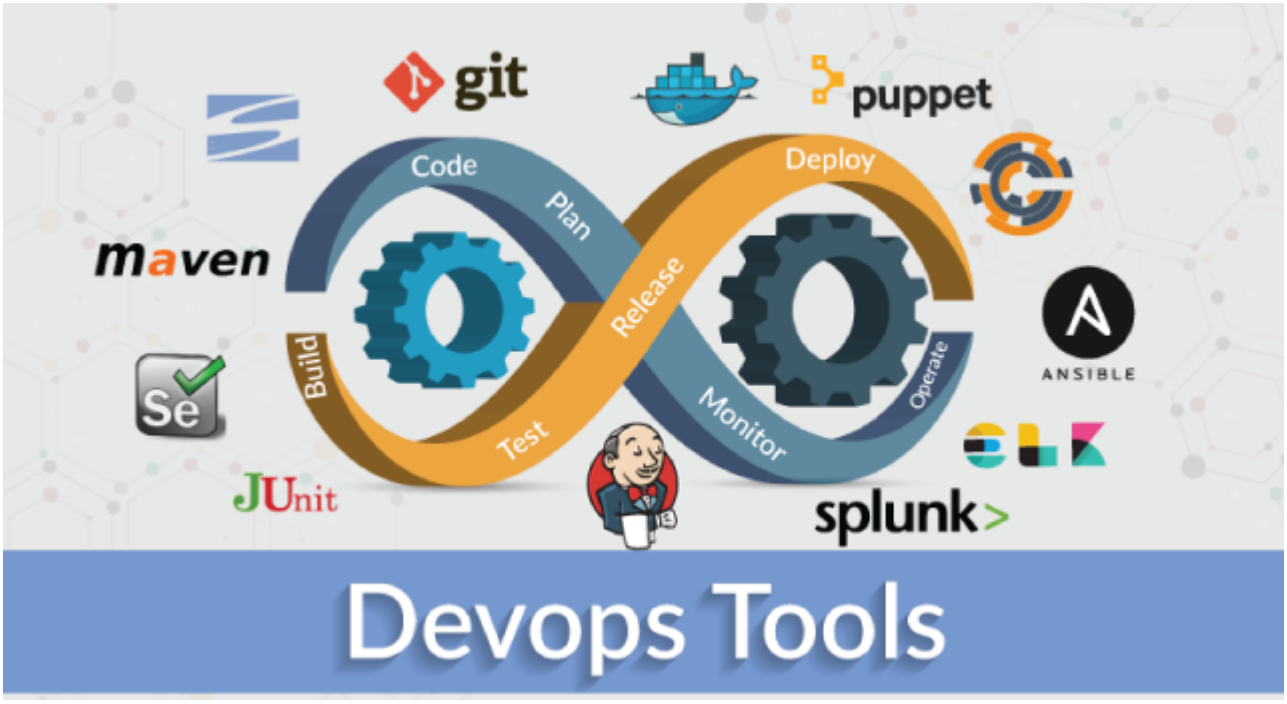
How To Orchestrate DevOps Tools Together To Solve Our Problems?

