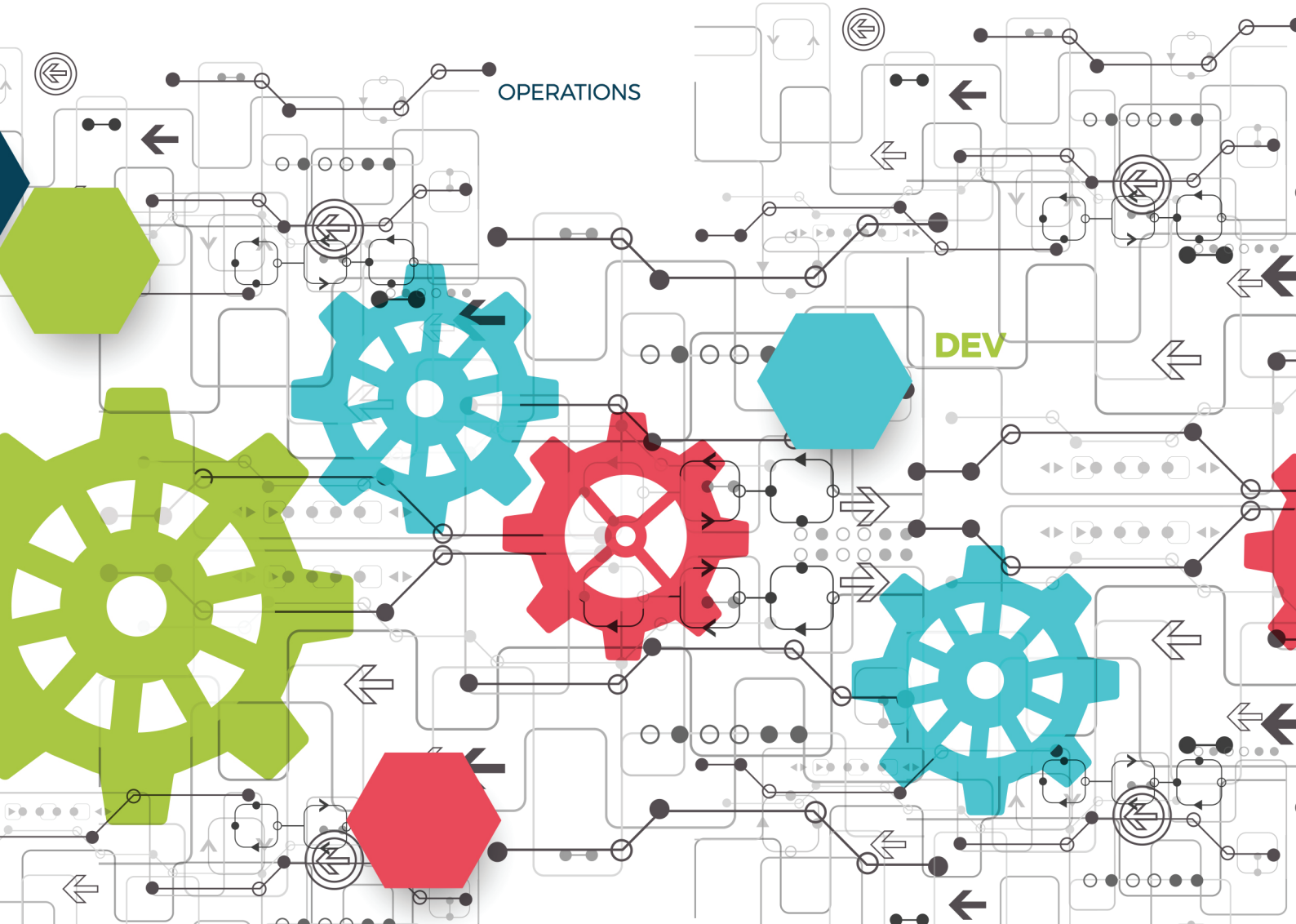
B.Tech Computer Science and Engineering in DevOps

Continuous Integration and Continuous Delivery

MODULE 3

Anatomy of a Continuous Delivery Pipeline



Contents

MODULE 3

Anatomy of a Continuous Delivery Pipeline

Facilitator Notes:

Provide the brief overview about the module to the participants.

Module Objectives

At the end of this module, you will be able to:

- Explain how to release an application to production
- Understand CD engineering practices
- Describe Continuous Development and Integration
- Explain what is continuous testing and promotion of builds
- Understand Continuous Deployment to successive environments before production
- Explain continuous monitoring for the delivery pipeline
- Understand the concept of continuous feedback

Facilitator Notes:

Explain the module objectives to the participants.

Module Topics

Let us take a quick look at the topics that we will cover in this modu

- 1.3.1 CD
  - 1.3.1.1 Simple Delivery Pipeline
  - 1.3.1.2 Continuous Deployment Pipeline
  - 1.3.1.3 Releasing an application to Production
  - 1.3.1.4 Zero-Downtime Releases
  - 1.3.1.5 Rolling back deployments
  - 1.3.1.6 Blue-Green Deployments
  - 1.3.1.7 Canary Releasing
  - 1.3.1.8 Emergency Fixes
- 1.3.2 Continuous Delivery engineering practices
- 1.3.3 Continuous Development/Integration
- 1.3.4 Continuous testing
  - 1.3.4.1 Deploying and Promoting your Application
  - 1.3.4.2 Modeling Your Release Process and Promoting Builds
- 1.3.5 Continuous Deployment to successive environments until before Production
- 1.3.6 Continuous monitoring for the delivery pipeline
  - 1.3.6.1 Nagios sampler report
- 1.3.7 Continuous feedback
  - 1.3.7.1 Continuous Feedback should be

- 1.3.7.2 Continuous Feedback should not be
- 1.3.7.3 Continuous Feedback rules

- 1.3.8 Compuware DevOps transformation case study

Facilitator Notes:

Inform the participants about the topics that they will be learning in this module.

1.3.1 CD

CD ensures

- Every successful build passed all the automated tests
- The build quality is not compromised
- Continuous Delivery enables deploying code to certain environments (such as test and QA)
- As part of CD every successful build that has passed all the tests and quality gates, can potentially be deployed into production

Facilitator Notes:

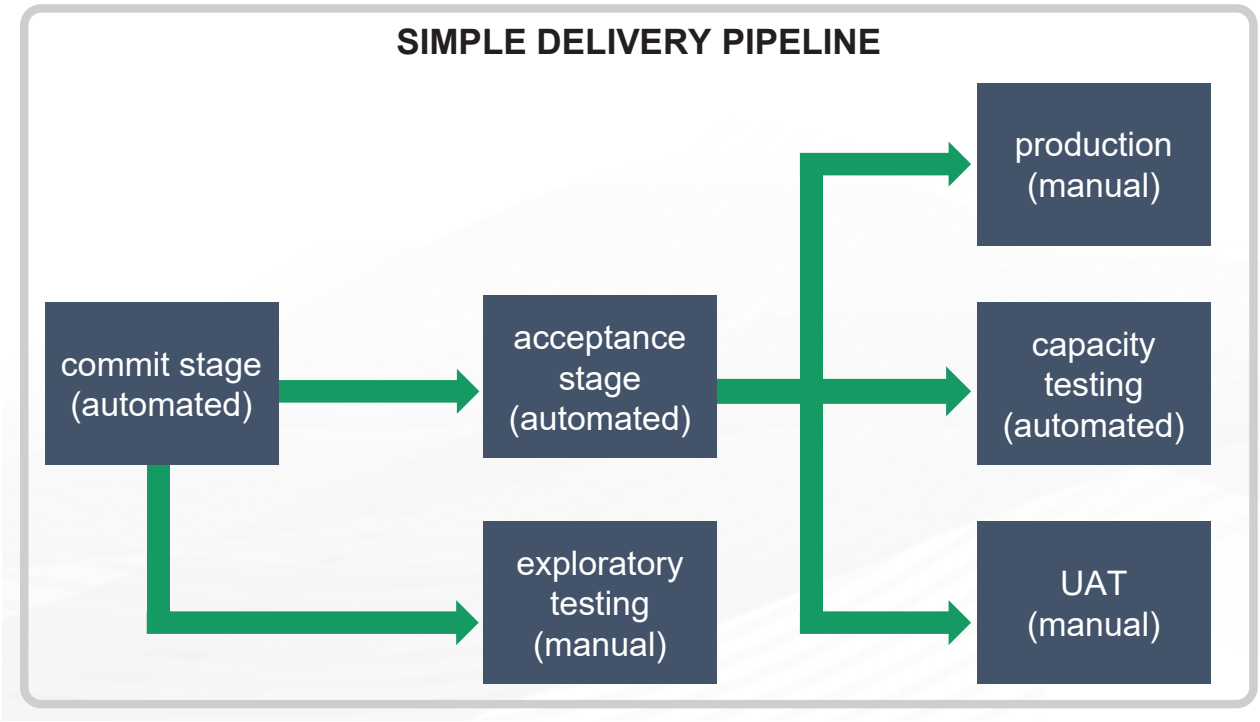
Give the participants a brief about continuous delivery.

- Implementing Continuous delivery involves more than just doing some automation work. It depends on effective collaboration between all the different teams involved in delivery, support, and on people on the ground willingness to make changes.

- Both IT and the business can try out the application at any time, perhaps to test some new feature, by self-servicing a deployment of the application to a UAT environment. For audit purposes, the pipeline provides a system of record as to exactly which versions of each application have been through which parts of the delivery process, and the ability to trace back from what's in every environment to the revision it came from in version control. Many of the tools in this space provide the facility to lock down who can do what, so that deployments can only be performed by authorized people.

- In particular incremental delivery and automation of the build, testing, and deployment process, are all designed to help manage the risk of releasing new versions of the software. Comprehensive test automation provides a high level of confidence in the quality of the application. Deployment automation provides the ability to release new changes and back out at the press of a button.
- Practices such as using the same process to deploy into every environment and automated environment, data, and infrastructure management are designed to ensure that the release process is thoroughly tested, the possibility for human error is minimized, and any problems—whether functional, nonfunctional, or configuration-related—are discovered well before release.

1.3.1.1 Simple Delivery Pipeline



Facilitator Notes:

Explain about the simple delivery pipeline to the Participants.

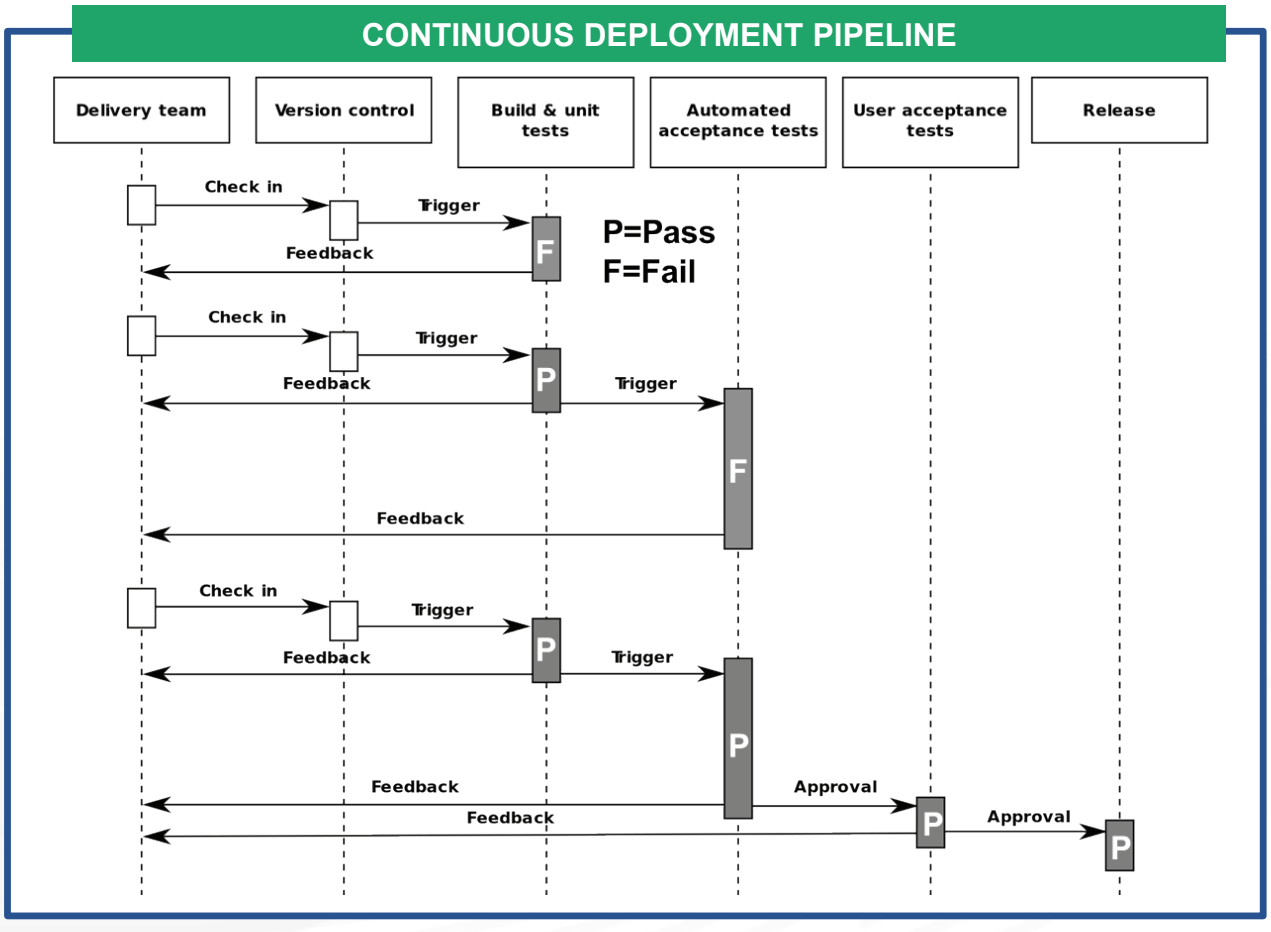
The above simple delivery pipeline consists of a commit stage, a testing stage(acceptance tests) and a deployment (release) stage.

These are some of the events that will need to be captured:

Commit Stage/Commit notifications, build status/build results, unit test results and code metrics for code quality.

The key to continuous delivery is a deployment pipeline. During the remainder of this module, we will use the general term of “Continuous Deployment” to refer to both Continuous Deployment and Continuous Delivery. Indeed, Continuous Delivery can be viewed as Continuous Deployment with the final step (deployment into production)

1.3.1.2 Continuous Deployment Pipeline



Facilitator Notes:

Give the participants a brief about continuous deployment pipeline.

