

夏虫而已

多见者博，多闻者智，拒谏者塞，专己者孤

< 2019年2月 >						
日	一	二	三	四	五	六
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	1	2
3	4	5	6	7	8	9

昵称：夏虫而已

园龄：2年10个月

粉丝：48

关注：1

+加关注

搜索

找找看

谷歌搜索

常用链接

[我的随笔](#)

[我的评论](#)

[我的参与](#)

[最新评论](#)

[我的标签](#)

我的标签

[java](#)(18)

[docker](#)(14)

[oracle](#)(9)

[mysql](#)(8)

[hadoop](#)(8)

[ignite](#)(7)

[linux](#)(7)

[ubuntu](#)(7)

[maven](#)(5)

[spring](#)(5)

[更多](#)

随笔分类

[docker](#)(16)

[docker-compose](#)(1)

[java IDE](#)(5)

[java 测试](#)(10)

[博客园](#) [首页](#) [新随笔](#) [新文章](#) [联系](#) [订阅](#) [XML](#) [管理](#)

posts - 200,comments - 26,trackbacks - 0

Ansible 进阶技巧

原文 http://www.ibm.com/developerworks/cn/linux/1608_lih_ansible/index.html?ca=drs-

简介

Ansible 是一个系统自动化工具，可以用来做系统配管理，批量对远程主机执行操作指令。我自己使用 Ansible 也有一段时间了，这里总结了一些使用 Ansible 过程中使用的心得与大家分享。

Ansible 性能优化

在使用 Ansible 的过程中，当管理的服务器数量增加时，不得不面对一个无法避免的问题执行效率慢，这里列出一些解决办法。

优化前的准备—收集数据

在做性能优化之前首先需要做的是收集一些统计数据，这样才能为后面做的性能优化提供数据支持，对比优化前后的结果。非常不错的是，在 github 发现一个 Ansible 任务计时插件“ansible-profile”，安装这个插件后会显示 ansible-playbook 执行每一个任务所花费的时间。Github 地址：<https://github.com/jlafon/ansible-profile>。这个插件安装很简单，只需要简单的三个命令即可完成安装。在你的 playbook 文件的目录下创建一个目录，目录名 callback_plugins 然后将下载的 profile_tasks.py 文件放到该目录下。

```
cd /etc/ansible
mkdir callback_plugins
cd callback_plugins
wget https://raw.githubusercontent.com/jlafon/ansible-profile/master/callback_plugins/profile_tasks.py
```

现在，执行 ansible-playbook 命令就会看到 playbook 中每个 tasks 的用时情况。

图 1.ansible-playbook tasks 用时情况

- [java 高级\(1\)](#)
- [java 基础\(3\)](#)
- [java 开发\(39\)](#)
- [java 设计模式\(7\)](#)
- [linux 系统\(28\)](#)
- [Mac 系统\(1\)](#)
- [paas\(11\)](#)
- [python\(2\)](#)
- [shell 开发\(1\)](#)
- [web 容器\(2\)](#)
- [产品架构\(2\)](#)
- [产品交付\(8\)](#)
- [大数据\(16\)](#)
- [分布式缓存\(8\)](#)
- [分布式开发\(1\)](#)
- [分布式配置\(1\)](#)
- [高并发](#)
- [工作流\(1\)](#)
- [计算机原理\(1\)](#)
- [前端\(11\)](#)
- [深度学习](#)
- [实时计算\(3\)](#)
- [数据库汇总\(23\)](#)
- [微服务\(4\)](#)
- [虚拟机\(2\)](#)
- [运维\(2\)](#)

