ANSIBLE

ansible.com
+1 800-825-0212

WHITEPAPER

The Benefits of Agentless Architecture

INTRODUCTION

Ansible is an open source IT configuration management, deployment, and orchestration tool. It is unique from other management tools in many respects, aiming to provide large productivity gains to a wide variety of automation challenges. While Ansible provides more productive drop-in replacements for many core capabilities in other automation solutions, it also seeks to solve other major unsolved IT challenges by unifying configuration, deployment, and complex IT process orchestration.

One of the most important challenges in this environment is to do all of the above while providing a robust, easy to manage architecture–a problem that is frequently not well solved in this application space. A management tool should not impose additional demands on one's environment–in fact, one should have to think about it as little as possible. It should be transparent and maximize productivity gains. *You shouldn't have to manage your management system*. Let's see how Ansible achieves these gains using a unique agentless architecture.

TECHNOLOGY OVERVIEW

The core Ansible project manages systems by connecting to them over SSH, either using paramiko (a Python library), or by using native openssh (parameter: -c ssh). When using OpenSSH, connection reuse features are enabled by default if supported by the SSH client, and in either case, SSH is used as a transport, but is not treated as a shell.

Instead, modules, which are small Ansible programs containing baked-in arguments, are transferred over SCP or SFTP to a temporary directory on the remote machine, executed, and then removed in one action. The modules return JSON over standard output, and this return data is processed by the Ansible program on the controlling machine.

The result of this is that a very large amount of remote activity can occur with a minimum of traffic interchange. Modules manage "idempotent resources" and are not simply commands or scripts. For instance, a module can decide that a package should be installed at a particular version, and knows not to execute any commands if the system is already in the proper working state.

IMPROVED NETWORK SECURITY

By not requiring any remote (or even central, technically!) server agents, Ansible has a very low attack surface. The only program you need to run is the OpenSSH daemon, which is one of the most critically reviewed programs in the entire world. Ansible recognizes that cryptography is an extremely difficult thing to get right, so it does not use its own daemon and certificate system, but rather relies on the most secure remote management system available. OpenSSH is available for an extremely wide variety of distributions and is very lightweight. When security issues in OpenSSH are discovered, they are patched extremely quickly.

While we recognize that it is possible to write a secure OpenSSL implementation, we also note a track record of remote exploits against similar tooling in this application space, and wish to avoid such problems as much as possible.

ENABLING NON-ROOT LEVEL ACCESS (AND SUDO)

Ansible playbooks can log in remotely as any user account. From this account, they can run modules as the user that initiated the connection, or they can sudo to any other user (including root). Direct root login, if desired, is also supported. Sudo with password, or password-less sudo, is supported equally. These approaches are ideal when managing parts of a system where root login is not allowed at all, or if root login is not allowed but users can sudo to root. One such example is an unprivileged user can manage content in their home directory with Ansible, even if they do not have root or sudo privileges on their machine.

File transfer is not limited even when using sudo. Ansible also contains a sudo-compatible file transfer facility, where content is transferred as normal with SFTP, and then Ansible moves the file into place with credentials. Ansible is also intelligent enough to not transfer files that do not need to be transferred if the source and target checksums already match.

LIMITING TRANSFER OF POTENTIALLY SENSITIVE DATA

Ansible transfers a bare minimum of data to machines it manages. Since the central server is in control of decision-making logic, only variables needed by remote nodes are sent to them. For instance, if there is a global variable set called "foo", this variable is never sent to the remote server unless it is explicitly used in a resource or template (all templates are evaluated on the central management machine). As such Ansible pushes out only what remote nodes need to see–the bare minimum.

Similarly, Ansible contains no custom file server implementation. It moves files using SFTP, SCP, and (serverless) rsync (over SSH), and only files that need to be transferred in the playbook. The result is that it impossible for a managed host to request files or templates meant for another machine and to access sensitive data not meant for it. There is no way for a remote host to 'browse' what data may apply to other computer systems.

This makes Ansible ideal for environments where data is extremely sensitive, including when working with social science workloads, healthcare, and government applications.

CREDENTIAL SEGREGATION

Ansible is useful in environments where different users have different levels of trust. It is possible to make a common definition of manageable hosts available, and then use the individual access credentials of users to allow them access to remote machines. This can allow, for instance, developers to have managed access to development machines, QA engineers to QA machines, and administrators to production machines, without the accidental risk of a developer pushing content to production.

NO TAKEOVER BY COMPROMISE OF CONTROL SERVER

Because Ansible playbooks require user credentials to run, there is no central point of takeover to establish a "botnet" by only having access to the configuration content, thus the management software chain, without access to SSH keys, does not become an attack vector. Users can individually encrypt their keys and do not have to share these passwords with anyone. In alternative management forms, access to commit to a software repository can result in the system automatically deploying a change. With Ansible, users need two things: access to the repository, and credentials to manage the remote system. When using locked keys (or even passwords), there is no central server to exploit for a security vulnerability, nor does gaining physical access to the management server allow for configuration of remote hardware.

