CICD

Need for DevOps

- DevOps—the amalgamation of development (Dev) and operations (Ops) teams—is an organizational approach that enables faster development of applications and easier maintenance of existing deployments.
- By enabling organizations to create stronger bonds between Dev, Ops and other stakeholders in the company, DevOps promotes shorter, more controllable iterations through the adoption of best practices, automation and new tools.
- DevOps is not a technology per se, but it covers everything from the organisation to culture, processes and tooling. Initial steps usually include Continuous integration and continuous delivery (CI/CD), real-time monitoring, incident response systems and collaboration platforms.

Benefits of DevOps

- Maximizes Efficiency with Automation
  - The late DevOps authority Robert Stroud said DevOps is all about "fueling business transformation" that encompasses people, process and culture change. The most effective strategies for DevOps transformation focus on structural improvements that build community. A successful DevOps initiative requires a culture—or mindset—change that brings greater collaboration between multiple teams—product, engineering, security, IT, operations and so on—as well as automation to better achieve business goals.
  - What kind of tangible benefits can DevOps bring? By managing engineering processes end to end, DevOps emphasizes deploying software more often, in a reliable and secure way through automation.

- Optimizes the Entire Business
  - System architect Patrick Debois, best known as the creator of the DevOps movement, says the biggest advantage of DevOps is the insight it provides. It forces organizations to "optimize for the whole system," not just IT siloes, to improve the business as a whole. In other words, be more adaptive and data-driven for alignment with customer and business needs

- Improves Speed and Stability of Software Development and Deployment
  - A multi-year analysis in the annual Accelerate State of DevOps Report has found that top-performing DevOps organizations do far better on software development/deployment speed and stability, and also achieve the key operational requirement of ensuring that their product or service is available to end users. But given the somewhat fuzzy definition of DevOps, how can an organization determine if its DevOps initiative is paying off? The 2019 Accelerate report also names five performance metrics—lead time (i.e., the time it takes to go from code committed to code successfully running in production), deployment frequency, change fail, time to restore and availability—that deliver a high-level view of software delivery and performance, and predict the likelihood of DevOps success.
- Gets You to Focus on What Matters Most: People
  - People, not tools, are the most important component of a DevOps initiative. Key roleplayers (i.e., humans) can greatly increase your odds of success, such as a DevOps evangelist, a persuasive leader who can explain the business benefits brought by the greater agility of DevOps practices and eradicate misconceptions and fears. And since automated systems are crucial to DevOps success, an automation specialist can develop strategies for continuous integration and deployment, ensuring that production and pre-production systems are fully software-defined, flexible, adaptable and highly available.

Challenges of DevOps

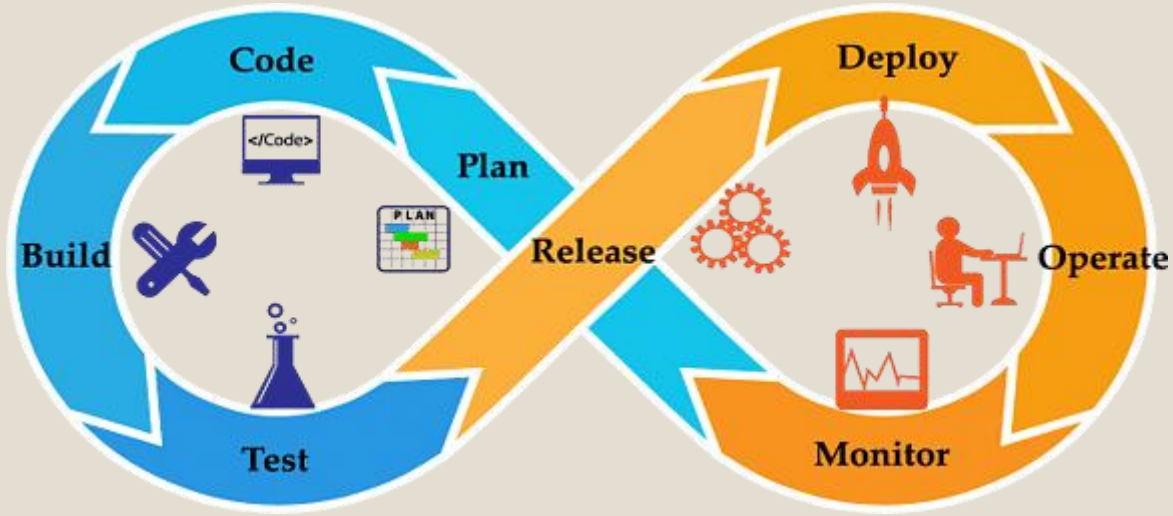
- Choosing the Right Metrics is Hard
  - Enterprises transitioning to DevOps practices need to use metrics to recognize progress, document success, and uncover areas that need improvement, Forrester notes. For example, an acceleration in deployment velocity without a corresponding improvement in quality is not a success. An effective DevOps effort needs metrics that drive smart automation decisions—and yet organizations often struggle with DevOps metrics.
  - So where to start? Find metrics that align with velocity and throughput success.

- Limited Funds
  - DevOps initiatives face other obstacles as well. Given the significant organizational and IT changes involved—with previously siloed teams joining forces, changing job roles, and encountering other transitions— adjustments will take time. According to a survey of IT executives from software company Pensa, the top challenges to DevOps success are:
    - Limited budgets (cited by 19.7% of respondents)
    - Legacy systems (17.2%)
    - Application complexity (12.8%)
    - Difficulty managing multiple environments (11.3%)
    - Company culture (9.4%)

- Complexity
  - DevOps efforts can be mired in complexity. IT leaders may have difficulty articulating the business value of their work to key executives. In terms of governance, will centralization and standardization lead to better results, or just more layers of innovation-killing bureaucracy? And then there's organizational change: Can your teams overcome resistance to change and inertia, unlearning many years of doing things a certain way, share their practices and learn from others, and integrate and orchestrate the right tools?
- Unrealistic Goals, Bad Metrics Can Wreck DevOps
  - DevOps efforts can fail for many reasons, such as setting unrealistic expectations, tracking metrics that don't align with business goals, or implementing a half-baked DevOps effort that embraces agile methodologies while keeping IT ops and engineering/development teams in traditional silos.

What is Continuous Integration?

- Continuous integration is a process in devops where changes are merged into a central repository after which the code is automated and tested. The continuous integration process is a practice in software engineering used to merge developers' working copies several times a day into a shared mainline.
- It refers to the process of automating the integration of code changes coming from several sources. The process comprises several automation tools that emphasize on the code’s correctness before Integration.
- Continuous Integration is the best practice for software development that has a set of critical principles. Some of the principles of CI are revision control, automated testing, and build automation. The process is not known to get rid of bugs but makes it easy to find and remove bugs.


Code
Plan
Build
Release
Test
Deploy
Operate
Monitor
</Code>
PLAN

What CI Does?

- Continuous Integration is a software development practice that integrates code into a shared repository frequently. This is done by developers several times a day each time they update the codebase. Each of these integrations can then be tested automatically.
- One of the main benefits of integrating regularly and testing each integration is that you can detect errors more quickly and locate them easily. Since each integration or update to codebase is usually small, pinpointing the exact change that causes the error can be done quickly.
-

How CI Can be Used?

- Over the past few years, Continuous Integration has become one of the best practices for software development. The goal is to detect the errors early on without having to wait until the end of the project.
- Here are some basic prerequisites for implementing Continuous Integration:
  - Automating builds - mvn clean install
  - Automating testing - mvn test
  - A single source code repository - common github/git lab
  - Visibility of the entire process
  - Real-time code access to everyone in the teamc- github
- For software development teams that don't practice CI, they should start with small steps instead of implementing the CI/CD (Continuous Integration/Continuous Development) pipeline immediately. They should continuously iterate on code and process in a way that helps the organization grow.

Importance of Continuous Integration

- Continuous Integration enables better transparency and farsightedness in the process of software development and delivery. It not only benefits the developers but all the segments of that company. These benefits make sure that the organization can make better plans and execute them following the market strategy.
- To understand the importance of CI, here are some of its benefits:
- 1. Reduces Risk
- The frequent testing and deployment of code reduce the project's risk level, as now the code defects and bugs can be detected earlier. This states that these bugs and errors can be easily fixed and take less time, making the overall process cheaper. The general working speeds up the feedback mechanism that makes the communication smoother and effective.

