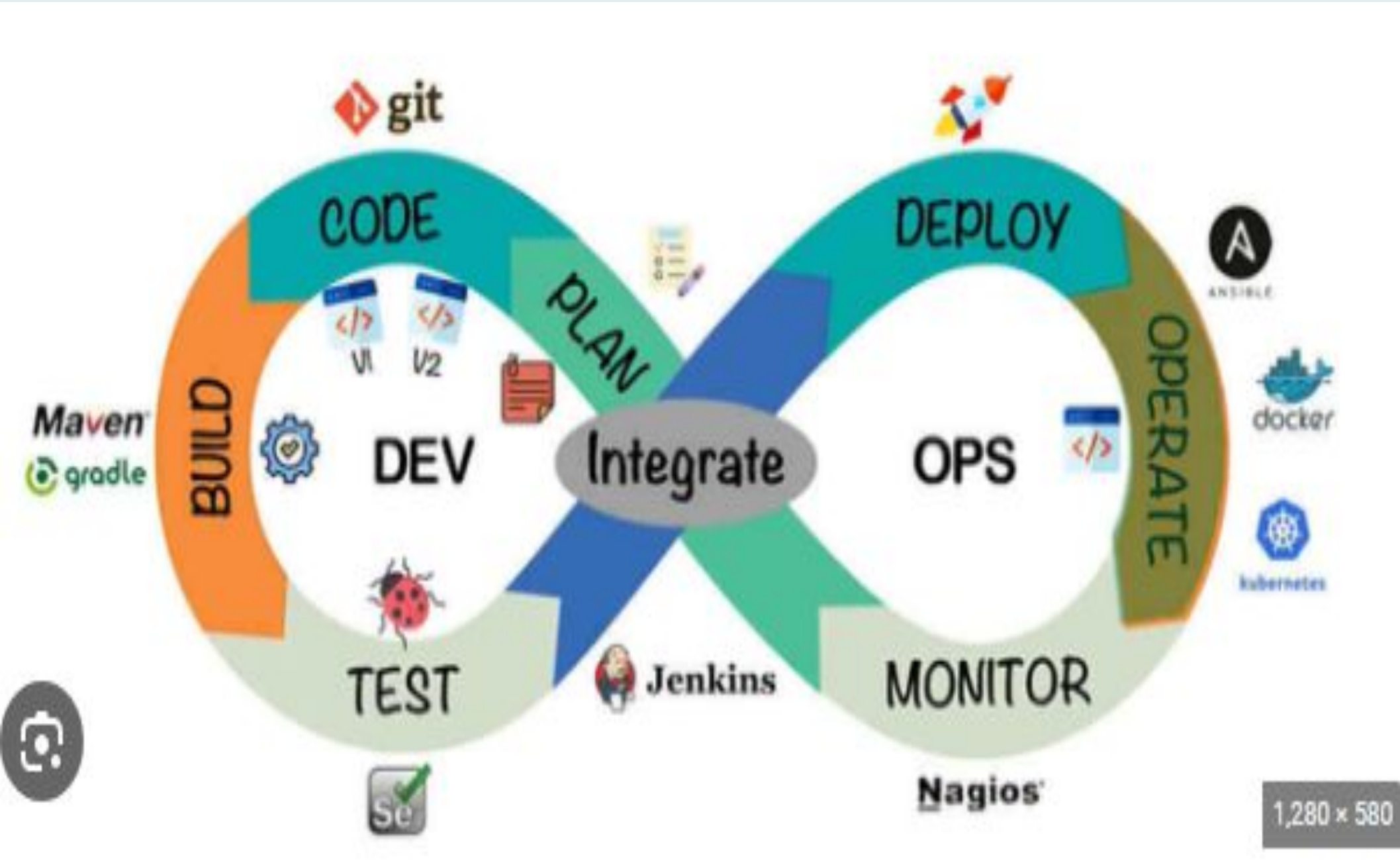
U20ITCM08
Automation Techniques and Tools-DevOps

Mrs.N.Kalaiselvi, Assistant Professor, Department of IT,SMVEC



Course Outcomes

- CO1 - Explains about traditional software methodologies and about software project estimation, roles of developers and IT operations conflicts (K2)
- CO2 - Realize the importance of agile software development practices in determining the requirements for a software system and about agile manifesto, values and principles (K3)
- CO3 - Provides basic ideas of devops and its role in terms of version control, automated testing, continuous integration and delivery (K3)
- CO4 - Illustrates the purposes of devops with MVP, continuous integration and delivery (K2)
- CO5 - Explains the role of CAMS in devops (K3)

Syllabus

- **UNIT I- TRADITIONAL SOFTWARE DEVELOPMENT**

  The Advent of Software Engineering - Software Process, Perspective and Specialized Process Models - Software Project Management: Estimation - Developers vs IT Operations conflict.

- **UNIT II- RISE OF AGILE METHODOLOGIES**

  Agile movement in 2000 - Agile Vs Waterfall Method - Iterative Agile Software Development - Individual and team interactions over processes and tools – Working software over comprehensive documentation -Customer collaboration over contract negotiation - Responding to change over following a plan

- **UNIT III- INTRODUCTION DEVOPS**

  Introduction to DevOps - Version control - Automated testing - Continuous integration - Continuous delivery -Deployment pipeline - Infrastructure management – Databases

- **UNIT IV – PURPOSE OF DEVOPS**

  Minimum Viable Product- Application Deployment- Continuous Integration- Continuous Delivery

- **UNIT V – CAMS (CULTURE, AUTOMATION, MEASUREMENT AND SHARING)**

  CAMS – Culture, CAMS – Automation, CAMS – Measurement, CAMS – Sharing, Test-Driven Development, Configuration Management-Infrastructure Automation- Root Cause Analysis- Blamelessness- Organizational Learning

Text Books and References

- **Text Books:**
  - Dev Ops – Volume 1 , Pearson and Xebia Press
  - Grig Gheorghiu, Alfredo Deza, Kennedy Behrman, Noah Gift, Python for DevOps,2019
- **Reference Books:**
  - The DevOps Handbook - Book by Gene Kim, Jez Humble, Patrick Debois, and Willis Willis
  - What is DevOps? - by Mike Loukides
  - Joakim Verona, Practical DevOps ,2016.
- **Websites:**
  - www.ibm.com/cloud/devops.
  - www.softwaretestinghelp.com>devops-automation.

UNIT III

UNIT III- INTRODUCTION DEVOPS

Introduction to DevOps - Version control - Automated testing - Continuous integration - Continuous delivery -Deployment pipeline - Infrastructure management – Databases

Introduction to DevOps

- DevOps is an evolving philosophy and framework that encourages faster, better application development and faster release of new or revised software features or products to customers.

- The practice of DevOps encourages smoother, continuous communication, collaboration, integration, visibility, and transparency between application development teams (Dev) and their IT operations team (Ops) counterparts.

- What is DevOps?
- DevOps is a collaboration between development and operation teams, which enables continuous delivery of applications and services to our end users.

History of DevOps

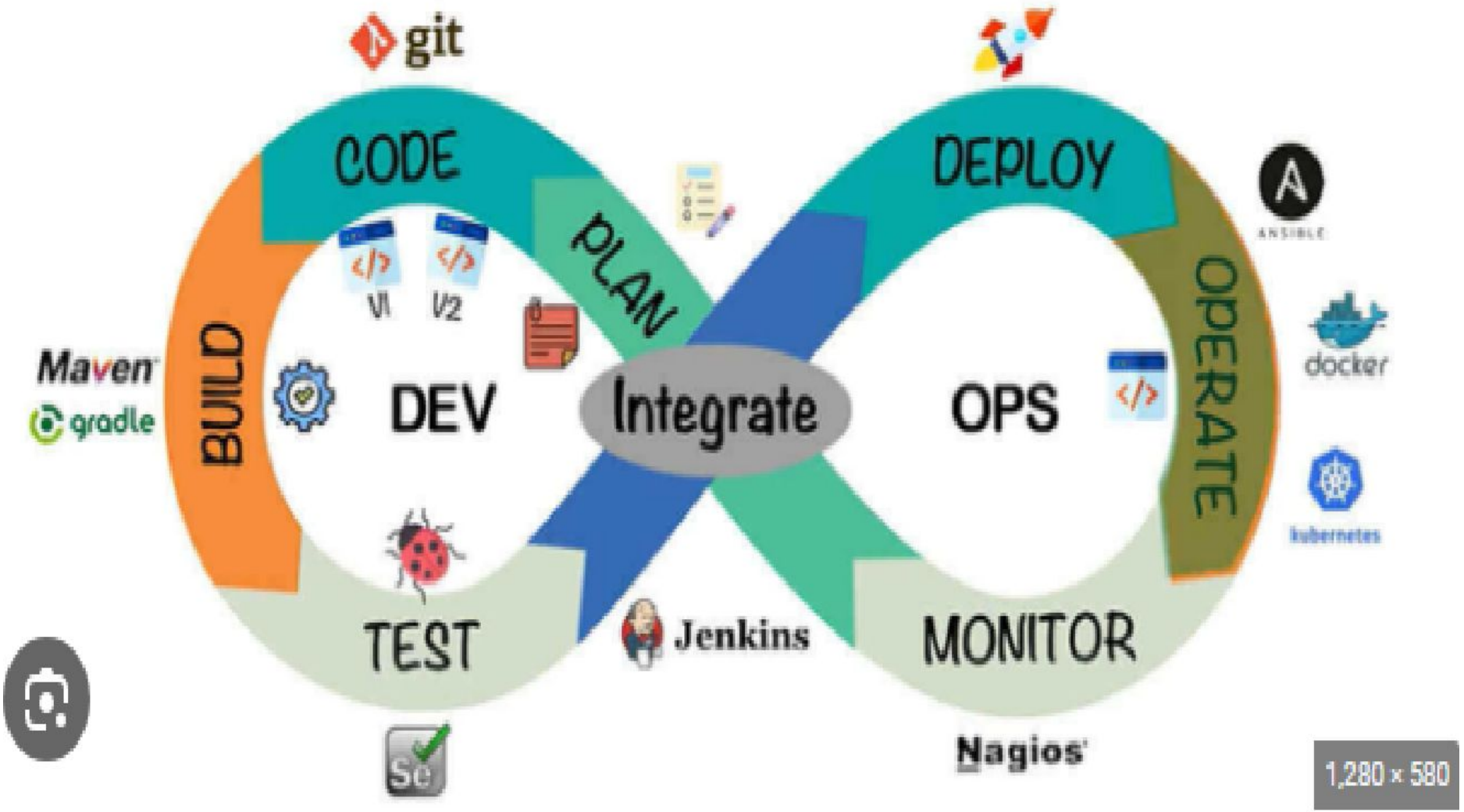
- The early 2000s saw the need to maintain availability of popular websites such as Google and Flickr against massive hits. This need led to the use of software reliability engineers (SREs)—operations people working closely with developers to ensure that the sites would keep running after code was released into production.
- In 2009, Flickr engineers John Allspaw and Paul Hammond presented their own DevOps-like methodology at a conference. Their presentation was entitled "10+ Deploys per Day: Dev and Ops Cooperation at Flickr." The same year, Patrick Debois organized the first "DevOps Day" in Belgium. A #DevOps hashtag was also incorporated and gained momentum as more DevOps Days were held around the world.
- Over the coming years, industry and open-source tools and frameworks were developed and proposed to further the goals of DevOps.

Benefits of DevOps

- Speed
  - Move at high velocity so you can innovate for customers faster, adapt to changing markets better, and grow more efficient at driving business results. The DevOps model enables your developers and operations teams to achieve these results. For example, microservices and continuous delivery let teams take ownership of services and then release updates to them quicker.
- Rapid Delivery
  - Increase the frequency and pace of releases so you can innovate and improve your product faster. The quicker you can release new features and fix bugs, the faster you can respond to your customers' needs and build competitive advantage. Continuous integration and continuous delivery are practices that automate the software release process, from build to deploy.
- Reliability
  - Ensure the quality of application updates and infrastructure changes so you can reliably deliver at a more rapid pace while maintaining a positive experience for end users. Use practices like continuous integration and continuous delivery to test that each change is functional and safe. Monitoring and logging practices help you stay informed of performance in real-time.
- Scale
  - Operate and manage your infrastructure and development processes at scale. Automation and consistency help you manage complex or changing systems efficiently and with reduced risk. For example, infrastructure as code helps you manage your development, testing, and production environments in a repeatable and more efficient manner.
- Improved Collaboration
  - Build more effective teams under a DevOps cultural model, which emphasizes values such as ownership and accountability. Developers and operations teams collaborate closely, share many responsibilities, and combine their workflows. This reduces inefficiencies and saves time (e.g. reduced handover periods between developers and operations, writing code that takes into account the environment in which it is run).
- Security
  - Move quickly while retaining control and preserving compliance. You can adopt a DevOps model without sacrificing security by using automated compliance policies, fine-grained controls, and configuration management techniques. For example, using infrastructure as code and policy as code, you can define and then track compliance at scale.

