MidVision

OVERCOMING THE PITFALLS OF WORKFLOW BASED DEPLOYMENT AUTOMATION

A MidVision eBook

Workflows have historically been used to perform routine system administration activities - such as running batch jobs, housekeeping activities, and executing scripts in specific orders. A workflow-based automation allows the creation and visualisation of sets of repeatable activities, and more recently tools have evolved to deploy applications using workflows.

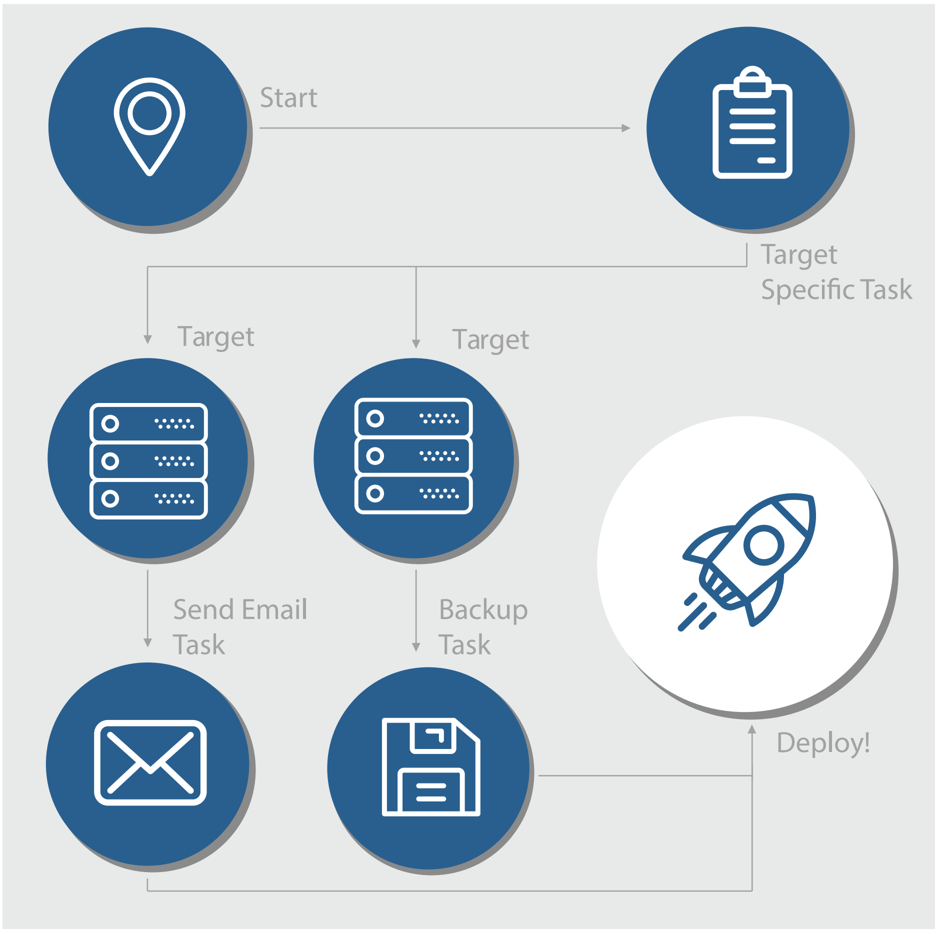
Recently, we've seen some articles discussing workflows for deployment automation, specifically the problems with using workflows for defining standardised, simple, repeatable and scaleable deployments. This approach has the advantage of being quick and easy to implement - for instance, there is no requirement to learn relatively complex script-based frameworks - but it comes with some challenges that can manifest themselves as you increase complexity and the scale of your deployment processes.

This eBook will explore some of these challenges which may crop up when using workflow-based deployment automation, and discusses the various ways that they can be overcome.

Workflows allow the user to visually model the deployment tasks that will run during a deployment. These tasks could range from the lowest level functions such as creating files, folders, users and groups, to the higher level tasks that might, as a single task, install an application into an application server with all of its attendant configuration.

Workflows remove the need for difficult-to-maintain code and allow the enterprise to move away from dependency on specialists in particular scripting languages. A clear visual overview of all processes means that failures are easy to detect, pinpoint and fix quickly. A mixture of low and higher level tasks allows the user to pick the correct task for the job, with the minimum of steps but with the flexibility to perform low level tasks where necessary, reducing the need for manual steps.

So what are we trying to achieve with our workflow? Ultimately we should be able to model the deployment of many different components, each at different versions, through multiple different target environments (targets) on our route to production. For example, an internet banking deployment might involve many different components, such as database updates, web server files and configuration, application server compiled code and configuration, messaging services etc. These will need to be tested in many environments, such as development, integration test, UAT, OAT, Pre-production and finally production. As we move through the environments we might go from one physical instance of each component in development through to production where scaleability and resiliency requirements mean we have multiple instances of each component. All of this should be handled in the workflow.



