自动化运维工具SALTSTACK

itnihao 2013-06-19

为什么要自动化运维（一）

- ◆ 规模变化所引起的运维手段不同
- ◆ 在互联网时代以运维的角度来看，对于不同规模、场景的应用，运维所处的位置、发展的历程也有一定差异。
- ■ 百台以下规模的运维
- 一般来说，小公司有几台到数十台机器的规模，运维主要关注的点还是在系统及应用服务的稳定性上，不一定有专门的运维人员，通常是RD自行维护所开发的系统。不会去系统化地梳理各项运维指标，而是以人的经验为主，来判断和设定系统正常与否的标准。
- 在这种场景下，运维工作通常扮演的是“救火队员”角色，一旦出故障，运维人员才开始跟进、解决。故障解决也基本是Case by Case的形式，不太会有流程化的总结和问题库、知识库。其运维方式无非是靠手工，辅以一些自行编写的小脚本。这种方式虽较为原始，但对许多初创公司来 说，也算是一种低成本下可灵活实施的手法了。
- 在这个阶段，一些小巧的工具显得很有特色，例如在系统管理方面，能够同时对多台机器进行操作的 Pssh、OmniTTY等，在数据库管理方面，phpMyAdmin等足以应付平时的工作。

为什么要自动化运维（二）

■ 千台规模的运维

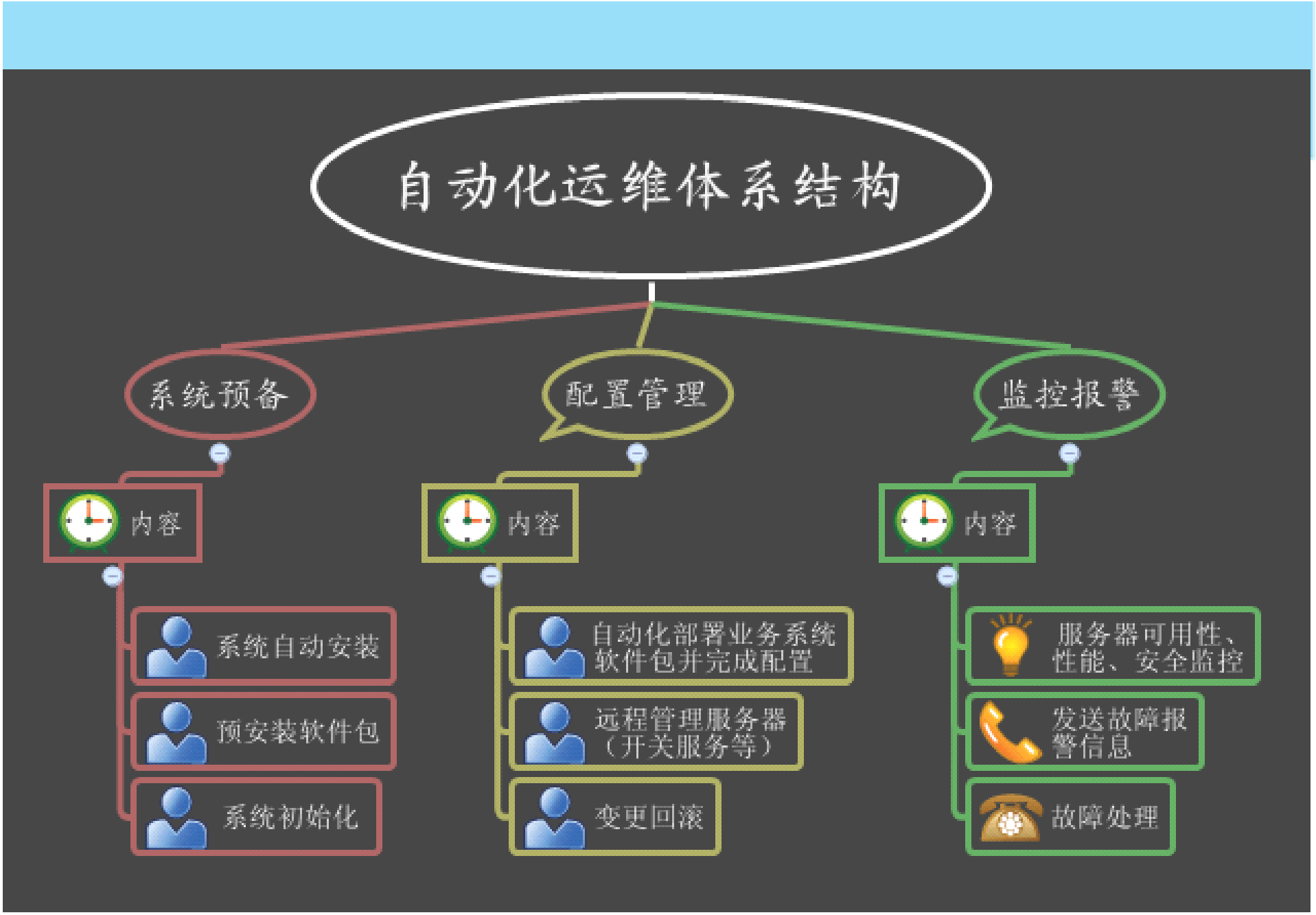
- 当规模增长到一定程度，依赖手工管理自然已无力应对。许多互联网公司的服务器早已跨入几百甚至千台规模，脚本化、批量化管理占据非常大的比例。
- 同时，在这种规模下，按垂直划分的运维工具也开始大量应用，无论是自行开发的还是利用现有开源软件，针对某一特定领域的管理系统显得尤为重要。
- 在这个阶段，运维主要精力放在监控（采集、报警、展现图表）、部署上线（配置管理）、数据备份方面，因为机器数量庞大，所以集中式的操作平台是必备的，针对不同的领域，也有相当成熟的开源软件可以直接使用。