Deployment pipeline:

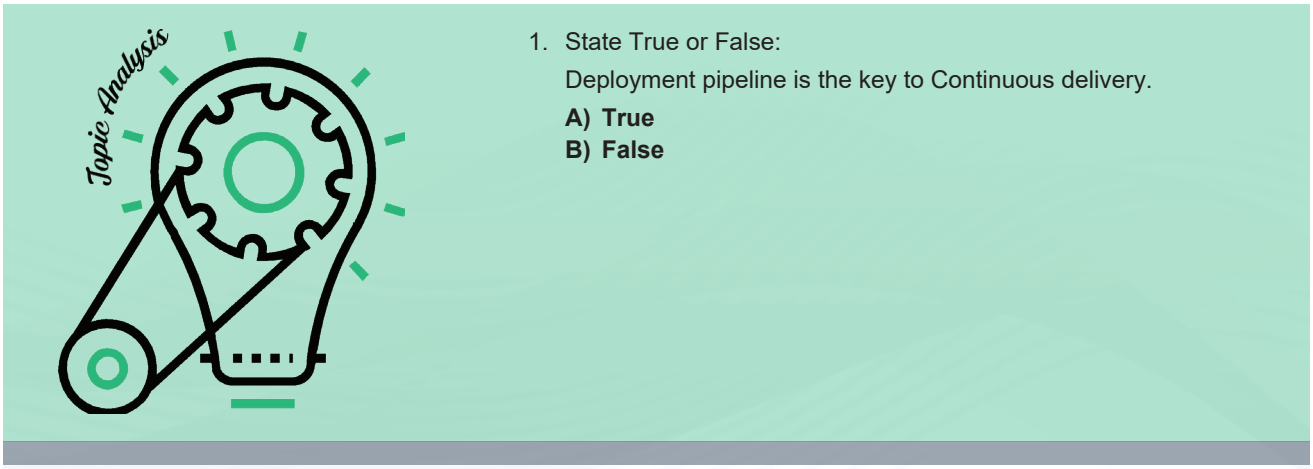
- Provides clear visibility in production readiness
- Enables self-service deployments
- Before releasing the product into production, most bugs are detected

A CD pipeline frequently and predictably delivers quality products in an automated fashion, depending on business needs, from testing to staging to production.

The various phases of the pipeline depend on the architecture of the product and provide more details on which artifacts are produced in each phase. Let's discuss ongoing delivery of the four common phases:

- Component phase (Code reviews, Unit tests, Static code analysis)
- Subsystem phase (Functional tests and negative test scenarios)
- System phase (Integration, performance and security tests)
- Production phase (Pipelines, zero downtime deployment)

What did You grasp?



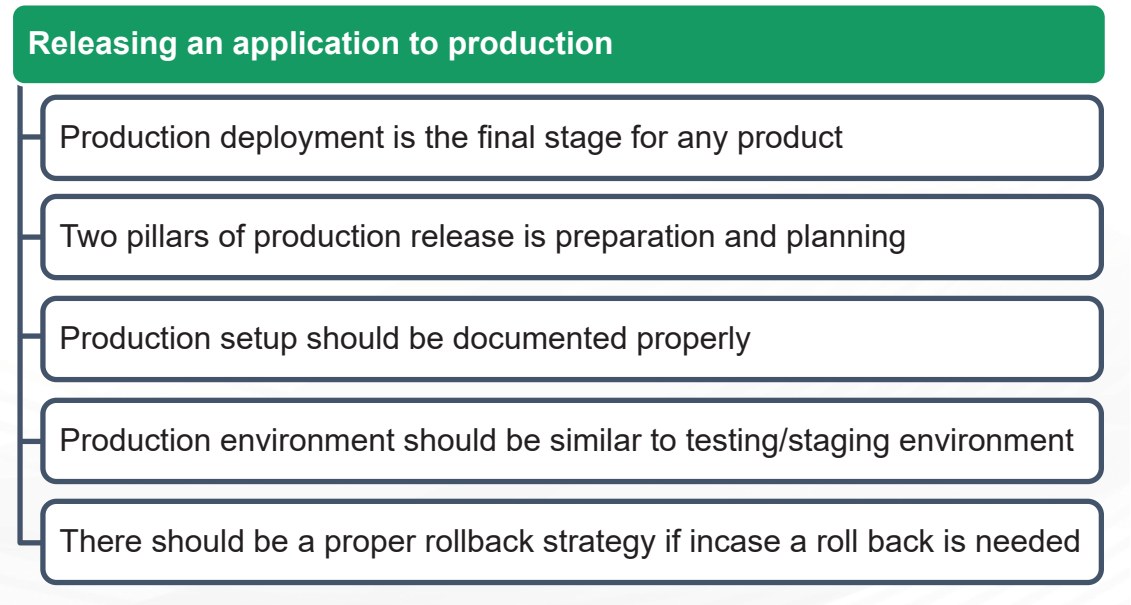
1. State True or False:
   Deployment pipeline is the key to Continuous delivery.
   A) True
   B) False

Facilitator Notes:

Ans:

1. A. True

1.3.1.3 Releasing an Application to Production



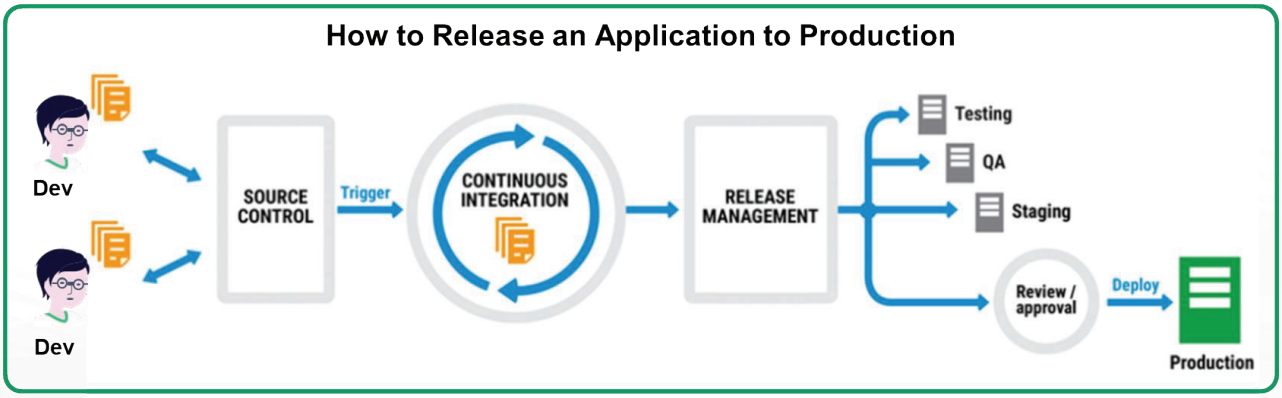
Facilitator Notes:

Give the participants a brief about releasing an application to production.

To release an application to production many stages are involved. Let us look in to the details:

- When releasing an application to production, the same process should be followed as for any other deployment like staging or UAT environments. Just few details like IP address and port number can be different. Trigger the automated deployment system to deploy the application to target environment without manual intervention. The same process should be used for all subsequent deployments and releases.

- The main difference between deploying and releasing is the ability to roll back, and we deal with this problem at length in coming slides. We also introduce two extremely powerful techniques that can be used to perform zero-downtime releases and rollbacks on even the largest of production systems: blue-green deployments and canary releasing.
- All of these processes-deploying to testing and production environments and rolling back-need to form part of your deployment pipeline implementation.



Facilitator Notes:

Give the participants a brief about releasing an application to production.

The most important part of releasing an application to production is:

- Creating a release strategy
- Having a release plan
- Releasing products

Creating a release strategy:

- The main difference between deploying and releasing is the ability to roll back, and we deal with this problem at length in coming slides.

- All of these processes—deploying to testing and production environments and rolling back—need to form part of your deployment pipeline implementation.

