesn't give the data to template. Before the modification it was:

```
{% set data = salt['mc_icinga.add_auto_configuration_host_settings'](...) %}
icinga-configuration-{{data.state_name_salt}}-add-auto-host-conf:
    ...
    - defaults
      data: |
            {{sdata}}
```

Now, the data are stored in a light dictionary in global variables in "mc_icinga.add_auto_configuration_host.objects" In the macro there is:

```
{% set data = salt['mc_icinga.add_auto_configuration_host'](...) %}
icinga-configuration-{{data.state_name_salt}}-add-auto-host-conf:
    ...
    - defaults
      hostname: |
            {{salt['mc_utils.json_dump'](data.hostname}}
```

The function "mc_icinga.add_auto_configuration_host" stores the object informations in a dictionary like:

```
{
    'hostname': {}
}
```

Each subdictionary contains all arguments given to the macro. But this methods requires to store a lot of data. For objects which are in localsettings, I have added a "fromsetting" argument. Instead of store all arguments given to the macro, only the key in localsettings is stored:

```
{
    'hostname': {'fromsettings': 'host1'}
}
```

Only key of this dictionary is given to template. The template get the object from localsettings by calling "get_settings_for_object" if a "fromsettings" key is found. And the settings are given to the previous "mc_icinga.add_auto_configuration_host_settings" function.

All is done in the template in order to avoid store a lot of data in memory during a long time. Then, a lot of memory is used during template compilation, when:

```
default:
  data: |
        {{sdata}}
```

is replaced with:

```
default:
  data: |
        {a big dictionary here which is the return of utf8 encode in order to use more memory}
```

With theses modifications, it is possible to manage only 7000 hosts with 10 services per host with 1Go of memory (another 800Mo of memory is needed to run salt-master).

Add a new service in configuration_add_auto_host macro If you want add a new service managed with this macro, you have to:

1. add arguments in macro and in add_auto_configuration_host_settings function
2. add the service in "services" or "services_loop" list
3. add the default values in "services_default_attrs"
4. if the service was added in "services_loop" list, add code to merge dictionaries
5. if the default "check_command" is new, add a "command" definition in "objects_definitions" dictionary (in "objects" function) and add the command with its arguments in "check_command_args"

Nagvis configuration See *mc_nagvis / nagvis functions* for configuration options.

This service configure nagvis with a connection to icinga using mklivestatus because nagvis currently can't connect to a postgresql database.

Nagvis doesn't depend on icinga unless mklivestatus socket is done with a unix pipe

Integration with icinga-web is done in icinga_web service.

Please note that we offer two special macros to generate configurations.

One macro is designed to add a map and the other is to add a geomap

add_map macro

```
{% import "makina-states/services/monitoring/nagvis/init.sls" as nagvis with context %}
{{ nagvis.add_map(name, _global, objects, keys_mapping, **kwargs) }}
```

with

name name of the map (it is the filename too, so each name must be unique)

_global dictionary in wich contians directives for 'define global{}' section

objects dictionary wich defines all objects.

keys_mapping dictionary to etablish the associations between keys of dictionaries and directives

The objects dictionary contains a subdictionary for each type of objects. In each subdictionary, subsubdictionaries contains directives for each 'define <type> {}'

The keys of subsubdictionaries are the values of directives defined in the keys_mapping dictionary. Because sometimes it is not possible to find a directive with unique value, if a value is set to "None", in the key_mapping dictionary, the subdictionary become a list

The objects dictionary looks like:

```
'objects' = {
    'host': {
        'h1': {
            'x': 4,
            'y': 3,

        },
    },
    'service': {
        'SSH': {
            'host_name': "h1",
            'x': 4,
            'y': 3,
         },
    },
}
```

With the key_mapping dictionary:

```
keys_mapping = {
    'host': "host_name",
    'hostgroup': "hostgroup_name",
    'service': "service_description",
    'servicegroup': "servicegroup_name",
    'map': "map_name",
    'textbox': None,
    'shape': None,
    'line': None,
    'template': None,
    'container': None,
}
```

So, "h1" is the value for "host_name" directive and "SSH" is the value for "service_description" directive

You can add directives as key:value in each subdictionary

The macro produces a cfg file in /etc/nagvis/maps/name.cfg. This file contains

With the example above, the file located in /etc/nagvis/maps/name.cfg will contain:

```
define global {
}
define host {
    host_name=h1
```

```
    x=4
    y=3
}
define service {
    host_name=h1
    service_description=SSH
    x=4
    y=3
}
```

add_geomap macro

```
{% import "makina-states/services/monitoring/nagvis/init.sls" as nagvis with context %}
{{ nagvis.add_geomap(name, hosts, **kwargs) }}
```

with

name name of the geomap (it is the filename too, so each name must be unique)

hosts dictionary in which each subdictionary defines a host

The hosts dictionary looks like:

```
'hosts': {
    'ham-srv1': {
        'alias': "Hamburg Server 1",
        'lat': 53.556866,
        'lon': 9.994622,
    },
    'mun-srv1': {
        'alias': "Munich Server 1",
        'lat': 48.1448353,
        'lon': 11.5580067,
    },
},
```

The macro produces a csv file like

```
muc-srv1;Munich Server 1;48.1448353;11.5580067
ham-srv1;Hamburg Server 1;53.556866;9.994622
```

This macro produces only the /etc/nagvis/geomap/name.csv file and not the /etc/nagvis/maps/name.cfg file. In order to produce the /etc/nagvis/maps/name.cfg file, you should call the "add_map" macro.

Pnp4nagios configuration See *mc_pnp4nagios / pnp4nagios functions* for configuration options.

This service configure pnp4nagios with a connection to icinga

Pnp4nagios depends on Icinga. The installation of pnp4nagios will trigger the installation of icinga. It seems not possible to have a pnp4nagios on a different host that icinga. (you can always use a shared filesystem...)

Pnp4nagios edits icinga.cfg file in order to enable performance data.

Integration with icinga-web is done in icinga_web service.

Openstack services

openstack controller configuration not finished yet, no ETA

Php services

Hooks

PHP orchestration hooks when used with apache

PHP orchestration hooks

Common states

Common php installation setup

Common php setup to link with an apache server

ModPhp

ModPhp configuration

FPM

PHPFPM configuration

PHPFPM configuration with apache

Proxy services

haproxy Configure haproxy, see the salt *mc_haproxy / haproxy functions* module to know which option to configure in pillar. There are shortcut modes to ease the configuration, please see bellow

Exposed hooks

- haproxy-pre-install-hook
- haproxy-post-install-hook
- haproxy-pre-conf-hook
- haproxy-post-conf-hook
- haproxy-pre-restart-hook
- haproxy-post-restart-hook

Example: http reverse proxy based on domain name Add the following entries to your pillar and re run the haproxy states

```
makina-states.services.proxy.haproxy.frontends.myapp.domain.com:
  mode: http
  bind: ':80'
  raw_opts:
    - acl host_myapp.domain.com hdr(host) -i myapp.domain.com
    - use_backend bck_myapp.domain.com if host_myapp.domain.com
```

```
makina-states.services.proxy.haproxy.backends.bck_myapp.domain.com:
    mode: http
    raw_opts:
      - option http-server-close
      - option forwardfor
      - balance roundrobin
    servers:
      - name: srv_myapp.domain.com1
        bind: 10.0.3.7:80
        opts: check
```

this will configure a reverse proxy for domain myapp.domain.com on port 80 -> 10.0.3.7 port 80.

uwsgi configuration See *mc_uwsgi / uwsgi functions* for configuration options.

Please note that we offer a special macro to generate configurations.

Configurations are enabled and deactivated a la debian, with the /etc/uwsgi/apps-{available/deactivated} directories.

```
{% import "makina-states/services/cgi/uwsgi/init.sls" as uwsgi with context %}
{{ uwsgi.config(config_name, config_file, enabled, **kwargs) }}
```

with

config_name name of configuration file

config_file template of configuration file

enabled if true, symlink configuration file in /etc/uwsgi/apps-enabled

Queue services

Services

rabbitmq configuration

Log services

rsyslog Configure rsyslog, see the salt *mc_rsyslog / rsyslog functions* module to know which option to configure in pillar. There are shortcut modes to ease the configuration, please see bellow

Exposed hooks

- rsyslog-pre-install-hook
- rsyslog-post-install-hook
- rsyslog-pre-conf-hook

- rsyslog-post-conf-hook
- rsyslog-pre-restart-hook
- rsyslog-post-restart-hook

ulogd Configure ulogd, see the salt *mc_ulogd / ulogd functions* module to know which option to configure in pillar. There are shortcut modes to ease the configuration, please see bellow

Exposed hooks

- ulogd-pre-install-hook
- ulogd-post-install-hook
- ulogd-pre-conf-hook
- ulogd-post-conf-hook
- ulogd-pre-restart-hook
- ulogd-post-restart-hook

Virtualization services

Hooks

Docker orchestration hooks

LXC orchestration hooks

Services

Docker configuration Manage docker base installation and containers

LXC configuration Manage lxc base installation and containers

Legacy

LEGACY: shorewall & docker integration Was used in the path to make glue & orchestration between docker network setup and shorewall

LEGACY: shorewall & lxc integration Was used in the path to make glue & orchestration between lxc network setup and shorewall

Monitoring services

mumble configuration

- This integrates configure and tune mumble as a server.
- Please have a loop to *mc_mumble / mumble registry* to know all configuration options.

Projects

Apache based project

Common stuff

Beecollab project

Hooks for project orchestration

Lizmap based Project This will install apache, with a PHPFPM pool and a qgis mapserver.

We also install aside a pureftpd server to let qgis users upload/sync qgis projectsto their lizmap projects.