DevOps practices

- **Continuous development.** This practice spans the planning and coding phases of the DevOps lifecycle. Version-control mechanisms might be involved.
- **Continuous testing.** This practice incorporates automated, prescheduled, continued code tests as application code is being written or updated. Such tests can speed the delivery of code to production.
- **Continuous integration (CI).** This practice brings configuration management (CM) tools together with other test and development tools to track how much of the code being developed is ready for production. It involves rapid feedback between testing and development to quickly identify and resolve code issues.
- **Continuous delivery.** This practice automates the delivery of code changes, after testing, to a preproduction or staging environment. An staff member might then decide to promote such code changes into production.


git
CODE
PLAN
BUILD
V1
V2
Maven
gradle
DEV
Integrate
OPS
DEPLOY
OPERATE
ANSIBLE
docker
kubernetes
TEST
Jenkins
MONITOR
Nagios
1,280 × 580

DevOps practices

- **Continuous deployment (CD).** Similar to continuous delivery, this practice automates the release of new or changed code into production. A company doing continuous deployment might release code or feature changes several times per day. The use of container technologies, such as Docker and Kubernetes, can enable continuous deployment by helping to maintain consistency of the code across different deployment platforms and environments.
- **Continuous monitoring.** This practice involves ongoing monitoring of both the code in operation and the underlying infrastructure that supports it. A feedback loop that reports on bugs or issues then makes its way back to development.
- **Infrastructure as code.** This practice can be used during various DevOps phases to automate the provisioning of infrastructure required for a software release. Developers add infrastructure "code" from within their existing development tools. For example, developers might create a storage volume on demand from Docker, Kubernetes, or OpenShift. This practice also allows operations teams to monitor environment configurations, track changes, and simplify the rollback of configurations.

DevOps methods

- **Scrum.** Scrum defines how members of a team should work together to accelerate development and QA projects. Scrum practices include key workflows and specific terminology (sprints, time boxes, daily scrum [meeting]), and designated roles (Scrum Master, product owner).
- **Kanban.** Kanban originated from efficiencies gained on the Toyota factory floor. Kanban prescribes that the state of software project work in progress (WIP) be tracked on a Kanban board.
- **Agile.** Earlier agile software development methods continue to heavily influence DevOps practices and tools. Many DevOps methods, including Scrum and Kanban, incorporate elements of agile programming. Some agile practices are associated with greater responsiveness to changing needs and requirements, documenting requirements as user stories, performing daily standups, and incorporating continuous customer feedback. Agile also prescribes shorter software development lifecycles instead of lengthy, traditional "waterfall" development methods.

Why did we need DevOps?

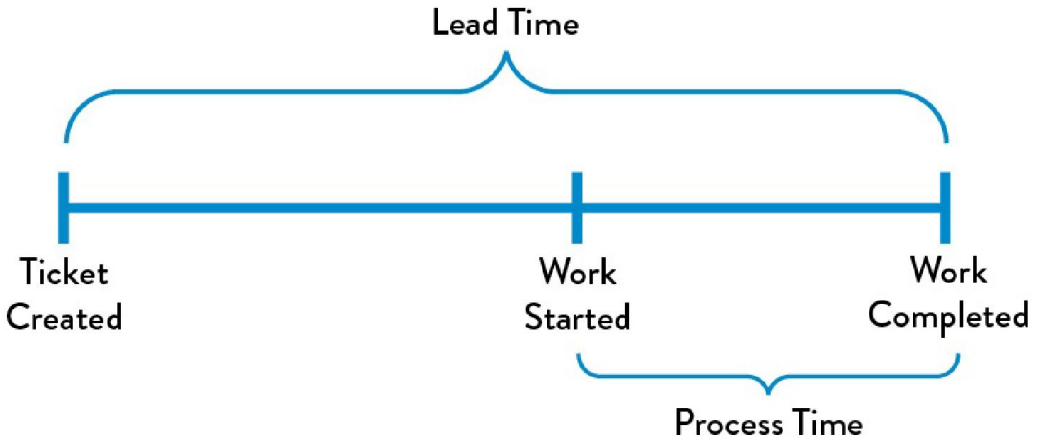
- As we know about the problems faced in traditional models like in the waterfall model, there is a problem of a one-way stream of work, due to which if there is any mistake, the whole process repeats, and there is no interaction with customers.

- Now, this is solved in agile by splitting the whole development plan into several iterations for better production efficiency. The agile model also includes customer interaction with the company to rectify mistakes. But there is another problem faced in Agile too.

- Here, the problem arises when the development team continuously changes the code for better performance and sends the code to the operations team for testing. But there may be a delay in the operations team feedback in situations like if the developers sent code for review at night but due to the unavailability of the operations team, there will be a delay in the project feedback.

- So, DevOps is the solution to this problem. DevOps is a practice or a methodology in which the development and operations teams work together by including automation at the initial stages. So they can work on rapidly changing systems, fix bugs, and help to deliver a good quality of software in time.

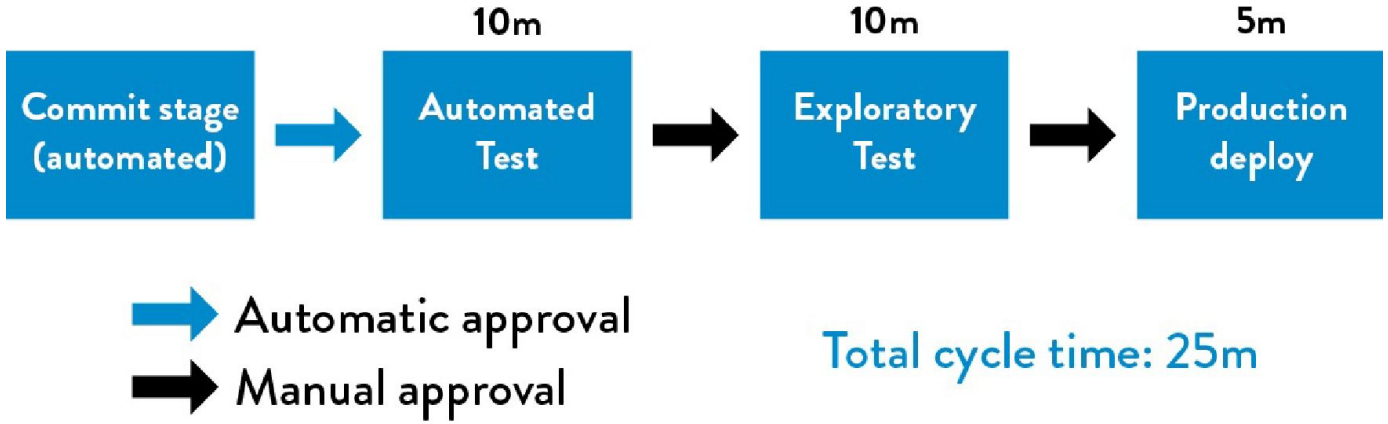
Need for DevOps



We often find ourselves in situations where our deployment lead times require months. This is especially common in large, complex organizations that are working with tightly-coupled, monolithic applications, often with scarce integration test environments, long test and production environment lead times, high reliance on manual testing, and multiple required approval processes.

We may discover that nothing works at the end of the project when we merge all the development team's changes together, resulting in code that no longer builds correctly or passes any of our tests. Fixing each problem requires days or weeks of investigation to determine who broke the code and how it can be fixed, and still results in poor customer outcomes.

DevOps ideal



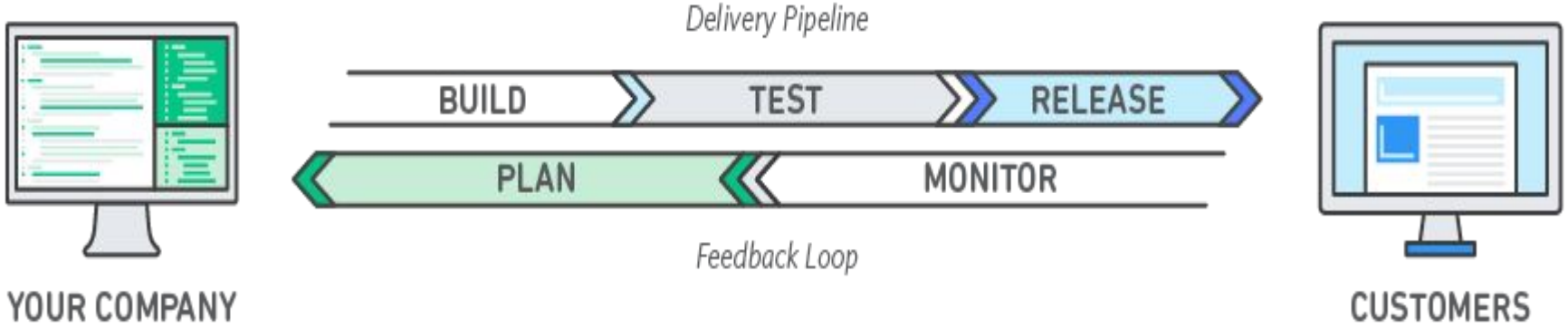
In the DevOps ideal, developers receive fast, constant feedback on their work, which enables them to quickly and independently implement, integrate, and validate their code, and have the code deployed into the production environment .

This is most easily achieved when we have architecture that is modular, well encapsulated, and loosely-coupled so that small teams are able to work with high degrees of autonomy, with failures being small and contained, and without causing global disruptions.

In this scenario, our deployment lead time is measured in minutes, or, in the worst case, hours.

DevOps Architecture:

- The phases of Devops architecture are



- **Plan** – In DevOps planning plays an important role. In this stage, all the requirements of the project and everything regarding the project like time for each stage, cost. etc are discussed. This will help everyone in teams to get a brief idea about the project.
- **Code** – In this Stage the code is written over here according to the client's requirements. Here the code is divided into small codes called Units. This is done to get a clear picture of the code. For example, if the team is doing a project on an online -Ekart application then the login part is divided as one unit, after login the page which shows all the categories is divided as another unit, user profile as another unit, etc.
- Some of the examples of the tools used are **Git, JIRA**

- **Build** – In this stage Building of the units is done. Some of the examples of the tools used are **maven, Gradle**.
- **Test** – Testing of all units is done in this stage. So we will get to know where exactly the code is having bugs and if there are mistakes found it is returned. Some of the examples of the tools used are **Selenium, PYtest**
- **Integrate** – In this stage, all the units of the codes are integrated. That means in this step we will be creating a connection between the development team and the operation team to implement Continuous Integration and Continuous Deployment. An example of the tool used is **Jenkins.**
- **Deploy** – In this stage, the code is deployed on the client's environment. Some of the examples of the tools used are **AWS, Docker.**
- **Operate** – Operations are performed on the code if required. Some of the examples of the tools used are **Kubernetes, open shift**.
- **Monitor** – In this stage monitoring of the application is done over here in the client's environment. Some of the examples of the tools used are **Nagios, elastic stack**.

How is DevOps different from Agile?

DevOps	Agile
DevOps deals with filling the time gap between the development team and the operations team.	Agile methodology deals with filling the gap between customers and the company.
Here the feedback will be coming from the Operations team to the development team.	Here the feedback will be coming from the customers to the company.
It focuses on constant testing and delivery.	It focuses on constant changes.
Some of the tools used in DevOps: Puppet, AWS	Some of the tools used for Agile: JIRA, Bugzilla, Kanboard

Applications of DevOps

- United Airlines
- KeyBank
- HP
- Rabobank
- Amazon
- Netflix
- Adobe

Advantages and Disadvantages of DevOps:

Advantages:

- Faster development of software and quick deliveries.
- DevOps is flexible and adaptable to changes easily.
- Compared to the previous software development models confusion about the project is decreased due to which the product quality and efficiency are increased.
- The gap between the development team and operation team was bridged. i.e, the communication between the teams has been increased.
- Efficiency is increased by the addition of automation which includes continuous integration and continuous deployment.
- Customer satisfaction is enhanced.

Disadvantages:

- DevOps is expensive.
- Certain levels of skills are required for maintaining the DevOps architecture.
- Adopting DevOps technology into the traditional style of industries is quite a challenge.

Version control

Version control

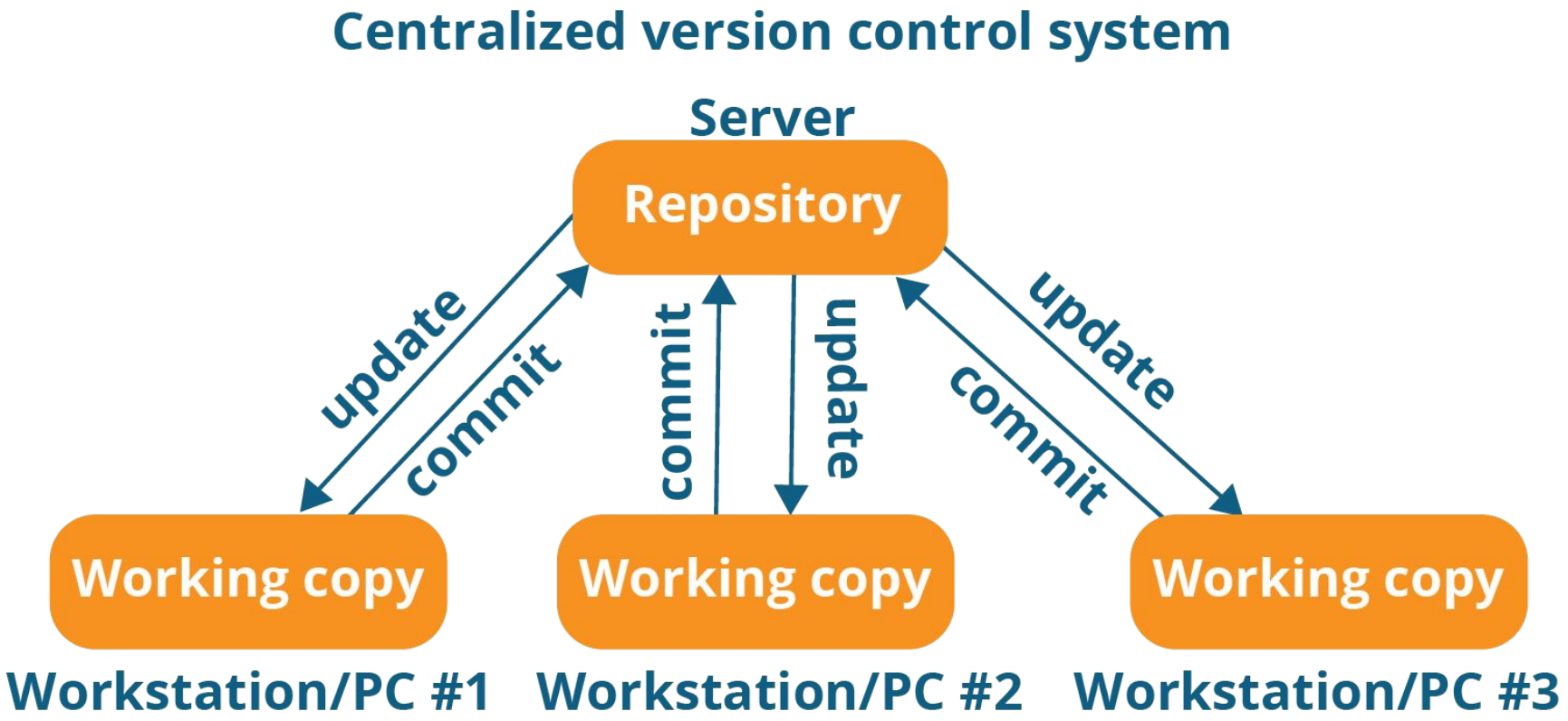
- Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage changes to source code over time.
- Why use version control?
- As organizations accelerate delivery of their software solutions through DevOps, controlling and managing different versions of application artifacts — from code to configuration and from design to deployment — becomes increasingly difficult.
- Version control software facilitates coordination, sharing, and collaboration across the entire software development team. It enables teams to work in distributed and asynchronous environments, manage changes and versions of code and artifacts, and resolve merge conflicts and related anomalies.

