SaltStack

Начало

SaltStack — это

- Система управления конфигурациями
- Шина на ZeroMQ[1]
- Python и модульность
- Профессия YAML-программист
- «Провиженинг» для IaaS
- В народе — «соль»

[1] https://zeromq.org/

Системы управления конфигурациями

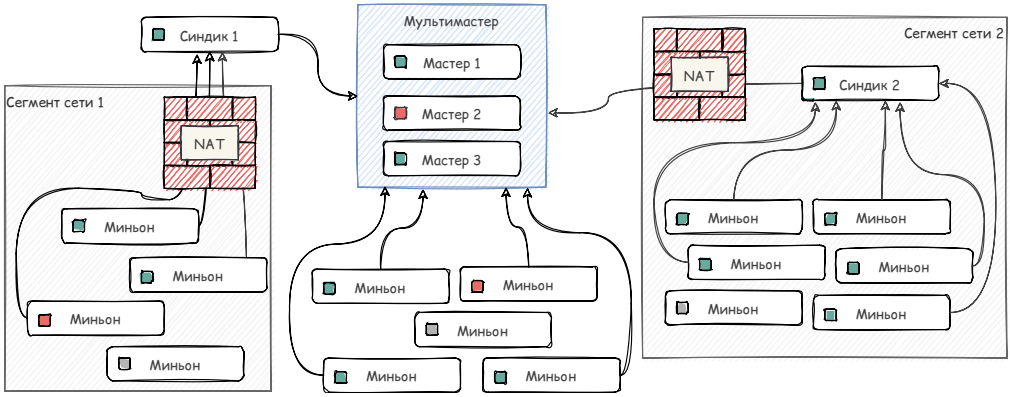
СУК	Год	На чём	DSL	Особенности
Puppet	2005	Ruby	Свой	Pull
Chef	2009	Ruby, Erlang	Ruby	Pull, веб-интерфейс
SaltStack	**2011**	**Python**[2]	**YAML**	**Pull (и Push), корп-версия**
Ansible	2012	Python, Ruby	YAML	Push (и Pull), Ansible Tower

[2] Сплошной Python

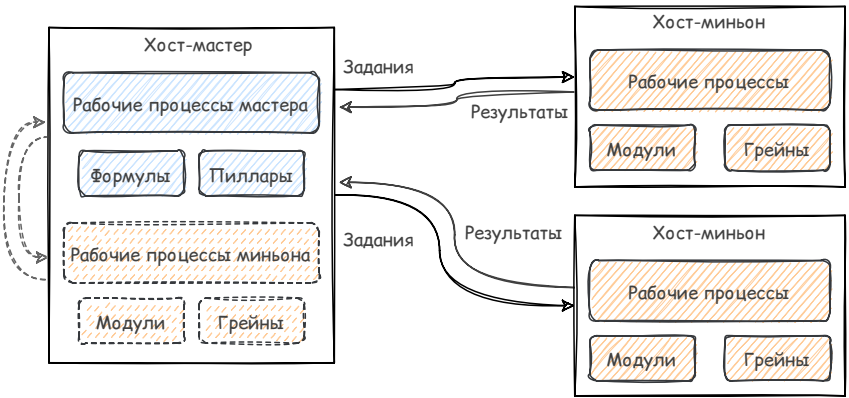
Глоссарий

В SaltStack	Значение	В Ansible
Мастер	Сервер	Узел (нода) управления
Миньон	Клиент	Управляемый узел
Формула	Декларативный конфиг клиента	Плейбук
Состояние	Элементарная единица конфига клиента	Таск
Грейн	Переменная, полученная от клиента	Факт
Топ-файл	Маппинг хостов	Инвентарь (хосты)
Пиллары	Устанавливаемая переменная	Инвентарь (переменные)

Топология


Синдик 1
Мультимастер
Мастер 1
Мастер 2
Мастер 3
Сегмент сети 1
NAT
Миньон
Миньон
Миньон
Миньон
Миньон
Миньон
Миньон
Миньон
Сегмент сети 2
NAT
Синдик 2
Миньон
Миньон
Миньон
Миньон
Миньон
Миньон

Архитектура



Данные

- Grains («крупицы соли»)
- Pillars («соляные столпы»)
- Salt Mine («соляная шахта»)

Данные

- Grains («крупицы соли»)
- Pillars («соляные столпы»)
- Salt Mine («соляная шахта»)

Данные

- Grains («крупицы соли»)
- Pillars («соляные столпы»)
- Salt Mine («соляная шахта»)

Типы модулей

- `salt.modules`
- `salt.states`
- `salt.grains`
- `salt.pillar`
- `salt.renderer`
- …
- Десятки их!

