

Deploying to Prod (Through Automation), or Not to Prod? That is the Question!

by Seth - Friday, October 14, 2016

<http://www.sethgagnon.com/deploying-to-prod-through-automation-or-not-to-prod-that-is-the-question/>

This article focuses on code deployment practices and how you deploy your code to production. Many are believers that automation is the key all the way, while others want to control their production deployments through manual intervention. Which method do you follow?

Background

I often hear that an application practicing continuous delivery is not truly embracing the practice if the team is still delivering their code to production manually. This got me thinking, as it is an interesting question. Am I still following a continuous delivery methodology, if I deploy my application to production in a manual or semi-manual fashion? Let me first clarify what Continuous Delivery really is. I believe Martin Fowler's [definition](#) does a great job outlining the key components:

You're doing continuous delivery when: [\[1\]](#)

- Your software is deployable throughout its life cycle
- Your team prioritizes keeping the software deployable over working on new features
- Anybody can get fast, automated feedback on the production readiness of their systems any time somebody makes a change to them
- You can perform push-button deployments of any version of the software to any environment on demand

You achieve continuous delivery by continuously integrating the software done by the development team, building executables, and running automated tests on those executables to detect problems. Furthermore you push the executables into increasingly production-like environments to ensure the software will work in production. To do this you use a [DeploymentPipeline](#).

So, you need to be using a deployment pipeline (check out my previous post [here](#)) and practicing the bullets outlined above. We should note that we are talking about Continuous Delivery here, not Continuous Deployment. Here's my two cents on what each of these items mean.

Software is Deployable Throughout its Life Cycle

I believe the key takeaway here is that you have your master branch or trunk (depending on your SCM

tool of choice) always in a state that has stable code and it can be deployed at any point in time. Obviously, as your team evolves and you take on additional work, the code will be evolving as well. You want to ensure that you have a strong versioning policy in place and a strong workflow in place so that the code is always in a stable state in the master branch or your trunk.

Team Prioritizes Keeping the Code Deployable Over Working New Features

Similar to the topic mentioned above, the idea here is that your delivery team always ensures that the focus is on keeping your master in a working state and ensuring it can be deployed. When issues arise and the master becomes unstable, getting it back into a stable state needs to be the top priority. Likewise, when you are working on new features and functionality and your code breaks a build, you also want to ensure that gets resolved first, before continuing work on new user stories and new features. The bottom line is that the master is always stable and your aren't breaking the builds.

Anyone Can Get Fast, Automated Feedback on the Production Readiness

With the continuous integration tools available on the market today, it is very easy to know the state of your code and how “ready” it is to deploy to any of your environments. Developers need to know when they commit code that is breaking a build and they need to correct the issue sooner, rather than later. If you have a source code management system in place like [Git](#), then it becomes fairly easy to track and audit the commits that have been made and who made them. This helps the team see their audit trail and learn how their code changed over time. When you add in a continuous integration engine like [Jenkins](#) to the mix, developers can now see when builds are failing and why they are failing. They can refer back to their SCM (Git) and track down who made a change that broke the code and get it resolved fairly fast. Tools like Jenkins can send out automated emails to your team and inform them when the code is unstable, so they can jump right on it to get things resolved.