We also install a postgis database dedicated to the project

Modphp based project

Ode api project Repository located at https://github.com/makinacorpus/ODE.

It uses the python project state.

The following services are included and can be customized through the pillar, check their respectives documentation for more information on how to do it:

- makina-states.services.db.postgresql
- makina-states.localsettings.python

The following settings can be modified through the pillar:

- makina-states.projects.ode_api.db_name
- makina-states.projects.ode_api.db_user
- makina-states.projects.ode_api.db_password
- makina-states.projects.ode_api.circus_port
- makina-states.projects.ode_api.circus_pubsup_port
- makina-states.projects.ode_api.circus_stats_port
- makina-states.projects.ode_api.port
- makina-states.projects.ode_api.admins

TODO Add circus service and configure watcher.

Ode frontend project Repository located at https://github.com/makinacorpus/django-ode.

It uses the python project state.

The following services are included and can be customized through the pillar, check their respectives documentation for more information on how to do it:

- makina-states.services.db.postgresql
- makina-states.services.http.nginx
- makina-states.localsettings.nodejs
- makina-states.localsettings.python

The following settings can be modified through the pillar:

- makina-states.projects.ode_frontend.db_name
- makina-states.projects.ode_frontend.db_user
- makina-states.projects.ode_frontend.db_password

TODO Add circus service and configure watcher.

PHPFPM based project Install a basic document root backed by a FPMPOOL and the document root is served via apache

Python project Install a python project. "python_version" can be passed to the install macro to change the python version used by the project.

If requirement is set to True (default), then it will install the requirements from the requirements.txt in the project directory using pip.

RVM project Project macro to leverage the installation of a ruby app using RVM.

Zope Generic Portal install helper This permit to install generic portals (generated via cgwb to be installed via makina states

Some nuances:

- We generate the 'etc/sys/settings-local.cfg' file from defaults settings found in configuration or pillar, see zope defaults
- We generate a special buildout file 'buildout-salt.cfg' file which extends the production one minus some parts that salt do itself
- We wire the vhost produced by generic buildout and do not use makina-states apache macro for that
- We wire the logrotate slug produced by generic buildout.
- We wire the supervisord init script by generic buildout and register that as a service.
- We use our own crons for zope maintainance

Salt Cloud integration

Introduction makina-states include a generic multi-drivers cloud-controller as a large part of a future upper level PaaS project. Indeed, This is the raw level of the corpus PaaS project.

At the moment:

- Bare metal servers are provisionned via the saltify driver
- LXC container are provisionned via the lxc driver.

In the idea:

- We have a cloud controller
- We have compute nodes which are bare metal slaves to host vms.
- We have vms of a certain virtualization type
- The cloud controller is driver agnostic, and the only thing to support a new technology is to add the relevant sls, modules & runners to mimic the awaitened interfaces.

Specifications

The sequence of makina-states.cloud.generic makina-states.cloud.generic do all the generic cloud related stuff:

- On the controller front:
  - run pre configured drivers specific hooks
  - generation of control ssh keys and minion keyss
  - generation and configuration of saltcloud related stuff
  - control of related services like new DNS records
  - run post configured drivers specific hooks
- On the compute node:
  - run pre configured drivers specific hooks
  - firewalld as the firewall
  - synchronnize/pull any neccessary image or VM templates
  - configure haproxy to load balance http; http and ssh traffic
    - http/https use standart port
    - ssh use a custom range (40000->50000) and one port is allocated for each vm.
  - run post configured drivers specific hooks
- On the VM driver specific front (each of those steps is hookable (post or pre))
  - run pre configured drivers specific hooks
  - spawn the new minion via the compute node
  - install default users
  - install marker grains
  - install the cloud controller ssh key on the vm
  - run highstate on the new vm

  - ping the new minion
  - run post configured drivers specific hooks
- On the compute node & vms (post provision):
  - Any task remaining to make the newly VM minion a good citizen.
- On the compute node & vms (post vm provisions):
  - Any task remaining to make the newly VM minion a good citizen.

See *Detailed documentation* for an exemple

How Basically the interface with this cloud controller is done:

- Via the `pillar` for configuratioin
- Via `execution modules` to make settings structures and some specific stuff like SSL certificate generation. They are heavily used by the runners.
- Via `runner modules` to make actions on controller, compute_nodes and vms.
  - The runner may in turn execute **slses** from the makina-states.cloud directory on the controller or on a compute_node or on a vm.

Install & configure the cloud ecosystem: Using salt runners When you pillar is ready for action, you next step is to send a command to provision and configure your infrastructure nodes.

Runners overview This is in order how is configured each part of the cloud. This can help you to understand, debug & learn how to act on your cloud.

Configure the controller If you want to only install the controller configuration, just do:

```
mastersalt-run -lall mc_cloud_controller.orchestrate no_saltify=True no_provision=true
```

This is a good idea to do that when there is a long time you did not touched to it.

Saltify any compute node The next step would certainly be to attach the compute nodes:

```
mastersalt-run -lall mc_cloud_controller.orchestrate no_configure=True no_provision=true
```

The next step would certainly be to attach via saltify a specific node:

```
mastersalt-run -lall mc_cloud_controller.orchestrate no_configure=True no_provision=true only=[minion
```

Configure the compute node After having the compute node linked, you can begin to configure it to host your VMs:

```
mastersalt-run -lall mc_cloud_compute_node.orchestrate  only=[minionid] only_vms=[vm_id]
```

Is equivalent and you have better to use:

```
mastersalt-run -lall mc_cloud_controller.orchestrate only=[minionid] only_vms=[vm_id] no_dns_conf=Tru
```

This call `provision_compute_nodes` which in turn calls all `compute_node` related stuff which will run generic and per drivers specific hooks (firewall, loadbalancer, driver images sync, etc.).

only and **only_vms** are optionals but recommended to limit the scope of your commands.

Spawning and running vm post-configuration Eg to provision only bar:

```
mastersalt-run -lall mc_cloud_controller.orchestrate only=[foo] only_vms=[bar] no_compute_node_provis
```

Is equivalent to:

```
mastersalt-run -lall mc_cloud_vm.orchestrate only=[bar]
```

only and **only_vms** are optionals but recommended to limit the scope of your commands.

Runners reference links

Runners:

- *mc_cloud / cloud registries & functions*
- *mc_cloud_controller / cloud related variables*
- *mc_cloud_compute_node / cloud compute node related functions*
- *mc_api*
- *mc_cloud_controller runner*
- *mc_cloud_compute_node runner*
- *mc_cloud_saltify runner*
- *mc_cloud_vm runner*
- *mc_cloud_lxc runner*
- *mc_cloud_kvm runner*

Configuration modules

- *mc_cloud / cloud registries & functions*
- *mc_cloud_compute_node / cloud compute node related functions*
- *mc_cloud_controller / cloud related variables*
- *mc_cloud_compute_node / cloud compute node related functions*
- *mc_cloud_saltify / cloud related variables*
- *mc_cloud_lxc / lxc registry for compute nodes*

Helpers

- *mc_api*

Further implementation reference

Makina-states cloud generic controller & compute node & vm documentation

Controller On this node, we mainly do:

- cloud configuration generation
- compute node & VMs deployment orchestration
- SSL managment
- Maintenance
- Images store for lxc containers

The SSL certificates managment and centralization The generation use and wait for such a layout:

```
|- <salt-root>/cloud-controller/ssl
|    |-<salt-root>/certs/<certname>.pub
|    |-<salt-root>/certs/wildcards/<certname>.pub
|
|- <pillar-root>/cloud-controller/ssl
|    |-<salt-root>/certs/wildcards/<certname>.pem
|    |-<salt-root>/certs/<certname>.pem
```

- The idea is that each controller is tied to a subset of SSL certificates. Each domain tied to a controller will need to have a corresponding SSL certificate even self signed.
- Corrolary, the cloud controller will also act as the signin certifates authority for self signed certificates in this default case of not having a registered certificate for a particular domain.
- Each of those certificates will also be tied to one ore more running vms.
- For each domain tied to a compute node, we check for a matching ssl certificate existence and generate a self signed one if not existing.
- We distribute those certificates using regular salt file.managed salt:// prior to reverse proxy configuration for the certicates, and pillar access key for private keys.
- The ssl mapping is only be done at generation time and graved inside generated sls files for compute nodes.

Compute nodes

Responsabilities

- Running vms
- Routing network traffic
- Basic network firewalling and redirections
- Reverse proxies
- Any other configured baremetal services

Haproxy Some notes:

- We use haproxy to load balance the http/https traffics to the vm.
- We generate a configuration file in **/etc/haproxy/extra/cloudcontroller.cfg**.
- The ssl termination is on the HAPROXY node !
- We load balance http/https traffic by taking care of using either the proxy protocol or using regular X-Forwarded-For http header (forwardfor haproxy option).

- For now as the proxy protocol is a bit young, we default to use the xforwardedfor method. This is managable as a per vm basis.
- The cloud controller as part of the generation process will have registered all SSL certificates to load for the https reverse proxy. We use the new haproxy-1.5+ SSL features to load the directory of certificates which we will grab from the master.

Settings:

makina-states.cloud.cloud_compute_node.ssh_port_range_start tweak the default ssh allocation port start point

makina-states.cloud.cloud_compute_node.ssh_port_range_end tweak the default ssh allocation port end point

makina-states.cloud.<provider>.<target>.<vm>.http_proxy_mode set to 'xforwardfor' to use xforwardfor (default). Setting to something else will use haproxy proxy protocol If nothing is set, use xforwardfor for the moment.

WARNING, This will bind ports **80** & **443** so it may conflict with any existing configuration, please double check.