Having a release plan:

To create a successful product release plan, follow below basic steps:

Step 1: Define your vision

Step 2: Rank the product backlog

Step 3: Hold a release planning meeting

Step 4: Finalize and share product release calendar

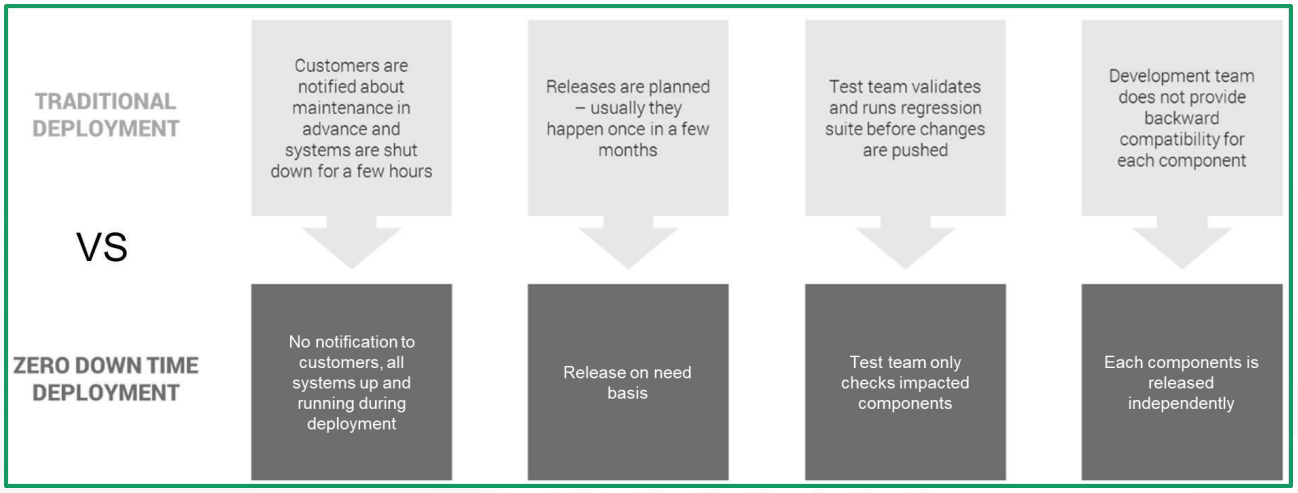
Releasing products:

The steps required to release the application for the first time:

- The steps required to backup and restore the application's state
- The steps required to upgrade the application without destroying the application's state
- The steps to restart or redeploy the application if it fails
- The methods of monitoring the application

Lets understand about Zero down time releases, rollback deployments and discuss two extremely powerful techniques that can be used to perform zero-downtime releases and rollbacks on even the largest of production systems: blue-green deployments and canary releasing. We will discuss about these techniques and how emergency fixes are delivered in coming slides.

1.3.1.4 Zero-Downtime Releases

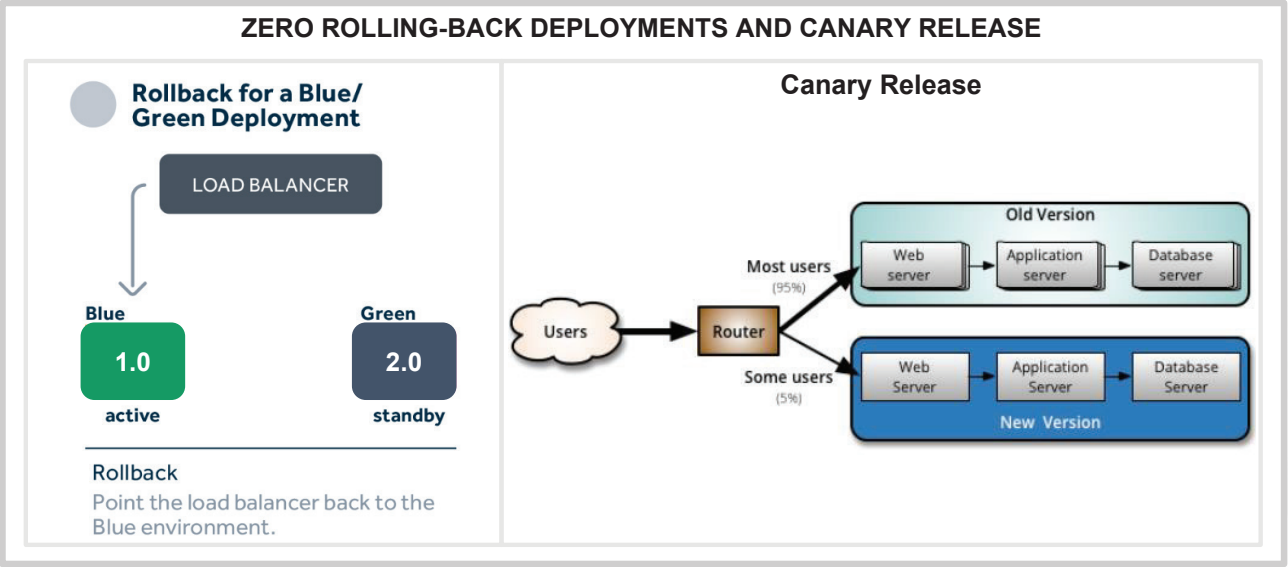


Facilitator Notes:

Give the participants a brief about Zero downtime releases.

- A zero-downtime release, also known as hot deployment, is one in which the actual process of switching users from one release to another happens nearly instantaneously. Crucially, it must also be possible to back users out to the previous version nearly instantaneously too, if something goes wrong.
- The key to zero-downtime releases is decoupling the various parts of the release process so they can happen independently as far as possible. In particular, it should be possible to put in place new versions of shared resources your applications depend on, such as databases, services, and static resources, before you upgrade your applications.

1.3.1.5 Rolling Back Deployments



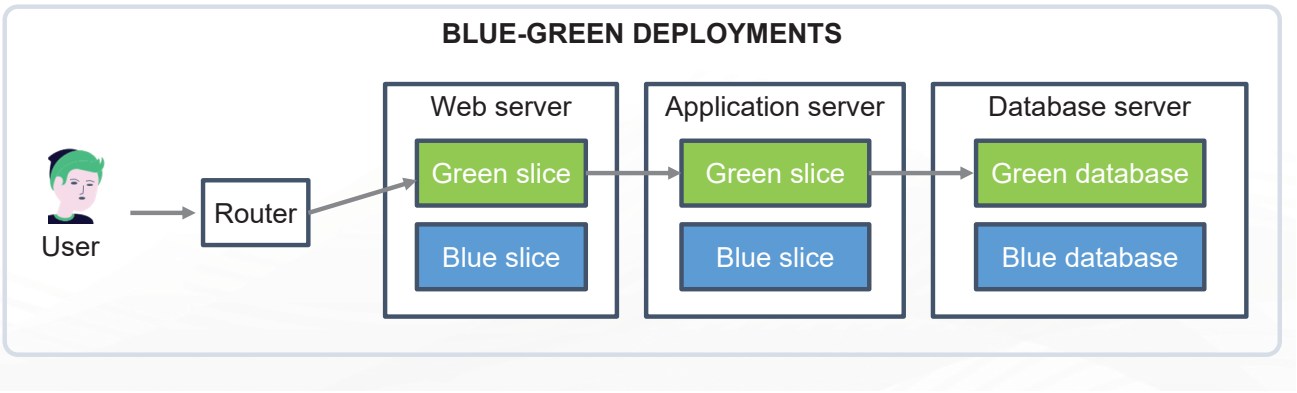
Facilitator Notes:

Give the participants a brief about rolling back deployments.