- 2. Better Communication
- The Continuous Integration process collaborates with the Continuous Delivery workflow that makes code sharing easy and regularized. This makes the process more transparent and collaborative among team members. In the long term, this makes the communication speed more efficient and makes sure that everyone in the organization is on the same page.
- 3. Higher Product Quality
- Continuous Integration provides features like Code review and Code quality detection, making the identification of errors easy. If the code does not match the standard level or a mistake, it will be alerted with emails or SMS messages. Code review helps the developers to improve their programming skills continually.

- 4. Reduced Waiting Time
- The time between the application development, integration, testing, and deployment is considerably reduced. When this time is reduced, it, in turn, reduces the waiting time that may occur in the middle. CI makes sure that all these processes continue to happen no matter what.
- We came across three different terms, Continuous Integration, Continuous Deployment, and Continuous Delivery. We must have a look at the difference between the three

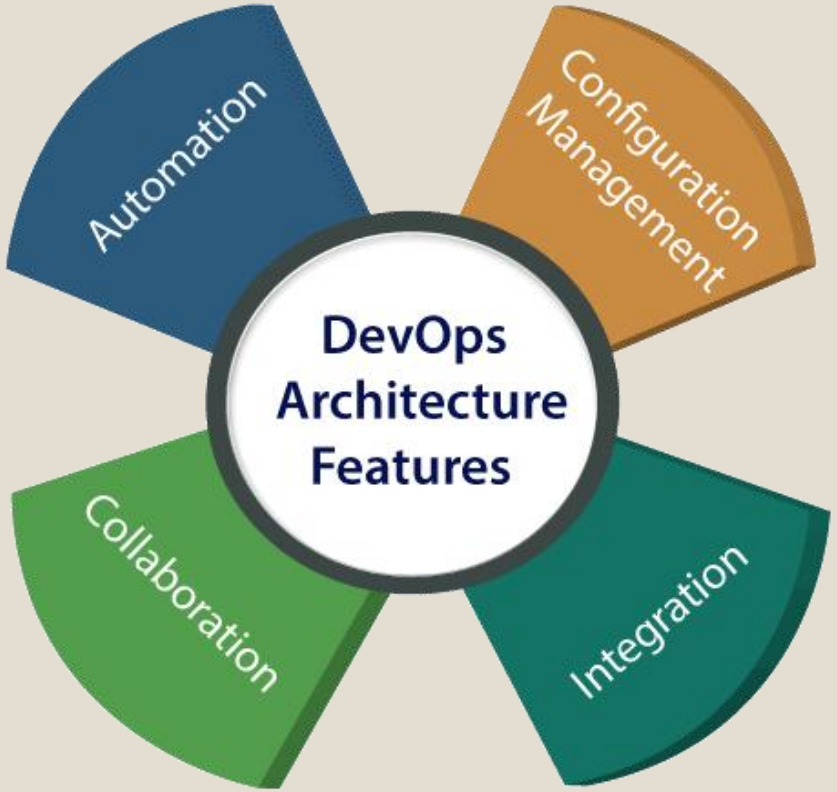
DevOps

- *DevOps is an approach to culture, automation, and platform design intended to deliver increased business value and responsiveness through rapid, high-quality service delivery. This is all made possible through fast-paced, iterative IT service delivery. DevOps means linking legacy apps with newer cloud-native apps and infrastructure.*
- **What is DevOps, anyway?**
- The word "DevOps" is a mashup of "development' and "operations" but it represents a set of ideas and practices much larger than those two terms alone, or together. DevOps includes security, collaborative ways of working, data analytics, and many other things. But what is it?
- DevOps describes approaches to speeding up the processes by which an idea (like a new software feature, a request for enhancement, or a bug fix) goes from development to deployment in a production environment where it can provide value to the user. These approaches require that development teams and operations teams communicate frequently and approach their work with empathy for their teammates. Scalability and flexible provisioning are also necessary. With DevOps, those that need power the most, get it—through self service and automation. Developers, usually coding in a standard development environment, work closely with IT operations to speed software builds, tests, and releases—without sacrificing reliability.

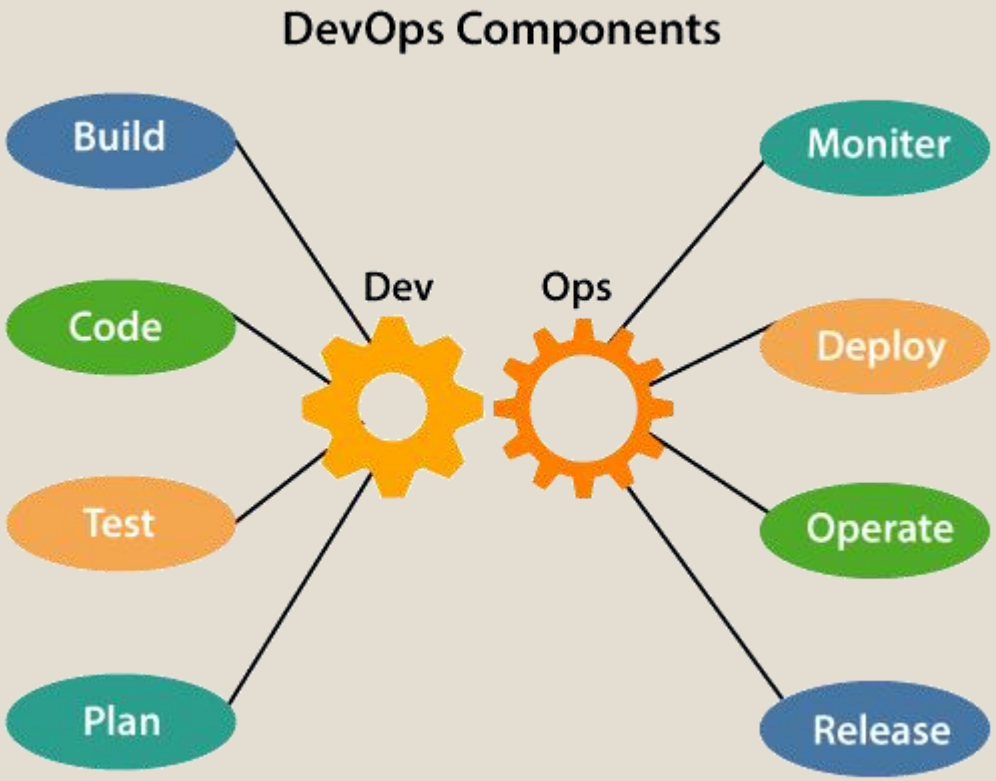
Why DevOps?

- Before going further, we need to understand why we need the DevOps over the other methods.
  - The operation and development team worked in complete isolation.
  - After the design-build, the testing and deployment are performed respectively. That's why they consumed more time than actual build cycles.
  - Without the use of DevOps, the team members are spending a large amount of time on designing, testing, and deploying instead of building the project.
  - Manual code deployment leads to human errors in production.
  - Coding and operation teams have their separate timelines and are not in synch, causing further delays.

DevOps Architecture Features



DevOps Components



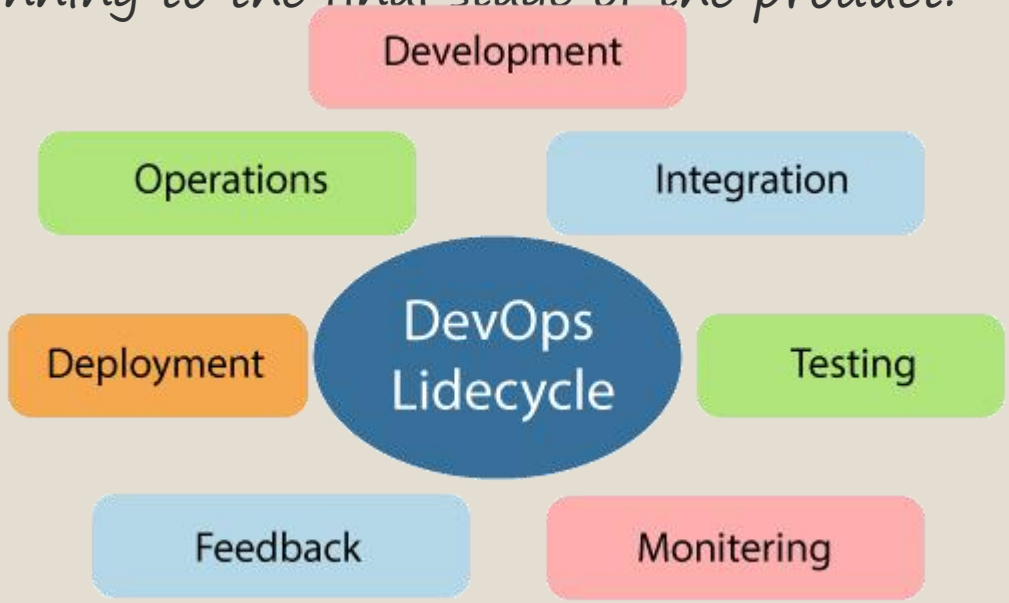
- 1) Build
- Without DevOps, the cost of the consumption of the resources was evaluated based on the pre-defined individual usage with fixed hardware allocation. And with DevOps, the usage of cloud, sharing of resources comes into the picture, and the build is dependent upon the user's need, which is a mechanism to control the usage of resources or capacity.
- 2) Code
- Many good practices such as Git enables the code to be used, which ensures writing the code for business, helps to track changes, getting notified about the reason behind the difference in the actual and the expected output, and if necessary reverting to the original code developed. The code can be appropriately arranged in **files, folders**, etc. And they can be reused.