SSL & reverse proxy

- We do the SSL termination on the haproxy node.
- For this, you will need to setup here the mapping between you client certificates and the underlying domains.
- For each node we generate a self signed certificate to ensure https connection without the need to have a valid certificate under the hood, but, hay, prefer a valid one.
- We redirect traffic based on the host providen on the request.
- A **X-SSL** header is added on the request for SSL terminated backends.

Inject custom configuration for http reverse proxy This can be done as usual via pillar

makina-states.cloud.compute_node.conf.<computenode_name>.http_proxy.raw_opts_pre insert before generated rules

makina-states.cloud.compute_node.conf.<computenode_name>.http_proxy.raw_opts_post insert after generated rules

Exemple:

```
.. code-block:: yaml
```

makina-states.cloud.compute_node.conf.devhost10.local.http_proxy.raw_opts_pre:

- acl host_myapp.foo.net hdr(host) -i myapp.foo.net
- use_backend bck_myapp.foo.net if host_myapp.foo.net

You can define the underlying backend also this way

```
makina-states.services.proxy.haproxy.backends.bck_myapp.foo.net:
    mode: http
    raw_opts:
      - option http-server-close
      - option forwardfor
      - balance roundrobin
    servers:
      - name: srv_myapp.foo.net1
```

```
        bind: 10.0.3.7:80
        opts: check
```

Custom configuration for https reverse proxy

makina-states.cloud.compute_node.conf.<computenode_name>.https_proxy.raw_opts_pre insert before generated rules

makina-states.cloud.compute_node.conf.<computenode_name>.https_proxy.raw_opts_post insert after generated rules

Exemple:

```
.. code-block:: yaml
```

makina-states.cloud.compute_node.conf.devhost10.local.https_proxy.raw_opts_pre:

- acl host_myapp.foo.net hdr(host) -i myapp.foo.net
- use_backend bck_myapp.foo.net if host_myapp.foo.net

You can define the underlying backend also this way

```
makina-states.services.proxy.haproxy.backends.bck_myapp.foo.net:
    mode: http
    raw_opts:
      - option http-server-close
      - option forwardfor
      - balance roundrobin
    servers:
      - name: srv_myapp.foo.net1
        bind: 10.0.3.7:80
        opts: check
```

Compute node Automatic grains We enable some boolean grains for the compute not to install itself:

- makina-states.cloud.is.compute_node
- makina-states.services.proxy.haproxy
- makina-states.services.firewall.shorewall

If lxc, we also have:

- makina-states.services.virt.lxc

LXC provision As always, the configuration is done via a registry module: *mc_cloud_lxc / lxc registry for compute nodes*. To provision a new lxc provider, you need to:

- Select a profile type (combination of backing mode (lvm/dir) and install mode (clone/create)
- Select a size profile if you are using block level backing stores (LVM)
- Select a mode: mastersalt (localmaster+minion and a minion linked to mastersalt) or salt (only a minion linked to master)
- add an entry to your lxc container in pillar reflecting those choices plus the other vm parameters.
- We create a default container in mastersalt node named **makina-states-precise**. In cloning mode, this is the default origin container.

- You can of course either start an lxc from scratch or begin with a template like a barebone ubuntu which will be cloned at a start.
- By default, we use the **10.5/16* network on tje **lxcbr1** bridge using the default gateway: **10.5.0.1**.
- We use **LVM** as the default backing store in production, and the **lxc** volume group.
- We use **overlayfs** with snapshot in development mode.

To create new VMS, just configure your pillar and add some entries as follow

```
makina-states.services.cloud.lxc.targets.devhost10.local:
  # -> autogenerated ip + mac + password stored in grains, name is id (mysupertest3.foo.co,
  #   will use default configured profile & settings
  mysupertest3.foo.com: {}
  # explicit configuration and not using mastersalt mode
  gfoobar.test.com:
    profile: small
    profile_type: lvm
    password: foobar
    ip: 10.5.10.16
  # -> autogenerated ip + mac
  gfoobar4.test.com:
    ip: 10.5.10.15
    profile: small
    mode: mastersalt
    profile_type: lvm
    password: foobar
  # -> autogenerated ip + mac + password stored in grains
  gfoobar3.test.com:
    ip: 10.5.10.17
    # from_container: makina-states-precise -> default
    profile_type: dir
    mode: mastersalt
    password: foobar
    bootsalt_branch: stable
  gfoobar2.test.coma:
    ip: 10.5.10.17
    # image: ubuntu -> default
    profile_type: dir-scratch
    mode: mastersalt
    password: foobar
```

- The first line ends (after **containers.**) with your targeted minion id, where the lxc containers will be installed.
- You have to assign the ip yourself to something that will be in the **10.5/16** network or the targeted minion
- MAC is autogenerated if not providen, so you dont need to give one
- IP is autogenerated if not providen, so you dont need to give one
- PASSWORD autogenerated if not providen, so you dont need to give one
- And the inner mappings define the container themselves.
- Please note that the name in makina-corpus way must be the NickName FQDN.
- Please do not use **snapshot** in production.

Misc notes

- Salt cloud profiles are just collections of default for your next provision.

- The naming scheme of raw salt-cloud is ms-**{encoded minion id}**[-**{size profile}]-{**profile type}**, eg profiles:

```
ms-devhost-10-local-dir-scratch
ms-devhost-10-local-dir
ms-devhost-10-local-small-lvm-scratch
ms-devhost-10-local-medium-lvm
```

- Those are the availables modes:

  salt cf saltity modes

  mastersalt cf saltity modes

- You can specify the makina-states branch to use with:

  bootsalt_branch branch name

- Those are the sizes available in profiles

  xxxtrem 2000g

  xxtrem 1000g

  xtrem 500g

  xxxlarge 100g

  large 20g

  medium 10g

  small 5g

  xsmall 3g

  xxsmall 1g

  xxxsmall 500m

- Those are the types available in salt-cloud profiles

  lvm-sratch starting a lxc container from scratch (lvm backing)

  lvm cloning from existing container (lvm backing)

  dir-scratch starting a lxc container from scratch (directory backing)

  dir cloning from existing container (directory backing)

  overlayfs-scratch starting a lxc container from scratch (overlayfs backing)

  overlayfs cloning from existing container (overlayfs backing)

- Attention, we need also that root from **controller** can connect both as saltmaster and via ssh to either the **computenode** and the **lxc** node as root without password (sshkey). Please not that the states normally do that setup for you but that may be a start of investigation in case of problems.

Remove a vm To destroy at once boxes and minion keys on master:

```
salt-cloud -d <name>
```

Get infos for a VM To know specific settings for a vm, like the generated ip and password, you can inspect the per vm settings. Those settings are mainly used at creation time but not reapplied after further setup, so they may be obsoletes. The only "live" settings are the gateway, the ip and ssh_reverse_proxy_port.

Please note that we also give here the **ssh_reverse_proxy_port** to access the vm from the host:

```
mastersalt-call mc_cloud_lxc.get_settings_for_vm <compute_node <vm_name>
```

For exemple, you can have something like that:

```
autostart:
    True
backing:
    overlayfs
bootsalt_branch:
    master
bridge:
    lxcbr1
dnsservers:
    - 8.8.8.8
    - 4.4.4.4
domains:
    - nmdcarto51.test.com
from_container:
    makina-states-precise
gateway:
    10.5.0.1
image:
    ubuntu
ip:
    10.5.0.12
lxc_conf:
lxc_conf_unset:
mac:
    00:16:3e:00:f1:81
master:
    10.5.0.1
master_port:
    4606
mode:
    mastersalt
name:
    nmdcarto51.test.com
netmask:
    16
network:
    10.5.0.0
password:
    balh
profile:
    ms-devhost10-local-overlayfs
script_args:
    -C --from-salt-cloud --mastersalt-minion -b master
snapshot:
    True
ssh_gateway:
    devhost10.local
ssh_gateway_key:
    /root/.ssh/id_dsa
```

```
ssh_gateway_password:
    None
ssh_gateway_port:
    22
ssh_gateway_user:
    root
ssh_reverse_proxy_port:
    40000
ssh_username:
    ubuntu
sudo:
    True
users:
    - root
    - sysadmin
```

Detailed documentation

Exemple of the makina-states.cloud.lxc and how will integrate itself in the previous sequence:

steps = ['spawn', 'hostsfile', 'sshkeys', 'grains', 'initial_setup', 'initial_highstate']

- On the controller front:
  - At run pre configured drivers specific hooks stage:
    - install the salt cloud lxc providers
    - install a cron that sync all defined images templates from controller to compute nodes.
  - At compute node post hook
    - install lxc
    - ensure images templates are installed
    - install lxc host specific grains
- On the vm pre hook:
  - spawn the vm
- on the vm post hook
  - configure specific lxc grains
  - configure host file
  - initial setup

LXC specific usage All of those are integrated directly withe mc_cloud_{controller,compute_node,vm} runners, you do not have to use them directly, this is purely for documentation purpose.

Controller Re run configuration of cloudcontroller:

```
mastersalt-run -lall mc_cloud_lxc.post_deploy_controller
```

compute node Install lxc:

```
mastersalt-run -lall mc_cloud_lxc.install_vt <computenode_id>
```

This will call in turn those runners:

- mc_cloud_lxc.configure_grains <computenode_id>
- mc_cloud_lxc.configure_install_lxc <computenode_id>
- mc_cloud_lxc.configure_images <computenode_id>

This will also run the LXC images (templates) syncrhonnisation runner on that specific node.

VM

Baremetal (via ssh) provision aka salt/saltify As always, the configuration is done via a registry: *mc_cloud_saltify / cloud related variables*.

This will ssh to the distant box, transfer and run boot-salt.sh (makina-states bootstrap), then install a (master)salt minion and reattach that distant box with the providen id to this salt master.