Saurabh Kulshrestha Follow
Oct 18, 2017 · 9 min read



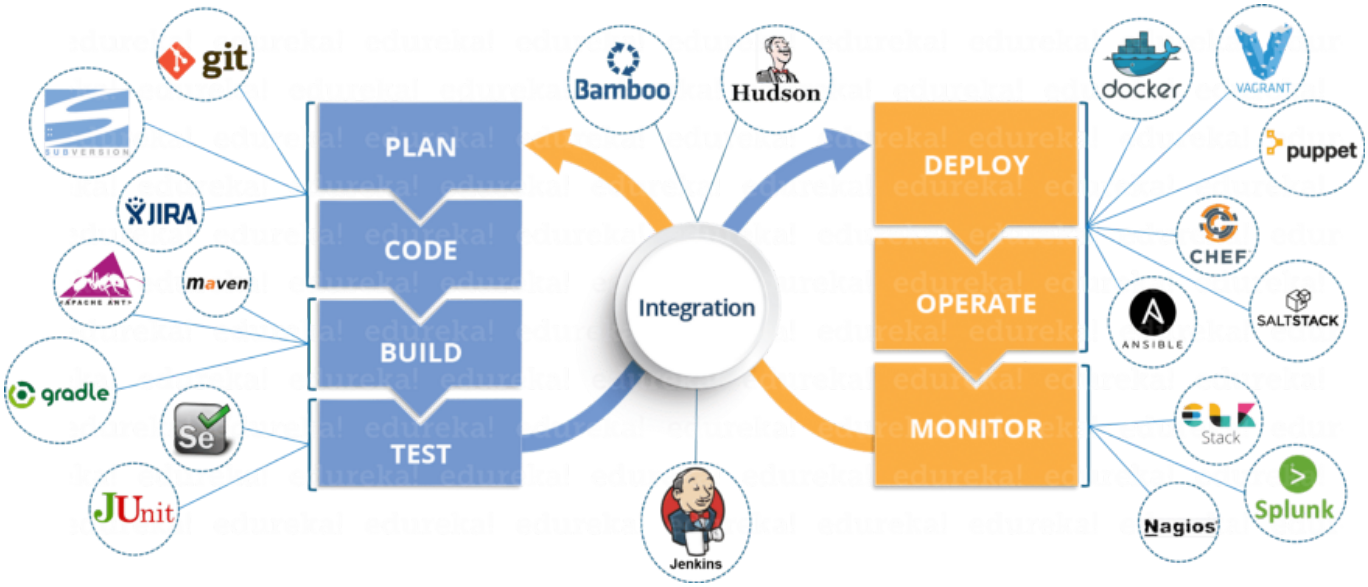
DevOps Tools - Edureka

DevOps is a software development approach that involves Continuous Development, Continuous Testing, Continuous Integration, Continuous Deployment, and Continuous Monitoring of the software throughout its development life cycle. To get through these stages, we need DevOps Tools. In this article, we will discuss various DevOps Tools that could be used in the life cycle.

Let's get started then.

DevOps Tools

Before going any further, let's recap what are the different tools and where they fall in the DevOps lifecycle.



DevOps Lifecycle Phases (DevOps Tools)

Well, I'm pretty sure you're impressed with the above image. But, you might still have problems relating the tools to various phases. Don't you?

In that case, let's take a step back and first understand what are the various phases present in the DevOps lifecycle. Below are the 5 different phases any software/ application has to pass through when developed via the DevOps lifecycle:-

1. Continuous Development
2. Continuous Testing
3. Continuous Integration
4. Continuous Deployment
5. Continuous Monitoring

1. Continuous Development

This is the phase that involves '**planning**' and '**coding**' of the software application's functionality. There are no tools for planning as such, but there are a number of tools for maintaining the code.

The vision of the project is decided during the 'planning' phase and then when they start writing the code, the act is referred to as the 'coding' phase.

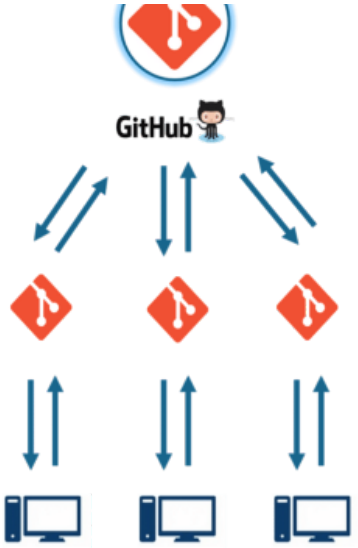
The code can be written in any language, but it is maintained by using **Version Control** tools. These are the Continuous Development DevOps tools. The most popular tools used are **Git**, **SVN**, **Mercurial**, **CVS,** and **JIRA**.

