Whitepaper
Release management -
An automated release process

Contents

Overview

Release Management is the process responsible for planning, scheduling, and controlling the build, in addition to testing and deploying Releases. Release Management ensures delivery of new and enhanced IT services required by the business, while protecting the integrity of existing services.

Objectives

It is about ensuring that an existing working application is not replaced with a non-working one. Besides it is about not introducing a working application which breaks other working applications or components. In real life, we see both these cases happening frequently. So, to reiterate, the agenda of release management is totally production, operation, infrastructure availability focused

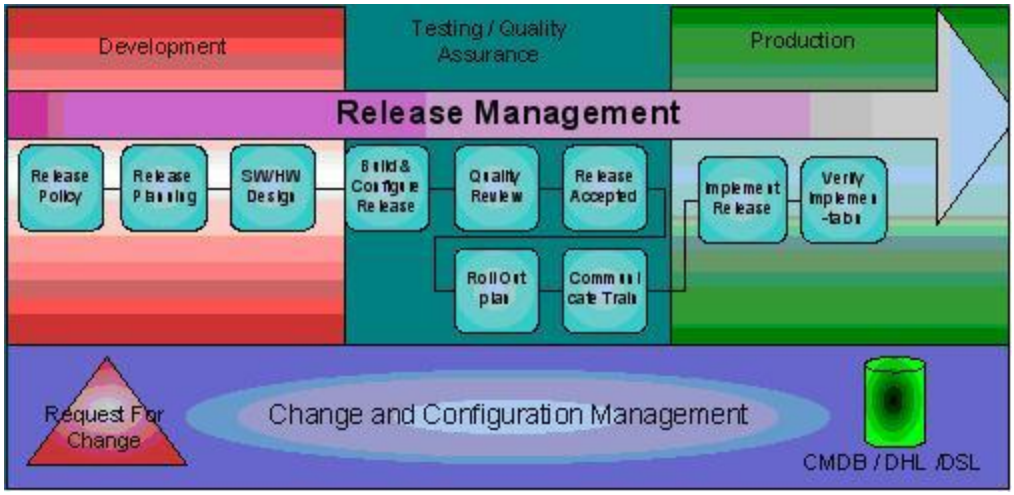
- Increase the number of successful Releases, including reducing the number of releases with unexpected outcomes.
- Decrease the number of incidents caused by releases.
- Create a single documented process for managing all releases.
- Maintain a single repository for recording all releases through the lifecycle.
- Ensure that the process is adopted, adhered to, and escalated to management if there are compliance issues.
- Improve coordination between groups to ensure smooth and timely delivery of IT services.
- Improve productivity by establishing standard release processes and tooling.
- Initiate the release management process to provide sufficient lead-time for adequate impact analysis by the CAB.
- Ensure that auditable release controls are established and documented.
- Communicate releases to affected client representatives, clients (where appropriate), and other IT organizations (where appropriate).
- Streamline the procedures so that there is an appropriate balance between the complexity of the Release and the required controls.
- Build lessons learned from the release management process that could be applied to other areas of Service Management

Basic Flow of Release Management

Figure below outlines the basic steps that constitute a “Release Management” process. In this diagram the movement of a Release from left to right depicts progress through various environments (Development, QA and Production), each of which is a distinctive operating environment that progressively seeks to replicate Production conditions and functions separately

from the other although they leverage common methods for promoting a Release between them

It is important to note that Figure below does not reflect certain environments (e.g. Sandbox and pre-Production) which exist in more mature operations.



Release Management workflow

When using Release Management, here are the steps that you typically follow:

Create release definitions: You start using Release Management by creating a release definition in the RELEASE hub of your team project. A release definition specifies (a) What to deploy - the set of artifacts that constitutes a new release, and (b) How to deploy - the series of automation tasks that should be run in each environment.

Add environments: You add one or more environments to a release definition. Each environment is simply a named logical entity that represents a deployment target for your release. For example, you create environments for test, quality assurance, staging, and production. You then edit the environments to specify the lists of users that must approve the deployment where this is appropriate.

Add tasks: You add automation tasks to each environment. These tasks describe the deployment and testing process. There is a wide range of pre-defined tasks you can use. These tasks can take advantage of shared custom variables and built-in properties in their configuration. Your tasks may need to connect to other services, cloud platforms, or third party deployment and testing services. For this, you define your global service endpoints.

Create and deploy releases: Once you have created a release definition, you can manually create a new release based on this definition and deploy it to various environments, or you can let a release be created automatically and deployed upon completion of a build. There is a wide range of options for creating a release and deploying it, including release date and time, pipeline or parallel deployment, and more. You can also monitor the new changes that went into each release.