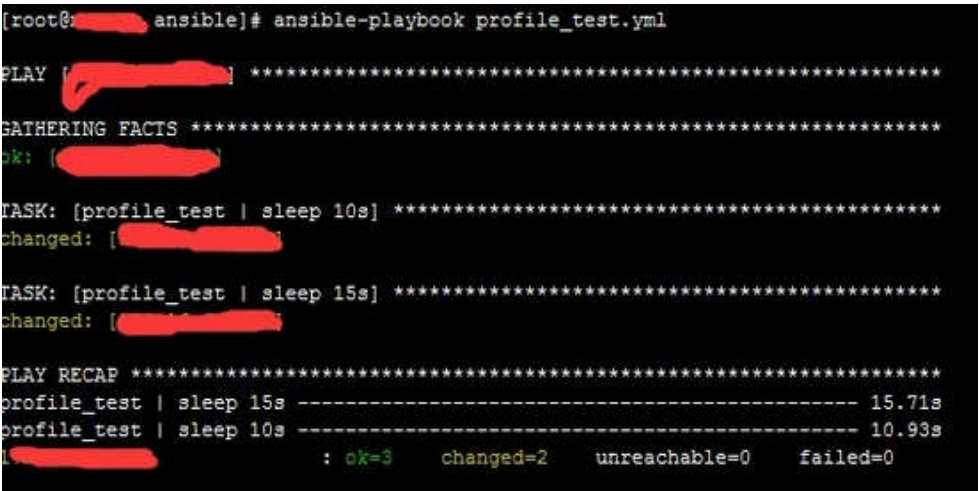
随笔档案

- [2019年1月 \(4\)](#)
- [2018年12月 \(7\)](#)
- [2018年11月 \(3\)](#)
- [2018年9月 \(1\)](#)
- [2018年8月 \(4\)](#)
- [2018年6月 \(3\)](#)
- [2018年5月 \(1\)](#)
- [2018年3月 \(1\)](#)
- [2018年1月 \(11\)](#)
- [2017年12月 \(7\)](#)
- [2017年11月 \(2\)](#)
- [2017年6月 \(11\)](#)
- [2017年4月 \(2\)](#)
- [2017年3月 \(6\)](#)
- [2017年2月 \(14\)](#)
- [2017年1月 \(3\)](#)
- [2016年12月 \(15\)](#)
- [2016年11月 \(1\)](#)
- [2016年10月 \(7\)](#)
- [2016年9月 \(9\)](#)
- [2016年8月 \(4\)](#)
- [2016年7月 \(15\)](#)
- [2016年6月 \(20\)](#)
- [2016年5月 \(44\)](#)
- [2016年4月 \(1\)](#)
- [2016年3月 \(4\)](#)

最新评论

- [1. Re:分布式锁的几种实现方式](#)
用了Redis的分布式锁,感觉锁不住

--oreo



在这里，我设置了 2 个 task，1 个 task sleep 10 秒，另 1 个 task sleep 15 秒，在 PLAY RECAP 处会汇总所有 task 执行消耗的时间。

关闭 gathering facts

如果您观察过 ansible-playbook 的执行过程中，您会发现 ansible-playbook 的第 1 个步骤总是执行 gather facts，不论你有没有在 playbook 设定这个 tasks。如果你不需要获取被控机器的 fact 数据的话，你可以关闭获取 fact 数据功能。关闭之后，可以加快 ansible-playbook 的执行效率，尤其是你管理大量的机器时，这非常明显。关闭获取 facts 很简单，只需要在 playbook 文件中加上“gather_facts: no”即可。如下

```
- hosts: 172.16.64.240
gather_facts: no
remote_user: liheng
sudo: yes
roles:
- {role: profile_test}
```

好的，来看关闭前后的执行时间变化。

图 2. 关闭 gather_facts 前后的执行变化

2.

Re:log_bin_trust_function_creators
为什么我执行了 但是没生效呢 还是
OFF

--super超人

3. Re:docker入门实战笔记

内容非常好!!

--Hunter_猎狼人

4. Re:jetty作为内嵌服务器自启动

@最后Q泪滴怎么会打成jar包,打zip
包啊,直接丢服务器上解压完启动不
就行了?...

--白熊

5. Re:最简单易懂的WebService客户端之soap+xml请求

@btbxin抱歉没看到评论, 函数找不到要验证一下wsdl地址能不能用
呢, 放在浏览器中试试...

--每周一个技术哦

阅读排行榜

1. docker入门实战笔记(51010)
2. java中json数据生成和解析(复杂对象演示)(32926)
3. 线程的五种状态及改变状态的三种方法(29680)
4. docker 运行redis(25648)
5. 最简单易懂的WebService客户端之soap+xml请求(24476)

评论排行榜

1. jetty作为内嵌服务器自启动(5)
2. ignite学习笔记(3)
3. java中json数据生成和解析(复杂对象演示)(3)
4. 分布式配置管理--百度disconf搭建过程和详细使用(3)
5. log4j2配置MDC分线程写日志(2)