YAML

- Форматирование отступами
- Скалярные типы
- Мультистрочные литералы[3]
- Словари и последовательности
- Включает в себя JSON
- Наследование (на десерт)

```yaml
---

state:
  name: доминатор2010
  description: >-
    Сильнейший воин
    на этом побережье
  health: 100
  online: true
  party: []

inventory:
  shield: null
  bag:
    - flask: red potion
    - flask: blue potion
    - unknown_charm:

...
```

[3] https://yaml-multiline.info

Установка

0.6.0 … 2018.3.0 → 2019.2.0 → 3000 … **3004.1**

- Salt Bootstrap[4]
- Из репозитория дистрибутива
- Из репозитория SaltStack[5]
- PyPI: `pip install salt`

[4] https://github.com/saltstack/salt-bootstrap
[5] https://repo.saltproject.io/

Конфиг

- `/etc/salt/master`
- `/etc/salt/minion`

Можно два в одном (но не нужно)

В помощь

```
salt-run fileserver.dir_list [saltenv=ОКРУЖЕНИЕ]
salt-run fileserver.file_list [saltenv=ОКРУЖЕНИЕ]
```

Пара слов о файлах

- Конфиг перечитывается в рантайме
- Данные времени исполнения в `/var/cache/salt/`

Запуск

- `salt-master` (команда и служба)
- `salt-minion` (команда и служба)
- Ключ `--log-level=debug`

Команды

- `salt-key`
- `salt ЦЕЛЬ МОДУЛЬ.ФУНКЦИЯ АРГУМЕНТЫ`
- `salt-cp ЦЕЛЬ ИСТОЧНИК1 [ИСТОЧНИК2 …] НАЗНАЧЕНИЕ`
- `salt-call МОДУЛЬ.ФУНКЦИЯ АРГУМЕНТЫ`
- `salt-run МОДУЛЬ.ФУНКЦИЯ АРГУМЕНТЫ`

Нацеливание

- По имени[6]
  ```
  salt '*-blue' test.ping
  ```
- По грейнам
  ```
  salt -G 'os:CentOS Stream' test.ping
  ```
- По группам
  ```
  salt -N blue test.ping
  ```
  …
- Комбинируя условия почти как угодно[7]
  ```
  salt -C '( G@os:CentOS* and *blue ) or S@192.168.122.112' test.ping
  ```

[6] Помним, что * без кавычек раскрывается шеллом

[7] Разбивка по пробелам, пробелы напрямую передать не получится (#21260)

Шина событий

- Это как общий чат
- Авторизованные клиенты подписываются
- Миньоны берут задачи, когда они есть в адресатах
- Миньоны отвечают обратно в шину
- Шина на ZeroMQ производительна

```
salt-run state.event [ПАТТЕРН_СОБЫТИЯ] [pretty=True]
```

Модули исполнения

- test, saltutil
- file, pkg, service, system, cmd…
- apache, postgres, nginx, redis…
- ansiblegate, chef, puppet

Пара полезных функций

```
>_ test.ping
```

```
>_ saltutil.sync_all
```

```
>_ sys.doc
```

```
>_ state.test[8]
```

[8] добавлена в 3001 как алиас для `state.apply test=True`

Формулы

Основные свойства

- Местные cookbook’и
- Идемпотентность (повторяемость)
- Топ-файл и хайстейт

Формулы

Синтаксис

```
my_state_id1:
  states_module1.function1
  states_module2.function2:
    - name: overrided_id
    - arg1: value
    - arg2:
        - value1
        - value2
        - value3

my_state_id2:
  states_module1.function1
```