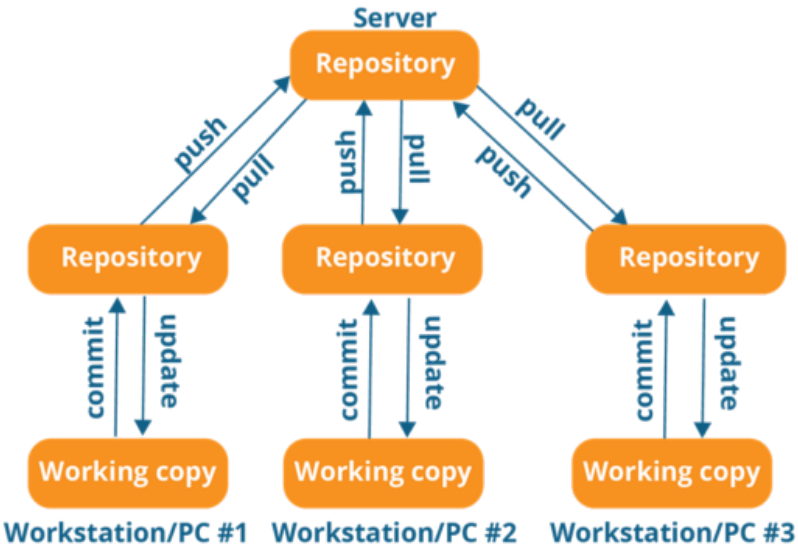
So why is it important to main versions of the code? Which of the Dev vs Ops problem does it solve? Let's understand that first.

- Versions are maintained (in a central repository), to hold a single source of truth. So that all the developers can collaborate on the 'latest committed' code, and even operations can have access to that same code when they plan to make a release.
- Whenever a mishap happens during a release, or even if there are lots of bugs in the code (faulty feature), there is nothing to worry. Ops can quickly rollback the deployed code and thus revert back to the previous stable state.

So, which is my favorite tool? That has got to be Git & GitHub. Why? Because **Git** allows developers to collaborate with each other on a Distributed VCS (Version Control System).

Since there is no dependency on the central server, 'pulls' & 'pushes' to the repository can be made from remote locations. This central repository where the code is maintained is called **GitHub**.





Git is in-fact the world's leading Version Control system. If you don't want to take my word for it, you can just google that up. So let's move on to the next topic in this DevOps tools blog.

2. Continuous Testing

When the code is developed, it is maddening to release it straight to deployment. The code should first be tested for bugs and performance. Can we agree on that statement?

If yes, then what would be the procedure to perform the tests? Would it be manual testing? Well, it can be, but it is very inefficient. So, what is better? Automation testing? Exactly! Sounds amazing right?

Automation testing is the answer to a lot of cries of manual testers. Tools like **Selenium**, **TestNG**, **JUnit/ NUnit** are used to automate the execution of our test cases. So, what are its benefits?

- Automation testing saves a lot of time, effort, and labor for executing the tests manually.
- Besides that, report generation is a big plus. The task of evaluating which test cases failed in a test suite gets simpler.
- These tests can also be scheduled for execution at predefined times. Brilliant right?

And the continuous use of these tools while developing the application is what forms the '**Continuous Testing**' phase during the DevOps lifecycle. Which of these is my favorite

tool? A combination of these tools actually!

Selenium is my favorite, but **Selenium** without **TestNG** is equivalent to a snake without a poisonous sting, at least from the perspective of the DevOps lifecycle.

Selenium does the automation testing, and the reports are generated by TestNG. But to automate this entire testing phase, we need a **trigger,** right? So, what is the trigger? This is where the role of **Continuous Integration** tools like **Jenkins** coming into the picture.

Now, let us move onto the next topic in this DevOps tools blog.

3. Continuous Integration

This is the most brilliant DevOps phase. It might not make sense during the first cycle of release, but then you will understand this phase's importance going forward.

Wait, that is not completely correct. **Continuous Integration** (CI) plays a major role even during the first release. It helps massively to integrate the CI tools with configuration management tools for deployment.

Undisputedly, the most popular **CI** tool in the market is **Jenkins**. And personally, Jenkins is my favorite DevOps tool. Other popular CI tools is **Bamboo** and **Hudson**.

