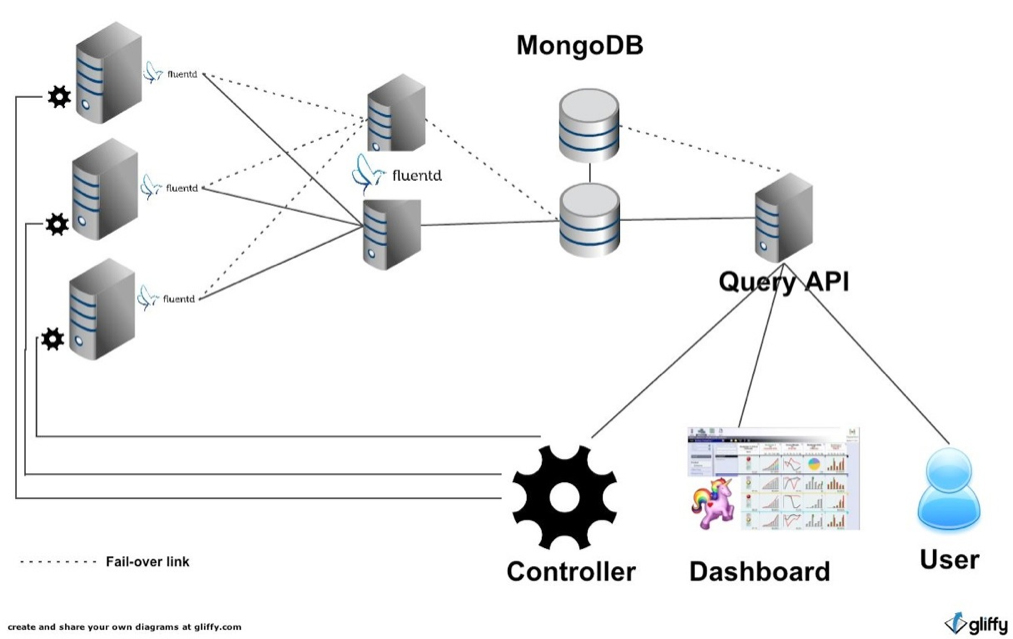
# 0x0A. Configuration management

TopicOn:**DevOps,SysAdmin,Scripting,CI/CD**

**Skynet** is a set of tools designed to monitor, scale and maintain a system in the Cloud. Put more simply, it’s a system that is aware about what’s happening on every single machine so it can also know about how the cluster is doing as a whole.

.

When I was working for SlideShare, I worked on an auto-remediation tool called [Skynet](https://intranet.alxswe.com/rltoken/0zbIzBqH_ktMmRQvJwZs2A) that monitored, scaled and fixed Cloud infrastructure. I was using a parallel job-execution system called MCollective that allowed me to execute commands to one or multiple servers at the same time. I could apply an action to a selected set of servers by applying a filter such as the server’s hostname or any other metadata we had (server type, server environment…). At some point, a bug was present in my code that sent nil to the filter method.

There were 2 pieces of bad news:

1. When MCollective receives nil as an argument for its filter method, it takes this to mean ‘all servers’
2. The action I sent was to terminate the selected servers

I started the parallel job-execution and after some time, I realized that it was taking longer than expected. Looking at logs I realized that I was shutting down SlideShare’s entire document conversion environment. Actually, 75% of all our conversion infrastructure servers had been shut down, resulting in users not able to convert their PDFs, powerpoints, and videos… Pretty bad!

**Thanks to Puppet, we were able to restore our infrastructure to normal operation in under 1H, pretty impressive. Imagine if we had to do everything manually: launching the servers, configuring and linking them, importing application code, starting every process, and obviously, fixing all the bugs (you should know by now that complicated infrastructure always goes sideways)…**

Obviously writing Puppet code for your infrastructure requires an investment of time and energy, but in the long term, it is for sure a must-have.

.

# An Introduction to Configuration Management.

* Configuration management (CM) refers to the process of systematically handling changes to a system in a way that it maintains integrity over time. Even though this process was not originated in the IT industry, the term is broadly used to refer to **server configuration management**.
* Automation plays an essential role in server configuration management. It’s the mechanism used to make the server reach a desirable state, previously defined by provisioning scripts using a tool’s specific language and features.
* Server Orchestration or IT Orchestration, since these tools are typically capable of managing one to hundreds of servers from a central controller machine.
* There are a number of configuration management tools available in the market. Puppet, Ansible, Chef and Salt are popular choices. They are all driven by the same purpose: to make sure the system’s state matches the state described by your provisioning scripts.

## [Benefits of Configuration Management for Servers](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "benefits-of-configuration-management-for-servers)

## **[Quick Provisioning of New Servers](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "quick-provisioning-of-new-servers)**

### **[Quick Recovery from Critical Events](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "quick-recovery-from-critical-events)**

### **[No More Snowflake Servers](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "no-more-snowflake-servers)**

### **[Version Control for the Server Environment](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "version-control-for-the-server-environment)**

### **[Replicated Environments](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "replicated-environments)**

## [Overview of Configuration Management Tools](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "overview-of-configuration-management-tools)

Most configuration management tools use a controller/master and node/agent model. Essentially, the controller directs the configuration of the nodes, based on a series of instructions or tasks defined in your provisioning scripts.

### [Automation Framework](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "automation-framework)

Each CM tool provides a specific syntax and a set of features that you can use to write provisioning scripts. Most tools will have features that make their language similar to conventional programming languages, but in a simplified way. Variables, loops, and conditionals are common features provided to facilitate the creation of more versatile provisioning scripts.