- Функции состояний принимают аргумент name
- name по умолчанию равен идентификатору
- В формуле может быть множество состояний
- В состоянии можно обращаться к нескольким разным модулям
- В состоянии нельзя обращаться к одному модулю несколько раз

Формулы

Синтаксис

Часто можно сделать одно разными путями

Хорошо выбрать один способ и придерживаться его

Модули состояния

- `test, saltutil`
- `file, pkg, service, system, cmd…`
- `apache, postgres_*, redismod…`
- `ansiblegate, chef`

Всего на сегодня 349 штук согласно документации

Порядок разбора

0. Разбор на миньоне
1. Рендеринг (`jinja | yaml`)
2. Определение порядка исполнения
3. Исполнение

При ошибке на любом этапе — ответ в шину

Порядок исполнения

Варианты

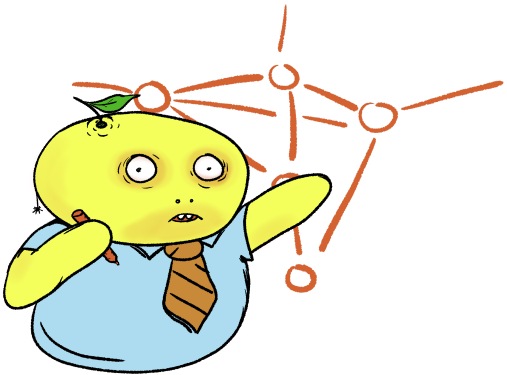
Три пути:

1. В лексикографическом порядке
2. По флагу `order`
3. По реквизитам

Документация призывает выбрать один способ, и следовать ему

Порядок исполнения

Предупреждение



Реквизиты позволяют элегантно запутать ваш код

Каверзы YAML

- ☹ Стандарт регламентирует использование пробелов
- ✓ Не используйте табы

- ☹ Вложенный словарь проигнорирован
- ✓ Увеличивать отступ

```
tree:
  - branch1:
    leaf1: yellow
    leaf2: green
  - branch2:
    leaf1: yellow
```

```
tree:
  - branch:
      leaf1: yellow
      leaf2: green
  - branch:
      leaf1: yellow
```

Каверзы YAML

☹ `Yes, yes, No, no` и т. д. грузятся как буль

☑ Оборачивать кавычками

☹ По стандарту только ASCII

☑ Стараться использовать ASCII, не вставлять emoji ☺

Каверзы YAML

- ☹ Строки с датой приводятся к `datetime`, а с временем — к целым
- ✓ Оборачивать кавычками

- ☹ Символ `%` имеет специальное значение, символ `_` игнорируется в целочисленных литералах
- ✓ Кавычки

Полезные ссылки

- Генерируемая документация (можно скачать)
- Когда что-то не работает, наперёд шерстим тикеты на GitHub

Jinja

Шаблонизация формул

Минутка истории

Дзиндзя — синтоистский храм [9]

- Синтаксис шаблонизатора Django, больше возможностей
- На Python и для Python
- Поддерживает расширения

[9] Потому что temple созвучно с template

Причины использовать шаблонизатор

- Уменьшается дублирование кода
- Доступны grains, pillars, любые внешние данные
- Гибкость: условное включение кода

Причины НЕ использовать шаблонизатор

СЛОЖНО

Очень легко написать запутанный неподдерживаемый код[10]

[10] Как и в случае с зависимостями

Синтаксис

Окружения

- {{ }} — выводимая инструкция
- {% %} — блок [Jinja-кода]
- {# #} — комментарий

```
<ul>
{% for s in ['Мир', 'Труд', 'Май',] %}
  {# Этот текст будет удалён шаблонизатором #}
  <li>{{ s }}</li>
{% endfor %}
</ul>
```

Синтаксис

Подрезка пробелов

- ✂ Минусы убирают пробельные символы при подстановке
- ⬅ `{%- КОД_JINJA %}` — подрезаются символы слева
- ➡ `{% КОД_JINJA -%}` — подрезаются символы справа
- ¶ `'\n'` — это тоже пробельный символ
- 🚫 Плюсы отменяют подрезку, указанную в конфиге шаблонизатора

Синтаксис

Инструкции

- Переменные

```
{% set my_list = [1, 2, 3, 4] %}
```

- Цикл for

```
{% for i in range(10) %}
{{ i }}
{% endfor %}
```

- Условный оператор

```
{% if True %} Правда взаправду правда {% else %} Неправда {% endif %}
```

- Фильтры и конвееры

```
{{ ['соль', 'мука', 'шаблонизатор'] | join(', ') | capitalize }}
```

Синтаксис

Макросы

А ещё есть макросы — это как процедуры,
но мы их пропустим.