Why do I hold such high regard for Continuous Integration tools? Because they are the ones that hold the entire '**DevOps structure**' together.

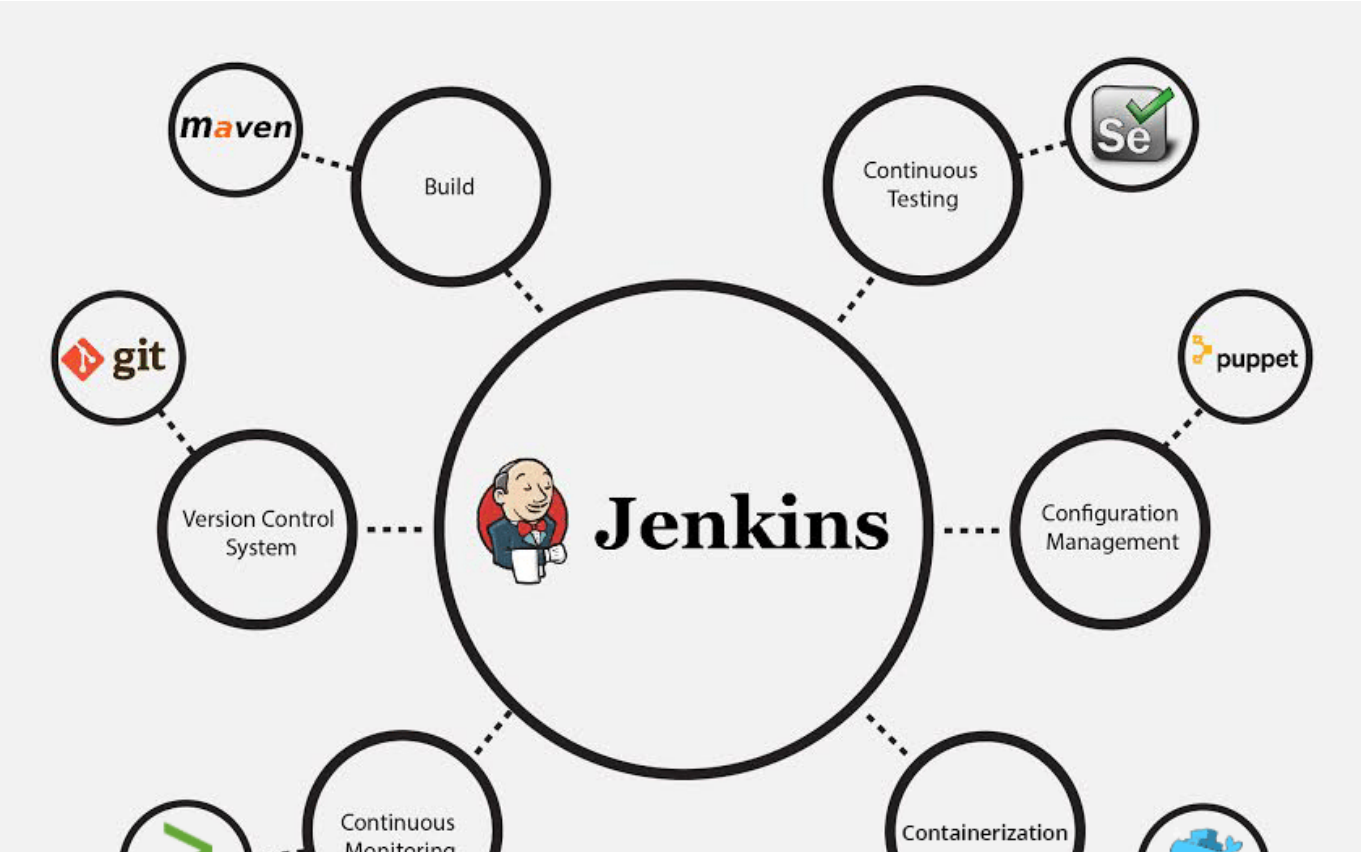
It is the CI tools that orchestrate the automation of tools falling under other DevOps lifecycle phases. Be it, Continuous Development tools, or Continuous Testing tools, or Continuous Deployment tools, or even Continuous Monitoring tools, the **Continuous Integration** tools can be integrated with all of them.

