3

UNIT - III

Continuous Integration and Test-Driven Development

Syllabus

Introduction to continuous integration, time to market and quality, Build in a Continuous Integration Scenario, Code Repository Server, Continuous Integration Server, Introduction to Continuous Delivery and chain, Differentiate Continuous Integration and Continuous Delivery, Strategies for Continuous Delivery, Benefits of Continuous Integration and Continuous Delivery, Designing a CI and CD System, Building Continuous Integration and Continuous Delivery Pipelines, Continuous Database Integration, Preparing the Build for Release, Identifying the Code in the Repository, Creating Build Reports, Putting the Build in a Shared Location, Releasing the Build

3.1 Introduction to Continuous Integration (CI)

- Continuous integration is a DevOps practice where developers merge their updated code regularly into code repository, which is followed by automated build and tests. CI refers to automated practice in software development which integrates codes from multiple contributors into a single software project. Automated tools help us to assert new code's correctness before integration.
- A source code version system is a central tool for CI activity. A source code version system is also supported with other checks like code quality check, syntax styling review, code deployment environment feasibility and many more. Continuous integration refers to the build or integration phase of the software release process. The main objective of CI is to find bugs early in the release process to improve product quality and reduce time required for validation and releasing new versions.
- What is the importance of CI ? To understand the importance of CI, let's consider a scenario wherein CI is absent. Every time a new feature is developed, the development team coordinates with other teams in the organization including the operation team and release team.

- The extended communication within the organization delays bugs identification, bugs fixing and release of new features.
- In the past, developers might work in isolation and only merge their code to master branch once the code changes or updates were ready. The time consuming isolated working environment takes more time to merge code, detect bugs and fix them for before expected release time. A non-CI pipeline environment becomes complex and costly practice by adding unnecessary bureaucratic cost to projects. Disadvantages of the Non-CI environment is slower Software release and higher rate of failure as number of bugs can be more after integration.

CI offers some organization level benefits as follow :

- **Scaling :** Organization can scale in engineering team size, codebase size and infrastructure. CI helps build agile and DevOps workflow by minimizing communication overhead and integration bureaucracy.
- **Improved feedback Loop** : One of the most powerful effects of CI is faster feedback on business decisions. Product team can test ideas and iterate product designs faster. Code changes can be rapidly pushed and measured for success.
- **Enhanced Communication** : Greater collaboration between development team and operations can improve engineering communication and accountability. A pull request workflow brings in passive knowledge gain across the development team.

3.2 Time to Market and Quality

- IT organizations compete to stay on top of the market in product sales and quality product development. Speed and flexibility have become of paramount importance for businesses. Building and releasing new products is not sufficient in the current scenario for organizations to be on top of competition. Organizations use an agile approach to understand and adapt to information, to meet demanding time-to-market features. With tremendous changes in build and release process, organizations adopt new developments and trends in the market.
- DevOps is a cultural shift in the IT industry which shows how software development and operations teams can collaborate and work together. The collaborative teams improve quality of product and release new features as early as possible to reduce time to market. There are many ways DevOps helps organizations accelerate the release process, reducing time to market for their product.

- Some of the effective ways opted by organizations are :
  - **Faster Delivery of Changes / Modifications** : On each change from development team, IT operations team has to set a test environment or has to add new interfaces, test cases and side applications for testing the service code provided by developers. Setting the environment each time and then performing testing in the proposed environment is a time-consuming process and results in delayed release of new features or new products. DevOps eliminate such dependencies and fasten the release process by automating test, deployment and bug or defect identification process. Organizations can deliver small or incremental updates with the help of DevOps whenever required.
  - **Enhance Efficiency** : DevOps encourage development teams to spend more time on value-creating tasks. Developers need not wait for setting up a test environment and integration of modules. It also reduces the workload of the IT operations team by automating integration and test environment settings for each incremental release. Example, European banks have achieved more than 25% efficiency in internet banking service with the help of DevOps.
  - **Improved code quality and Quick recovery** : Once the software is released to market, developers are engaged in the next project or new module development. They have no motive to predict or prevent future failures or problems in software, leaving it to the operations team. DeOps involve developers throughout the project lifecycle, which result in high quality of features. ONly few fixes result in improved quality of code and quick recovery in case of failure. Continuous feedback helps detect any chance of failure and immediate action on such feedback minimizes failure percentage. Also human errors can be reduced, thanks to automation provided by DevOps for the development and release process.

3.3 Build in a Continuous Integration Scenario

- The pipeline example illustrated here is a sample pipeline which can be used to implement CI processes in general with or without modifications. Regardless of the tools used to implement CI processes, these steps will help you to build a CI strategy for your application.
- CI process is used to integrate developer codes into a shared repository. Shared repository can be hosted locally or it can be a non-hosted repository. Each integration is then tested for verification on successful builds. Each build is a new version of software with some modification in features or addition of new features.

- Each integration should be tested with automated tests so that errors and bugs can be detected early in the software development lifecycle and modifications can be done before releasing the software into production.
- To implement a CI process, first we need to install a code version control system. All developers should push the code to VCS after each change. As soon as changes are pushed to the repository, an automated build process is initiated in the pipeline. Once the build is ready, automated tests are fetched from VCS and then executed to test build for any error due to new changes. As new changes are small, pinpointing any error due to new change is very easy.
- Continuous integration can be implemented by simply adhering to following practices :
  - Maintain a single source repository
  - Automate the Build
  - Self-testing of Build
  - Everyone commits to the mainline every day
  - Every commit should build the mainline on an Integration Machine
  - Fix Broken Builds immediately
  - Keep the build fast
  - Test in a production like environment
  - Make latest executable easily available for all
  - Everyone can see what's happening
  - Automate Deployment

3.4 Code Repository Server

- Source code repository is one of the important tools in the Developer's toolkit. A good source code repository is useful for DevOps team collaboration and working with open source projects. There are a number of source code repository hosts available ranging from widely used BitBucket and GitHub to lesser known but yet useful repo hosts.
- Each host has its specific targeted user and specific projects like open source projects, multi-developer projects, private repo hosts and many more. Having a wide range of hosts gives a luxury to choose one host for a project. The problem lies with the selection of the best host for the project from a number of hosts.

- Version control system and repository hosting services are two different entities. Repository hosting services wrap and enhance the version control system and provide web based applications for using version control system features through command line or interface. Version control systems are command line utilities which control and manage software development lifecycle changes with respect to collection of source code files.
- You cannot develop or use a repository hosting service to full extent without use of the underlying version control system. Repository hosting services are available as web applications and some of those can be used locally to enhance project security. Bitbucket allows your team to securely and privately host the sensitive project of organization within the VPN and other internal networks.

3.5 Continuous Integration Server

- CI server, also known as a build server, is used to manage shared repositories. CI server works as referee for the code coming in. Every time the developer commits code changes to the expository, CI server initiates the build process and dreates document of build results. Developers who committed code to the repository will receive an email notification with the build result document. The majority of teams building software using CI approach, rely on CI server to automate builds.
- The CI server gives more control over source code, testing and commit processes. Teams that bypass CI server, build and testing has to be done manually and becomes a time consuming process with additional overhead of environment setting and code integration.
- Majority of tools used as continuous integration tools are CI servers. Some of those offer some additional features like source code management, continuous delivery, testing, continuous feedback and more. Some of the CI servers are : Jenkins, GitLab CI, Circle CI.

CI server offers certain process advantages :

- **Automated test** : As soon as code is committed, CI server initiates the code testing and provides immediate feedback to committer for better quality and improvements. The automated tests reduce manual build process load from developer tasks.
- **Better collaboration** : CI server maintains integrity of all the code coming in. Through automated builds and tests, CI server maintain integrity and help developers focus on their own project instead of worrying about other projects due to which test cases are failing.
- **Streamlined workflow** : With source code saved on a local machine, developers work on various layers of software applications. Teams with manual testing approaches will come across this potential issue and this issue can be resolved using CI server. CI servers eliminate this extra layer of coordination from workflow.

3.6 Introduction to Continuous Delivery and Chain

- Continuous delivery refers to a process where developers' changes are automatically tested for bugs and uploaded to a code repository, from where they are deployed to production-like environments. It resolves the issue of poor visibility and communication between development and operation teams. Continuous delivery ensures that it will take minimal effort to deploy new code.
- In Continuous delivery (CD), which is an extension to continuous integration, "developers hand off the code to the QA and operations team for testing and deployment". With Continuous Delivery, developers can build, test and deploy software anytime and more frequently. As a result, software release time, risk and cost is reduced. CD picks up where CI leaves off. CD focuses on automated deployment of successfully tested, validated and verified builds.
- CD is heavily dependent on tools for automated testing and deployment. The validation of requirements is verified before deployment of builds using CD pipeline. Small incremental iteration ensures any problem revealed in testing is quickly remediated and fixed with minimal efforts and is less costly. Various steps involved in continuous delivery are : Functional testing, User acceptance testing and configuration automation, load testing and deployment.
- Source code has already passed static testing and now the completed build is ready to enter the next phase of advance testing in CD. It usually start with functional or unit testing of new code followed by regression testing to ensure new changes will not affect existing features and application will behave as intended. After successfully passing subsequent testing like user acceptance testing, integration and performance testing, build is considered as a valid candidate for deployment in a production environment.
- In continuous delivery, these builds are passed to humans for approval and deployment. Whereas, in continuous deployment, the build is automatically deployed in production as soon as it passes the test suite. The deployment step involves creating a deployment environment by provisioning resources and services, and moving the validated build to deployment environment. These steps can be scripted or set using workflows in tools. Also Continuous Deployment is connected to error tracking tools to monitor live errors and alert developers with error messages. Users can submit errors or bugs and raise tickets to inform perceived errors.

3.7 Differentiate Continuous Integration and Continuous Delivery

DevOps is a software development activity consisting of automated build, test and deploy processes with error reporting and automated immediate feedback feature to develop high quality software.

Continuous Integration	Continuous Delivery	Continuous Deployment
Developers merge their code back to the main branch as often as possible. New code is validated by creating builds and running automated tests. This avoids waiting time for merging code on release day.	It is an extension of continuous integration. It automatically deploys new code to the production or testing environment after the build stage.	Continuous deployment is one step ahead of continuous delivery. No human intervention is required to deploy new code passed all stages of the production pipeline. Only failed tests will prevent a new code from deployment.
CI focuses on testing automation to ensure application does not break on integration of new commits with the main branch.	On top of automated testing, you have an automated release process to deploy application any time by just clicking on button.	It will accelerate the feedback loop and take pressure off the team as no release day exists.
	You decide release day, from the options like daily, weekly, monthly or whenever required based on business need. With immediate release to production, you can avail all benefits of continuous delivery.	Developers can focus on development of features and they can see their feature or module going live immediately after the commit.

Continuous Integration	Continuous Delivery	Continuous Deployment
Cost of CI stage : • Team should write automated tests for each feature, bug fixing or for improvement over existing features. • Requires a CI server to monitor code repository and execution of automated tests for each commits pushed. • Developers should merge their code changes as early as possible, often daily once.	Cost of CD stage : • Strong foundation in CI along with sufficient test suite to cover enough of the codebase. • Trigger is manual and deployment is automated. Once deployment starts, no human intervention is required. • Team members will embrace feature flags to avoid incomplete features affecting your application.	Cost of CD stage : • Quality of test culture determines quality of product release. • The documentation process and deployment pace should be lined up. • Features flags make sure to coordinate with other departments.
Gain : • Less bugs shipped to production due to early regression captured by tests. • Building the release is possible with minimum efforts. • Less context switching : developers are alerted as soon as build breaks and developers start fixing bugs before moving ahead with the next feature. • Less testing cost : CI server can run hundreds of tests within seconds. • The QA team focuses on significant improvements to the quality culture and spends less time on testing.	Gain : • Team doesn't spend days preparing for release and hence deployment complexity is removed. • Often releases accelerate feedback loops. • Faster iterating with less pressure for small changes in code or feature.	Gain : • Faster development - no need to pause for release. Deployments pipelines are triggered for each update. • Less risky release - small changes are deployed every time which reduces release risk. • Continuous improvements - customers can experience continuous improvements in software everyday instead of every month, quarter or year.

3.8 Strategies for Continuous Delivery

- Continuous delivery is a DevOps methodology in which new builds are deployed in various environments with predefined test suites and without any downtime due to upgrade or updates on software. Continuous delivery is highly dependent on automated test results and successful builds from new code pushes. Automated tests are highly required for successful deployment and delivery of new builds. Without automated tests, with high quality of tests in the deployment pipeline, production level deployment errors may occur. Code coverage was considered as a good representative of the quality of the code, but in the cloud native world, it's not sufficient for code quality. In a cloud environment, test cases with test suites should be mapped to business functionality and business outcomes to ensure that everything goes correct in testing, deployment and production environments. In situations where a comprehensive approach is prevented by time limits on delivery and deployments, first automate the critical functional capabilities and workflow, and then perform cost assessment to identify benefits of automated tests.
- To successfully deliver software builds to production, always ensure closeness of data and environment for testing as that of production, as much as possible. Most pipelines follow a pre-production staging environment for testing purposes and then push software to the production environment. You may build multiple stages across multiple regions of a cloud service provider to test feasibility, latency and performance before pushing to production.
- If you are planning to frequently update software in the production environment, use a deployment tool which can manage multi region deployment easily. Multi region deployments are possible with the help of blue/green deployment and canary deployment, to reduce or minimize downtime during software update.
- Development teams can benefit from following these key strategies.
    - **Culture :** A cultural shift in development teams can lead to successful adoption of continuous delivery processes. Developers are now responsible for development, testing, automation tools and, most importantly, for building operational capabilities into the pipeline for easy move to production environment. With this, onboarding your team is a hard and time consuming process, but the project will not succeed without this.
    - **Automation Platform :** Selecting an automation tool in DevOps is a very important and critical task. Many of the automation tools that you will use in the delivery pipeline, available in the market are not suitable for each use case. One tool may be useful for phasing the rollout based on A/B testing result, whereas another tool with advanced release management capabilities is useful in case of continuous or frequent updates in a short span of time.

- **App Architecture :** Design your application as cloud-native app. A cloud -native application can be a microservice based application , using containers for deployment in live or test environments. Deployment pipelines consist of a number of stages executed in sequence to successfully deploy applications in production. With the advanced deployment techniques and microservice architecture, a number of customers bypass the stages of deployment. Each stage in itself can create an environment in the cloud for deploying and testing the application and later will be decommissioned after use as a part of deployment pipeline. Always look for a tool which will create and destroy the environment in the cloud for selected stages.
- **Security :** Waiting for security check final deployment is not a recommended process in the current scenario. Security is the most important and crucial aspect of application and data security during and after deployment of an application. Security should be part of CI/CD pipeline and should be an automated task. Number of companies are implementing testing, code scanning, template checking and security tests as an automated and integrated part of CI/CD pipeline.
- **Insight :** Sharing a common view across teams will help track and monitor progress and development stages. Tracking and visualizing pipelines, environment variables, issues, various test results, versions and other artifacts will help share a common view amongst developers and operations team.
- **Delivery Strategies :** Automated tests are useful to be successful in deployment and delivery pipelines. Production failures are the results of manual and incomplete tests performed in delivery and deployment phases. Test cases and test suites should be mapped to business outcomes and requirements to ensure the test team gives green signal to deploy in production. Test data and automation purpose data should match the production data and environment closely as much as possible.
- **Monitoring and Feedback :** Finally, continuous feedback on user experience, response time, performance of services, and the technical metrics from all stages of life cycle development, test and production. This will ensure no degradation will occur with frequent releases of applications. Improving early in the stages and reducing the wait time will increase the performance and delivery pipeline efficiency.

- With these strategies, you can deliver your software or application very easily with updated features and with more proficiency in terms of less number of errors, happy users and less downtime.

3.9 Benefits for Continuous Integration and Continuous Delivery

- Releasing a software can be a painful and time consuming process. One is involved in manual integration, setting environment for testing and deployment, setting test cases and deploying successful builds. A threat or simple bug may force everyone back to the initial stage.
- A timely release puts pressure on the team to deliver code at best within a specific time span and fix bugs, if any, found after deployment. Another way to deal with this problem is the CI / CD approach. With CI/CD, many organizations are able to deliver products with continuous improvements observed daily, that too without compromising software quality. '
- With CI/CD, code changes are shepherded through an automated pipeline that handles the repetitive build, test and deployment process and alerts developers about any issues.

CI/CD offers many benefits, listed as follow :

- **Faster time to Market :** The primary goal of CI/CD pipeline is to deliver new features and updates quickly and frequently to customers. With a quick and continuous feedback loop, you can come up with new innovative features and add them in the next release or update. New features can be shipped as early as possible and can address pain-points as they emerge.
- **Reduced Risk :** Frequent releases integrate new features in existing software and provide a best way to identify bugs at an early stage in the test or live environment. With early testing, you can save your time which would have been wasted in developing features which do not solve problems for your user.
- **Shorter Review Time :** Frequent code commits, at least once a day, ensures everyone is building on the same foundation. All code changes can be integrated easily and faster code review is possible.
- **Better code quality :** A central part of CI/CD pipeline is automated tests. With automated tests, time required to test your code's behaviour is easy and faster. As anyone who has to follow manual scripting for test cases knows how painful it is to write scripts every time and perform testing manually. Automated tests ensure quick feedback to developers with status of tests. If any bug is detected, the developer can fix it quickly and commit new changes. Faster the test, faster the improvements and results in better quality of code.
- **Smoother path to production :** As you know, Practice makes perfection, same is applicable to software releases as well. Every time you commit changes, automated tests detect bugs and developers can fix them. With each incremental development new features are released frequently and hence making it easy to release in production.

- **Faster bug fixes :** Even with improved code quality, bugs sneak their way to the productio environment. If you commit your changes frequently and regularly, each release to production will have less number of changes. If you roll back from changes, you will take less useful changes with it.
- **Efficient Infrastructure :** Establishing a strong CI foundation stage is followed by automation of deployment in test and production environments. Using infrastructure-as-code involves automating the creation of those environments. A number of tools are available to automate and manage infrastructure creation tasks. This will help to create an environment with new configuration whenever changes are required and configuration files can be saved in the repository.

Some other benefits of CI and CD are measurable progress, tighter feedback loops, easy collaboration and communication and maximized creativity in features.

3.10 Designing a CI and CD System

A CI/CD pipeline resembles and mimics various stages of software development lifecycle :

1. **Source** : take new changes in code or configuration and trigger a new instance of the pipeline.
2. **Build** : The pipeline compiles new code and creates a new redistributable package without any error or bug in build.
3. **Test** : Perform testing of code, configuration and data.
4. **Deploy** : A successful build is deployed to production and tests are initiated against deployment to verify performance and security.

Let's understand steps performed in each stage with more details to form CI/CD pipeline.

(1) Source Stage : In this stage, source code repository is implemented and version control system features are used for better control over source code. Across distributed teams, a collaborative work environment is established using version control systems and code repositories. This stage forms the first step of CI/CD pipeline. The pipeline instance is triggered by the version control system based on various events like push or pull request validation and new commits. The pipeline instances can be executed on triggers scheduled by user or on initialization by user manually.

Various step involved in this stage are :

- Select version control system and repository host
- Decide hosting location, local hosting or use providers such as Github, BitBucket.
- Create repositories to maintain source code along with pipelines.

(2) **Build Stage** : This stage deals with selection of CI server, creating build by compiling code and testing the build. Important goal of this stage is to provide feedback to developers and maintain the build in deployable state or in a state where it can be released to an environment. It is imperative to provide feedback to developers for each commit to resolve any issue like syntax error or compilation. Also, it ensures that code can successfully generate artifacts which can be released to an environment.

Various step involved in this stage are :

- Determine CI / Build server such as Jenkins, Jenkins X, GitHub action, CircleCI or Azure pipelines.
- Set up a pipeline-as-code, in version control system, to trigger software releases on certain events.
- Implement a stage / job to build the application source code, like build a Docker image.
- A CI server provided plugins can be used to perform static analysis and style check to verify organization-wide code style consistency.
- The pipeline then generates a versioned and built artifact which is then made available in feed or artifacts registry for testing.

(3) **Testing stage** : This stage deals with testing tasks with the help of tools and publishing results along with release to production. The main goal of this stage is to ensure safe and working builds are released without affecting any existing feature. Various tests performed in this stage are : unit test, integration and functional tests. Unit tests ensure functionality of each small module without any dependencies. For example, a class in OOP languages can be a small unit of testing. Integration tests analyze working of individual units of code together in a group to ensure successful integration of different units to perform a task in collaboration. This will ensure the application will not break after integration of different units. The end-to-end functional testing is performed in a production-like environment to test complete applications.

Various step involved in this stage are :

- Choose a plugin, like JUnit, that enables integration of test runners within the pipeline.
- Publish test reports and code coverage reports
- Maintain a predetermined benchmark of code coverage, which decides the result of code coverage based on threshold.

(4) Deploy Stage : This stage focuses on Delivering and Deploying final build. A successful build artifact can be tested in various environments before releasing to production. Also, deploy stage is used to add resources to host the application in the cloud environment. Various step involved in this stage are :

- Choose application deployment strategy before final deployment. Some of the strategies are blue-green, canary and rolling deployments. For container based deployments, the deployment should involve orchestration platforms like Kubernetes.
- For deployment in the Cloud, use the cloud provider's infrastructure-as-code option, such as terraform or Azure resource manager template. The selected IaC should support idempotent deployment.
- You can look for configuration management tools like Ansible, Puppet to automate configuration inside the operating system.
- Select tools which can perform both cloud resource deployments and application release strategies. This will fres team from managing scripts to deploy infrastructure and applications.
- CI/CD pipeline principles and best practices :
- Creating CI/CD pipeline should follow some best practices to make it more useful :
  - Fix bugs immediately if build is broken
  - With a large number of tests, First run fundamental and faster tests like code quality tests.
  - Use a consistent environment to deploy, an application in a handcrafted environment may work but in a production environment mostly it tends to fail.
  - Bake in code quality check - integrate open source tools to provide documentation to developers.
  - Gauge CI/CD pipeline speed - check the time required for pipeline to start and time required to complete pipeline stages. If the time consumed is too much, then the pipeline will not provide fast feedback.
  - Document everything - To enhance collaboration across teams, provide a document to understand how the pipeline functions in real time. This will help developers to fix the issues, if any.