IOW, We attach distant boxes with the saltify salt-cloud driver. For this, we need to:

- indicate the ssh_host user & passwords
- indicate which profile to use (minion or mastersalt minion)
- Maybe indicate a gateway to each the host to attach
- The following modes:

  salt (default) Attach the linked box as a salt minion only

  mastersalt Attach the linked box as a masteralt minion. This will also install on it a saltmaster/minion couple.

- You can specify the makina-states branch to use with:

  bootsalt_branch branch name

The idea is to add to your specific minion pillar some salty entries as follow:

```
# minion id to set and also nick fqdn is after "targets."
makina-states.cloud.saltify.targets.gfoobar.test.com:
  ssh_username: ubuntu
  sudo_password: ubuntu
  password: ubuntu
  ssh_host: 10.5.10.16
  sudo: True
  master: 10.5.0.1 (default to grains['fqdn'])
  master_port: 4506 (default to 4506 or 4606 in mastersalt mode)
makina-states.cloud.saltify.targets.gfoobar2.test.com:
  ssh_host: 10.5.10.15
  mode: mastersalt
  ssh_username: ubuntu
  password: ubuntu
  sudo_password: ubuntu
  branch: stable
  sudo: True
  master: 10.5.0.1
  master_port: 4506
```

You can even use a ssh gateway to initiate an host behind a firewall:

```
ssh_gateway: foo
ssh_key: /tmp/id_dsa
```

To activate the driver you need to install the generic & the saltify formulaes

```
makina-states.cloud.generic: true
makina-states.cloud.saltify: true
```

Makina-states Macros

Base macros

Macros You can use the circusAddWatcher macro to add a watcher:

circusAddWatcher(name, cmd, **kwargs)

- name: name of the watcher
- cmd: command to execute and to monitor
- any keyword argument is putted inside the watcher definition in the circus configuration file
  - SPECIAL CMD LINE ARGS: :args: arguments of cmdline :conf_priority: special keyword to specify the file order priority in configuration directory

Helpers macros

Salt Those utilities have the following goals:

- Generate the hooks for orchestration purpose (macro: salt_dummies)
- Install makina states with our custom layout (macro: install_makina_states)
- Configure a salt master using this layout (macro: install_master)
- Configure a salt minion using this layout (macro: install_minion)

They are used in the base **controllers** states to install the (**master**)salt daemons.

All of the layout and base parameters can be controlled at state & pillar level

Three macros are exposed:

- install_makina_states: install the common saltstack stuff
- install_master: master specific bits
- install_minion: minion specific bits

We extensivly use the **mc_services** & the **mc_salt** modules registries.

Common values must be updated in the common section, and the same for **master** or **minion** sections.

To override values in pillar .. code-block:: yaml

salt:

common:

- prefix: /foobar

master:

- ret_port: 4706

minion:

- master_port: 4706

mastersalt:

common:

- prefix: /foobar/mastersalt

master:

- ret_port: 4506

minion:

- master_port: 4506

Formulas & code management:

- We do not like formulas served with gitfs as it may not be resilient to network problems and also consume more network resources.
- For thus, we do prior checkouts and then we have a local checkout of the formula repository, and we then must link the inner formula subfolder inside our salt states tree root.

That's why you will maybe add to core repositories new formulaes by modyfing the confRepos keys as follow

```
salt:
 common:
  confRepos:
    <id>:
      name: git url of the repo
      target: path on the filesystem
      link: (optionnal and mainly useful for formulaes)
        name: target path name on the fs
        target: absolute path of the symlink
salt:
 common:
  confRepos:
    docker-formulae:
      name: https://github.com/saltstack-formulas/docker-formula.git
      target: '{salt_root}/formulas/docker'
      link:
        name: '{salt_root}/docker'
        target: '{salt_root}/formulas/docker/docker'
```

Apache helper

php helpers

2.1.4 Misc & Obsolete

Misc & Obsolete

ubuntu log to package and get PPA for a package

Configure sources if not already done

```
deb     http://mirror.ovh.net/ubuntu/ trusty main restricted universe
multiverse
deb-src http://mirror.ovh.net/ubuntu/ trusty main restricted
deb     http://mirror.ovh.net/ubuntu/ trusty-updates main restricted
universe multiverse
deb     http://security.ubuntu.com/ubuntu trusty-security main restricted
universe multiverse
```

Add debian developper packages

```
apt-get install -y bzr git devscripts bzr-builddeb pbuilder ubuntu-dev-tools distro-info-data
```

configure /etc/pbuilderrc MIRRORSITE to use:

```
http://mirror.ovh.net/ubuntu/
```

Init

```
cd && mkdir pkg && cd pkg
```

Patch package get source:

```
bzr branch lp:~ubuntu-branches/ubuntu/trusty/net-snmp/trusty/ snmp
```

do appropriate modifications:

```
vim snmp/$WTF
```

add new version:

```
dch -v 5.7.2~dfsg-8.1ubuntu4
```

commit:

```
bzr whoami "Mathieu Le Marec - Pasquet <kiorky@cryptelium.net>"
bzr commit -m  "Add missing net-snmp-create-v3-user"
```

Build source & bin

```
apt-get builddep snmpd
bzr builddeb
bzr builddeb -- -S -sa
```

Upload ~/.dput.cfg:

```
[snmp]
fqdn = ppa.launchpad.net
method = sftp
incoming = ~makinacorpus/snmp/ubuntu/
login = kiorky
allow_unsigned_uploads = 0
```

```
dput snmp net-snmp_5.7.2~dfsg-8.1ubuntu4_source.changes
```

Execution modes

OBSOLETE We do not use that much execution modes, prefer to split your formulaes in small chunks. See for example the "makina-states.services.monitoring.supervisor" state.

As we now extensivly use auto inclusion, we are particularly exposed to the **state bloat megalomania** when including a small little state will make rebuild the most part of makina-states. To prevent this, there are 2 main modes of execution and when the full mode will configure from end to end your machine, the standalone mode will skip most of the included states but also some of your currently called sls file.

The main use case is that the first time, you need to install a project and do a lot of stuff, but on the other runs, you just need to pull your new changesets and reload apache. The full mode is the standart mode, and the **light** mode is called **standalone**. That's why you ll see most formulaes declined in two sls flavors.

By convention, in those formulaes, we make the **standalone** one as a macro taking at least a **full** boolean parameter and then call it at the end with **full=false**. Then we write the normal sls calling this same little macro with **full=true**.

It is then up to the user of in the other sls includes to choose which sls to apply for the specific use case.

For example, consider the following sls files

- /srv/salt/makina-states/services/awesome/mysuperservice-standalone.sls:

```
{% macro do(full=True) %}
{% if full %}
OneLongStep:
    super.long: []
{% endif %}

OneShortStep:
    super.short []
{% endmacro %}
{{ do(full=False) }}
```

- /srv/salt/makina-states/services/awesome/mysuperservice.sls:

```
{% import "makina-states/services/awesome/mysuperservice-standalone.sls" as base with context %}
{{ do(full=True) }}
```

By hand, i will do the first time:

```
salt-call -lall makina-states.services.awesome.mysuperservice
```

And in a second time:

```
salt-call -lall makina-states.services.awesome.mysuperservice-standalone
```

As you can guess, OneLongStep will only be called the first time, and OneShortStep will be called both time.

Now, to make the inclusion dynamic in an intermediary sls file, you can also do a condtionnal include:

```
include:
    {% if cond %}
    - makina-states.services.awesome.mysuperservice-standalone
    {% else %}
    - makina-states.services.awesome.mysuperservice
    {% endif %}
```

Macros

OBSOLETE We know do not use any more base macros but access directly to salt execution modules, read some formulaes to see in action.

Base macros To access the registries and make a thin and easier higth level settings API, we use macros as importable modules inside the states files. Thus, we have in _macros those macros:

- **_macros/funcs.jinja** macro containing some helper functons
- **_macros/localsettings.jinja** macro related to localsettings
- **_macros/controllers.jinja** macro related to controllers
- **_macros/nodetypes.jinja** macro related to nodetypes
- **_macros/services.jinja** macro related to services

Utilities macros For certain complicated states, we have made some macros to leverage:

- The use of those states in your own states files
- The use of the settings themselves

The macros:

- **_macros/apache.jinja** macro containing some helpers to make vhosts
- **_macros/php.jinja** macro containing some helpers to make phpfpm pools
- **_macros/salt.jinja** internal macros to configure all the salt infra.

Bootstrap registries related helpers

Generic functions helpers to work with registries

Controller kind helpers

localsettings kind helpers

nodetypes kind helpers

Services kind helpers

This macro leverage the usage of **mc_services** module by having some registry shortcuts, please have a look at the code of the macro to know what settings are availables.

