ANSIBLE

ansible.com
+1 800-825-0212

WHITEPAPER

Achieving Rolling Updates & Continuous Deployment with Zero Downtime

Get started with ANSIBLE now:
ansible.com/get-started-with-ansible
or contact us for more information:
info@ansible.com

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.

One of these challenges is to enable continuous application deployment with zero downtime. This goal has often required extensive custom coding, working with multiple software packages, and lots of in-house-developed glue to achieve. Ansible provides all of these capabilities in one package, being designed from the beginning to orchestrate exactly these types of scenarios.

WHY CONTINUOUS DEPLOYMENT

Over the last decade, analysis of software and IT practices have taught us that longer release cycles (aka "waterfall projects") have dramatically higher overhead than more frequently released, ("iterative" or "agile") shorter cycles. As a release begins, a quality assurance team is spun down, waiting for things to test. IT does not have anything to deploy. Towards the end of a release cycle, QA is running at full force as is IT, but development is split between bugs and planning the next major release. Context switches are frequent.

Perhaps more urgently, longer release cycles also mean a longer delay between the time when bugs are discovered and when they are addressed, which is especially a problem in large traffic web properties where single issues can affect millions of users. To get out of this problem, the software development industry is rapidly moving towards "release early, release often", often under the banner of "Agile Software Development". Releasing early and often means there is less switching of gears between operation modes for any team members, and changes can be made, qualified, and deployed in much shorter times. Various approaches such as QA automation engineering and Test Driven Development (TDD) increase the effectiveness of these techniques.

It stands to reason that if "release early, release often" is progress, continuous application delivery/deployment may be nirvana. While "release early, release often" and agility is a progressive spectrum, an organization's movement anywhere towards this direction can yield impressive results. To achieve this, automation is key, so there is a huge focus around the technology that enables quick turnaround, requiring human intervention only when necessary.

In this document, we'll highlight why Ansible is the ideal tool to connect your computer systems to enable these kind of processes.

WHY ZERO DOWNTIME?

Downtime and outage windows result in either lost revenue or customer unhappiness. For a major web application with users in all timezones around the world, going down for an outage is only to be reserved for the most dire and complicated of upgrade processes–certainly not updating your application versions. Even for internal applications at large organizations (imagine an accounting or intranet system), an outage in a critical system can mean major dents in productivity. The goal of any automation process should be to allow configuration and updates in a manner that does not dent operational capacity.

Zero downtime is achievable, but it requires support from tooling–a strong multi-tier, multi-step orchestration engine like Ansible–to make it manageable.

JENKINS AND THE BUILD SYSTEM

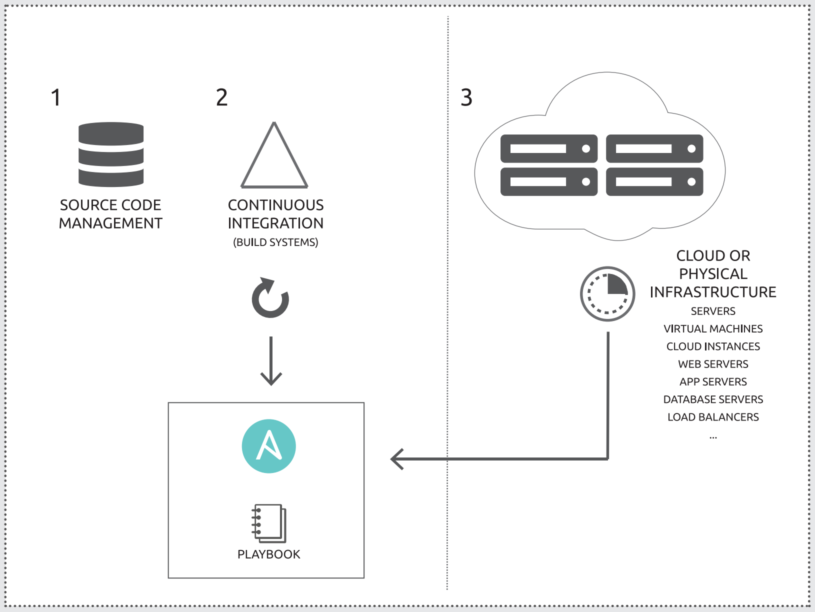
Before Continuous Delivery/Deployment can be achieved, the first step is to achieve Continuous Integration. Simply put, Continuous Integration systems are build systems that watch various source control repositories for changes, run any applicable tests, and automatically build (and ideally test) the latest version of the application from each source control change.