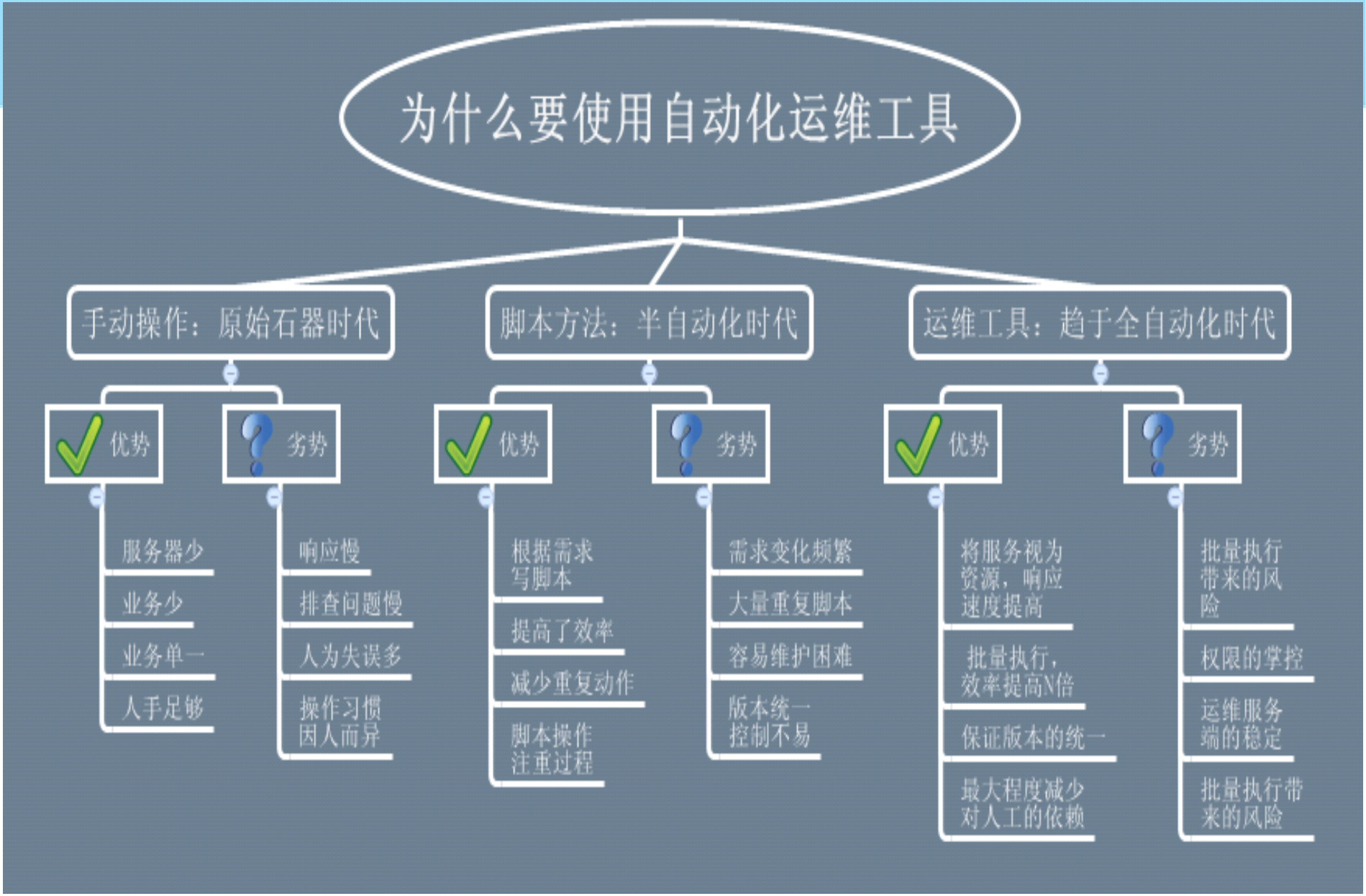
自动化运维的意义

- ➢有统一的自动化运维体系，运维与开发会是平行视角。更关心产品在架构层面的优化以及超大规模集群下的自动化管理和切换。

- ➢ 能利用自动化平台完成各种产品线的监控、部署、关联关联管理。运维开始在整体架构层面为研发提供前置服务。

- ➢ 通常在大几千台到上万台机器的规模公司运维视角：以服务为粒度


自动化运维体系结构
系统预备
内容
系统自动安装
预安装软件包
系统初始化
配置管理
内容
自动化部署业务系统软件包并完成配置
远程管理服务器（开关服务等）
变更回滚
监控报警
内容
服务器可用性、性能、安全监控
发送故障报警信息
故障处理