List of some of tools used in various stages

Source	Build	Test	Deliver / Deploy
Atom, Visual Studio, PyCharm Git, DockerHub, JFrogartifactory	Gradle, Maven, Meister, Vulnerability scan, Netsparker	Katalon, Kobiton, Saucelabs, Selenium, TestArchitect	Ansible Tower, Atlassian Bamboo, CircleCI, Codeship, Jenkins, Teamcity

3.11 Building Continuous Integration and Continuous Delivery Pipelines

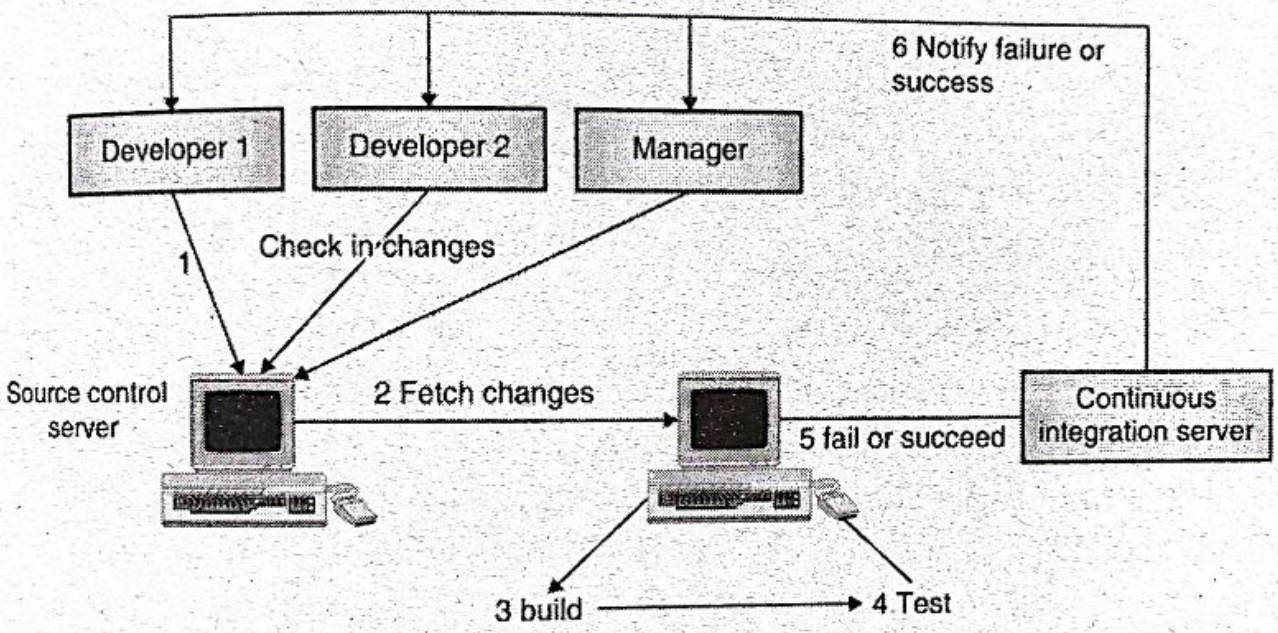


Fig. 3.11.1 : Building CICD pipeline

- Continuous integration involves steps to integrate code from various developers to build an application and then perform basic tests for validation of error free code integration. During the waterfall model, developers were spending months or years in development of various services of applications and later in the integration phase they started integrating code. This was a painful and time consuming process.
- With agile methodology and extreme programming, integration became a frequent and regular task where developers integrate their code as soon as a unit is completed. Before merging or integration, some basic tests are performed continuously and with automated scripts.
- Thus Continuous integration. Common flow of CI follow some basic steps :
  1. Pushing to the code repository
  2. Static analysis

3. Testing before deployment
4. Packing (artifact generation, docker image creation) and deployment to the test environment
5. Testing after deployment

A Continuous delivery pipeline consists of five basic phases to be performed after CI pipeline. build/develop, commit, test, stage and deploy. Let's consider each step and understand actions performed in detail :

1. **Build / Develop** : Depending on language and IDE used, built-in plugins or external tools can be used to build the application. Additionally we can use scripts and virtual machines (VM) or a container. Steps performed in build are :
   - Pull source code from private or public repository
   - Link various libraries, module and dependencies
   - Compile code to build binary artifacts
2. **Commit** : The commit phase checks code changes and sends new changes to the code repository. Every time a code commit initiates a new instance of deployment pipeline. Once the first phase is successful, a release candidate is created. This will ensure bug-free applications are deployed to production and alert developers in case of failure. Steps performed in commit phase are :
   - Compile source code for each commit
   - Run commit tests
   - Create binaries for later use
   - Perform health check / analysis of code
   - Prepare artifacts for later use
   - These jobs run on build grid, a feature of continuous integration server. It will ensure execution of the commit stage in less than ten minutes.
3. **Test** : A completed build undergoes comprehensive dynamic testing after it passes static testing. Dynamic testing include :
   - Unit or functional testing - verify intended working of new feature
   - Regression testing - verify new additions do not break previous functions

- Some additional tests that can be performed are user acceptance test, performance test and integration test. Any error detected during testing is loop backed to developers for analysis and remediation. Automated tests in CI/CD pipeline saves developer time from waiting for test results. This will catch errors which are missed and ensure objective and reliable testing.

4. **Stage** : Staging phase is a replica and mimic of the production environment for testing deployment of builds. Before going live, you will get confidence on deployment of the application in the expected user environment. You can detect errors and bugs in the staging stage and fix them to reduce risks of errors end users may face, data quality problems and integration problems. Staging environment can be used as part of the release cycle and later can be removed after deployment in production.

5. **Deploy** : A deployment phase is the next phase to staging. A candidate released for deployment is deployed either manually (in continuous delivery) or automatically (in continuous deployment). Developers use scripts or automation tools workflow for the deployment.

Deployment involves following tasks, not mandatory :

- Create deployment environment
- Connect with error reporting and ticketing tools - identify unexpected errors, alert developers and allow users to submit bug tickets
- Common deployment strategies used are : Beta tests, blue/green tests, A/B tests and other crossover periods

3.12 Continuous Database Integration

- Rapid integration of database schema and logic changes into application development efforts is known as continuous database integration. Developers will be notified immediately using feedback loop for any issues identified during integration and build.
- A step ahead, database continuous delivery seeks to produce releasable database code in incremental and short cycles. The main goal of the database continuous integration tool is to bring all benefits and CI and CD best practices to the database and enable SQL code to flow through the software release pipeline. This is in synchronization with application code.
- Developers see much more returns on investment in tooling and process updates by alignment database deployment with application deployment. With this, new innovations with higher quality are brought to market in less time.

3.13 Preparing the Build for Release

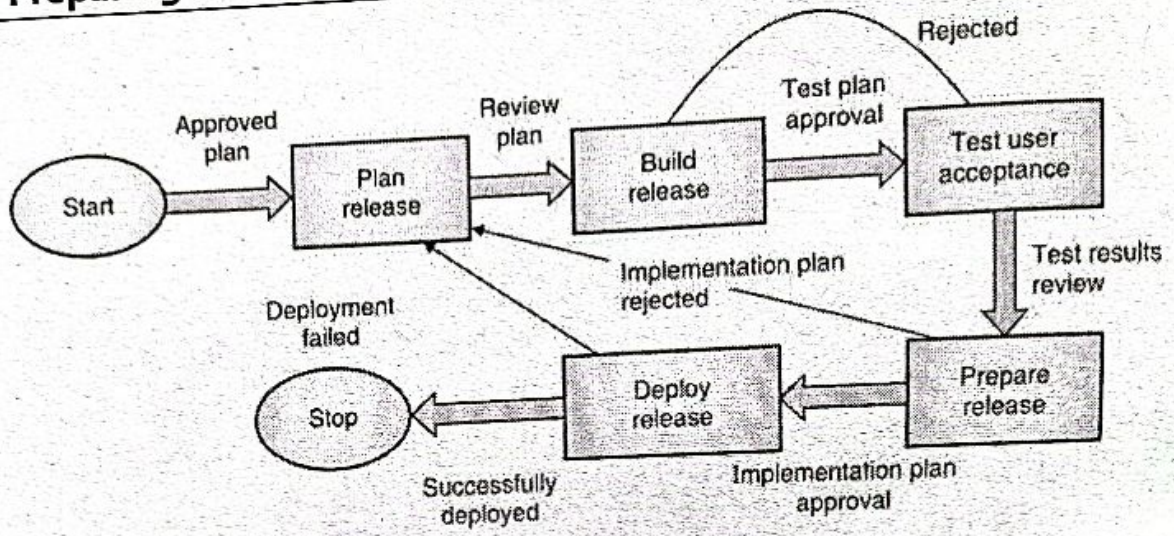


Fig. 3.13.1

- If your company is planning for a significant software change, you would appreciate the reliable release management process to be used in the release process. Release management is required for any new release or even for small changes in software.
- It will oversees all the stages involved in a software release process. The most time intensive release planning phase, with a robust release plan, will keep your team on track and ensure all standards and requirements are met. A team can approach release plan in a number of ways, out of which most popular is the systems development life cycle.
- Once planning is completed, start designing and building the product for release as per the plan. The actual development will happen at this stage. After addressing all the issues, if any, you can start testing the build in a real-world scenario.
- In a number of iterations, developers build the product and send builds for testing in a real-world environment, obviously with automated release and test. During testing, if any issue is identified, the build is sent back to stage two for development and fixing the issues. In the iterative release management process, build will flow from stage two to stage three until release is approved.

3.14 Identifying the Code in the Repository

- The primary container for all versions of source code is code repository. Along with source code, the repository also stores configuration files, build scripts, deployment scripts, and test scripts as well. Some projects leverage to store binaries, but each binary should be stored in a different manner, as repositories may have a number of binaries. So what to store in the repository ? Only human readable files should be stored in a repository.

- The version control system is the primary component of DevOps pipeline is which allows following :
  - Manage all code changes
  - Track all code changes
  - Enable collaborative working for multiple developers
- The source code repositories should be managed properly by adhering to branching strategies to avoid the problem of bloating and unwieldy repositories. A branching strategy is useful when multiple developers are working on the same project and on different services to avoid overlapping and conflicts while merging code.
- The goal of the version control system is to manage and write code to the repository. A properly created branching strategy is the key to perfect DevOps pipeline. DevOps is focused on creating an efficient, streamlined and fast workflow and that too without compromising code quality. Some of the advantages of using branching are : define delivery team functionality; define methods to handle bug fixing, each feature and any improvement in features; developers focus on development and deployment on specific relevant branches without worrying affecting overall product.
- Branches and functions of branch :

1. **Master :**
   - Primary branch
   - Stores production code
   - If Code from develop branch is ready to release, changes are merged to master branch and deployed in production.
2. **Develop :**
   - Actual development branch
   - Store Pre-production code
   - Completed code of al supporting branches merged here
   - Some sub branches are created by developers with Develop as base
     - **Feature** - these branches are used to develop new features, they branch off from develop branch

- **Hotflix** - to deal with production issues, this branch is used so quick bug fixing can be done. They can branch off from master branch itself and need to be merged in both master and develop branch
- **Release** - this branch aggregate fixes and improvements. It is used to prepare for production release and is branched from develop branch.

3.15 Creating Build Reports

- The build report is a synopsis of the build created from source code after each merge or push and pull request on repositories. The first / title page of the synopsis identifies date and time of build along with scope and mode used for build. It also lists out the member names that were considered for build and project definition specified on SCLM main menu. The report also lists the components that passed the compilation phase.
- Basically the build report consist of :
  - Date and time of build
  - Mode used for build
  - Component name requested to be build
  - Last update date and time of component
  - Project definition used for build
  - Software components translated successfully by this build
  - Build maps that require regeneration
  - The out-of-date components name that caused regeneration
  - Software components and build maps deleted from build group

3.16 Putting the build in a shared Location

- Build is an executable software artifact generated from source code. The basic difference between build and release is that, Build is a version handed over to the testing team to perform testing in a staging / test environment whereas release is a software handed over by the testing team to customers and is deployed to production.
- You can store your build artifacts in the artifact registry after the CI pipeline is completed successfully and are made available for automated deployment in test, stage and production.

- Build artifacts are files like WAR files, reports, logs and packages which can be stored in repositories on CI server or in a remote location available to CI server. Regular commits to the repository will create a number of builds through CI/CD pipeline, out of which some will be deployed to few environments before identification of issue and some will make their way to release to live environment.
- An artifact repository is a central location to those builds and provides APIs for deployment to various environments as specified in the pipeline. Through conditional logic, you can remove artifacts from the registry after some time and free up the space.

3.17 Releasing the Build

An artifact is an deployable component, which forms a release when artifacts are grouped for an application. DevOps pipelines can deploy artifacts stored in different types of artifact repositories. For example, the Azure pipeline can deploy artifacts from various artifacts repositories. Authoring a release pipeline will link the artifact resources to the release pipeline. Example, in figure Azure pipelines build pipeline is linked to release pipeline. You will specify the exact version of artifact sources while creating a release. Example, version of a build coming from the Jenkins project. A versioned release is deployed to all stages, the same artifact is deployed and validated in each stage for testing and identifying issues. A single release pipeline can be linked to one or more than one artifact sources, of which one will act as primary source. In this case, you will specify the individual version of each of the source.

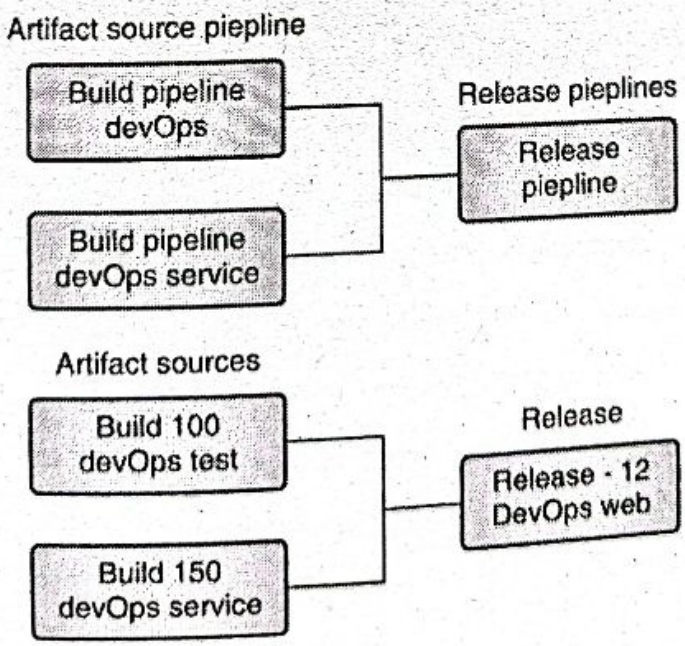


Fig. 3.17.1

- **Auto-trigger Releases** : Some artifact sources support automatic creation of new release for each new version of artifact.
- **Trigger Condition** : On satisfying a particular condition, a configuration file will create a new release or trigger the deployment of the release to stage. Example, create new release only when new artifact is build from a certain branch.
- **Artifact versions** : Configuring a release to automatically use a specific version of an artifact from a build collection, to always use latest release.

REVIEW QUESTIONS

Q. 1 What is Continuous integration? What are the benefits of CI?

Q. 2 What is Version control? what is Git?

Q. 3 How CI helps to improve the time to market aspect for businesses and development teams?

Q. 4 How do you related CI and version control system?

Q. 5 What is git branch? How long should a branch live?

Q. 6 What are the most important characteristics of CI/CD platform?

Q. 7 List out practices useful to implement CI and explain some of those.

Q. 8 What is the use of a code repository server? How to select a good repository server.

Q. 9 Which advantages are offered by CI servers?

Q. 10 Explain the working of the delivery chain.

Q. 11 Compare continuous delivery and continuous development.

Q. 12 Compare continuous delivery and continuous deployment.

Q. 13 What are the continuous strategy scenarios?

Q. 14 List and explain various best practices for CI/CD pipeline.

Q. 15 Draw a diagram to visualize CI/CD pipeline and explain all stages in detail.

Q. 16 Why do we need continuous database integration?

Q. 17 What is build stage? Write step-wise process to generate builds?

Q. 18 How will you identify code from a repository for deployment in various environments?

Q. 19 How are builds deployed from a shared location?

4 Continuous Deployment and Orchestration

UNIT - IV

Syllabus

Implementing a testing Strategy: Types of Tests, Integration testing, managing defect backlogs, what is Continuous Deployment? Changes moving through the deployment pipeline, Trade-offs in the deployment pipeline, Basic Deployment pipeline, Deployment pipeline practices & Commit stage, Automated Acceptance Test Gate, Subsequent test stages, preparing to release, Implementing a deployment pipeline.

4.1 Implementation of Testing Strategies

- Testing is a process of checking or verifying or validating capabilities or features of any object.
- Software Testing is a process in which we test software units or complete modules or software applications to verify correctness of software outputs, effectiveness of solution in customer or user problem resolution, quality of software application along with performance achieved using latest technology.
- It is one of the organizational processes followed in the development of software. With the help of testing we can ensure the developed product and its features are behaving as expected. Testing will save an organization time and money from being invested in re-development and bug resolving, after the customer or user reports a bug while using the software application.
- Development time can be used for new feature development instead of fixing bugs. A set of test cases can be executed against each new feature and existing features to ensure expected behavior.
- Continuous delivery (CD) pipeline allows developers to perform code evaluation using various testing strategies, once the code is pushed to the pipeline.

- Let's take an example, consider the first scenario, you are traveling in a war-torn region and you are examined at each kilometer to ensure safety of the region. Consider the second scenario, in the same war-torn region you are allowed to enter without any check and no one is taking measures for the safety of the region, it's just free entrance.
- In which scenario, will you find yourself secure ? Correct, its scenario one. Security and stability are ensured at each step and ultimately ensure a safe future. CD pipeline implements continuous testing, also known as Incremental testing, the process of testing software at each phase of SDLC. Software developers go from one phase to another during SDLC - from downloading dependencies, module development, versioning changes in UAT to versioning changes for production ready software. In each phase of SDLC, continuous testing will ensure that each phase is bug free and ready for release.

4.2 Types of Tests

There are several types of testing, majorly categorized as Manual Testing and Automated Testing or functional and non-functional testing. Each type performs testing of software functionality from a different view point.

1. **Manual Testing :** A group of testers or any individual manually operates a software and verifies software behavior is as expected or not. This process is carried out manually and may consume more time to generate testing results.
2. **Automated Testing :** Automated software testing is a process carried out with the help of different testing tools which exhibit varying capabilities. These capabilities range from isolated and automated code correctness checks to human-driven manual software testing.

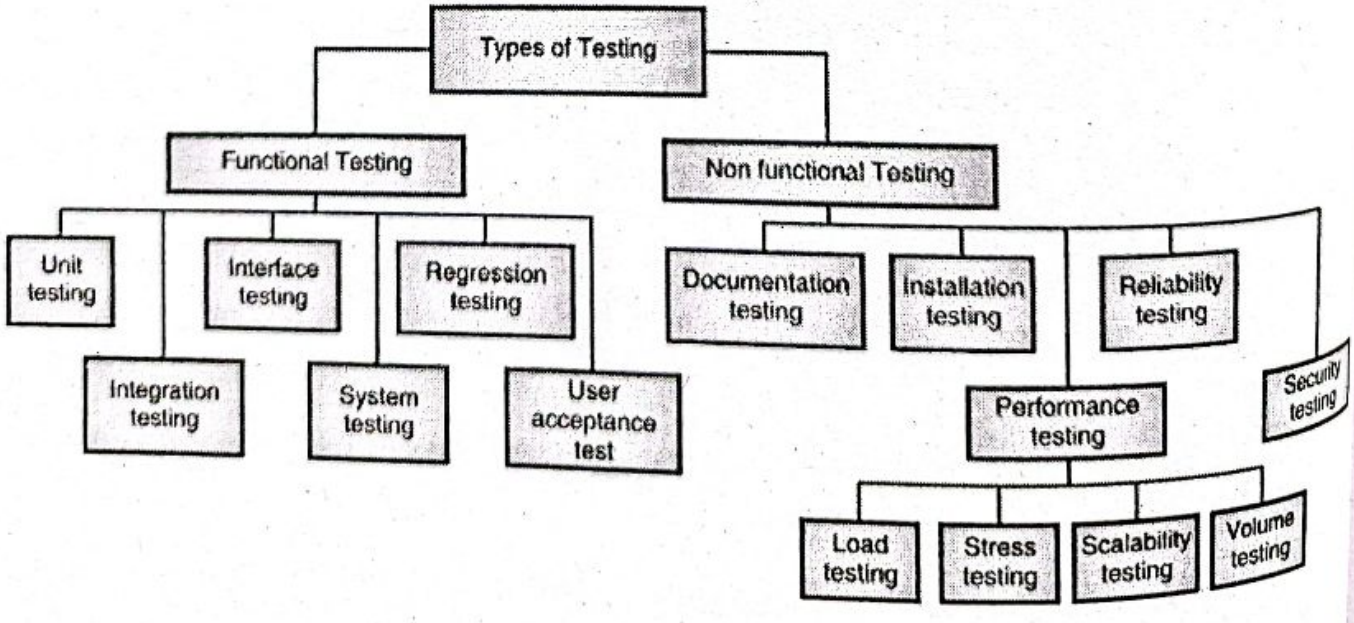


Fig. 4.2.1

A. Functional Testing :

Functional testing is a process of Software testing performed to ensure functional requirements and specifications are satisfied by the software. Functional testing verifies operations and actions of software. It is easy to perform functional testing. Functional testing helps to enhance the behavior of the software.

