MidVision

9 DEPLOYMENT BEST PRACTICES

A MidVision eBook

INTRODUCTION

It's quite likely that at some point you've had a conversation about the importance of automating your deployments. The benefits of this in terms of organizational agility and efficiency should be clear, but very few companies have fully automated their entire deployment process. There will often be a certain level of automation, whether it is Continuous Integration, use of a configuration management tool to enforce desired state, or something more advanced. While there are a plethora of tools available to help you to achieve Continuous Delivery, processes and the mindset of your staff can be just as important in achieving a full Continuous Deployment IT ecosystem. Therefore, before you jump in and begin looking at which automation tools you want to go for, it's probably worth taking a closer look at the IT processes which you'll be wanting to achieve with them, so that you can find the best fit for the particular issues your business faces. MidVision have outlined 9 of the key best practices that should help you to achieve seamless Continuous Delivery on an enterprise scale, and hopefully they should provide a thought-provoking starting point for your journey further into the world of deployment automation.

1. DESIRED STATE

We'll start by assuming that your eventual goal here is an agile Continuous Delivery and Devops deployment pipeline. The first thing you'll need is a definition of how your desired state should look. The desired state is a detailed description of the ideal configuration of either an application of a larger system e.g. middleware or virtual machine, including all the dependencies it relies on.

Desired state can exist in the past, as a previous state that was achieved at some point, in the present as a state that is currently being achieved, or in the future as a state that you are looking to achieve. Ideally there shouldn't be any divergence between what's described in your ideal desired state and what's actually applied in your systems, thought this can be difficult without the correct processes and tooling. Ideally you want to be able to automatically enforce your desired state, with data-driven processes that are reliable and repeatable. This will ensure that desired state is achieved consistently every time, with the minimum user input each time – the goal of automation is to reduce the necessity of human interference as much as possible. You should also be able to put this desired state into source control so that you can easily manage and changes that are made to the desired state, efficiently and simply.

2. PROVISIONING AUTOMATION

The next two sections cover two important aspects of how we provision our initial infrastructure. Traditionally provisioning has been seen as a one-off activity, where we provision a set of servers for development, test or production usage, and they stay relatively static over time. More and more, however, modern usage patterns are necessitating that developers are able to provision as they decide that they want to test their code, or that testers are able to create additional infrastructure on demand when they need it. This will allow engineers to exponentially increase the speed of their workflows, enabling competitive business agility in an IT world where this is increasingly becoming the norm.

The first section is going to be about the provisioning aspect of this. For the purpose of simplicity we're going to class provisioning as creating operating systems automatically on private or cloud server infrastructure, installing any components or updates if needed and creating the initial configuration.

Now, as modern consumption patterns arise that are requiring us to provision infrastructure on demand - for activities like autoscaling - this means that the provisioning process has to be fully automated as part of this. It is possible for you to fully automate the entire process of creating operating systems on servers (creating the instance), as well as making a call to your middleware, your platform, and putting in the base configuration.
As your applications move from development to production, slow provisioning can be a bottleneck that can cause significant delays in the software release lifecycle.

Being able to provision an environment isn't much help if you can't also automate the configuration that goes on top of it, however. To improve your efficiency in provisioning, you can use a configuration management capability. Alternatively, you can employ a deployment aid with configuration management capabilities to leverage your existing configuration tools and consolidate your workflows in one package. This will enable you to automate the definition, creation and scaling of your environments via one simple interface, as well as creating middleware, then patching it, etc. This should also integrate with your toolchain, meaning that tools are automatically updated as environments are instantiated or deprovisioned.

3. DEPLOYMENT AUTOMATION

Once you've automated the provisioning of your environments, the next step is to speed up the delivery of applications. You'll be aware of the problems with manual deployment processes that use deployment plan spreadsheets – they can be slow, error prone and waste important resources. This will in turn increase the length of time you're in the work in progress phase, and impede operational agility and competitive efficacy. With a deployment capability, however, you can instantly deploy, upgrade and rollback complex applications with the click of a button. This will enable you to reduce repetitive workloads, free up resources and reduce errors. It will also allow you to provide deployment capabilities to the entire team, improving everyone's ability to work independently while also speeding up collaboration. It is worth considering whether you will be using it in agentless systems, which have a central authority that is responsible for scanning all the machines in the enterprise, and initiating actions on them, or on systems with agents – small pieces of software that work with network management software to collect information from and take action on managed devices.

4. APPLICATION AND ENVIRONMENT VERSIONING

Now you've defined a desired state, and begun to get to grips with provisioning and deployment automation, you can now use your automation capability to achieve desired state by implementing version control. You should use source control to not only store all changes and updates to source code – but also for configurations of applications and environments. Version control will allow you to view every change that is made to your software and configurations, along with when it was made, who made it and why, making it far easier to track down the sources of issues, and in turn maintain a healthy IT ecosystem.

Version control is an essential part of any distributed collaborative project, and can give full visibility of the entire process to all members of the team, not just some, making troubleshooting far faster.