- It is essential to be able to rollback a deployment in case it goes wrong. Debugging problems in a running production environment is almost certain to result in late nights, mistakes with unfortunate consequences, and angry users. You need to have a way to restore service to your users when things go wrong, so you can debug the failure in the comfort of normal working hours. There are several methods of performing a rollback that we will discuss here. The more advanced techniques-blue-green deployments and canary releasing-can also be used to perform zero-downtime releases and rollbacks.

- Before we start, there are two important constraints. The first is your data. If your release process makes changes to your data, it can be hard to roll back. Another constraint is the other systems you integrate with. With releases involving more than one system (known as orchestrated releases), the rollback process becomes more complex too.

1.3.1.6 Blue-Green Deployments

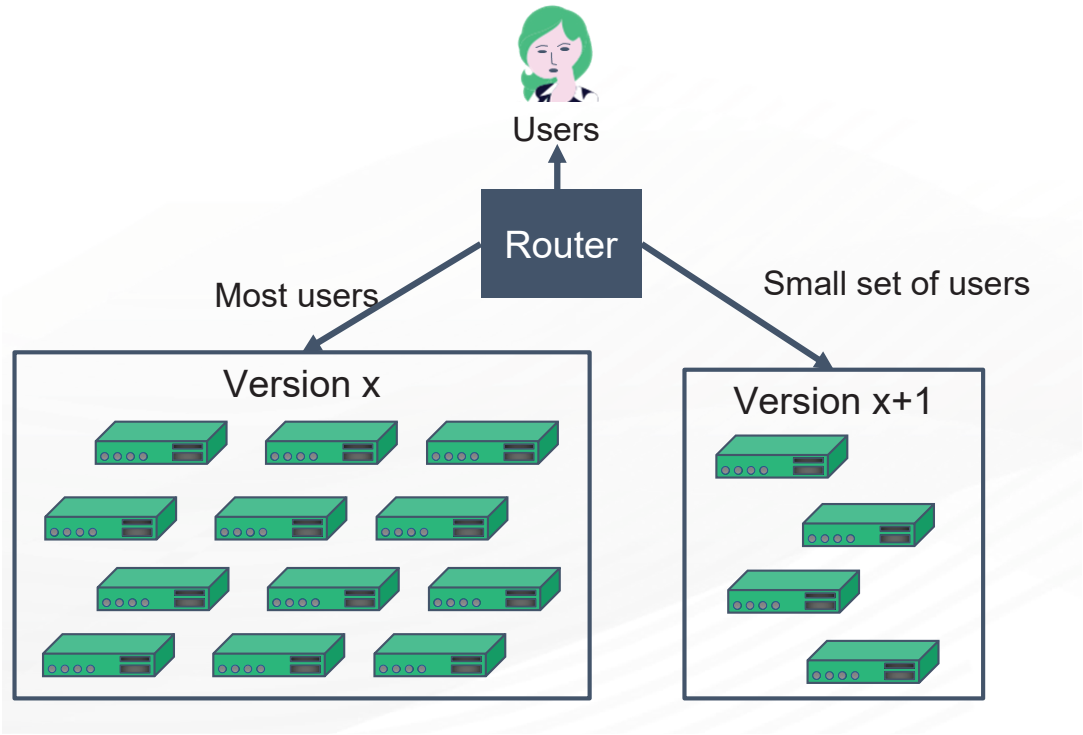


Facilitator Notes:

Give the participants a brief about Blue-Green Deployments

- Blue-Green is one of the most powerful release management techniques available. The concept is to keep the production environment in two identical versions, which is called as call blue and green deployment model.
- In the figure above, system users are routed to the green environment, which is the production that is currently being designated. We want a new version of the application to be released. It is deployed in the blue environment, allowing the application to start up. This in no way affects the green environment operation.
- To check that it works properly, we can run smoke tests against the blue environment. Moving to the newer version when everything is ready is as simple as changing the configuration of the router to point to the blue environment rather than the green environment. Therefore, the blue environment becomes production. It is typically possible to switch to a new version in much less than a second.
- We simply switch the router back to the green environment if something goes wrong. We can then debug on the blue environment what went wrong.

1.3.1.7 Canary Releasing



Facilitator Notes:

Give the participants a brief about Canary releasing.

- Canary releasing, as shown in the figure above, involves rolling out a new version of an application for quick feedback to a subset of production servers. This quickly uncovers any problems with the new version without affecting the majority of users like a canary in a coal mine. Canary release is the best way to reduce the risk of a new version being released.

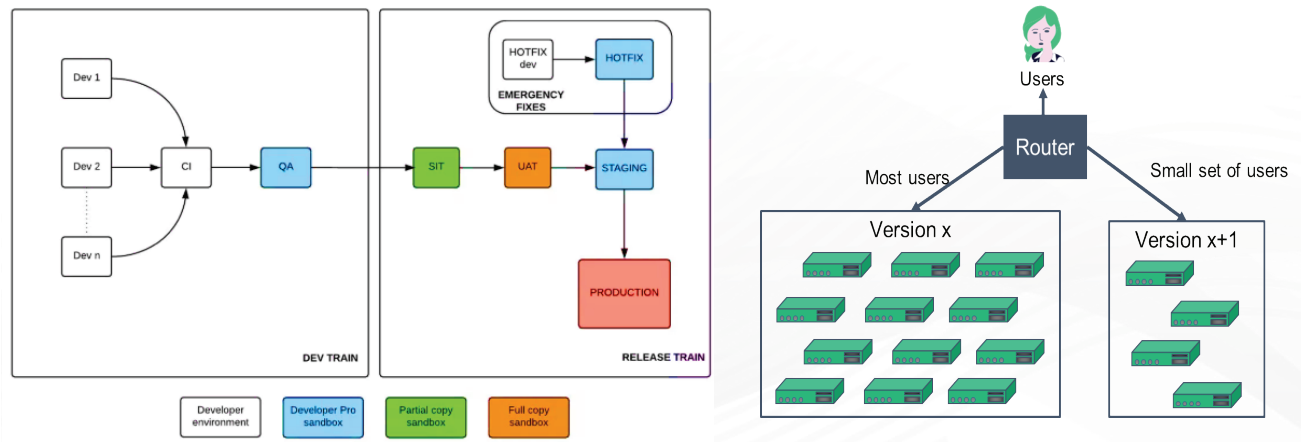
- Like blue-green deployments, you need to initially deploy the new version of the application to a set of servers where no users are routed to. You can then do smoke tests and, if desired, capacity tests, on the new version. Finally, you can start to route selected users to the new version of the application. Some companies select “power users” to hit the new version of the application first. You can even have multiple versions of your application in production at the same time, routing different groups of users to different versions as required.

- Canary releasing is not for everyone, though. It is harder to use it where the users have your software installed on their own computers. There is a solution to this problem (one used in grid computing)—enable your client software or desktop application to automatically update itself to a known-good version hosted by your servers.

- Canary releasing imposes further constraints on database upgrades (which also apply to other shared resources, such as shared session caches or external services): Any shared resource needs to work with all versions of the application you want to have in production. The alternative approach is to use a shared-nothing architecture where each node is truly independent of other nodes, with no shared database or services,3 or some hybrid of the two approaches.

1.3.1.8 Emergency Fixes

The following image displays Emergency fixes.



Facilitator Notes:

Give the participants a brief about Emergency bug fixes.

