Technology Conversations

Configuration Management in the Docker World

Anyone managing more than a few servers can confirm that doing such a task manually is a waste of time and risky. **Configuration management (CM)** exists for a long time and there is no single reason I can think of why one would not use one of the tools. The question is not whether to adopt one of them but which one to choose. Those that already embraced one or the other and invested a lot of time and money will probably argue that the best tool is the one they chose. As things usually go, the choices change over time and the reasons for one over the other might not be the same today as they were yesterday. In most cases, choices are not based on available options but by the architecture of the legacy system we are sworn to maintain. If such systems are to be ignored or someone with enough courage and deep pockets would be willing to modernize them, today's reality would be dominated by containers and microservices. In such a situation, the choices we made yesterday are definitely different from choices we could make today.

CFEngine

CFEngine can be considered father of configuration management. It was created in 1993 and revolutionized the way we approach server setups and configurations. It started as an open source project and become commercialized in 2008 when the first enterprise version was released.

CFEngine is written in C, has only few dependencies and is lightning fast. Actually, as to my knowledge, no other tool managed to overcome CFEngine's speed. That was, and still is its main strength. However, it had its weaknesses with requirement for coding skills being probably the main one. In many cases, an average operator was not able to utilize CFEngine. It requires a C developer to manage it. That did not prevent it from becoming widely adopted in some of the biggest enterprises. However, as youth usually wins over age, new tools were created and today rarely anyone chooses CFEngine without being "forced" to do so due to the investment the company made into it.

Puppet

Later on Puppet came into being. It also started as an open source project followed by the enterprise version. It was considered more "operations friendly" thanks to its model driven approach

and small learning curve when compared to CFEngine. Finally there was a configuration management tool that operations department could leverage. Unlike C utilized by CFEngine, Ruby proved to be easier to reason with and more accepted by ops. CFEngine's learning curve was probably the main reason Puppet got its footing into the configuration management market and slowly sent CFEngine into history. That does not mean that CFEngine is not used any more. It is and it doesn't seem it will disappear any time soon in the same way as Cobol is still present in many banks and other finance related businesses. However, it lost its reputation for being the weapon of choice.

Chef

Then came Chef promising to solve some of the nuances of Puppet. And it did, for a while. Later, as popularity of both Puppet and Chef continued increasing, they entered the "zero sum game". As soon as one of them came up with something new or some improvement, the other one adopted it. Both feature an ever increasing number of tools which tend to increase their learning curves and complexity. Chef is a bit more "developer friendly" while Puppet could be considered more oriented towards operations and sysadmin type of tasks. Neither has a clear enough advantage over the other and the choice is often made based on personal experience than anything else. Both Puppet and Chef are mature, widely adopted (especially in enterprise environments) and have a huge number of open source contributions. The only problem is that they are too complicated for what we are trying to accomplish. Neither of them was designed with containers in mind. Neither of them could know that the "game" would change with Docker since it didn't exist at the time they were designed.

All of the configuration management tools we mentioned thus far are trying to solve problems that we should not have the moment we adopt containers and immutable deployments. The server mess that we had before is no more. Instead of hundreds or even thousands of packages, configuration files, users, logs, and so on, we are now trying to deal with a lot of containers and very limited amount of anything else. That does not mean that we do not need configuration management. We do! However, the scope of what the tool of choice should do is much smaller. In most cases, we need a user or two, Docker service up and running and a few more things. All the rest are containers. Deployment is becoming a subject of a different set of tools and redefining the scope of what CM should do. **Docker Compose** and **Kubernetes** are only a few of a rapidly increasing number of deployment tools we might use today. In such a setting, our configuration management choice should value simplicity and immutability over other things. Syntax should be simple and easy to read even to those who never used the tool. Immutability can be accomplished by enforcing a push model that does not require anything to be installed on the destination server.