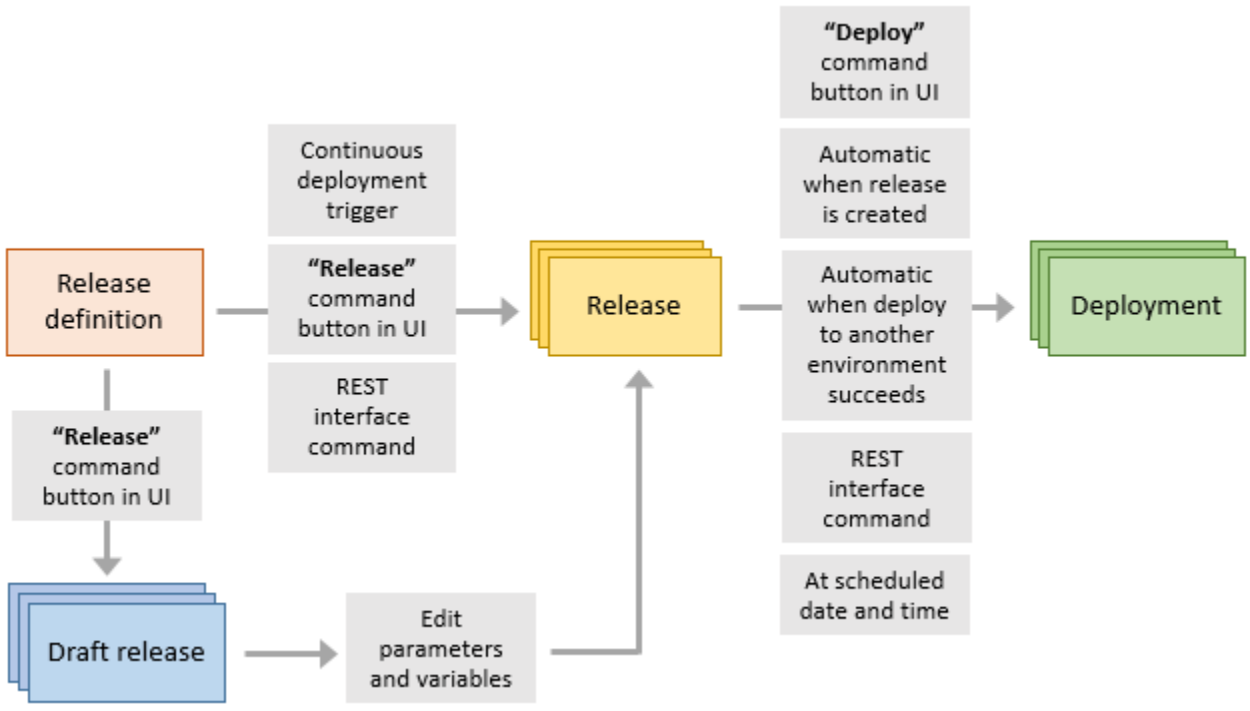
Track deployments: As a release is deployed to various environments, you track its progress. You can approve new deployments to an environment, and view the logs of deployments as they happen.

Understand releases and deployments

In Release Management, the terms release and deployment have very different meanings:

- A release is the package or container that holds a versioned set of artifacts specified in a release definition. It includes a snapshot of all the information required to carry out all the tasks and actions in the release definition, such as the environments, the task steps for each one, the values of task parameters and variables, and the release policies such as triggers, approvers, and release queuing options. There can be multiple releases from one released definition, and information about each one is stored and displayed in Release Management for the specified retention period.
- A deployment is the action of running the tasks for one environment, which results in the application artifacts being deployed, tests being run, and whatever other actions are specified for that environment. A release creates, initializes, and starts each deployment based on the settings and policies defined in the original release definition. There can be multiple deployments from each release.

The following schematic shows the relationship between release definitions, releases, and deployments.



Releases (and, in some cases, draft releases) can be created from a release definition in several ways:

- By a continuous deployment trigger that creates a release when a new version of the source build artifacts is available
- By using the Release command in the UI to create a release manually
- By sending a command over the network to the REST interface

However, the action of creating a release does not mean it will automatically or immediately start a deployment. For example:

- There may be deployment triggers defined for an environment, which force the deployment to wait; this could be for a manual deployment, a network request to the REST interface, until a scheduled day and time, or for successful deployment to another environment.
- A deployment started manually from the Deploy command in the UI, or from a network command sent to the REST interface, may specify a final target environment other than the last environment in a release pipeline. For example, it may specify that the release is deployed only as far as the QA environment and not to the production environment.
- There may be queuing policies defined for an environment, which specify which of multiple deployments will occur, or the order in which releases are deployed.
- There may be pre-deployment approvers defined for an environment, and the deployment will not occur until all necessary approvals have been granted.
- Approvers may defer the release to an environment until a specified date and time.