- In every system, there comes a moment when a critical defect is discovered and has to be fixed as soon as possible. In this situation, the most important thing to bear in mind is: Do not, under any circumstances, subvert your process. Emergency fixes should go through the same build, deploy, test, and release process as any other change. Why do we say this? Because we have seen so many occasions where fixes were made by logging directly into production environments and making uncontrolled changes.

- This has two unfortunate consequences. The first is that the change is not tested properly, which can lead to regressions and patches that do not fix the problem and may even exacerbate it. Secondly, the change is often not recorded (and even if it is, the second and third changes made to fix the problems you introduced with the first change do not get recorded). Because of this the environment ends up in an unknown state that makes it impossible to reproduce, and breaks further deployments in unmanageable ways.
- The moral of the story is: Run every emergency fix through your standard deployment pipeline. This is just one more reason to keep your cycle time low.
- Sometimes it is not actually worth fixing a defect through an emergency fix. You should always consider how many people the defect affects, how often it occurs, and how severe the defect is in terms of its impact on users. If the defect affects few people, occurs infrequently, and has a low impact, it may not make sense to fix it immediately if the risks associated with deploying a new version are relatively high. Of course this is a great argument for reducing the risks associated with deployment through effective configuration management and an automated deployment process.
- One alternative to making an emergency fix is to roll back to the previous known good version, as described earlier.
- Here are some considerations to take into account when dealing with a defect in production:
    - Never do them at late night, and always pair with somebody else.
    - Make sure the emergency fix process is tested.
    - Make sure the emergency fix is tested using your staging environment.
    - Sometimes it’s better to roll back to the previous version than to deploy a fix. Do some analysis to find out what is the best solution.

What did You Grasp?

1. Fill in the blank.
   Failed deployments can be rolled back easily using ……
   A) Backup Tools
   B) Blue-Green deployment
   C) Staging deployment
   D) Tape drives

2. State True or False.
   Canary releasing is rolling out new features to a set of users.
   A) True
   B) False

3. State True or False.
   Emergency fixes can be deployed into Production directly.
   A) True
   B) False

4. Which of the following options is the another name of zero downtime deployment?
   - **A) Hot deployment**
   - **B) Cold deployment**
   - **C) Manual deployment**
   - **D) Staging deployment**

Facilitator Notes:

Ans:

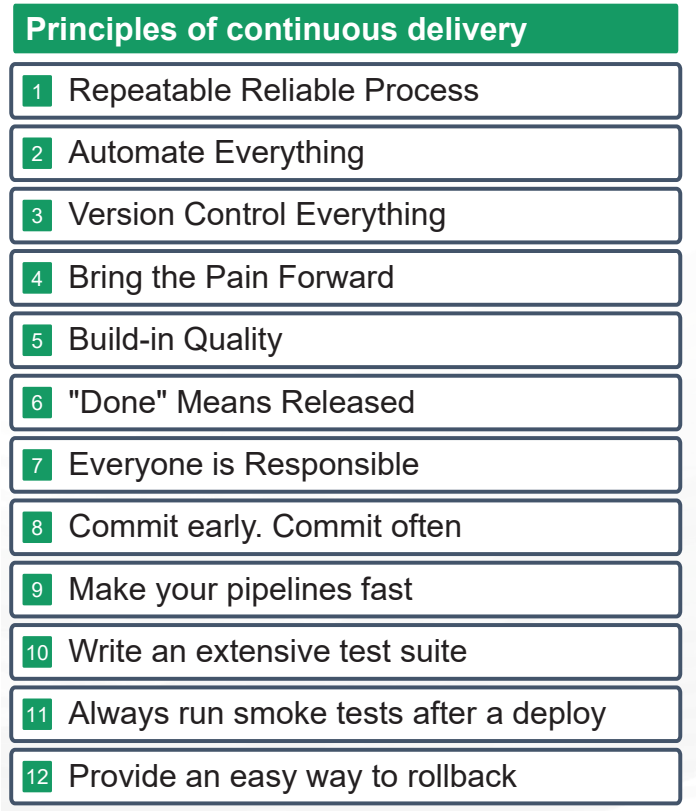
1. B. Blue/Green deployments

2. A. True

3. B. False

4. A. Hot deployment

1.3.2 Continuous Delivery Engineering Practices



Facilitator Notes:

Explain the participants about CD engineering practices.

Continuous delivery encourages the usage of an automated deployment pipeline to release the software into production reliably and quickly.

The goal of CD is to establish an optimized end-to-end process, lower the risk of release problems, enhance the development to production cycles, and provide a quicker time to market. The seven principles of continuous delivery is as follows:

1. **Repeatable Reliable Process**

Use the same release process in all environments.

2. **Automate Everything**

Automate your builds, your testing, your releases, your configuration changes and everything else.

3. Version Control Everything

Code, configuration, scripts, databases, documentation. Maintaining everything in one source that is a reliable one

4. Bring the Pain Forward

Deal with the hard stuff first. The tasks that are time-consuming or error prone should be dealt with as soon as possible.

5. Build-in Quality

Create feedback methods to deal with the bugs as soon as they are created. By routing the issues back to developers as soon as they fail post-build test, it will enable them to produce higher quality code quicker.

6. "Done" Means Released

- A feature is marked as completely done only when it is in production. Setting a clear definition of "done" criteria right from the beginning will help everyone communicate better, and realize the value in each feature.

7. Everyone is Responsible

- It works on development machine, is never a acceptable excuse. Responsibility should extend all the way to production. Cultural change can be the hardest to implement.

8. Commit early. Commit frequently.

- This is a fundamental principle to be able to implement CI/CD in your organization. We cannot ignore this.

9. Make your pipelines fast.

- This is a very important requirement for the team to be more productive and to enable fast turn around if incase of any issues. If a team has to wait an hour long for the pipeline to finish just to be able to deploy a one line change they will get frustrated very quickly.

10. Write an extensive test suite (unit and integration tests).

- This is one of the hardest parts to get right apart from organizational changes. Your team needs to develop a culture of writing tests for each code change that they integrate into the master branch in order to be successful with CI / CD. This includes creating the unit tests, regression, integration and smoke.

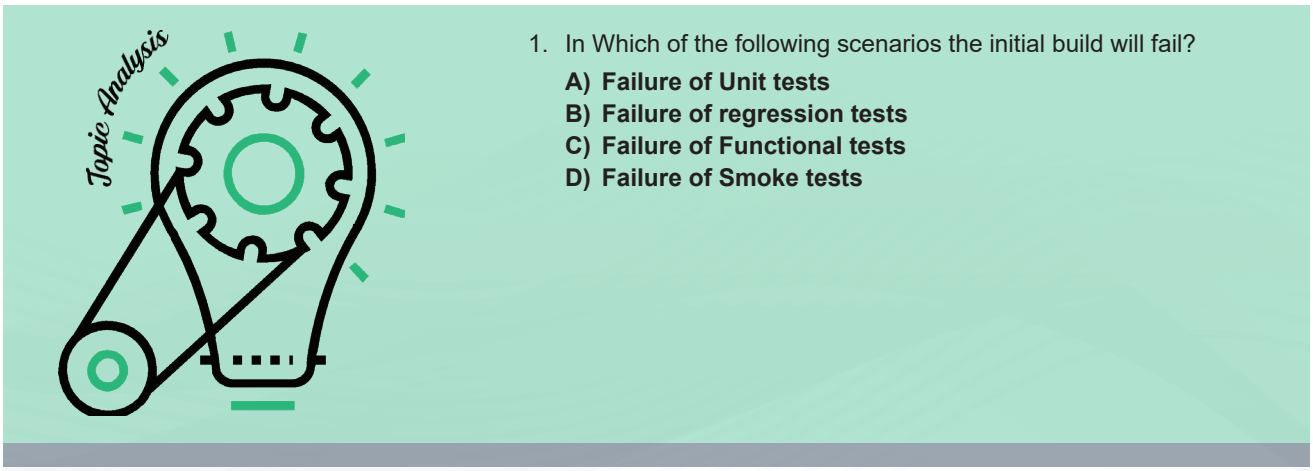
11. Always run smoke tests after a deploy.