Functional testing comprises of following types of testing :

i) **Unit Testing** : An individual unit of code is tested with specific input and expected output. This ensures each unit is behaving as expected irrespective of other components / units of software. Example, consider the function addXY() on input x and y, always returns x+y. The unit test would execute function addXY() with two values and compare returned value with expected value.

ii) **Integration Testing** : Any software test case covering more than one unit is considered as Integration testing. Example : any unit dependent on third party packages or code can be tested using mocking third party code.

iii) **Smoke Testing** : Smoke testing is also called build acceptance testing or build verification testing. Smoke test ensures behavior of most crucial functions of a program. Smoke testing is performed after a build and before a release.

iv) **Regression testing** : Regression software testing is a testing performed to verify software behaves as expected even after some code changes, updates or integrations. Regression testing is used to ensure overall stability and functionality of software. Under continuous improvements, we can update features or functions of software and after each update perform regression testing.

B. Non-Functional Testing :

A software testing performed to verify non-functional requirements of application is known as non-functional testing. It verifies the behavior of an application, unlike functional testing which verifies operations and actions of software. Based on the performance requirements captured during requirement specification, non-functional testing can be performed.

Non-Functional testing comprises of following types of testing :

- **Performance Testing** : Performance testing is useful to determine scalability, stability, performance in terms of speed and responsiveness of software under a given workload. With varying workload, you can observe performance of software and enhance the software performance by taking certain actions. Important goal of performance testing is to analyze software performance parameters like number of concurrent users handled, response time, processing time, memory utilization, data transfer rate, network usage statistics and so on.

- **Load Testing** : It is a subset of performance test. Under varying workloads, we will te the software performance to analyze behavior of software for a number of concurren users. This test is mainly used to check speed of processing for number of requests an capacity to handle concurrent users. Example : a software system with a login page ca be tested to check time consumed for loading a login page with 10 users and with 500 user requests. If the time required to load the login page is consistent with any number o requests then the system is considered as passed in load testing. Here, we can say tha the system is handling a load of 5000 users with consistent response time.
- **Stress Testing** : Anything stressed beyond a certain limit results in consequences. An web application, if stressed heavily, may get permanently damaged. To avoid any damag to the application, we need to find breakpoints and solutions to avoid damage conditions Example, any eCommerce website may stop responding during a shopping festival o some festival sale. Stress testing is performed for ecommerce websites or web app financial applications, social networking applications and emailing applications.
- **Scalability testing** : Scalability testing is a software testing performed to chec applications capability to scale up and down depending on number of users and use requests. A robust system should be able to manage its resources based on sudden fall o spike of user requests. This test will check the capability of software application in suc scenarios. A robust system will adapt to such changes and deploy all of its resources t optimize the cost and result in great user experience. Using the testing parameters like CPU utilization, network usage you can account for various vulnerabilities and weaknesses of software applications.

4.3 Integration Testing

- Any code, though it seems to be perfect, is unpredictable. Dynamic projects can be considered as the best case for unpredictable behavior, as pre-approved requirements of such projects change over time and may lose their relevance in application. One of the reasons behind changing requirements is new feature requirements from clients.
- In addition to this, you can think of poorly designed requirements, unforeseen problems like design defect and test case failures, marketing aspect, missing stakeholders including developer or requirement provider, very optimistic deadlines or budget, product development life cycles not addressed completely and many more as reasons behind changing requirements in product lifecycle.

- With each change in feature, you may observe changes in product behavior. To ensure expected behavior and to verify expected behavior, integration testing is performed after each change or update in software application.

Advantages of Integration Testing :

- Simultaneous testing of number of units is possible in Integration Testing
- Integration Testing is a convenient, easy and practically feasible approach to test more than one unit
- Integration Testing can be performed at any stage of SDLC
- It is a lifesaving step for developers involved in projects with constantly changing requirements or where code is under review after the initial development phase.
- In one sprint, testers can cover any amount of program code.

Algorithmic Steps to perform Integration Testing :

- Write complete test plan
- Create all test cases along with use cases
- Run integration testing once units are integrated
- In Integration testing, detect errors and note down or report errors / bugs
- Retest functionality to verify if bug fixed correctly
- Re-run Integration testing repetitively until all bugs are fixed.

Effective implementation of Integration testing is important to perform nex level of testing. A tester should be aware of all units and modules, project structure and interaction amongst units. With detailed understanding of all such conditions, testers can develop a complete set of test cases and execute integration testing to fix all bugs before releasing the software for the next level of testing.

Example :

Consider a software application with following features :

- Login / signup
- Standard plans
- Customized plans

- Customize your plan
- Suggestions
- Plan progress

A user when login to an application with his/her own credentials, a user details page may b displayed to the user. If the user is not redirected to a second page, its error. Such redirecti errors can be fixed by developer and tester then rerun test to verify bug is fixed.

4.4 Managing Defect Backlogs

- Defects , sometimes referred to as faults, are an inevitable part of any software applicati delivery. Defects is an area where software delivery gets messy, even in agile softw methodology. By looking at defect backlogs, some may feel good, some may start think over the list. A defect backlog is considered as a prioritized list of all defects. Priority defects depends on business value.
- You might have a question in your mind, how to manage growing defects list ? Following a some common strategies which help to manage growing defect list :
  - **Do nothing :** Depending on who is concerned about defects, you can decide which actio to take. If no one is concerned, then no need to take any action.
  - **Filter out the noise :** Depending on who cares about defects, like developer, owne manager, tester or someone else you can filter out defects with low priority or which ar noisy. Example, a developer is not working on some module, but still the developer getting a list of defects from such modules. These defects are noisy and can be ignored o removed from the list.
  - **Stop logging low impact defects :** A simple way to reduce the number of defects fro the list is to stop logging low impact bugs. Ignoring low impact bugs like this is not th efficient solution. People may notice the same bug again and keep raising those bug again.
  - **Close off low impact defects :** It is similar to not logging low impact defects. With thi you will never fix defects and just close the raise ticket. Problem with this action is th bugs may get raised again by people.

- **Prune the backlog :** Occasionally prune the backlog list to check updates in list. With system updates, some items may become irrelevant and such items can be ignored or removed or closed.
- **Batch the defects :** With this you can batch related items together and then address such items in batch. These individual items may have low priority but a batch may result in high priority and product owners can prioritize a batch over individual items.
- **Blitz the defects :** Often after a major launch, a team can be allotted to fix defects for a sprint.
- **Fix some every sprint :** With each sprint, allocate team time to fix defects. Allocated portion of team time is useful for rolling out fixed bugs in the same sprint.
- **Automate the tests :** Automated tests address all defects within the same sprint and reduce the number of defects logged in the list.

4.5 Continuous Deployment

What is Continuous deployment?

- Continuous deployment is a DevOps strategy, without human intervention, for software releases for automated release of code to production environments. Any code which passes the automated test can be released to the production environment. This automated release makes all changes visible and accessible to software users.
- Continuous Delivery helps to eliminate the human safeguards against any unproven and untested code in production environments. Continuous deployment is one step ahead of continuous delivery. Only in case of a failed test case(s), new code changes are prevented from deployment to production. Every code change that passes production pipeline stages is released to production and made available to users. A staging area is not required in case of continuous deployment, as no lag in new code changes and moving new changes to live platforms is observed. Continuous deployment is implemented with a continuous feedback system seeking feedback from customers without any delay in the feedback process.
- Developers can concentrate on development of modules and software , instead of waiting for software testing and then getting feedback from the test team. Automated tests help developers to provide immediate feedback on new changes and in turn help reduce bugs in released versions.

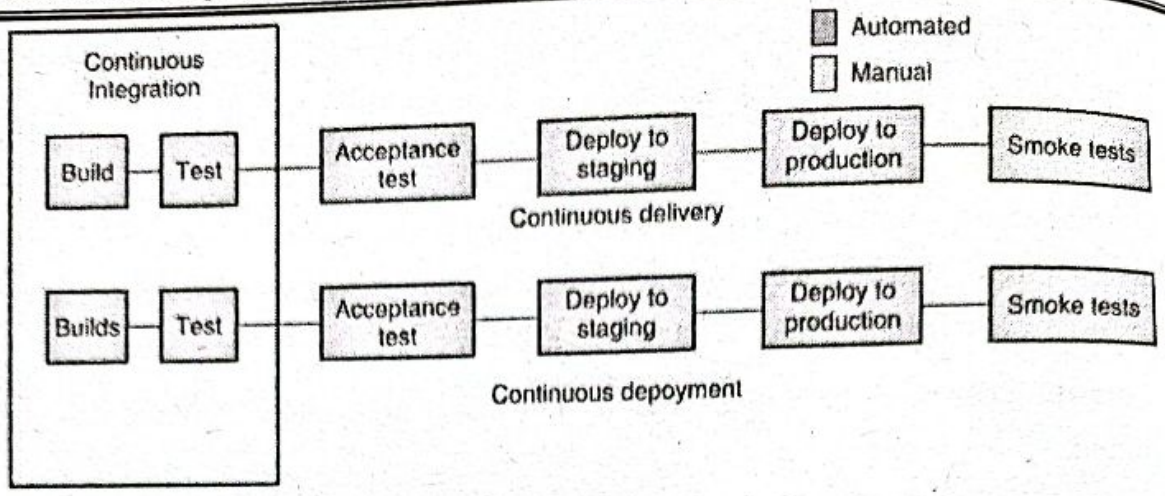


Fig. 4.5.1

4.5.1 Changes moving through the Deployment Pipeline

- Continuous Delivery is a process going through various phases starting from obtaining cod from the version control system to releasing it in live or production environment. Develope working on development of any feature of a software need a reliable and efficient system t build, test and deploy the feature code. In the early times of the software development lifecycl this was done manually with a requirement of lots of communication within various teams an team members. Manual processes and more dependency may result in a lot of errors.
- Deployment pipeline can be divided in three steps as : Build, Test and Deploy. These steps in the pipeline help automated test and deployment of new code changes pushed on code version control systems. Deployment pipeline will ensure that code pushed to the repository will move quickly to the production environment through the deployment phase.
- Each step in deployment pipeline can be explained as follow :
  - **Build** : A developer pushes all new code changes to the code repository. New feature is now integrated with the environment similar to the production environment.
  - **Test** : A new code pushed on a repository can be identified by deployment tools like Jenkins or Ansible, and a series of tests is triggered for new code changes. If new changes pass all automated tests, a build is released to the production environment.
  - **Deploy** : A successfully released build from the previous step is deployed to the production environment and made available to all intended users.
- Along with development, continuous testing should be implemented to provide continuous feedback to developers. Continuous integration is one of the important strategies used in the development and deploy phase to help developers for testing and merging new code changes in repositories.

4.5.2 Trade-offs in Deployment Pipeline

- Deployment pipeline is an automated phase of software deployment starting from developers commits and moving towards automated deployments of new releases. The time to deploy refers to creation of deployment pipeline and deploying new features as and when ready or fixing the bugs and redeploying the system with existing features.
- Basically any module can be deployed as a monolithic or microservice.
  - **Monolithic** : In this approach, the whole application is deployed using a single delivery / deployment pipeline. With a single pipeline and the whole application is deployed each time, a high risk of application downtime is possible. If anything goes wrong with new changes and release fails, it may result in a disastrous situation. There is a high risk of violating the zero downtime principle which states the application should be available for the users everytime.
  - **Microservice** : A software system can be divided in services and each service can be deployed individually. Each microservice needs its own deployment pipeline so that each microservice is delivered separately. Whenever we add a new feature to any service, only that particular service is deployed again. If service goes down, only that service will remain unavailable but the system will be available to users. This will reduce risk of taking applications down, hence we can have more number of releases with less risk.

4.5.3 Basic Deployment Pipeline

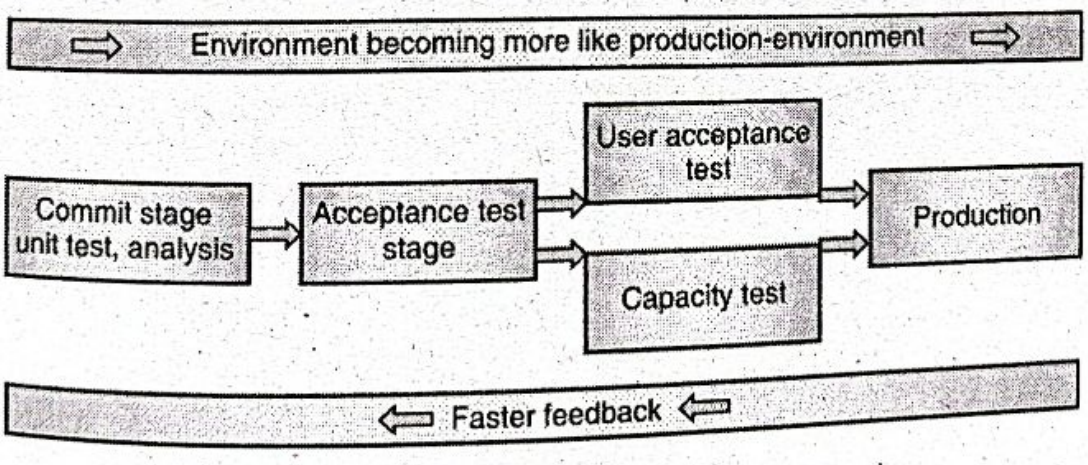


Fig. 4.5.2

- Continuous Testing and operations are the robust levels in the deployment pipeline. With each commit, a new version of software is tested and deployed into production without requiring any human intervention. This type of automated behavior is rarely observed and is unnecessary. The organization with thousands of employees working in the development team are following this strategy.

- Developers are developing a number of features every day and the number of commits from all developers results in a number of releases everyday. In such cases, on request or as a matter of requirement, automated deployments to staging or production can be configured.
- Above diagram depicts the general structure of the deployment pipeline and provides an overview of the importance of deployment pipeline behavior. The deployment pipeline process starts with developers committing code changes to version control system, also referred as code repository. The Continuous integration (CI) management system triggers a new instance of deployment pipeline for each commit by the developer. In the first stage, commit stage, of pipeline following set of tasks is executed :
  - Code compilation
  - Unit testing
  - Code analysis, and
  - Create binaries or installers
- The second stage is a time consuming automated acceptance test. A CI server splits these tests in suites and lets you execute them in parallel. This enhances test performance by providing test results as feedback within minimum time, usually one or two hours. On successful completion of the commit stage, the acceptance test stage is triggered automatically.
- If the second stage is passed, the pipeline continues with the third stage. In this stage, pipeline enables deployment of your new build to various environments such as UAT, Staging, Testing and Production.

4.5.4 Deployment Pipeline Practices

Deployment pipeline is a series of actions or tasks performed automatically starting from commits till build deployments. To take full advantage of the Deployment pipeline and to understand its benefits, one should follow some best practices as explained here in this section.

Let's go through some best practices before jumping into pipeline stages.

- **Only Build Your Binaries Once :** Binaries refer to collection of executable files. During various phases of pipelines, code may be compiled repetitively which may result in differences in compiled output. As compiler versions may change, environment packages may differ and third-party packages may differ you may find differences after each compilation. To reduce risk of differences in binaries, you can build a binary once and the same binary should be used for all phases like build, test and deployment in various environments. This will help deployment easy as configuration files will differ for each environment but code will remain same for all environments.

- **Deploy the Same Way to Every Environment :** It is good practice to deploy services or applications in each environment in the same way just by changing some environment parameters. Different environments may differ from others in some way, but basic infrastructure might not change. You can maintain a properties file for the configuration in the version control system and fetch while deploying the service. If you maintain separate configurations for each environment and follow different deployment processes, it's difficult to detect which will work in production. With same deployment process, in case of failed deployment you can narrow down failure reason to one of the following :
  - Error in settings configured in environment-specific configuration file
  - Error in infrastructure or some other service on which new service is dependent
  - Current environment configuration
- **Smoke-Test Your Deployments :** Every time you deploy your application, perform a smoke test. Smoke test will ensure the application is running and the home screen appears as designed. Also ensure dependent software is running and are available for application. A single script is written which will check if the application is running or not and if not then provide a reason for failure.
- **Deploy into a Copy of Production :** Many times, deployment in different environments becomes problematic. Production environment is significantly different from other environments like testing and development. To get confidence on deployment, always create an environment similar to production and deploy the application in that environment.
- **Each Change Should Propagate through the Pipeline Instantly :** Manual processing of deployment phases is a time consuming task and it may delay your feature introduction. Continuous deployment will trigger various phases based on some activity. Developer commits the code and the test is triggered. Once the stage one is completed, it will trigger an acceptance test. On successful completion of the previous stage, the next stage is triggered. If multiple developers commit at different times then for each commit a new instance is initiated. So each phase should communicate its result as quickly as possible so that the next phase is triggered else it may result in conflict while working with multiple developers and multiple commits.
- **Identify your needs :** To automate your current manual processes, evaluate your needs and what your organization is looking to achieve. Those goals can be one the following : release changes frequently, optimize downtime cost and monitor live software. Next goal is to remove bottlenecks and eliminate costs added due to bottlenecks.

- **Keep track of versions and issues :** In DevOps version control is an important and most useful tool. With each version of the system, maintain a list of issues. Creates a single source of truth by pushing a configuration file in the version control system.
- **Automate Ops side and Use Infrastructure as code :** Infrastructure as code is a way to manage and provision infrastructure for system deployment using scripts. Manual and standard operating procedures are time consuming and may lead to errors.
- **Automate testing :** Manual testing is an error prone process and may delay the build process. As build is delayed, it will take more time for release to reach production. Five types of testing and automation are : integration testing, functional testing, acceptance testing staging testing and quality testing. Automated build and testing will ensure builds are correct, error free and are deployed only if build is successful.
- **Keep security in mind all the time :** Security is sometimes considered as a hindrance to production and hence is overlooked. Security should be a priority to protect confidential data and maintain compliance. Security if implemented at all four checkpoint (development, build production build and push to production) of production, resultant software will be of a high quality.

Automation plays an important role in today's SDLC. All these best practices will help you to use automation to the fullest extent. Automated deployment pipeline will increase deployment speed but reduce errors.

4.5.5 Commit Stage

- Each commit of new code creates a new instance of deployment pipeline. If the first stage is successful, you will see a release candidate as a result of successful build for a new commit. In the development pipeline, the first stage is targeted to eliminate builds which are not fit for production and signal the development team that the application is broken.
- The feedback should be provided as quickly as possible to fix the bugs, save the developer time and reduce chances of applications being unavailable for users. We should minimize time spent on broken applications. As soon as a developer commits changes to a versioning system, we should evaluate a new version of a system. Developers will wait for the feedback and evaluation results before moving to a next lined up task.
- In commit stage, perform following task :
  - Compile the code if required
  - Run a set of commit tests
  - Create binaries, these binaries can be reused later in pipeline

- Perform health checks and code analysis
- Prepare artifacts, which are usable later in the pipeline

- Implementation of only the commit stage in the development phase will improve development teams output in terms of reliability and quality. However, a few more stages need to be implemented as a minimum requirement in the deployment pipeline.

4.5.6 Automated Acceptance Test Gate

- The automated acceptance test gate is the second significant milestone in the release candidate of a software application. The primary goal of the accepta..ice test is to assert that the system delivers value outcomes the customer is expecting.
- It also asserts that the acceptance criteria set by the customer is met by the release candidate. This acceptance test serves as the regression test suite ensuring no bugs in existing or previously deployed features of the system. As new changes rolled out to the live environment, new bugs in existing modules or features attend to negative feedback from customers and result in system failure.
- A commit test suite is useful for detection of many errors in early stages, but not sufficient for all errors. A unit test consists of a vast set of commit tests to ensure each new unit is working as expected independent of other modules or units. Developers are trapped in the behavior of a solution resulting from a particular way to resolve the issue. Sometimes it's difficult for developers to predict the ultimate results of a solution to solve a problem.
- Every commit-test executed against the latest commit provides us with the latest feedback on recent builds and on bugs in the application, if any due to new commits. But without acceptance tests, we are not sure about the real-time existence of applications. Acceptance tests help us to decide whether the deployment will be successful or not, and If deployed will it survive in the real world. To get feedback on all such aspects in real time and continuous, we should extend the continuous integration process to test and continue practicing these aspects too.

4.5.7 Subsequent Test stages

- A release candidate is ready to move from development team domain or environment to a wider area of interest and use in a live environment. The acceptance test is a significant milestone in the lifecycle of a module which is ready for release.

- As per definition, any build which passes the acceptance test is ready for release and can be released in a live environment for users. But if the build fails the test, it's not fit for the release and should be modified as per feedback or report received from acceptance test.
- A release candidate has progressed till this point by passing various deployment pipeline stages and tests. A new code pushed to the repository is considered as a trigger point for deployment pipeline and from there it progresses in incremental way till deployment phase. A project environment configured with an incremental delivery approach has automated deployment to production.
- During the system testing, for many of the systems manual testing is desirable, even though you have a set of automated tests configured for deployment pipeline. Many projects are tested in environments like staging environment, integration testing environment, capacity testing and exploratory testing environments which are somewhat production-like environments. Each of these environments have their own unique configurations and are intended for specific tests.

4.5.8 Preparing to Release

- Each successful tested build results in a new release of the software system. Some sort of risk is associated with each release of a software application. A serious risk identified at the time of release may delay introduction of new capability and feature in live application. Each release should be backed up with a roll out plan to deal with any risk, else it will leave the business without mission-critical resources. As a part of the new release, these resources had to be decommissioned and with no back-out plan, no resources were available as a result of new risks identified in the new release.
- The problem mitigation is simple with a complete understanding of the deployment pipeline. If we view the release step as the outcome of the deployment pipeline , it becomes easy to mitigate any problem that arises after the implementation of the deployment pipeline.

