ABSTRACT

Running long duration tasks using Ansible, results in inefficient resource utilization, and connection time-outs. At times, environment specific limitations also put restrictions on running an Ansible job for longer time periods. This paper describes a solution to overcome these issues using an approach to run ansible jobs in an asynchronous way.

INTRODUCTION:

Ansible is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs.

There are few use-cases however, where Ansible after performing some pre-requisite tasks, simply **waits** for a process that executes the main task like:

- Installing a big software, that may run for long
- Taking huge database backups, which may run for hours
- Waiting for any events that might take longer on remote hosts
- Waiting for cloud resources to be in a required state.
- And the list goes on…

In such cases:

- Ansible itself does not perform any useful task and remains in a wait state.
- Also, there is un-necessarily a session maintained between the Ansible tower and the remote host, which consumes the network bandwidth.
- This job utilizes the tower's capacity to run concurrent jobs, which can otherwise be better utilized.

ENVIRONMENT SPECIFIC CONSTRAINTS:

In addition to the above, there can also be limitations on the duration for which the session can remain established between the Ansible Tower and the remote host.

For example, in case of AWS when we make use of 'AssumeRole' feature *(It returns a set of temporary security credentials that can be used to access AWS resources that normally do not have access to. These temporary credentials consist of an access key ID, a secret access key, and a security token),* the Maximum Session duration for which the security token remains valid, can be set for a maximum of 12hrs.

But for security or any other reason it may further be reduced to a lesser duration, for example in our client's environment, it is set to 1hr only.

REQUIREMENTS:

1. **SCHEDULE THE JOB VIA ANSIBLE TOWER**
   The tasks are required to be triggered using Ansible Tower, and hence needs to be set up as a scheduled job/s.
2. **GET THE RESULT BACK ON ANSIBLE TOWER**
   The result of the above triggered job/s should be visible in Ansible Tower. The user is not expected to connect to remote machine for checking the status of the job executed in point 1.

IMPLEMENTATION:

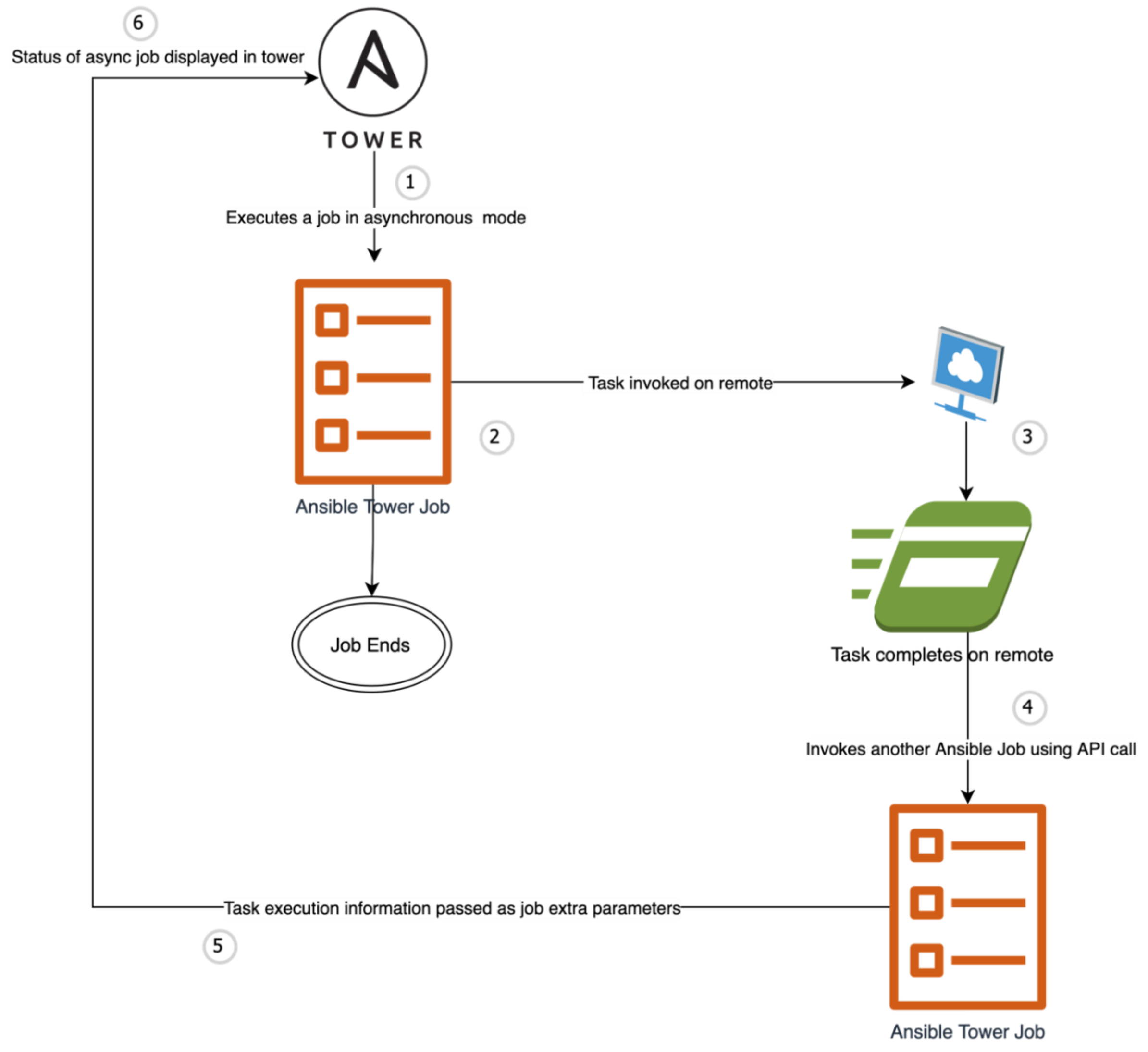
The steps involved in the solution are

ASYNCHRONOUS APPROACH

1. Ansible tower executes the tasks which are required to be run on the remote machine in asynchronous mode. With this approach the connection does not remain established between the Ansible Tower and the remote machine.
2. The Ansible tower job gets completed and finishes its run, after invoking the task execution on the remote machine.
3. The task takes its time and completes its execution on the remote machine.

INSPIRED BY SIGNALLING

4. On completion it invokes (a signal) another Ansible Tower job using simple web based API calls.
5. The result of task that ran on the remote machine, is passed as extra parameter to job.
6. The extra parameter value is displayed in the Ansible Tower as part of the job run which shows the status of the job that was initially invoked on the remote using asynchronous mode.


6
Status of async job displayed in tower
TOWER
1
Executes a job in asynchronous mode
Task invoked on remote
2
3
Ansible Tower Job
Job Ends
Task completes on remote
4
Invokes another Ansible Job using API call
Task execution information passed as job extra parameters
5
Ansible Tower Job

EVALUATION:

- Without this approach
  - The session was timing out after an hour at client's environment.
  - As a result of which the automation of database backups using Ansible was not feasible.
- With this approach
  - It became possible to take the backup of a database having an approximate size of 17TB.

- The SQL query for taking the backup, runs for approximately 20 hours.
- Since this is an asynchronous approach
    - The Ansible job terminates after invoking the SQL query on the remote machine, hence no sustained session.
    - The SQL query runs detached on the remote machine.
    - The status of SQL query is sent back to the Ansible Tower by invoking another independent job template.

WORK AROUNDS:

Subsequent jobs:

If there are any subsequent tasks in the Ansible playbook, that are required to be executed based on the success of the task executed asynchronously, then they can no longer be part of the same playbook.

However, there can be two ways to run them

1. Append the subsequent tasks to the task which is executed asynchronously and run all of them in an asynchronous way.
2. Execute the subsequent tasks from a separate play after receiving the status of the task that was executed asynchronously.

CONCLUSION:

This method made it possible to run time consuming tasks with Ansible, using an asynchronous approach, without maintaining a session or a connection between the Ansible tower and remote nodes.

With this approach, we can use Ansible to run long duration tasks like:

- Install a big software, that may run for long
- Take huge database backups, which may run for hours
- Wait for any events that might take longer on remote hosts
- Wait for cloud resources to be in a required state.