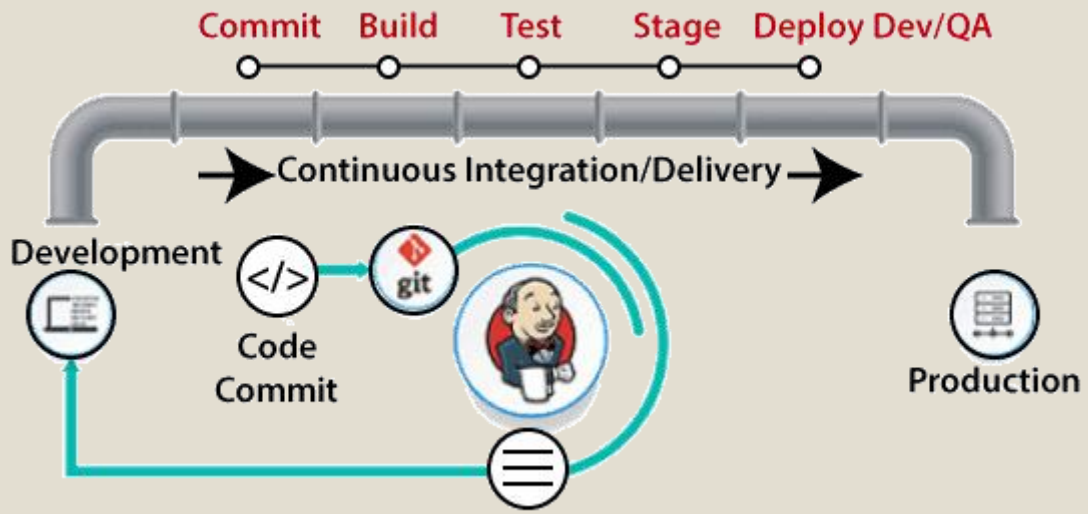
- 3) Test
- The application will be ready for production after testing. In the case of manual testing, it consumes more time in testing and moving the code to the output. The testing can be automated, which decreases the time for testing so that the time to deploy the code to production can be reduced as automating the running of the scripts will remove many manual steps.
- 4) Plan
- DevOps use Agile methodology to plan the development. With the operations and development team in sync, it helps in organizing the work to plan accordingly to increase productivity.
- 5) Monitor
- Continuous monitoring is used to identify any risk of failure. Also, it helps in tracking the system accurately so that the health of the application can be checked. The monitoring becomes more comfortable with services where the log data may get monitored through many third-party tools such as **Splunk**.

- *6) Deploy*
- Many systems can support the scheduler for automated deployment. The cloud management platform enables users to capture accurate insights and view the optimization scenario, analytics on trends by the deployment of dashboards.
- *7) Operate*
- DevOps changes the way traditional approach of developing and testing separately. The teams operate in a collaborative way where both the teams actively participate throughout the service lifecycle. The operation team interacts with developers, and they come up with a monitoring plan which serves the IT and business requirements.
- *8) Release*
- *Deployment to an environment can be done by automation. But when the deployment is made to the production environment, it is done by manual triggering. Many processes involved in release management commonly used to do the deployment in the production environment manually to lessen the impact on the customers.*

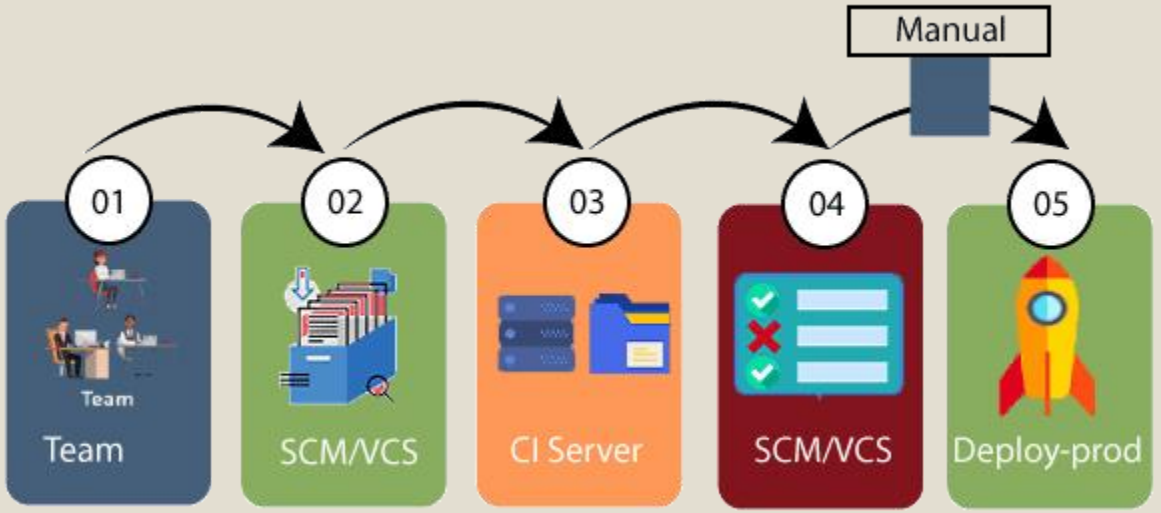
DevOps Lifecycle

- DevOps defines an agile relationship between operations and Development. It is a process that is practiced by the development team and operational engineers together from beginning to the final staae of the product.

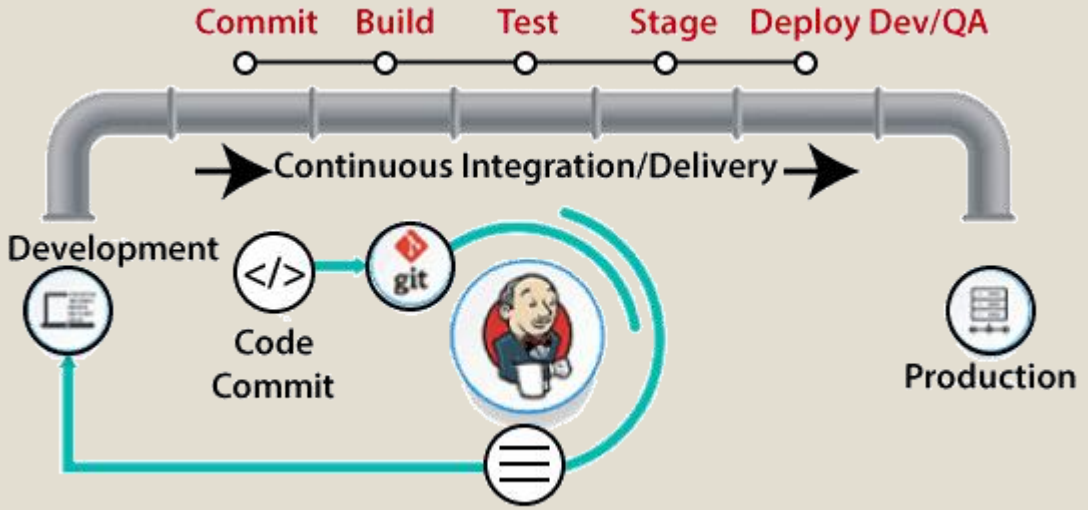



Commit
Build
Test
Stage
Deploy Dev/QA
Continuous Integration/Delivery
Development
Code
Commit
git
Production

- Learning DevOps is not complete without understanding the DevOps lifecycle phases. The DevOps lifecycle includes seven phases as given below:
- *1) Continuous Development*
- This phase involves the planning and coding of the software. The vision of the project is decided during the planning phase. And the developers begin developing the code for the application. There are no DevOps tools that are required for planning, but there are several tools for maintaining the code.
- *2) Continuous Integration*
- This stage is the heart of the entire DevOps lifecycle. It is a software development practice in which the developers require to commit changes to the source code more frequently. This may be on a daily or weekly basis. Then every commit is built, and this allows early detection of problems if they are present. Building code is not only involved compilation, but it also includes ***unit testing, integration testing, code review***, and ***packaging***.