It is easy to see that creating a single workflow to model all of the above, in a single diagram, would be potentially huge, unwieldy, overcomplicated, near impossible to maintain and require constant updates. Creating new workflows to model further deployment scenarios would involve repeating a large number of the steps of previous workflows, which is time consuming and error prone. So how can we make the best use of workflows without the pitfalls?

TASK FLOWS (SERVER ORCHESTRATION)

Here at MidVision, we've given this problem a lot of thought. We believe that a deployment tool should start by supporting small pieces of workflow that encapsulate all the tasks required to deploy or manage a single component on a target server (or group of servers in a cluster). We call this a task flow or orchestration. This orchestration should be target neutral, capable of being run on every target where this component will be deployed. We should aim to make the orchestration declarative, such that the target is always brought to the desired state. Careful design of tasks as well as the flexibility of workflow will enable us to achieve this. Of course, we'll need to introduce target specificity into this workflow, and this can be achieved by injecting target specific parameters at deployment time. Such settings as database passwords, port values, heap sizes or anything defined in the orchestration should be capable of being injected, by the use of variables in the workflow. We call these the data dictionary parameters. The data dictionary should be settable globally but be capable of being overridden at target scopes, and together with the orchestration definition and payload (internal and external resources) we define this as a blueprint.

TARGET SPECIFIC AND ERROR BRANCHES

Of course there will be some task steps that only run on some server targets (for example additional checks against change management tools in production, or perhaps some testing requiring a firewall to be stopped in development). The deployment tool should allow you to model your orchestration to have a main stream with tasks running on all servers, but occasionally branch off to perform 'Target Specific' tasks for those occasional target task differences we all know exist. In this way your task flow is a common pattern you use over and over again in a fully repeatable way. The task flow should also allow multiple failure branches to allow different cleanup or rollback scenarios to be invoked depending on the task that fails.

CONDITIONALS AND SCRIPTS

Support for simple conditionals and loops in the task flow enables the user to avoid writing code in scripts, in most cases. If you do need to write scripted sections, you should be able to easily call the scripts, passing variables and command line arguments, which can also be environment specific data dictionary values. It is even better if you can enter the script code directly into an orchestration task, which removes the need of calling the actual script.

INTERRUPTING TASK FLOW AND LINKED DEPLOYMENTS

So having a task flow that runs on any server will have some drawbacks. Needing to perform operations on one server part way through a deployment on a different server is a particular concern. For example we might need to update a database between stopping and starting an application. To obviate this situation, it should be possible in the deployment tool to allow the task flow orchestration running on a server to call back, via a web service, to the framework server to call another orchestration to run on the same or a different server, either synchronously or asynchronously. We call this a linked deployment. In the case above, we'd make the synchronous linked deployment call to the database job between the stop and start application tasks, waiting for it to return before starting the application. Alternatively, if t's a manual task, you could add a 'Manual Step' task to email subscribers to let them know to perform the manual step before restarting the paused deployment through the deployment tool console.

COMBINING ORCHESTRATION/TARGET COMBINATIONS

Now we have a number of server orchestrations with the available targets to run on, we can combine these into our higher level workflow or pipeline. A pipeline is a workflow where we choose to design the order of orchestration/target combinations, including scheduling, parallelism, promotions and approvals. Some tasks will mirror those at the server orchestration scope, such as Rest callouts to change management tools, but mostly they will not. For example we might have a database orchestration with development, test and production targets, together with application server and web server orchestrations running over similar targets. These could all be ordered into a single pipeline deploying everything, as well as in separate pipelines to deploy just the database, application or web-server changes alone, but reusing the same set of orchestration/targets.

So instead of a single large workflow, we now have a pipeline managing a number of orchestration/target combinations. Management now becomes rather simple. A change to the task flow will make the change in all targets with no changes needed in the pipeline. Adding a new target is simply a matter of copying (cloning) a set of data dictionary (injected) values and modifying those that are different in the new environment. Then just add the new orchestration/target to the correct location in the pipeline.

VERSIONING AND COMPOSITE RELEASES

Up until this point, we haven't considered versioning. Every time we make a change to an orchestration or set of target data dictionary values, we should be able to version this in a zipped deployment package, which is a point in time representation of the blueprint. When we create our pipeline, not only do we select an orchestration/target combination, but also a version. All targets for a specific orchestration in the pipeline will be at the same version. It should be possible to copy pipelines and change the versions of the components, to create multiple composite release pipelines. We might also want to be able to specify pipelines with the latest versions of some or all components, or indeed new versions of some or all components, where triggering the pipeline will automatically create new versions of the underlying components prior to deployment.

IN CONCLUSION

Whilst the benefits of visibility and ease of use of graphical workflows are clear, they can quickly become large and difficult to manage for the deployment of complex environments with many components. By breaking down the workflows into management pipelines of combinations of target neutral server orchestrations with injected environment differences, management can be greatly simplified because task flows can be reused across multiple targets and pipelines, without the need to duplicate changes.

MidVision