Git

- **What is the purpose of Git?**
- Git is primarily used to manage your project, comprising a set of code/text files that may change.
- **There are two types of VCS:**
- Centralized Version Control System (CVCS)
- Distributed Version Control System (DVCS)

Centralized VCS

- Centralized version control system (CVCS) uses a central server to store all files and enables team collaboration. It works on a single repository to which users can directly access a central server.
- Please refer to the diagram below to get a better idea of CVCS:

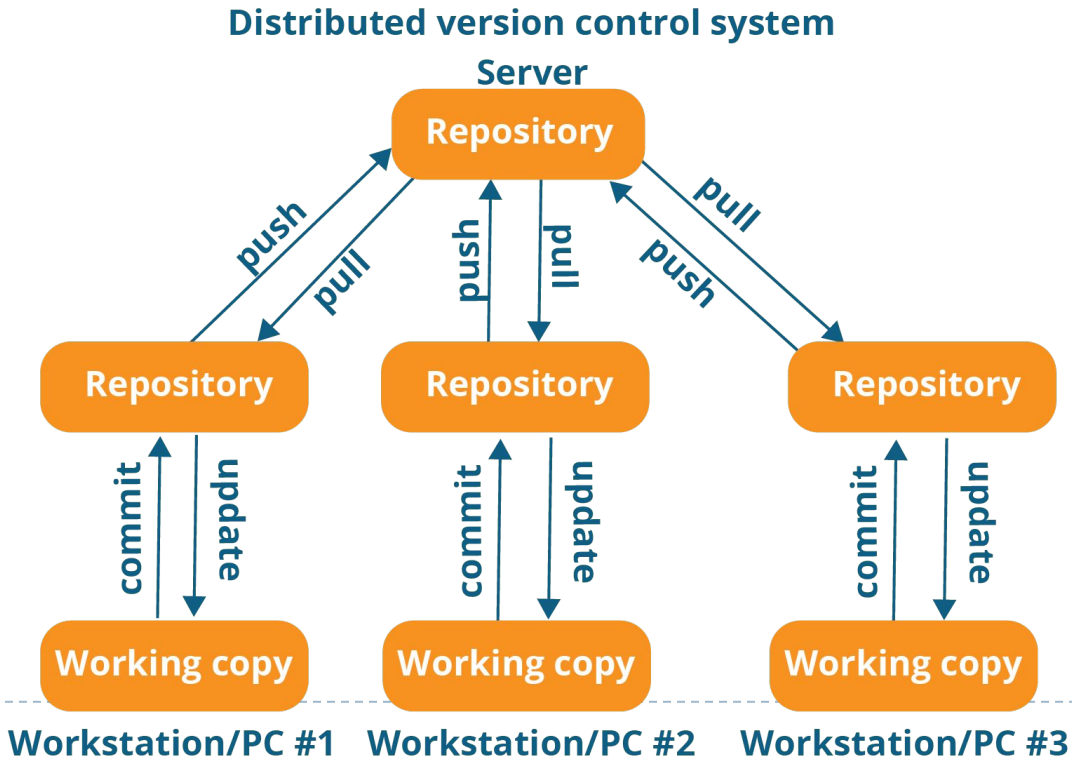


Centralized VCS

- The repository in the above diagram indicates a central server that could be local or remote which is directly connected to each of the programmer's workstation.
- Every programmer can extract or **update** their workstations with the data present in the repository or can make changes to the data or **commit** in the repository. Every operation is performed directly on the repository.
- Even though it seems pretty convenient to maintain a single repository, it has some major drawbacks. Some of them are:
- It is not locally available; meaning you always need to be connected to a network to perform any action.
- Since everything is centralized, in any case of the central server getting crashed or corrupted will result in losing the entire data of the project.
- This is when Distributed VCS comes to the rescue.

Distributed VCS

- These systems do not necessarily rely on a central server to store all the versions of a project file.
- In Distributed VCS, every contributor has a local copy or "clone" of the main repository i.e. everyone maintains a local repository of their own which contains all the files and metadata present in the main repository.
- You will understand it better by referring to the diagram below:



- As you can see in the above diagram, every programmer maintains a local repository on its own, which is actually the copy or clone of the central repository on their hard drive. They can commit and update their local repository without any interference.
- They can update their local repositories with new data from the central server by an operation called "**pull**" and affect changes to the main repository by an operation called "**push**" from their local repository.
- The act of cloning an entire repository into your workstation to get a local repository gives you the following advantages:
- All operations (except push & pull) are very fast because the tool only needs to access the hard drive, not a remote server. Hence, you do not always need an internet connection.
- Committing new change-sets can be done locally without manipulating the data on the main repository. Once you have a group of change-sets ready, you can push them all at once.
- Since every contributor has a full copy of the project repository, they can share changes with one another if they want to get some feedback before affecting changes in the main repository.
- If the central server gets crashed at any point of time, the lost data can be easily recovered from any one of the contributor's local repositories.

What Is Git?

- Git is a Distributed Version Control tool that supports distributed non-linear workflows by providing data assurance for developing quality software.

- Git provides with all the Distributed VCS facilities to the user that was mentioned earlier. Git repositories are very easy to find and access.

GIT

- **Free and open source:**
- Git is released under GPL's (General Public License) open source license. You don't need to purchase Git. It is absolutely free. And since it is open source, you can modify the source code as per your requirement.
- **Speed:**
- Since you do not have to connect to any network for performing all operations, it completes all the tasks really fast.
- Performance tests done by Mozilla showed it was an order of magnitude faster than other version control systems.
- Fetching version history from a locally stored repository can be one hundred times faster than fetching it from the remote server.
- The core part of Git is written in C, which avoids runtime overheads associated with other high level languages.

- **Scalable:**
- Git is very scalable. So, if in future , the number of collaborators increase Git can easily handle this change. Though Git represents an entire repository, the data stored on the client's side is very small as Git compresses all the huge data through a lossless compression technique.

- **Reliable:**
- Since every contributor has its own local repository, on the events of a system crash, the lost data can be recovered from any of the local repositories. You will always have a backup of all your files.

- **Secure:**
- Git uses the ***SHA1*** (Secure Hash Function) to name and identify objects within its repository. Every file and commit is check-summed and retrieved by its checksum at the time of checkout. The Git history is stored in such a way that the ID of a particular version (a *commit* in Git terms) depends upon the complete development history leading up to that commit. Once it is published, it is not possible to change the old versions without it being noticed.
- **Economical:**
- In case of CVCS, the central server needs to be powerful enough to serve requests of the entire team. For smaller teams, it is not an issue, but as the team size grows, the hardware limitations of the server can be a performance bottleneck.
- In case of DVCS, developers don't interact with the server unless they need to push or pull changes. All the heavy lifting happens on the client side, so the server hardware can be very simple indeed.

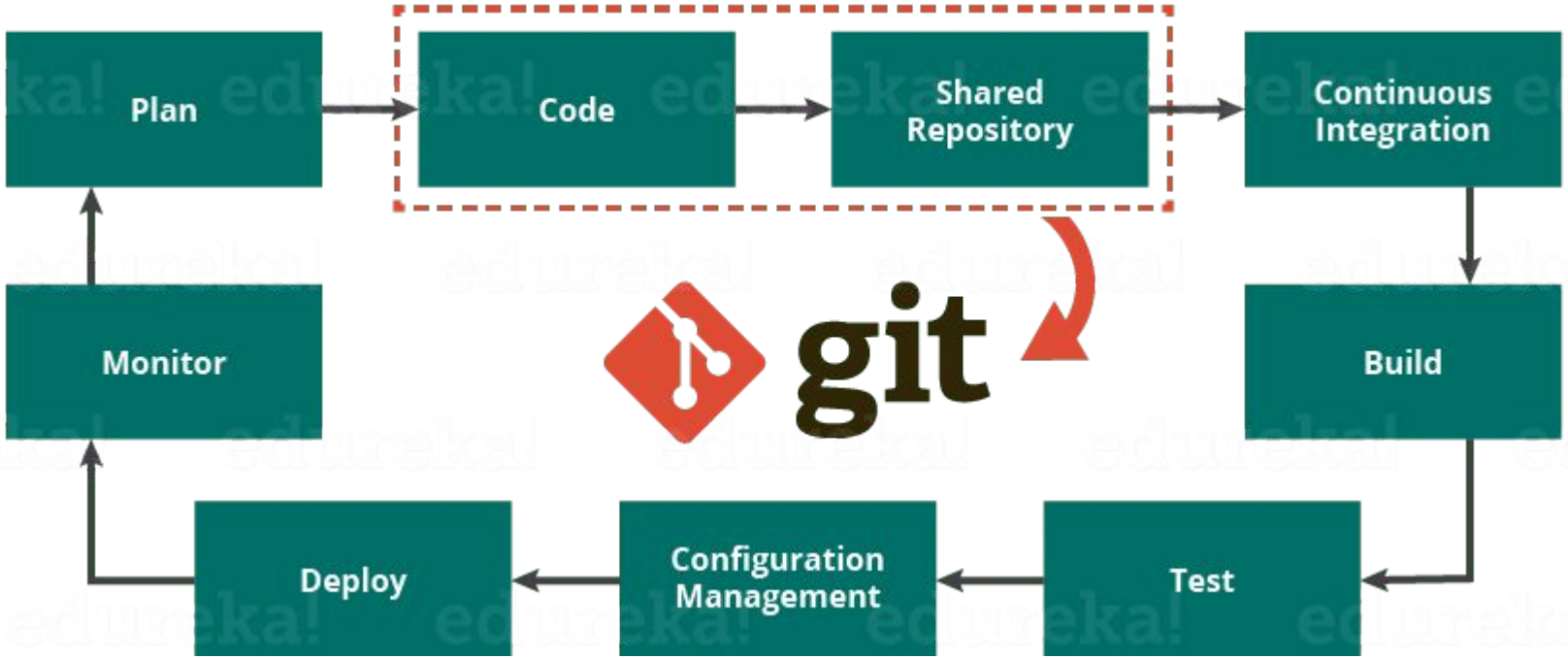
- **Supports non-linear development:**
- Git supports rapid branching and merging, and includes specific tools for visualizing and navigating a non-linear development history. A core assumption in Git is that a change will be merged more often than it is written, as it is passed around various reviewers. Branches in Git are very lightweight. A branch in Git is only a reference to a single commit. With its parental commits, the full branch structure can be constructed.
- **Easy Branching:**
- Branch management with Git is very simple. It takes only few seconds to create, delete, and merge branches. Feature branches provide an isolated environment for every change to your codebase. When a developer wants to start working on something, no matter how big or small, they create a new branch. This ensures that the master branch always contains production-quality code.

- **Distributed development:**
- Git gives each developer a local copy of the entire development history, and changes are copied from one such repository to another. These changes are imported as additional development branches, and can be merged in the same way as a locally developed branch.

- **Compatibility with existing systems or protocol**
- Repositories can be published via http, ftp or a Git protocol over either a plain socket, or ssh. Git also has a Concurrent Version Systems (CVS) server emulation, which enables the use of existing CVS clients and IDE plugins to access Git repositories. Apache SubVersion (SVN) and SVK repositories can be used directly with Git-SVN.

What is Git – Role Of Git In DevOps?

- DevOps is the practice of bringing agility to the process of development and operations. It's an entirely new ideology that has swept IT organizations worldwide, boosting project life-cycles and in turn increasing profits.
- DevOps promotes communication between development engineers and operations, participating together in the entire service life-cycle, from design through the development process to production support.



- The diagram above shows the entire life cycle of Devops starting from planning the project to its deployment and monitoring. Git plays a vital role when it comes to managing the code that the collaborators contribute to the shared repository. This code is then extracted for performing continuous integration to create a build and test it on the test server and eventually deploy it on the production.
- Tools like Git enable communication between the development and the operations team. When you are developing a large project with a huge number of collaborators, it is very important to have communication between the collaborators while making changes in the project. Commit messages in Git play a very important role in communicating among the team. The bits and pieces that we all deploy lies in the Version Control system like Git. To succeed in DevOps, you need to have all of the communication in Version Control. Hence, Git plays a vital role in succeeding at DevOps.

