The Art of Serving HPC Products to Business while they are still hot

Vadim Dyadechko

Disclaimer

The opinions expressed here are solely my own and do not express the views or opinions of my employer.

Larger than coding

This year's theme of the Collegeville Workshop is developer productivity; the theme suggests that the major s/w development expense is coding, which I would like to contest. While improving developers productivity is important, coding itself is just one part of the larger picture; there are just too many other aspect of s/w technology that constantly toll the budget and beg to be improved. I strongly believe that the bigger prize is in optimizing the entire s/w product pipeline/ecosystem. At the end, *exploring a larger configuration set always yields a more optimal result*.

The void

The pressure of (over)achievement makes many of us commit sins that contribute to mysterious and seemingly inevitable product release delays. It is not uncommon for a developer or a project manager to declare a certain feature to be implemented the moment it compiles with no errors in a local working copy, way before it is committed to the central repo, passed any review, quality control, or had a chance to be documented or tested in the production environment. This creates a poorly understood and managed void, the lag between the moment the feature was reported to be completed and the moment it actually becomes useful to the end users. This void has roots in a widespread misconception that the s/w is ready once the coding is completed, and putting it to work is unimportant trivial routine that somehow happens on its own. I would like to address this misconception and talk about little things that usually stay under the radar but distinguish a successful s/w shop from a sloppy one.

DevOps

The Development Operations (DevOps) is a popular concept that promotes close business engagement and high degree of automation in the s/w development pipeline. There are many DevOps definitions floating around, most saying very little about the actual process, but all unanimous on *the ultimate goal: delivering s/w features and bug-fixes to business ASAP*. In other words, the goal is the reduction of the lag mentioned above to days, hours, and possibly minutes. I would risk to offer yet another DevOps manifest that emphasizes the main driving forces behind it:

- *pre- and post-coding tasks are critical parts of s/w product pipeline,*
- *building, testing, and deployment are core responsibilities of the development team,*
- *all operations, configuration, and maintenance expenses come out of the development team budget.*

Everything else: automation, short delivery times, mindset on productivity – are just natural implications of the positive feedback loop created by consolidation of development and operations in one hands.

The formal transfer of responsibilities does not change anything unless it also grants freedom to operate, freedom to select the right tools, freedom to to control testing and production environments. When executed properly, the DevOps re-org prevents the diffusion of task ownership (nothing falls between the cracks) and encourages wide range of optimizations, now internal to the team:

- investment in quality code and systematic testing saves a lot of bugfixing time,
- custom automation helps to prevent otherwise inevitable human errors,
- freedom to select the right tools/environment/hardware for the job saves a lot of integration/porting/tuning time,
- power to control development and production environments greatly simplifies the configuration and integration tasks.

In a long run, all these improvements reshape your code into a robust product: maintainable, testable, deployable.

While many of DevOps task and activities fall into the category of IT operations, it is critical not to delegated DevOps outside the development team. The production pipeline has a lot of moving parts that must fit perfectly. Involvement of developers in daily operations has enormous positive impact on the quality/usability of the DevOps components; the best pieces of s/w were created for personal use: Unix, C, Perl, TeX, Git.

Personal touch

Our company has a rich history of proprietary HPC development. The PDE solvers developed at ExxonMobil demonstrate industry-leading performance and scalability and were featured in media and technical papers. Much less publicity is given to the streamlined and highly automated daily DevOps that made such achievements possible.

Every day the simulator development team releases multiple bugfixes and improvements, each passing the QC pipeline of hundreds of regression tests, and becoming available for downstream integration just 15 minutes after tagging. We maintain several versions of the simulator, build and test executables for multiple HPC systems. A large processing volume of operations like this cannot be sustained by a small team without investing time and effort into robust development and testing environment.

What started a decade ago as a collection of helper scripts has evolved into a comprehensive suite, a proprietary toolchain for automating QA/QC and deployment processes on HPC. This technology takes the burden of routine tasks off the engineers and scientists shoulders so that they can focus on solving the problems that are simply too hard for machines: making high fidelity predictions about the behavior of complex subsurface systems. The suite has short list of dependencies and does not require elevated privileges; much of its success is due to the conservative scope, focus on productivity and automation, and active support.

I will be happy to talk about the governing principles and the challenges of DevOps, as well as share my personal observation, experiences, and thoughts on running a busy HPC shop.