As a fundamental step, we want

- A release plan created and maintained by everyone involved in the development and deployment phase, including developer, tester, operations, infrastructure and support staff.
- Automate error prone phases as much as possible to minimize the effect of people making more mistakes
- Starting with the most error-prone stage, practice the procedure often in an environment similar to the production environment. This will help to detect any possible errors and identification of the technology supporting it.

- Keep a plan to back-out of a release if the release is not performing as expected and things don't go as per plan in the production environment.
- Have a plan to migrate configuration and production data. This is required for the upgrade and rollback process.

4.5.9 Implementing a Deployment Pipeline

Any project development, started from scratch or new feature development for existing system, wherein automated deployment pipeline is to be created for, you should always consider incremental approach. Implementation of the incremental approach for deployment pipeline comprises of following steps :

- Model your value stream and create a walking skeleton
- Automate Build and deployment process
- Automated Unit testing, Integration testing and code analysis (On successful build)
- Automated User Acceptance tests (As unit test is successful)
- Automate release phase for production environment (For accepted version and code changes)

REVIEW QUESTIONS

Q. 1 What is testing? How to integrate testing in continuous delivery stages?

Q. 2 How does testing fit in CI/CD?

Q. 3 What is test coverage? Does test coverage need to be 100%? How can you optimize tests in in CI?

Q. 4 List all types of tests and explain in detail.

Q. 5 Compare performance testing and non-performance testing.

Q. 6 Explain integration testing with the help of example.

Q. 7 What is the difference between end-to-end testing and acceptance testing?

Q. 8 Which strategies are useful to manage defects list? List and explain.

Q. 9 What is continuous delivery? What is continuous deployment?

Q. 10 What is continuous testing? What are the benefits of continuous testing?

Q. 11 What are the benefits of continuous delivery and continuous deployment?

5 Continuous Monitoring and Site Reliability

UNIT - V

Syllabus

What is a monitoring system? Factors involved in monitoring systems, why monitoring is important, white-box and black-box monitoring, building a monitoring system, monitoring infrastructure and applications, collecting data, logging, creating dashboard, behavior driven monitoring, what is site reliability engineering? SRE and DevOps, roles, and responsibilities of SRE, common tools used by SREs.

5.1 What is a monitoring system?

- Monitoring is the process of gathering and analyzing data related to a critical system, service, or application's performance.. Monitoring helps ensure systems and services are running as intended and helps teams keep a pulse on the performance and availability of any internal or external application, system, or service. If a disruption or outage occurs, teams are immediately aware via the monitoring system and can take action towards a resolution immediately.
- A monitoring system is software that helps system administrators monitor their infrastructure. These tools monitor system devices, traffic, and applications, and sound the alarm in the event of malfunctions and disruptions. There are lots of monitoring systems on the market, from freeware to professional software.
- Monitoring systems are responsible for controlling the technology used by a company (hardware, networks and communications, operating systems or applications, among others) in order to analyse their operation and performance.

- System monitoring involves the continuous monitoring of an infrastructure – aka an IT system – by an IT manager. It includes the monitoring of CPU, server memory, routers, switches, bandwidth, and applications, as well as the performance and availability of important network devices.
- Monitoring services can not only prevent incidents, but allows you to detect them faster whenever they happen. If you detect an incident fast or even prevents it, you can save time and money from the company. A good monitoring system will warn you of problems as they arise, allowing you to resolve the issue immediately and minimize the time your page is down or running slowly and, therefore, the money you will lose.
- By avoiding falls in the service and minimizing the time of resolution, the overall customer service will be improved. As the monitoring system will take care of your infrastructure, you will be able to spend your resources in a more efficient way. Your qualified staff will be able to devote more time to other tasks, knowing that, if a problem arises, you will receive the corresponding alerts. This will also cause an increase in your productivity, which is probably a very important reason why you do need a monitoring system.
- Monitoring in simple terms can simply be defined as " Collecting data about the system and analysing it based on needs for alerting / performance and other needs".

So sub categories of monitoring can be :

1. **Alerting** - This category mainly focuses on defining certain criteria where you want to be notified if certain things go wrong and is usually the main motive for monitoring.
2. **Debugging** - This category helps in finding the issues and errors with the system, as when alerted about faulty state of a system you might want to identify the issue with it, so this category helps with identifying the root causes for issues.
3. **Pattern Analysis** - This category is helpful if you want to analyse certain patterns with system like trends as with above two you would only look into in case of trouble but you might also want to know the trends that your system follows.

5.2 Factors Involved in Monitoring Systems

- In a time-to-market oriented business, DevOps with its open and Agile attitude help organizations to deliver business value quicker, adapt to change easier and overcome IT complexity with collaboration and automation.
- With Continuous Integration and Deployment (CI/CD), developers ensure frequent delivery of the newer code to the production environment. However, to be successful with the CI/CD cycle, it is imperative to continuously monitor the test and production environment.

DevOps monitoring is the key to continuously analyze an application for errors so as to avoid risks, maintain compliance, and fix issues as soon as possible. DevOps monitoring can be done at six different levels.

DevOps monitoring and its 6 components ensure on-time, error-free delivery of a project. Depending upon your application, its complexity, and scale, one or all monitoring components can be adopted.

1. Application Performance Monitoring

- Application Performance Monitoring (APM): This is the process of monitoring the backend architecture of an application to resolve performance issues and bottlenecks on time. The APM methodology works in three phases:
- Identifying the Problem: This phase involves proactively monitoring an application for issues before a problem actually occurs. For this, a number of tools are used to discover problems at the infrastructure and application level, which includes user experience monitoring, synthetic monitoring wherein the user interactions are synthesized to unveil the problems.
- Isolating the Problem: Once the problems are identified, they must be isolated from the environment to ensure that they do not impact the entire environment.
- Solving Problem by Diagnosing the Cause: Once the problem is detected and isolated, it is diagnosed at code-level to understand the cause of the problem and fix it.

2. Network Performance Monitoring

- It's the practice of consistently checking a network for deficiencies or failure to ensure continued network performance. This may include monitoring network components such as servers, routers, firewalls, etc.
- If any of these components slows down or fails, network administrators are notified about the same, ensuring that any network outrage is avoided.

3. Infrastructure Monitoring

Infrastructure monitoring verifies availability of IT infra components in a data center or on cloud infrastructure (IaaS). This involves monitoring the resources, their availability, checking under-utilized and over-utilized resources to optimize IT infra and operational cost associated with it.

4. Database Performance Monitoring

By monitoring the database, it is possible to track performance, security, backup, file growth of the DB. The main goal of database monitoring is to examine how a DB server- both hardware and software are performing. This can include taking regular snapshots of performance indicators that help in determining the exact time at which a problem occurred. When DBAs can examine the time when a problem occurred, it's possible for them to figure out the possible reason for it as well.

5. API Monitoring

API monitoring is the practice of examining applications' APIs, usually in a production environment. It gives visibility of performance, availability, and functional correctness of APIs, which may include factors like the number of hits on an API, where an API is called from, how an API responds to the user request (time spent on executing a transaction), keep a track of poorly performing APIs, etc.

6. QA Monitoring

Quality Assurance includes activities that ensure that all processes, procedures, and standards of application development are in compliance. By writing test cases, QA monitoring can be automated, enabling the development team to ensure that they are on the right track and working as per the requirement.

5.3 Why Monitoring is Important ?

- With DevOps speeding up the application lifecycle, developers have to adapt themselves by creating more comprehensive automated tests for their codes, making QA as automated as possible. QA is dependent on continuous integration (CI) – the practice of automating the integration of code changes from multiple contributors into a single software project – and monitoring systems are becoming more dependent on every part of the DevOps tool chain.
- With continuous deployment, all the automation in the DevOps tool chain moves code into production as soon as it passes all its tests. But organization can't just trust a black box that automatically deploys code, hoping that it works. That is where monitoring systems come into play!
- Properly implemented monitoring systems provide relevant insight, helping businesses to have a clear view of every piece of the application stack, thanks to API-driven code written by developers. Moreover, many monitoring systems benefit from code hooks into the application logic itself.

- Monitoring services have also widened their focus from production environments to the entire application stack, including the compiling stage, the state of unit tests, integration tests, and how well the code performs under load. For instance, Google's deployment monitoring services watch its project management software and flag individual files that have more bug reports than others, targeting the files to look out for in the future.
- Monitoring in DevOps is proactive, meaning it finds ways to enhance the quality of applications before bugs appear. The monitoring also helps improve the DevOps tool chain by showing the areas which might need more automation.
- Having monitoring systems embedded into the DevOps lifecycle allows organizations to better track business key performance indicators and monitor business metrics in production, as well as automate the transmission of embedded monitoring results between monitoring and deployment tools to improve application deployments. Monitoring systems can also use identified business requirements to develop a pipeline for delivering new functionality and continuous learning and feedback across stakeholders and product managers.
- Therefore, this continuous feedback loop doesn't only decrease time spent manually checking for bugs, but also speeds up communication between database development and operational teams. Most importantly, this takes place in non-production environments, which means fewer bad customer experiences when accessing production data.
- Monitoring is thus prompting DevOps teams to introduce third-party tools, offering more advanced features like the ability to integrate with the most popular deployment, alerting, and ticketing tools.
- With complex applications, which are updated and deployed multiple times a day, sophisticated monitoring becomes vital. Hence, monitoring needs to evolve and take into account all new data.

5.4 White-box and Black-box monitoring

5.4.1 White Box Monitoring

This type of monitoring mainly refers to the monitoring the internal states of the applications running on your system. White box monitoring is the monitoring of applications running on a server. Monitoring based on metrics exposed by the internals of the system, including logs, interfaces like the Java Virtual Machine Profiling Interface, or an HTTP handler that emits internal statistics.

White box monitoring involves using tools like Prometheus which enables you to expor metrics like total number of http requests received by the application , errors logged etc.

With white box testing, you will be monitoring your application's essential services, like the requests and all traffic in general. Analyzing these numbers can give you some valuable information.

White-box monitoring can also give some valuable statistics on how long a request took to complete. It becomes essential when you are trying to identify traffic and understand the resources at your hand.

Other types of white box monitoring include:

- Monitoring MySQL queries running on a database server.
- Looking at the number of users utilizing a web application throughout the day, and alerting if this goes above a predefined threshold.
- Considering the above example of HTTP requests—splitting these out into monitoring the different kinds to ascertain how the application is performing, or whether users are getting served the correct content. For example, a 403 would demonstrate a user has tried to get to a part of the website they're not allowed to visit. Likewise, a 200 would indicate their request was successful and they were served the content.
- Performing advanced detection of behavior we don't expect to see, such as a user not going through the normal steps you'd expect when signing into your application or resetting a password.

a) List of White Box Monitoring tools

- Prometheus
- Pandora FMS
- Zabbix
- NewRelic
- AppDynamics

b) Applications of White Box Monitoring

- The potential applications of white-box monitoring depend on the kind of application you are running. For instance, monitoring user logs and request logs is a case of white-box monitoring.

- If you are running a web application, many parameters, including the number of active users, number of requests to access user profiles, requests to post a comment, can monitor everything carefully using white-box methods.
- White box monitoring also includes keeping an eye on the HTTP handlers along with usage logs

c) Who Maintains White-Box Monitoring?

White-box monitoring also used to be a job for both system admins and SREs. However, since it's all internal, you can use many automation tools to reduce the engineers' workload. As a result, we saw a shift of workload from system admins to DevOps engineers or SREs.

5.4.2 Black Box Monitoring

This type of monitoring mainly refers to the monitoring state of services in the system. Using this type of monitoring we ensure things like status of the application being alive or dead ,cpu / disk usage, memory usage, load averages etc.

Other types of black box monitoring include:

- Monitoring of network switches and other networking devices such as load balancers from the system metrics perspective, as defined above.
- Looking at hypervisor-level resource usage for all virtual machines running on the hypervisor (such as VMware, KVM, Xen, etc.).
- Alerting on hard disk errors that may present a problem if a disk isn't replaced soon (using SMART, for instance).

a) List of Black Box Monitoring tools

- Using the Black Box Monitoring involves using tools like Nagios , Zabbix, Datadog and Sysdig which are mainly based on the ideas of running custom checks on the systems to identify status of various applications / services whose response are mainly as 0 or 1 to indicate the status of the service being monitored. This approach mainly focuses on identifying the present status of the application rather then focusing upon the trend or issues with the applications.
- We can also define Black-box monitoring as monitoring externally visible resources. All the metrics monitored in black box monitoring are visible to the end-user, such as disk drive, RAM modules.

b) Applications of Black-Box Monitoring

- Black-Box monitoring has numerous applications. Some of the primary applications include monitoring the network switches and devices such as load balancers. Black-box monitoring is also used to monitor the hypervisor-level resources. Hypervisors are devices that allow multiple virtual machines to run on a system.
- Monitoring such vital devices can show much information about the devices and resource distribution.

c) Who Controls Black-Box Monitoring?

Usually, either DevOps or System Administrators are used to monitor the servers and other networking devices. Nowadays, we either see a shared responsibility model between DevOps and System Admins or SREs and admins for black box monitoring.

5.5 Building a Monitoring System

Most of the monitoring tools in DevOps have been categorized into the following types.

- Server monitoring: Also known as resource monitoring or infrastructure monitoring, server monitoring collects information about your server's performance. It provides statistics on RAM consumption, CPU loads, and disk space left. It is helpful in aggregating data from virtual server systems for cloud-based scenarios.
- The following are a few server monitoring tools in DevOps:
  - Prometheus
  - Splunk
  - Nagios
  - Zabbix
  - Sysdig
  - Sematext
  - Sensu
- Network monitoring: It examines the data flowing in and out of your computer network. Using this monitoring tool, you can keep track of incoming requests and responses across all components, from switches to firewalls and servers.
- Catchpoint is a network monitoring tool in DevOps

- Application performance monitoring: It measures how well a service is performing overall. These tools query the server on their own and keep track of metrics such as response time and completion. It is essential to ensure that application performance issues are promptly detected and diagnosed so that services can function as intended.

  The following are a few application performance monitoring (APM) tools in DevOps :

  - SignalFx
  - AppDynamics
  - Raygun
  - New Relic

- Here is a list of the best DevOps monitoring tools. These tools will help you to gear up for your work and progress in your career.

1. Sensu

- Sensu is one of the top DevOps monitoring tools; it is used for monitoring infrastructure and applications solutions. This platform allows you to measure and monitor the health of your infrastructure, applications, and business KPIs.
- Sensu combines dynamic, static, and temporary infrastructure to solve modern challenges in modern infrastructure platforms. Sensu does not offer software-as-a-service (SaaS,) but you can monitor your system just the way you want.

Features :

- It sends alerts and notifications.
- It provides dynamic registration and de-registration of clients.
- It is not affected by the presence of mission-critical applications or multi-tiered networks.
- It is perfect for automating processes.
- Despite being open-source, it has excellent commercial support.

2. PagerDuty

- PagerDuty is an operations performance platform designed to work closely with operations staff to assess the reliability and performance of apps and address errors as early as possible.
- When timely alerts come in from the development environment to the production environment, the operations team can detect, triage, and resolve the alerts faster. PagerDuty offers an excellent, easy-to-use incident response and alerting system.

Features :

- The intuitive alerting API of PagerDuty makes it very popular among users.
- If an alert does not respond after a set amount of time, the system will auto-escalate by the originally established SLA.
- It is a powerful GUI tool for scheduling and escalation policy.

3. Datical Deployment Monitoring Console

Datical deployment monitoring console is the solution that you would use to automatically track the deployment status of each database across the enterprise. This software receives and records SQL script execution events across the entire deployment environment. It does that to minimize human errors. In addition, it simplifies database auditing and deployment monitoring.

Features :

- A significant advantage that datical deployment monitoring console (DDMC) offers is the simplified auditing of databases.
- It tracks deployments and errors automatically.
- It provides access to deployment information on-demand.
- In addition, it simplifies the release processes so that both users and administrators can automatically track, audit, and resolve all deployment-specific database issues.

4. Tasktop Integration Hub

- Tasktop integration hub incorporates all tools within an organization into a single application to offer value to the organization. Tasktop integration hub is a single-point solution that handles all software delivery integration requirements without referring to another tool.
- Tasktop integration hub is a powerful tool to deliver the right information to the right people at the right time, using the right tool with the right interface.

Features :

- Connectivity is available for 45 tools that are fully functional without a problem.
- It allows the addition of new tools to existing software integration quickly.
- You can route artifacts as well as specific field updates according to a filter that complies with customer requirements around frequency and direction.
- It provides a secure login via a web-based interface.
- By monitoring changes to artifacts, this software runs at the lowest possible footprint and reduces the load on other tools.

5. Splunk :

Splunk is a sophisticated platform for analyzing machine data, especially logs generated frequently but seldom used effectively. Splunk is used for searching, monitoring, and analyzing machine-generated data using a web-based interface. It compiles all pertinent data into a central index that allows users to find the required information quickly.

Features :

- It enables the examination of data from networks, servers, apps, and various other data sources.
- It is simple enough to deploy in a production environment.
- It provides attributes such as Splunk light to transfer data from many servers to the main Splunk engine for analysis.
- It indexes data in such a way that it produces powerful analytic insights.
- Its reports are accurate to the decimal point, allowing any organization to identify steps for improvement and take action, if necessary.

5.6 Infrastructure Monitoring and Applications

Organizations can't afford to wait for alerts to come in when a system or application component has failed, especially if they plan to honor end-user service level agreements (SLAs). Instead, they need to adopt a proactive posture in which they identify and resolve potential infrastructure issues before they impact the user experience. Infrastructure monitoring helps organizations achieve this goal by accelerating root cause analysis, which in turn empowers cross-functional teams to effectively collaborate and fix problems before they flare into five-alarm fires.

Infrastructure monitoring also helps organizations analyze performance trends on an ongoing basis so they can better understand what peak performance looks like, optimize performance where appropriate, and flag potential issues well in advance.

DevOps teams can even leverage infrastructure monitoring as part of their A/B testing experiments. This way, teams can determine upfront how certain features or enhancements will impact application performance down the road. DevOps teams can also utilize infrastructure monitoring to validate deployments.

Your infrastructure monitoring will only deliver high-quality insights if you have the right tools with access to the right information. With that in mind, include these types of observability data sources :

- **Metrics** : Quantitative data is especially useful for creating visualizations and identifying patterns in performance over time. Values represented as counts or measures calculated or aggregated over a time period deliver crucial information for performance and state-based analysis.
- **Event logs**: Every system and service generates event logs, which can give you insights into what's happening and aid in troubleshooting.
- **Distributed traces**: For better insight into how various aspects of your environment interact with one another, capture distributed traces to record the journeys of specific transactions as they make their way through your infrastructure.
- **Metadata:** Additional information, such as topology details, name spaces, and priority data, will help you understand the significance and impact of events as they interact with other components of your infrastructure.
- **UX data**: A view into how users are experiencing your site or applications is one of the most important dimensions to understand how your infrastructure is performing. User experience data, such as page load times and latency, will give you better insight into your UX in real-time and empower you to adjust the UX if needed .
- **Open-source telemetry**: There are many open-source options designed to help you achieve better observability across your entire environment. These industry-standard tools include Open Telemetry , Prometheus, and StatsD, to name a few.
- **Cloud integrations**: Modern infrastructure includes cloud infrastructure, which is why cloud integrations, such as Cloud Watch for Amazon Web Services (AWS), can be helpful sources of observability data for infrastructure monitoring.

Following a few best practices will help you get the most value from your infrastructure monitoring program. For example:

- **Leverage automation:** Augment your capabilities with infrastructure monitoring tools that feature automation. This will help you gain complete end-to-end observability across the full stack and transition to AIOps for infrastructure monitoring.
- **Configure comprehensive alerts:** When your alerts are specific, they're less likely to result in false positives. Comprehensive alerts that provide a certain level of redundancy will be more likely to deliver the heads-up you need in any situation.
- **Prioritize alerts:** Organize and prioritize notifications so you don't miss the most important alerts — particularly the ones reporting impacts to the user experience.

- **Create role-specific dashboards:** Infrastructure monitoring tools allow you to create a range of custom dashboards that various teams within your organization will find useful when monitoring KPIs of importance to them. Set up dashboards for your ITOps teams, your security teams, and business leaders so everyone has access to the insights they need at a glance.
- **Do a test run:** As with any mission-critical system, you'll want to make sure your infrastructure monitoring tools are working as expected before you begin to rely on them on a day-to-day basis. Schedule a test run and make sure everything is running according to plan.
- **Regularly review metrics:** As your business goals change and your infrastructure evolves, so will the metrics and KPIs you need to track. Review them at regular intervals so you don't unintentionally develop any blind spots across your infrastructure.
- **Tap your vendor's expertise:** Struggling to fine-tune or optimize your infrastructure monitoring as your organization digitally transforms? Take full advantage of your infrastructure monitoring vendor's expertise. They've overseen infrastructure monitoring deployments across countless organizations and can help you achieve your monitoring goals faster.

With organizations focusing on modern infrastructure, traditional monitoring tools are not going to cut it. An all-in-one approach to monitoring cloud platforms and supporting applications and infrastructure is the best approach. Here are some key attributes to look for:

- **All-in-one platform :** Break down silos between apps and infrastructure teams with end-to-end visibility across the entire IT stack.
- **AI-assistance :** Use AI to detect anomalies and benchmark your system. This will allow your IT team to focus on what matters: proactive action, innovation, and business results.
- **Contextual information :** Go beyond metrics, logs, and traces with UX and topology data to understand billions of interdependencies.
- **Root-cause analysis :** Get actionable answers to problems in real-time, down to the code level.
- Automation for large-scale dynamic environments: This includes discovery, instrumentation, baselining, agent life-cycle management, and problem analysis.
- End-to-end coverage for hybrid cloud: Gain comprehensive support for multi-cloud, third-party integrations — including public cloud, on-prem virtualization, mainframes, database vendors, etc.