Automated Testing

- The prime objective of any software project is to get a high-quality output while reducing the cost and the time required for completing the project. This is achieved by testing software regularly.
- Software Testing is an integral part of any IT project. The software is tested to detect bugs and find issues that may negatively affect the user experience.
- Testing is mainly classified as Manual Testing and Automation Testing. Through this Automation Testing tutorial, we will learn all about test automation.
- **What is Automation Testing?**
- *Automation Testing is the process of using the assistance of tools, scripts, and software to perform test cases by repeating pre-defined actions. Test Automation focuses on replacing manual human activity with systems or devices that enhance efficiency.*

- Testing is crucial to the success of any software product. If your software doesn't work properly, chances are that most people won't even buy or use your software product, if they did at least not for long.
- But testing to find defects or bugs manually is time-consuming, expensive, often repetitive, and subject to human error. This is where automation comes into the picture.
- Automation is essential for software development teams to keep pace with the rising demands for higher-quality software at lightning speed.
- When you begin testing, one of the primary decisions that you'll have to make is if you're going to test manually or use automated testing. So, you should be aware of the distinct differences between manual testing and automated testing.

- Neither of these options are technically better or worse than the other. But the size, budget and time allowance of a project will certainly be determining factors that affect which method will work best in your testing procedure.

- Manual testing allows a human mind to draw insights from a test that might otherwise be missed by an automated testing program.
- While automated testing is well-suited for large projects that require testing the same areas over and over again. But can we automate every test case?

Which test cases to automate?

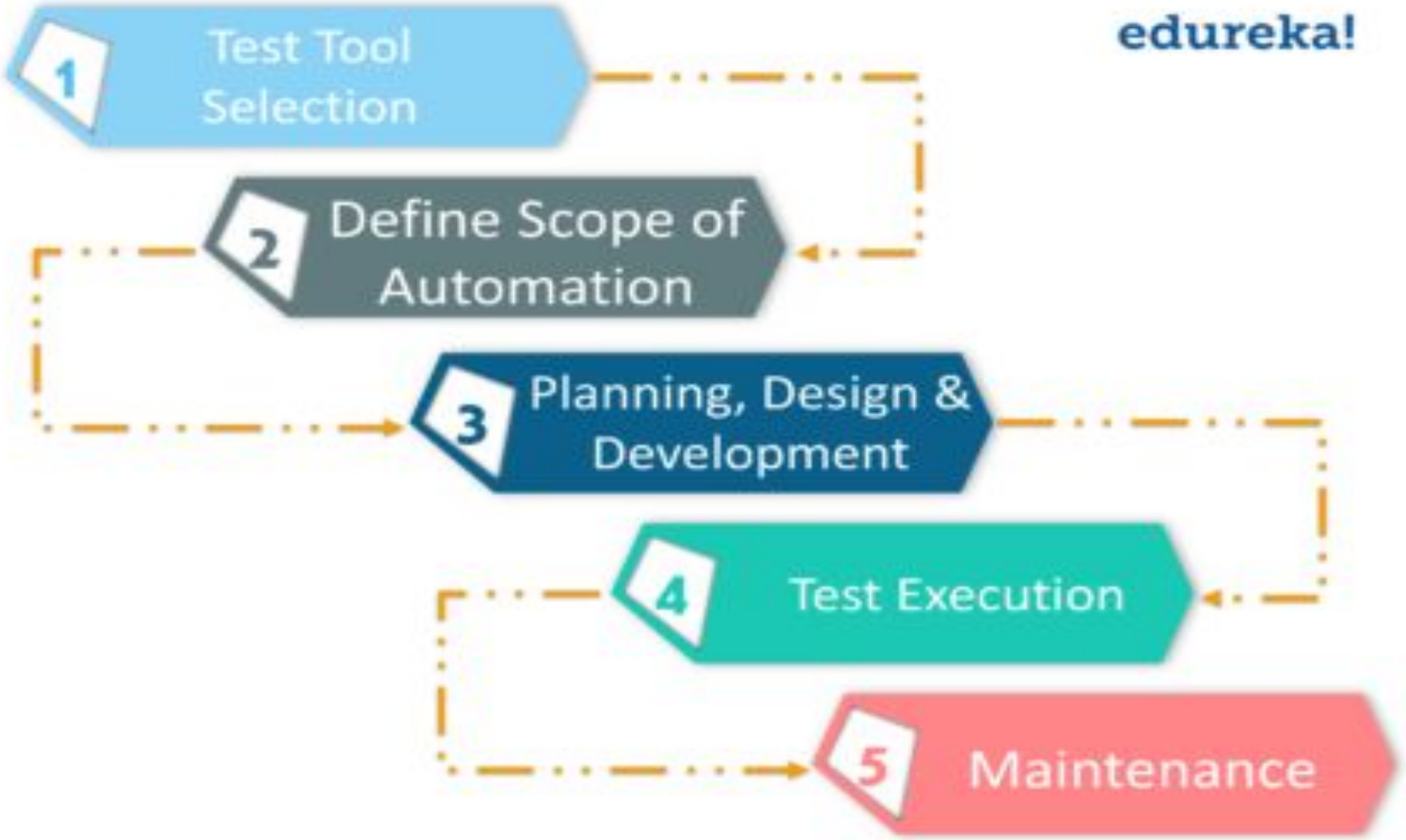
- Well, it's impossible to automate all testing, so it is important to determine which test cases should be automated. Some test cases where you can apply automation are:
  - Repetitive tasks are prime candidates for automation. Not only are these tasks boring to you, but they're often the ones where you make mistakes.
  - Rather than manually exporting your data, crunching the numbers, and making complex graphs by yourself, invest in a tool or automation strategy that will do this for you.
  - You can apply automation for the tests that require multiple data sets. Instead of manually typing in information into forms, automate the process.
  - Another good case where you can apply automation is load testing.
  - You can apply automation for test cases that run on several different hardware or software platforms and configurations.

Difference between Manual Testing and Automation Testing

Features	Manual Testing	Automation Testing
Accuracy & Reliability	Low, as manual tests are more prone to human error	High, as tools and scripts are used
Time required	High	Relatively Low
Investment Cost	Low, Return of Investment(ROI) is low	High, Return of Investment(ROI) is high
Usage	Suitable for Exploratory, Usability and Ad hoc Testing	Suitable for Regression Testing, Performance Testing, Load Testing
Human Element	Allows for human observation to find out any glitches	No human observation involved
Customer Experience	Helps in improving the customer experience	No guarantee of positive customer experience

How do you perform Automation Testing?

- Success in test automation requires careful planning and design work. The following steps are followed in an automation process:



Test tool selection

- Any process starts with the definition, so before applying to test automation you should define the goal of automation. Once you are sure of what kind of tests are you performing, you need to select the tool.
- There are several kinds of testing tools available, however, choosing the right tool that suits your test requirements, is very important for automation.
- Consider these key pointers while selecting an automation tool:
  - Is it easy to develop and maintain the scripts for the tool or not?
  - Does it work on platforms like web, mobile, desktop etc?
  - Does the tool have a test reporting functionality?
  - How many testing types can this tool support?
  - How many languages does the tool support?

- **Define the scope of automation**
- Next, you define the scope of automation, as in you need to decide which test cases to automate. Some pointers that you can follow are:
  - Scenarios which have a large amount of data
  - Test cases which have common functionalities across applications
  - Technical feasibility
  - The extent to which business components are reused
  - The complexity of test cases
- **Planning, Design, and Development**
- After determining your goal and which types of tests to automate, you should decide what actions your automated tests will perform. Planning, design, and development include:
- **Developing Test Cases:** Develop test cases of your choice. Don't just create test cases that test various aspects of the application's behavior all at one time. Large, complex automated tests are always very difficult to edit and debug. It is best to divide your tests into several simple, logical and smaller tests.
- **Developing Test Suites:** Develop test suits to hold your test cases. Test suites make sure that the automated test cases run one after another without any manual intervention. This can easily be done by creating a test suite that has multiple test cases, a library and command line tool that runs the test suite.

- **Test Execution**
- Automation Scripts are executed during this phase. Execution can be performed using the automation tool directly or through the test management tool which will invoke the automation tool.
- To get the most out of your automated testing, testing should be started as early as possible and executed as often as needed. The earlier testers get involved in the life cycle of the project the better, and the more you test, the more bugs you find.
- **Maintenance**
- Once test cases are executed, the next step is to create reports so that the actions performed during testing are recorded.
- As new functionalities get added to the software that you are testing with successive cycles, automation scripts need to be added, reviewed and maintained for each release cycle.
- Maintenance becomes necessary to improve the effectiveness of automation.
- So, you can follow these steps when performing Automation Testing to get efficient results. Next comes the automation tools.
- There are several innovative Automation Testing tools but before we discuss that, we need to understand the different kinds of approaches to automation.

Different approaches to Automation Testing

- The three main approaches that you can consider to perform Automation Testing are as follows:
- **Code-Driven:** Here the focus is mainly on test case execution to find out if the various sections of code are performing as per expectations or not. Code-driven testing approach is a popular method used in agile software development.
- **Graphical user interface (GUI) testing:** Applications that have GUIs can be tested using this approach. Testers can record user actions and analyze them any number of times. Test cases can be written in a number of programming languages like C#, Java, Perl, Python etc.

- **Test Automation Framework:** Framework is a set of guidelines used to produce beneficial results of the automated testing activity. It brings together function libraries, test data sources, object details, and other reusable modules. Different types of frameworks include:
  - **Linear Scripting Framework:** Recording and replaying test scripts in a sequential or linear fashion.
  - **Data-driven Framework:** Focused on separating the test scripts logic and the test data from each other.
  - **Keyword-driven Framework:** Based on the keywords specified in the excel sheet test scripting is done and tests are executed.
  - **Modular Testing Framework:** Testers divide the application into multiple modules and create test scripts individually.
  - **Hybrid Testing Framework:** A combination of frameworks to leverage the strengths of each.

Automation Testing Tools

- Selecting an automated testing tool is essential for test automation. There are a lot of automated testing tools on the market, and it is important to choose the automated testing tool that best suits your overall requirements. Consider these key points when selecting an automated testing tool:
  1. Understand your project requirements thoroughly and identify the testing scenarios that you want to automate.
  2. Search for the list of tools that suit your project's requirements.
  3. Identify your budget for the automation tool.
  4. Now compare each tool for key criteria like: is it easy to develop and maintain the scripts for the tool or not, does it work on platforms like web, mobile, desktop etc. Does the tool have a test reporting functionality? How many testing types can this tool support? How many languages does the tool support?
  5. Once you have compared the tools, select the tool which is within your budget. Make sure it gives you more advantages based on the key criteria listed above.
- **Selenium** – Selenium is a popular testing framework to perform web application testing across various browsers and platforms like Windows, Mac, and Linux.
- **Watir** – Watir, pronounced as water, is an open source testing tool made up of Ruby libraries to automate web application testing.
- **Ranorex** – Ranorex is flexible, all in one, GUI testing tool using which you can execute automated tests flawlessly throughout all environments and devices.
- **Appium** – Appium is an Open source mobile test automation software is free and supported by a highly active community of developers and experts.

Continuous Integration

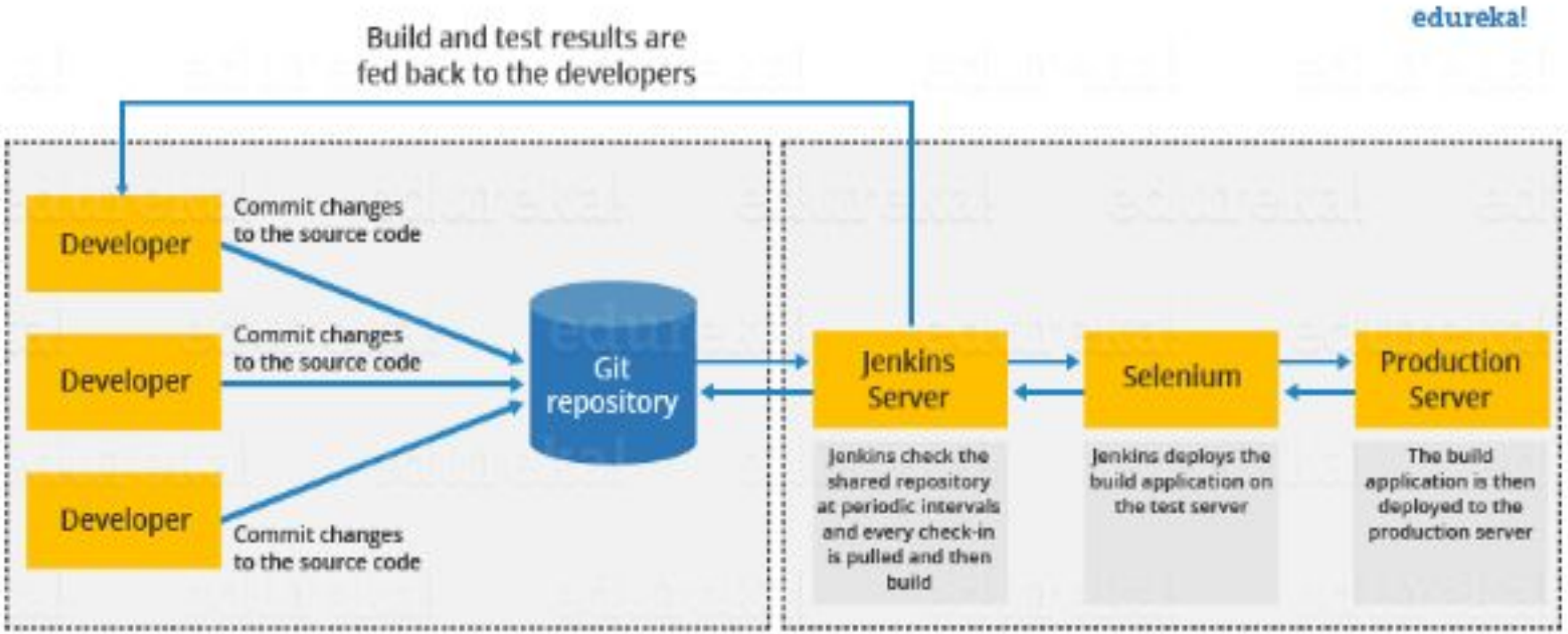
Continuous Integration

- Continuous Integration *(CI)* is a development practice in which the developers are needs to commit changes to the source code in a shared repository at regular intervals. Every commit made in the repository is then built. This allows the development teams to detect the problems early.
- Continuous integration requires the developers to have regular builds. The general practice is that whenever a code commit occurs, a build should be triggered.
- **What is Jenkins?**
- Jenkins is an open source automation tool written in Java programming language that allows continuous integration.
- Jenkins **builds** and **tests** our software projects which continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.
- It also allows us to continuously **deliver** our software by integrating with a large number of testing and deployment technologies.