Ansible

Ansible tries to solve the same problems as other configuration management tools but in a very different way. One important difference is that it performs all its operations over SSH. CFEngine and Puppet require clients to be installed on all servers they are supposed to manage. While Chef claims

that it doesn't, its support for agent-less running has limited features. That in itself is a huge difference when compared to Ansible that does not require servers to have anything special since SSH is (almost) always present. It leverages well defined and widely used protocol to run whatever commands need to be run in order to make sure that the destination servers comply with our specifications. The only requirement is Python that is already pre-installed on most Linux distributions. In other words, unlike competitors that are trying to force you to setup servers in a certain way, Ansible leverages existing realities and does not require anything. Due to its architecture, all you need is a single instance running on a Linux or MacOS computer. We can, for example, manage all our servers from a laptop. While that is not advisable and Ansible should probably run on a "real" server (preferably the same one where other continuous integration and deployment tools are installed), laptop example illustrates its simplicity. In my experience, push based systems like Ansible are much easier to reason with than pull based tools we discussed earlier.

Learning Ansible takes a fraction of the time when compared to all the intricacies required to master the other tools. Its syntax is based on YAML (Yet Another Markup Language) and with a single glimpse over a playbook, even a person who never used a tool would understand what's going on. Unlike Chef, Puppet and, especially CFEngine that are written by developers for developers, Ansible is written by developers for people who have better things to do than learn yet another language and/or DSL.

Some would point out that the major downside is Ansible's limited support for Windows. The client does not even run on Windows and the number of modules that can be used in playbooks and run on it is very limited. This downside, assuming that we are using containers is, in my opinion, an advantage. Ansible developers did not waste time trying to create an all around tool and concentrated on what works best (commands over SSH on Linux). In any case, Docker is not yet ready to run containers in Windows. It might be in the future but at this moment (or at least the moment I was writing this text), this is on the road map and with questionable results. Even if we ignore containers and their questionable future on Windows, other tools are also performing much worst on Windows than Linux. Simply put, Windows architecture is not as friendly to the CM objectives than Linux is.

I probably went to far and should not be too harsh on Windows and question your choices. If you do prefer Windows servers over some Linux distribution, all my praise of Ansible is in vain. You should choose Chef or Puppet and, unless you already use it, ignore CFEngine.

Personal choice

If someone asked me few years ago which tool should we use I would have a hard time answering. Today, if one has the option to switch to containers (be it Docker or some other type) and immutable deployments, the choice is clear (at least among tools I mentioned). Ansible (when combined with Docker and Docker deployment tools) wins any time of the day. We might even argue whether CM tools are needed at all. There are examples when people rely completely on, let's say, CoreOS,

containers, and deployment tools like Docker Swarm or Kubernetes. I do not have such a radical opinion (yet) and think that CM continues being a valuable tool in the arsenal but. Due to the scope of the tasks CM tools needs to perform, Ansible is just the tool we need. Anything more complex or harder to learn would be an overkill. I am yet to find a person who had trouble maintaining Ansible playbooks. As a result, configuration management can easily become responsibility of the whole team.I'm not trying to say that infrastructure should be taken lightly (it definitely shouldn't). However, having contributions from the whole team working on a project is a big advantage for any type of tasks and CM should not be an exception. CFEngine, Chef and Puppet are an overkill with their complex architecture and their steep learning curve. At least when compared with Ansible.

The four tools we briefly went through are by no means the only ones we can choose from. You might easily argue that neither of those is the best and vote for something else. Fair enough. It all depends on preferences and objectives we are trying to archive. However, unlike the others, Ansible can hardly be a waste of time. It is so easy to learn that, even if you choose not to adopt it, you won't be able to say that a lot of valuable time was wasted. Besides, everything we learn brings something new and makes us better professionals.

Please search for Ansible in this blog if you'd like to see more hands-on examples of its utilization with Docker.

This entry was posted in Docker and tagged Ansible, CFEngine, chef, CM, Configuration Management, Docker, puppet on August 26, 2015 [http://technologyconversations.com/2015/08/26/configuration-management-in-the-docker-world/] by Viktor Farcic.