为什么要使用自动化运维工具
手动操作：原始石器时代
优势
服务器少
业务少
业务单一
人手足够
劣势
响应慢
排查问题慢
人为失误多
操作习惯因人而异
脚本方法：半自动化时代
优势
根据需求写脚本
提高了效率
减少重复动作
脚本操作注重过程
劣势
需求变化频繁
大量重复脚本
容易维护困难
版本统一控制不易
运维工具：趋于全自动化时代
优势
将服务视为资源，响应速度提高
批量执行，效率提高N倍
保证版本的统一
最大程度减少对人工的依赖
劣势
批量执行带来的风险
权限的掌控
运维服务端的稳定
批量执行带来的风险

自动化运维工具介绍

预备类工具	配置类工具	监控报警工具
Kickstart	Chef	Cacti
Cobbler	Puppet	Nagios
Rpmbuild	Func	zabbix
	Saltstack	

自动化运维的基础

- **统一是一切的基础**

a）操作系统的统一，预安装软件
b）软件安装的统一，位置，目录
c）服务器硬件的统一
d）业务类型的统一

- 批量执行，配置的基础是基础设施的统一，才能确保批量执行的成功

- 下面开始了解自动化运维工具Saltstack

什么是saltstack

- SaltStack开源项目始于2011年，是一个相对较新的项目，但在系统管理员和DevOps工程师中拥有越来越多的粉丝
- 项目地址https://github.com/saltstack/salt
- 官网地址http://www.saltstack.com/
- 开发语言：python
- 运行模式：C/S
- 特点：简单易用，模块编写方便，社区活跃，解决问题速度快

saltstack的特点

- Salt是开源的管理基础设置的轻量级工具，容易搭建，并且快速管理成千上万的服务器（保持秒级响应时间）
- 可以执行任意命令，或者预定义的模块（复杂）命令。针对单独独立服务器，或者同名一组服务器，或者相同角色，系统硬件信息，操作系统，当前版本等等
- 通过写简单的命令列表和属性，就可以将你的服务器配置为已知状态，而不需要学习其他的语言(只需要掌握python语言即可)
- 1，轻量级配置管理系统，能够维持远端节点运行在预定状态（例如，确保指定的软件包已经安装和特定的系统服务正在运行）
- 2，分布式远程执行系统，用于在远端节点执行命令和查询数据，可以是单独，也可以是选定的条件