Continuous Integration with Jenkins

- Let's consider a scenario where the complete source code of the application was built and then deployed on test server for testing. It sounds like a perfect way to *develop software*, but this process has many problems.
  - Developer teams have to wait till the complete software is developed for the test results.
  - There is a high prospect that the test results might show multiple bugs. It was tough for developers to locate those bugs because they have to check the entire source code of the application.
  - It slows the software delivery process.
  - Continuous feedback pertaining to things like architectural or coding issues, build failures, test status and file release uploads was missing due to which the quality of software can go down.
  - The whole process was manual which increases the threat of frequent failure.

- It is obvious from the above stated problems that not only the software delivery process became slow but the quality of software also went down. This leads to customer dissatisfaction.
- So to overcome such problem there was a need for a system to exist where developers can continuously trigger a build and test for every change made in the source code.
- This is what Continuous Integration (CI) is all about. Jenkins is the most mature Continuous Integration tool available so let us see how Continuous Integration with Jenkins overcame the above shortcomings.
- Let's see a generic flow diagram of Continuous Integration with Jenkins:



- **Let's see how Jenkins works**. The above diagram is representing the following functions:
  - First of all, a developer commits the code to the source code repository. Meanwhile, the Jenkins checks the repository at regular intervals for changes.
  - Soon after a commit occurs, the Jenkins server finds the changes that have occurred in the source code repository. Jenkins will draw those changes and will start preparing a new build.
  - If the build fails, then the concerned team will be notified.
  - If built is successful, then Jenkins server deploys the built in the test server.
  - After testing, Jenkins server generates a feedback and then notifies the developers about the build and test results.
  - It will continue to verify the source code repository for changes made in the source code and the whole process keeps on repeating.

Advantages and Disadvantages of using Jenkins

Advantages of Jenkins

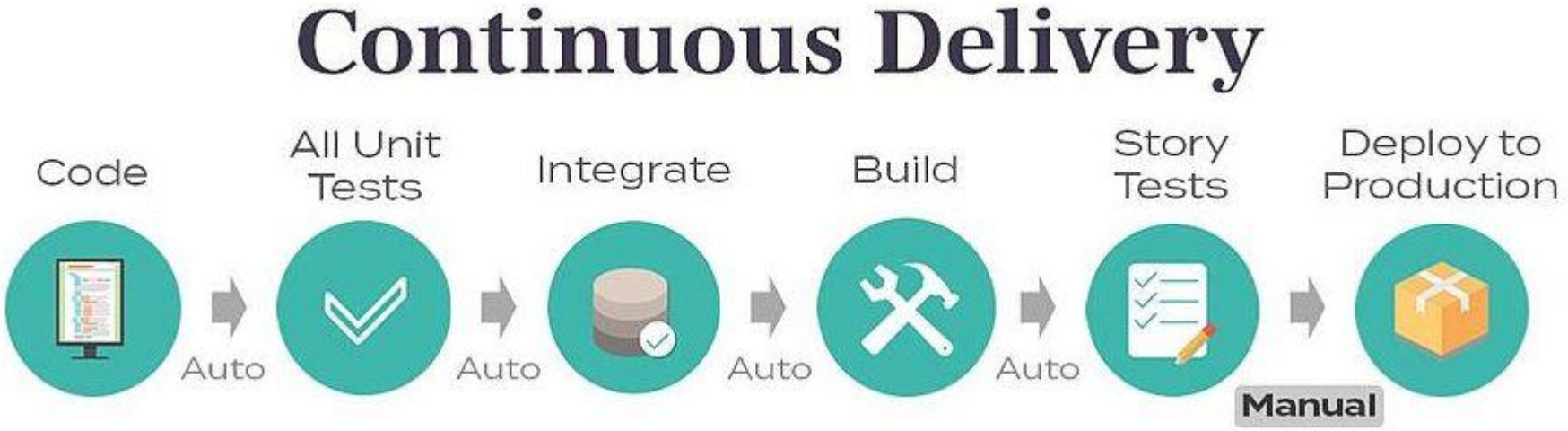
- It is an open source tool.
- It is free of cost.
- It does not require additional installations or components. Means it is easy to install.
- Easily configurable.
- It supports 1000 or more plugins to ease your work. If a plugin does not exist, you can write the script for it and share with community.
- It is built in java and hence it is portable.
- It is platform independent. It is available for all platforms and different operating systems. Like OS X, Windows or Linux.
- Easy support, since it open source and widely used.
- Jenkins also supports cloud based architecture so that we can deploy Jenkins in cloud based platforms.

Disadvantages of Jenkins

- Its interface is out dated and not user friendly compared to current user interface trends.
- Not easy to maintain it because it runs on a server and requires some skills as server administrator to monitor its activity.
- CI regularly breaks due to some small setting changes. CI will be paused and therefore requires some developer's team attention.

Continuous Delivery

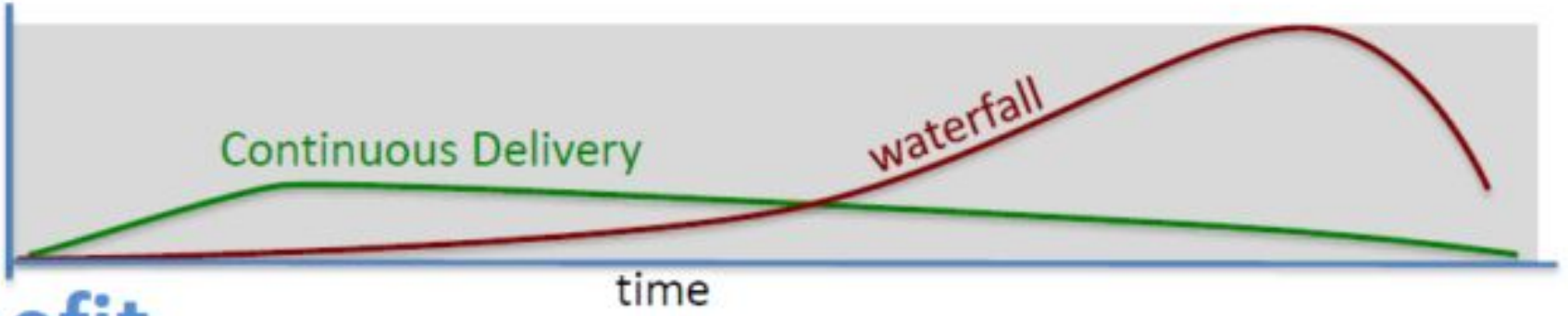
- Continuous delivery (CD) is a software development methodology in which code modifications are automatically packaged and deployed to production. It aims to accelerate development, cut expenses, and lower risks without losing code quality.
- CD is attained by designing a simple release procedure that is easily reproducible and restricts manual activities. In an ideal CD procedure, only application deployment into production requires human participation.



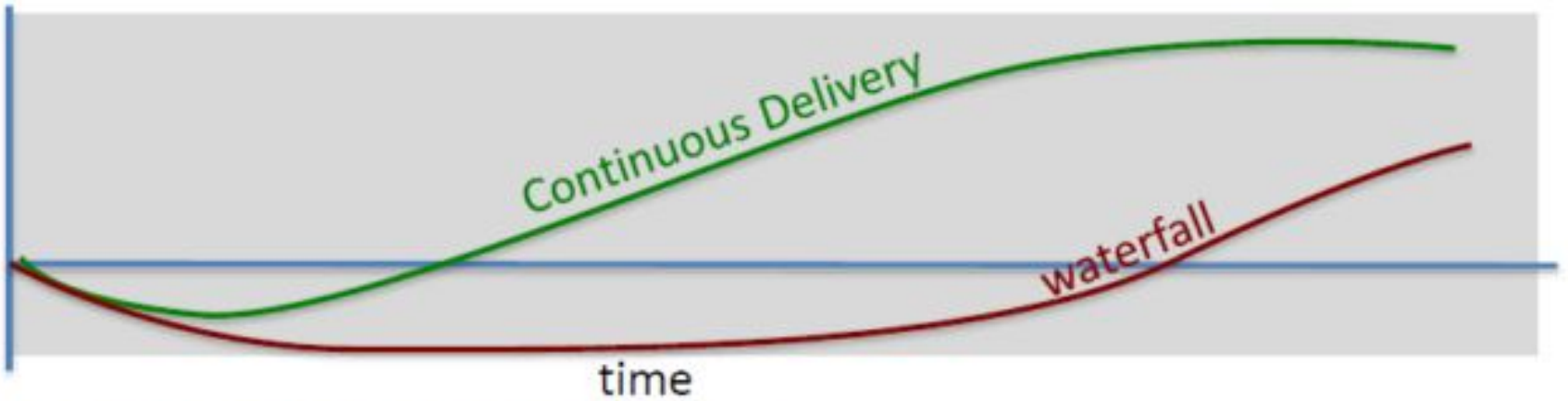
- Once a development team has accomplished continuous integration, CD is the next step in software pipeline automation (CI). CI automates the merging and testing of code modifications, focusing particularly on unit testing.
- The application is deployed to a staging environment for additional testing once the code has passed evaluations.
- These assessments consist of integration testing, performance testing, user interface testing, and more.
- CI and CD constitute the CI/CD pipeline, which transfers code from the computers of individual developers via automated testing to a production-ready build.
- At this point, all that is required is for a team member to deploy the latest version manually, often at regular intervals.
- With the emphasis on automation and velocity, CI/CD is a pillar of the DevOps concept.

- Why is it important?
- In a typical waterfall lifecycle, a customer release (minor or major) is a significant milestone that requires months of work. The entire crew is focused on achieving this objective.
- The release's features are developed, tested, and merged into the main branch as a single unit. There is a huge cost associated with failed releases and intense pressure to achieve deadlines.
- Consumers also wait lengthy durations for resolutions to their problems. Significant downtime occurs when the release patch is applied.
- Contrast this with the Agile model's shorter release lifecycle. Here, features can be checked into the central repository every day (Continuous Integration).
- Automated suites of unit, regression, and system tests ensure that freshly contributed code is quickly validated. Effect on relevant characteristics is examined. If this code can then be deployed promptly to customers, the strain surrounding a customer release is eliminated.
- Releasing becomes a common, low-risk, and automated occurrence. Manual errors are eliminated from the releasing procedure.
- Even during holidays, urgent fixes and updates can be issued with minimum manpower. This led to the evolution of Continuous Delivery.

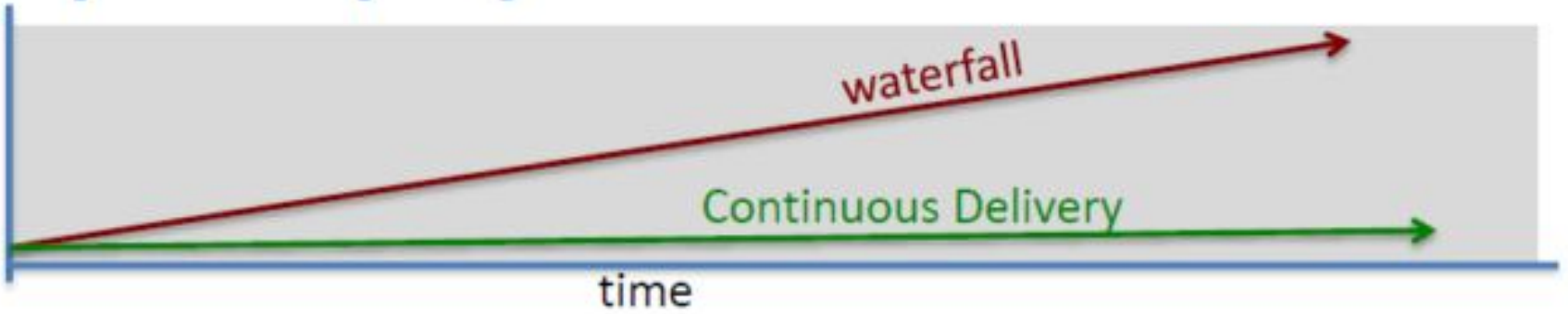
Bugs



Benefit

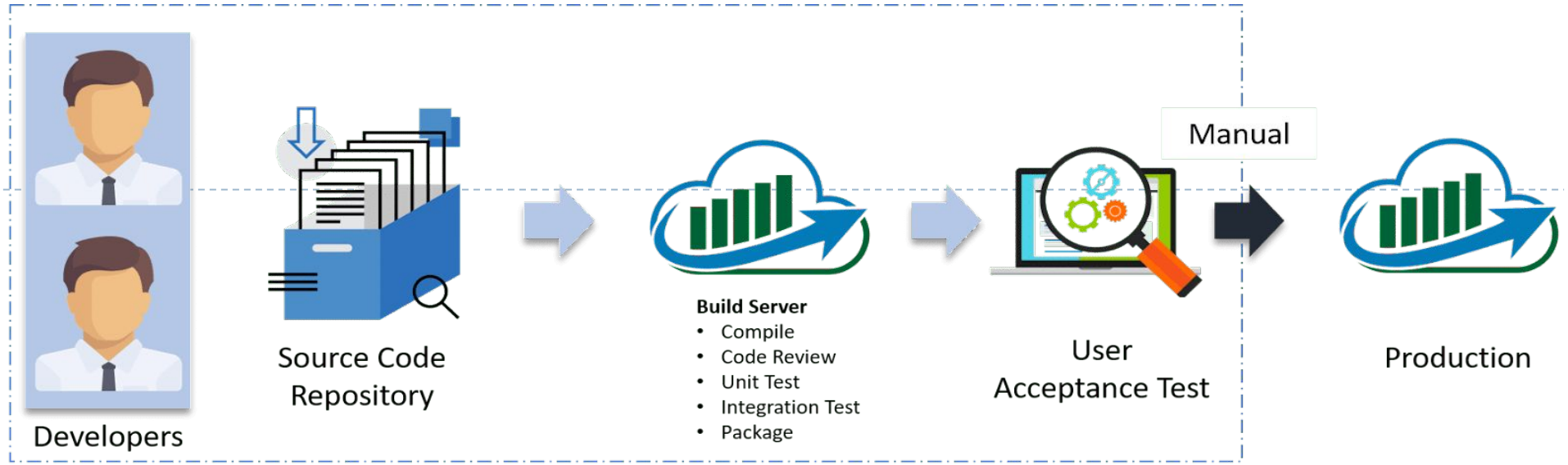


Cost per Deployment



How to Build a Continuous Delivery Pipeline

- Continuous delivery pipelines extend the operations and technologies currently implemented for a CI pipeline.
- Throughout the CI process, the code has been compiled to construct the application and unit-tested to ensure its functionality and quality.
- If the application fails to build or the unit tests fail, the code is returned to the developers to remediate and test again.
- Now, the application is prepared for additional testing. You will need a method to generate a staging environment identical to the production setting.
- Development teams generally resort to cloud services to provide a multistage environment to host the application and organise the testing workflow, since cloud hosting can expand to match processing demands.