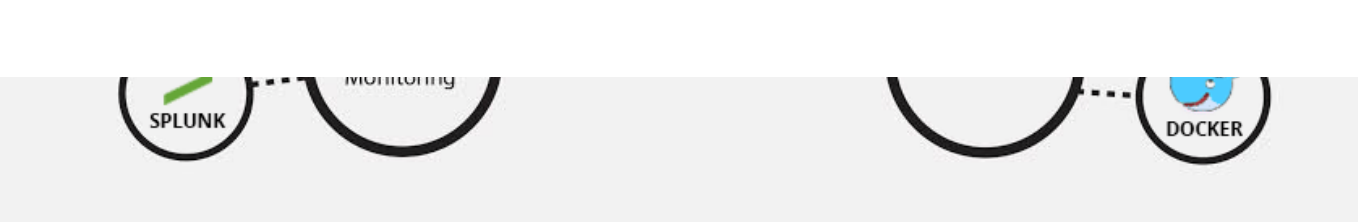
- When integrated with Git/ SVN, Jenkins can **schedule jobs** (pulling the code from shared repositories) automatically and make it ready for **builds** and **testing** (Continuous Development). Jenkins can build jobs either at scheduled times of the day or whenever there is a commit pushed to the central repository.

- When integrated with testing tools like Selenium, we can achieve Continuous Testing. How? The developed code can be built using tools like Maven/ Ant/ Gradle. When the code is built, then Selenium can ***automate the execution of that code***.
  How does it automate it? By creating a suite of test cases and executing the test cases one after the other.
  The role of Jenkins/ Hudson/ Bamboo here would be to schedule/ automate "Selenium to automate test case execution".

- When integrated with Continuous Deployment tools, Jenkins/ Hudson/ Bamboo can **trigger** the **deployments** planned by configuration management/ containerization tools.

- And finally, Jenkins/ Hudson can be integrated with monitoring tools like Splunk/ ELK/ Nagios/ NewRelic, to continuously **monitor** the **status & performance** of the server where the deployments have been made.





Because CI tools are capable of this and so much more, they are my favorite. Hence my statement: *Jenkins is an elementary DevOps tool*.