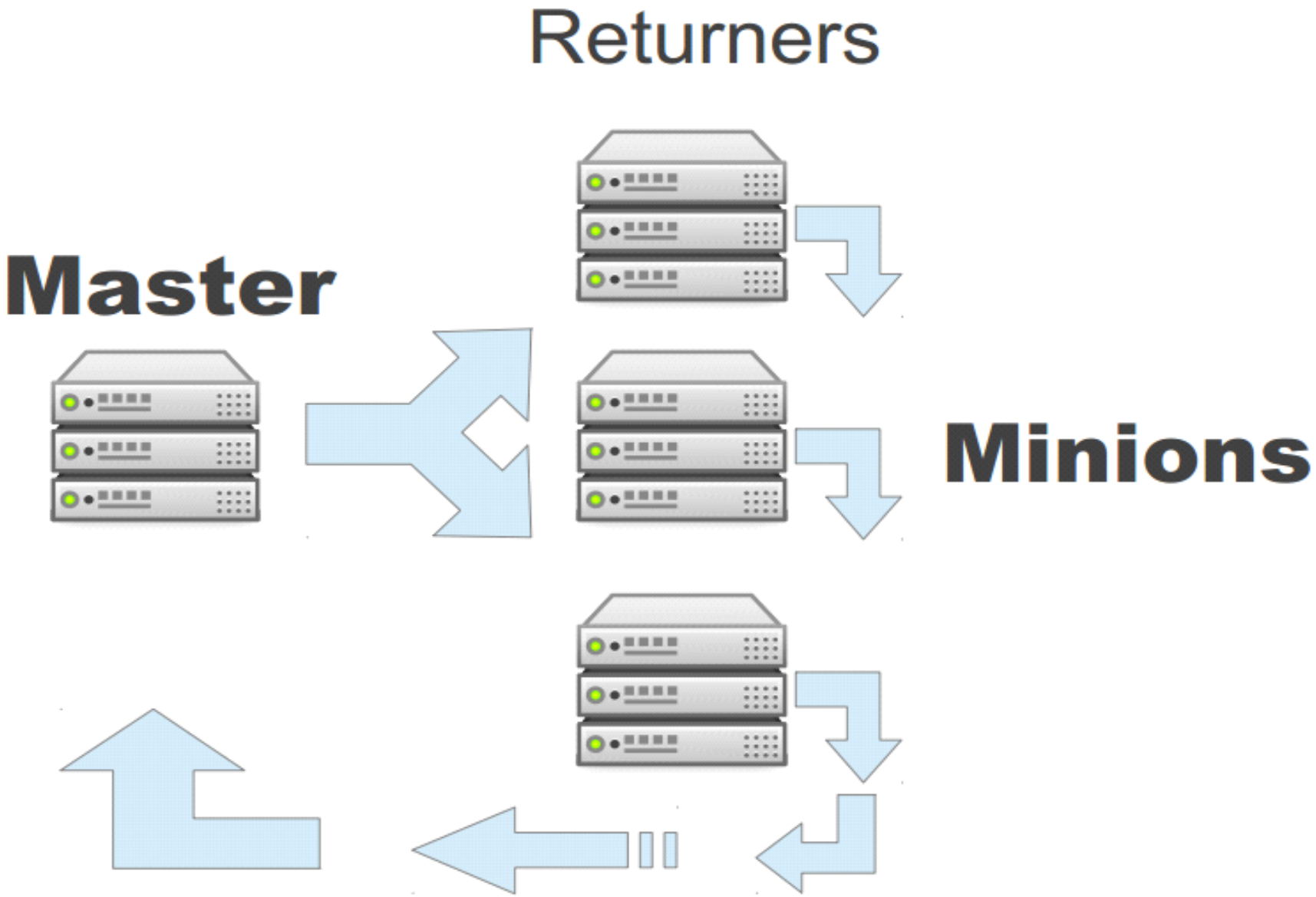
saltstack特性

- 简单:
- 兼顾大规模部署与更小的系统的同时提供多功能性是很困难的，Salt是非常简单配置和维护，不管项目的大小。
- Salt可以胜任管理任意的数量的服务器，不管是本地网络，还是跨数据中心。架构采用C/S模式，在一个后台程序中集成必要功能。默认不需要复杂的配置就可以工作，同时可以定制用于特殊的需求。
-
- 并行执行:
- Salt的核心功能
- 通过并行方式让远端节点执行命令
- 采用安全的加密/解析协议
- 最小化使用网络和负载
- 提供简单的程序接口
- Salt引入了更细粒度的控制，允许不通过目标名字，二是通过系统属性分类

saltstack特性

- 构建在成熟技术之上
- Salt采用了很多技术和技巧。网络层采用优秀的ZeroMQ库，所以守护进程里面包含AMQ代理。Salt采用公钥和主控通讯，同时使用更快的AES加密通信，验证和加密都已经集成在Salt里面。Salt使用msgpack通讯，所以更快速和更轻量网络交换。
-
- Python 客户端接口
- 为了实现简单的扩展，Salt执行例程可以写成简单的Python模块。客户端程序收集的数据可以发送回主控端，可以是其他任意程序。可以通过Python API调用Salt程序，或者命令行，因此，Salt可以用来执行一次性命令，或者大型应用程序中的一部分模块。
-
- 快速，灵活，可扩展
- 结果是一个系统可以高速在一台或者一组服务器执行命令。Salt速度很快，配置简单，扩展性好，提供了一个远程执行架构，可以管理多样化需求的任何数量的服务器。整合了世界上最好的远程执行方法，增强处理能力，扩展使用范围，使得可以适用任何多样化复杂的网络。