Manual
01
02
03
04
05
Team
Team
SCM/VCS
CI Server
SCM/VCS
Deploy-prod

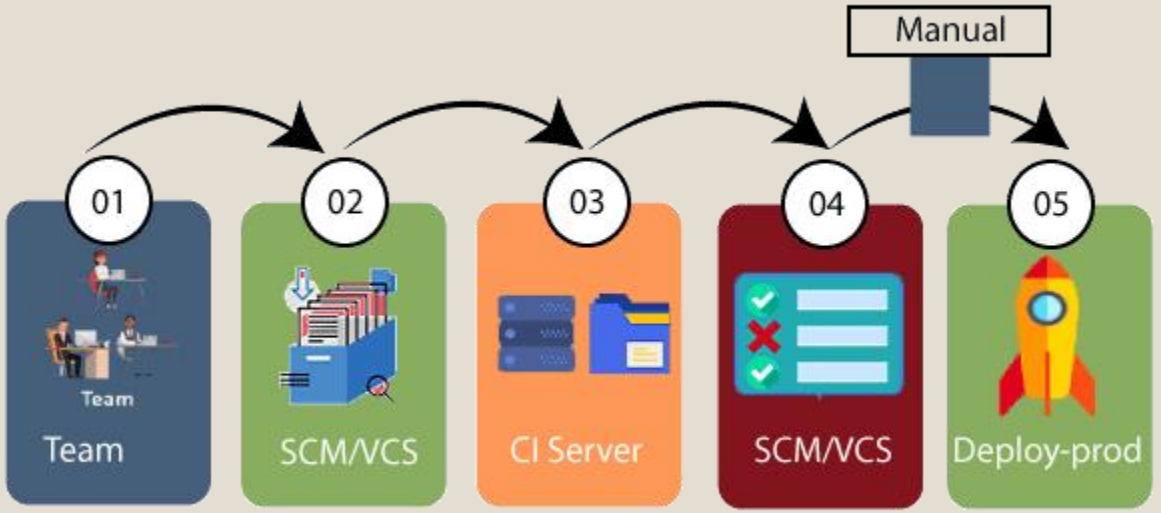
- The code supporting new functionality is continuously integrated with the existing code. Therefore, there is continuous development of software. The updated code needs to be integrated continuously and smoothly with the systems to reflect changes to the end-users.



- Jenkins is a popular tool used in this phase. Whenever there is a change in the Git repository, then Jenkins fetches the updated code and prepares a build of that code, which is an executable file in the form of war or jar. Then this build is forwarded to the test server or the production server.
- 3) Continuous Testing
- This phase, where the developed software is continuously testing for bugs. For constant testing, automation testing tools such as **TestNG, JUnit, Selenium**, etc are used. These tools allow QAs to test multiple code-bases thoroughly in parallel to ensure that th **cker** Containers can be used fo

- **Selenium** does the automation testing, and TestNG generates the reports. This entire testing phase can automate with the help of a Continuous Integration tool called **Jenkins**.
- Automation testing saves a lot of time and effort for executing the tests instead of doing this manually. Apart from that, report generation is a big plus. The task of evaluating the test cases that failed in a test suite gets simpler. Also, we can schedule the execution of the test cases at predefined times. After testing, the code is continuously integrated with the existing code.
- 4) Continuous Monitoring
- Monitoring is a phase that involves all the operational factors of the entire DevOps process, where important information about the use of the software is recorded and carefully processed to find out trends and identify problem areas. Usually, the monitoring is integrated within the operational capabilities of the software application.
- It may occur in the form of documentation files or maybe produce large-scale data about the application parameters when it is in a continuous use position. The system errors such as server not reachable, low memory, etc are resolved in this phase. It maintains the security and availability of the service.

- 5) Continuous Feedback
- The application development is consistently improved by analyzing the results from the operations of the software. This is carried out by placing the critical phase of constant feedback between the operations and the development of the next version of the current software application.
- The continuity is the essential factor in the DevOps as it removes the unnecessary steps which are required to take a software application from development, using it to find out its issues and then producing a better version. It kills the efficiency that may be possible with the app and reduce the number of interested customers.
- 6) Continuous Deployment
- In this phase, the code is deployed to the production servers. Also, it is essential to ensure that the code is correctly used on all the servers


Manual
01
02
03
04
05
Team
Team
SCM/VCS
CI Server
SCM/VCS
Deploy-prod

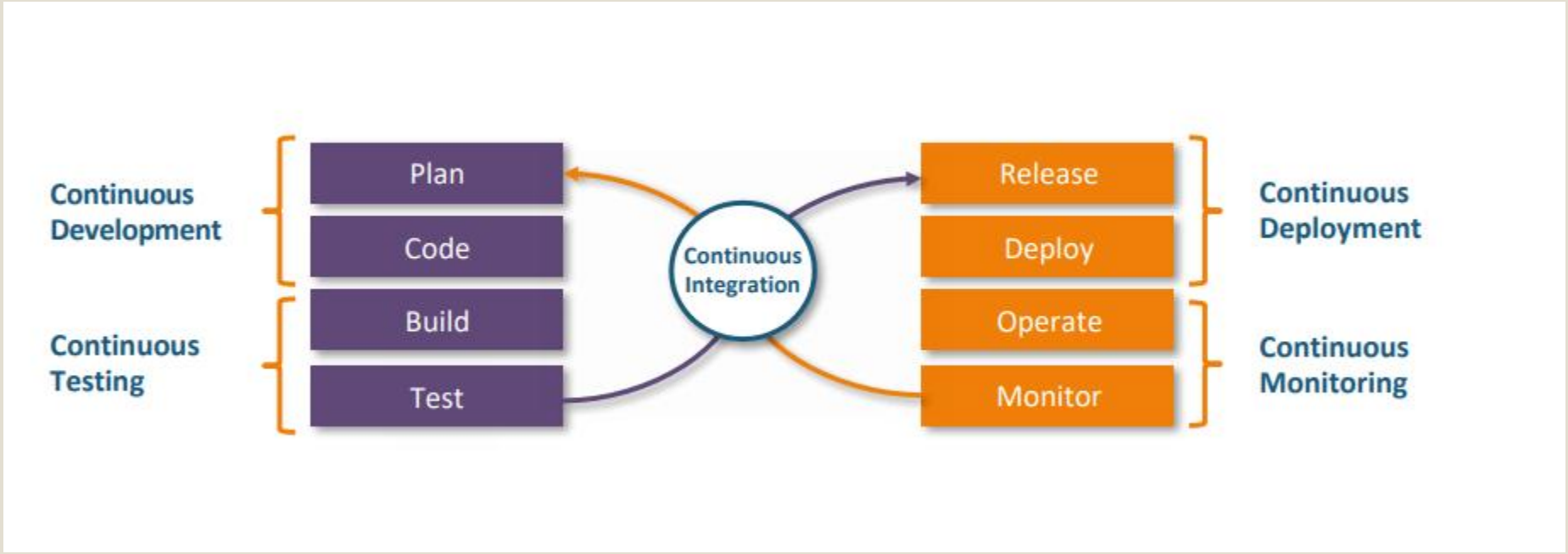
- The new code is deployed continuously, and configuration management tools play an essential role in executing tasks frequently and quickly. Here are some popular tools which are used in this phase, such as Chef, Puppet, Ansible, and SaltStack.