Understanding application release and deployment challenges

Existing manual and semi-automated approaches result in uncoordinated, time-consuming, error-prone and non-scalable release mechanisms draining both effort and budgets. The typical problems faced by enterprise release and deployment management functions are:

High volume and frequency of releases

Typical banks have on an average more than 1000 changes, production issues and enhancements to be done and moved to production. Assuming an average of 4 environments and number changes/fixes/enhancements are delivered incrementally, the number of promotions is easily closer to a 5 digit number.

Wide technology stack coupled with heterogeneous environments

Added to these is the wide technology stack- horizontally and vertically. i.e. each application spans across multiple technologies. Portfolio of applications is spread across multiple technologies.

Expert Overload

Due to the multiple technologies involved, Manual (or partially manual) deployment processes remain dependent on experts with specific application knowledge and technologies. Crucial deployment knowledge is in the heads of people, not stored and automated in systems.

Excessive costs and delays

Manual deployments involving multiple experts across multiple environments is time consuming. This translates to higher costs and schedule delays.

Outages

According to a 2010 Gartner, Inc. Research report, “through 2015, 80% of mission critical outages will be caused by people and process issues, and more than 50% of those outages are caused by change/configuration/release integration and handoff issues.

Do no harm! …and don’t break the working production environment with new things that work!

Point of View

Bulky deployment manuals or Custom Scripts

Manual deployments are guided by bulky manuals and contain endless sequences of deployment steps. It is extremely difficult to keep these manuals up-to-date each time a change occurs in the physical infrastructure, middleware or the application itself. Custom scripts created to automate the deployments steps are also tedious and need to be updated when there is a change.

Non-orchestrated tool sets

Release and Deployment activities are managed using a plethora of tools. These are not orchestrated to provide a seamless end to end release and deployment process

Lack of visibility

Organizations are unable to trace whether contents of the package being promoted to various environments are the baseline contents (e.g. whether the UAT certified build is the same one that is deployed into Production)

Release Management Quality

While the activities to create an artifact are used for both processes, which is encouraged for efficiency purposes, the utilization of that artifact can be different for each process; an example is the use of a quality assurance.

Quality assurance determines if defects exist in the construction of the application or infrastructure component for the purpose of correcting the defect before going into production. While this same information is important for release management, the same information can be used to identify potential points of failure that could occur once the release is implemented into production. The difference is that project management uses the information in a diagnostic manner that will only be used during the project lifecycle, where release management will use the information in a proactive manner and will use it throughout the life of the service until the bugs are corrected or until the service is retired.

These types of similarities exist throughout the development and project lifecycle. It should be remembered that the key difference is how the information and artifacts will be used—project management will only use the information and artifacts for the length of the project and release management will use it for the life of the service. The reusability of the information and artifacts is a benefit of release management. Reusability reduces development time and the cost of future releases simply because development teams do not need to recreate the artifact from scratch to understand the existing version of the release. When a new release is created, the development team simply refers to release notes and documentation from the previous release.

In the same way that there are activities, tasks, and artifacts that are created and used for both project management and release management, there are also activities, tasks, and artifacts created for one process that complement the completion of other artifacts for the other process. An example of a complementary task is the creation of the business requirements document (BRD), which is a project management task. The BRD feeds the creation of service level requirements, the service offering, and finally results in the creation of a service level agreement between the customer and IT operations. The service offering is also a critical component for creating the organization's service catalog. In this example, the BRD is not a requirement of the release lifecycle, however the service level agreement is, and without understanding the requirements and expectations of the customer, the service cannot be built and operated to the customer's needs and expectations.
The release and project management processes have similar and complementary tasks, activities, and artifacts, however there are some activities of each process that are not valued by the other. It is not that these activities and artifacts are not value added; it is simply that they do not have relevance to the goals and objectives of the related process. It was discussed that the BRD is not part of release deliverables; however, it is a key activity and deliverable of the project process. A support and escalation document is not considered an added value to the project process, however it is essential to the release process since

the release process is concerned with the quality operational delivery of the service through retirement.