- A CD pipeline may contain quality gates that establish success criteria. Before the program may advance to the subsequent phase, the performance, integration, and user interface (UI) tests must achieve these criteria. AI may be effective for identifying failure causes and viable solutions.
- You should automate as many tests and processes as possible as part of this testing strategy. This reduces the risk of human mistakes associated with manual methods, such as executing tests in a different order, as well as enhances speed. Everything related to Continuous Delivery in DevOps should be uniform and repeatable.
- Lastly, invest in observation and monitoring systems, as a failure in one segment of the pipeline could cause the entire system to fail. Including automated alarms and redundancies will ensure that, for instance, the failure of a single testing tool does not affect client delivery.

Benefits of Continuous Delivery in DevOps

1. Quicker detection of defects

- Continuous Delivery in DevOps is based on a robust testing technique, automatically testing an application against expected behaviour after deployment in the "real world." This allows developers to uncover flaws before pushing the code to production, where they may cause user disruptions and irritation.
- Not only does Continuous Delivery in DevOps boost user satisfaction, but it also enables the development team to anticipate these flaws in future releases, whereas a fault that does not actively disrupt the program may never be found and hence never be repaired.

2. Reduction of Costs

- By removing manual processes, Continuous Delivery in DevOps reduces the cost of delivering new software and upgrades, allowing developers to spend more time on higher-order tasks.
- Also, the speed of a CI/CD pipeline allows for the quicker delivery of additional features. This boosts the development team's output and frees up bandwidth to explore more projects without the need to hire additional engineers.

- 3. Improve quality.
- Continuous Delivery in DevOps is used to standardise an application's requirements by embodying them in test cases, hence increasing the likelihood that the result will meet users' needs.
- Continuous Delivery in DevOps also enables development teams to offer a minimally viable product (MVP) more quickly, allowing the customer to provide immediate feedback on improvement areas. Developers want feedback in order to continue offering consumer value.

- 4. Reduce Risk
- The primary objective of every software deployment should be "do no harm." The second objective is to provide value to the client, but they cannot appreciate an improved UI, for instance, if the program is unavailable due to the upgrade.
- By standardising the release process and implementing test validations to discover defects before they are published into production, Continuous Delivery (CD) reduces the risk associated with each deployment and instils greater confidence in the application among developers.

- 5. Increase employee satisfaction.
- By establishing an automated CD workflow, a business removes pain points such as manual repetitive testing for its developers and frees them to focus on strategy and optimization.
- In addition, the speed of CD pipelines expedites the deployment of engineers' code, allowing them to observe the impact of their work and how it assists customers in achieving their objectives.
- 6. Hasten Product Delivery
- CD eliminates obstacles in the development process so that updates can be deployed as soon as they have been validated. Its efficiency enables the engineering team to put out new features quickly to meet consumer demands.
- When an important issue arises, this speed advantage pays greater returns since it enables developers to deploy security updates and other solutions quickly.

Deployment Pipeline

Deployment Pipeline

- In software development, a **deployment pipeline** is a system of automated processes designed to quickly and accurately move new code additions and updates from version control to production.
- In past development environments, manual steps were necessary when writing, building, and deploying code. This was extremely time consuming for both developers and operations teams, as they were responsible for performing tedious manual tasks such as code testing and code releases.
- Main Stages of a Deployment Pipeline
- There are four main stages of a deployment pipeline:
  - Version Control
  - Acceptance Tests
  - Independent Deployment
  - Production Deployment

- **Version Control** is the first stage of the pipeline. This occurs after a developer has completed writing a new code addition and committed it to a source control repository such as GitHub.
- Once the commit has been made, the deployment pipeline is triggered and the code is automatically compiled, unit tested, analyzed, and run through installer creation.
- If and when the new code passes this version control stage, binaries are created and stored in an artifact repository. The validated code then is ready for the next stage in the deployment pipeline.
- In the **Acceptance Tests** stage of the deployment pipeline, the newly compiled code is put through a series of tests designed to verify the code against your team's predefined acceptance criteria.
- These tests will need to be custom-written based on your company goals and user expectations for the product.
- While these tests run automatically once integrated within the deployment pipeline, it's important to be sure to update and modify your tests as needed to consistently meet rising user and company expectations.

- Once code is verified after acceptance testing, it reaches the **Independent Deployment** stage where it is automatically deployed to a development environment.
- The development environment should be identical (or as close as possible) to the production environment in order to ensure an accurate representation for functionality tests.
- Testing in a development environment allows teams to squash any remaining bugs without affecting the live experience for the user.
- The final stage of the deployment pipeline is **Production Deployment**. This stage is similar to what occurs in Independent Deployment, however, this is where code is made live for the user rather than a separate development environment.
- Any bugs or issues should have been resolved at this point to avoid any negative impact on user experience. DevOps or operations typically handle this stage of the pipeline, with an ultimate goal of zero downtime.
- Using Blue/Green Drops or Canary Releases allows teams to quickly deploy new updates while allowing for quick version rollbacks in case an unexpected issue does occur.

Blue/Green Deployments

- Utilisation of a Blue/Green Deployment process reduces risk and down time by creating a mirror copy your production environment naming one Blue and one Green. Only one of the environments is live at any given time serving live production traffic.

- During a deployment, software is deployed to the non-live environment – meaning live production traffic is unaffected during the process. Tests are run against this currently non-live environment and once all tests have satisfied the predefined criteria traffic routing is switched to the non-live environment making it live.

- The process is repeated in the next deployment with the original live environment now becoming non-live.

Canary Deployments

- Different from Blue/Green deployments, Canary Deployments do not rely on duplicate environments to be running in parallel. Canary Deployments roll out a release to a specific number or percentage of users/servers to allow for live production testing before continuing to roll out the release across all users/servers.

- The prime benefit of canary releases is the ability to detect failures early and roll back changes limiting the number of affected users/services in the event of exceptions and failures.

Benefits of a Deployment Pipeline

- Building a deployment pipeline into your software engineering system offers several advantages for your internal team, stakeholders, and the end user. Some of the primary benefits of an integrated deployment pipeline include:
  - Lower-risk releases. Blue/Green deployments and canary releases allow for zero downtime deployments which are not detectable by users and make rolling back to a previous release relatively pain free.
  - Teams are able to release new product updates and features much faster.
  - There is less chance of human error by eliminating manual steps.
  - Automating the compilation, testing, and deployment of code allows developers and other DevOps team members to focus more on continuously improving and innovating a product.
  - Troubleshooting is much faster, and updates can be easily rolled back to a previous working version.
  - Production teams can better respond to user wants and needs with faster, more frequent updates by focusing on smaller releases as opposed to large, waterfall updates of past production systems.

How to Build a Deployment Pipeline

- A company's deployment pipeline must be unique to their company and user needs and expectations, and will vary based on their type of product or service. There is no one-size-fits-all approach to creating a deployment pipeline, as it requires a good amount of upfront planning and creation of tests.
- When planning your deployment pipeline, there are three essential components to include:
- **Build Automation (Continuous Integration):** Build automation, also referred to as Continuous Integration or CI for short, are automated steps within development designed for continuous integration – the compilation, building, and merging of new code.
- **Test Automation:** Test automation relies on the creation of custom-written tests that are automatically triggered throughout a deployment pipeline and work to verify new compiled code against your organization's predetermined acceptance criteria.
- **Deploy Automation (Continuous Deployment/Delivery):** Like continuous integration, deploy automation with Continuous Deployment/Delivery (CD for short) helps expedite code delivery by automating the process of releasing code to a shared repository, and then automatically deploying the updates to a development or production environment.

https://learn.microsoft.com/en-us/fabric/cicd/deployment-pipelines/get-started-with-deployment-pipelines?tabs=from-fab
2Cnew%2Cstage-settings-new

- Deployment Pipeline Tools
- Making use of available tools will help to fully automate and get the most out of your deployment pipeline. When first building a deployment pipeline, there are several essential tool categories that must be addressed, including source control, build/compilation, containerization, configuration management, and monitoring.
- A development pipeline should be constantly evolving, improving and introducing new tools to increase speed and automation. Some favorite tools for building an optimal deployment pipeline include:
  - Jenkins
  - Azure DevOps
  - CodeShip
  - PagerDuty

Infrastructure Management

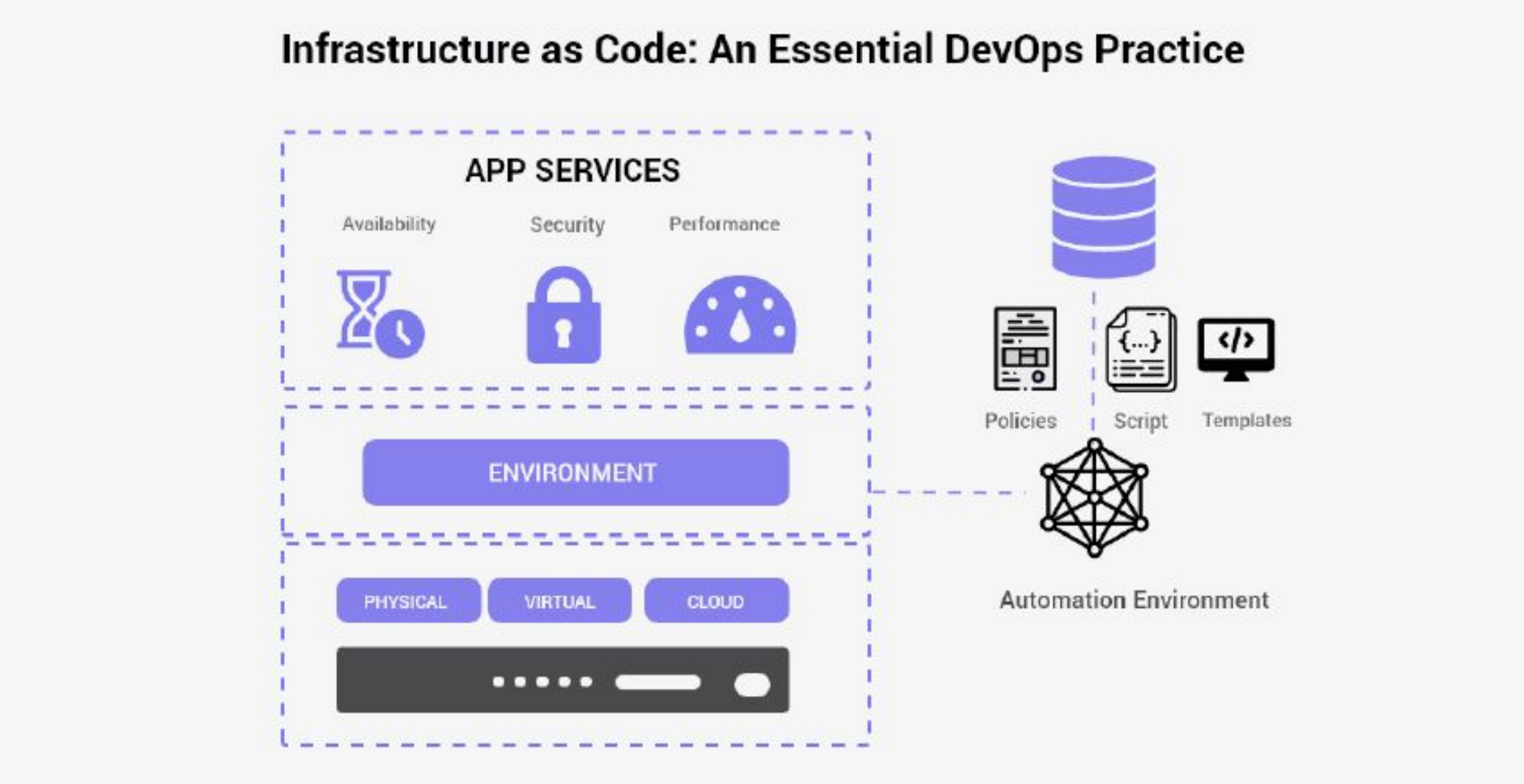
DevOps infrastructure

- DevOps infrastructure refers to the combination of tools, processes, and resources that support the seamless integration, collaboration, and automation of software development and operations.
- Some of the leading IAC tools are listed below.
  - Ansible
  - Terraform
  - SaltStack
  - CloudFormation
  - Azure Resource Manager
  - SpectralOps

What is DevOps Infrastructure as Code?