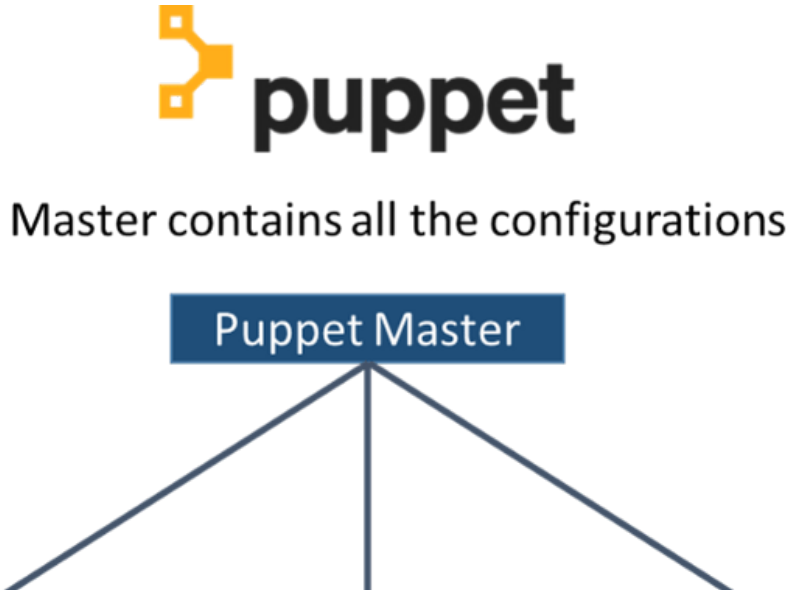
4. Continuous Deployment

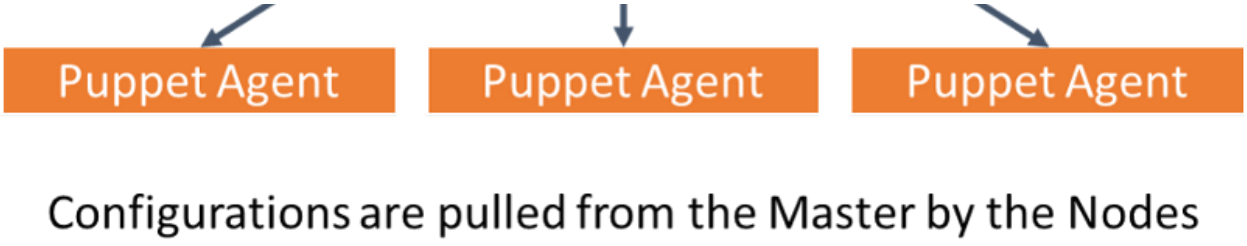
This (Continuous Deployment) is the phase where the action actually happens. We have seen the tools which help us build the code from scratch and also those tools which help in testing. Now it is time to understand why DevOps will be incomplete without **Configuration Management** tools or **Containerization** tools. Both sets of tools here help in achieving Continuous Deployment (CD).

Configuration Management Tools

- Configuration Management is the act of establishing and maintaining consistency in an applications' functional requirements and performance. In simpler words, it is the act of **releasing deployments** to servers, **scheduling updates** on all servers and most importantly keeping the **configurations consistent** across all the servers.

- For this, we have tools like **Puppet**, **Chef**, **Ansible**, **SaltStack** and more. But the best tool here is Puppet. Puppet & the other CM tools work based on the master-slave architecture. When there is a deployment made to the master, the master is responsible for replicating those changes across all the slaves, no matter the number! Amazing right?



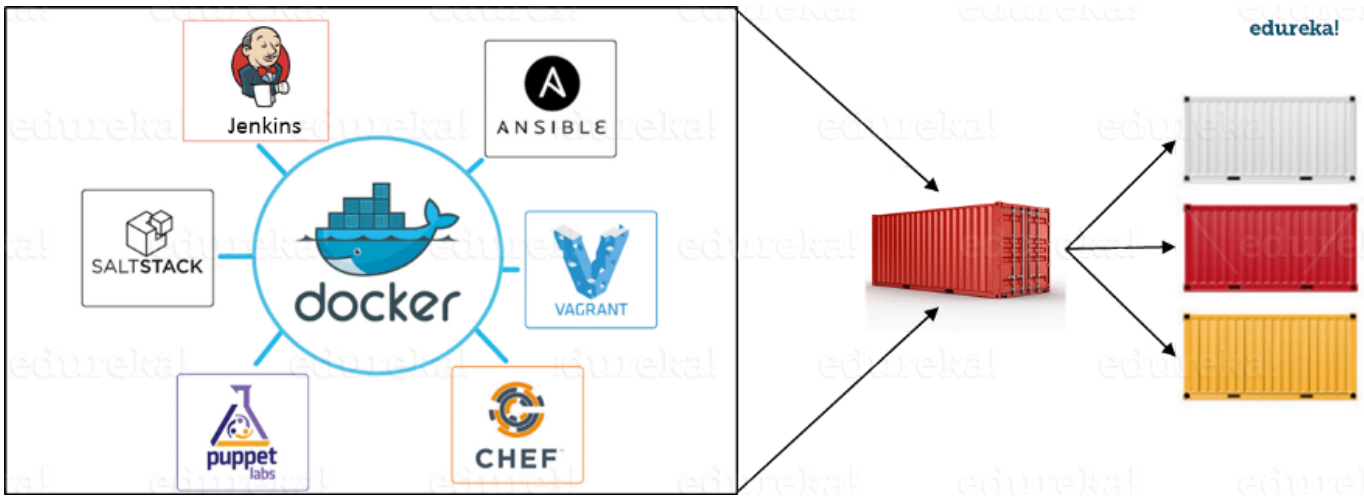


Now let's move onto Containerization.