Release management Processes & Automation

Release management involves five major processes necessary to successfully plan and deploy authorized releases into an IT infrastructure. They are:

1. Release Planning
2. Release Building
3. Acceptance Testing
4. Release Preparation
5. Release Deployment

1. Release Planning - The first step in the release process is the creation of a plan identifying the activities and the resources required to successfully deploy a release into the production environment. This process includes the identification of scope and content of an approved change, and the release requirements for successful deployment, performing a risk assessment for the release and gaining signoff from the appropriate groups, prioritizing, planning, and scheduling release activities, establishing a suitable team for the release if required, liaising with experts and interested parties to determine the required resources and strategy for the release & finally documenting and tracking all release planning activities.

2. Release Building - Once the release plan is in place & agreed on, it is required to identify and develop the processes, tools, and technologies required to deploy the release into production. This process flow leads to the creation of a release package containing all of the components necessary to deploy the release. It includes selection of a suitable release mechanism that is a strategic fit, is repeatable, and is consistent, designing and building a release package that allows it to be successfully deployed, testing of the release package to deliver the change effectively in line with requirements & ensuring the release package is updated to the CMDB.

3. Acceptance Testing - Acceptance testing allows QA testers and business representatives to see how the release and release package perform together in an environment that closely mirrors production. This process includes designing and building an accurate test environment that models the conditions in production, performing key functionality user acceptance tests aligned to the requirements of the change and release to ensure confidence in the release & evaluating acceptance testing results to make a confident decision to move toward release preparation.

4. Release Preparation - After the completion of the acceptance testing, the next step is to prepare the production environment for the release, move through the preparation process and agree on the action to be taken. This process ensures that adequate resources are available for deployment of the release, effective communication pertaining to the release is carried out & all the training and user activity needed for deployment have been completed. It also confirms the suitability of implementation plans and readiness of the production environment for receiving the release, reviews the preparation and suitability of the release for deployment into the production environment, ensures all related changes have been handled by the change management process & manages the discussion around the Release Readiness Review inputs.

5. Release Deployment - The process of deploying the release into the production environment depends on the type and nature of the release and on the selected release mechanism. It tracks through the implementation plan to carry out the actual deployment procedure & reviews the deployed release, taking into account feedback and comments from all parties involved. If the release fails to meet expectations or if serious problems are encountered during deployment, problem management may need to help identify and diagnose the root cause of the problem. If a suitable fix or workaround can be found, this should be documented and an RFC created to deploy it into production. The RFC should also ensure that the fix and any supporting documentation are added to the release package. Ultimately, the feedback is provided back to the change management & the release is completed following the successful deployment and completion of the review.