- With the help of DevOps Infrastructure as Code, the IT environment can be provisioned with multiple lines of code.
- The incorporation of IaC systems helps to speed up the procedures which would, otherwise, take hours or sometimes even days for configuration if manually processed.

Infrastructure Automation in DevOps



Infrastructure Automation in DevOps

- There might be a number of factors such as irregular allocation of resources, lack of metrics, fragmented procedures, and misalignment or improper allocation of responsibilities, that might hinder an organization from adopting and scaling the DevOps practices at full length.
- Of all the roadblocks, improper infrastructure management might be the most bothersome.
- When infrastructure is properly managed, it makes sure that the resources are configured properly, secured, backed up on a regular basis, and monitored from time to time.
- However, it may not be feasible to perform all these tasks manually at the enterprise level. This is where DevOps steps in. DevOps infrastructure is automated which ensures ease of management.

https://www.hex64.net/devops-infrastructure-management/

Why Use DevOps in Infrastructure?

- DevOps helps in the automation of infrastructure. This enables the operations team and the developers to manage, facilitate, and monitor the resources automatically.
- This saves the hassle and effort of having to do everything manually, which can slow down the process and cause an organization to function inefficiently.
- Since the infrastructure is automated, the DevOps teams can test the applications very early during the development cycle. This allows the provision for multiple environments for testing and helps to keep at bay the common issues related to deployment.

Listed below are some of the benefits of using DevOps in infrastructure:

- **Predictability:** With the help of DevOps, there will be a relatively lower rate of failure of the new releases. This is because the products will be tested at the initial stage itself.
- **Better Quality:** DevOps helps the teams to come up with improved and better-quality applications by taking the infrastructure issues into consideration.
- **Error elimination and reduced time for recovery:** The practices put to use by DevOps are ideally suited for eliminating the impact of rollbacks, bottlenecks, or deployment failures that might affect efficiency. When these issues are addressed at an early stage, they can be lived through easily. Quick detection of errors eases the process for both the operations as well as the developer's teams.
- **Reduced marketing time:** Since DevOps streamlines the procedure of software delivery, the process of marketing can be made quicker.

- **How DevOps Infrastructure Management Impacts Provisioning and Deployment**
- To understand how infrastructure in DevOps works, you need to have a look at the stages that are involved.
- **Stage 1: Planning**
- In the planning stage, the goals and requirements for the project are chalked out. It is important to identify the most suitable and compatible team members who will represent both Dev and Ops, to work on the project. All the information has to be thoroughly shared by both verticals so that the operation team has a clear idea about the development goals, needs, and timelines.
- **Stage 2: Development**
- Based on the specifications that were laid out during the planning stage, the most appropriate environments for development are provisioned. The team that is in charge of the infrastructure makes use of the automated configuration tools which help to get the task done with great ease. Some of the tools that are used in this context are Infrastructure as Code, Software-Defined Networking, and Infrastructure Orchestration.

- **Stage 3: Testing**
- Testing continues throughout the entire procedure; it cannot be flagged as a separate stage. This falls under the infrastructure side, where tests are automatically run based on the new IaC configurations. Any issue thus found has to be remediated prior to the commencement of the development stage. This eliminates the risk of bottlenecks.
- **Stage 4: Deployment**
- Based on the pipeline, build, production, and delivery, the servers are automatically configured. It is in the deployment stage that the IaC helps in the building of the production environment, and then harnesses the use of different tools to release the build.
- **Stage 5: Support/Feedback**
- Any issue that might arise with the software or the infrastructure is detected by the process of automation and is reported automatically. The issues are then automatically forwarded for rectification. As is the essence of DevOps in infrastructure, members in charge of both development as well as operations come together to troubleshoot the problems and brainstorm together so that they can come up with the most feasible solutions for the errors in infrastructure.

https://zeet.co/blog/devops-infrastructure

How Does DevOps Infrastructure Automation Work?

- 1. Infrastructure as Code (IaC):
  - This method abstracts infrastructure settings and represents them in scripts and configuration files.
  - By predefined rules, these scripts enable automatic provisioning of resources, setup and monitoring.
  - By packaging infrastructure configurations in code, team can apply the same scripts across various environments, guaranteeing uniformity and predictability in their infrastructure deployment. This strategy reduces the chance of errors and divergences in setting, resulting in more dependable and effecive deployment process.
  - IaC improves security by addressing misconfiguration risks. Organizations can consistently implement standardized security measures by utilizing predifined configuration in scripts.
  - IaC helps to achieve scalability and fexibility. Deploying multisystems simultaneously eliminates bottlenecks and speeds up development and delivery processes.

How Does DevOps Infrastructure Automation Work?

2. Continous Integration/Continous Delivery

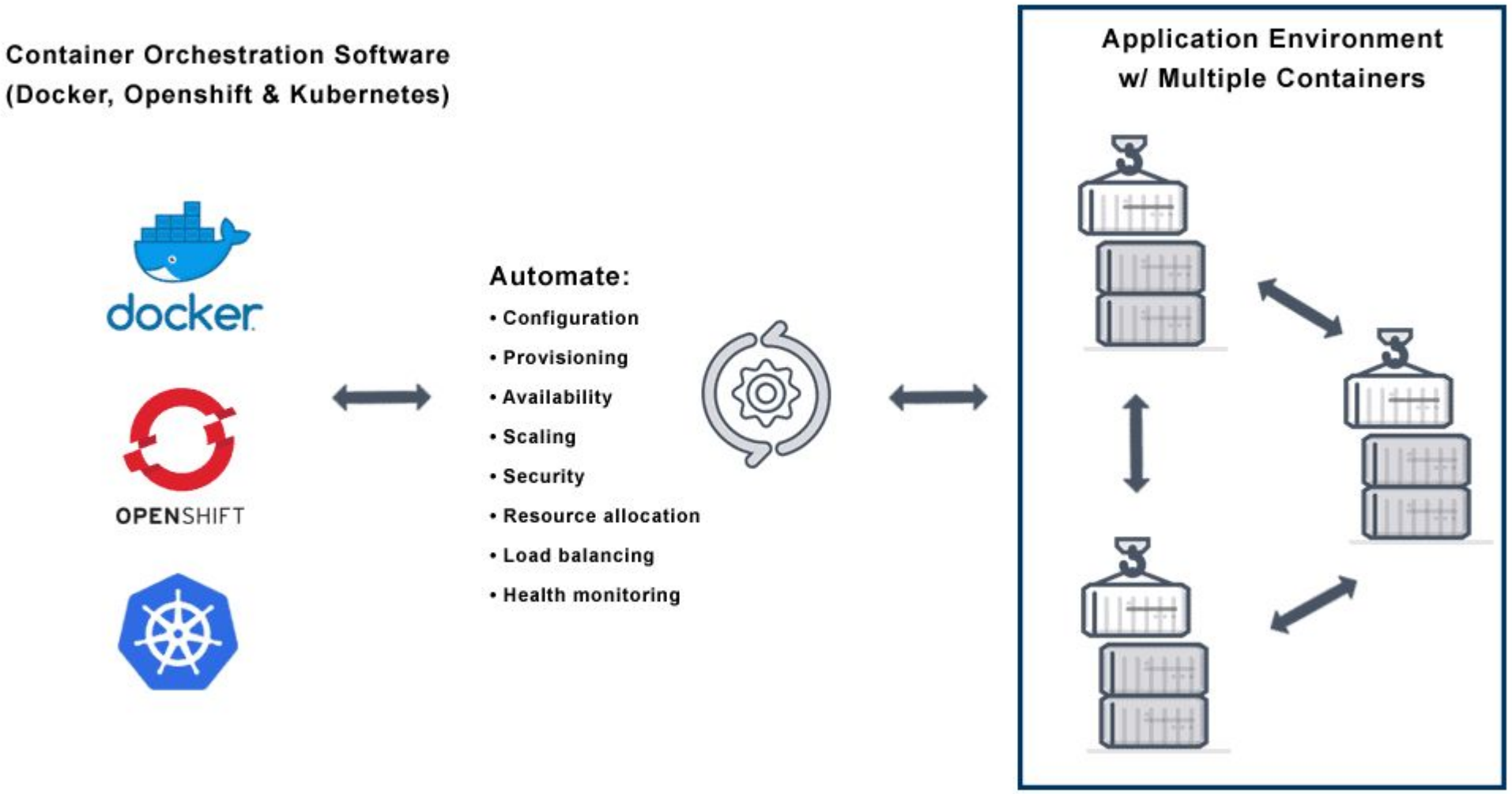
- Continous Integration (CI):
  - It involves automating the integration of code changes into a shared repository multiple times daily. Developers regulary commit code changes to the rpository, initiating autmated build adn test processes.
  - By automating the integration and testing of code changes, CI helps identify and address issues early in the development process, reducng the risk of integration conflicts and ensuring code quality.
- Continous Delivery (CD):
  - CD builds upon CI by automating the process of deploying code changes to procution or staging environments. This involves automating steps such as packaging, deployment and application testing.
  - CD pipelines automate deployment, allowing organizations to release software updates quickly and reliably. Automated testing and validation ensure that deployments adhere to quality standards and are prepared for production use.

How Does DevOps Infrastructure Automation Work?

3. Container and Orchestration

- Containers are lightweight virualization that consolidates applications and their dependencies into seperate units, enabling them to operate consistanetly across various environments. All the components for an application to function are encompassed within containers, encompassing code, runtime, system tools, libraries and configurations, guaranteeing consisten performance regardless of the environment.
- Orchestration involves the automatic control and organization of containerized applications throughout a distributed infrastructure. Platforms dedicated to container orchestration, such as Kubernetes, Docker Swarm and Amazon ECS, deliver tools and functionalities for deploying, scaling, managing and monitoring containerized applications on a large scale.

How Does DevOps Infrastructure Automation Work? - Container and Orchestration



https://www.simform.com/blog/container-orchestration/

How Does DevOps Infrastructure Automation Work?

4. Automated Storage Resource Optimization

- Organizations can improve performance, availability and data security by matching storage configurations with application needs and workload features. Additionally, adeqately optimized storage resouces greately simplify storage management, lowering administrative overhead and complexities and increasing operational efficiency.
- More over efficient storage management facilitates disaster recovery and business continuity by facilitating prompt data backups, replication and recovery functions.

5. Reserved Instance(RI) Optimization

- RIs are pricing model that provides users discounts in return for their long-term.

DevOps Infrastructure Automation Tools

- The tools can be categorized into the following categories
  - Tools for automating storage resource optimization
  - Infrstructure as Code
  - Continuous integration and delivery
  - Container orchestration and image management
  - Config/Secret management
  - Monitoring and logging

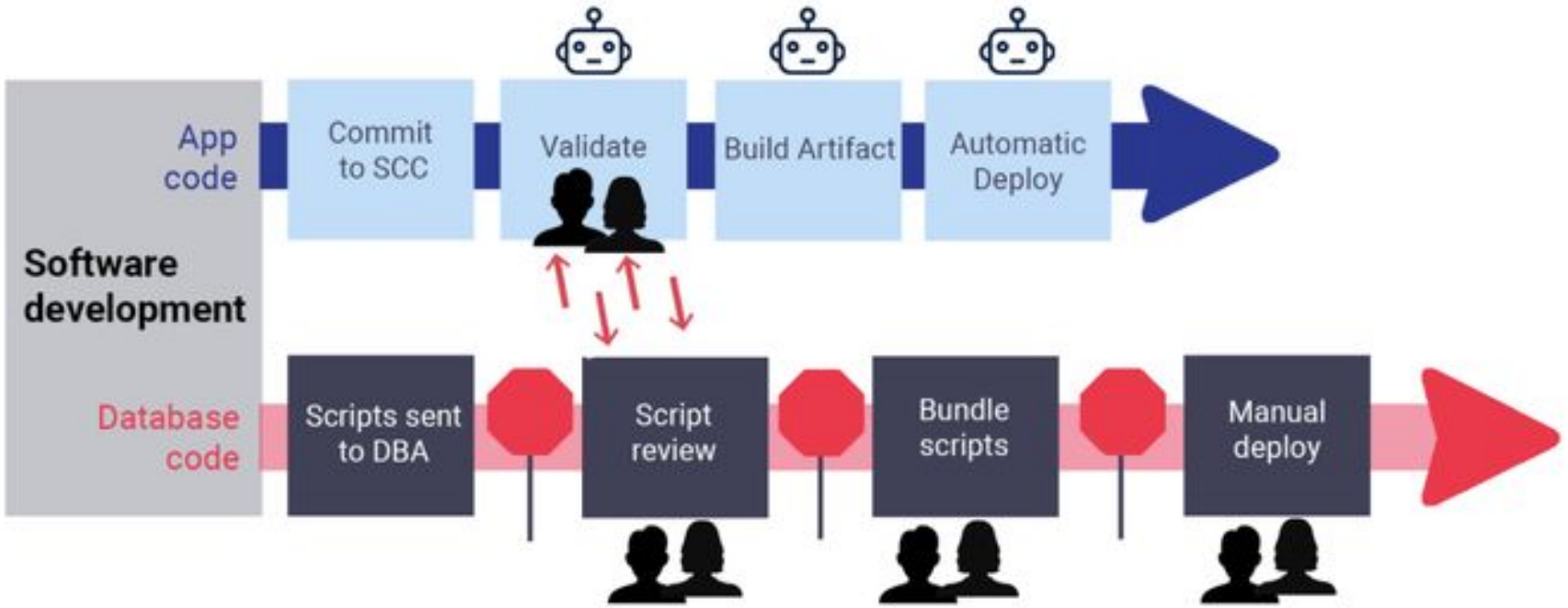
https://www.lucidity.cloud/blog/devops-infrastructure-automation