推荐排行榜

1. docker入门实战笔记(6)
2. ignite学习笔记(3)
3. 深入理解OSGI: Java模块化之路(3)
4. linux 正则表达式和通配符(2)
5. Ansible 进阶技巧(2)

```

[root@master ansible]# time ansible-playbook profile_test.yml

PLAY [redacted] *****

GATHERING FACTS *****
ok: [redacted]

TASK: [profile_test | sleep 10s] *****
changed: [redacted]

TASK: [profile_test | sleep 15s] *****
changed: [redacted]

PLAY RECAP *****
profile_test | sleep 15s ----- 15.93s
profile_test | sleep 10s ----- 10.77s
: ok=3    changed=2    unreachable=0    failed=0

real    0m27.784s
user    0m1.242s
sys     0m0.223s
[redacted] ← 未关闭 facts 获取

[redacted]# vim profile_test.yml
[redacted]# time ansible-playbook profile_test.yml

PLAY [redacted] *****

TASK: [profile_test | sleep 10s] *****
changed: [redacted]

TASK: [profile_test | sleep 15s] *****
changed: [redacted]

PLAY RECAP *****
profile_test | sleep 15s ----- 15.71s
profile_test | sleep 10s ----- 10.64s
: ok=2    changed=2    unreachable=0    failed=0

real    0m26.619s
user    0m0.872s
sys     0m0.158s
[redacted] ← 关闭facts 获取
[redacted]#
  
```

关闭前后, 执行时间相关 1 秒, 因为我这里只有一台机器, 所以时间差距并不是很明显。不过, 从这个例子也可以看出, 关闭 facts 获取后, 执行速度是快了的。

SSH PIPELINING

SSH pipelining 是一个加速 Ansible 执行速度的简单方法。ssh pipelining 默认是关闭, 之所以默认关闭是为了兼容不同的 sudo 配置, 主要是 requiretty 选项。如果不使用 sudo, 建议开启。打开此选项可以减少 ansible 执行没有传输时 ssh 在被控机器上执行任务的连接数。不过, 如果使用 sudo, 必须关闭 requiretty 选项。修改 /etc/ansible/ansible.cfg 文件可以开启 pipelining

将

```
pipelining=False
```

修改为

```
pipelining=True
```

修改完后, 可以批量对机器执行命令试下, 可以明显感受到速度的提升。

ControlPersist

ControlPersist 特性需要高版本的 SSH 才支持, CentOS 6 默认是不支持的, 如果需要使用, 需要自行升级 openssh。ControlPersist 即持久化 socket, 一次验证, 多次通信。并且只需要修改 ssh 客户端就行, 也就是 Ansible 机器即可。

升级 openssh 的过程这里不做介绍。这里只介绍下 ControlPersist 设置的办法。

```
cat ~/.ssh/config
Host *
    Compression yes
    ServerAliveInterval 60
    ServerAliveCountMax 5
    ControlMaster auto
    ControlPath ~/.ssh/sockets/%r@%h-%p
    ControlPersist 4h
```

在开启了 ControlPersist 特性后, SSH 在建立了 sockets 之后, 节省了每次验证和创建的时间。在网络状况不是特别理想, 尤其是跨互联网的情况下, 所带来的性能提升是非常可观的。有这边需求的, 试试就知道了。

Ansible-playbook 技巧

获取执行命令的输出 --Register

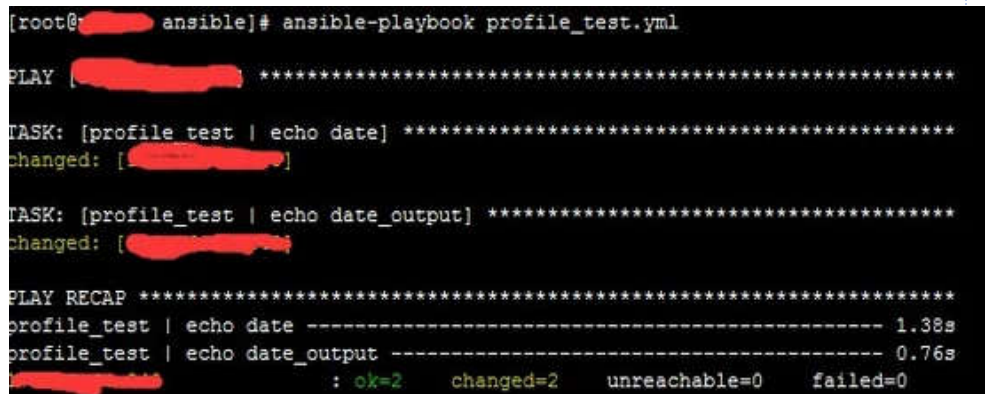
在刚开始使用 ansible-playbook 做应用程序部署的时候, 因为在部署的过程中有使用到 command 或 shell 模块执行一些自定义的脚本, 而且这些脚本都会有输出, 用来表示是否执行正常或失败。如果像之前自己写脚本做应用程序部署的, 这很好实现。但现在是用 Ansible 做, 那么要怎么样做可以获取到 ansible playbook 中 command 模块的输出呢? Ansible 也提供的解决办法, 这时我们就可以通过使用 register 关键字来实现, register 关键字可以存储指定命令的输出结果到一个自定义的变量中, 我们通过访问这个自定义变量就可以获取到命令的输出结果。Register 的使用很方便, 只需要在 task 声明 register 关键字, 并自定义一个变量名就可以。如下:

```
- name: echo date
  command: date
  register: date_output

- name: echo date_output
  command: echo "30"
  when: date_output.stdout.split(' ')[2] == "30"
```