To use it in your states, just do something like that

```
{% import "makina-states/_macros/services.jinja" as services with context %}
foo:
    cmd.run:
        -name: echo {{ services.apacheSettings.mpm}}
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

m

mc_states.grains.makina_grains, 131
mc_states.modules.mc_apache, 75
mc_states.modules.mc_apparmor, 75
mc_states.modules.mc_autoupgrade, 77
mc_states.modules.mc_bind, 77
mc_states.modules.mc_bootstraps, 54
mc_states.modules.mc_burp, 80
mc_states.modules.mc_casperjs, 82
mc_states.modules.mc_circus, 83
mc_states.modules.mc_cloud, 88
mc_states.modules.mc_cloud_compute_node, 83
mc_states.modules.mc_cloud_controller, 85
mc_states.modules.mc_cloud_images, 86
mc_states.modules.mc_cloud_lxc, 87
mc_states.modules.mc_cloud_saltify, 89
mc_states.modules.mc_controllers, 55
mc_states.modules.mc_dbsmartbackup, 90
mc_states.modules.mc_dbus, 74
mc_states.modules.mc_dhcpd, 91
mc_states.modules.mc_djutils, 71
mc_states.modules.mc_dns, 72
mc_states.modules.mc_dnshelpers, 71
mc_states.modules.mc_dumper, 90
mc_states.modules.mc_env, 91
mc_states.modules.mc_etckeeper, 92
mc_states.modules.mc_etherpad, 92
mc_states.modules.mc_fail2ban, 92
mc_states.modules.mc_firewalld, 72
mc_states.modules.mc_grub, 93
mc_states.modules.mc_haproxy, 94
mc_states.modules.mc_icinga, 94
mc_states.modules.mc_icinga_web, 94
mc_states.modules.mc_java, 97
mc_states.modules.mc_kernel, 97
mc_states.modules.mc_locales, 98
mc_states.modules.mc_localsettings, 55
mc_states.modules.mc_locations, 98
mc_states.modules.mc_logrotate, 99
mc_states.modules.mc_lxc, 99
mc_states.modules.mc_macros, 122
mc_states.modules.mc_memcached, 99
mc_states.modules.mc_mongodb, 99
mc_states.modules.mc_mumble, 99
mc_states.modules.mc_mvn, 72
mc_states.modules.mc_mysql, 100
mc_states.modules.mc_nagvis, 101
mc_states.modules.mc_network, 102
mc_states.modules.mc_nginx, 103
mc_states.modules.mc_nodejs, 106
mc_states.modules.mc_nodetypes, 55
mc_states.modules.mc_ntp, 106
mc_states.modules.mc_pgsql, 106
mc_states.modules.mc_phantomjs, 107
mc_states.modules.mc_php, 107
mc_states.modules.mc_pkgs, 109
mc_states.modules.mc_pnp4nagios, 110
mc_states.modules.mc_postfix, 111
mc_states.modules.mc_project, 56
mc_states.modules.mc_project_2, 56
mc_states.modules.mc_provider, 112
mc_states.modules.mc_psad, 112
mc_states.modules.mc_pureftpd, 112
mc_states.modules.mc_python, 114
mc_states.modules.mc_rabbitmq, 114
mc_states.modules.mc_rdiffbackup, 114
mc_states.modules.mc_redis, 114
mc_states.modules.mc_remote, 66
mc_states.modules.mc_rsyslog, 114
mc_states.modules.mc_rvm, 115
mc_states.modules.mc_salt, 115
mc_states.modules.mc_screen, 115
mc_states.modules.mc_services, 55
mc_states.modules.mc_shorewall, 115
mc_states.modules.mc_snmpd, 116
mc_states.modules.mc_ssh, 116
mc_states.modules.mc_ssl, 117
mc_states.modules.mc_supervisor, 120
mc_states.modules.mc_timezone, 120

mc_states.modules.mc_tomcat, 120
mc_states.modules.mc_ulogd, 121
mc_states.modules.mc_updatedb, 121
mc_states.modules.mc_usergroup, 121
mc_states.modules.mc_utils, 124
mc_states.modules.mc_uwsgi, 122
mc_states.modules.mc_www, 122
mc_states.runners.mc_api, 132
mc_states.runners.mc_cloud_compute_node, 134
mc_states.runners.mc_cloud_controller, 132
mc_states.runners.mc_cloud_kvm, 138
mc_states.runners.mc_cloud_lxc, 137
mc_states.runners.mc_cloud_saltify, 139
mc_states.runners.mc_cloud_vm, 136
mc_states.runners.mc_lxc, 132
mc_states.saltapi, 139
mc_states.states.bacula, 131
mc_states.states.mc_apache, 129
mc_states.states.mc_git, 129
mc_states.states.mc_postgres_database, 130
mc_states.states.mc_postgres_extension, 130
mc_states.states.mc_postgres_group, 130
mc_states.states.mc_postgres_user, 130
mc_states.states.mc_proxy, 131
mc_states.states.mc_registry, 131
mc_states.utils, 141

A

a2dismod() (in module mc_states.modules.mc_apache), 75
a2enmod() (in module mc_states.modules.mc_apache), 75
absent() (in module mc_states.states.mc_postgres_database), 130
absent() (in module mc_states.states.mc_postgres_extension), 130
absent() (in module mc_states.states.mc_postgres_group), 130
absent() (in module mc_states.states.mc_postgres_user), 131
add_aliased_interfaces() (in module mc_states.modules.mc_firewalld), 72
add_geomap_settings() (in module mc_states.modules.mc_nagvis), 101
add_map_settings() (in module mc_states.modules.mc_nagvis), 101
add_natted_networks() (in module mc_states.modules.mc_firewalld), 72
allow_lowlevel_states() (in module mc_states.modules.mc_controllers), 55
apply_sls() (in module mc_states.runners.mc_api), 132
assert_good_grains() (in module mc_states.modules.mc_utils), 124
autoinclude() (in module mc_states.modules.mc_macros), 122

B

backup_database() (in module mc_states.modules.mc_djutils), 72
build_docker_from_rootfs() (in module mc_states.modules.mc_cloud_images), 86

C

ca_ssl_certs() (in module mc_states.modules.mc_ssl), 117
cache_check() (in module mc_states.modules.mc_utils), 124
cached_zone_headers() (in module mc_states.modules.mc_bind), 77
check_version() (in module mc_states.modules.mc_apache), 75
clean_global_variables() (in module mc_states.modules.mc_icinga), 94
clean_salt_git_commit() (in module mc_states.modules.mc_project_2), 56
cleanup_allocated_ips() (in module mc_states.modules.mc_cloud_compute_node), 83
cleanup_ports_mapping() (in module mc_states.modules.mc_cloud_compute_node), 83
cli() (in module mc_states.runners.mc_api), 132
client() (in module mc_states.saltapi), 140
ClientNotActivated (class in mc_states.modules.mc_provider), 112
complete_rich_rules() (in module mc_states.modules.mc_firewalld), 72
composer_command() (in module mc_states.modules.mc_php), 107
concat_res_or_rets() (in module mc_states.saltapi), 140
config_settings() (in module mc_states.modules.mc_uwsgi), 122
configure_firewall() (in module mc_states.runners.mc_cloud_compute_node), 134
configure_host() (in module mc_states.runners.mc_cloud_compute_node), 134
configure_hostsfile() (in module mc_states.runners.mc_cloud_compute_node), 134
configure_images() (in module mc_states.runners.mc_cloud_lxc), 137
configure_install_vt() (in module mc_states.runners.mc_cloud_kvm), 138
configure_install_vt() (in module mc_states.runners.mc_cloud_lxc), 137
configure_network() (in module

mc_states.runners.mc_cloud_compute_node), 134
configure_node() (in module mc_states.runners.mc_cloud_controller), 132
configure_prevt() (in module mc_states.runners.mc_cloud_compute_node), 134
configure_reverse_proxy() (in module mc_states.runners.mc_cloud_compute_node), 134
configure_sshkeys() (in module mc_states.runners.mc_cloud_compute_node), 134
configure_sslcerts() (in module mc_states.runners.mc_cloud_compute_node), 134
configure_vm() (in module mc_states.runners.mc_cloud_controller), 132
construct_registry_configuration() (in module mc_states.modules.mc_macros), 122
copy_dictupdate() (in module mc_states.modules.mc_utils), 124
cyaml_dump() (in module mc_states.modules.mc_dumper), 90
cyaml_dump() (in module mc_states.modules.mc_utils), 124
cyaml_load() (in module mc_states.modules.mc_dumper), 90
cyaml_load() (in module mc_states.modules.mc_utils), 124

D

default_net() (in module mc_states.modules.mc_network), 103
default_settings() (in module mc_states.modules.mc_cloud), 88
default_settings() (in module mc_states.modules.mc_cloud_compute_node), 83
default_settings() (in module mc_states.modules.mc_cloud_images), 86
default_settings() (in module mc_states.modules.mc_cloud_saltify), 89
defaults() (in module mc_states.modules.mc_utils), 124
del_conf_for_target() (in module mc_states.modules.mc_cloud_compute_node), 84
delete_remote() (in module mc_states.modules.mc_remote), 66
deploy() (in module mc_states.modules.mc_project_2), 56
deploy() (in module mc_states.runners.mc_cloud_compute_node), 134
deploy() (in module mc_states.runners.mc_cloud_controller), 132
deployed() (in module mc_states.states.mc_apache), 129
destroy() (in module mc_states.runners.mc_api), 132
destroy() (in module mc_states.runners.mc_cloud_lxc), 137
destroy() (in module mc_states.runners.mc_cloud_vm), 136
dictupdate() (in module mc_states.modules.mc_utils), 124
dns_conf() (in module mc_states.runners.mc_cloud_controller), 132
domain_match() (in module mc_states.modules.mc_ssl), 117
domain_match_wildcard() (in module mc_states.modules.mc_ssl), 117

E

edit_configuration_object_settings() (in module mc_states.modules.mc_icinga), 94
exclude_module() (in module mc_states.states.mc_apache), 129
exists() (in module mc_states.runners.mc_cloud_controller), 133
ext_ip() (in module mc_states.modules.mc_network), 103
ext_pillar() (in module mc_states.modules.mc_cloud), 88
ext_pillar() (in module mc_states.modules.mc_cloud_compute_node), 84
ext_pillar() (in module mc_states.modules.mc_cloud_images), 86