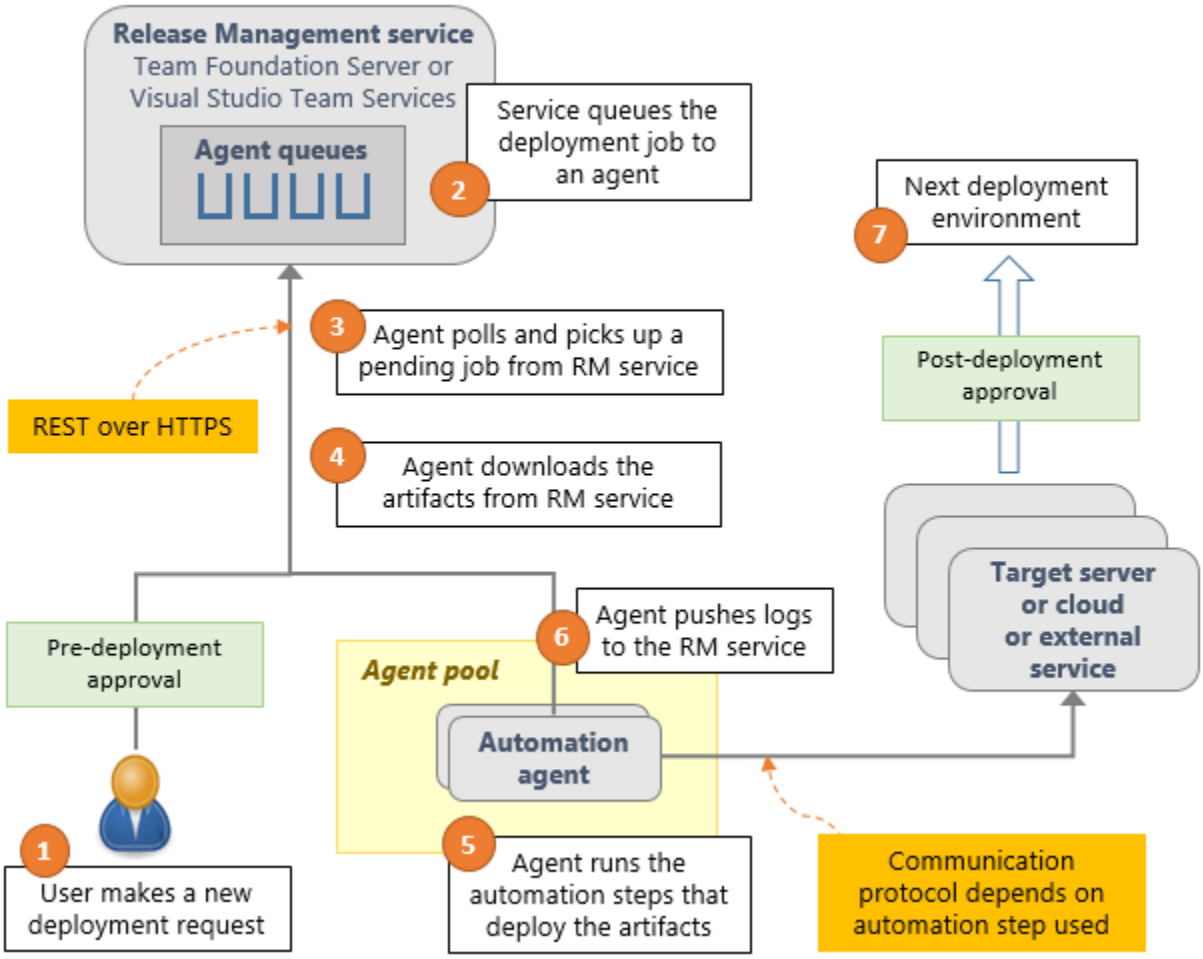
Release Management Automation

Application release and deployment automation is the automation of the release planning, control and entire deployment process from build all the way to production. A Release Automation platform is a toolset used to enable a more accurate, reliable and accelerated release and deployment experience for business and to offer an overall simplification of the complex release management process.

- The automation platform typically covers
  - Planning of release, dependencies and schedules
  - Monitoring and control of environments, consistency and access
  - Automation of deployment, rollback and configuration

Let us see how an automation platform changes the way release and deployment happens.

The Release Management service stores the data about your release definitions, environments, tasks, releases, and deployments on real time environments.



Release Management runs the following steps as part of every deployment:

Pre-deployment approval: When a new deployment request is triggered, Release Management checks whether a pre-deployment approval is required before deploying a release to an environment. If it is required, it sends out email notifications to the appropriate approvers.

Queue deployment job: Release Management schedules the deployment job on an available automation agent. An agent is a piece of software that is capable of running tasks in the deployment.

Agent selection: An automation agent picks up the job. The agents for Release Management are exactly the same as those that run your Builds in Team Services and Team Foundation Server. A release definition can contain settings to select an appropriate agent at runtime.

Run the deployment tasks: The agent then runs all the tasks in the deployment job to deploy the app to the target servers for an environment.

Generate progress logs: The agent creates detailed logs for each step while running the deployment, and pushes these logs back to the Server.

Post-deployment approval: When deployment to an environment is complete, Release Management checks if there is a post-deployment approval required for that environment. If no approval is required, or upon completion of a required approval, Release Management proceeds to trigger deployment to the next environment.

In manual release management, Planning is done using excel sheets. It is very difficult to track the multitude of tasks to be completed for each release, their dependencies. Also it is very difficult to map to the environments that will be used and any potential conflicts.

The automation platform enables planning out the release calendar providing a holistic view of all upcoming releases.

The various tasks under each release can be scheduled and the dependencies mapped. The dependencies between releases can be mapped to enable identification and resolution of release conflicts

Test environments required for a particular release can be reserved. This enables a view of what is happening on each environment at any point of time. Configuration parameters for the particular release are defined; this provides the flexibility to customize the deployment according to that particular release context. The process flow can be customized based on release types

Release Control

In manual release and deployment activities multiple personnel need to have access to environments, component storage locations. This spawns a number of issues related to environment inconsistencies, security and compliance.

Tracking the reason for a release failure is a time consuming activity. The dynamic schedule changes to various releases also create potential resource conflicts which are difficult to detect.

The release control aspect of the automation platform enables tracking the progress of releases and identification of both process and application issues. The platform identifies issues such as, a release package not having an associated application version or configuration files required to guide the roll out, also alerts are provided for rollout issues, resource conflicts if environments are used for something else.

The platform provides a clear audit trail mechanism of who approved a particular release, when was the release promoted to a particular environment, issues if any, roll back status etc making compliance a breeze.

Automation Deployment

When software developers decide their build package is ready, it is automatically passed to the solution. The build package then moves through a pre-defined software development lifecycle, complete with all the

compliance-savvy functions like automated process, workflows, notifications and approvals. During the entire process, release and deployment teams can easily track changes and provide the necessary compliance reports. From the time the build is assembled and imported into the system, the solution tracks, manages, and deploys the application through automated workflows and best practices.