- In the bullet above, we mentioned smoke tests, but we still feel that they deserve a special note. It is very useful to be able to verify that the successful deployment actually works as expected and has not broken any common user flows.

12. Provide an easy way to rollback.

- Last but not least, we feel that a successful CI / CD pipeline's most important part is an easy, "one-button" way to roll back your changes if something goes wrong. This usually involves idempotent deployments where doing a rollback simply means redeploying a previous release.

What did You Grasp?



1. In Which of the following scenarios the initial build will fail?
   - **A) Failure of Unit tests**
   - **B) Failure of regression tests**
   - **C) Failure of Functional tests**
   - **D) Failure of Smoke tests**

2. When do you recommend running a smoke test?
   - **A) After a release**
   - **B) After deployment**
   - **C) After UAT phase**
   - **D) Before deployment**

3. Which of these components should be version controlled?
   - **A) Code**
   - **B) Configuration**
   - **C) Databases**
   - **D) All of the above**

Facilitator Notes:

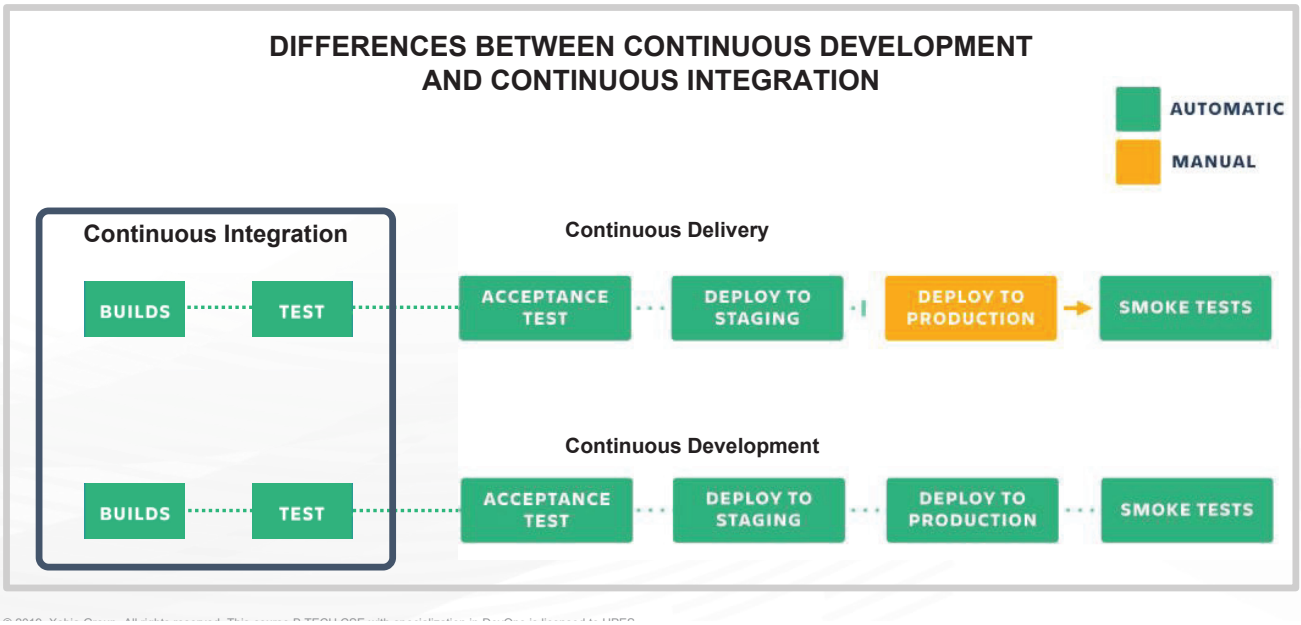
Ans:

1. A. Failure of Unit tests

2. B. After deployment

3. D. All of the above

1.3.3 Continuous Development/Integration



Facilitator Notes:

Explain the participants about Continuous integration and continuous development

The differences between continuous development and continuous integration:

Continuous development (CD) is an iterative development of software and is considered as a umbrella over several other processes including continuous integration, continuous delivery, continuous testing, and continuous deployment.

Continuous Integration (CI) is a software engineering practice where developers will integrate their code multiple times a day into a shared repository to get quick feedback on the feasibility of that code. It is possible to carry out automated building and testing through CI. So the teams can work together quickly on a single project.

What did You Grasp?

1. Fill in the blank.

   _____ is considered as an umbrella for CI/CD.

   A) Continuous Integration
   B) Continuous Deployment
   C) Continuous Delivery
   D) Continuous Development

Facilitator Notes:

Ans:

1. D. Continuous Development

1.3.4 Continuous Testing

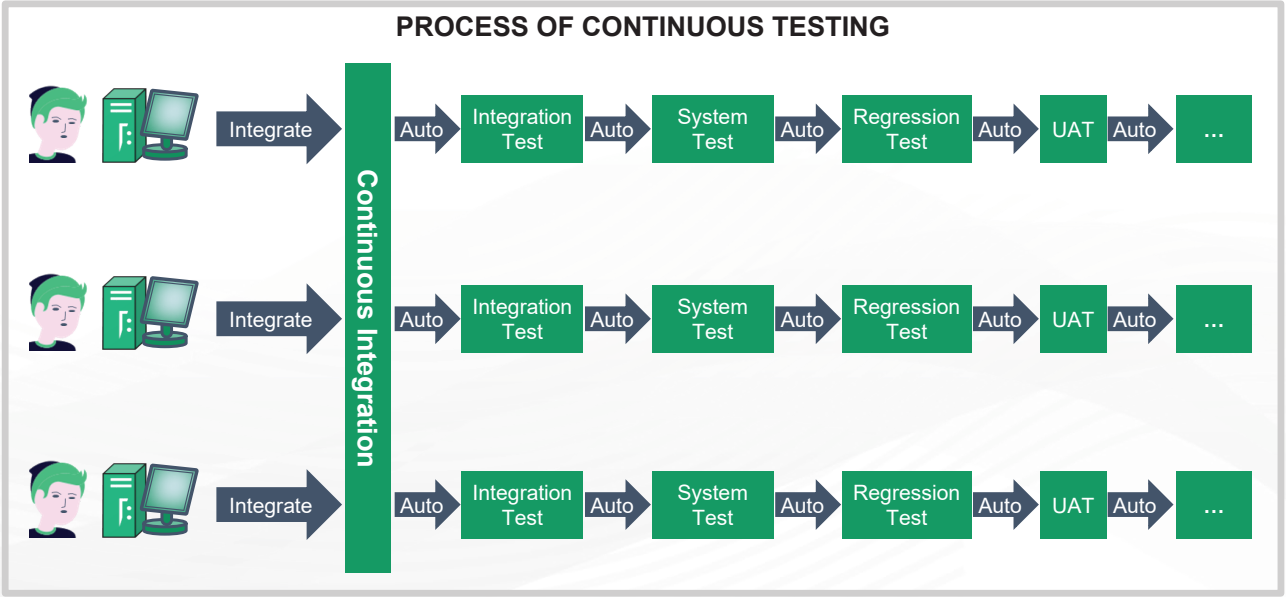
Continuous testing

- Speed and quality are the huge benefits of Continuous testing
- The feedback on new code will be faster and quicker
- It boosts the confidence of the team and encourages them to improve continuously.
- Continuous testing is important process in the continuous delivery pipeline
- Continuous testing includes, various testing phases where automated quality gates and automated tests are executed automatically

Facilitator Notes:

Explain the participants about the Continuous Testing.