extpillar_settings() (in module mc_states.modules.mc_cloud), 89

F

fdconfig() (in module mc_states.states.bacula), 131
feed_http_reverse_proxy_for_target() (in module mc_states.modules.mc_cloud_compute_node), 84
feed_network_mappings_for_target() (in module mc_states.modules.mc_cloud_compute_node), 84
file_read() (in module mc_states.modules.mc_utils), 125
filecache_fun() (in module mc_states.modules.mc_macros), 122
filter_compute_nodes() (in module mc_states.runners.mc_cloud_compute_node), 134
filter_compute_nodes() (in module mc_states.runners.mc_cloud_saltify), 139

find_ip_for_vm() (in module mc_states.modules.mc_cloud_compute_node), 84
find_mac_for_vm() (in module mc_states.modules.mc_cloud_compute_node), 84
find_password_for_vm() (in module mc_states.modules.mc_cloud_compute_node), 84
format_resolve() (in module mc_states.modules.mc_utils), 125
fpmpool_settings() (in module mc_states.modules.mc_php), 107

G

gather_expositions() (in module mc_states.modules.mc_cloud), 89
generate_stored_password() (in module mc_states.modules.mc_utils), 125
get() (in module mc_states.modules.mc_utils), 125
get_all_vts() (in module mc_states.modules.mc_cloud_compute_node), 84
get_allocated_ips() (in module mc_states.modules.mc_cloud_compute_node), 84
get_broadcast() (in module mc_states.modules.mc_network), 103
get_cert_for() (in module mc_states.modules.mc_ssl), 118
get_common_vars() (in module mc_states.modules.mc_project), 56
get_conf_for_target() (in module mc_states.modules.mc_cloud_compute_node), 85
get_conf_for_vm() (in module mc_states.modules.mc_cloud_compute_node), 85
get_configuration() (in module mc_states.modules.mc_project_2), 57
get_configuration_item() (in module mc_states.modules.mc_project_2), 58
get_configured_cert() (in module mc_states.modules.mc_ssl), 118
get_custom_cert_for() (in module mc_states.modules.mc_ssl), 118
get_default_sysadmins() (in module mc_states.modules.mc_usergroup), 121
get_dnss() (in module mc_states.modules.mc_network), 103
get_fo_broadcast() (in module mc_states.modules.mc_network), 103
get_fo_netmask() (in module mc_states.modules.mc_network), 103
get_gateway() (in module mc_states.modules.mc_network), 103
get_home() (in module mc_states.modules.mc_usergroup), 121
get_installed_cert_for() (in module mc_states.modules.mc_ssl), 118
get_local_cache() (in module mc_states.modules.mc_utils), 126
get_local_client() (in module mc_states.saltapi), 141
get_makina_grains() (in module mc_states.grains.makina_grains), 131
get_makina_grains() (in module mc_states.modules.mc_nodetypes), 55
get_mc_server() (in module mc_states.modules.mc_utils), 126
get_netmask() (in module mc_states.modules.mc_network), 103
get_pillar_fqdn() (in module mc_states.modules.mc_localsettings), 55
get_project() (in module mc_states.modules.mc_project_2), 58
get_registry() (in module mc_states.modules.mc_macros), 123
get_selfsigned_cert_for() (in module mc_states.modules.mc_ssl), 118
get_service_enabled_state() (in module mc_states.modules.mc_services), 55
get_service_function() (in module mc_states.modules.mc_services), 56
get_settings_for_object() (in module mc_states.modules.mc_icinga), 94
get_targets() (in module mc_states.modules.mc_cloud_compute_node), 85
get_uniq_keys_for() (in module mc_states.modules.mc_utils), 126
get_version() (in module mc_states.modules.mc_apache), 75
get_view() (in module mc_states.modules.mc_bind), 78
get_vms() (in module mc_states.modules.mc_cloud_compute_node), 85
get_vms_for_target() (in module mc_states.modules.mc_cloud_compute_node), 85
get_vms_per_type() (in module mc_states.modules.mc_cloud_compute_node), 85
get_vts() (in module mc_states.modules.mc_cloud_compute_node), 85
get_zone() (in module mc_states.modules.mc_bind), 78

H

has_system_services_manager() (in module

mc_states.modules.mc_nodetypes), 55
hash() (in module mc_states.modules.mc_utils), 126
highstate() (in module mc_states.modules.mc_remote), 66
hook() (in module mc_states.states.mc_proxy), 131

I

ImgStepError (class in mc_states.saltapi), 139
include_module() (in module mc_states.states.mc_apache), 129
init_pillar_dir() (in module mc_states.modules.mc_project_2), 58
init_project() (in module mc_states.modules.mc_project_2), 59
init_remote_structure() (in module mc_states.modules.mc_project_2), 59
init_repo() (in module mc_states.modules.mc_project_2), 59
init_salt_dir() (in module mc_states.modules.mc_project_2), 59
init_ssh_user_keys() (in module mc_states.modules.mc_project_2), 59
install() (in module mc_states.modules.mc_project_2), 60
install_vt() (in module mc_states.runners.mc_cloud_kvm), 138
install_vt() (in module mc_states.runners.mc_cloud_lxc), 138
install_vts() (in module mc_states.runners.mc_cloud_compute_node), 134
interactive_ssh() (in module mc_states.modules.mc_remote), 67
invalidate_memoize_cache() (in module mc_states.modules.mc_utils), 126
is_a_bool() (in module mc_states.modules.mc_utils), 126
is_a_complex() (in module mc_states.modules.mc_utils), 126
is_a_dict() (in module mc_states.modules.mc_utils), 126
is_a_float() (in module mc_states.modules.mc_utils), 126
is_a_int() (in module mc_states.modules.mc_utils), 126
is_a_list() (in module mc_states.modules.mc_utils), 126
is_a_long() (in module mc_states.modules.mc_utils), 126
is_a_number() (in module mc_states.modules.mc_utils), 126
is_a_set() (in module mc_states.modules.mc_utils), 126
is_a_str() (in module mc_states.modules.mc_utils), 127
is_a_tuple() (in module mc_states.modules.mc_utils), 127
is_active() (in module mc_states.modules.mc_macros), 123
is_item_active() (in module mc_states.modules.mc_macros), 123
is_iter() (in module mc_states.modules.mc_utils), 127
is_lxc() (in module mc_states.modules.mc_cloud_lxc), 87
is_lxc() (in module mc_states.modules.mc_lxc), 99
iyaml_dump() (in module mc_states.modules.mc_dumper), 90
iyaml_dump() (in module mc_states.modules.mc_utils), 127

J

json_dump() (in module mc_states.modules.mc_dumper), 90
json_dump() (in module mc_states.modules.mc_utils), 127
json_load() (in module mc_states.modules.mc_dumper), 90
json_load() (in module mc_states.modules.mc_utils), 127

L

latest() (in module mc_states.states.mc_git), 129
link() (in module mc_states.modules.mc_project_2), 60
link_into_root() (in module mc_states.modules.mc_project_2), 60
link_pillar() (in module mc_states.modules.mc_project_2), 60
link_salt() (in module mc_states.modules.mc_project_2), 60
list_cache_keys() (in module mc_states.modules.mc_utils), 127
load_certs() (in module mc_states.modules.mc_ssl), 119
load_sample() (in module mc_states.modules.mc_project_2), 60
load_selfsigned_certs() (in module mc_states.modules.mc_ssl), 119
local_mastersalt_call() (in module mc_states.modules.mc_remote), 67
local_minion_id() (in module mc_states.modules.mc_utils), 127
local_salt_call() (in module mc_states.modules.mc_remote), 67

M

magicstring() (in module mc_states.modules.mc_utils), 127
manage_file() (in module mc_states.modules.mc_utils), 127
mastersalt_call() (in module mc_states.modules.mc_remote), 67
mc_states.grains.makina_grains (module), 131
mc_states.modules.mc_apache (module), 75
mc_states.modules.mc_apparmor (module), 75
mc_states.modules.mc_autoupgrade (module), 77
mc_states.modules.mc_bind (module), 77
mc_states.modules.mc_bootstraps (module), 54
mc_states.modules.mc_burp (module), 80

mc_states.modules.mc_casperjs (module), 82
mc_states.modules.mc_circus (module), 83
mc_states.modules.mc_cloud (module), 88
mc_states.modules.mc_cloud_compute_node (module), 83
mc_states.modules.mc_cloud_controller (module), 85
mc_states.modules.mc_cloud_images (module), 86
mc_states.modules.mc_cloud_lxc (module), 87
mc_states.modules.mc_cloud_saltify (module), 89
mc_states.modules.mc_controllers (module), 55
mc_states.modules.mc_dbsmartbackup (module), 90
mc_states.modules.mc_dbus (module), 74
mc_states.modules.mc_dhcpd (module), 91
mc_states.modules.mc_djutils (module), 71
mc_states.modules.mc_dns (module), 72
mc_states.modules.mc_dnshelpers (module), 71
mc_states.modules.mc_dumper (module), 90
mc_states.modules.mc_env (module), 91
mc_states.modules.mc_etckeeper (module), 92
mc_states.modules.mc_etherpad (module), 92
mc_states.modules.mc_fail2ban (module), 92
mc_states.modules.mc_firewalld (module), 72
mc_states.modules.mc_grub (module), 93
mc_states.modules.mc_haproxy (module), 94
mc_states.modules.mc_icinga (module), 94
mc_states.modules.mc_icinga_web (module), 94
mc_states.modules.mc_java (module), 97
mc_states.modules.mc_kernel (module), 97
mc_states.modules.mc_locales (module), 98
mc_states.modules.mc_localsettings (module), 55
mc_states.modules.mc_locations (module), 98
mc_states.modules.mc_logrotate (module), 99
mc_states.modules.mc_lxc (module), 99