- Cloud-native architectures: These provide support for containers and serverless, including open standard, such as OpenTelemetry, Prometheus, StatsD, and Telegraf.

5.7 Collecting Data and Logging

- Logging and monitoring are two different processes that work together to provide a range of data points that help track the health and performance of your infrastructure. APM uses application metrics to measure availability and manage performance. Logging creates a record of log events generated from applications, devices, or web servers that serves as a detailed record of occurrences within a system.
- Using a combination of log management to collect, organize, and review data and monitoring tools to track metrics offers a comprehensive view of your system's availability along with detailed insight into any issues which could potentially affect the user experience. APM tells you how applications are behaving and log data from applications, network infrastructure, and web servers provides greater insight that will tell you why the application is performing as it is. An effective logging strategy enhances application performance monitoring.
- In a metaphorical sense, monitoring metrics is like the security alarm that alerts you to a possible intrusion; log files act as the security camera footage that will provide clues to tell you what happened and how.
- There will be some use cases where you will only need one or another, but having both offers you a greater ability to fully understand your system and its vulnerabilities.
- Ultimately, your goal is to maintain healthy applications and user experience. By integrating logging and monitoring to accomplish this goal, your developers and operations teams will be able to plan for and troubleshoot application issues faster.
- Create an effective strategy to optimize the integration of your logging and monitoring solutions by using the following best practices:

1. Enable both methods to work together

- If your ultimate goal is to optimize the benefits of analyzing log data and application metrics, facilitate that process by configuring your system to send log data directly to your monitoring tool. Storing log messages on the disk or sending them solely to a logging tool creates more of a drain on resources as well as a potential workflow bottleneck.
- Make sure that your monitoring tool supports your application's programming language to ensure compatibility and ease of use.

2. Log the right data

Log data needs to tell a succinct but complete story. Data should be selective, descriptive, and provide the correct context to assist with troubleshooting. Helpful log data generally includes actionable items, and includes information such as a timestamp, user IDs, session IDs, and resource-usage metrics. Collecting a full range of applicable data enhances the information obtained from your monitoring tool.

3. Use structured log data

Streamline your data by making it easier to search, index, and store be ensuring that it is structured. Structured data provides a more complete view as to what happened, and can provide your monitoring tool with unique identifiers such as which customer ID experienced the error. Providing customer ID information obtained via logging enables your monitoring tool to see how that specific user was affected, and what other issues they may be experiencing as a result.

4. Take full advantage of log data

- Logging offers more than simple troubleshooting and debugging. Identify application and system trends by applying statistical analysis to system events. Log data contains important information about your applications and underlying infrastructure, including all of your databases. Use the historical information provided by log data to determine averages that will make it easier to definitively identify anomalies, or to group event types in a way that allows for accurate comparisons. This data can also be beneficial for collecting, aggregating and viewing this data according to your enterprises' needs.
- Having statistical data sets for review allows for a more accurate analysis and an improved opportunity to make informed business decisions.

5.8 Creating Dashboard

- You build a great product. You offer it as a service. You define quality and performance Service Level Agreements (SLA) for your clients. You deploy monitoring to track Service Level Indicators to ensure you fulfill your SLA.
- And now you want to create monitoring dashboards. To visualize the metrics you collect and understand your product's behavior. How should you do that? What dashboards to create? What metrics should you add in each dashboard?

Strategy 1 : Yeah, I do not know. We have metrics, we plot metrics

- **All metrics, one dashboard**: One image is worth 1 word, so you add 1000 small charts in one dashboard. Wonder why metrics or trends get missed.
- **No correlation or flow between metric charts/panels:** Dashboard cohesion does not matter. Plot metrics as you remember then in any order from any level and layer. Force your team to spend time scanning across the dashboard for related metric panels placed very far apart. Wonder why nobody is using the dashboard.
- **Do not aggregate metrics :** Plot a separate line on each panel for each process/server/service instance. Do you have 100 service instances reporting response time? Plot 100 lines on your response time panel. Wonder why it's hard to assess the overall system behavior.
- **Mix metrics from different levels in the same panel :** Mix infrastructure and application level metrics in the same panel. Plot service instances count and response time in the same panel. Plot the number of processes and error rate in same panel. Wonder why panels are hard to read.
- **No variables, no drill-down :** It is what it is. Do not provide any variable parameters or drill-down options to select and view metrics for a certain client/service/environment. Assume that the dashboard will be cloned and any such selections hardcoded in each new clone. Wonder why it's becoming more and more difficult to find the original dashboard.
- **No overview dashboard** : Create a separate dashboard for each metric level. Create a separate dashboard for each component. Make your team go through 3 dashboards and mentally link business to application to infrastructure metrics for determining overall system health. Wonder why onboarding new team members takes forever.

Strategy 2 : Overview. Top-down. Left-right. Cohesive. Consistent.

- **Overview dashboard** : Build a dashboard to give a quick overview in the health of your system. Provide one top panel trumping everything, showing the highest level metric indicating system performance (or what we are tracking). A single glance at that panel should indicate if things are ok or not with our system.
- **Top-down structure** : Structure dashboard panels in rows and columns. Start from the highest level metrics at the top rows, and go down in metric level as you add additional rows. E.g. business-impacting metrics would be among the top panel rows, then application metrics, and infrastructure would be among the last.

- **Column per component :** If possible, reserve one column of panels per component or processing phase. Plot the same metric for each component/phase in separate horizontally arranged panels, on the same row. You should see from a first glance at the dashboard if the system has a problem. At a second glance you should see which component/processing phase has a problem. At a third glance you should see the problem source.

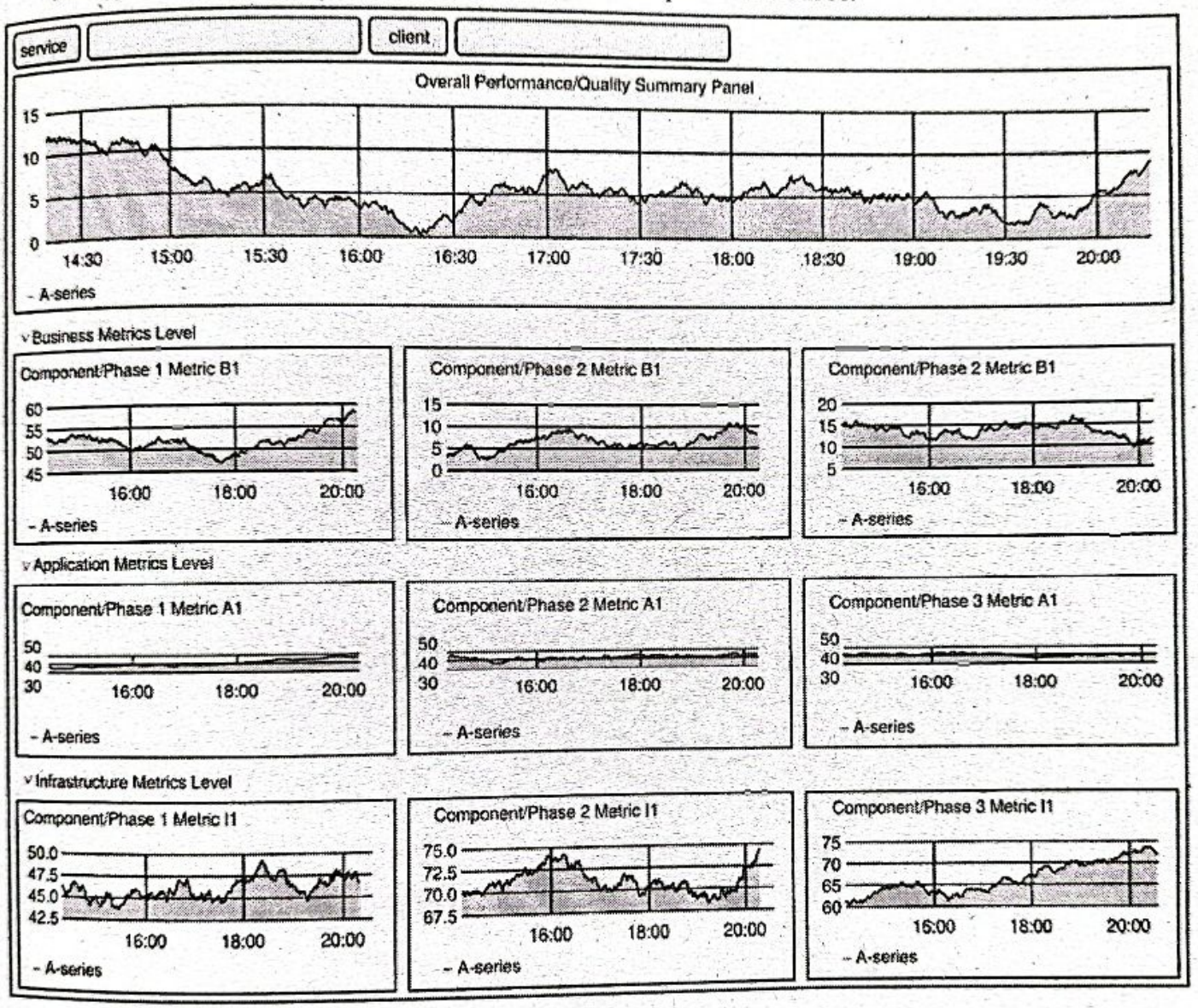


Fig. 5.8.1

- **Left-right structure:** As you navigate your dashboard left to right, if possible, the panels should plot metrics according to the data flow in your system. You should be able to quickly identify where in the data flow there is a problem.
- **Variable parameters and drill-down:** Provide the option torestrict the display of metrics to particular clients, services, or environments, using variableparameters and drill-down menu options. This is extremely useful when problems appear only with individual clients or services, and enables an engineer to focus on them while debugging.

- **Individual detailed dashboards:** Provide individual dashboards for debugging individual services and components. They should enable deep dives and debugging problems with specific services or components. Keep the same top-down left-right structure for your individual dashboards. Consistency across dashboards decreases debugging effort.

5.9 Behavior Driven Monitoring

- Behavior Monitoring constantly monitors endpoints for unusual modifications to the operating system or installed software.
- Behavior Monitoring is composed of the following sub-features:
  1. Malware Behavior Blocking
  2. Event Monitoring

1. **Malware Behavior Blocking** : Malware Behavior Blocking provides a necessary layer of additional threat protection from programs that exhibit malicious behavior. It observes system events over a period of time and as programs execute different combinations or sequences of actions, Malware Behavior Blocking detects known malicious behavior and blocks the associated programs. Use this feature to ensure a higher level of protection against new, unknown, and emerging threats.
2. **Event Monitoring :** Event Monitoring provides a more generic approach to protecting against unauthorized software and malware attacks. It uses a policy-based approach where system areas are monitored for certain changes, allowing administrators to regulate programs that cause such changes.

   If attempts to change the system are made, Event Monitoring will:

- Refer to the Event Monitoring policies and perform the configured action.
- Notify the user or administrator

5.10 What is Site Reliability Engineering

- Site Reliability Engineering (SRE) uses software engineering to automate IT operations tasks that would normally be performed by system administrators. These tasks include on-call monitoring, performance and capacity planning, and disaster response, production system management, change management and incident response. SRE teams use software as a tool to manage systems, solve problems, and automate operations tasks.

- Thus, SRE does the work traditionally done by operations but instead using engineers with software expertise to solve complex problems.
- Therefore, site reliability engineering can be considered a set of practices that incorporates aspects of software engineering into operations thereby increasing the efficiency and reliability of software systems and improving workflow. The goal of SRE is to promote a faster and more efficient workflow.
- The concept of site reliability engineering comes from the Google engineering team and is credited to Ben Treynor Sloss.
- SRE helps teams find a balance between releasing new features and making sure that they are reliable for users. Standardization and automation are 2 important components of the SRE model. Site reliability engineers should always be looking for ways to enhance and automate operations tasks.
- The SRE team's main goal is to find the best ways to prevent problems that can cause downtime. This is crucial especially when you manage large-scale systems. Another benefit is that Site Reliability Engineering helps brands eliminate manual work which gives developers much more time to innovate. Any flaws are found and fixed quickly and efficiently.
- According to SRE, a key precondition for a system's success is availability. If your service is unavailable at a certain time, it can't perform its functions.
- To measure the availability and thus ensure that everything goes right, SRE provides three metrics.

1. **Service-Level Indicator (SLI)** is a quantitative measurement of a system's behavior. The main SLI for most services is *request latency* — or the time needed to respond to a request. Other commonly used SLIs are *throughput of requests per second* and *errors per request*. These metrics are usually collected within a certain period of time and then converted into rates, averages, or percentiles.
2. **Service-Level Objective (SLO)** is a target range of values set by stakeholders (say, the average request latency must be under 100 milliseconds). The system is supposed to be reliable if its SLIs continuously meet SLOs.
3. **Service-Level Agreement (SLA)** is a promise to customers that your service will meet certain SLOs over a certain period. Otherwise, a provider will pay some kind of penalty. SRE isn't directly involved in setting SLAs. However, it helps to avoid missed SLOs and the financial losses they entail.

5.11 SRE and DevOps

- DevOps and SRE seem like two sides of the same coin. Both titles aim to bridge the gap between development and operation teams, with a unified goal of enhancing the release cycle without any compromises.
- SRE and DevOps have many things in common as they both are methodologies put in place to monitor production and ensure that operation management works as expected. Their common goal is a better result for complex distributed systems. Both believe that change is necessary to improve. Both focus on people working together as a team with shared responsibilities. DevOps and SRE believe that keeping everything working is everyone's responsibility. Ownership is shared – from initial code writing to software builds to deployment to production and maintenance. Both Site Reliability Engineering and DevOps engineers write and optimize code before deployment in production.

The Differences Between DevOps and SREs

- Site reliability engineering may be thought of as a specific implementation of DevOps, even though they were developed separately. There are many similarities in intent and foundational perspectives. Differences mainly result from a narrowing of team focus.
- Site Reliability Engineering is the next stage implementation of DevOps. DevOps is a philosophy with a wide range of implementation styles available. SRE is more prescriptive about how things are to be done and what the priorities of the team explicitly are, specifically, the job is to keep the site reliable and available and only things that contribute to this goal are prioritized.
- One big difference is that SRE teams always focus on service level objectives (SLOs), keeping them and improving systems to maximize effectiveness based on them. DevOps tend to think about what the data tells them about the system, how it is running, where it is weak or failing, and so on. SREs tend to be more specifically practical, thinking about how to use the same data to improve performance on one or more SLOs, even using machine learning techniques to have systems adapt themselves to changing circumstances.
- The biggest difference is that SRE has an intentionally narrowed focus on keeping services and platforms available to customers while DevOps tends to focus on overall processes, which is much broader.

DevOps	Site Reliability Engineering (SRE)
Devops is mindset and culture of collaboration	SRE is set of practices and metrics
Break down organizational silos	Share ownership of product across teams
Focus is on continuity and speed of product development and delivery	Focus is on system availability and reliability
Accept failure and fail fast	Closely examine failures in order to ensure that they don't happen twice, and plan for failure by incorporating failure costs into the budget
Introduce change gradually	Push changes out in small scale, and carefully test before a full-blown release
Leverage tools and automation	Consistently look for automation opportunities and try to implement tools that can optimize processes and remove manual work
Measure everything	Define and measure key performance indicators to track progress and health of systems

5.12 Roles and Responsibilities of SRE

- The role of Site Reliability Engineering in an organization is to keep the organization focused on what ultimately matters to customers: making sure the platforms and services customers rely on are available when customers want to use them.
- A Site Reliability Engineering team prioritizes reliability work using service level objectives (SLOs). SLOs are specific means of measuring the performance of the site, system, or service and are based on service level indicators (SLIs), which measure the service level provided to customers. Ultimately, these are used to create service level agreements (SLAs), which are the promises made to customers. The SLAs are the most lenient and the SLOs the most strict, to make sure we hold ourselves to a higher level of accountability than we promise our customers.

- SREs specialize in systems (operating systems, storage subsystems, networking), while implementing best practices for availability, reliability and scalability, with varied interests in algorithms and distributed systems.
- Site reliability engineers create a bridge between development and operations by applying a software engineering mindset to system administration topics. They split their time between operations/on-call duties and developing systems and software that help increase site reliability and performance.
- SREs collaborate closely with product developers to ensure that the designed solution responds to non-functional requirements such as availability, performance, security, and maintainability. They also work with release engineers to ensure that the software delivery pipeline is as efficient as possible.
- In an SRE team, all team members share responsibility for system maintenance, incident management, automation, and Chaos Engineering. Anything a team member can do that contributes to enhanced system reliability and availability is within scope.
- Every member of the SRE team is responsible for incident response and management and for reducing the time spent to fix problems when they arise. It is the team that is responsible for meeting the SLOs that have been committed to. The good news is that SRE teams are set up for maximum success because of this combination of shared responsibilities and diverse membership.
- Here are some general roles and responsibilities in a site reliability engineer job that SREs need to perform.

1. Software Engineering

Site reliability engineers incorporate various software engineering aspects to develop and implement services that improve IT and support teams. Services can range from production code changes to alerting and monitoring adjustments. The site reliability engineer job also includes tasks like building proprietary tools from the scratch to mitigate weaknesses in incident management or software delivery.

2. Troubleshooting Support Escalation

Site reliability engineers may have to spend a considerable amount of time fixing cases related to support escalation. They should fully know critical issues to route support escalation incidents to concerned teams. Critical support escalation cases, however, go down as site reliability engineering operations mature.

3. On-Call Process Optimization

In many organizations, the site reliability engineer job will involve the implementation of strategies that increase system reliability and performance through on-call rotation and process optimization. Site reliability engineers will also have to add automation for improved collaborative response in real-time, besides updating documentation, runbook tools, and modules to ready teams for incidents.

4. Documenting Knowledge

As site reliability engineers take part in on-call duties, IT operations, software development, and support, they gain substantial historical knowledge. To ensure a seamless flow of information between teams, site reliability engineer job may require documenting the knowledge gained. SREs are tasked with putting together internal documentation, playbooks, and other consolidated knowledge repositories that can help existing teams and future hired resources.

5. Optimizing SDLC (Software Development Life Cycle)

Site reliability engineers must ensure that IT professionals and software developers are reviewing incidents and documenting the findings to enable informed decision-making. Based on post-incident reviews, site reliability engineers will need to optimize the Software Development Life Cycle (SDLC) to boost service reliability.

5.13 Common Tools used by SREs

A. APM or General Monitoring Tools

Application performance monitoring (APM) tools provide granular visibility into the entire application stack, reporting on performance from the user's perspective. General monitoring tools usually focus on a single application. SREs use APM and monitoring tools to capture, measure, and track reliability metrics across the environment.

1. Datadog

Datadog offers cloud monitoring functionality. You can use Datadog to set up monitors, view existing infrastructure hosts, collect events, and more. Datadog offers features that let you customize and integrate the solution with other systems.

2. **Kibana**

Kibana is a free data visualization platform that collects metrics, typically from Elasticsearch clusters. If you are using the Elastic stack (ELK stack), then Kibana is the most suitable tool for the job. Kibana also offers many other services such as SecOps and business analytics that make it a valuable tool. Since it's free, Kibana is a good option for small businesses and startup too.

3. **New Relic :**

New Relic was the pioneer of application performance monitoring (APM). It offers a cloud-based full-stack observability platform that specializes in performance monitoring, and telemetry. You can use the platform to track the performance of distributed applications and services on a single dashboard.

4. **Appdynamics :** Full-stack observability platform that provides real-time data insights for system performance and helps in driving business growth and productivity

5. **Dynatrace :** This tool has got observability, security features, intelligent solutions, and automation features in a single platform that helps developers to monitor the performance of the system effectively

B. Continuous integration / Continuous Deployment (CI/CD) Tools

Continuous integration is the automatic testing practice of every change that has been affected on the source code. And continuous deployment follows continuous integration by pushing the tested codebase to the production environment. Here are few tools that can help in executing these functions,

- **Jenkins** : CI/CD Automation platform that supports automation across development, deployment, and testing of any project
- **CircleCI** : A CI/CD platform that helps in automating the application development process either across the platform's cloud or organization's own infrastructure
- **GitLab** : It is an open core model of open-source DevOps platform that helps with collaboration, gaining visibility, and enhances development velocity
- **GoCD** : Free open-source CI/CD server that helps with easy modeling and visualization of complex workflows

C. Configuration Management Tools

SRE teams use configuration management tools to track changes to applications and infrastructure, prevent and monitor for unauthorized changes, and automate deployments and infrastructure updates to make them predictable and reliable.

- **Terraform :** Terraform is a tool from HashiCorp that symbolizes the term infrastructure-as-code (IaC). It allows DevOps teams to describe their infrastructure components, such as VM, Kubernetes clusters, databases, or VPCs using a domain-specific configuration language. Then it takes these descriptions and creates the infrastructure components in the cloud environments.
- **Ansible :** Ansible is a tool that automates IT infrastructure. It primarily uses YAML files to describe roles, services and tasks that need to run in a specific order. On each run, Ansible will connect to machines using SSH and run the tasks described in the playbooks as scripts. At that point, it removes any scripts or temporary information from the connected hosts and reports the status back to the user. Since it's written in Python, Ansible is very extensible and it can handle a wide variety of roles and scripts.
- **SaltStack :** SaltStack is another IT infrastructure and configuration management tool with an unusual approach. It relies on agents installed in hosts, which then use a data-driven orchestration of communicating commands. If configured correctly, it can automate deployments into thousands of nodes with minimal effort. SaltStack was acquired by VMware.
- **Chef** : Streamlines configuration management tasks across cloud platforms to automatically provision new machines
- **Puppet** : Model-driven software configuration management tool used to manage the entire lifecycle of IT infrastructure

D. Real-Time Communication Tools

Using real-time communication tools greatly improves response readiness. SRE teams need to collaborate in a timely manner and quickly solve problems before they escalate. To do this, they need a messaging platform that enables interpersonal communication in a closed, secure environment, and can integrate with operational systems to stream notifications and alerts to SREs.