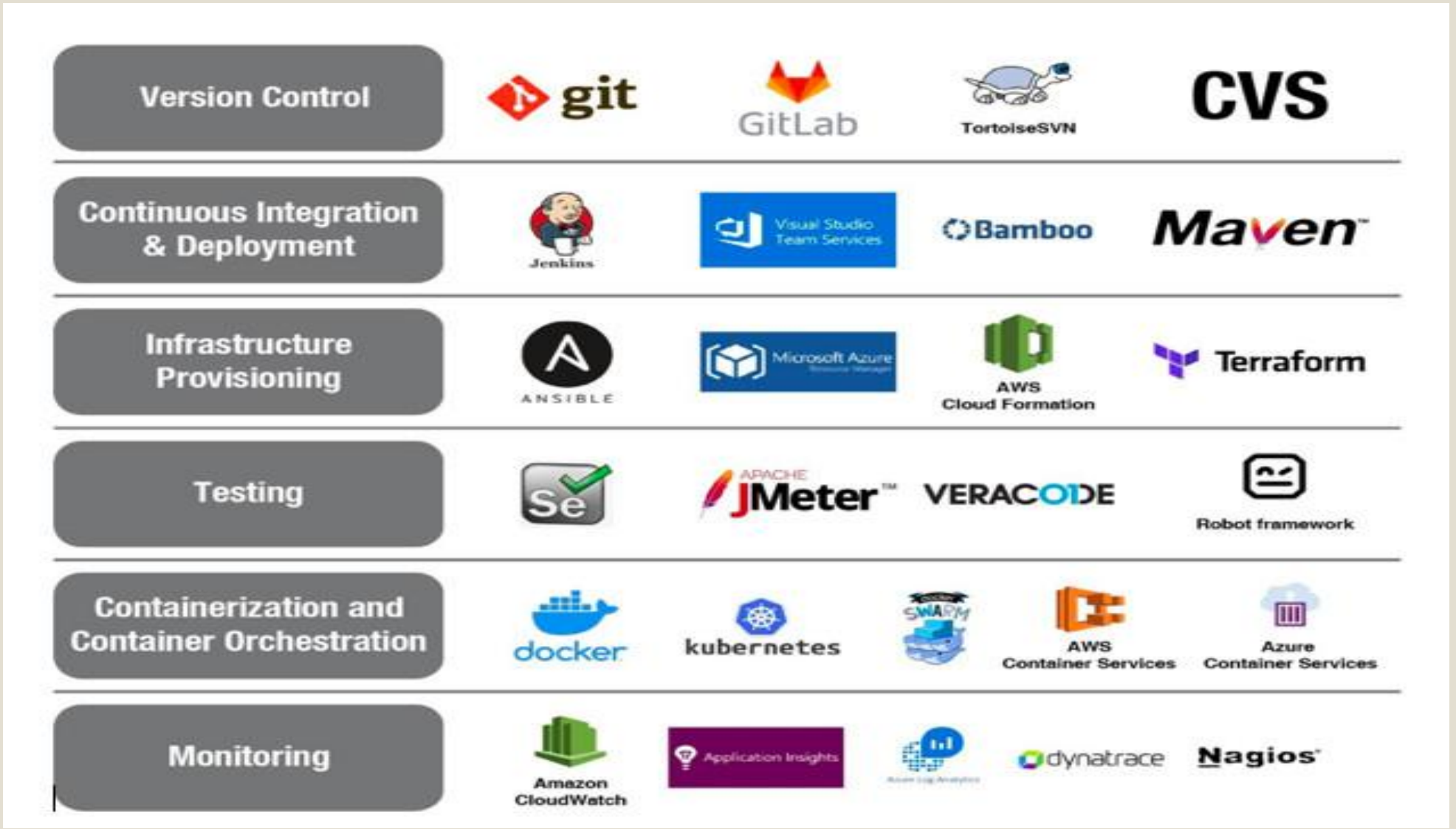
- Containerization tools are also playing an essential role in the deployment phase. Vagrant and Docker are popular tools that are used for this purpose. These tools help to produce consistency across development, staging, testing, and production environment. They also help in scaling up and scaling down instances softly.

- Containerization tools help to maintain consistency across the environments where the application is tested, developed, and deployed. There is no chance of errors or failure in the production environment as they package and replicate the same dependencies and packages used in the testing, development, and staging environment. It makes the application easy to run on different computers.

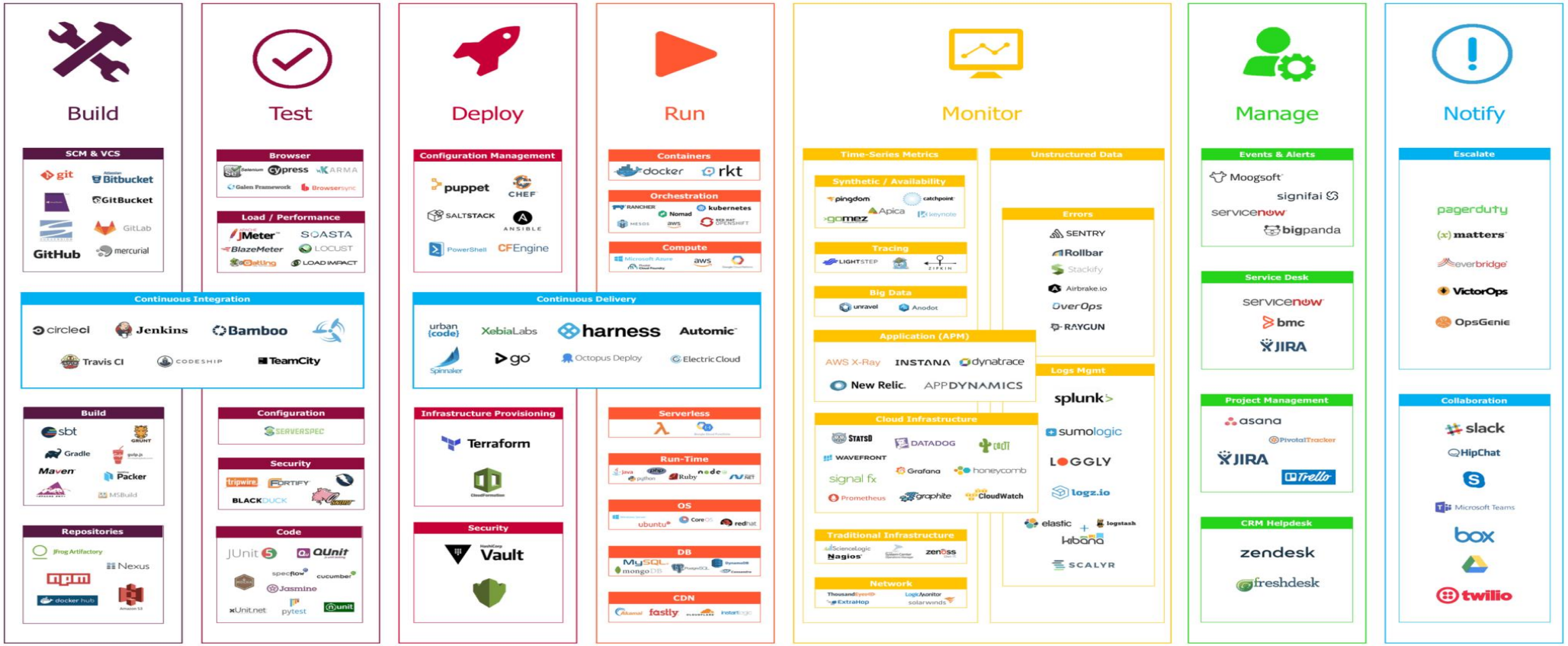
- 7) Continuous Operations
- All DevOps operations are based on the continuity with complete automation of the release process and allow the organization to accelerate the overall time to market continuingly.
- It is clear from the discussion that continuity is the critical factor in the DevOps in removing steps that often distract the development, take it longer to detect issues and produce a better version of the product after several months. With DevOps, we can make any software product more efficient and increase the overall count of interested customers in your product.