mc_states.modules.mc_macros (module), 122
mc_states.modules.mc_memcached (module), 99
mc_states.modules.mc_mongodb (module), 99
mc_states.modules.mc_mumble (module), 99
mc_states.modules.mc_mvn (module), 72
mc_states.modules.mc_mysql (module), 100
mc_states.modules.mc_nagvis (module), 101
mc_states.modules.mc_network (module), 102
mc_states.modules.mc_nginx (module), 103
mc_states.modules.mc_nodejs (module), 106
mc_states.modules.mc_nodetypes (module), 55
mc_states.modules.mc_ntp (module), 106
mc_states.modules.mc_pgsql (module), 106
mc_states.modules.mc_phantomjs (module), 107
mc_states.modules.mc_php (module), 107
mc_states.modules.mc_pkgs (module), 109
mc_states.modules.mc_pnp4nagios (module), 110
mc_states.modules.mc_postfix (module), 111
mc_states.modules.mc_project (module), 56
mc_states.modules.mc_project_2 (module), 56
mc_states.modules.mc_provider (module), 112
mc_states.modules.mc_psad (module), 112
mc_states.modules.mc_pureftpd (module), 112
mc_states.modules.mc_python (module), 114
mc_states.modules.mc_rabbitmq (module), 114
mc_states.modules.mc_rdiffbackup (module), 114
mc_states.modules.mc_redis (module), 114
mc_states.modules.mc_remote (module), 66
mc_states.modules.mc_rsyslog (module), 114
mc_states.modules.mc_rvm (module), 115
mc_states.modules.mc_salt (module), 115
mc_states.modules.mc_screen (module), 115
mc_states.modules.mc_services (module), 55
mc_states.modules.mc_shorewall (module), 115
mc_states.modules.mc_snmpd (module), 116
mc_states.modules.mc_ssh (module), 116
mc_states.modules.mc_ssl (module), 117
mc_states.modules.mc_supervisor (module), 120
mc_states.modules.mc_timezone (module), 120
mc_states.modules.mc_tomcat (module), 120
mc_states.modules.mc_ulogd (module), 121
mc_states.modules.mc_updatedb (module), 121
mc_states.modules.mc_usergroup (module), 121
mc_states.modules.mc_utils (module), 124
mc_states.modules.mc_uwsgi (module), 122
mc_states.modules.mc_www (module), 122
mc_states.runners.mc_api (module), 132
mc_states.runners.mc_cloud_compute_node (module), 134
mc_states.runners.mc_cloud_controller (module), 132
mc_states.runners.mc_cloud_kvm (module), 138
mc_states.runners.mc_cloud_lxc (module), 137
mc_states.runners.mc_cloud_saltify (module), 139
mc_states.runners.mc_cloud_vm (module), 136
mc_states.runners.mc_lxc (module), 132
mc_states.saltapi (module), 139
mc_states.states.bacula (module), 131
mc_states.states.mc_apache (module), 129
mc_states.states.mc_git (module), 129
mc_states.states.mc_postgres_database (module), 130
mc_states.states.mc_postgres_extension (module), 130
mc_states.states.mc_postgres_group (module), 130
mc_states.states.mc_postgres_user (module), 130
mc_states.states.mc_proxy (module), 131
mc_states.states.mc_registry (module), 131
mc_states.utils (module), 141
memoize_cache() (in module mc_states.modules.mc_utils), 127
metadata() (in module mc_states.modules.mc_bootstraps), 54
metadata() (in module mc_states.modules.mc_controllers), 55
metadata() (in module mc_states.modules.mc_localsettings), 55

metadata() (in module mc_states.modules.mc_nodetypes), 55
mod_watch() (in module mc_states.states.mc_proxy), 131
MoveError, 137
msgpack_dump() (in module mc_states.modules.mc_dumper), 90
msgpack_dump() (in module mc_states.modules.mc_utils), 127
msgpack_load() (in module mc_states.modules.mc_dumper), 90
msgpack_load() (in module mc_states.modules.mc_utils), 127
msr() (in module mc_states.modules.mc_utils), 127

N

NoRegistryLoaderFound (class in mc_states.saltapi), 139
ns_whois() (in module mc_states.modules.mc_network), 103
nyaml_dump() (in module mc_states.modules.mc_utils), 127

O

objects() (in module mc_states.modules.mc_icinga), 94
old_latest() (in module mc_states.states.mc_git), 129
old_yaml_dump() (in module mc_states.modules.mc_dumper), 90
old_yaml_dump() (in module mc_states.modules.mc_utils), 127
orchestrate() (in module mc_states.modules.mc_project_2), 60
orchestrate() (in module mc_states.runners.mc_cloud_compute_node), 134
orchestrate() (in module mc_states.runners.mc_cloud_controller), 133
orchestrate() (in module mc_states.runners.mc_cloud_saltify), 139
orchestrate() (in module mc_states.runners.mc_cloud_vm), 136
orchestrate_task() (in module mc_states.modules.mc_project_2), 62
output() (in module mc_states.modules.mc_utils), 127

P

pack_docker() (in module mc_states.modules.mc_cloud_images), 86
pack_dump_local_registry() (in module mc_states.modules.mc_macros), 123
PortConflictError, 139
post_configure() (in module mc_states.runners.mc_cloud_controller), 133
post_deploy() (in module mc_states.runners.mc_cloud_compute_node), 135
post_deploy_controller() (in module mc_states.runners.mc_cloud_kvm), 138
post_deploy_controller() (in module mc_states.runners.mc_cloud_lxc), 138
post_post_deploy_compute_node() (in module mc_states.runners.mc_cloud_kvm), 138
post_post_deploy_compute_node() (in module mc_states.runners.mc_cloud_lxc), 138
post_provision() (in module mc_states.runners.mc_cloud_vm), 136
post_provision_compute_nodes() (in module mc_states.runners.mc_cloud_compute_node), 135
post_provision_vms() (in module mc_states.runners.mc_cloud_vm), 136
prepare_controller() (in module mc_states.runners.mc_cloud_controller), 133
present() (in module mc_states.states.mc_postgres_database), 130
present() (in module mc_states.states.mc_postgres_extension), 130
present() (in module mc_states.states.mc_postgres_group), 130
present() (in module mc_states.states.mc_postgres_user), 131
provision() (in module mc_states.runners.mc_cloud_vm), 136
provision_compute_nodes() (in module mc_states.runners.mc_cloud_compute_node), 135
provision_vms() (in module mc_states.runners.mc_cloud_vm), 136
purge_memoize_cache() (in module mc_states.modules.mc_utils), 127
push_changesets_in() (in module mc_states.modules.mc_project_2), 63

R

rebuild_ms_docker() (in module mc_states.modules.mc_cloud_images), 86
reconfigure_front() (in module mc_states.runners.mc_cloud_compute_node), 135
register() (in module mc_states.modules.mc_macros), 124
register_dns_masters() (in module mc_states.modules.mc_dnshelpers), 71
register_memcache_first() (in module mc_states.modules.mc_utils), 127

registry() (in module mc_states.modules.mc_bootstraps), 54
registry() (in module mc_states.modules.mc_controllers), 55
registry() (in module mc_states.modules.mc_localsettings), 55
registry() (in module mc_states.modules.mc_nodetypes), 55
remote_deploy() (in module mc_states.modules.mc_project_2), 63
remote_project_hook() (in module mc_states.modules.mc_project_2), 64
remote_run_task() (in module mc_states.modules.mc_project_2), 64
remote_task() (in module mc_states.modules.mc_project_2), 64
remove() (in module mc_states.runners.mc_api), 132
remove() (in module mc_states.runners.mc_cloud_controller), 133
remove() (in module mc_states.runners.mc_cloud_lxc), 138
remove_allocated_ip() (in module mc_states.modules.mc_cloud_compute_node), 85
remove_cache_entry() (in module mc_states.modules.mc_utils), 127
remove_configuration_object() (in module mc_states.modules.mc_icinga), 94
remove_entry() (in module mc_states.modules.mc_utils), 128
remove_path() (in module mc_states.modules.mc_project_2), 64
RenderError (class in mc_states.saltapi), 139
report() (in module mc_states.modules.mc_project_2), 64
report() (in module mc_states.runners.mc_cloud_compute_node), 135
report() (in module mc_states.runners.mc_cloud_controller), 133
restore_from() (in module mc_states.modules.mc_djutils), 72
rich_rules() (in module mc_states.modules.mc_firewalld), 72
rollback() (in module mc_states.modules.mc_project_2), 64
rotate_archives() (in module mc_states.modules.mc_project_2), 64
run() (in module mc_states.modules.mc_remote), 67
run_task() (in module mc_states.modules.mc_project_2), 65
run_vt_hook() (in module mc_states.runners.mc_cloud_compute_node), 135
run_vt_hook() (in module mc_states.runners.mc_cloud_controller), 133

S

salt_call() (in module mc_states.modules.mc_remote), 67
salt_root() (in module mc_states.modules.mc_utils), 128
SaltCallFailure (class in mc_states.saltapi), 140
saltify() (in module mc_states.runners.mc_cloud_saltify), 139
saltify_node() (in module mc_states.runners.mc_cloud_controller), 134
search_aliased_interfaces() (in module mc_states.modules.mc_firewalld), 73
search_matching_certificate() (in module mc_states.modules.mc_ssl), 119
search_matching_selfsigned_certificate() (in module mc_states.modules.mc_ssl), 119
selfsigned_last() (in module mc_states.modules.mc_ssl), 119
selfsigned_ssl_certs() (in module mc_states.modules.mc_ssl), 119
set_allocated_ip() (in module mc_states.modules.mc_cloud_compute_node), 85
set_conf_for_target() (in module mc_states.modules.mc_cloud_compute_node), 85
set_configuration() (in module mc_states.modules.mc_project_2), 65