- **Slack :** Slack is a popular real-time communication platform now offered by Salesforce. It provides a primary collaboration tool for businesses and teams. SRE teams can use Slack for interpersonal messaging, and also as a programmatic platform that can help automate responses and coordinate events. You can use Slack to set up hooks to other systems, such as ChatOps services.
- **Telegram :** Telegram offers simple and reliable messaging functionality. The application is free and provides an application programming interface (API) that enables programmatic access. Some SRE teams use it as a lightweight alternative to Slack.

- **Microsoft Teams :** Teams is a real-time communication solution offered by Microsoft, typically used by organizations already using Office 365. Teams offers chat-based collaboration functionality that comes with features including online meetings and document sharing. Microsoft offers a free version of Teams for a maximum of 100 users.

E. Incident Management / On-call Alerting System Tools

An incident management tool is an essential part while managing system architecture. Automated incident response systems can help detect and respond to incidents as they occur.

- **Pagerduty :** PagerDuty offers cloud-based incident response functionality designed especially for incident management and on-call rotations. It integrates with a variety of DevOps tools. A major advantage of PagerDuty is that it provides a native app that lets you receive notifications and calls on your mobile devices or smartwatches.
- **Opsgenie :** Opsgenie is an incident response solution offered by Atlassian. It provides actionable alerting with automated grouping and filtering of alerts, on-call scheduling with routing rules and escalations, and a reporting and analytics module that lets you track incident response metrics and team productivity.
- **Squadcast :** Cloud-based incident management platform built around Site reliability engineering (SRE) best practices that helps to improve incident resolution metrics and ultimately, the reliability of systems
- **VictorOps (Splunk On-Call) :** VictorOps is offered as part of the Splunk On-Call solution, which provides enterprise-grade incident response capabilities. It provides on-call scheduling tools like schedules and automations, adds context to alerts to enable easier remediation, and provides native apps for both iOS and Android.

F. Project Tracking Tools

You should also keep track of incidents and record events using a project tracker. I would recommend the following:

- **Jira :** This is Atlassian's main and most ubiquitous product. It's an agile platform for tracking projects and team progress, and it's used by professional organizations of all sizes. The downside is that it looks and feels very slow sometimes.
- **Trello :** This is also from Atlassian, but it's more approachable and easier to use than Jira. You can get started with Trello for free, and it scales really well without much investment.
- **Asana :** This agile project management service is free to start and grows with your business. It's a good alternative to Jira and has a growing user base.

G. Monitoring and Observability Tools

Monitoring and observability are two main functions in maintaining system health. SREs work closely with these monitoring tools. The prime role of site reliability engineers is to develop custom queries across alert managers that are present inside the monitoring tools' architecture. These functions check whether all the system functionalities are working as expected. And helps to generate alerts when there is any deviation in system behavior.

- **Prometheus :** An open-source monitoring tool that provides a dimensional (time-series) data model of all system performance characteristics
- **InfluxDB :** Supports the development team to build and monitor time-stamped data series across the infrastructure
- **OpenTelemetry :** Open-source observability framework for monitoring cloud-native software applications with telemetry data. OpenTracing and OpenConsensus have merged to form a standardized OpenTelemetry tool
- **Jager :** Open-source end-to-end distributed tracing platform that helps in monitoring and troubleshooting issues across a distributed network

H. Dashboarding Tools

Dashboarding tools help SREs to scrutinize issues more efficiently by displaying all the necessary data (Key Performance Indicators and Critical data points) in one screen. These tools facilitate pictorial or graphical representation of system data, thereby giving precise information about the system's health.

- **Grafana** : Provides an integrated solution to metrics and logs for composing observability characteristics in the form of graphical representation
- **Stashboard** : Status based dashboard solution for APIs and service-based software solutions
- **Redash** : Helps to connect and create queries on data sources to visualize all the data in the form of a dashboard for easy collaboration across various teams
- **Metabase** : An open-source tool for self-hosted platforms that enables them to connect data points for visualization purposes. Whereas, Metabase Cloud platform has exclusive advanced features like single sign-on and embedded analytics

DevOps should not only focus on delivery pipelines, fancy technologies and cutting-edge tools, but consider communication and culture as well. This should be considered not just in a development team but in an organization as a whole. With Behavior-Driven Development (BDD), useful methods can help us establish a standardized communication between a Project Owner (PO), development teams including quality assurance (Dev) and operations (Ops). When BDD is implemented properly, it can prevent social friction and misunderstandings.

6

UNIT - VI

DevOps Tooling and Case Studies

Syllabus

Continuous Development/ Version Control: Git, Serverless orchestration: Kubernetes, Container Technology: Docker, Continuous Integration: Jenkins, Continuous delivery: Jenkins, Continuous Deployment: Ansible, Continuous Testing: Selenium, Monitoring: Prometheus, Bug tracking tool: Jira, elk stack. Case study: Spotify: Using Docker, Bank of New Zealand, EtSy.

6.1 Version Control : Git

- Version control systems are software tools that help software teams manage changes to source code over time. Version control, also known as source control, is the practice of tracking and managing changes to software code.
- Source code acts as a single source of truth and a collection of a product's knowledge, history, and solutions. Version control (or code revision control) serves as a safety net to protect the source code from irreparable harm, giving the development team the freedom to experiment without fear of causing damage or creating code conflicts.
- The most common type of VCS is a centralized VCS, which uses a server to store all the versions of a file. Developers can check out a file from the server, make changes, and check the file back in. The server then stores the new version of the file
- The purpose of version control is to allow software teams track changes to the code, while enhancing communication and collaboration between team members. Version control facilitates a continuous, simple way to develop software.
- As development environments have accelerated, version control systems help software teams work faster and smarter. They are especially useful for DevOps teams since they help them to reduce development time and increase successful deployments.

A] Why Version Control

- Basically, Version control system keeps track on changes made on a particular software and take a snapshot of every modification. Let's suppose if a team of developer add some new functionalities in an application and the updated version is not working properly so as the version control system keeps track of our work so with the help of version control system we can omit the new changes and continue with the previous version. Version control used to collaborate and maintains the history of work as the project progresses so developers can work in parallel without overriding each other's changes.
- Version control system provides following benefits
  - It enhances the project development speed by providing efficient collaboration
  - Improves the productivity, expedites product delivery, and skills of the employees through better communication and assistance
  - It reduce possibilities of errors and conflicts meanwhile project development through traceability to every small change
  - Employees or contributors of the project can contribute from anywhere irrespective of the different geographical locations through this VCS
  - For each different contributor to the project, a different working copy is maintained and not merged to the main file unless the working copy is validated. The most popular example is Git, Helix core, Microsoft TFS,
  - It helps in recovery in case of any disaster or contingent situation,
  - Informs us about Who, What, When, Why changes have been made.

B] Types of Version control System

Following are the three types of version control system

1. Local Version Control Systems
2. Centralized Version Control Systems
3. Distributed Version Control Systems

1. Local Version Control Systems :

It is one of the simplest forms and has a database that kept all the changes to files under revision control. RCS is one of the most common VCS tools. It keeps patch sets (differences between files) in a special format on disk. By adding up all the patches it can then re-create what any file looked like at any point in time.

2. Centralized Version Control Systems :

- A centralized version control system (CVCS) is a type of version control system (VCS) where all users are working with the same central repository. This central repository can be located on a server or on a developer's local machine. CVCSs are typically used in software development projects where a team of developers needs to share code and track changes.
- Two things are required to make your changes visible to others which are :
  - You commit
  - They update

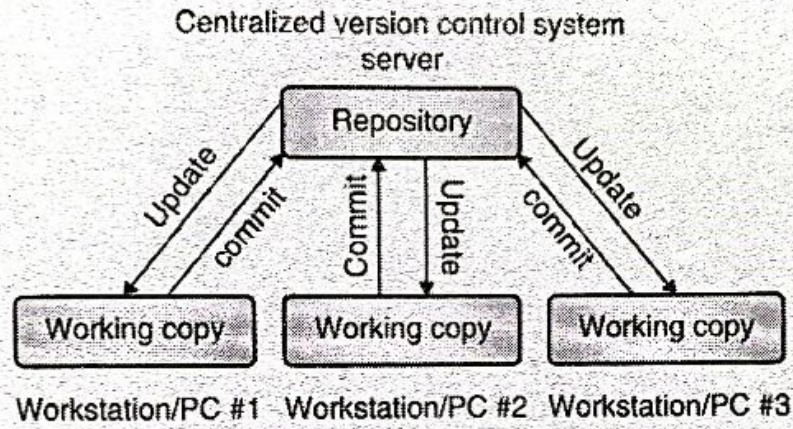


Fig.6.1.1 : Centralized Version Control System

- It has some downsides as well which led to the development of DVS. The most obvious is the single point of failure that the centralized repository represents if it goes down during that period collaboration and saving versioned changes is not possible.
- What if the hard disk of the central database becomes corrupted, and proper backups haven't been kept? You lose absolutely everything. Here, distributed version control system (DVCS) comes into picture.

3. Distributed Version Control Systems :

- Distributed version control systems contain multiple repositories. Each user has their own repository and working copy. Just committing your changes will not give others access to your changes. This is because commit will reflect those changes in your local repository and you need to push them in order to make them visible on the central repository. Similarly, When you update, you do not get others' changes unless you have first pulled those changes into your repository.
- A distributed version control system (DVCS) brings a local copy of the complete repository to every team member's computer, so they can commit, branch, and merge

locally. The server doesn't have to store a physical file for each branch — it just needs the differences between each commit.

- DVCSs are often used by developers who need to work on projects from multiple computers or who need to collaborate with other developers remotely.
- To make your changes visible to others, 4 things are required:
  - You commit
  - You push
  - They pull
  - They update
- The most popular distributed version control systems are Git, and Mercurial. They help us overcome the problem of single point of failure.

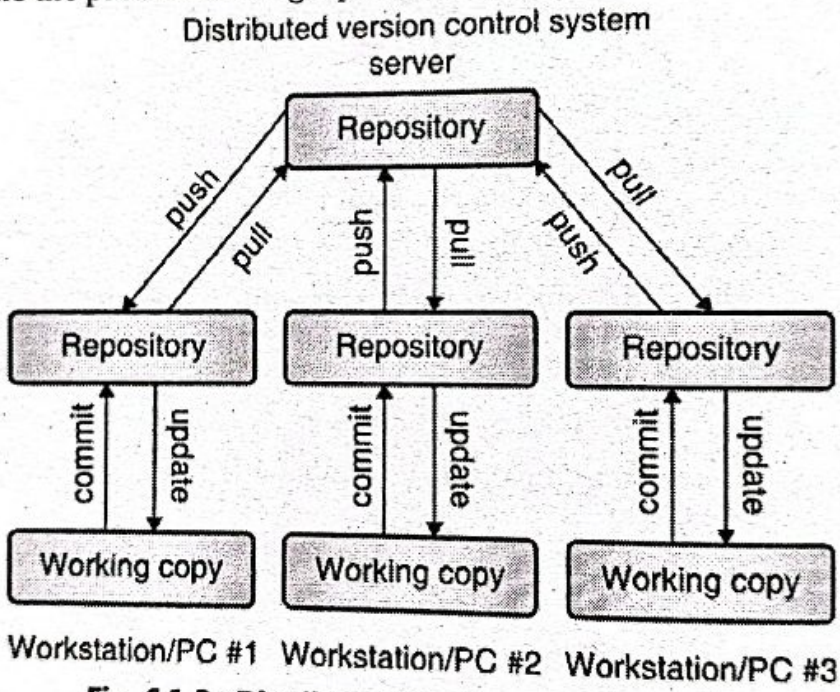


Fig. 6.1.2 : Distributed Version Control System

- Distributed version control systems help software development teams create strong workflows and hierarchies, with each developer pushing code changes to their own repository and maintainers setting a code review process to ensure only quality code merges into the main repository.
- With Distributed version control systems, multiple people can work simultaneously on a single project. Everyone works on and edits their own copy of the files and it is up to them when they wish to share the changes made by them with the rest of the team.

6.1.1 Git

- Git is a modern and widely used distributed version control system in the world. It is developed to manage projects with high speed and efficiency. The version control system allows us to monitor and work together with our team members at the same workspace.
- It originally created by Linus Torvalds in 2005. Unlike older centralized version control systems such as SVN and CVS, Git is distributed: every developer has the full history of their code repository locally.
- Git also has excellent support for branching, merging, and rewriting repository history, which has led to many innovative and powerful workflows and tools.
- Git is the most widely used version control system in the world today and is considered the modern standard for software development.
- It is mainly used for Tracking code changes ,tracking who made changes and coding collaboration. Git is foundation of many services like GitHub and GitLab, but we can use Git without using any other Git services. Git can be used privately and publicly. Git is easy to learn, and has fast performance. It is superior to other SCM tools like Subversion, CVS, Perforce, and ClearCase.

A] Features of Git

- **Open Source :** Git is an open-source tool. It is released under the GPL (General Public License) license.
- **Scalable :** Git is scalable, which means when the number of users increases, the Git can easily handle such situations.
- **Distributed :** One of Git's great features is that it is distributed. Distributed means that instead of switching the project to another machine, we can create a "clone" of the entire repository. Also, instead of just having one central repository that you send changes to, every user has their own repository that contains the entire commit history of the project.
- **Security :** Git is secure. It uses the SHA1 (Secure Hash Function) to name and identify objects within its repository. Files and commits are checked and retrieved by its checksum at the time of checkout.
- **Speed :** Git is very fast, so it can complete all the tasks in a while. Most of the git operations are done on the local repository, so it provides a huge speed.

- **Branching and Merging :** Git allows the creation of multiple branches without affecting each other. We can perform tasks like creation, deletion, and merging on branches, and these tasks take a few seconds only.
- **Maintain the clean history :** It fetches the latest commits from the master branch and puts our code on top of that. Thus, it maintains a clean history of the project.

B] Move Source Code to Git from command line

- We are going to show how to move the source code to Git from the command line interface, so that the knowledge of how Git can be used is clearer to the end user.

Step 1 : Initialize the Git Repository. Go to the command prompt, go to your project folder and issue the command git init. This command will add the necessary Git files to the project folder, so that it can be recognized by Git when it needs to be uploaded to the repository.

Step 2 : Adding your files which need to be added to the Git repository. This can be done by issuing the git add command. The dot option tells Git that all files in the project folder need to be added to the Git repository.

Step 3 : The final step is to commit the project files to the Git repository. This step is required to ensure all files are now a part of Git. The command to be issued is given in the following screenshot. The –m option is to provide a comment to the upload of files.

6.2 Serverless Orchestration : Kubernetes

- Workflows that involve branching logic, different types of failure models and retry logic typically use an orchestrator to keep track of the state of the overall execution.
- Many organizations that use Amazon Web Services (AWS) initially used AWS Lambda, Amazon's Function-as-a-Service product, to automate workflows around running and scaling code. While this service does save time for developers, many companies have found challenges using AWS Lambda with more complex applications.
- To address these issues, Amazon launched AWS Step Functions, a serverless function orchestrator that simplifies the process of managing and sequencing Lambda functions. With AWS Step Functions, businesses can improve application resiliency, leverage other powerful AWS services, write efficient code, and much more.
- AWS Step Functions is a serverless orchestration service that lets developers create and manage multi-step application workflows in the cloud. At each step of a given workflow, Step Functions manages input, output, error handling, and retries, so that developers can focus on higher-value business logic for their applications.

- AWS Step Functions is useful for any engineering teams who need to build workflows across multiple Amazon services. Use cases for Step Functions vary widely, from orchestrating serverless microservices, to building data-processing pipelines, to defining a security-incident response.

6.2.1 Kubernetes

- Kubernetes is software that automatically manages, scales, and maintains multi-container workloads in desired states
- Modern software is increasingly run as fleets of containers, sometimes called microservices. A complete application may comprise many containers, all needing to work together in specific ways. Kubernetes is software that turns a collection of physical or virtual hosts (servers) into a platform that :
  - Hosts containerized workloads, providing them with compute, storage, and network resources, and
  - Automatically manages large numbers of containerized applications — keeping them healthy and available by adapting to changes and challenges

A] How does Kubernetes work ?

- When developers create a multi-container application, they plan out how all the parts fit and work together, how many of each component should run, and roughly what should happen when challenges (e.g., lots of users logging in at once) are encountered.
- They store their containerized application components in a container registry (local or remote) and capture this thinking in one or several text files comprising a*configuration*. To start the application, they "apply" the configuration to Kubernetes.
- Kubernetes job is to evaluate and implement this configuration and maintain it until told otherwise. It:
  - Analyzes the configuration, aligning its requirements with those of all the other application configurations running on the system
  - Finds resources appropriate for running the new containers (e.g., some containers might need resources like GPUs that aren't present on every host)
  - Grabs container images from the registry, starts up the new containers, and helps them connect to one another and to system resources (e.g., persistent storage), so the application works as a whole

- Then Kubernetes monitors everything, and when real events diverge from desired states, Kubernetes tries to fix things and adapt. For example, if a container crashes, Kubernetes restarts it. If an underlying server fails, Kubernetes finds resources elsewhere to run the containers that node was hosting. If traffic to an application suddenly spikes, Kubernetes can scale out containers to handle the additional load, in conformance to rules and limits stated in the configuration.

B] Why use Kubernetes ?

- One of the benefits of Kubernetes is that it makes building and running complex applications much simpler. Here's a handful of the many Kubernetes features:
  - Standard services like local DNS and basic load-balancing that most applications need, and are easy to use.
  - Standard behaviors (e.g., restart this container if it dies) that are easy to invoke, and do most of the work of keeping applications running, available, and performant.
  - A standard set of abstract "objects" (called things like "pods," "replicasets," and "deployments") that wrap around containers and make it easy to build configurations around collections of containers.
  - A standard API that applications can call to easily enable more sophisticated behaviors, making it much easier to create applications that manage other applications.

C] Kubernetes Architecture

- A Kubernetes cluster has two main components-the control plane and data plane, machines used as compute resources.
- **The control plane** hosts the components used to manage the Kubernetes cluster.
- **Worker nodes** can be virtual machines (VMs) or physical machines. A node hosts pods, which run one or more containers.
- Kubernetes Control Plane contain following component
  - kube-apiserver
  - kube-scheduler
  - kube-controller-manager
  - etcd
  - cloud-controller-manager

- Kubernetes Worker Nodes component
  - Nodes
  - Pods
  - Container Runtime Engine
  - kubelet
  - kube-proxy
  - Container Networking

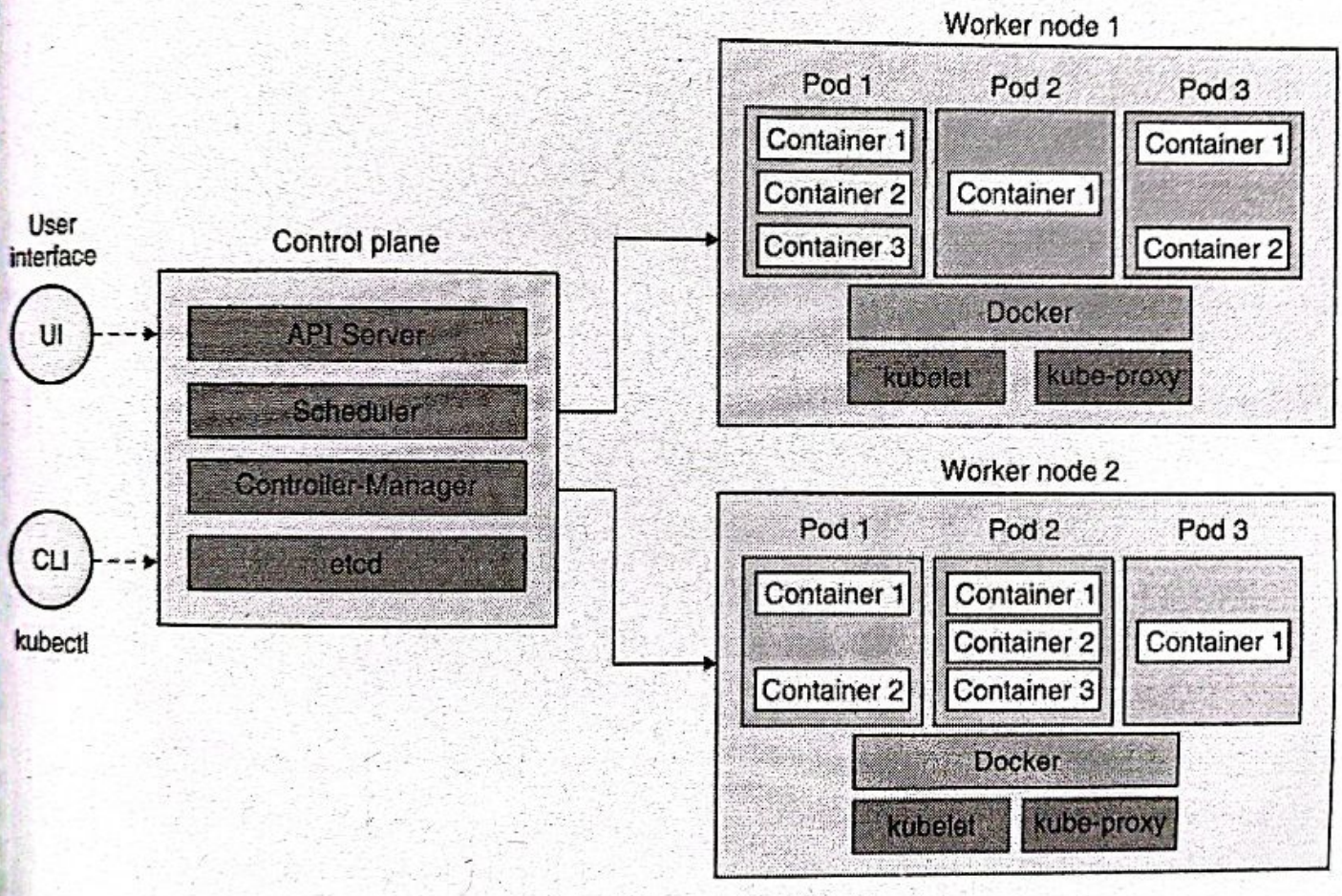


Fig. 6.2.1 : Kubernetes Architecture

Control Plane

- A control plane serves as a nerve center of each Kubernetes cluster. It includes components that can control your cluster, its state data, and its configuration.
- The Kubernetes control plane is responsible for ensuring that the Kubernetes cluster attains a desired state, defined by the user in a declarative manner. The control plane interacts with individual cluster nodes using the kubelet, an agent deployed on each node.
- Here are the main components of the control plane :

1. **kube-apiserver :**

   Provides an API that serves as the front end of a Kubernetes control plane. It is responsible for handling external and internal requests—determining whether a request is valid and then processing it. The API can be accessed via the kubectl command-line interface or other tools like kubeadm, and via REST calls.