### [Idempotent Behavior](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "idempotent-behavior)

Configuration management tools keep track of the state of resources in order to avoid repeating tasks that were executed before. If a package was already installed, the tool won’t try to install it again. The objective is that after each provisioning run the system reaches (or keeps) the desired state, even if you run it multiple times. This is what characterizes these tools as having an idempotent behavior. This behavior is not necessarily enforced in all cases, though.

### [System Facts](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "system-facts)

Configuration management tools usually provide detailed information about the system being provisioned. This data is available through global variables, known as facts. They include things like network interfaces, IP addresses, operating system, and distribution. Each tool will provide a different set of facts. They can be used to make provisioning scripts and templates more adaptive for multiple systems.

### [Templating System](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "templating-system)

Most CM tools will provide a built-in templating system that can be used to facilitate setting up configuration files and services. Templates usually support variables, loops, and conditionals that can be used to maximise versatility. For instance, you can use a template to easily set up a new virtual host within Apache, while reusing the same template for multiple server installations. Instead of having only hard-coded, static values, a template should contain placeholders for values that can change from host to host, such as NameServer and DocumentRoot.

### [Extensibility](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "extensibility)

Even though provisioning scripts can be very specialized for the needs and demands of a particular server, there are many cases when you have similar server setups or parts of a setup that could be shared between multiple servers. Most provisioning tools will provide ways in which you can easily reuse and share smaller chunks of your provisioning setup as modules or plugins.

Third-party modules and plugins are often easy to find on the Internet, specially for common server setups like installing a PHP web server. CM tools tend to have a strong community built around them and users are encouraged to share their custom extensions. Using extensions provided by other users can save you a lot of time, while also serving as an excellent way of learning how other users solved common problems using your tool of choice.

## [Choosing a Configuration Management Tool](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "choosing-a-configuration-management-tool)

There are many CM tools available in the market, each one with a different set of features and different complexity levels. Popular choices include Chef, Ansible, and Puppet. The first challenge is to choose a tool that is a good fit for your needs.

There are a few things you should take into consideration before making a choice:

### [Infrastructure Complexity](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "infrastructure-complexity)

Most configuration management tools require a minimum hierarchy consisting of a controller machine and a node that will be managed by it. Puppet, for example, requires an agent application to be installed on each node, and a master application to be installed on the controller machine. Ansible, on the other hand, has a decentralized structure that doesn’t require installation of additional software on the nodes, but relies on SSH to execute the provisioning tasks. For smaller projects, a simplified infrastructure might seem like a better fit, however it is important to take into consideration aspects like scalability and security, which may not be enforced by the tool.

Some tools can have more components and moving parts, which might increase the complexity of your infrastructure, impacting on the learning curve and possibly increasing the overall cost of implementation.

### [Learning Curve](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "learning-curve)

As mentioned earlier in this article, CM tools provide a custom syntax, sometimes using a Domain Specific Language (DSL), and a set of features that comprise their framework for automation. As with conventional programming languages, some tools will demand a higher learning curve to be mastered. The infrastructure requirements might also influence the complexity of the tool and how quickly you will be able to see a return of investment.

### [Cost](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "cost)

Most CM tools offer free or open source versions, with paid subscriptions for advanced features and services. Some tools will have more limitations than others, so depending on your specific needs and how your infrastructure grows, you might end up having to pay for these services. You should also consider training as a potential extra cost, not only in monetary terms, but also regarding the time that will be necessary to get your team up to speed with the tool you end up choosing.

### [Advanced Tooling](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "advanced-tooling)

As mentioned before, most tools offer paid services that can include support, extensions, and advanced tooling. It’s important to analyse your specific needs, the size of your infrastructure and whether or not there is a need for using these services. Management panels, for instance, are a common service offered by these tools, and they can greatly facilitate the process of managing and monitoring all your servers from a central point. Even if you don’t need such services just yet, consider the options for a possible future necessity.

### [Community and Support](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "community-and-support)

A strong and welcoming community can be extremely resourceful for support and for documentation, since users are typically happy to share their knowledge and their extensions (modules, plugins, and provisioning scripts) with other users. This can be helpful to speed up your learning curve and avoid extra costs with paid support or training.

## [Overview of Popular Tools](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "overview-of-popular-tools)

The table below should give you a quick overview of the main differences between three of the most popular configuration management tools available in the market today: Ansible, Puppet, and Chef.

|  | ****Ansible**** | ****Puppet**** | ****Chef**** |
| --- | --- | --- | --- |
| **Script Language** | YAML | Custom DSL based on Ruby | Ruby |
| **Infrastructure** | Controller machine applies configuration on nodes via SSH | Puppet Master synchronizes configuration on Puppet Nodes | Chef Workstations push configuration to Chef Server, from which the Chef Nodes will be updated |
| **Requires specialized software for nodes** | No | Yes | Yes |
| **Provides centralized point of control** | No. Any computer can be a controller | Yes, via Puppet Master | Yes, via Chef Server |
| **Script Terminology** | Playbook / Roles | Manifests / Modules | Recipes / Cookbooks |
| **Task Execution Order** | Sequential | Non-Sequential | Sequential |

## [Next Steps](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "next-steps)

So far, we’ve seen how configuration management works for servers, and what to consider when choosing a tool for building your configuration management infrastructure. In subsequent guides in this series, we will have a hands-on experience with three popular configuration management tools: Ansible, Puppet and Chef.

In order to give you a chance to compare these tools by yourself, we are going to use a simple example of server setup that should be fully automated by each tool. This setup consists of an Ubuntu 18.04 server running Apache to host a simple web page.

## [Conclusion](https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management" \l "conclusion)

Configuration management can drastically improve the integrity of servers over time by providing a framework for automating processes and keeping track of changes made to the system environment. [In the next guide](https://www.digitalocean.com/community/tutorials/configuration-management-101-writing-ansible-playbooks) in this series, we will see how to implement a configuration management strategy in practice using Ansible as tool.