MidVision™
Release the innovation

CIO's Guide: How to Avoid Configuration Drift

The failure to detect and understand unplanned changes in the configuration or content of a managed resource increases the risk of operational failure and hinders troubleshooting efforts. Unplanned changes in configuration or in content are referred to as *drift*.

Overview

Proper deployment and configuration is crucial for production and significant effort is spent in pre-production to ensure that managed resources are correctly configured and deployed with specific, verified versions of software. The failure to detect and understand unplanned changes in the configuration or content of a managed resource increases the risk of operational failure and hinders troubleshooting efforts. Unplanned changes in configuration or in content are referred to as *drift*. Production, staging, and/or recovery configurations are designed to be identical in certain aspects in order to be resilient in the event of failures in production. When these configurations drift from one another, they leave what is commonly called a *configuration gap* between them. Configuration drift is a natural occurrence in enterprise and data center environments due to frequent hardware and software changes. Disaster recovery failures and HA system failures are frequently a result of configuration drift.

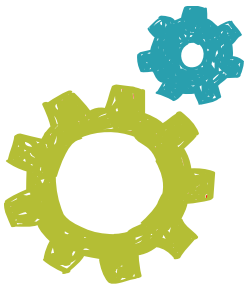
Scenarios

Let's consider some scenarios to illustrate how configuration drift can adversely affect an enterprise. These are intended to be motivating example to illustrate where/when configuration drift management might be applicable.

Scheduled Database Password Change

As part of regularly scheduled maintenance, the passwords for production databases will be updated. This will in turn require the data source configurations on production JBoss EAP servers to be updated with new passwords. The IT organization is not currently using RHQ and is instead using ad-hoc scripts to automate and apply the changes. Any number of things could happen in this scenario that could result in serious problems.

Programmatic errors in the script could result in an incorrect data source configuration keeping production EAP servers offline for an extended period of time as a result of not being able to communicate with the database. Or if the script needs to be manually run on each server, an administrator forgets to apply the changes to one of the servers. These two problems would likely be remedied quickly as the source of the problem could be quickly and easily identified.



Now let's assume that the IT organization is using RHQ to manage its infrastructure. Administrators could create a group for the production EAP servers and then apply the data source configuration change as a group configuration update. Suppose the update fails on one of the EAP servers fails. Maybe the host machine is offline. RHQ will report the results of the operation, making it clear that the configuration update failed on one of the EAP servers. Now if the administrator is waiting intently for RHQ to report the results of the operation, the error is easily rectified. But suppose the administrator gets distracted by some other production issues and forgets about the configuration update. Discovering the problem will be delayed, requiring additional time and maintenance which adds to overall operational costs.

Database URL Change

Let's consider another example involving data source configuration changes. A new production database server has been installed and EAP servers need to be updated such that JDBC URLs refer to the new server. To make things more interesting, let's say that the old database server is kept running. Maybe it will be used for backups. Again let's start with an IT organization that is using an ah-hoc scripting solution to apply the updates. If administrators fail to update one of the EAP servers or if programmatic errors in the scripts result in one of the EAP servers not getting updated, this could result in serious problems that may not manifest themselves nearly as easily as in the database password example.

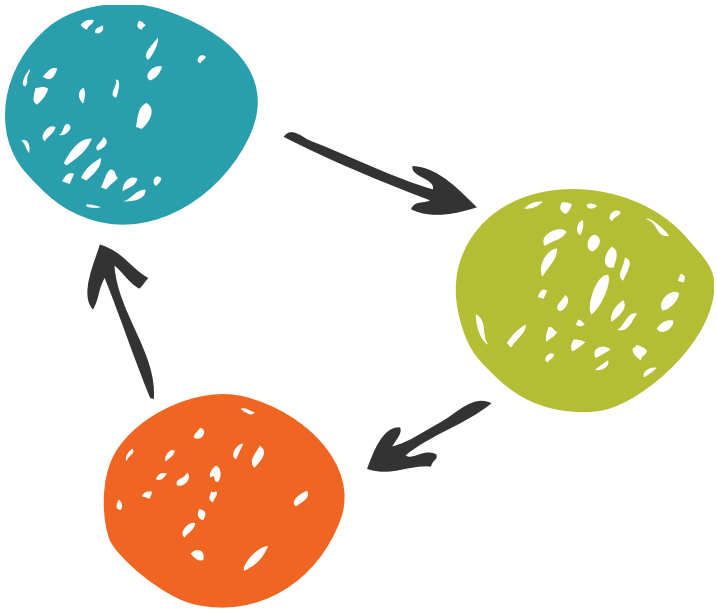
Now let's say that the IT organization is using RHQ to manage its EAP servers. Let's further suppose that there are advanced RHQ users/administrators who are utilizing the CLI to further automate the task. A CLI script is written to apply the group configuration update. The script iterates over each EAP server in the group and fetches and updates the resource for each data source as necessary using ConfigurationManager.updateResourceConfiguration. The script writer fails to understand a subtle yet important point which is when that method returns, the operation may not have completed. The script in effect assumes that since no exception has been thrown the updates completed successfully. Let's assume that one of the EAP servers fails to get updated for some reason. Although RHQ will report the failure, administrators may not recognize it soon enough still resulting in serious problems that are costly both in terms of time and money.
Increase heap size for EAP Servers