2. **kube-scheduler:**

   This component is responsible for scheduling pods on specific nodes according to automated workflows and user defined conditions, which can include resource requests, concerns like affinity and taints or tolerations, priority, persistent volumes (PV), and more.

3. **kube-controller-manager :**

   - The Kubernetes controller manager is a control loop that monitors and regulates the state of a Kubernetes cluster. It receives information about the current state of the cluster and objects within it, and sends instructions to move the cluster towards the cluster operator's desired state.
   - The controller manager is responsible for several controllers that handle various automated activities at the cluster or pod level, including replication controller, namespace controller, service accounts controller, deployment, statefulset, and daemonset.

4. **etcd**

   A key-value database that contains data about your cluster state and configuration. Etcd is fault tolerant and distributed.

5. **cloud-controller-manager**

   - This component can embed cloud-specific control logic - for example, it can access the cloud provider's load balancer service. It enables you to connect a Kubernetes cluster with the API of a cloud provider. Additionally, it helps decouple the Kuberneters cluster from components that interact with a cloud platform, so that elements inside the cluster do not need to be aware of the implementation specifics of each cloud provider.
   - This cloud-controller-manager runs only controllers specific to the cloud provider. It is not required for on-premises Kubernetes environments. It uses multiple, yet logically-independent, control loops that are combined into one binary, which can run as a single process. It can be used to add scale a cluster by adding more nodes on cloud VMs, and

leverage cloud provider high availability and load balancing capabilities to improve resilience and performance.

- Kubernetes Core Components: Worker Nodes

6. **Nodes**

Nodes are physical or virtual machines that can run pods as part of a Kubernetes cluster. A cluster can scale up to 5000 nodes. To scale a cluster's capacity, you can add more nodes.

7. **Pods**

A pod serves as a single application instance, and is considered the smallest unit in the object model of Kubernetes. Each pod consists of one or more tightly coupled containers, and configurations that govern how containers should run. To run stateful applications, you can connect pods to persistent storage, using Kubernetes

8. **Container Runtime Engine**

Each node comes with a container runtime engine, which is responsible for running containers. Docker is a popular container runtime engine, but Kubernetes supports other runtimes that are compliant with Open Container Initiative, including CRI-O and rkt.

9. **kubelet**

Each node contains a kubelet, which is a small application that can communicate with the Kubernetes control plane. The kubelet is responsible for ensuring that containers specified in pod configuration are running on a specific node, and manages their lifecycle.. It executes the actions commanded by your control plane.

10. **kube-proxy**

All compute nodes contain kube-proxy, a network proxy that facilitates Kubernetes networking services. It handles all network communications outside and inside the cluster, forwarding traffic or replying on the packet filtering layer of the operating system.

11. **Container Networking**

- Container networking enables containers to communicate with hosts or other containers. It is often achieved by using the container networking interface (CNI), which is a joint initiative by Kubernetes, Apache Mesos, Cloud Foundry, Red Hat OpenShift, and others.
- CNI offers a standardized, minimal specification for network connectivity in containers. You can use the CNI plugin by passing the kubelet --network-plugin=cni command-line option. The kubelet can then read files from --cni-conf-dir and use the CNI configuration when setting up networking for each pod.

6.3 Container Technology : Docker

Containers provide a way of creating an isolated environment, sometimes called a sandbox, in which applications and their dependencies can live.

Why containers ?

- **Portability** : the isolated environment that containers provide effectively means the container is decoupled from the environment in which they run. Basically, they don't care much about the environment in which they run, which means they can be run in many different environments with different operating systems and hardware platforms.
- **Consistency** : since the containers are decoupled from the environment in which they run, you can be sure that they operate the same, regardless of where they are deployed. The isolated environment that they provide is the same across different deployment environments.
- **Speed to deploy** : for the same reasons as above. There is no need for considerations around how the application will operate in a production environment. If it runs in a container in one environment (say, your local machine), then it can be made to run in a container in another environment (say, in a cloud provider) very quickly

6.3.1 Docker

Docker is a software platform that allows you to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime. Using Docker, you can quickly deploy and scale applications into any environment and know your code will run.

A. How Docker Works ?

Docker works by providing a standard way to run your code. Docker is an operating system for containers. Similar to how a virtual machine virtualizes (removes the need to directly manage) server hardware, containers virtualize the operating system of a server. Docker is installed on each server and provides simple commands you can use to build, start, or stop containers.

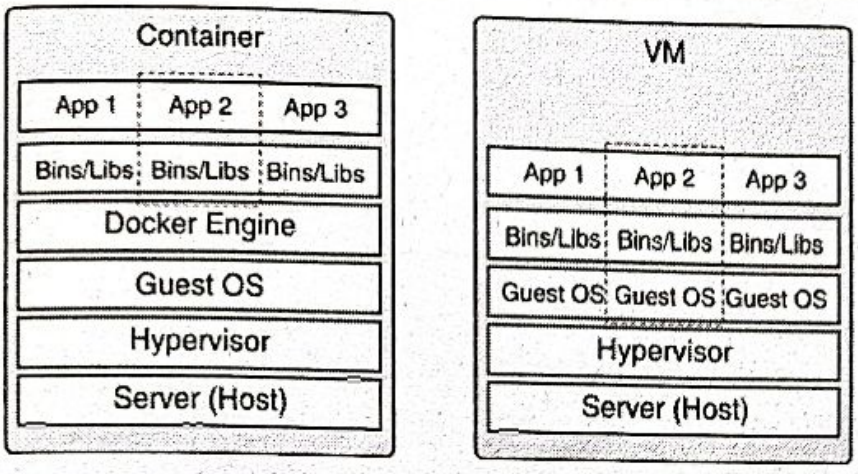


Fig.6.3.1

B. Why use Docker ?

Using Docker lets you ship code faster, standardize application operations, seamlessly move code, and save money by improving resource utilization. With Docker, you get a single object that can reliably run anywhere. Docker's simple and straightforward syntax gives you full control. Wide adoption means there's a robust ecosystem of tools and off-the-shelf applications that are ready to use with Docker.

C. When to use Docker ?

You can use Docker containers as a core building block creating modern applications and platforms. Docker makes it easy to build and run distributed microservices architecures, deploy your code with standardized continuous integration and delivery pipelines, build highly-scalable data processing systems, and create fully-managed platforms for your developers. The recent collaboration between AWS and Docker makes it easier for you to deploy Docker Compose artifacts to Amazon ECS and AWS Fargate.

D. Components of a Docker architecture

Docker comprises the following different components within its core architecture :

1. Images
2. Containers
3. Registries
4. Docker Engine

1. Images

Images are like blueprints containing instructions for creating a Docker container. Images define application dependencies and processes that should run when the application launches. You can get images from DockerHub or create your own images by including specific instructions within a file called Dockerfile.

2. **Containers**
   - Containers are live instances of images on which an application or its independent modules are run.
   - In an object-oriented programming analogy, an image is a class and the container is an instance of that class. This allows operational efficiency by allowing to you to multiple containers from a single image.
3. **Registries**
   - A Docker registry is like a repository of images. The default registry is the Docker Hub, a public registry that stores public and official images for different languages and platforms. By default, a request for an image from Docker is searched within the Docker Hub registry. You can also own a private registry and configure it to be the default source of images for your custom requirements.
4. **Docker Engine**
   - The Docker Engine is of the core components of a Docker architecture on which the application runs. You could also consider the Docker Engine as the application that's installed on the system that manages containers, images, and builds.
   - A Docker Engine uses a client-server architecture and consists of the following sub-components :
     - **The Docker Daemon** is basically the server that runs on the host machine. It is responsible for building and managing Docker images.
     - **The Docker Client** is a command-line interface (CLI) for sending instructions to the Docker Daemon using special Docker commands. Though a client can run on the host machine, it relies on Docker Engine's REST API to connect remotely with the daemon.
     - **A REST API** supports interactions between the client and the daemon.

6.4 Continuous Integration

Continuous integration is a DevOps software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. Continuous integration most often refers to the build or integration stage of the software release process and entails both an automation component (e.g. a CI or build service) and a cultural component (e.g. learning to integrate frequently). The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

Continuous Integration Benefits :

- **Improve Developer Productivity :** Continuous integration helps your team be more productive by freeing developers from manual tasks and encouraging behaviors that help reduce the number of errors and bugs released to customers.
- **Find and Address Bugs Quicker :** With more frequent testing, your team can discover and address bugs earlier before they grow into larger problems later.
- **Deliver Updates Faster :** Continuous integration helps your team deliver updates to their customers faster and more frequently.

6.4.1 Jenkins

- Jenkins is an open-source continuous integration tool which is written in Java. By default, Jenkins will be running on port 8080. It is a master-slave topology which distributes the build and testing efforts over slave servers with the results automatically accumulated on the master.
- It is not only a continuous integration server but also has a highly active community that works towards improving codes, write plugins, participates on mailing lists, writes bug reports, etc. It can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed. Jenkins helps an organisation to advance the software development process through automation.
- **Jenkins** is a server-based application and requires a web server like Apache Tomcat to run on various platforms like Windows, Linux, macOS, Unix, etc. To use Jenkins, you need to create pipelines which are a series of steps that a Jenkins server will take. Jenkins Continuous Integration Pipeline is a powerful instrument that consists of a set of tools designed to **host, monitor, compile** and **test** code, or code changes, like:
  - **Continuous Integration Server** (Jenkins, Bamboo, CruiseControl, TeamCity, and others)
  - **Source Control Tool** (e.g., CVS, SVN, GIT, Mercurial, Perforce, ClearCase and others)
  - **Build tool** (Make, ANT, Maven, Ivy, Gradle, and others)
  - **Automation testing framework** (Selenium, Appium, TestComplete, UFT, and others)
- By default, Jenkins comes with a limited set of features. If you want to integrate your Jenkins installation with version control tools like Git, then you need to install plugins related to Git. In fact, for integration with tools like Maven, Amazon EC2, you need to install respective plugins in your Jenkins.

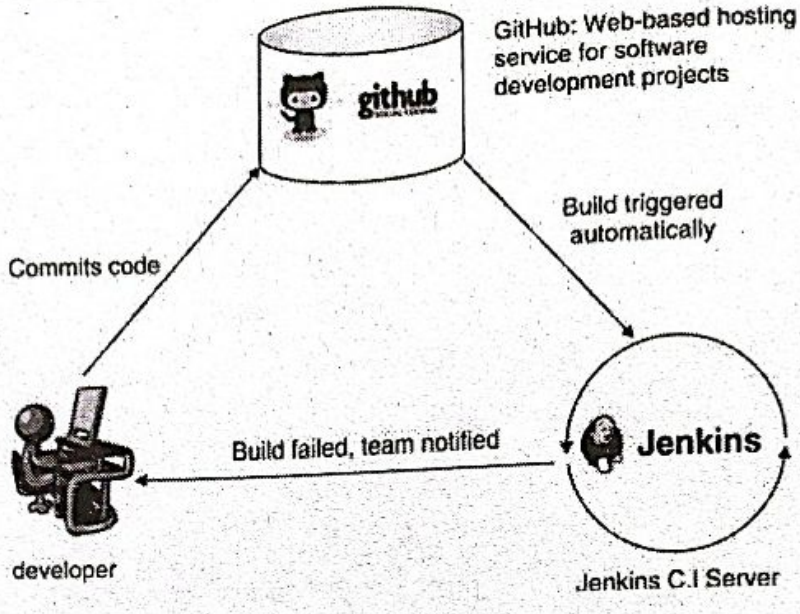


Fig.6.4.1

A] Advantages of using Jenkins

- **Open-Source :** It's free, so any organization can get started with it regardless of budgetary constraints. It's also easy to install: Jenkins was developed as a self-contained Java program, which means it can run on most devices and operating systems.
- **Community Support :** Because it is open-source and has been the go-to CI tool for many years, Jenkins now has a strong community behind it. This reinforces Jenkins' popularity, as it's easy to find tutorials and other resources to get the most out of it.
- **Plugins :** One of the main benefits of Jenkins is the huge catalog of plugins that can extend its functionality. At latest count, Jenkins has more than 1,800 community-developed plugins that allow any organization to adapt or enhance the base functionality to suit their own DevOps environment. Alongside the plugins, Jenkins is also easy to modify and extend to suit most organizational needs.
- **Distributed :** One server is enough for complex projects, so Jenkins uses a Master-Slave architecture to manage distributed builds. The main server is the 'Master', which can then distribute workload to other 'slave' servers, allowing for multiple builds and test environments to be running simultaneously. This approach could be used, for example, to build and test code on different operating systems.

B] How to create a CI pipeline in Jenkins

Jenkins allows ypu to specify pipelines using a Jenkinsfile. This is just a textfile that contains the necessary data for Jenkins to execute the pipeline. It is called Jenkinsfile and should be placed in the root of your project.

1. Declarative Pipeline :

- Jenkins pipeline can be written in a declarative manner with the "Jenkins syntax". Jenkins syntax is a Groovy DSL (Domain Specific Language) which is used to write stages of the pipeline. The DSL internally gets converted into XML to make the Jenkins pipeline. A "stage" in a pipeline is a group of steps to be executed in the pipeline.
- Let us consider a stage – "git checkout" that can clone the git repo (step) and tag it with a version (step).

Sample Jenkins scripted pipeline

```
pipeline {
agent any
stages {
stage('Git Checkout'){
steps {
echo 'Cloning repository'
sh'git clone https://github.com/sample.git'
}
}
stage('Build'){
steps {
echo 'Building project'
sh'gradle clean build'
}
}
}
}
```

2. Jenkins Job Builder

JJB (Jenkins job builder) is a tool to create a pipeline using YAML configuration. The Jenkins pipeline can be written in YAML and with the help of the JJB tool, it converts the YAML configuration in XML format and pushes it into Jenkins to create pipelines. It is comparatively easy to write YAML configuration than writing full-fleshed code in Jenkins syntax.

6.5 Continuous Delivery : Jenkins

- In Continuous Integration, every code commit is built and tested, but, is not in a condition to be released. I mean the build application is not automatically deployed on the test servers in order to validate it using different types of Blackbox testing like - User Acceptance Testing (UAT).
- In Continuous Delivery, the application is continuously deployed on the test servers for UAT. Or, you can say the application is ready to be released to production anytime. So, obviously Continuous Integration is necessary for Continuous Delivery.
- Jenkins provides good support for providing continuous deployment and delivery
- we took Continuous Integration to the next step i.e. Continuous Delivery and introduced a couple of simple, automated Acceptance Tests that proved that the application ran and could perform its most fundamental function. The majority of the tests running during the Acceptance Test stage are Functional Acceptance Tests.
- Basically, we built a Continuous Delivery pipeline, in order to make sure that the application is seamlessly deployed on the production environment, by making sure that the application works fine when deployed on the test server which is a replica of the production server.

6.6 Continuous Delivery : Ansible

- Ansible is an open source automation and orchestration tool for software provisioning, configuration management, and software deployment. Ansible can easily run and configure Unix-like systems as well as Windows systems to provide infrastructure as code. It contains its own declarative programming language for system configuration and management.
- Ansible aims to provide large productivity gains to a wide variety of automation challenges. While Ansible provides more productive drop-in replacements for many core capabilities in other automation solutions, it also seeks to solve other major IT challenges. One of these challenges is to enable continuous integration and continuous deployment (CI/CD) with zero downtime. This goal has often required extensive custom coding, working with multiple software packages, and lots of in-house-developed glue to achieve success. Ansible provides all of these capabilities in one composition, being designed from the beginning to orchestrate exactly these types of scenarios.
- Ansible prepares you to adapt to future technology advances and trends, including popular and emerging CI/CD tools like AzureDevOps, GitHub Actions, UrbanCode, TeamCity, and ServiceNow. For example, many companies are adopting cloud-native Kubernetes environments to gain more flexibility, speed, and innovation.

A] How does Ansible work ?

- Ansible works by connecting to your nodes and pushing out small programs—called modules—to these nodes. Modules are used to accomplish automation tasks in Ansible. These programs are written to be resource models of the desired state of the system. Ansible then executes these modules and removes them when finished.
- Without modules, you'd have to rely on ad-hoc commands and scripting to accomplish tasks. Ansible is agentless, which means the nodes it manages do not require any software to be installed on them. Ansible reads information about which machines you want to manage from your inventory. Ansible has a default inventory file, but you can create your own and define which servers you want to be managed.
- Ansible uses SSH protocol to connect to servers and run tasks. By default, Ansible uses SSH keys with ssh-agent and connects to remote machines using your current user name. Root logins are not required. You can log in as any user, and then su or sudo to any user.
- Once it has connected, Ansible transfers the modules required by your command or playbook to the remote machine(s) for execution. Ansible uses human-readable YAML templates so users can program repetitive tasks to happen automatically without having to learn an advanced programming language.
- Ansible contains built-in modules that you can use to automate tasks, or you can write your own. Ansible modules can be written in any language that can return JSON, such as Ruby, Python, or bash. Windows automation modules are even written in Powershell.

B] Ansible use cases

Use case 1 : Provisioning

Infrastructure provisioning is the first step in automating the operational life cycles of your applications. Red Hat Ansible Automation Platform can provision popular cloud platforms, virtualized hosts and hypervisors, network devices, and bare-metal servers. After bootstrapping, you can connect nodes to storage, add them to a load balancer, apply security patches, or perform many other operational tasks.

Use case 2: Configuration management

Configuration management is essential for maintaining consistency, efficiency, and security within your environment. Ansible allows you to manage your infrastructure by defining sets of desired-state descriptions. No matter what state a system is in, Ansible understands how to transform it to the desired state, allowing you to reliably and repeatably configure your IT infrastructure.

Use case 3: Application deployment

Applications must be properly configured and deployed to be useful. Ansible allows you to deploy multitier applications reliably, consistently, and simply. Using one common system, you can configure needed applications services and push application artifacts.

Use case 4: Continuous deployment

A subset of application deployment, continuous deployment pipelines help you release new software features and updates more frequently to support modern business demands. Ansible provides the multitier, multistep application orchestration needed for fast, reliable deployment of new features, bug fixes, and code changes, while reducing the need for human intervention throughout the release process.

6.7 Continuous Testing : Selenium

- **Continuous Testing** in DevOps is a software testing type that involves testing the software at every stage of the software development life cycle. The goal of Continuous testing is evaluating the quality of software at every step of the Continuous Delivery Process by testing early and testing often.
- The Continuous Testing process in DevOps involves stakeholders like Developer, DevOps, QA and Operational system.

How is Continuous Testing different ?

- The old way of testing was hand off centric. The software was handed off from one team to another. A project would have definite Development and QA phases. QA teams always wanted more time to ensure quality. The goal was that the quality should prevail over project schedule.
- However, business wants faster delivery of software to the end user. The newer is the software, the better it can be marketed and increase revenue potential of the company. Hence, a new way of testing was evolved.
- Continuous means undisrupted testing done on a continuous basis. In a Continuous DevOps process, a software change (release candidate) is continuously moving from Development to Testing to Deployment.
- The code is continuously developed, delivered, tested and deployed. For Example, whenever a developer checks the code in the Source Code Server like Jenkins automated set of unit tests are executed in the continuous process. If the tests fail, the build is rejected, and the developer is notified. If the build passes the test, it is deployed to performance, QA servers for exhaustive functional and load tests. The tests are run in parallel. If the tests pass, the software is deployed in production.

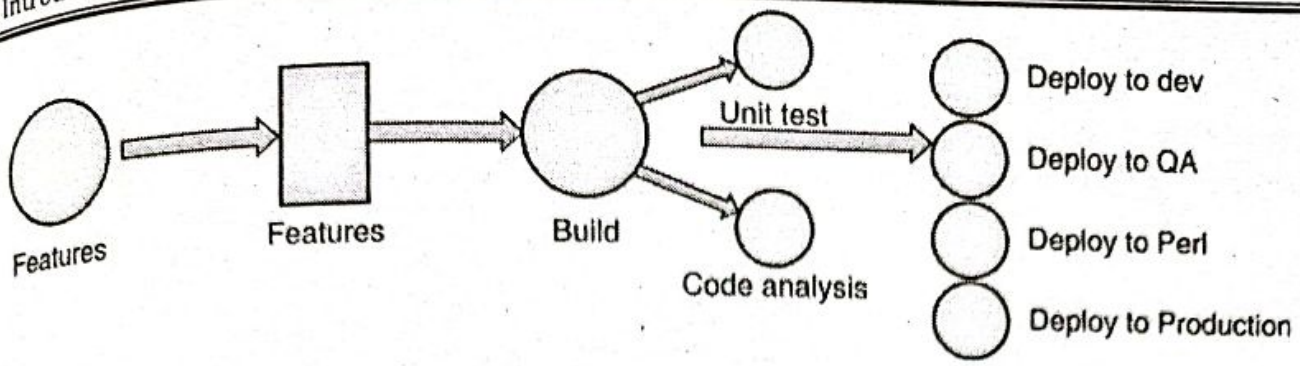


Fig. 6.7.1 : Continuous Testing

Selenium

- Selenium is open-source software testing tool. It supports all the leading browsers like Firefox, Chrome, IE, and Safari. Selenium WebDriver is used to automate web application testing.
- Selenium is an umbrella project for a range of tools and libraries that enable and support the automation of web browsers.
- It provides extensions to emulate user interaction with browsers, a distribution server for scaling browser allocation, and the infrastructure for implementations of the W3C WebDriver specification that lets you write interchangeable code for all major web browsers.
- This project is made possible by volunteer contributors who have put in thousands of hours of their own time, and made the source code freely available for anyone to use, enjoy, and improve.
- Selenium brings together browser vendors, engineers, and enthusiasts to further an open discussion around automation of the web platform.