DevOps Tools


Continuous Development
Continuous Testing
Plan
Code
Build
Test
Continuous Integration
Release
Deploy
Operate
Monitor
Continuous Deployment
Continuous Monitoring


Version Control
git
GitLab
TortoiseSVN
CVS
Continuous Integration & Deployment
Jenkins
Visual Studio Team Services
Bamboo
Maven
Infrastructure Provisioning
ANSIBLE
Microsoft Azure
AWS Cloud Formation
Terraform
Testing
APACHE JMeter™
VERACODE
Robot framework
Containerization and Container Orchestration
docker
kubernetes
SWARM
AWS Container Services
Azure Container Services
Monitoring
Amazon CloudWatch
Application Insights
dynatrace
Nagios

DevOps Lifecycle Mesh



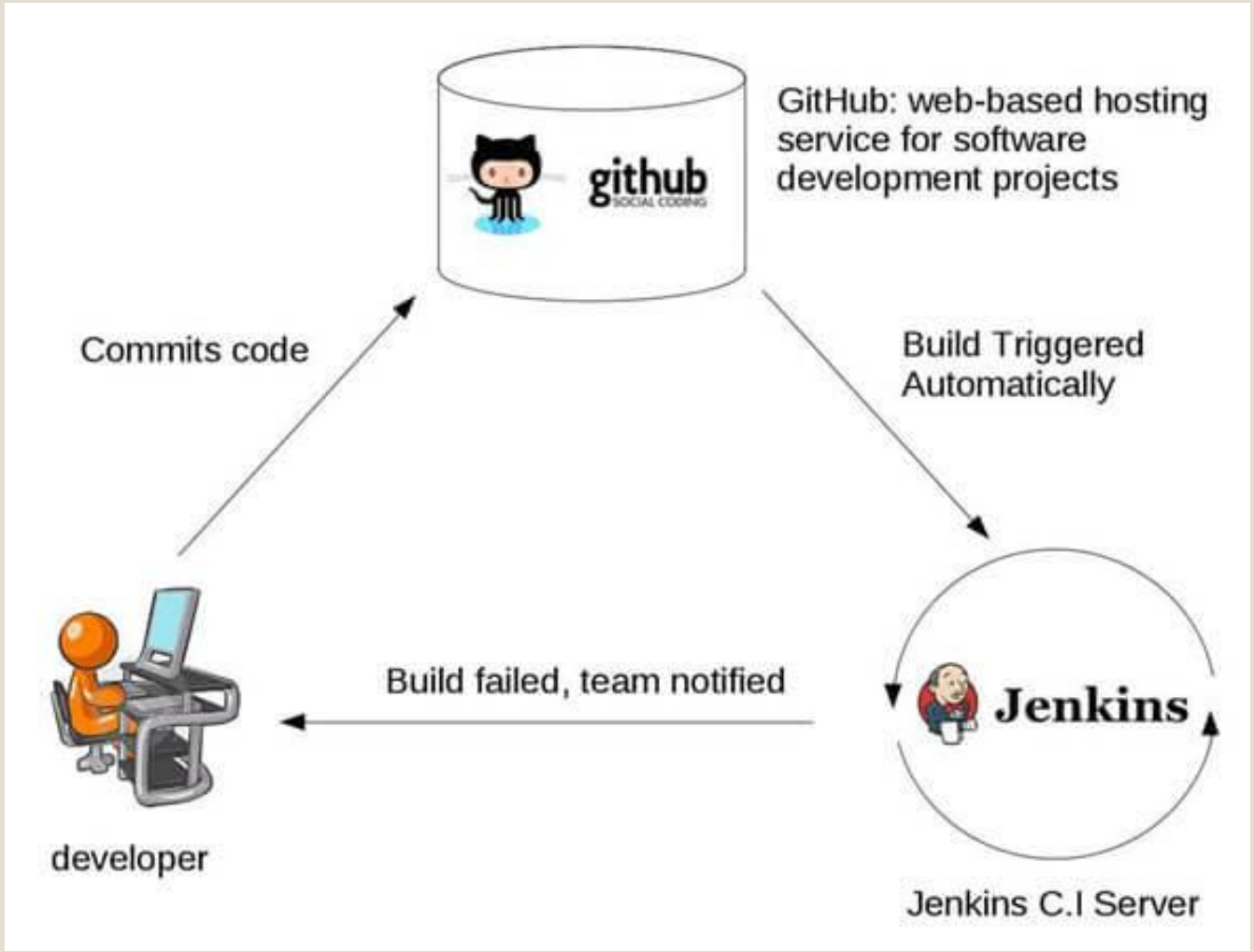
harness

Continuous Delivery As–A-Service

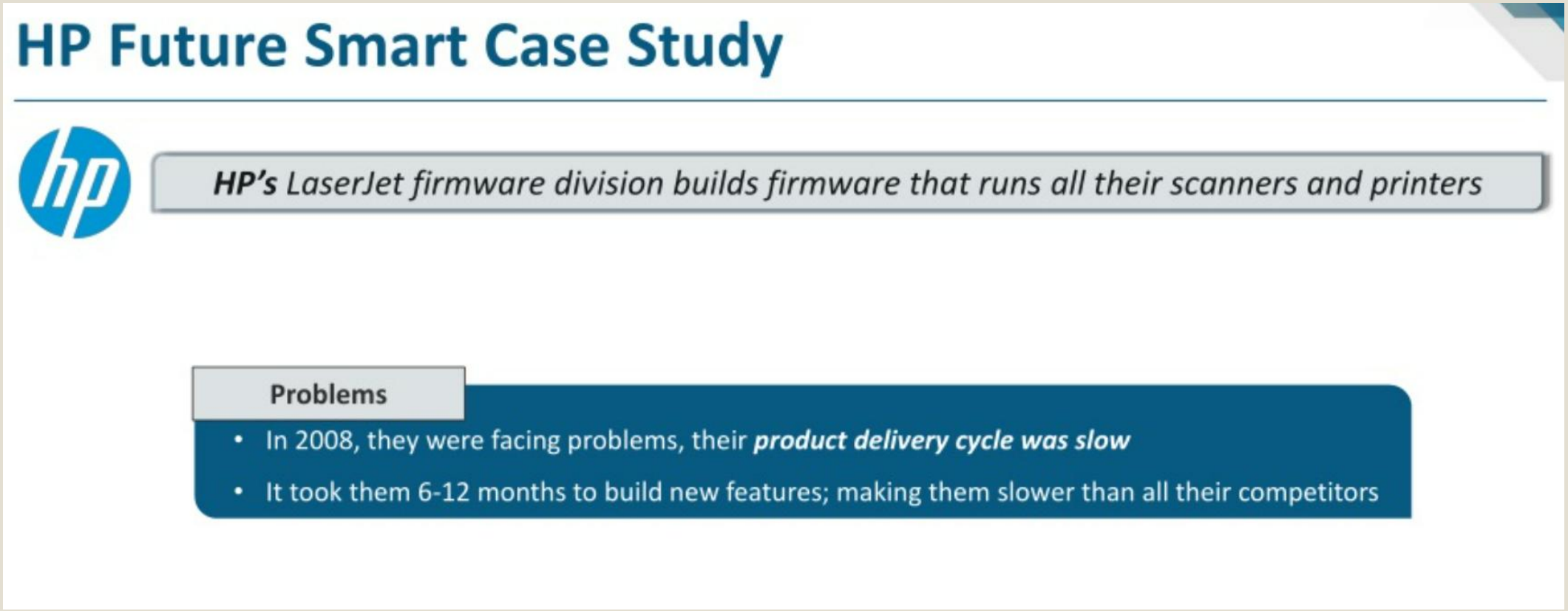
www.harness.io

Real-world case study of Continuous Integration

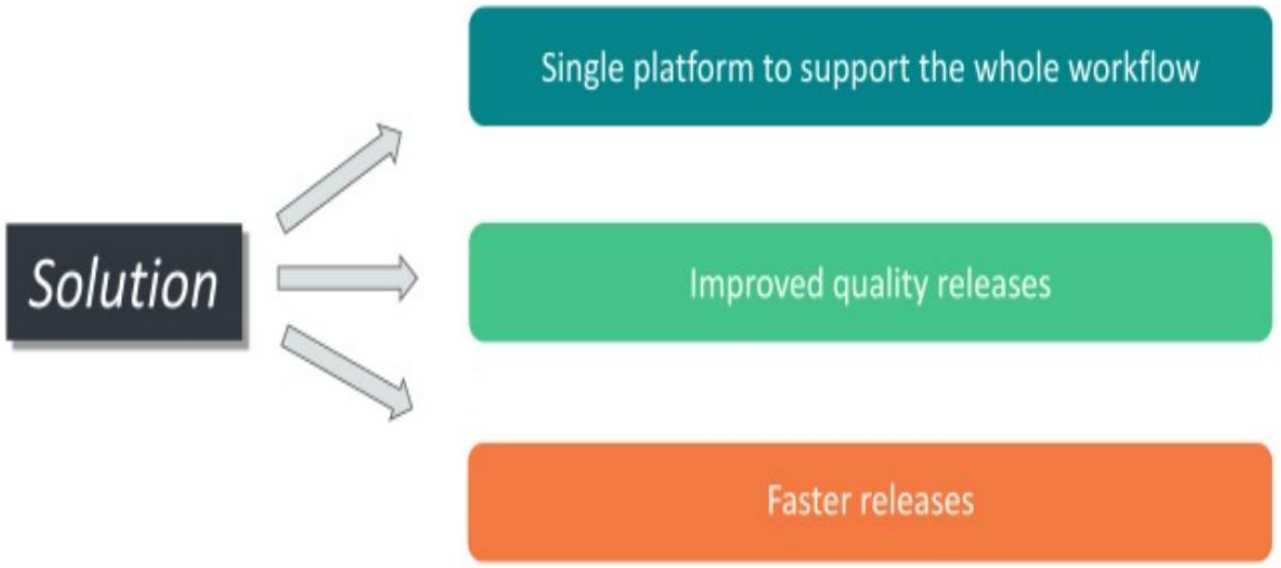
- I am sure all of you aware of old phone Nokia. Nokia used to implement a procedure called nightly build. After multiple commits from diverse developers during the day, the software built every night. Since the software was built only once in a day, it’s a huge pain to isolate, identify, and fix the errors in a large code base.
- Later, they adopted Continuous Integration approach. The software was built and tested as soon as a developer committed code. If any error is detected, the respective developer can quickly fix the defect.
-


github
SOCIAL CODING
