



Don't Install Software by Hand

Diomidis Spinellis

THE SETUP AND configuration of an IT system is a serious affair. It increasingly affects us developers mainly due to the proliferation and complexity of internet-facing systems. Fortunately, we can control and conquer this complexity by adopting IT-system configuration management tools.

Veni, Vidi, Vici

Start with the complexity of Internet-facing systems. In the past, most software applications were monolithic contraptions that could be installed in a basic standard environment. For enterprise and end-user applications, the environment would be the operating system; for embedded systems, it would be the underlying hardware. All we developers had to do was test the software's deployment procedures and manage our software's configuration with a version control system, so that we could deliver a known baseline to the customer for installation.

Internet-facing systems consist of many parts that we can no longer control as a monolithic block of software. Availability and performance requirements drive the adoption of application

servers, load-balancing solutions, relational database management systems, and disaster recovery setups, while interoperability requirements and complex standards make us use a multitude of third-party libraries and online services.

Then consider the ubiquity of Internet-facing systems. Whereas in the past, most software happily ran as an isolated island; today, we expect all our applications to be accessible through the Web or connected through the Internet. Systems ranging from our flowerpot's watering automation to an airliner's jet engines regularly phone home to upload their latest status and receive instructions. All self-respecting smartphone apps require Internet connectivity. Companies, large and small, offer their products and services through the Web and increasingly use social networks to market and even design their offerings. The ecologically unchallenged simplicity of monolithic software had gone the way of the dodo.

Finally, realize that complexity and ubiquity feed on the cheaper infrastructure costs offered by cloud-based solutions and virtualization. Setting up a datacenter used to be something only a large company could afford. Now, armed with a credit card, a startup can use a cloud provider to set up separate development, test, and production servers, a database management system, a storage subsystem, a computation cluster, monitoring, and disaster recovery in a week.

Tools, Benefits, Practices

The revolution I've described affects us developers, because as we build software systems with many complex dependencies, we're no longer expected to provide working code but contribute toward a working IT system. For this, we need to cooperate closely with the team handling IT operations in order to coordinate and integrate software development, technology operations, and quality assurance.

When the development and operations teams work together in a so-called DevOps setting, developers don't toss software deliverables over a wall to operations for deployment. Instead, the two coordinate through various agile development processes like continuous deployment and automated testing. DevOps can work wonders when the organization provides software as a service (like Google), as a shrink-wrapped device (like the iPhone), or as a customized application (like the SAP ERP). Note, however, that applying the same principles on shrink-wrapped software isn't realistic.

A major DevOps enabler is an IT-system configuration management tool. Such tools allow us to control and automate the configuration of all elements comprising an IT system: hosts, installed software, users, running services, configuration files, scheduled tasks, networking, storage, monitoring, and security. (Note that although commonly termed *configuration management tools*, they're unrelated to the

Post your comments online
by visiting the column's blog:

www.spinellis.gr/tools

software configuration management tools, like Subversion and Git, we use to manage our software's revisions.)

Popular tools in this domain include CFEngine, Puppet, and Chef, but these are only three among the list of 20 open source tools listed in a comparison article on Wikipedia (http://en.wikipedia.org/wiki/Comparison_of_open_source_configuration_management_software). Their main function is to automate a system's configuration. You write some rules expressing how an IT system is to be configured, and the tool will set up the system accordingly. You can use configuration management tools to set up a new system starting from a blank slate, to add functionality to an existing system, and even to repair a system whose configuration is no longer up to spec. In all cases, through the specification, you end up having at hand precise executable documentation of the system's configuration. This, according to Philip Armour's view of software as executable knowledge, makes IT-system configuration management not only an essential tool of our trade as developers but also an important craft and vital skill.

Configuration management tools work by specifying a system's setup through rules. For instance, we would specify rules for configuring the system's basic infrastructure (storage, networking), each running an application or service, and the roles of the system's users. Each step of the configuration often depends on others: to build an application, we must install a compiler; to run a Web-facing service, we might need an application and a database server. To handle these dependencies, we typically express each rule of the configuration together with its pre- and postconditions. As an example, the following Puppet rule expresses that to run the Postfix mail server as a service, the system must have installed the corresponding package and configuration file. At the end, the service will be in a running state:

```
service { 'postfix':
  require => [
    Package['postfix'],
    File['/etc/postfix/main.cf'],
  ],
  enable => 'true',
  ensure => 'running',
}
```

High-level declarative specifications, such as the above, allow us to use familiar software engineering techniques for organizing large complex configurations of diverse hosts. Specifically, by breaking a system's configuration into rules and modules, we can decompose it into manageable pieces. By organizing rules into modules and classes we can hide irrelevant information and organize our configuration in a hierarchy. For instance, we specify a named application's configuration through discrete rules, we group multiple applications into a running service, and multiple services into a system. Through parameterization and class inheritance, we can reuse common elements and express variation in the configuration of


“Infrastructure as Code.” Given that a configuration script really contains code, treat it as such and work on it to build up your existing coding skills. Use meaningful identifier names and helpful comments, and follow style conventions regarding naming and layout.

At the process level, again, there are familiar best practices. By putting the configuration scripts under version control, you can collaborate with others, work on branches, merge features, tag stable deployments, and document the complete history of your system's state. Want to experiment with an upgrade? Create a branch, work on it, and when you're satisfied, merge the changes back to your master branch. Want to know when a new service was installed? Look when it made its first appearance in the configuration file's revision history. The daily build practice also has its parallel here. It's instructive to regularly build a system from scratch to ensure that its configuration is current and that no manually performed changes have crept into it. By stipulating that all modifications to a system's configuration can only

We're no longer expected to provide working code, but contribute toward a working IT system.

our hosts. Building on the preceding example, we can configure a running system's production and testing instances by building on a common baseline. Finally, the pre- and postconditions provide the system's specification.

By now, I'm sure you're convinced that an IT-system's configuration management script is essentially code like all the other code we deal with in our everyday work. The DevOps community has aptly termed this phenomenon

be performed through its configuration management system, an organization ensures that the IT systems it delivers to its clients are not inscrutable monoliths that just happen to work but documented modular engines that work by design. 

DIOMIDIS SPINELLIS is a professor in the Department of Management Science and Technology at the Athens University of Economics and Business and the author of the books *Code Reading* and *Code Quality: The Open Source Perspective* (Addison-Wesley, 2003, 2006). Contact him at dds@aub.gr.