NO "MANAGING THE MANAGEMENT"

One of the major problems of many configuration management solutions is one of "Managing The Management". In order to start managing machines (see "Zero Bootstrapping") software must be installed on the remote machines. When updating the management software, often the various agents must be updated first (and many systems cannot self update!).

Sometimes, compatibility problems arise between server and agent versions, or between agent and language runtime versions. Ansible avoids this problem of transferring modules over SSH, which is a binary that is already part of the OS and is in the core of every major operating system. Further, by not requiring any agents, any sort of agent crash scenario is avoided, so you will have low risks of severing your ability to manage the box.

CENTRAL SERVER SCALABILITY

Since Ansible pushes out changes to remote servers (and can easily configure rolling updates because of this, see other whitepapers), Ansible is immune to the "thundering herd" management problem. In some other solutions, management agents checking in periodically hammer the server, often overwhelming it and causing the need to scale out the management control system horizontally and vertically. Further, the management server frequently has to do very expensive computations for the remote nodes. Ansible solves this problem by being push oriented, and only has to talk to a finite, but configurable number of nodes at one time. It offloads a maximum amount of remote computing needs to remote nodes, therefore sharing the workload among computer systems. Even a laptop is a sufficient deployment platform for an Ansible control server.

RESOURCE UTILIZATION

When Ansible is not managing remote nodes, it is not doing anything on those nodes. This means there is no daemon to consume memory or CPU. It has been reported that with some solutions application servers can yield visible performance degradation (per monitoring data) during “wake up” configuration windows, or agents that can consume 400MB+ of memory each. In a virtualized environment, all of this resource consumption can quickly add up, requiring more hardware outlays. Ansible allows your performance-critical workloads to use all of your CPU, and you can choose when you wish to run your management intervals–there is no chance of memory leak or agents that may also crash, cutting off your ability to manage the box.

FIREWALL FRIENDLY & DETERMINISTIC

Unlike some message-bus based systems, Ansible does not need to hold persistent connections open between the management machine and central node. In production cases, this can play havoc with firewalls who do not like long lived connections, which Ansible avoids. Sometimes when connections drop management connectivity to these applications cannot be reset until the agents are restarted, and Ansible does not have this problem.

While not strictly a property of an agentless system, Ansible also avoids another problem found in these architectures–getting a deterministic response. With Ansible, if a node is down, you will know about it, and get a failure message, which you can either ignore or act on. There is not a notion of only the nodes that are present responding, and having silence from those you meant to talk to but were not reachable. This is super important information–are you are deploying a software update to all of your nodes or just an arbitrary subset? Knowing what nodes you cannot contact is vitally important, and a publish-subscribe architecture frequently does not provide this information.

ZERO BOOTSTRAPPING

Ansible can start managing remote machines immediately, without any agent software installed. In a “brown field” deployment scenario, a site may have thousands of existing machines and need to deploy a software change to all of these systems. Ansible can start communicating to all of these machines right away, reaching out and managing them without a lengthy setup process. While Ansible does require some software installation to manage legacy hosts (such as a JSON library), Ansible playbooks can be used to bootstrap this via the “raw” module.

The ability to manage systems without installation of additional agents also makes Ansible a consultant or vendor’s best friend. Often a customer may be running any number of software systems and may not want to commit to adopting a particular new system for management. Writing automation in Ansible ensures that when configuration is done, the customer will not have to maintain the deployment system unless they want to. Of course, if they want, they can keep using Ansible and the simplicity should be appealing.

KERBEROS SUPPORT

When used with the native OpenSSH transport, Ansible supports Kerberos authorization. For users with Kerberos environments, this provides excellent security features and very clean central administration.

JUMP HOSTS AND TUNNELS

Also when using the native OpenSSH transport, Ansible can take advantage of user configured SSH "jump hosts" (bastion hosts) and tunneling, as set up in the user's SSH configuration file. In cases where IT policies require logging in to one host to get at others, Ansible can make this experience be as seamless as possible.

HOW RESULTS ARE ACHIEVED

As mentioned previously, Ansible achieves its agentless support by leveraging OpenSSH, and by transferring compact auto-generated modules to remote machines that "self destruct", rather than actually executing Unix commands. These modules describe desired states as well as ordered processes, and return JSON data.

Because of the way operations are executed, (especially with the OpenSSH transport) this
is a very efficient approach that can reuse connections and uses a minimal amount of network traffic.

This approach adds numerous security benefits, and also improves both client and central management server resource utilization, while eliminating all of the concerns of "managing the management" that comes with classic agent based systems. Additional options such as non-root access and a reduced attack surface further add to the appeal of the configuration.

ADDITIONAL DOCUMENTATION

More information about Ansible, including complete documentation, can be found at
docs.ansible.com. An open source project mailing list is available and linked on the project site.

EXAMPLES AND FURTHER INFORMATION

Some basic examples of Ansible content can be found at:
https://github.com/ansible/ansible-examples

For more information about Ansible, services, support, and other details, contact AnsibleWorks at info@ansible.com.