GitHub: web-based hosting service for software development projects
Commits code
Build Triggered Automatically
Build failed, team notified
Jenkins
developer
Jenkins C.I Server

CASE STUDY


HP Future Smart Case Study
HP's LaserJet firmware division builds firmware that runs all their scanners and printers
Problems
• In 2008, they were facing problems, their product delivery cycle was slow
• It took them 6-12 months to build new features; making them slower than all their competitors

HP Future Smart Case Study



HP Future Smart Case Study

How did they achieve this?

They implemented a CD pipeline with two key features

- Practice CI
- Automation at every step

- Overall development cost reduced ~ 40%
- Programs under development increased ~ 140%
- Development cost per program went down ~ 78%

Before Jenkins Pipeline

Over the years, there have been multiple Jenkins pipeline releases including, **Jenkins Build flow**, **Jenkins Build Pipeline plugin**, **Jenkins Workflow**, etc.

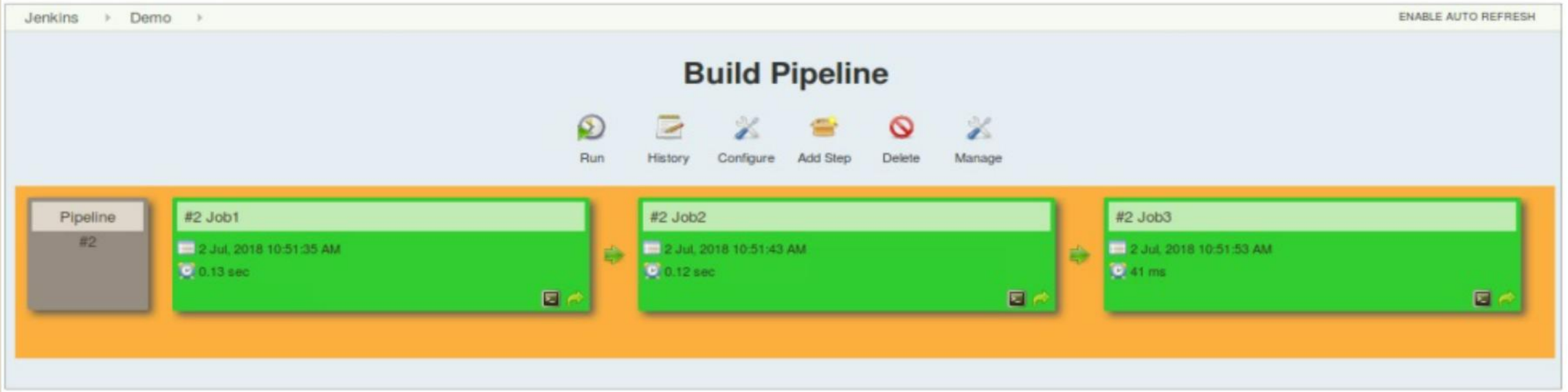
What are the key features of these plugins?

- Represent multiple Jenkins jobs as one pipeline
- What do these pipelines do?

  These pipelines are a collection of Jenkins jobs which trigger each other in a specified sequence

Build Plugin Pipeline Example

- **3 jobs**: Job1 → building, Job2 → testing the application and Job3 → deployment
- Chain these jobs together & run using Build Pipeline Plugin (better for small deployment)



