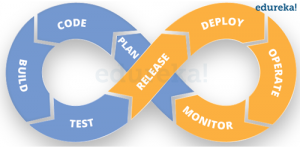
**DevOps**

**What is DevOps?**

DevOps integrates developers and operations team to improve collaboration and productivity.

It is a set of process between software development and IT teams in order that they can build, test and release software faster and more reliable.



According to the DevOps culture, a single group of Engineers (developers, system admins, QA’s. Testers etc. turned into DevOps Engineers) has end to end responsibility of the Application (Software) right from gathering the requirement to development, to testing, to infrastructure deployment, to application deployment and finally monitoring & gathering feedback from the end users, then again implementing the changes.

**Why DevOps?**

Before DevOps the traditional waterfall model was used.

The processes involved in a waterfall software development model as follows:

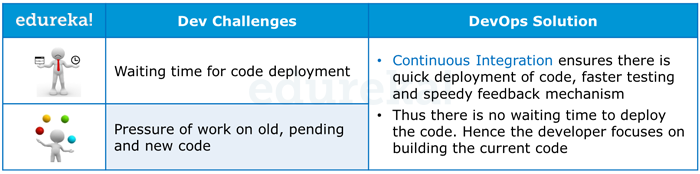
* Requirement Definition (Analysis and Specification)
* Software Design
* Implementation (Coding)
* Testing
* Deployment (Installation)
* Maintenance

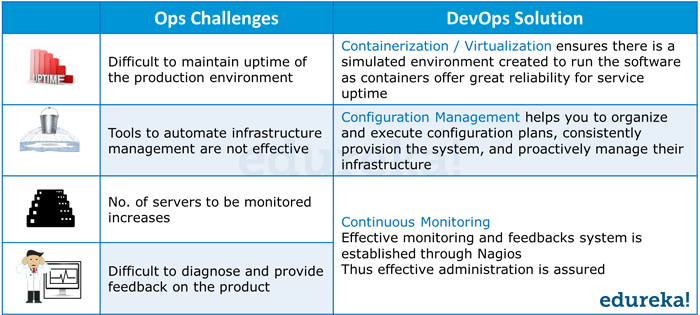
**WATERFALL IN TERMS OF CULTURE, CONTINUOUS DELIVERY AND INTEGRATION**

The culture of the waterfall model is such that testing only occurs at the end of the project which may be very expensive, provided that there is need for modification, this will cause the re-start of the entire process, from the requirements down to the very last phase in the lifecycle.

It is apparent that the waterfall does not, in any way, support the continuous delivery which entails the fast and automated feedbacks on the production of software which, in other words leads to the deployment of software at the end of each phase in the lifecycle. These processes are clearly not present in this model. The only time feedback and deployment is made is at the end of the waterfall model. This fact equally shows that it doesn’t incorporate continuous integration as this is what leads to continuous delivery.

DevOps takes care of the challenges faced by Development and Operations:

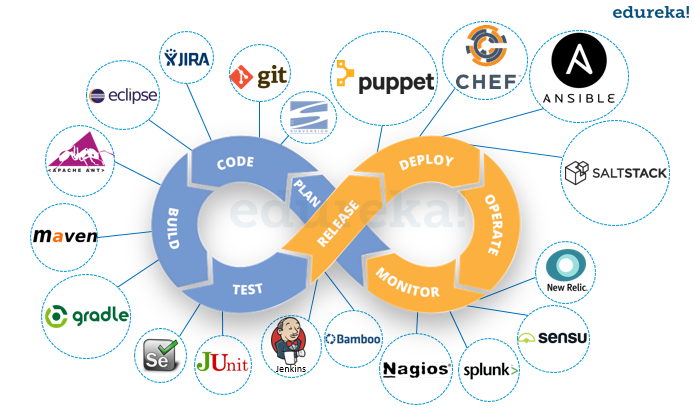




However, you would still be wondering, how to implement DevOps.

One also needs various DevOps tools like **Puppet**, **Jenkins**, **GIT**, **Chef**, **Docker**, **Selenium**, **AWS** etc. to achieve automation at various stages which helps in achieving Continuous Development, Continuous Integration, Continuous Testing, Continuous Deployment, Continuous Monitoring to deliver a quality software to the customer at a very fast pace.

Now take a look at the below DevOps diagram with various DevOps Tools closely and try to decode it.



These tools has been categorized into various stages of DevOps.