Рендеринг произвольного файла

```
1  import jinja2
2
3  filename = 'template.txt'
4
5  with open(filename, 'r') as f:
6      template = jinja2.Template(f.read())
7
8  print(template.render())
```

Jinja и SaltStack

- Jinja подключается как text renderer
- Jinja используется по умолчанию
- Доступны специфичные для SaltStack словари (`grains`, `pillars`) и специальный объект `salt`

Jinja в формулах

Pillar

```
{%
  set settings = pillar.get(
    'power_settings',
    ['standby', 'hibernate', 'monitor', 'disk'])
%}

power_management_timeouts:
  cmd.run:
    - names:
      {% for setting in settings %}
      {% for power_mode in pillar.get('power_modes', ['ac', 'dc']) %}
      - powercfg /change /{{ setting }}-timeout-{{ power_mode }} 0
      {% endfor %}
      {% endfor %}
```

`salt-call saltutil.refresh_pillar` чтобы синхронизировать пиллары.

Полезные ссылки

- Документация Jinja
- Раздел документации SaltStack про Jinja

SaltStack

Ещё кое-что

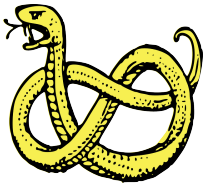
Пишем модули

Особенности

Все мы в душе́ немного программисты

- Python
- Единые соглашения для разных модулей
- Различающийся контекст
- Работает через интроспекцию
- Аннотирование типов ломает интроспекцию

Не забываем `salt '*' saltutil.sync_all`



Пишем модули

Соглашения

- Ищутся любые подходящие функции
- Функции возвращают словари
- `__virtual__: function`
- `__init__: function`, получающая настройки миньона
- `__salt__: dict`[12]
- `__virtualname__: str`
- `__func_alias__: dict`

[12] На деле `NamedLoaderContext`

Реакторы

- Действие по событию (как callback)
- Прописываются последовательностью в конфиге мастера
- Сопоставляют действие тэгу события
- Бывают local, runner, wheel и caller
- Переменные Jinja доступны ограниченно, реквизитов нет

```
>_ salt-run event.send ТЭГ СЛОВАРЬ для отладки
```

Шедулинг

- Похоже на cron или systemd.timer
- Прописывается в конфиге миньона или мастера
- Запуск модулей, применение состояний
- На миньоне модули исполнения, на мастере модули раннеров

```
schedule:
  write_to_file:
    function: cmd.run
    seconds: 5
    args:
      - 'date >> /tmp/salt.log'
    kwargs: {}
    splay: 3
```

В двух словах о Salt Mine

- Шахты похожи на грейны
- Значения доступны для миньонов
- Можно настроить в конфиге миньона
- Можно настроить в пилларах
- Для получения значений `mine.get`
- Пример в следующей секции

Salt Cloud

- Провиженинг виртуальных машин
- Куча коннекторов[13]
- Определяем провайдера и профиль машины
- Элегантно разворачиваемся
- mapfile для подхода «инфраструктура как код»

[13] Но не для отечественных ☁☁😐

Salt Cloud

Команда

```
salt-cloud [-P] -p ПРОФИЛЬ ИМЯ_1 ИМЯ_2 … ИМЯ_N
```
Создать машины из профиля (-P — параллельно)

```
salt-cloud [-P] -m МАПФАЙЛ
```
Создать машины по карте

```
salt-cloud -m МАПФАЙЛ -d
```
Удалить машины, указанные в карте

Ссылки на документацию

- Разработка модулей
- Реакторы
- Планируемые задачи
- Оркестрация
- Salt Cloud