WHITEPAPER

CONTINUOUS INTEGRATION AND DELIVERY WITH ANSIBLE

INTRODUCTION

Ansible is a very powerful open source automation language. What makes it unique from other management tools, is that it is also a deployment and orchestration tool. In many respects, it aims 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 IT challenges.

One of these challenges is to enable continuous integration and continuous deployment (CI/CD) 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 success. Ansible provides all of these capabilities in one composition, being designed from the beginning to orchestrate exactly these types of scenarios.

WHY CI/CD

Over the last decade, analysis of software and IT practices have taught us that longer release cycles ("waterfall projects") have dramatically higher overhead than more frequently released ("iterative" or "agile") shorter cycles. As a release begins, a quality assurance team is prepared, waiting for things to test, and 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 resolve this problem, the software development industry is rapidly moving towards "release early, release 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, CI/CD may be nirvana. While agility is a progressive spectrum, an organization's movement in 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 and Red Hat® Ansible® Tower are the ideal tools 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 time zones 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 impacts on productivity. The goal of any automation process should be to enable updates in a manner that does not impact operational capacity.

The goal of any automation process should be to enable updates in a manner that does not impact 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.

OTHER BUILD SYSTEMS

Before Continuous Delivery (CD) can be achieved, the first step is to achieve Continuous Integration (CI). Simply put, CI 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, such as Jenkins (jenkins.io).

The key handoff for CD is that the build system can invoke Ansible upon a successful build. 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. Jenkins can utilize Ansible Tower to deploy the built artifact into multiple environments, but a QA/stage environment modeled after production ups the ante and substantially improves predictability along the lifecycle. The data provided back by Ansible can then be referenced, and directly correlated to an Ansible Tower job in the Build Systems job.

In fact, as we'll detail later, we can use Ansible Tower against a stage environment to do testing prior to a production deployment.

Ansible's unique multi-tier, multi-step orchestration capabilities, combined with its push-based architecture, allow for extremely rapid execution of these types of complex workflows.

ROLLING UPDATES AND MULTI-TIER APPLICATIONS

The absolute requirement in a CD system is the ability to 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, allow 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 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 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 CD 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 Gerrit (gerritcodereview.com), 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 that work in an automated manner every time without manual intervention.

INTEGRATED STAGE TESTING WITH ANSIBLE

Using Ansible Tower 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 modeling 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.

INTEGRATIONS WITH MONITORING TOOLS

Because Ansible is an orchestration system, integrations can be made with APM monitoring tools as well. For instance, let's assume that during an integration test or deployment, an APM system has an agent that needs to be installed/updated on the application. Ansible can have a Role that handles the install of the agent against the application stack. Once the agent is online, Ansible can then set alerts up (if they don't already exist) in the monitoring stack. This monitoring can then provide immediate data back to teams interested in the deployment; letting them know if the application is running without issue.

If something were to occur with the application in production after the upgrade, Ansible could be invoked by the monitoring tool to revert to a previous known good build of the application. Obviously this can be done if, and only if, the application will allow for the reverted build.

CALLBACKS AND PLUGGABILITY

When in a culture of CI/CD, 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 DESIRED STATE RESOURCE MODEL AS APPLIED TO DEPLOYMENT

One of the key features that makes Ansible interesting for software deployment is 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 the 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 make 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.

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 DEPLOYMENT

With great power comes great responsibility. One of the dangers of setting up an automated CD 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. It should be first understood and agreed upon by a human prior to "pushing the button." In these cases, a CD 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.

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.

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.

PUTTING DEPLOYMENT ON AUTOPILOT

Ansible contains numerous tools and features to make it an ideal CI/CD 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 errors, is something that is expressly critical for any core internal IT service or high-traffic web property.

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

EXAMPLES AND FURTHER INFORMATION

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

Try Ansible Tower now:
ansible.com/tower

ABOUT RED HAT ANSIBLE TOWER

Ansible, an open source community project sponsored by Red Hat, is the simplest way to automate IT. Ansible is the only automation language that can be used across entire IT teams – from systems and network administrators to developers and managers. Red Hat® Ansible® Automation provides enterprise-ready solutions to automate your entire application lifecycle – from servers to clouds to containers and everything in between. Red Hat® Ansible® Tower is a commercial offering that helps teams manage complex multi-tier deployments by adding control, knowledge, and delegation to Ansible-powered environments.

ABOUT RED HAT

Red Hat is the world's leading provider of open source software solutions, using a community-powered approach to provide reliable and high-performing cloud, Linux, middleware, storage, and virtualization technologies. Red Hat also offers award-winning support, training, and consulting services. As a connective hub in a global network of enterprises, partners, and open source communities, Red Hat helps create relevant, innovative technologies that liberate resources for growth and prepare customers for the future of IT.

NORTH AMERICA
1 888 REDHAT1
www.redhat.com

EUROPE, MIDDLE EAST, AND AFRICA
00800 7334 2835
europe@redhat.com

ASIA PACIFIC
+65 6490 4200
apac@redhat.com

LATIN AMERICA
+54 11 4329 7300
info-latam@redhat.com

redhat.