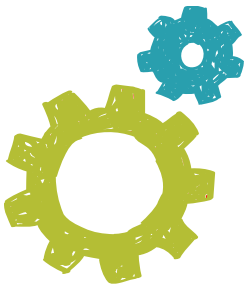
We run out of heap space for an EAP server in a production environment which results in some time. The fix is easy enough - increase the maximum heap size. Suppose that the heap size is increased as a one off change where an administrator logs onto the

server, increases the heap size, and restarts the EAP server. Then at some point down the road, the machine on which the EAP server is running is taken offline for maintenance. Another EAP server is brought online. The problem though is that the new EAP server has the lower heap settings, and it too will eventually start running out of memory.

Deploy an Updated Version of an Application

Let's consider an example in which we are deploying an updated version of an internally hosted application in the form of an EAR or WAR. The application is running on an EAP cluster. This example is different from the previous one in that it deals with a content change as opposed to a configuration change. The application update is to be done as a rolling outage where one server will be taken out of rotation at a time, have the updated application deployed to it, and then go back into rotation. There are a number of different possible failure points. We will look at a few of them. First, we could wind up deploying the wrong version of the application. Depending on how rigorously we verify that have deployed the correct bits, the problem could go undetected for some time. A second problem that could arise is failing to update of the cluster nodes. Speaking from personal experience, I know that this can wind up being a difficult problem to debug. Lastly, suppose that the updated version of the application requires an update to some other library that is deployed separately and that library does not get updated.





Terminology –

User Level and GUI Terminology

Drift

Unplanned changes in monitored files such as configuration files, or content.

Drift Monitoring

In RHQ, the general notion of periodically scanning the file system for changes to files that likely indicate drift.

Drift Detection Definition

A definition guiding drift monitoring for a resource. It specifies, using a base directory and includes/excludes filters, which directories should be monitored. Additionally, it sets monitoring properties such as scan interval, enablement, and drift handling options. Also called a Drift Definition. A resource can have zero or more drift definitions.

Drift Detection Run

A single execution of drift detection for a resource. In other words a single drift definition applied to a single resource, at a specific time.

Drift Instance

In RHQ, a specific occurrence of drift. Meaning a file change detected during a drift detection run. In the GUI, just known as Drift.

Note: In general a drift instance reflects an unexpected change. But RHQ does provide a 'planned changes' mode in the drift definition. Although drift detection is always performed the same way, RHQ will handle the drift instance differently in planned changes mode, specifically, by disabling alerting for the drift instance and omitting it from the drift history view.

Snapshot

The file-set (really, file-version-set, as it's not just file names, it's the actual bits) resulting from a drift detection run. In other words, a 'snapshot' of the actual files on disk at a particular time.

Initial Snapshot

The snapshot resulting from the first drift detection run for a drift detection definition. The initial snapshot is marked as version 0. Variations from the initial snapshot will generate drift.

Snapshot View, Snapshot Delta View

A snapshot always represents a full file-set, as it exists for the resource at the time of the drift detection run. The GUI provides two views of a snapshot. The 'Snapshot View' shows the complete file-set. The 'Snapshot Delta View' shows only the file differences between tow snapshots (typically the previous snapshot). Other than the initial snapshot, the snapshot delta view is the default. The user can toggle between views as desired.

Snapshot Diff

A diff between two snapshots which can be from the same resource or from different resources. The diff identifies files present in one snapshot and vice-versa. It also identifies the files that exist in both but whose content differs.

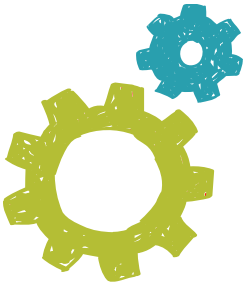
Remediation

The act of resolving drift. This is analogous to resolving a merge conflict in a version control system like Git. Resolving drift can be done in a number of ways including:

- Revert back to a previous state
- Acknowledge and accept the change
- Change to something other than a previous state

Pinned Snapshot

By default a drift detection run looks for changes between the previous snapshot and the current file system state. This rolling snapshot approach ensures each change to a file will result in only one drift instance. For more strict environments we offer the ability to always detect against a specific, or 'pinned' snapshot. The user can pin a snapshot via the GUI (or CLI). In this situation a drift detection run always compares the current file system to the pinned snapshot. This can result in the same drift being reported on each detection run, until the situation is remedied.



Golden Snapshot

A golden snapshot is a customer's expected snapshot when performing a drift detection run on a specific drift definition. A golden snapshot is associated with a resource type, and then can be pinned to any number of resources, or compatible groups, of that resource type.
Golden snapshots result from promoting a trusted resource's snapshot. Meaning, a customer can iteratively work to define a drift definition at the resource level and, when satisfied with a resulting snapshot, promote the snapshot to golden. At which point it can then be manipulated at the type level, thus forcing sets of resources to comply and report drift for variations from the golden snapshot.