简介

在当前配置管理工具大行其道的应用中，ansible凭借其轻量级、agentless等特性得以占据一席之地。然而其也并不是完美无缺，事实上，其最为人所诟病的就是在大规模服务器应用场景当中表现出的性能问题。所以本篇文档就来说一说如何通过一些配置优化来加速ansible的运行。

SSH Multiplexing

默认情况下，ansible基于ssh连接被控端，当ansible在运行playbook的时候，会建立很多的ssh连接来执行相应的task，而每个task都会创建一个新的ssh连接，ssh连接的建立毫无疑问需要额外tcp建立连接的开销。

openssh支持一个优化，叫做ssh nultiplexing，也被称作ControlPersist。当启用了该特性，则多个连接到相同主机的ssh会话将会共享相同的tcp连接。这样就只有第一次连接的时候需要进行TCP三次握手。

当启用Multiplexing后：

- 第一次ssh连接到主机的时候，openssh会创建一个主连接
- 紧接着openssh会创建一个控制套接字，通过主连接与远程主机关联
- 当有新的ssh尝试连接到主机时，openssh将使用控制套接字与远程主机通信，并不会创建新的tcp连接

开启该配置项：

```
#cat ansible.cfg
[ssh_connection]
ssh_args = -C -o ControlMaster=auto -o ControlPersist=10m
```

配置项说明：

- ControlMaster：启用ssh multiplexing，auto用于告诉ssh在主连接和控制套接字不存在的情况下，自动创建它们
- ControlPersist：在主连接和控制套接字创建后的多长时间内，如果一直没有ssh连接，就关闭它们。

pipelining

在说明pipelining之前，需要先了解下ansible是如何执行一个task的：

1. 基于playbook中的task生成一个python脚本
2. 将生成的python脚本复制到被控主机上
3. 在被控主机上运行这个脚本

上面三个步骤中，后面两个步骤会产生两个ssh会话。而在pipelinin模式下，ansible执行python脚本时并不会复制它，而是通过管道传递给ssh会话。这样一来，就会让原本需要的两个ssh会话减少为一个，以降低开销，节省时间。

启用pipelining：

```
#cat ansible.cfg
[defaults]
pipelining = True
```

需要注意的是，如果开启pipelining，则需要在被控端的/etc/sudoers文件中关闭requiretty：

```
#cat /etc/sudoers.d/ansible
ansible    ALL=(ALL)       NOPASSWD:ALL
Defaults:ansible !requiretty
```

并发执行

默认情况下，ansible在批量连接客户端执行play时的并发数是5个，可以通过调整该值来提高并发数：

```
[defaults]
forks = 20
```

即使提高了并发个数，playbook也可能在执行一个需要较长时间的任务时导致阻塞，针对这类任务，可以使用异步的方式来运行：

```
- hosts: all
  tasks:
    - name: Install mlocate
      yum:
        name: mlocate
        state: installed
    - name: Run updatedb
      command: /usr/bin/updatedb
```

```
        async:300
        poll:10
```

当使用上面的任务控制超过forks设置的节点个数时，'install mlocate'任务会先在 `forks` 个节点上跑，完成后再继续下一批，这个并发数是由我们设置的 `forks` 选项控制的。而'Run updatedb'这个任务会一次性在所有节点上都执行，执行的超时时间为300s，然后每10s轮循一次检查它是否完成。需要说明的是，因为需要等待其执行完成，所以如果这个任务比较耗时，仍然需要等待其执行完毕后才能开始下一个任务。

但是如果该任务只是一个后台任务，比如只是在后台执行一个脚本或者启动一个服务，不需要等待其返回结果。那就不需要检查这个任务是否完成，只需要继续执行其它任务，最后再使用wait_for这个模块去检查之前的脚本或者服务是否按预期中运行了便可。这时候只需要把poll的值设置为0，便可以按上面的要求配置ansible不等待job的完成。

还有一种需求，假如一个task，它就是需要运行很长的时间，不能让它超时退出，需要一直等待这个task完成。这个时候就需要将async的值设置为0。

总结来说，大概有以下的一些场景需要使用到ansible的polling特性：

- 某个task需要运行很长的时间,这个task很可能会达到timeout。
- 某个task需要在大量的机器上面运行
- 某个task是不需要等待它完成的

当然也有一些场景是不适合使用：

- 这个任务是需要运行完后才能继续另外的任务的
- 这个任务能很快的完成

accelerate模式

Ansible官方建议在可以使用pipelining的情况下，尽量使用pipelining而不是accelerate模式。accelerate只在如下情况下有用处：

- 管理centos6或者更早的那些依然使用paramikio作为连接被控节点的场景
- 无法使用pipelining的场景

加速模式通过启动一个临时的 SSH守护进程来工作。只要这个守护进程在运行，Ansible将会直接通过socket来连接。Ansible通过在连接时交换临时的AES key来确保安全(这个秘钥对每个主机都是不同的并且会定期重新生成).。默认配置下，Ansible会为加速模式开启5099端口(此配置可修改)。一旦运行了，守护进程将会维持连接30分钟，过了时限后该连接将会自动终结，你需要重启一个 SSH。

accelerate模式配置：

```
#cat ansible.cfg
[accelerate]
accelerate_port = 5099
```

```
accelerate_timeout = 30
accelerate_connect_timeout = 5.0
```

> 需要说明的是，如果要开启accelerate模式，需要在管理端和被控端都安装python-keyczar软件包