The ability to track and roll back changes instantly will also enable you to enormously improve operational efficiency and reduce the effects of errors. It will also make it much easier for you to directly compare and keep track of different required versions of software and environments. Version control can also be very useful for ensuring the maintenance of regulatory compliance - all changes are tracked so any changes that aren't compliant can be immediately spotted and rectified.

5. SELF SERVICE

The primary effect of self-service provisioning and deployment is that it will enable staff to take control of the infrastructure they use. This will mean they won't have to wait for operations engineers to provision and configure their software or environments, eliminating bottlenecks and all other constraints of manual provisioning and configuration by a small group of operations staff, such as the inconsistencies and mistakes that can be caused by poor communication and human error.

Instituting self-service IT infrastructure will result in improved agility and efficiency, and should substantially reduce resource usage. Giving all teams the ability to provision and deploy configured environments and applications will also improve individual initiative, as teams become able to achieve things in the manner and timeframe they want, rather than having to follow formalised protocols to access infrastructure. This will in turn also give team members greater involvement with the wider project, leading to an increased sense of personal responsibility, as they concern themselves more with achieving project goals and less with completing a narrowly defined job function.

6. ROLE-BASED ACCESS CONTROLS

When you've set up self-service as part of your deployment process, it should confer a great many benefits, as explained in the previous section, but you need to have the appropriate access controls in place. If staff are given too much access to base environments it can lead to configuration drift creeping in as specific application configuration changes are made to generalized environments, and users forget or neglect to return the infrastructure to a prior state. This can waste time, as unpicking these configuration changes each time the environment is created in a different context can be a complex process. It can also lead to difficulties with security, as it becomes harder to track configuration changes.

Establishing privilege gates will mean that users can enjoy greater freedom to achieve their project goals, yet it is possible to maintain desired state to the extent that clean environments can be generated when demand occurs, without errors creeping in.

Role based security will also help enormously with maintaining compliance and conducting audits, as there will be clear boundaries defining what certain personnel can and cannot do, and there will be full logs detailing all changes, when they've been made and who has made them. With role-based access controls it is also far simpler to assign quality gates, so reviews and checks can occur as and when they become necessary, by specifically defined personnel.

7. DEV TOOLCHAIN INTEGRATION

When you're looking to implement a continuous delivery pipeline, you will find that any parts of the process which are not automated will become bottlenecks to agile code delivery and deployment, and a lack of integration with the development toolchain is no exception.
The actual process of moving software to servers has generally been only semi-automated, requiring developer coordination with system administrators. By fully automating the deployment to development servers, you save resources, and render the process far quicker, more reliable and repeatable. You should be able to integrate your deployment solution with development tools, whether they be open source tools like Git, SVN and Jenkins, or Commercial ones like Bamboo or TFS, and it will help to make the entire development process self-documenting.
This will allow you to move from continuous integration, where code is integrated regularly into a shared repository, to continuous delivery, where the integrated code is regularly released into the live environment, as you get closer to automating the entire deployment process.

8. FULL STACK DEPLOYMENT

Once you can integrate with your development, testing and operations tools, and your middleware, you can automate the build, provisioning and deployment of your entire software stack. This means that you can very quickly set up entire environments with running software whenever you need them, in the exact configuration you require.
If you aren't reprovisioning the operating system and all dependent services each time you deploy, it can create artifacts that can lead to production bugs that cannot be replicated in the production or test environment, making deployment a far riskier process.
When your entire stack is automated, you neatly sidestep this issue, maintaining a consistent environment across the entire release process, from development to deployment. You should get to a point where you are able to reprovision your runtime, whether that based on containers or virtual machines, and dependent services on each deployment. This will create a much cleaner and healthier software stack, free from old configuration artifacts that would come with working on previously used operating systems or environments.
When you're deploying to a cloud service such as AWS, this also makes it much easier and more reliable to autoscale your production systems in identical environments to handle fluctuations in demand.
A full stack approach will also allow you to build, provision and deploy not just middleware, but technologies such as Docker and Hadoop, pulling together small units of work into one consistent whole.

9. CONFIGURATION AND DRIFT MANAGEMENT

Once you're able to achieve all the above best practices, you should have reached the point where you're able to automate the achievement of desired state. The final thing to consider here is how you will maintain it.

Configuration and drift management will allow you to automate resource-intensive and error-prone configuration and compliance management processes within your IT systems. It will help you to discover devices, servers, operating systems, databases, middleware and applications, and create an inventory of their configuration settings. You will also be able to map components to dependent services to analyse the impact of changes that are made, and it will allow you to capture dependency information in snapshots, so you can establish configuration baselines and standards to maintain desired state, and continuously detect and track configuration changes which lead to drifts.

Administrators can then be notified of all potentially harmful configuration changes, and will have the option to ok them, or revert to a prior state.

This will hugely increase operational efficiency, and allow you to enforce all compliance policies with relevant rigour, reducing the likelihood of outages and helping you to maintain healthy software lifecycle management

© MidVision 2016