The core aspect of the release and deployment automation is the Deployment model. The majority of the platforms enable definition of a deployment model which consists of three distinct sub models.

The first sub model is the logical application model, which defines “What” needs to be deployed.

The second sub model is the Environment model, which defines “Where” the release component(s) need to be deployed.

The third sub model is the Work flow execution model, which defines “How”, i.e the order in which the release components (s) need to be deployed.

Application Model

The logical application model is not about packaging of physical binaries, it is the logical packaging. The logical model captures what are the different types of components that comprise the release and their mapping to the specific container/server type. This model captures -

- Dependencies on middle ware, database or previous version of the same applications and how they are related to each other.
- Configuration status, i.e. the versions of the application, where the binaries are organized, which versions are sunset and the sequence of patches or upgrades that have to be installed to get to the current version.
- Inventory list of components and the versions of the component(s) that make up a particular release.

Environment Model

Enterprises have different types of environments such as Dev, SIT, UAT, Pre-Prod and Productions. The details of these environments along with their setting such as the number of different servers, interfaces etc. are captured as part of the environment model. These need to be modeled so that it can affect the work flow as it is executed.

Profiles for deployment of components to each container/server type example database server, app server etc. are also captured as part of the environment model.

Workflow Model

As part of the application and environment models, components of particular applications along with their various environments are identified. Most of the enterprises have different release types, for example customization release, support release, non-functional and emergency releases with each release type having its own route to live. The work flow execution model defines the sequence of release activities by each release type.

The work flow can be easily defined using pre-defined standardized business process functions with a drag and drop interface. For each of these steps a threshold for success/failure along with recovery steps for failure conditions can be defined. You have options to set breaks at each of these steps to pause the deployment and examine the logs.

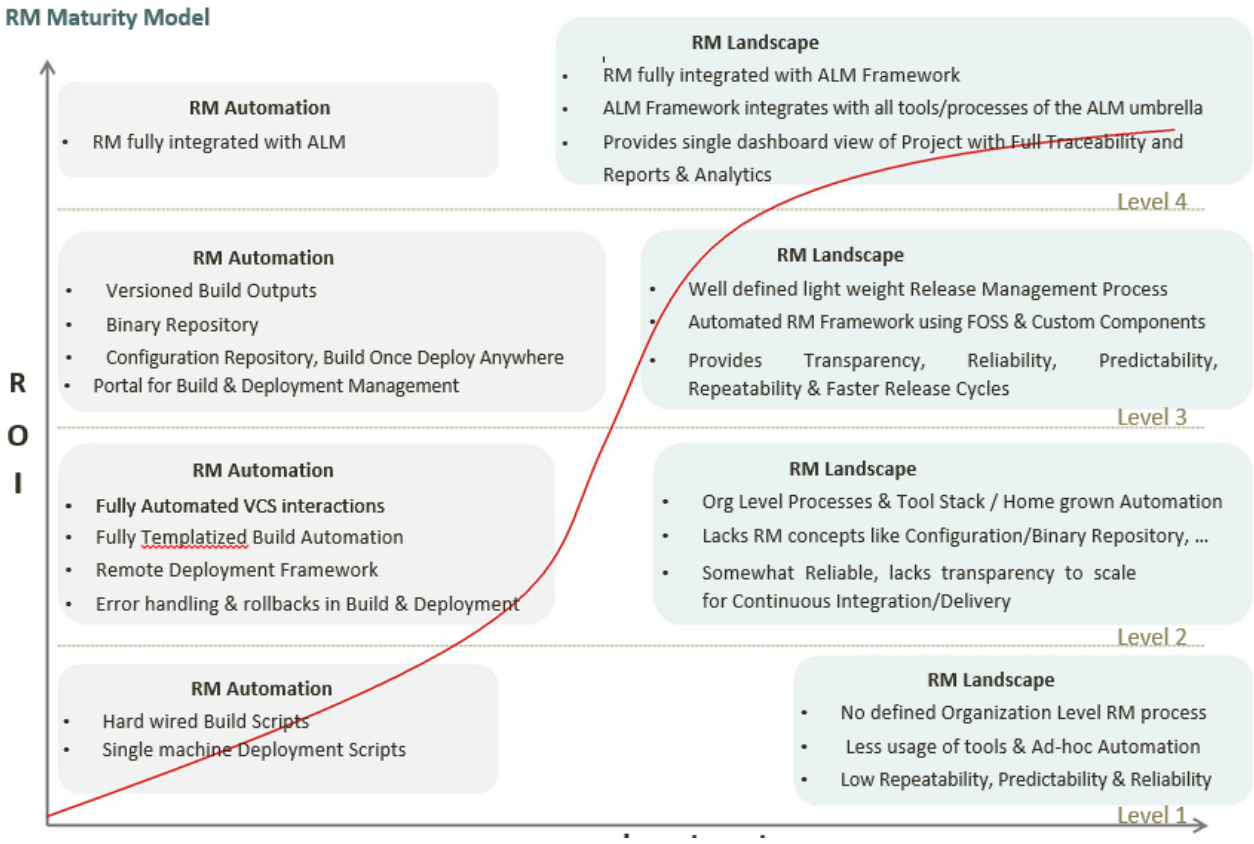
The workflow execution model uses the data in application and environment models to repeat the required activities without the need for complex if-then-else logic. The platform is intelligent to repeat steps based on the number of servers, skip a server update if the release does not impact the server.