Saltstack的架构

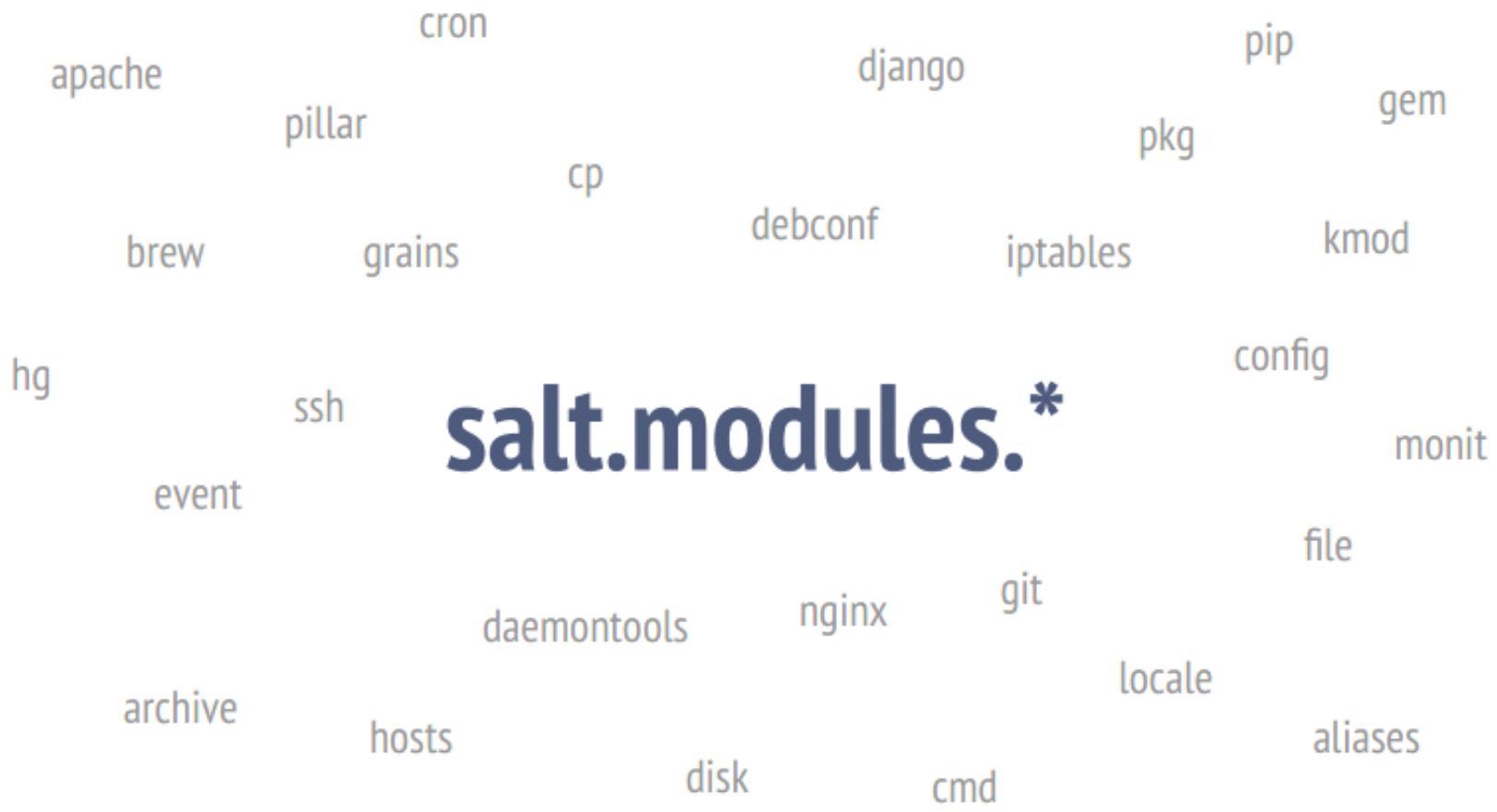


基本术语

- **Master** – 控制中心，salt命令运行和资源状态管理端
- **Minions** – 需要管理的客户端机器，会主动去连接Master端，并从Master端得到资源状态信息，同步资源管理信息
- **States** – 配置管理的指令集
- **Modules**– 包含命令行下运行的指令，和在配置文件里面使用的指令
  – 模块可以的函数可以在命令行下运行
- **Grains** – **minion端的变量，静态**
- pillar – minion端的变量，动态
- highstate – 给minion永久下添加状态，从sls配置文件读取到的
- salt schedule – 自动保持客户端配置

丰富的模块

https://github.com/saltstack/salt/tree/develop/salt/modules



安装配置

- 安装方法，yum，esay_install，source等
- rpm -ivh http://mirrors.sohu.com/fedora-epel/6/x86_64/epel-release-6-8.noarch.rpm
- master端的安装
- yum install salt-master
- 启动服务
- /etc/init.d/salt-master start
- 监听端口
- 4505(publish_port):salt的消息发布系统
- 4506(ret_port):salt客户端与服务端通信的端口
- 要保证这2个端口能通信正常，如果开启iptables 需要添加以下2条规则
- -A INPUT -m state --state new -m tcp -p tcp --dport 4505 -j ACCEPT
- -A INPUT -m state --state new -m tcp -p tcp --dport 4506 -j ACCEPT
- minion端的安装
- #yum install salt-minion
- #vim /etc/salt/minion
- master: 服务端主机名
- id: 客户端主机名(在服务端看到的客户端的名字)
- 启动服务
- #/etc/init.d/salt-minion start

salt-key

- 采用证书认证通信
- #salt-key
- Accepted Keys:
- Unaccepted Keys:
- test.itnihao.com
- Rejected Keys:

- #salt-key -a test.itnihao.com
- Accepted Keys:
- test.itnihao.com
- Unaccepted Keys:
- Rejected Keys:

Targeting Minions

Several ways to match

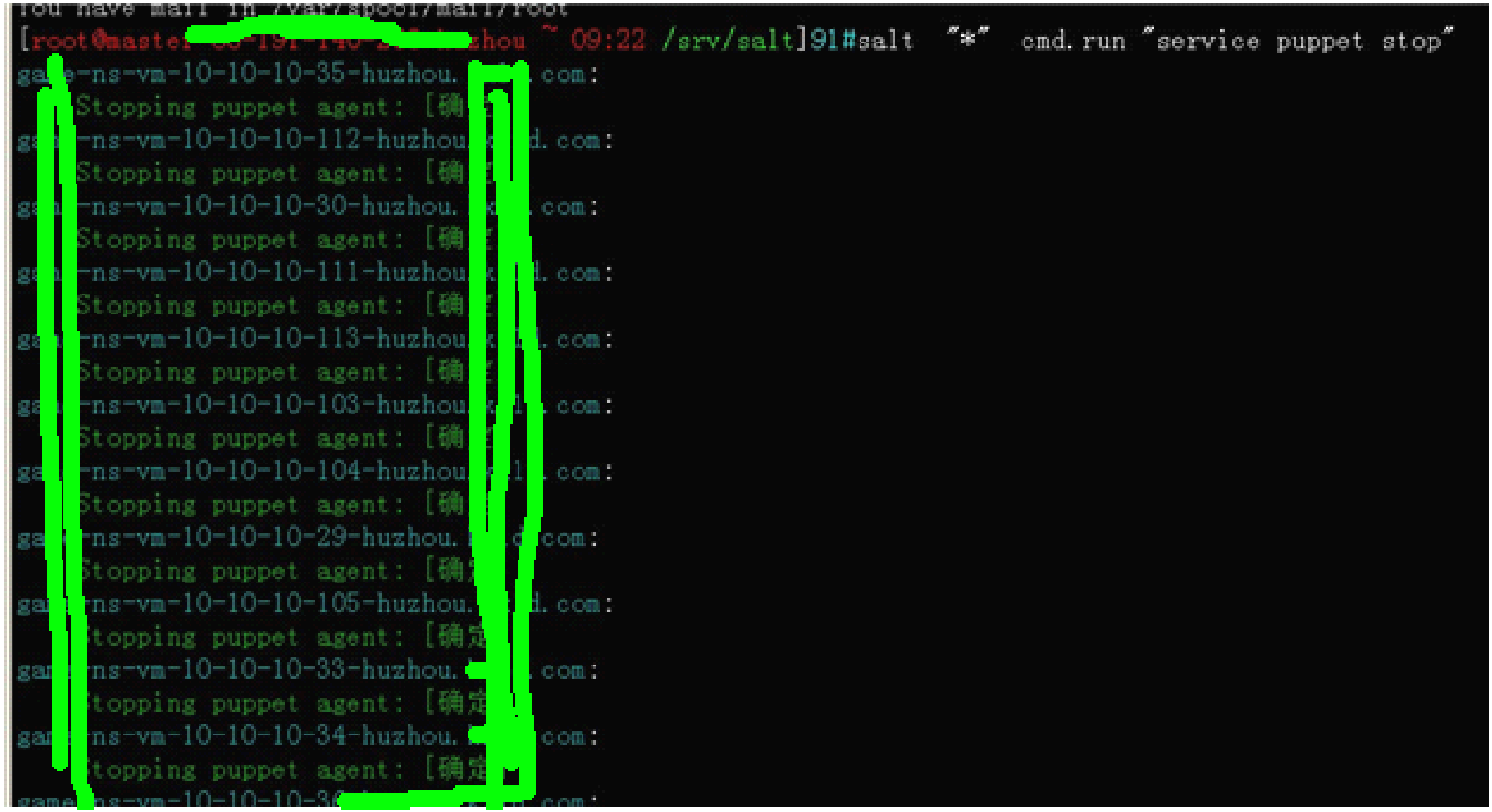
- **Glob (default):** 'web*.domain.com'
- **PCRE:** 'web0[1-].(chi|ny).domain.com'
- **List:** 'foo.domain.com,bar.domain.com'
- **Grains:** 'os:CentOS', 'os:Arch*'
- **Grain PCRE:** 'os:(Linux|.+BSD)'
- **Nodegroup:** (defined in master config file)
- **Pillar:** 'proxy_ip:10.1.2.3'

Grains

- grains　第一次启动minion收集的系统信息常量，这意味着此类信息是静态的。
- grains接口可以为salt模块和组建提供一些信息，例如，使用适当的salt minion命令可以自动汇报一些系统信息
- 查看所有的grains，可以用**grains.items命令**
- `salt \* grains.items`
- 查看单个grain，使用**grains.item命令**
- `salt \* grains.item os`

批量管理服务

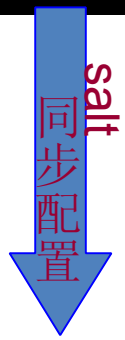
服务的管理:start stop restart



用户管理

salt配置文件如下

```
root:
    group.add:
        - gid: 0
        - system: True
root:
    user.present:
        - fullname: root
        - password: '$6$xl9EqPKOhQEeyG8a$IUcEU4UAMxWDx.n/gAr3VaW1rGHA
        - shell: /bin/bash
        - home: /root
        - uid: 0
```




```
----------
    State: - user
    Name:      root
    Function:  present
        Result:    True
        Comment:   Updated user root
        Changes:   pwd: $6$xl9EqPKOhQEeyG8a$IUcEU4UAMxWDx.n/gAr3VaW1rGHA
```

几百台服务器如何一分钟更改密码？
用salt轻松搞定

uid
gid
passwd
home
shell
……

文件的管理

- /srv/salt/myapp.sls

```
/etc/myapp.conf:
    file.managed:
        - source: salt://files/myapp.conf
        - mode: 644
        - owner: root
        - group: root
        - template: jinja
```

