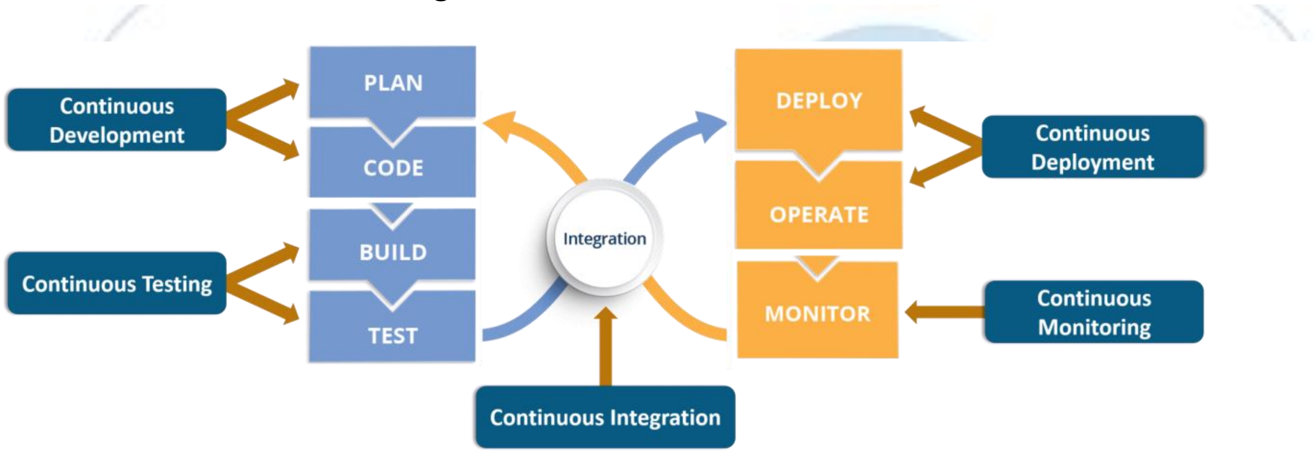
DevOps Lifecycle Phases (DevOps Tools)

DevOps lifecycle:

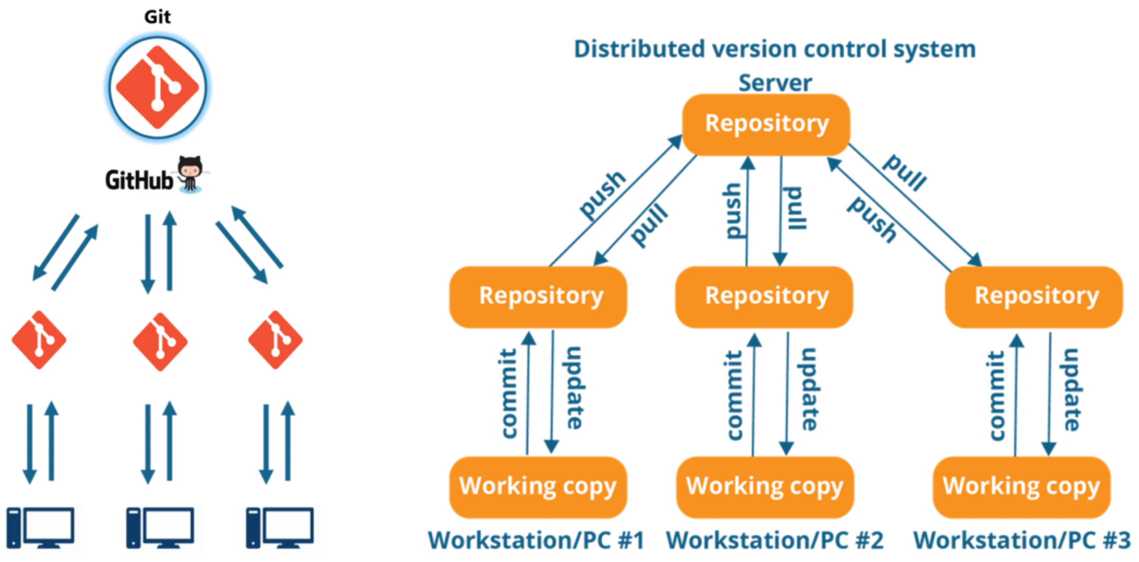
1. Continuous Development
2. Continuous Testing
3. Continuous Integration
4. Continuous Deployment
5. Continuous Monitoring



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 several tools for maintaining the code.

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 of which are Git, SVN, Mercurial, CVS, and JIRA.



Git version control system

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.

Automation testing tools like Selenium, TestNG, JUnit are used to automate the execution of test cases.