To sum it up, this model enables encapsulating the entire set of deployment data and facilitates a repeatable, scalable and secure deployment process

Building a Release Management LIFECYLE

1. Change Management	5. Deployment Management
- Bug Fixes	- Deployment Request Portal

- New Enhancements	- Environment Management
- Project Tracking	- Notifications
2. Version Control Management	6. Environment Provisioning
- Branching & Merging	- Bare Metal Provisioning
- Labeling/Tagging	- System Configuration
- Versioning	- Application Sanity Checks
3. Build Management	7. Configuration Repository
- Build Request Portal	- Repository Structure
- Continuous Integration	- Environment specific data
- Dependency Management	8. Deployment Framework
4. Binary Repository	- Remote Deployments
- Repository Structure	- Orchestration
- External Libraries Integration - Error Handling	- Reusable Deployment Scripts
- High Availability	



Version Control Best Practices

Check-In

Define a periodic check-in (logical chunks) and refresh policy for your project. Check-in can be as frequent as hourly to daily or completely left to developers as in distributed VCS. Project complexity, quantum of integration effort, team co-location are some critical factors to be considered while taking decision.

Do not check-in binaries files into VCS. VCS should be used for files which get updated, strictly speaking an update to a binary results in a new version (i.e. binaries should be treated read only)

Branching & Merging

- Branch only if unavoidable and branch late
- Do not create separate branches for individual developers. VCS is not a backup server.
- Do not create separate branches for SDLC phases (DEV, ST, SIT, UAT, …). A change to an application results in a new version and a version should be deployable to any environment. A change should not be made for a particular environment.
- Define policies for syncing branches. Changes from mainline/trunk to other branches should flow continually, branch to mainline for production releases and child branches to parent branches based on situation. Project complexity, quantum of integration effort, team co-location are some critical factors to be considered while taking decision.

Tagging & Versioning

- Label/Tag logical points from which builds are taken for releases
- Use a common project structure and naming convention

Build Management Best Practices

General

- A build should always produce a versioned binary package
- Check in build scripts into version control system
- An output of a build run should be automatically placed in the binary repository as per the structure
- Build Requests should be captured for audit and other uses. Use a web portal as a build management tool.
- Schedule periodic automated builds with the help of CI tools

Modularize

- Break down your project into components and build only changed components & their dependencies
- In the J2ee world, use Maven/Ivy to manage component dependencies
- Break down your build script into reusable templatized components. Do not hard code.

Binary Repository Best Practices

- Should have a defined structure & naming convention
- High Availability (e.g. SAN clusters with redundancy)
- One Binary Repository for a company/unit, can be geographically clustered with replication for performance
- Should support promotion of binaries to SDLC environments (DEV/ST/UAT/…)
- Should have automatic redundancy detection and avoidance
- Can be realized using products like Archiva, Artifactory or a custom solution on top of the file system
- Should only be accessible to the deployment frameworks based on right ACL

- Should only support file creation, update/delete should not be provided

Configuration Repository Best Practices

- Should have a defined structure & naming convention and match that of a binary repository
- Should be realized on top of a version control system
- One Configuration Repository for a company/unit, can be geographically clustered with replication for performance
- Should only be accessible to the deployment frameworks based on right ACL

Deployment Management & Deployment Framework Best Practices

- Deployment Management
    - Deployment Requests should be captured for audit and other uses. Use a web portal as a build management tool.
    - Capture the revision/version number of the configuration repository at the time of deployment request
    - Deployment portal should present the deployment options after linking the Binary & Configuration Repository