这里第 1 个 task 是执行了一个 date 命令, register 关键字将 date 命令的输出存储到 date_output 变量名。第 2 个 task 对输出进行分析, 并使用 when 对关键字对分析后的进行判断, 如果匹配, 则执行这个 task, 不匹配就不执行。这里要重点说下的, 因为 register 获取到的输出内容都是字符串, 而 ansible 又是 python 写的, 你可以使用 python 字符串的方法对其做处理, 比如本文中使用的 split, 还可以使用 find 方法。个人觉得, 真是非常灵活方便。

图 3.register 执行结果 1



```
[root@redacted ansible]# ansible-playbook profile_test.yml

PLAY [redacted] *****

TASK: [profile_test | echo date] *****
changed: [redacted]

TASK: [profile_test | echo date_output] *****
changed: [redacted]

PLAY RECAP *****
profile_test | echo date ----- 1.38s
profile_test | echo date_output ----- 0.76s
: ok=2    changed=2    unreachable=0    failed=0
```

这里由于条件匹配, 两个 task 都执行了。然后把第 2 个 task 中的条件改动了下, 使其不匹配, 执行结果如下:

图 4.register 执行结果 2


```
[root@redhat ansible]# ansible-playbook profile_test.yml

PLAY [redhat] *****

TASK: [profile_test | echo date] *****
changed: [redhat]

TASK: [profile_test | echo date_output] *****
skipping: [redhat]

PLAY RECAP *****
profile_test | echo date ----- 0.66s
profile_test | echo date_output ----- 0.02s
: ok=1    changed=1    unreachable=0    failed=0
```

这里第 2 个 task 条件不匹配，skipping 了。

Delegate_to(任务委派功能)

场景介绍：在对一组服务器 server_group1 执行操作过程中，需要在另外一台机器 A 上执行一个操作，比如在 A 服务器上添加一条 hosts 记录，这些操作必须要在一个 playbook 联动完成。也就是说 A 服务器这个操作与 server_group1 组上的服务器有依赖关系。Ansible 默认只会在定义好的一组服务器上执行相同的操作，这个特性对于执行批处理是非常有用的。但如果在这过程中需要同时对另外 1 台机器执行操作时，就需要用到 Ansible 的任务委派功能（delegate_to）。使用 delegate_to 关键字可以委派任务到指定的机器上运行。在 playbook 的操作如下：

```
- name: add host record
  shell: 'echo "192.168.1.100 test.xyz.com" >> /etc/hosts'

- name: add host record to center server
  shell: 'echo "192.168.1.100 test.xyz.com " >> /etc/hosts'
  delegate_to: 192.168.1.1
```

任务委派功能还可以用于以下场景：

- 在部署之前将一个主机从一个负载均衡集群中删除。
- 当你要对一个主机做改变之前去掉相应 dns 的记录
- 当在一个存储设备上创建 iscsi 卷的时候
- 当使用外的主机来检测网络出口是否正常的时候

本地操作功能 --local_action

Ansible 默认只会对控制机器执行操作，但如果在这个过程中需要在 Ansible 本机执行操作呢？细心的读者可能已经想到了，可以使用 delegate_to(任务委派) 功能呀。没错，是可以使用任务委派功能实现。不过除了任务委派之外，还可以使用另外一外功能实现，这就是 local_action 关键字。

```
- name: add host record to center server
  local_action: shell 'echo "192.168.1.100 test.xyz.com " >> /etc/hosts'
```

当然您也可以使用 connection:local 方法，如下：

```
- name: add host record to center server
  shell: 'echo "192.168.1.100 test.xyz.com " >> /etc/hosts'
  connection: local
```

这两个操作结果是一样的。

Check 模式

当以 - check 参数来运行 ansible-playbook 时，将不会对远程的系统作出任何修改。相对的，任何带有检测功能的模块只要支持‘检测模式’将会报告它们会做出什么改变而不是直接进行改变。其他不支持检测模式的模块将即不响应也不提出相应的报告（事实上几乎所有主要核心模块都是支持‘检测模式’）。检测模式只是一种模拟。如果你的 playbook 是以先

前命令的执行结果作为条件的话，那它可能作用就不明显了。但是在正式运行前，使用 check 模式做个语法检查也是不错的。

选择性执行task-- Tag (标签)