settings() (in module mc_states.modules.mc_apache), 76
settings() (in module mc_states.modules.mc_apparmor), 75
settings() (in module mc_states.modules.mc_autoupgrade), 77
settings() (in module mc_states.modules.mc_bind), 78
settings() (in module mc_states.modules.mc_bootstraps), 54
settings() (in module mc_states.modules.mc_burp), 80
settings() (in module mc_states.modules.mc_casperjs), 83
settings() (in module mc_states.modules.mc_circus), 83
settings() (in module mc_states.modules.mc_cloud), 89
settings() (in module mc_states.modules.mc_cloud_compute_node), 85
settings() (in module mc_states.modules.mc_cloud_controller), 86
settings() (in module mc_states.modules.mc_cloud_images), 87
settings() (in module mc_states.modules.mc_controllers), 55
settings() (in module mc_states.modules.mc_dbsmartbackup), 90
settings() (in module mc_states.modules.mc_dbus), 75
settings() (in module mc_states.modules.mc_dhcpd), 91

settings() (in module mc_states.modules.mc_dnshelpers), 71
settings() (in module mc_states.modules.mc_env), 92
settings() (in module mc_states.modules.mc_etckeeper), 92
settings() (in module mc_states.modules.mc_etherpad), 92
settings() (in module mc_states.modules.mc_fail2ban), 92
settings() (in module mc_states.modules.mc_firewalld), 73
settings() (in module mc_states.modules.mc_grub), 93
settings() (in module mc_states.modules.mc_haproxy), 94
settings() (in module mc_states.modules.mc_icinga_web), 96
settings() (in module mc_states.modules.mc_java), 97
settings() (in module mc_states.modules.mc_kernel), 98
settings() (in module mc_states.modules.mc_locales), 98
settings() (in module mc_states.modules.mc_localsettings), 55
settings() (in module mc_states.modules.mc_locations), 99
settings() (in module mc_states.modules.mc_logrotate), 99
settings() (in module mc_states.modules.mc_lxc), 99
settings() (in module mc_states.modules.mc_memcached), 99
settings() (in module mc_states.modules.mc_mongodb), 99
settings() (in module mc_states.modules.mc_mumble), 100
settings() (in module mc_states.modules.mc_mvn), 72
settings() (in module mc_states.modules.mc_mysql), 100
settings() (in module mc_states.modules.mc_nagvis), 101
settings() (in module mc_states.modules.mc_network), 103
settings() (in module mc_states.modules.mc_nginx), 103
settings() (in module mc_states.modules.mc_nodejs), 106
settings() (in module mc_states.modules.mc_nodetypes), 55
settings() (in module mc_states.modules.mc_ntp), 106
settings() (in module mc_states.modules.mc_pgsql), 106
settings() (in module mc_states.modules.mc_phantomjs), 107
settings() (in module mc_states.modules.mc_php), 108
settings() (in module mc_states.modules.mc_pkgs), 109
settings() (in module mc_states.modules.mc_pnp4nagios), 110
settings() (in module mc_states.modules.mc_postfix), 111
settings() (in module mc_states.modules.mc_provider), 112
settings() (in module mc_states.modules.mc_psad), 112
settings() (in module mc_states.modules.mc_pureftpd), 112
settings() (in module mc_states.modules.mc_python), 114
settings() (in module mc_states.modules.mc_rabbitmq), 114
settings() (in module mc_states.modules.mc_rdiffbackup), 114
settings() (in module mc_states.modules.mc_redis), 114
settings() (in module mc_states.modules.mc_rsyslog), 114
settings() (in module mc_states.modules.mc_rvm), 115
settings() (in module mc_states.modules.mc_salt), 115
settings() (in module mc_states.modules.mc_screen), 115
settings() (in module mc_states.modules.mc_services), 56
settings() (in module mc_states.modules.mc_shorewall), 115
settings() (in module mc_states.modules.mc_snmpd), 116
settings() (in module mc_states.modules.mc_ssh), 116
settings() (in module mc_states.modules.mc_ssl), 119
settings() (in module mc_states.modules.mc_supervisor), 120
settings() (in module mc_states.modules.mc_timezone), 120
settings() (in module mc_states.modules.mc_tomcat), 120
settings() (in module mc_states.modules.mc_ulogd), 121
settings() (in module mc_states.modules.mc_updatedb), 121
settings() (in module mc_states.modules.mc_usergroup), 121
settings() (in module mc_states.modules.mc_uwsgi), 122
settings() (in module mc_states.modules.mc_www), 122
setup_database() (in module mc_states.modules.mc_djutils), 72
sf_release() (in module mc_states.modules.mc_cloud_images), 87
sls_() (in module mc_states.modules.mc_remote), 68
ssh() (in module mc_states.modules.mc_remote), 69
ssh_kwargs() (in module mc_states.modules.mc_remote), 69
ssh_retcode() (in module mc_states.modules.mc_remote), 70
ssh_transfer_dir() (in module mc_states.modules.mc_remote), 70
ssh_transfer_file() (in module mc_states.modules.mc_remote), 71
SSHCommandFailed (class in mc_states.saltapi), 139
SSHCommandFinished (class in mc_states.saltapi), 139
SSHCommandTimeout (class in mc_states.saltapi), 139
SSHExecError (class in mc_states.saltapi), 139
SSHInterruptError (class in mc_states.saltapi), 139
SSHLoginError (class in mc_states.saltapi), 139
SSHTimeoutError (class in mc_states.saltapi), 139
SSHTransferFailed (class in mc_states.saltapi), 139
SSHVtError (class in mc_states.saltapi), 140

ssl_certs() (in module mc_states.modules.mc_ssl), 119
ssl_chain() (in module mc_states.modules.mc_ssl), 120
ssl_infos() (in module mc_states.modules.mc_ssl), 120
ssl_key() (in module mc_states.modules.mc_ssl), 120
ssl_keys() (in module mc_states.modules.mc_ssl), 120
step() (in module mc_states.runners.mc_cloud_vm), 137
sync_git_directory() (in module mc_states.modules.mc_project_2), 65
sync_hooks_for_all() (in module mc_states.modules.mc_project_2), 65
sync_images() (in module mc_states.runners.mc_cloud_kvm), 138
sync_images() (in module mc_states.runners.mc_cloud_lxc), 138
sync_images() (in module mc_states.runners.mc_lxc), 132
sync_modules() (in module mc_states.modules.mc_project_2), 65
sync_remote_working_copy() (in module mc_states.modules.mc_project_2), 65
sync_working_copy() (in module mc_states.modules.mc_project_2), 65

T

target_extpillar() (in module mc_states.modules.mc_cloud_saltify), 90
target_for_vm() (in module mc_states.modules.mc_cloud_compute_node), 85
test_cache() (in module mc_states.modules.mc_utils), 128
TransformError (class in mc_states.saltapi), 140
traverse_dict() (in module mc_states.modules.mc_utils), 128

U

uncache_project() (in module mc_states.modules.mc_project_2), 66
uncached_get() (in module mc_states.modules.mc_utils), 128
unix_crypt() (in module mc_states.modules.mc_utils), 128
unlink() (in module mc_states.modules.mc_project_2), 66
unlink_pillar() (in module mc_states.modules.mc_project_2), 66
unlink_salt() (in module mc_states.modules.mc_project_2), 66
unregister() (in module mc_states.modules.mc_macros), 124
update() (in module mc_states.states.mc_registry), 131
update_local_registry() (in module mc_states.modules.mc_macros), 124
update_no_list() (in module mc_states.modules.mc_utils), 128
update_registry_params() (in module mc_states.modules.mc_macros), 124
upgrade_vt() (in module mc_states.runners.mc_cloud_kvm), 139
upgrade_vt() (in module mc_states.runners.mc_cloud_lxc), 138
upgrade_vts() (in module mc_states.runners.mc_cloud_compute_node), 136

V

vhost_settings() (in module mc_states.modules.mc_apache), 77
vhost_settings() (in module mc_states.modules.mc_nginx), 105
vm_extpillar() (in module mc_states.modules.mc_cloud_lxc), 87
vm_hostsfile() (in module mc_states.runners.mc_cloud_lxc), 138
vm_initial_highstate() (in module mc_states.runners.mc_cloud_vm), 137
vm_initial_setup() (in module mc_states.runners.mc_cloud_lxc), 138
vm_markers() (in module mc_states.runners.mc_cloud_vm), 137
vm_ping() (in module mc_states.runners.mc_cloud_vm), 137
vm_preprovision() (in module mc_states.runners.mc_cloud_lxc), 138
vm_preprovision() (in module mc_states.runners.mc_cloud_vm), 137
vm_reconfigure() (in module mc_states.runners.mc_cloud_lxc), 138
vm_spawn() (in module mc_states.runners.mc_cloud_lxc), 138
vm_sshkeys() (in module mc_states.runners.mc_cloud_vm), 137
vt_default_settings() (in module mc_states.modules.mc_cloud_lxc), 87
vt_extpillar() (in module mc_states.modules.mc_cloud_lxc), 88
vt_for_vm() (in module mc_states.modules.mc_cloud_compute_node), 85

W

whois_data() (in module mc_states.modules.mc_network), 103
working_copy_in_initial_state() (in module mc_states.modules.mc_project_2), 66
wrapper() (in module mc_states.modules.mc_pgsql), 107

Y

yaml_dump() (in module mc_states.modules.mc_dumper), 90
yaml_dump() (in module mc_states.modules.mc_utils), 128
yaml_load() (in module mc_states.modules.mc_dumper), 90
yaml_load() (in module mc_states.modules.mc_utils), 128
yamldump_arg() (in module mc_states.modules.mc_remote), 71
yencode() (in module mc_states.modules.mc_dumper), 90
yencode() (in module mc_states.modules.mc_utils), 128