Policy to be adopted as part of best practices

- All projects, bundled changes to an existing service (releases containing multiple enhancements/fixes) and cyclical changes to an existing service must go through the Release Management process and must have a completed request for change (RFC) with appropriate approvals.
- Whenever possible, changes to an existing service should be bundled together and released on a regular (e.g., monthly) basis using the Release Management process.
- A single “Release Engineer” must be identified for every Release. The Release Engineer will be responsible for successful coordination and execution of the Release, as well as ensuring all required documentation related to the Release exists.
- Proof that controls (initiation, testing, and approval) have been followed for all auditable Releases shall be stored with the ability to be reproduced.
- Each Release should be initiated through a standardized and approved process (service request, incident management, problem management).
- Each Release should be well tested and verified prior to implementation.
- All implementation work on the Release should be completed by the Planned End Date/Time.

  Validation that the Release has been completed successfully should be confirmed through post-release testing

- Release – Captures overall detail. Release Engineer role req.
- Release Item - Details the separate pieces of work within the Release. Release Engineer role

required.

Release Task – Details work required to deploy each Release. Release Engineer or Release User roles required.

Ensuring an effective Automation Platform

An effective application release automation platform should be able to:

Version Control Integration

Internal as well as Vendor development teams use different version control tools like Subversion, CVS, and Microsoft Visual SourceSafe. The automation platform should preferably have capabilities to pick up baseline code from any of these version control tools and initiate the build processes. This enables vendor development teams to continue using their preferred version control tools while release automation toolset handles moving the build package through the release lifecycle.

Automation of all movement along the path to production

The automation platform should automate From the time the build is created to final deployment, the promotion/ hand-offs across users, applications and environments should be automated. This operation should be completely configurable using simple point-and –click functions to meet the individual business requirements that the organization has established.

Point and click distribution and deployment

For companies managing multi-platform, distributed environments, the solution should automatically deploy all necessary components to the appropriate target location throughout the development lifecycle. It should gather, package, distribute and install application components at each stage of the lifecycle.

Total Visibility of Objects and Package contents

From the minute a build is created, ability to trace all information associated with a valid package from a revision number to a tag or label, as it moves through the entire development lifecycle. This includes who approved, who touched it, what happened to it, where it went next, etc.

Deployment rollback features

Rollback recovery options are absolutely necessary. When it comes to application development, deployment is often the stage where disaster will strike. Rollback features allow users to simply label the last state of the application pre-deployment before moving into production.

Parallel development and conflict resolutions

The simultaneous development of multiple versions of software application can be complex and difficult to manage.

All too often, development teams are faced with conflicts between versions, which can confuse and disrupt even the best managed project teams. The right solution should allow quick identification of version conflicts using descriptive and structured status tags (active, pending, cleared etc..) so that developers can clearly see and resolve existing conflicts at the appropriate time.

Secure and consistent distribution packages

The right solution should eliminate the security concerns and inconsistencies of manual file transfers, coordinate arrival regardless of location or system(windows, Linux, IBM i), log all code and content transfers for audit purposes and limit access to objects by user.

Workflow and change management

The solution should automate, accelerate and enforce workflow and change management process.

Software Configuration management

The solution should include functions that automatically build organize and maintain a central inventory of all application components.

Release merging capabilities

The right solution should allow seamless merging of releases without losing version and historical information in the merged release. Companies with scheduled release and tight controls, software houses who want to consolidate releases without losing the object's identifying mark(the version number'), and financial institutions with strict auditing procedures should benefit from the "merge to parent" functionality .

Flexibility

The solution selected should be both open and flexible in nature with integration efforts, tool options and adoption of process methodologies. This is essential in enforcing process across release management. It should plug in to variety of tools so users can pursuer best-of-breed strategy.

Simple Version Control Integration

Release management process also need to support simple version control tools like Subversion, CVS, Microsoft Visual SourceSafe and Microsoft team foundation. Most of them are great for keeping track of all the various version of file, they work fast and developers love them. The solution allows software teams to keep using their own version control tools while it handles moving the build package through a pre-defined software development lifecycle.

Benefits of automation release

The Benefits enterprises achieve by automation release and deployment are:

- Shorten application release and service times, for both routine and non-routine tasks across the application portfolio.

- Provide centralized control and automatic execution of application release tasks such as rollouts, patches, hot fixes and rollbacks.
- Streamline and coordinate processes across users, applications and environments (Dev, QA, Ops).
- Support automation across heterogeneous infrastructures, including physical, virtual and cloud environments.
- Scale application service workload capacities.
- Provide granular audits and application service reports.
- Provide a sophisticated and comprehensive dashboard of release trends, enabling high-level IT managers to monitor and audit the deployment process.
- Enable seamless integration with existing automation and monitoring tools