DevOps Lifecycle can be broadly broken down into the below DevOps Stages:

* Continuous Development
* Continuous Integration
* Continuous Testing
* Continuous Monitoring
* Virtualization and Containerization

These stages are the building blocks to achieve DevOps as a whole.

Let us discuss about one of the DevOps tool that is Ansible.

**ANSIBLE**

* Ansible is software that automates software provisioning, configuration management, and application deployment. Simply, we can say it as **configuration management tool.**
* Ansible is a free software for configuring and managing nodes. It easy to use and includes many **built-in modules** to allow easy configuration management.
* Configuration management systems are designed to make controlling large numbers of servers easy for administrators and operations teams. They allow you to control many different systems in an automated way from one central location.
* It uses ssh (secure shell - secure remote logins) to connect the different nodes and configure them. The only thing you need on your machine in order to run Ansible is python installed (> 2.4).
* Ansible is installed on a single host, which can even be your local machine, and uses SSH to communicate with each remote host. This allows it to be incredibly fast at configuring new servers, as there are no prerequisite packages to be installed on each new server. It is incredibly easy to use and understand, since it uses playbooks in yaml format using a simple module based syntax.
* No need to do anything on the remote server, everything can be handled from your local computer.

**Simple. Powerful. Agentless.**

With Ansible you can start to do real work in just minutes due to its simple, **human-readable language**. Altogether its powerful capabilities allow orchestration of your entire application lifecycle regardless of where it’s deployed. And Ansible’s agentless architecture means it is one less thing to keep secure. In the field of information technology, **agentless** data collection involves collecting data from computers without installing any new agents on them.

**How Does Ansible Work?**

* Ansible works by configuring client machines from a computer with Ansible components installed and configured.
* It communicates over normal SSH channels in order to retrieve information from remote machines, issue commands, and copy files.
* This is one way that Ansible simplifies the administration of servers.
* Any computer that you can administer through SSH, you can also administer through Ansible.
* Ansible takes on a modular approach, making it easy to extend to use the functionalities of the main system to deal with specific scenarios.
* Modules can be written in any language and communicate in standard JSON.
* Configuration files are mainly written in the **YAML** data serialization format due to its expressive nature and its similarity to popular markup languages.
* Ansible can interact with clients through either command line tools or through its configuration scripts called Playbooks.

**Ansible Playbooks:**

* Ansible playbooks are a way to send commands to remote computers in a scripted way. Instead of using Ansible commands individually to remotely configure computers from the command line, you can configure entire complex environments by passing a script to one or more systems.
* Ansible playbooks are written in the YAML data serialization format. If you don't know what a data serialization format is, think of it as a way to translate a programmatic data structure (lists, arrays, dictionaries, etc) into a format that can be easily stored to disk. The file can then be used to recreate the structure at a later point. JSON is another popular data serialization format, but YAML is much easier to read.
* Each playbook contains one or more plays, which map hosts to a certain function. Ansible does this through something called tasks, which are basically module calls.
* Script be run at once and configure multiple servers.

**Ansible Installation:**

We can install ansible from github. So first get the git

* yum install git

Now, give the command

$ git clone git://github.com/ansible/ansible.git --recursive

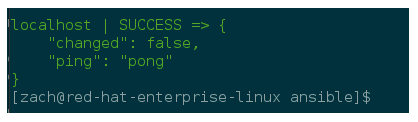
$ cd ./ansible

We still need to install Ansible, but we’re done with the setup portion. Installing Ansible via yum works much the same way as any other yum package. Simply pop open a terminal and run the following command

* sudo yum install ansible

Now, we are done. To test things out, simply ping your Ansible server on localhost. You should receive a “pong” in response.

* ansible localhost –m ping



And that’s it! Ansible is now ready, you can check for the version,

Check for ansible version:

* ansible –version

**Ansible Inventory:**

Ansible can work for multiple systems at a time. It achieves this by selecting portions of systems listed in Ansible’s inventory, which is by default saved in the location */etc/ansible/hosts*. You can specify a different inventory file using the*-i <path>* option on command line.

The inventory file can be in one of many formats, depending on the inventory plugins you have. For this example, the format for */etc/ansible/hosts* is an INI-like and looks like this:

[webservers]

192.168.153.155

192.168.153.156

It is easy to assign variables to hosts that will be used in playbooks:

**[testservers]**

host1 http\_port=80 maxRequestsPerChild=808

host2 http\_port=303 maxRequestsPerChild=909

**Ansible Ad-Hoc Commands:**

An ad-hoc command is something that you might type in to do something rapidly, but don’t want to save for later. Just like executing a command in the shell instead of creating the shell script for that. An ad-hoc command contains two different parameters; the host group on which task is going to run and the module to run. If you want to ping each host with a single command, you can do it using the following:

ansible host\_group -m ping

Similarly, you can perform many other operations using ansible like copying a file, managing packages, gathering facts, etc.