Benefits of Test Automation:

1. Faster Feedback

Automated testing comes as a relief for validation during various phases of a software project. This improves communication among coders, designers, and Product Owners, and allows potential glitches to be immediately rectified. Automated testing assures higher efficiency of the development team.

2. Accelerated Results

Owing to the quick implementation of automated testing, plenty of time is saved even for intricate and enormous systems. This allows for the testing to be carried out repeatedly, delivering faster results each time with lesser effort and time.

3. Reduced Business Expenses

It comes as no surprise that while the initial investment may be on the higher side, automated testing saves companies many a penny. This is predominantly due to the sharp drop in the amount of time required to run tests. It contributes to a higher quality of work, thereby decreasing the necessity for fixing glitches after release and reduces project costs.

4. Testing Efficiency Improvement

Testing takes up a significant portion of the overall application development lifecycle. This goes to show that even the slightest improvement of the overall efficiency can make an enormous difference to the overall timeframe of the project. Although the setup time takes longer initially, automated tests eventually take up significantly lesser amounts of time. They can be run virtually unattended, leaving the results to be monitored towards the end of the process.

5. Higher Overall Test Coverage

Through the implementation of automated tests, more tests can be executed pertaining to an application. This leads to a higher coverage that in a manual testing approach, would imply a massive team limited heavily with their amount of time. An increased test coverage leads to testing more features and higher quality applications.

6. Reusability of Automated Tests

Due to the repetitive nature of test automation test cases, in addition to the relatively easy configuration of their setup, software developers can assess program reaction. Automated test cases are reusable and can hence be utilized through different approaches.

7. Earlier Detection of Defects

The documentation of software defects becomes considerably easier for the testing teams. This helps increase the overall development speed while ensuring correct functionality across areas. The earlier a defect is identified, the more cost-effective it is to fix the glitch.

8. Thoroughness in Testing

Testers tend to have different testing approaches, and their focus areas could vary due to their exposure and expertise. With the inclusion of automation, there is a guaranteed focus on all areas of testing, thereby assuring best possible quality.

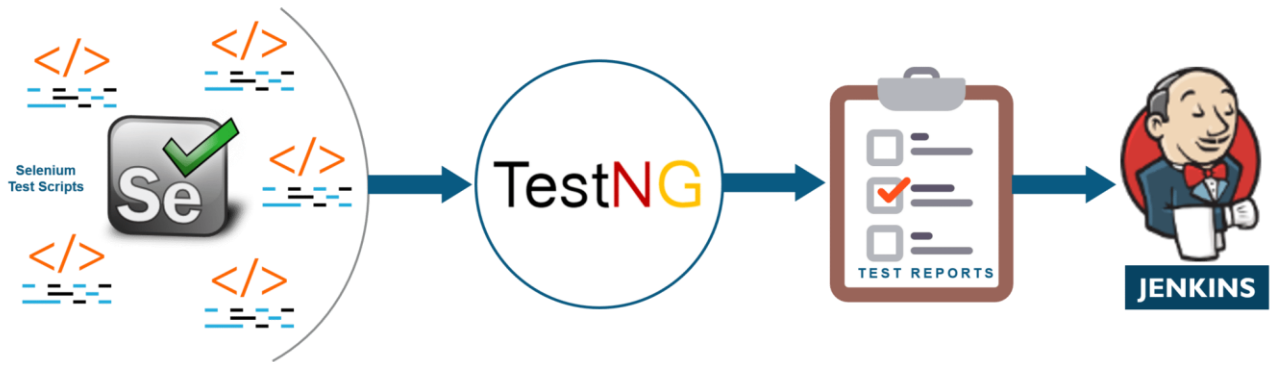
9. Faster Time-to-Market

Test Automation greatly helps reduce the time-to-market of an application by allowing constant execution of test cases. Once automated, the test library execution is faster and runs longer than manual testing.

10. Information Security

The effectiveness of testing will be largely dependent on the quality of the test data you use. Manually creating quality test data takes time, and as a result, testing is often performed on copies of live databases. Automation solutions can help with creating, manipulating, and protecting your test database, allowing you to re-use your data time and again. The time and cost savings in this area are potentially huge.

Selenium does the automation testing, and the reports are generated by TestNG. But to automate this entire testing phase, the role of continuous integration tools like Jenkins coming into the picture.



Selenium Testing Jenkins

3. Continuous Integration

In Continuous Integration after a code commit, the software is built and tested immediately. In a large project with many developers, commits are made many times during a day. With each commit code is built and tested. If the test is passed, build is tested for deployment. If deployment is a success, the code is pushed to production. This commit, build, test, and deploy is a continuous process and hence the name continuous integration/deployment.