Databases

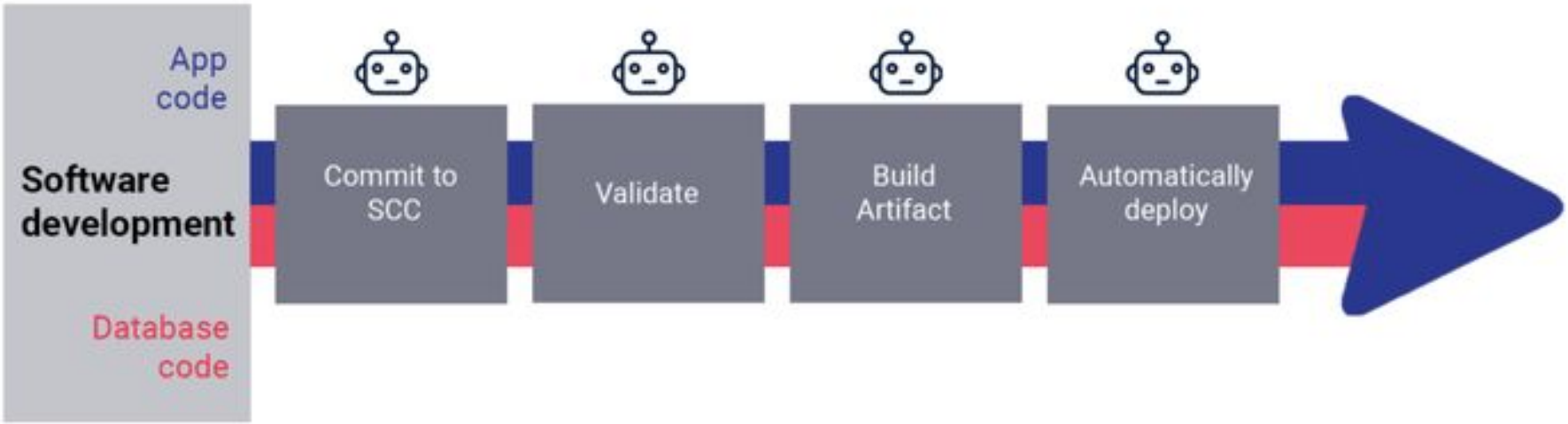
Database DevOps

- Building stable and consistent products need the integration of databases and applications. By integrating DevOps, applications and databases can be constructed and deployed via a single infrastructure.
- Database development and management are done using DevOps techniques. These adjustments are also done in response to comments from the delivery and application development phases. This is done to ensure a seamless delivery.

https://www.linkedin.com/pulse/introduction-database-devops-sachin-agarwal



A 2019 State of Database Deployments in Application Delivery report



Database DevOps

- Database DevOps applies these same principles, making sure that the database code is included in the same process as development code. Database DevOps helps teams identify and streamline the application development and release process further by addressing a known bottleneck: database code changes.
- **Database DevOps is the extension of DevOps principles, technology, and processes to the database management workflow that keeps updates to data stores in sync with the rest of the application development pipeline. It bridges the worlds of database management with the agile and collaborative spirit of DevOps to ensure that updates to databases march in lockstep with software delivery, making the life cycle smoother, faster, and more synchronized.**

Database DevOps Features :

- Version control
- Source control
- Automated deployment
- Release management
- Script validation
- Schema maintenance
- Visibility into database changes
- Rollbacks

Top Database DevOps Challenges

- **Data Synchronization:** Keeping development and production databases in sync can be complex. Changes made to database schemas, data, or configurations need to be accurately synchronized between environments.
- **Data Security:** Ensuring that sensitive data is properly protected throughout the DevOps process, especially in non-production environments, is crucial. Masking, encryption, and access control are key considerations.
- **Testing and Validation:** Comprehensive testing of database changes is essential. It can be challenging to automate testing procedures for databases, including data validation and performance testing.
- **Schema Evolution:** Managing database schema changes and handling backward compatibility while maintaining existing data can be tricky. Ensuring that database changes do not break existing applications is crucial.
- **Version Control:** Applying version control to databases and ensuring that database changes are tracked, reviewed, and documented properly can be challenging. Database code should be treated with the same rigor as application code.

Top Database DevOps Challenges

- **Manual Processes:** Over-reliance on manual processes for database deployments can slow down the DevOps pipeline and introduce errors.
- **Legacy Systems:** Integrating Database DevOps practices into legacy systems can be challenging, especially if these systems were not initially designed with DevOps in mind.
- **Cultural and Organizational Changes:** Implementing Database DevOps often requires cultural and organizational shifts, including breaking down silos between development and operations teams, which can be met with resistance.
- **Tooling:** Finding and implementing the right tools for Database DevOps can be daunting, as the ecosystem is vast and rapidly evolving.
- **Compliance and Auditing:** Meeting regulatory requirements and ensuring that database changes are audited and compliant with industry standards is an ongoing challenge.
- **Performance Tuning:** Optimizing database performance in a DevOps environment requires continuous monitoring and fine-tuning, which can be resource-intensive.
- **Rollback and Recovery:** Having effective rollback and recovery procedures in place is crucial in case a database change causes unexpected issues in production.

How can DevOps help in solving the above challenges?

- Data Synchronization:Automation Tool: AnsibleUse Ansible playbooks to automate database schema updates and ensure consistent configurations across environments.Example: Use Ansible to deploy database schema changes, ensuring that development and production databases remain synchronized.
- Data Security:Automation Tool: HashiCorp VaultHashiCorp Vault automates secrets management and encryption, enhancing data security.Example: Store and retrieve database credentials securely from HashiCorp Vault, ensuring sensitive data remains protected.
- Testing and Validation:Automation Tool: JenkinsIntegrate Jenkins as part of the CI/CD pipeline to automate database testing.Example: Implement automated database testing scripts within Jenkins jobs to validate data integrity and performance.
- Schema Evolution:Version Control Tool: GitUse Git for version control of database schema scripts, enabling efficient collaboration.Example: Maintain a Git repository for tracking and managing database schema changes, allowing for easy rollback if needed.

How can DevOps help in solving the above challenges?

- Version Control:Database Version Control Tool: LiquibaseLiquibase is designed for database schema versioning and change management.Example: Use Liquibase to track and deploy database schema changes with version control, enabling a structured approach to database evolution.
- Manual Processes:Automation Tool: ChefChef automates infrastructure provisioning, including database setups.Example: Use Chef recipes to automate database server configurations, reducing manual intervention.
- Legacy Systems:Gradual Transition Tool: DockerContainerization with Docker can facilitate gradual migration to modern Database DevOps practices.Example: Containerize legacy databases using Docker to make them more manageable and integrate them into the DevOps pipeline.
- Cultural and Organizational Changes:Collaboration Tool: SlackUse communication tools like Slack to foster collaboration among development and operations teams.Example: Create dedicated Slack channels for cross-functional teams to encourage communication and knowledge sharing.

How can DevOps help in solving the above challenges?

- Tooling:Database Deployment Tool: AWS RDSAWS Relational Database Service (RDS) simplifies database management and can be integrated into DevOps workflows.Example: Use AWS RDS to provision, scale, and manage databases in a DevOps-friendly manner.
- Compliance and Auditing:Compliance Tool: Sysdig SecureSysdig Secure provides automated compliance checks and auditing for containerized environments.Example: Implement Sysdig Secure to continuously monitor and audit containerized databases for compliance with industry standards.
- Performance Tuning:Monitoring Tool: PrometheusPrometheus offers robust monitoring capabilities for database performance.Example: Set up Prometheus for real-time monitoring of database metrics and establish alerting to address performance issues promptly.
- Rollback and Recovery:Backup and Restore Tool: AWS BackupAWS Backup automates data backup and recovery for AWS resources, including databases.Example: Configure automated backups with AWS Backup to ensure quick recovery in case of unexpected issues caused by database changes.

Why Do You Need Database DevOps?

- Database DevOps enables you to put a stronger emphasis on built-in security while eliminating database-related hindrances. These hindrances include avoiding system crashes.
- Using Database DevOps may help you lessen or get rid of this friction between developers and Database Administrators (DBAs). It makes it easier for both sides to comprehend each other's needs by adopting a precise, repeatable method.
- The bridged gap and better collaboration between teams help in obtaining faster and better feedback during the process of DevOps. Faster feedback makes it simpler to spot and resolve problems like empty procedures, unused views and tables, and user or query conflicts. When implementing infrequently used features, this is quite helpful. This helps in addressing and solving problems quickly.

Why Do You Need Database DevOps?

- Utilizing DevOps database tools to automate parts of the processes further minimizes the workload that requires human intervention. This results in a further reduction in the team's employee needs. This helps in reducing overhead costs to a considerable extent.
- DevOps is all about collaboration by reducing gaps between teams. Since database DevOps, developers, and database administrators collaborate closely, issues on applications and databases are addressed readily and solved promptly.

What Are the Principles of Database DevOps?

Here are a few basic principles of database DevOps-

Continuous Supply

With database DevOps, the procedure for applying a change across a number of environments, using unit tests, and culminating in being deployed to production is largely automated. With this approach, major problems are no longer made in the production environment. This will ensure the avoidance of unforeseen contingencies in the future.

Repeated Deployments

Being able to incorporate minor changes in the system using the same process will ensure reliable outcomes can be achieved readily.

Unit Tests

Testing is a very important part of any release. It is required for any modifications that are developed and distributed. They are needed in order to determine whether the modifications and developments adhere to specifications, or break the deployed environment or both.

Source control

Tracking and managing changes to software code is done through source control. A database's known state in a certain environment at a specific point in time is made possible by the storage of all database code, from initial scripts used to create the schema to each iterative revision.

Top Database DevOps Challenges & Strategies

- There are a good number of challenges that come with database DevOps. Take a look at them:
- Databases have different scalability and size problems. Schema implementation on a large database in a production setting can be a challenging task. This can also result in performance problems and deteriorated services. When multiple applications bank on one single database, making modifications to it becomes even more challenging.
- Database procedures and technologies don't always work well with pipelines that already exist. These technologies frequently have database and environment requirements, which might make it challenging to integrate them into flexible pipelines.
- A compartmentalized perspective makes it impossible for different parties to come together in a way that makes integration possible. This has been the main obstacle to implementing database DevOps.
- You cannot shift databases to a different environment or modify the schema without taking the data inside into account. Instead, you must guarantee that data integrity is upheld and carefully analyze how changes may influence data. Compared to application and deployment changes, this is significantly different. Data persistence is not an issue because these modifications simply involve modifying the code.

How To Overcome Database DevOps Challenges?

- If you want to stay relevant, you need to spend money on professional developers who are up to date on all the most recent technological developments. This will help you stay ahead and create newer and better solutions.
- Incremental modifications and regular evaluation are the foundation of DevOps processes. Incremental modifications and regular evaluation are the foundation of DevOps processes. With the help of this process, the possibility of modifying work when necessary becomes higher. By doing this, it's possible to make sure that problems are found prior to application deployment.
- With the help of staging databases, you can be sure of testing the performance of the applications you develop in a realistic environment. These databases offer a better approach to testing database modifications before implementing them in live databases.
- Automate your database builds from a familiar and source-controlled version. By doing this you can ensure easier coordination during database upgrades. Testing changes more frequently and early will help you find errors earlier in the pipeline when they are simpler to address and save them from becoming problems.

- **DevOps Database Best Practices**
- **Employ Tight Security Measures**
- Establish database access permissions so that warnings can be provided to deter unauthorized use and enforce compliance. Set permissions for specific alterations to the database code to restrict access to the database.
- **Record the Changes**
- Make sure you keep a record of all the modifications made by stakeholders. The ability to trace versions and modifications, as well as a complete history of all database changes made at any given time will go a long way.
- **Choose Adaptable Automation Solutions**
- Choose adaptable and compatible automation solutions for your tools. Try avoiding difficulties in tasks by fragmenting way too much. Try to avoid too much use of code. Do not forget to see that these tools work in sync.

- **Use a KPI**
- Being able to measure your process is the first step to avoiding repeated mistakes. This will help you ensure a more streamlined procedure. Choose your desired KPI to improve your performance and avoid bottlenecks.
- **Perform Frequent Testing**
- Frequently perform the process of testing. Continuous testing verifies source code and executes tests concurrently. This helps in alerting developers when a build is unsuccessful.
- **Get Proper Feedback**
- It is crucial to incorporate database feedback loops. Input loops are techniques used to validate and obtain feedback regarding the software development process. Receiving continuous feedback helps in optimizing the process of development. Getting feedback during the stage of development is essential to ensure the seamless delivery of products. Obtaining this feedback can also assist you in improving the processes and avoiding repeating mistakes.

End of Unit III

What is a Test Suite?

- Test suites are the logical grouping or collection of test cases to run a single job with different test scenarios.

- For instance, a test suite for product purchase has multiple test cases, like:

- Test Case 1: Login
- Test Case 2: Adding Products
- Test Case 3: Checkout
- Test Case 4: Logout

Test Case Form

- You must consider certain standard fields when preparing a test case template. The standard test case format contains:

- Test Case ID
- Test Scenario ID
- Test Case Description
- Test Priority
- Prerequisite
- Postrequisite
- Test Data
- Test Execution Steps
- Expected/Intended Results
- Actual Results
- Test Result – Pass/Fail

Testcase form

Test Scenario ID		Test Case ID	
Test Case Description		Test Priority	
Pre-Requisite		Post-Requisite	

Test Execution Steps:

S.No	Action	Inputs	Expected Output	Actual Output	Test Browser	Test Result	Test Comments