Limitations Of The Build Pipeline Plugin

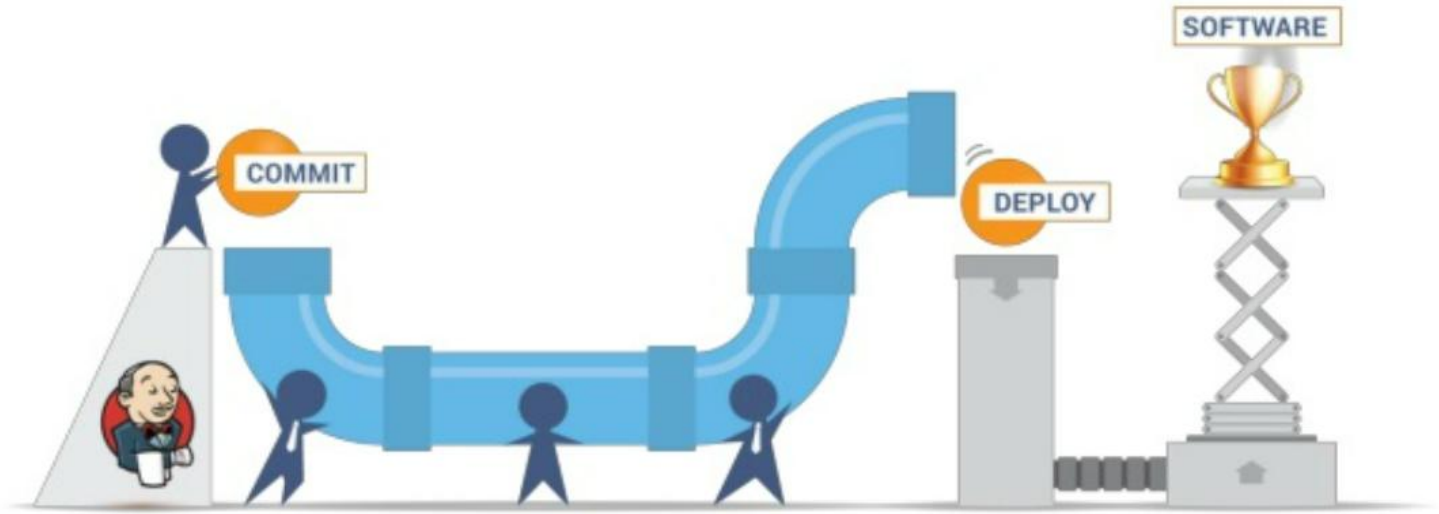
Complex pipelines are hard to implement

The maintenance cost is huge and increases with the number of processes

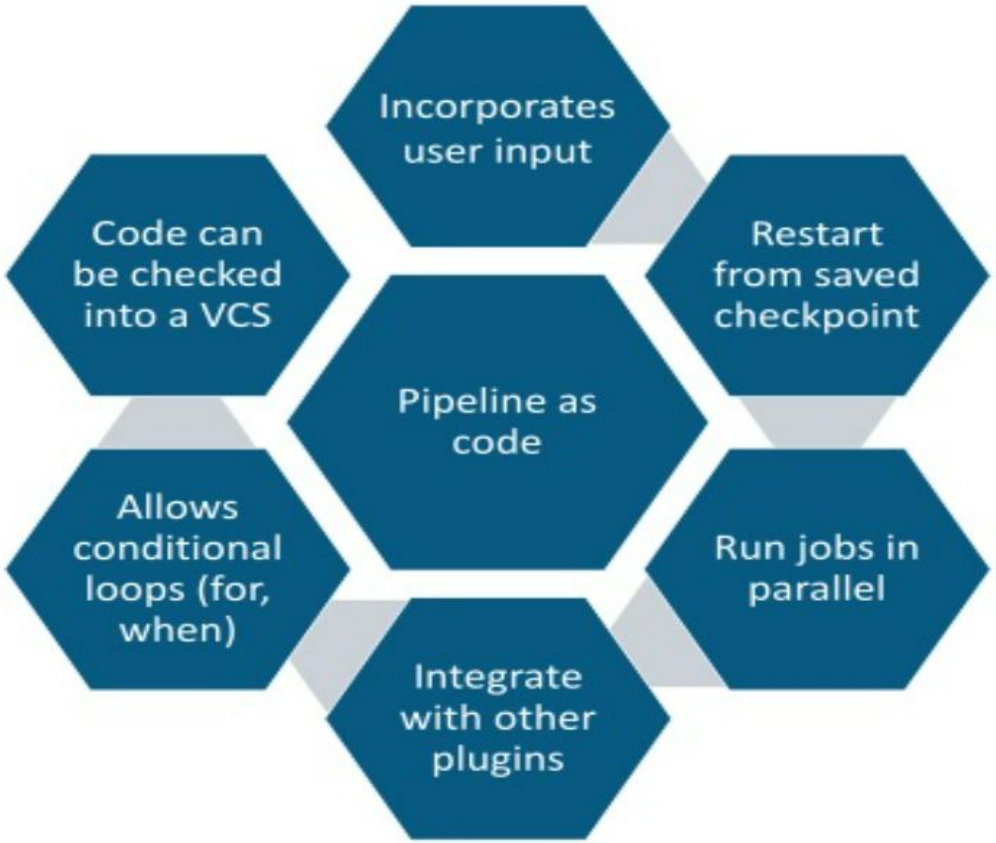
Tedious to build and manage such a vast number of jobs

What Is A Jenkins Pipeline?

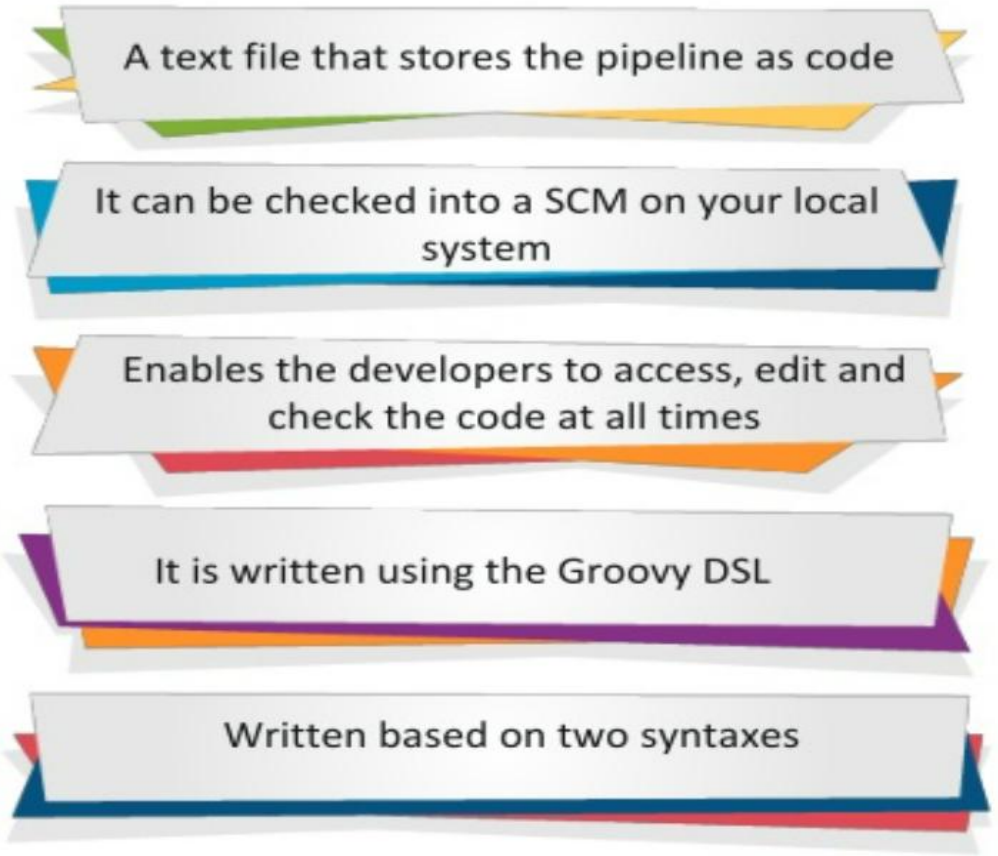
- **Jenkins pipeline** is a single platform that runs the entire *pipeline as code*
- All the standard jobs defined by Jenkins are manually written in one script and they can be stored in a VCS
- Instead of building several jobs for each phase, you can now code the entire workflow and put it in a Jenkinsfile



Key Features Of Jenkins Pipeline



What Is A Jenkinsfile?



Two Ways Of Writing Jenkinsfile

DECLARATIVE PIPELINE

- Recent feature
- Simpler groovy syntax
- Code is written locally in a file and is checked into a SCM
- The code is defined within a 'pipeline' block

SCRIPTED PIPELINE

- Traditional way of writing the code
- Stricter groovy syntax
- Code is written on the Jenkins UI instance
- The code is defined within a 'node' block

Pipeline Concepts

Pipeline: A user defined block which contains all the stages. It is a key part of declarative pipeline syntax.

Note: Stages are explained in the following slides

```
pipeline {

}
```

Node: A node is a machine that executes an entire workflow. It is a key part of the scripted pipeline syntax.

```
node {

}
```

Pipeline Concepts

Agent: instructs Jenkins to allocate an executor for the builds. It is defined for an entire pipeline or a specific stage.
It has the following parameters:

- *Any*: Runs pipeline/ stage on any available agent
- *None*: applied at the root of the pipeline, it indicates that there is no global agent for the entire pipeline & each stage must specify its own agent
- *Label*: Executes the pipeline/stage on the labelled agent.
- *Docker*: Uses docker container as an execution environment for the pipeline or a specific stage.

```
pipeline {
    agent {
        docker {
            image 'ubuntu'
        }
    }
}
```

Pipeline Concepts

Stages: It contains all the work, each stage performs a specific task.

```
pipeline {
        agent any
        stages {
                stage ('Build') {
                        ...
                }
                stage ('Test') {
                        ...
                }
                stage ('QA') {
                        ...
                }
                stage ('Deploy') {
                        ...
                }
                stage ('Monitor') {
                        ...
                }
        }
}
```

Steps: steps are carried out in sequence to execute a stage

```
pipeline {
        agent any
        stages {
                stage ('Build') {
                        steps {
                                echo 'Running build phase...'
                        }
                }
        }
}
```

Using environment variable

Jenkins Pipeline exposes environment variables via the global variable env, which is available from anywhere within a Jenkinsfile. The full list of environment variables accessible from within Jenkins Pipeline is documented at ${YOUR_JENKINS_URL}/pipeline-syntax/globals#env and includes:

- BUILD_ID
  - The current build ID, identical to BUILD_NUMBER for builds created in Jenkins versions 1.597+
- BUILD_NUMBER
  - The current build number, such as "153"
- BUILD_TAG
  - String of jenkins-${JOB_NAME}-${BUILD_NUMBER}. Convenient to put into a resource file, a jar file, etc for easier identification
- BUILD_URL
  - The URL where the results of this build can be found (for example http://buildserver/jenkins/job/MyJobName/17/)

- EXECUTOR_NUMBER
  - The unique number that identifies the current executor (among executors of the same machine) performing this build. This is the number you see in the "build executor status", except that the number starts from 0, not 1
- JAVA_HOME
  - If your job is configured to use a specific JDK, this variable is set to the JAVA_HOME of the specified JDK. When this variable is set, PATH is also updated to include the bin subdirectory of JAVA_HOME
- JENKINS_URL
  - Full URL of Jenkins, such as https://example.com:port/jenkins/ (NOTE: only available if Jenkins URL set in "System Configuration")
- JOB_NAME
  - Name of the project of this build, such as "foo" or "foo/bar".
- NODE_NAME
  - The name of the node the current build is running on. Set to 'master' for the Jenkins controller.
- WORKSPACE
  - The absolute path of the workspace