您可能因为某些原因，会创建一个很大型的 playbook，但是您可能只想运行其中特定部分的配置而无需要运行整个 playbook。那么这时您可能需要用到 tag 功能。示例如下：

```
- name: yum install package
  yum: name={{ item }} state=installed
  with_items:
    - httpd
    - memcached
  tags:
    - packages

- name: configuration modify
  template: src=templates/src.j2 dest=/etc/foo.conf
  tags:
    - configuration
```

如果你只想运行 playbook 中的”configuration”和”packages”，你可以这样做

```
ansible-playbook example.yml - tags "configuration,packages"
```

如果你只想执行 playbook 中某个特定任务之外的所有任务，你可以这样做：

```
ansible-playbook example.yml - skip-tags "configuration"
```

tag 特性是一个不错的功能，但如果真的是要维护一个大型的 playbook，还是建议将 playbook 按功能或应用拆分成多个 playbook，然后再在主 playbook include 其他子 playbook，这样即既利于维护也方便管理

错误处理

Ansible 默认会检查命令和模块的返回状态，并进行相应的错误处理，默认是遇到错误就中断 playbook 的执行，这些默认行为都是可以改变的。

忽略错误

command 和 shell 模块执行的命令如果返回非零状态码则 ansible 判定这 2 个模块执行失败，可以通过 ignore_errors 忽略返回状态码（前提是要确定这 command 与 shell 执行错误不会影响后面 task 的执行）。如下：

```
- name: this will not be counted as a failure
  command: /bin/false
  ignore_errors: yes
```

自定义错误判定条件

命令不依赖返回状态码来判定是否执行失败，而是要查看命令返回内容来决定，比如返回内容中包括 failed 字符串，则判定为失败。示例如下：

```
- name: this command prints FAILED when it fails
  command: /usr/bin/example-command -x -y -z
  register: command_result
  failed_when: "'FAILED' in command_result.stderr"
```

ansible 会自动判断模块执行状态，command、shell 及其它模块如果修改了远程主机状态则被判定为 change 状态，不过也可以自己决定达到 changed 状态的条件，示例如下：

```
- name: copy in nginx conf
  template: src=nginx.conf.j2 dest=/etc/nginx/nginx.conf
```

```
- name: validate nginx conf
  shell: "/data/app/nginx/sbin/nginx -t"
  register: command_result
  changed_when: command_result.stdout.find('successful')
```

命令返回中有“successful”字符串，则为 changed 状态，下面这个设定将永远也不会达到 changed 状态。

```
- name: validate nginx conf
  shell: "/data/app/nginx/sbin/nginx -t"
  changed_when: false
```

结束语 .

本文介绍了一些关于 Ansible 的执行性能优化与 playbook 使用的技巧，这些都是在我们使用 Ansible 过程中需要面对的问题，希望今天列出的这些内容对大家学习和使用 Ansible 能有所帮助。

分类: [paas](#)

标签: [ansible](#)





[夏虫而已](#)
[关注 - 1](#)
[粉丝 - 48](#)
[+加关注](#)

2

0

« 上一篇: [ansible playbook对错误的处理](#)

» 下一篇: [ansible普通用户su切换](#)

posted on 2017-06-13 13:41 [夏虫而已](#) 阅读(8910) 评论(0) [编辑](#) [收藏](#)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

[【推荐】全源码开放:大型组态\工控\监控电力仿真CAD免费下载2019!](#)

[【推荐】专业便捷的企业级代码托管服务 - Gitee 码云](#)

相关博文:

- [Python进阶03 模块](#)
- [Python进阶09 动态类型](#)
- [Java进阶02 异常处理](#)
- [Python进阶01 词典](#)
- [Python进阶08 异常处理](#)

最新新闻:

- [谷歌发布ARCore 1.7: 新API为开发者带来更多特性](#)
 - [OPPO和vivo新机都采用了升降式摄像头, 它会成为另一种主流设计吗?](#)
 - [中国品牌占据了欧洲手机市场三分之一, 华为和小米暴涨](#)
 - [亚马逊大肆扩张物流与航运能力 拟打造物流业巨头](#)
 - [同时抱上腾讯、阿里 B站或成最大赢家?](#)
- » [更多新闻...](#)

历史上的今天:

- 2016-06-13 [oracle多种导入导出数据方法](#)
- 2016-06-13 [java设计模式案例详解:工厂模式](#)
- 2016-06-13 [java设计模式案例详解:观察者模式](#)

Copyright ©2019 夏虫而已 Powered By[博客园](#) 模板提供: [沪江博客](#)