Containerization Tools

- Containerization tools are other sets of tools that help in **maintaining consistency** across the environments where the application is developed, tested and deployed. It eliminates any chance of errors/ failure in the production environment by **packaging** and **replicating** the same dependencies and packages used in the development/ testing/ staging environment.

- The clear winner here is **Docker**, which was among the first containerization tool ever. Earlier, this act of maintaining consistency in environments was a challenge because VMs and servers were used, and their environments would have to be managed manually to achieve consistency. ***Docker containers*** threw this challenge up above and blew it out of the water. (Pun intended!)



- Another containerization tool is **Vagrant**. But off-late, a number of cloud solutions have started providing support for container services. **Amazon ECS**, **Azure Container Service,** and **Google Container Engine** are a few of the cloud services that have started radical support for Docker containers. This is the reason why Docker is the clear winner.

So now, let's move on to the final topic in this DevOps tools blog.

5. Continuous Monitoring

Well, what is the point of developing an application and deploying it, if we do not monitor its performance? **Monitoring** is as important as developing the application because there will always be a chance of bugs that escape undetected during the testing phase.

Which tools fall under this phase? **Splunk**, **ELK Stack**, **Nagios**, **Sensu**, **NewRelic** are some of the popular tools for monitoring. When used in combination with Jenkins, we achieve Continuous Monitoring. So, how does monitoring help?

- To minimize the consequences of buggy features, monitoring is a big add-on. Buggy features most often tend to cause financial loss. So, all the more reason to perform continuous monitoring.
- Monitoring tools also report failure/ unfavorable conditions before your clients/ customers get to experience the faulty features. Don't we all prefer this?

Which is my favorite tool here? I would prefer either Splunk or ELK stack. These two tools are major competitors. They pretty much provide the same features. But the way they provide the functionality is where they are different.

Splunk is a propriety tool (paid tool). But, this also effectively means that working on Splunk is very easy. ELK stack, however, is a combination of 3 open-source tools: ElasticSearch, LogStash & Kibana. It may be free, but setting it up is not as easy as a commercial tool like Splunk. You can try both of them to figure out the better for your organization.

Well, these were the various phases of the DevOps lifecycle and the tools that fit seamlessly in those situations. I hope you understood the application of these DevOps tools in the industry.

I hope you have enjoyed reading this post. If you wish to check out more articles on the market's most trending technologies like Artificial Intelligence, Python, Ethical Hacking, then you can refer to Edureka's official site.

Do look out for other articles in this series which will explain the various other aspects of DevOps.

1. DevOps Tutorial
2. Git Tutorial
3. Jenkins Tutorial
4. Docker Tutorial
5. Ansible Tutorial
6. Puppet Tutorial
7. Chef Tutorial
8. Nagios Tutorial
9. Continuous Delivery
10. Continuous Integration
11. Continuous Deployment
12. Continuous Delivery vs Continuous Deployment
13. CI CD Pipeline
14. Docker Compose
15. Docker Swarm
16. Docker Networking
17. Ansible Roles
18. Ansible Vault
19. Ansible for AWS

20. Jenkins Pipeline

21. Top Git Commands

22. Top Docker Commands

23. Git vs GitHub

24. DevOps Interview Questions

25. Who Is A DevOps Engineer?

26. DevOps Life cycle

27. Git Reflog

28. Ansible Provisioning

29. Top DevOps Skills That Organizations Are Looking For

30.Waterfall vs Agile

31. Maven For Building Java Applications

32. Jenkins CheatSheet

33. Ansible Cheat Sheet

34. Ansible Interview Questions And Answers

35. 50 Docker Interview Questions

36. Agile Methodology

37. Jenkins Interview Questions

38. Git Interview Questions

39. Docker Architecture

40. Linux commands Used In DevOps

41. Jenkins vs Bamboo

42. Nagios Interview Questions

43.DevOps Real-Time Scenarios

44.Difference between Jenkins and Jenkins X

45.Docker for Windows

46.Git vs Github

Originally published at www.edureka.co on October 18, 2017.

DevOps Devops Tools Git Jenkins Docker