One of the most popular CI systems among Ansible users is the open-source Jenkins (http://jenkins-ci.org) though any CI system can be used for this step.

The key handoff for Continuous Deployment is that the build system will invoke an Ansible playbook upon a successful build. In fact, as we'll detail later, we can use an Ansible playbook against a stage environment to do testing prior to a production deployment! Users who also run unit or integration tests on code as a result of the build will also be one step ahead of the game, but a QA/stage environment modeling production ups the ante and substantially improves predictability.

ROLLING UPDATES AND MULTI-TIER APPLICATIONS

The absolute requirement in a Continuous Deployment system is the ability to finely orchestrate rolling updates among various architecture tiers in an application. Ansible's unique multi-tier, multi-step orchestration capabilities, combined with its push-based architecture, allows for extremely rapid execution of these types of complex workflows. As soon as one tier is updated, the next can begin, easily sharing data between tiers.



One of the core features of Ansible that enables this is the ability to define "plays", which select a particular fine-grained group of hosts and assign some tasks (or "roles") for them to execute. Tasks are typically declarations that a particular resource be in a particular state, such that a package be installed at a specific version (or be up to date, etc), or that a source control repository be checked out at a particular version.

In most web application topologies, it is necessary to update particular tiers in a tight sequence. For instance, the database schema may need to be migrated and the caching servers flushed prior to updating the application servers.

More importantly, it is vital to not be managing the applications and configuration of the system for all systems in production at the same time.

When services restart, they may not be immediately available. Replacing an application version may also not be instantaneous. It is important, then, to be able to take machines out of a load balanced pool prior to update. It is also then critical to be able to automate the actions of taking machines in and out of the pool.

Ansible allows you to control of the size of a rolling update window through the 'serial' keyword. Additionally, Ansible's rolling update handling is smart enough so that if a batch fails, the rolling update will stop, leaving the rest of your infrastructure online.

CONTINUOUS DEPLOYMENT OF MANAGEMENT CONTENT

In addition to continuous deployment of production services, it's also possible to continuously deploy Ansible playbook content itself. This allows systems administrators and developers to commit their Ansible playbook code to a central repository, run some tests against a stage environment, and automatically apply those configurations to production upon passing stage. This allows the same software process used for deploying software to be applied to configuration management, reaping many of the same benefits.

As software or configuration change is a significant cause of unintended outage, both automated testing and human review can be useful measures to implement. This can be coupled with a code review system, like Gerritt, to require sign off from multiple users before changes are applied.

ROLLING UPDATES AS APPLIED TO LOAD BALANCING & MONITORING

Ansible is adept at working with tools such as load balancers in the course of executing a rolling update. In the middle of any playbook “host loop”, it is possible to say “do this action on system X on behalf of host Y”.

This includes both communicating with load balancers of all kinds, as well as flagging outage windows for particular hosts to disable monitoring alerts for the hosts currently being updated. Simple idioms like “disable monitoring - remove from pool - update tier - add to pool - enable monitoring” allow for updates without downtime or false alarm pager alerts, all hands off, that work in an automated manner every time without manual intervention.

INTEGRATED STAGE TESTING WITH ANSIBLE

Using Ansible with multiple inventory sources allows for testing an Ansible playbook with a stage environment and requiring successful execution of an update prior to rollout into production. This setup, while optional, is ideal for modelling production update scenarios (perhaps at a smaller scale) prior to updating systems in the real world. Just use the “-i” parameter to an Ansible playbook to specify what inventory source the playbook should run against, and use the same playbook for both stage and production.

VERSION CONTROL BASED DEPLOYMENT

Various organizations like to package their applications with OS packages (RPM, debs, etc) but in many cases, particularly for dynamic web applications, such packaging is unnecessary. For this reason Ansible contains numerous modules for deploying straight from version control. A playbook can request that a repository be checked out at a specific tag or version, and then the system will make sure this is true across the various servers. Ansible can report change events to trigger additional follow up actions when the version of software needed to change, so that associated services are not restarted unnecessarily.

CALLBACKS AND PLUGGABILITY

When in a culture of continuous integration and deployment, it can often be useful to provide alerts when events occur. Ansible includes several facilities to do this, including an integrated mail module. Additionally, a callback facility allows for plugging in notifications with arbitrary systems, which can include chat broadcasts (IRC, Campfire, etc), custom logging, or internal social media.