- 入口文件/srv/salt/top.sls

```
base:
    'test.kx1d.com':
    - myapp
```

shell#salt "test.kx1d.com" state.highstate -v

然后在客户端即可发现/etc/myapp.conf文件

对机器分组的管理

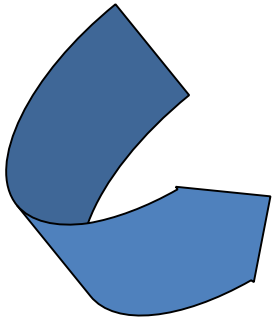

nodegroups:
-ns: '-ns-vm-10-10-10-*-
-xmj: '-xmj-vm-10-10-10-*-
/etc/salt/master [FORMAT=unix] [TYPE=CONF] [POS=416,20][89%
]35#salt -N '-xmj' test.ping
-xmj-vm-10-10-10-29-
True
-xmj-vm-10-10-10-28-
True
对分组执行命令
]36#salt -N '-ns' test.ping
-ns-vm-10-10-10-31-
True
-ns-vm-10-10-10-113-
True
-ns-vm-10-10-10-164-
True
-ns-vm-10-10-10-36-
True
-ns-vm-10-10-10-124-
True
-ns-vm-10-10-10-126-
True
-ns-vm-10-10-10-176-
True

软件版本的控制

```
zabbix_agentd:
   service:
      - running
      - watch:
         - file: /etc/zabbix/zabbix_agentd.conf
         - pkg: zabbix-agentd
   require:
      - pkg: zabbix-agentd

zabbix-agentd:
   pkg.installed:
      - version: '2.0.5-2.el6.zbx'

/etc/zabbix/zabbix_agentd.conf:
   file.managed:
      - source: salt://server_game/game_ns/base_system_setting/zabbix/conf/zabbix_agentd.conf
      - mode: 644
      - user: zabbix
      - group: zabbix
      - template: jinja
```



```
----------
    State: - pkg
    Name:      zabbix-agentd
    Function:  installed
        Result:    True
        Comment:   Version 2.0.5-2.el6.zbx of package "zabbix-agentd" is already inst
        Changes:
----------
```

运用salt模块进行hosts文件的管理

```
[root@[redacted]-ns-vm-10-10-10-30-huzhou ~]# cat /etc/hosts
127.0.0.1       localhost.localdomain   localhost.localdomain   localhost4
::1             localhost.localdomain   localhost.localdomain   localhost6
10.10.10.1      nat.[redacted].com
10.10.10.1      mirrors.[redacted].cn
10.10.10.1      master-[redacted]-huzhou.[redacted]
[root@[redacted]-ns-vm-10-10-10-30-huzhou ~]# cat /etc/hosts
127.0.0.1       localhost.localdomain   localhost.localdomain   localhost4
::1             localhost.localdomain   localhost.localdomain   localhost6
10.10.10.1              mirrors.[redacted]
10.10.10.1              master-[redacted]-huzhou.[redacted]
[root@[redacted]-ns-vm-10-10-10-30-huzhou ~]#
```

执行过程如下

```
          Comment:    Host mirrors.orshsoft.com.cn already pr
          Changes:

----------
    State: - host
    Name:       nat.[redacted].com
    Function:   absent
        Result:     True
        Comment:    Removed host nat.[redacted].com
        Changes:    host: nat.[redacted].com

----------
```

模块的开发

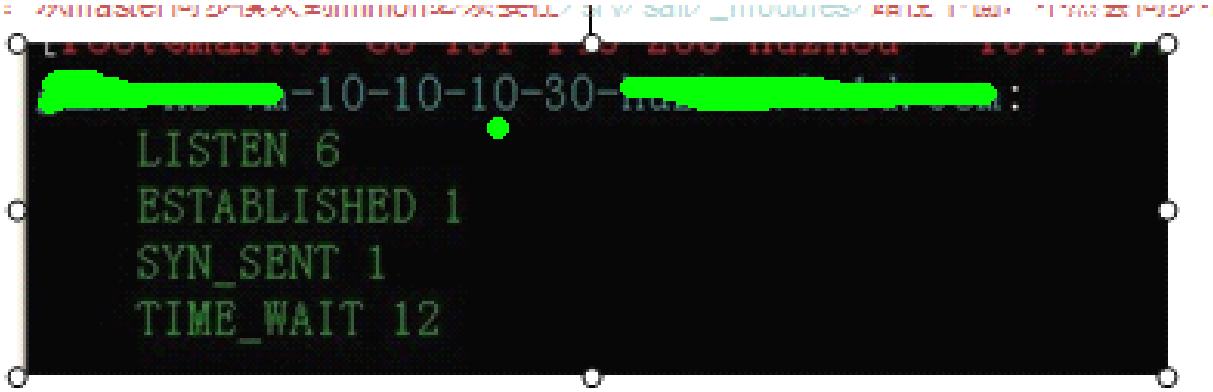
步骤

- mkdir /srv/salt/_modules && cd /srv/salt/_modules
- /srv/salt/_modules/get_tcp_connect.py

```
#!/usr/bin/env python

import subprocess
def tcpconnect():
    '''
    get tcp connect status
    CLI Example:
    salt '*' get_tcp_connect.tcpconnect
    '''
    tcp_status=subprocess.Popen("netstat -an|awk '/^tcp/ {++S[$NF]} END {for (a in S) print a,S[a]}'",stdout=subprocess.PIPE,shell=True)
    return tcp_status.stdout.read()
```

- shell#salt "vm-10-10-10-30" saltutil.sync_all
- shell#salt "vm-10-10-10-30" get_tcp_connect.tcpconnect
- 模块的开发很容易，并且可以单独把不同模块推送到不同服务器
- 特别提示：从master同步模块到minion必须要在/srv/salt/_modules/路径下面，不然会同步不成功