Push-Button Deployments of any Version of Software to any Environment on Demand

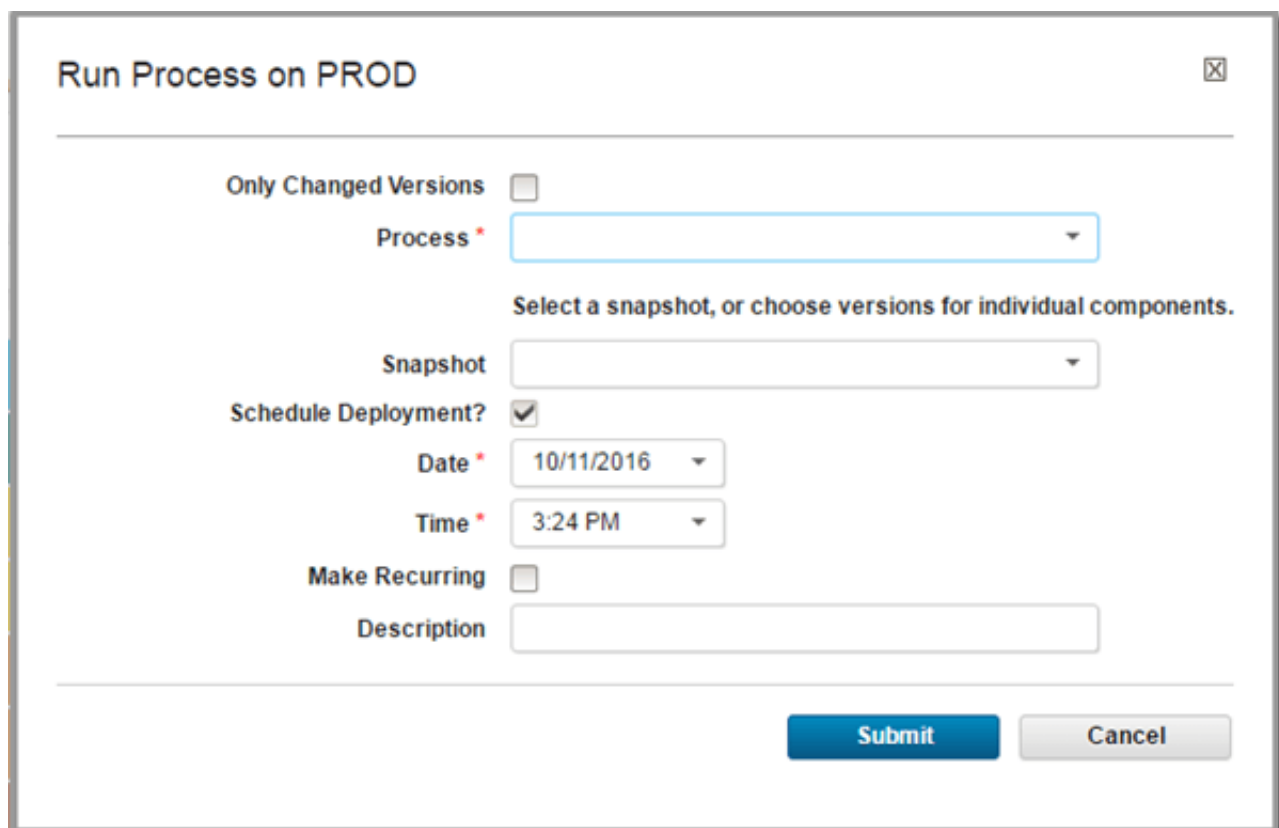
So, now that the above items have some definition to them, this is where the heart of discussion truly lies. There are many organizations out there that want to ride the automation train through all the environments, and you can't blame them right? Automation certainly can reduce errors, improve productivity, and increase speed to market. Of course, this is when the automation works properly. The automation will only work to the best of its ability based upon how you design it, and this all depends on the prep work that went into the automation. Think of when you are painting a room and the amount of prep work that is needed before you even start putting paint on the wall. I would say that from beginning to end, about 70% of the project work is all prep work. This same concept holds true with your automation for deploying code. You will need the following, at least, to get things started:

- Established SCM system and versioning policies
- Workflow for the team for interacting with the SCM (branching, etc)
- Continuous integration configured

- Automated unit testing established
- Automated deployments to your testing environments
- Automated smoke testing in your environments (to ensure successful deployment)
- And the list goes on.....

If you are confident that all the items above have been addressed and your pipeline is pretty mature, then I believe you can have the discussion around automating your deployments to production and possibly increases the frequency of your releases to the production environment. However, this can take a fairly long time to get to a mature state and some organizations may not ever get to a mature state. With that said, it is understandable why plenty of organizations out there have no issue with automation through their lower testing environments, but when it comes to deploying to production, they still want a person involved managing that deployment.

Now there are some tools out there that can help with this type of automation for deploying your code. [IBM UrbanCode](#) is a product designed for this very purpose. It allows you to orchestrate your deployment process through a series of steps and it can handle virtually all types of deployments. There are some additional features like approvals in the flow that will require a person or group of people to approve the request before the code can be deployed, or a scheduling feature that will allow a requester to kick off a process for deployment but not have the code get deployed until a specified date and time (as seen below).



The screenshot shows a dialog box titled "Run Process on PROD" with a close button in the top right corner. The dialog contains several configuration options:

- Only Changed Versions**: A checkbox that is currently unchecked.
- Process ***: A dropdown menu.
- Select a snapshot, or choose versions for individual components.**: A text label.
- Snapshot**: A dropdown menu.
- Schedule Deployment?**: A checkbox that is checked.
- Date ***: A date picker showing "10/11/2016".
- Time ***: A time picker showing "3:24 PM".
- Make Recurring**: A checkbox that is unchecked.
- Description**: A text input field.

At the bottom right of the dialog are two buttons: "Submit" (in blue) and "Cancel" (in grey).

Figure 1 – IBM UrbanCode Scheduling Feature

Conclusion

In summary, there are many options for deploying your code to your environments and there are always many reasons why the code is deployed the way it is being deployed. Some of those reasons can include organizational culture & politics, technology at play, lack of knowledge around deployment automation, etc. So, before you are quick to ask your colleagues “*Why aren’t we just deploying everything automatically, it would make our lives so much easier?*”, please remember these reasons and understand the state of the pipeline maturity for your organization.

References:

1. Continuous Delivery. Martin Fowler. <http://martinfowler.com/bliki/ContinuousDelivery.html>

PDF generated by Kalin's PDF Creation Station