THE IDEMPOTENT RESOURCE MODEL AS APPLIED TO DEPLOYMENT

One of the key features that makes Ansible so interesting for software deployment is it allows the use of state-based resource model, made popular in configuration management tooling, within the process of updating software. While traditionally some open source tooling has had to be supplemented with additional deployment software and scripts, this is not the case for Ansible.

This is made possible by the ability for Ansible to finely control order of events between different tiers of systems, to delegate actions to other systems, and also to mix resource model directives (“the state of package X should be Y”) versus traditional command patterns (“run script.sh”) in the same process.

Ansible also allows easily executing commands to test for various conditions, and then making conditional decisions based on the results of those commands. By unifying configuration and deployment under one toolchain, it is possible to reduce overhead of managing different tooling, and also achieve better unification between OS and application policy.

EMBEDDED TESTS INTO DEPLOYMENTS

With great power comes great responsibility. One of the dangers of setting up an automated continuous deployment system is the chance for an bad configuration to be propagated to all of one's systems. To help protect against this, Ansible allows for adding tests directly in playbooks that can abort a rolling update if something goes wrong. Arbitrary tests, deployed with the 'command' or 'script' module, or even written as native Ansible modules, can be deployed to check for various conditions. This may include the functioning state of the service.

At any point, usage of the 'fail' module can abort execution for the host. Thus, it's easy to catch errors early on in a rolling update window. For instance, suppose a difference between stage and production resulted in a configuration error that would only break a production deployment. In this case, Ansible playbooks can be written to stop the update process in the first stage of the rolling update window. If there were 100 servers and the update window was 10, the update window would stop and allow intervention. Once corrected, simply re-run the playbook to continue the update.

In the event of a failure, Ansible does not keep going and half-configure the system. It raises the error to your attention so it can be managed, and produces summaries of what hosts in the update cycle had problems and how many changes were executed on each platform.

COMPLIANCE TESTING

In many environments, a configuration change should not be applied unless some change is actually needed, and that it should be first understood and agreed upon by a human prior to "pushing the button". In these cases, a continuous deployment system can still be leveraged with some caveats.

In this case, Ansible contains a dry-run mode via a "--check" flag that will report on what changes would have been made by running a playbook process. As this will skip the actual command executions and not account for possibly failing scripts, this is just a guideline, but it is a very useful guideline and a useful tool to have in one's toolkit.

Even if deploying continuously when new build products are available, it is possible to run compliance checks even more frequently, to report on when things in production might have changed due to human invention and need to be corrected by another playbook run. This can be an easy way to detect that a software change may need to occur, a permission may need to be corrected, and so on.

CONCLUSIONS

Ansible contains numerous tools and features to make it an ideal Continuous Delivery/Deployment solution. These include the ability to finely orchestrate multi-tier, multi-step processes in zero-downtime rolling update workflows. This is made possible by Ansible's unique architecture, and coupled with the agentless system (which adds security and no need to manage-the-management), it is easy to establish simple-human readable representations of what were previously complex manual IT processes. The days of locking an ops team in a conference room to perform a mix of manual and automated steps are gone, Ansible puts deployment on full autopilot.

"Full autopilot" of course means enabling the true promise of agile processes: delivering change faster, with fewer errors, and with fewer expensive context shifts. Eliminating outage windows, and especially manual change error, is something that is expressly critical for any core internal IT service or high-traffic web property.

Though architectural features in Ansible, and the ability to tightly integrate with continuous integration systems like Jenkins, not only can configuration be automated, but all aspects of IT processes. This is why we refer to Ansible as an "Orchestration Engine", as opposed to filing it under just configuration management or software deployment.

USE CASES AND QUOTES

AppDynamics

AppDynamics uses Ansible for all of our OS and deployment tasks. We have Ansible integrated with Gerrit for code review and do continuous deployment from Jenkins to all of our servers via Ansible playbooks every 15 minutes.

– Thomas Morse, Director of IT & SaaS Operations, AppDynamics

GAWKER

Gawker uses Ansible to automate deployment for the Kinja platform, powering sites such as jalopnik.com, where it interacts with Jenkins and our Netscaler load-balancers to carefully roll out new builds across the application servers. Often more than ten times a day.

– Jim Bartus, Director of Tech Operations, Gawker Media

DOCUMENTATION

More information about Ansible, including complete documentation, can be found at 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 implementing zero-downtime rolling updates can be found at https://github.com/ansible/ansible-examples. Users looking to integrate such a process with their source control, establish a build system, or integrate with their network environment may wish to reach out to us for more information.

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

ANSIBLE