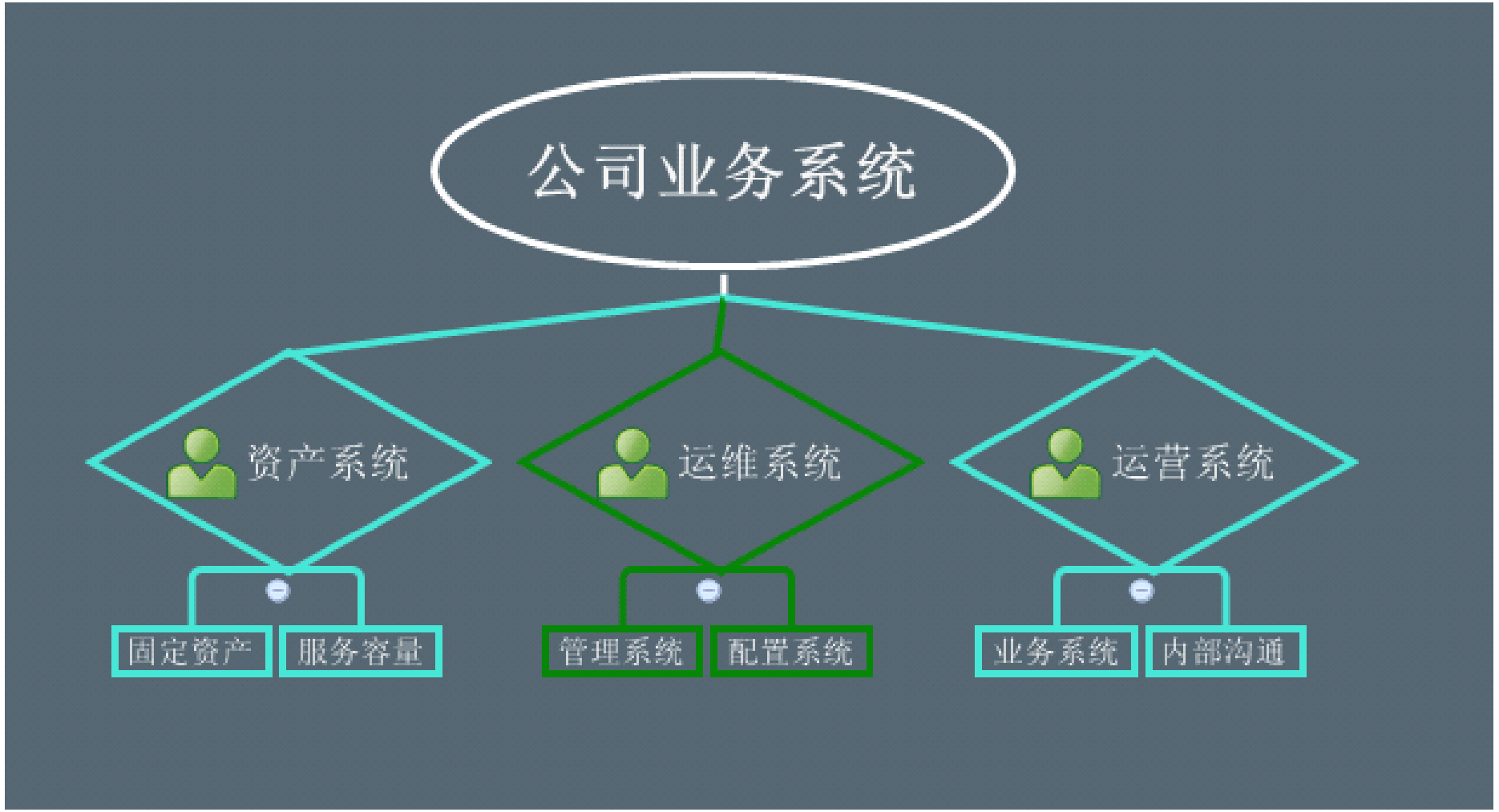
salt运维工具需要改进的地方

- 无web管理界面，官方web目前无太大动作
- 几乎是命令行操作，需要熟练掌握操作
- salt提供api接口，可以基于此进行开发
- 业务模块需要自己开发(这也是Saltstack)

公司未来运维的趋势

- 1.使用salt实现资源统一管理，实现配置。
- 2.salt整合zabbix，有效的结合监控系统
- 3.开发salt的web页面，在此基础上构建一套自动化运维平台。实现对游戏开服关服的web操作
- 4.利用salt api+zabbix api构建资产管理系统
- 5.运维平台实现的功能：
- A文件备份的统一管理
- B集中化的账号管理系统
- 6.运维核心偏重于数据库优化，架构优化，服务器性能优化

自动化运维与其他系统的整合



自动化运维未来潜在的问题

- 1.zabbix监控数据库的调优，横向扩展，可以采用增加zabbix服务器，数据库分表等方式进行
- 2.自动化运维系统的持续更新问题，需在构建之前考虑到业务变更带来的影响
- 3.人员变更对运维系统的影响