Ad-hoc commands are a powerful yet straightforward feature of Ansible.

**Ansible Playbook and Modules**

Playbooks are a completely different way to use Ansible than in ad-hoc task execution mode and are particularly powerful. There is a way to send commands to the remote node using the script, like a shell script that contains the set of command. Ansible Playbooks are written in the YAML format. YAML is a data serialization language.

In every Playbook, there are one or more "plays" in a list. The goal of the play is to map hosts with a certain function. Ansible does it through the task, which is nothing more than a call to an Ansible module.

Example of a playbook:

---

- hosts: webservers

vars:

http\_port: 80

max\_clients: 200

remote\_user: root

tasks:

- name: ensure apache is at the latest version

yum: name=httpd state=latest

- name: write the apache config file

template: src=/srv/httpd.j2 dest=/etc/httpd.conf

notify:

- restart apache

- name: ensure apache is running (and enable it at boot)

service: name=httpd state=started enabled=yes

handlers:

- name: restart apache

service: name=httpd state=restarted

**Roles**

As you add more and more functionality to your Ansible playbooks, it becomes difficult to manage it as a single file.

Roles allow you to prepare a minimal Ansible playbook that defines how a server is supposed to perform rather than specifying the steps to get a server to act in a specific way.

According to Ansible documentation, “Roles in Ansible build on the idea of **include** files and combine them to form clean, reusable abstractions – they allow you to focus more on the big picture and only dive down into the details when needed.”

To correctly use roles with Ansible, you need to create a roles directory in your working Ansible directory, and then any necessary sub-directories.

**Command:**

# ansible-galaxy init *role\_name*

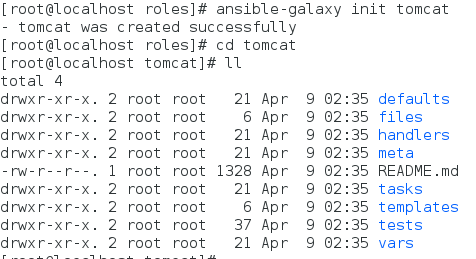
I am creating my role ‘tomcat’ in /etc/ansible/roles/

# ansible-galaxy init tomcat

The Ansible roles structure:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | Roles  |\_\_ defaults          |\_\_ main.yml  |\_\_ files  |\_\_ templates  |\_\_ tasks          |\_\_ main.yml  |\_\_ meta          |\_\_ main.yml  |\_\_ vars          |\_\_ main.yml  |\_\_ handlers          |\_\_ main.yml |

There are two ways to build the roles directory format, manually or by using Ansible-Galaxy. Ansible galaxy is free site for finding, reusing and sharing community developed roles. To create a role using ansible galaxy, use the ansible-galaxy command :



The above shown is the directory structure created for the role ‘tomcat’

Every playbook starts with three dashes (---) followed by host list, then a variable list, then a task list, and at the end there are handlers.

The **host** list contains the list of hosts where we want to run the task.

The**variable** list is to set the properties for the current play.

The**task** list contains the number of tasks which are going to execute.

The **handlers** are also tasks; the only difference is that in order to execute handler we need some trigger in the list of task. For example, notify. These ‘notify’ actions are triggered at the end of each block of tasks in a play, and will only be triggered once even if notified by multiple different tasks.

To run a playbook, we can use the following command:

ansible-playbook playbook\_name

Ansible ships with many modules (called the "module library") that can be executed directly on remote hosts or through Playbooks.

Users can also write their own modules. These modules can control system resources like services, packages, or files (anything really), or handle executing system commands.

Links:

https://www.quora.com/What-are-the-best-books-on-devops

<https://medium.com/@banshy4real/comparison-between-waterfall-agile-lean-and-devops-e50236977df2>

<https://www.quora.com/What-are-the-best-books-on-devops>

<https://github.com/centminmod/centminmod/issues/37>

<https://dzone.com/articles/getting-started-with-ansible>

<https://www.digitalocean.com/community/tutorials/how-to-use-ansible-roles-to-abstract-your-infrastructure-environment> ---> chk and cmplte the doc