A Continuous Integration Pipeline is a powerful instrument that consists of a set of tools designed to host, monitor, compile and test code, or code changes, like:

- Continuous Integration Server (Jenkins)
- Source Control Tool (e.g., CVS, SVN, GIT)
- Build tool (Maven)
- Automation testing framework (Selenium, Appium)

4. Continuous Deployment

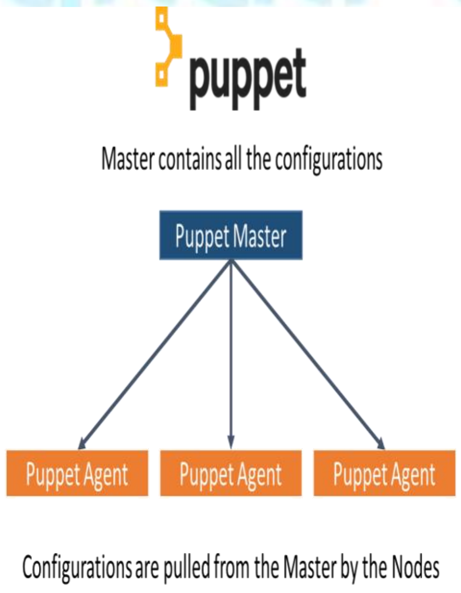
Continuous Deployment is the phase where the action happens. Continuous delivery (CD) is a set of practices outlined to make sure that code can be swiftly and safely deployed to production by providing every change to a production- like environment and ensuring business applications and services function as expected through meticulous automated testing.

Benefits

- We can have multiple versions of an Application. Faster build and deployment of Applications.
- An application can be tested quickly once deployed.
- Entire Workflow is automated from building an image using docker and deploying it on kubernetes.

Configuration Management Tools

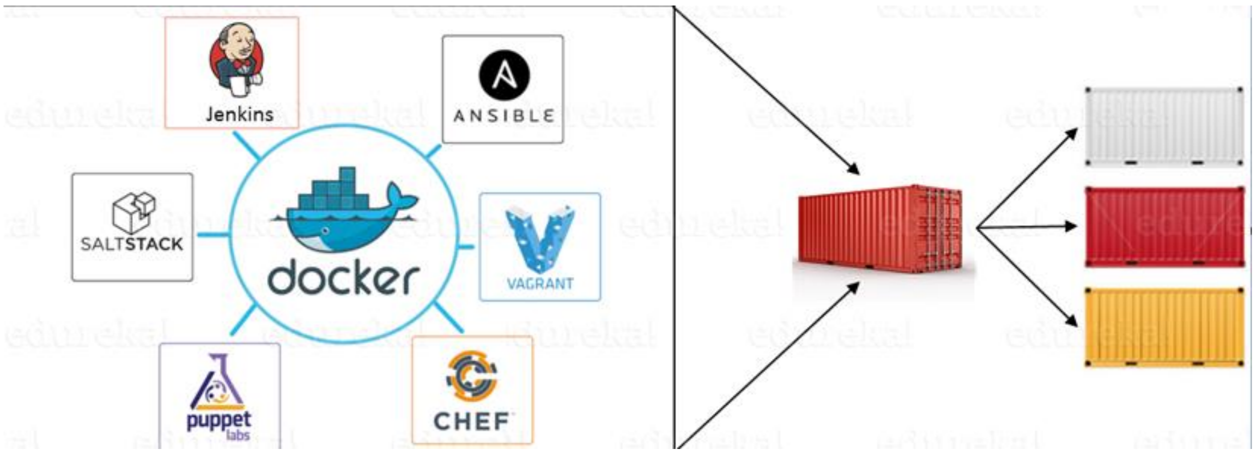
- Configuration Management is the act of establishing and maintaining consistency in an applications' functional requirements and performance. In simpler words, it is the act of releasing deployments to servers, scheduling updates on all servers and most importantly keeping the configurations consistent across all the servers.
- For this, we have tools like Puppet, Chef, Ansible, and more. Puppet and the other CM tools work based on the master-slave architecture. When there is a deployment sent to the master, the master is responsible for replicating those changes across all the slaves.


puppet
Master contains all the configurations
Puppet Master
Puppet Agent
Puppet Agent
Puppet Agent
Configurations are pulled from the Master by the Nodes

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

The Docker, which is a containerization tool. Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), can handle more applications and require fewer VMs and Operating systems.



Docker Integrations

5. Continuous Monitoring

Monitoring is as important as developing the application because there will always be a chance of bugs that escape undetected during the testing phase.

Splunk, ELK Stack, Nagios, and NewRelic are some of the popular tools for monitoring. When used in combination with Jenkins, we achieve continuous monitoring.

Splunk is a propriety tool. But this also effectively means that working on Splunk is very easy. The ELK stack, however, is a combination of three open-source tools: ElasticSearch, LogStash, and Kibana.