6.8 Monitoring: Prometheus

- Monitoring applications is essential for ensuring their performance and availability. Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud. and released as an open-source project in 2016. It collects metrics from configured targets at given intervals, stores them in a time-series database, and provides various ways to examine and analyze them. Prometheus offers many features for DevOps teams, including :
  - A powerful query language that allows you to collect, process, and analyze data
  - Multiple storage back ends, including a time-series database, so you can store your data long term

- Built-in visualization tools so you can easily view your data
- Integration with other popular DevOps teams tools like Puppet and Chef

- Prometheus is widely popular in organizations for all sizes for monitoring applications, networks, servers, and other infrastructure components

A] Instrumenting Applications With Prometheus

- Prometheus offers a variety of techniques for instrumenting your applications so that they can be monitored. The most common way to do this is to use the exporter libraries included with Prometheus. But there are exporters for many popular programming languages and frameworks, including Java, Python, NodeJS, and Ruby on Rails. These libraries allow you to export metrics from your application into Prometheus format. And make life easier for DevOps teams.
- If your application is not already in operation, you can use the Prometheus client libraries to do so. The Prometheus client libraries allow you to write custom collectors and exporters for your applications. This gives you the maximum flexibility in how you want to monitor your applications.

B] How does it Work ?

- Prometheus requires an exposed HTTP endpoint to obtain metrics. Prometheus can begin scraping numerical data, capturing it as a time series, and storing it in a local database designed for time-series data as an endpoint is ready. Remote storage repositories can also be coupled with Prometheus.
- Prometheus was created to help software developers and administrators manage production computer systems, such as the applications, tools, databases, and networks that support popular websites.
- Users can construct temporary time series from the source using queries. Metric names and labels are used to define this series. Queries are made in PromQL, a one-of-a-kind programming language that enables users to select and aggregate time-series data in real-time. PromQL can also be used to create alert conditions that will send messages to external systems such as email, PagerDuty, or Slack.
- In its web-based user interface, Prometheus can present collected data in tabular or graph form. APIs can also be used to connect to third-party visualization tools like Grafana.

C] Prometheus Architecture

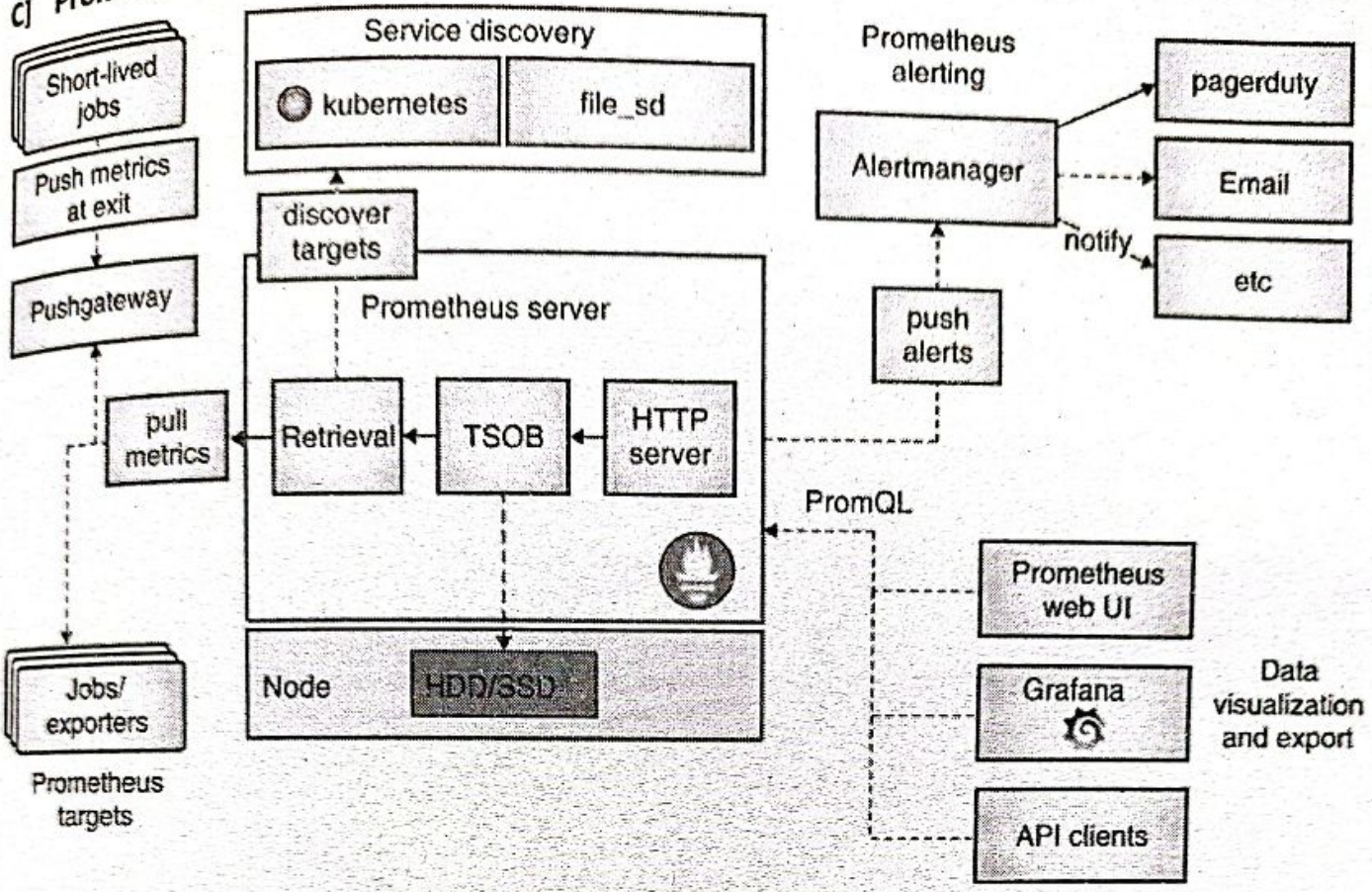


Fig.6.8.1

- **Prometheus server :** It is the core component of Prometheus consists of three components .
  - **Data retriever :** helps us to retrieve the data from the application endpoint .
  - **Time series database :** stores the metrics data so later they can be retrieved and analyzed .
  - **HTTP server :** accepts PromQL query language push the collected data to the dashboard for visualization purpose
- **Push gateway :** Push all the collected data to the database but only for short-lived jobs. This gateway API is useful for one-off jobs that run, capture the data, transform that data into the Prometheus data format and then push that data into the Prometheus server.
- **Exporters :** It is a script needed for configuration like exposing the /metrics endpoint. Prometheus Targets define how does prometheus extract the metrics from the different sources.Exporters are modules that extract information and translate it into the Prometheus format, which the server can then ingest.
- **Client Libraries :** Provides language support for monitoring different applications.

- **Alert Manager :** Send alerts in case of misconfiguration to different channels like slack, email, notification It has advanced features for deduplicating, grouping, and routing alerts and can route through other services like PagerDuty and OpsGenie.
- **The web UI :** It allows you to access, visualize, and chart the stored data. Prometheus provides its own UI, but you can also configure other visualization tools, like Grafana, to access the Prometheus server using PromQL (the Prometheus Query Language).

6.9 Bug tracking tool: Jira

- Bug and issue tracking tools help software teams find, record, and track bugs in their software. It's critical that everyone on your team is able to find and record bugs, and even more important, assign them to the right team members at the right time.
- A great bug and issue tracking tool gives your team a single view of all items in the backlog, regardless of whether the work item is a bug or a task related to new feature development. Having a single source of truth of all issue types helps teams prioritize against their big picture goals, while continually delivering value to their customers.

A] Why Jira Software for bug tracking ?

- Quickly capture, assign and prioritize bugs with Jira Software and track all aspects of the software development cycle. Jira's powerful workflow engine provides a clear view of a bug's status, and automation keeps you in the know with notifications as issues transition from backlog to done. Jira Software is the connective tissue for your software team, giving you full visibility and control of your products' end-to-end development.
- **Capture and track bugs in your software :** Capture bugs anywhere in your software projects with Jira Software. Once you've identified a bug, create an issue and add all relevant details, including descriptions, severity level, screenshots, version, and more. Issues can represent anything from a software bug, a project task to a leave request form, and each unique issue type can have its own custom workflow.
- **Assign and prioritize with ease :** Once bugs are captured, prioritize them based on the level of importance, urgency, and the team's capacity. Assign bugs with a few keystrokes and prioritize them by dragging and dropping them in your team's backlog or to-do column. By having a single source of truth, you can keep everyone on the same page and ensure the team is working on the highest priority items first.
- **Track bugs from backlog to done :** Always stay in the know by tracking bugs and issues through your team's workflow. Jira Software has a powerful workflow engine, planning features, comprehensive search, and reporting capabilities designed to help you find, record, and track bugs in your software.

- **Stay up to date with notifications :** Ensure the right people are notified at the right time. New bugs can be routed to the correct team member immediately. Notifications within Jira are not just @mentions, but can also be configured to update someone automatically when a status change occurs. Customize your project, so as things change, Jira Software will keep everyone in sync.

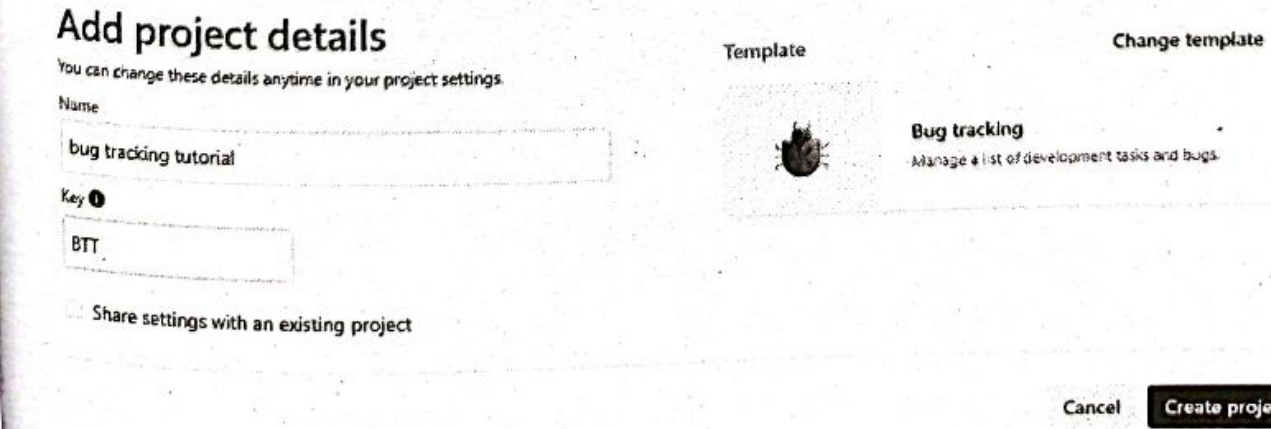
B] Best practices for bug tracking in Jira Software

- Empower your team with the right information: Ensure bugs are well documented, so engineers have the specifics needed to diagnose or fix the bug. To capture key details faster, create custom fields in Jira.
- Quickly assign and prioritize: Automatically assign bugs to individuals based priority and send notifications with automation in Jira.
- Ensure bugs are re-solved on time: Create and customize a workflow specifically for bugs so the team can focus on managing and resolving them efficiently and effectively.
- Take your bug resolution process to the next level: Reduce manual tasks for your team and let Jira automation inform any watchers of fixes, new releases, and more.
- Integrate Jira with your dev tools: Save time by passing information directly to engineers, and gain visibility into your development pipeline right from Jira.

C] Jira Bug Tracking

Setting up a bug tracking project in Jira Software is relatively simple.

1. Log in to your organization's workspace. The URL looks something like this : your-organization.atlassian.net/jira.
2. In the toolbar at the top, click "Projects", then "Create Project".
3. Under "Project Templates", choose "Software Development", then "Bug Tracking".
4. Click "Use Template", then choose a name and a key.



5. Click "Create Project", and you're good to go

- This project is now your dedicated bug tracker. This is where your QA team and clients will report and track bugs and issues going forward—for the entire software development lifecycle.
- With this setup, you can already create issues.
- The default screen includes:
  - Issue Type
  - Summary
  - and Description.

6.10 ELK Stack

- The ELK stack is an acronym used to describe a collection of three open-source projects - Elasticsearch, Logstash, and Kibana. Elastic search is a full-text search and analytics engine. Logstash is a log aggregator that collects and processes data from multiple sources, converts, and ships it to various destinations, such as Elasticsearch. And finally, Kibana provides a user interface, allowing users to visualize, query, and analyze their data via graphs and charts.

- The core components of the ELK Stack were :
  - **Elasticsearch :** The core component of ELK. It works as a searchable database for log files. Elasticsearch is a NoSQL database which is based on Lucene search engine and is built with RESTful APIs. It is a highly flexible and distributed search and analytics engine. Also, it provides simple deployment, maximum reliability, and easy management through horizontal scalability. It provides advanced queries to perform detailed analysis and stores all the data centrally for quick search of the documents.
  - **Logstash :** Logstash is the data collection pipeline tool. It the first component of ELK Stack which collects data inputs and feeds it to the Elasticsearch. It collects various types of data from different sources, all at once and makes it available immediately for further use. i.e.It can be configured to retrieve data from many different sources and then to send to Elasticsearch.
  - **Kibana :** Kibana is a data visualization tool. It is used for visualizing the Elasticsearch documents and helps the developers to have an immediate insight into it. Kibana dashboard provides various interactive diagrams, geospatial data, timelines, and graphs to visualize the complex queries done using Elasticsearch. Using Kibana you can create and save custom graphs according to your specific needs.
- Thus, ELK is a log management platform that works by enabling you to gather massive amounts of log data from anywhere across your infrastructure into a single place, then search, analyze and visualize it in real time. Among the most common ELK use cases, we can name monitoring, troubleshooting, web analytics, risk management, business intelligence, compliance, fraud detection and security analysis.

A] ELK Stack Architecture

Logstash processes the application log files based on the filter criteria we set and sends those logs to Elasticsearch. Through Kibana, we view and analyze those logs when required.

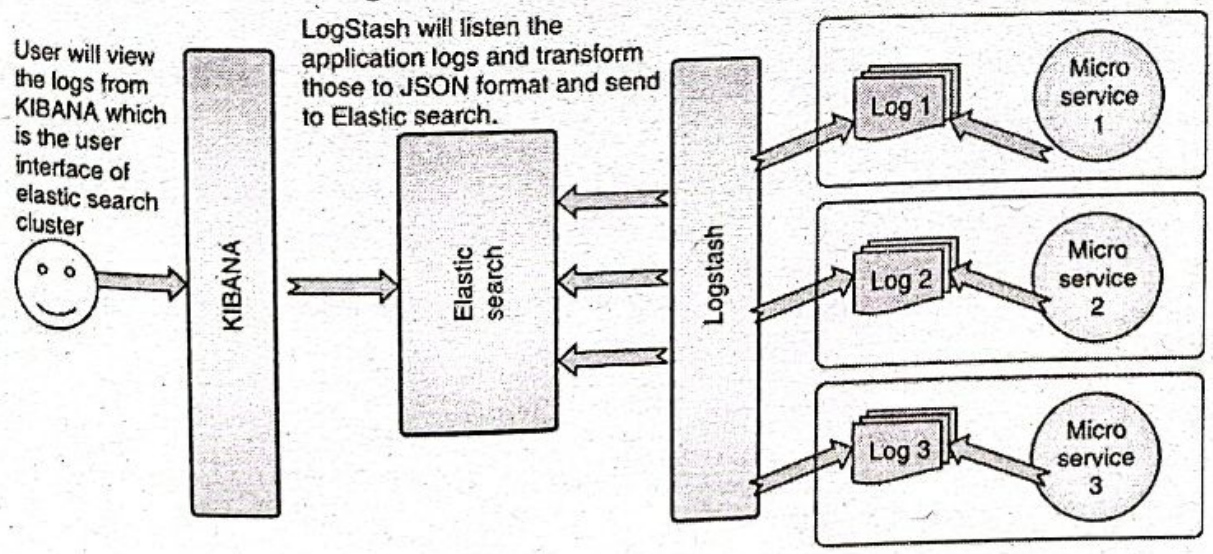


Fig. 6.10.1

Advantages of ELK stack

- ELK works best when logs from various Apps of an enterprise converge into a single ELK instance
- It provides amazing insights for this single instance and also eliminates the need to log into hundred different log data sources
- Rapid on-premise installation
- Easy to deploy Scales vertically and horizontally
- Elastic offers a host of language clients which includes Ruby. Python. PHP, Perl, .NET, Java, and JavaScript, and more
- Availability of libraries for different programming and scripting languages

6.11 Case study

6.11.1 Spotify : Using Docker

Spotify - the music streaming service that has taken the world by storm. You may not know that Spotify is a prime example of how effective DevOps can be in improving developer productivity. Spotify achieved massive growth quickly due partly to its use of DevOps principles.

1) **Spotify placed a strong emphasis on automation**

They used various automation tools to help manage their large user base and keep things running smoothly. Spotify automated the whole process of building data pipelines, backend services, and websites. That helped free their developers to focus on new features and improvements rather than fire-fighting issues.

2) **They use the DevOps tool to experiment quickly and safely**

Spotify has a high-speed release cycle. They use various DevOps tools to help them experiment quickly and safely. These tools help them to automatically roll back changes if something goes wrong. Their developers can try out new features quickly and easily without worrying about breaking things.

3) They automated the project setup process by developing tools such as a test-certification program, Tingle CI/CD, Golden path, and Backstage.

With the help of these tools, they achieved the following :

- Increased developer productivity by automating the project setup process
- Improved quality of their codebase by implementing a strict test-certification program
- Reduced cycle time from idea to production by using CI/CD pipelines

The **Golden Path tool** helps reduce the decision-making required to build backend services, application features, data products etc.

Backstage allows engineers to spend less than 5 minutes setting up Skeleton for a new website, which used to take more than 14 days.

The CI/CD system developed by Spotify called Tingle automatically builds the project when the engineer commits it to GitHub and automatically runs all the tests. This reduced the cycle time from idea to production by more than 50%.

6.11.2 Bank of New Zealand

- BNZ started its modernisation journey by moving two of its staff assisted applications—Cash Management Online and Loyalty Loader (Flybuys) to Azure. These applications are now leveraging Azure cloud-native services & features including – Application identity & access management with Azure Active Directory, AI-enabled analytics with applications insights and improved monitoring. BNZ Developers & Operations teams now have rich analytics and monitoring data to optimise the application for its performance and better customer experience. BNZ is experiencing a significant improvement in the Recovery Time Objective (RTO) and Recovery Point Objective (RPO) of the applications with a fully automated failover.
- BNZ also automated the deployment of continuous integration and continuous delivery (CI/CD) pipeline for the databases. "In the past, the manual deployment for database took up to two hours, and today by using Azure Pipelines under DevOps, BNZ able to straight away cut it down to five minutes. "It's not just that our CI/CD pipeline became faster, our security scanning and data classification also became faster.
- In order to document its applications, infrastructure, data and integration, BNZ has been using seven different platforms. These include SharePoint for technical documentation, Confluence for wiki documentation, Team Foundation Server for storing code, and yet another server & tools for housing database scripts. When it moved to Azure, it was able to consolidate these different tools and platforms under one roof.BNZ brought everything under Azure DevOps in the Git repository, which is very convenient and useful

6.11.3 EtSy

- Back in 2005, Etsy's engineering teams were siloed into developers, operations teams, and database admins. Although the team was relatively small — close to 35 employees — they faced many team collaboration challenges. This barrier was hindering Etsy's progress as an organization.

- In 2008, Etsy's engineers realized the cons of monolithic architecture and the waterfall business model :
  - Frequent file changes
  - Deployment inconsistency
  - Lack of confidence in deployment
  - Increased deployment time
  - Less flexibility to experiment and iterate
- Following this, they felt the need to change their software development approach.
- Etsy focused on making a cultural shift towards DevOps where their teams could collaborate and synchronize all their tasks in real time. It was the early vision and inclination towards the change that helped Etsy survive a massive DevOps culture shift.
- As one of the earliest adopters of DevOps, Etsy's road to success was a bumpy ride. Let's discuss some of the lessons that Etsy learned during its DevOps journey.

Lessons to Learn from Etsy's DevOps Strategy

1. Changing business culture is not a smooth process. You need to deal with stakeholders at different levels who might have a different opinion about this altogether. The whole process might involve demonstrating the positive aspects of the cultural shift.
2. Have an honest assessment of where your codebase and team are relative to the deployments and how you want them to proceed towards deployment.
3. This might come as a surprise, but take the initial steps as slowly as possible. Concentrate on preparing small chunks of code. Being slow and able to iterate is the way to move ahead.
4. Focus on the quality of the code instead of the stability of the deployment platform. You can address the above goal by deploying smaller chunks of code more often. Having smaller chunks to deploy makes it easier to verify code quality.
5. Leverage automation to its full potential. Automation grants you greater accuracy, reliability, consistency, and speed.
6. Try to evaluate your business and deployment process via visualizations. It will help you in making informed business decisions.
7. Focus on reinforcing value-based learning and collaboration between teams to bring cultural and transformational changes within the organization.