- Continuous testing is executing multiple tests in the build process and it is considered a significant method in the continuous delivery pipeline along with Continuous Integration.
- DIfferent types of tests like unit testing, integration testing, load testing, static code analysis,and performance testing are executed automatically in a pipeline.



Facilitator Notes:

Explain the participants about the Continuous Testing.

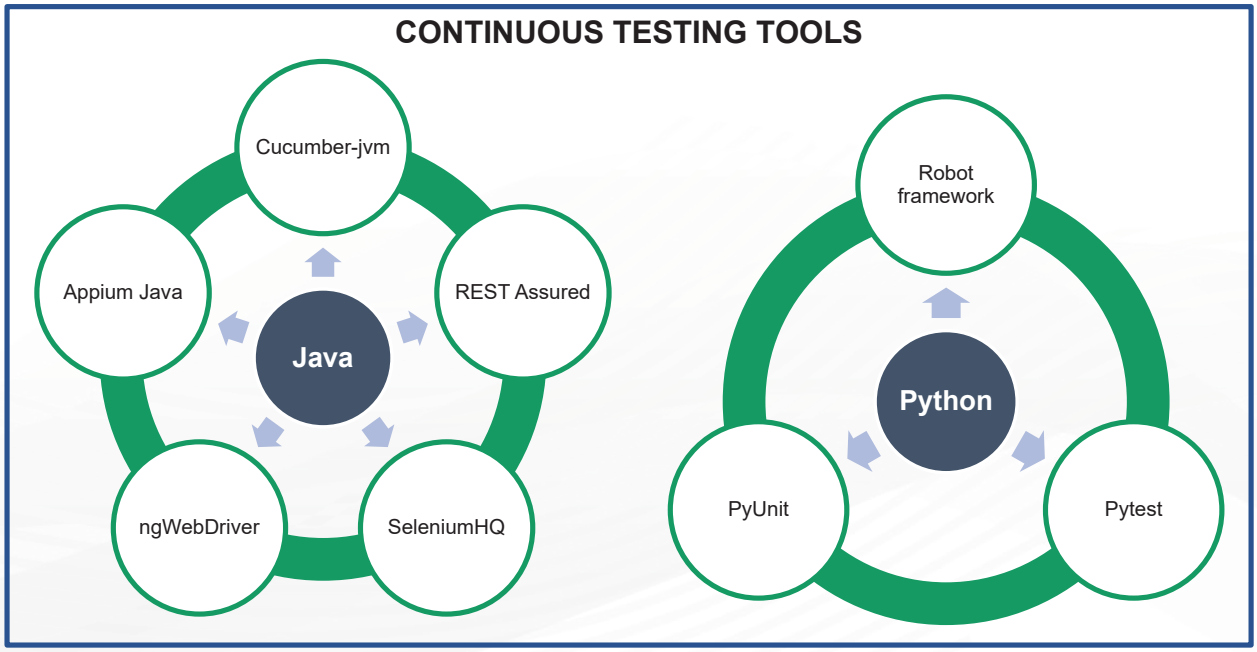
Continuous testing is the process of running automated tests as part of CI/CD pipeline. The code built during the Continuous Integration process is sent into a pipeline of where integration, performance, system, regression, and user acceptance tests get executed automatically – without any intervention.

Advantages of Continuous testing:

- Issues can be found in the stage of development itself.
- Since QA is now part of the system, there is no need to push the product into a separate QA environment for cumbersome hour-long manual testing.
- After release, there is less risk that the product will go bad.

- This type of multi-stage testing helps to achieve better code quality.
- With continuous testing, same glitches can all be found at a time, helping to prevent their future recurrence. Prioritizing defects first helps to address critical issues.

1.3.4 Continuous Testing Tools



Facilitator Notes:

Explain the participants about the Continuous Testing tools.

Several open source tools assist you write automated acceptance tests and reuse the abilities of your development teams. If Java is your main development language, the following choices are helpful:

Java:

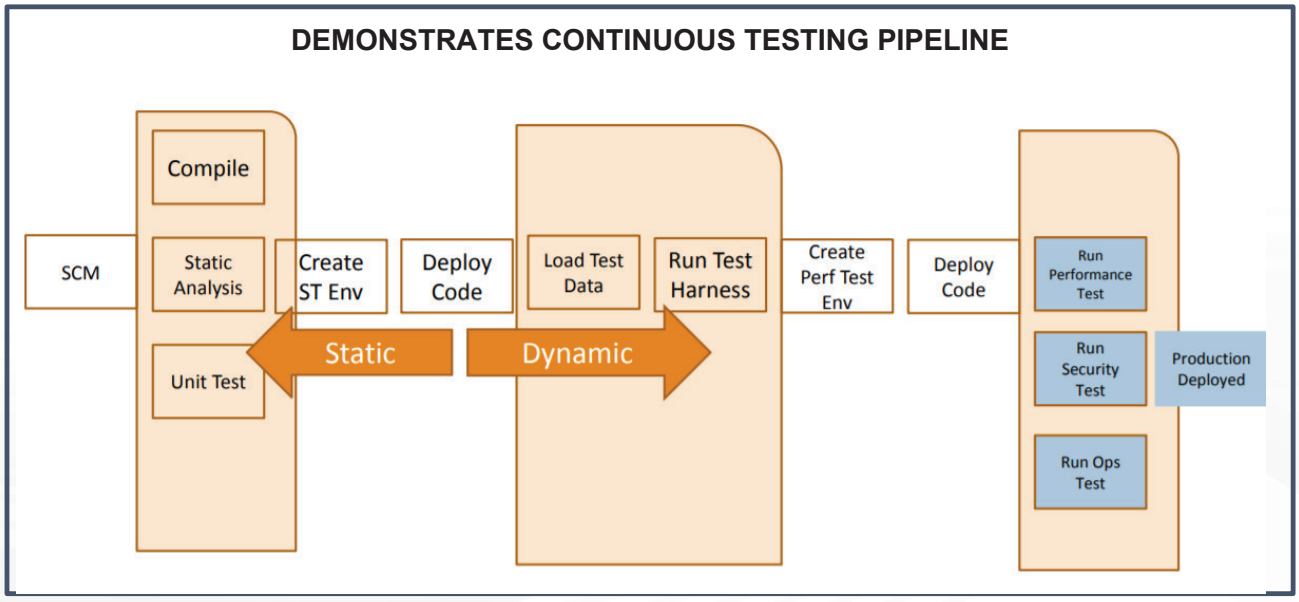
- Cucumber-jvm: It is used for implementing executable specifications in Java for both API and UI automated testing
- REST Assured: It is used for API testing

- SeleniumHQ: It is used for web testing
- ngWebDriver: These locators are used for Selenium WebDriver.
- Appium Java: This is used for mobile testing using Selenium WebDriver

If Python is your main development language, the following choices are helpful:

- Robot framework : Used mainly for test-driven development as well as acceptance testing
- Pytest : Pytest is another preferred Python Test Automation Framework.
- PyUnit : PyUnit, which comes with Python, is the normal unit testing automation framework.

1.3.4 Continuous Testing Pipeline



Facilitator Notes:

Explain the participants about the Continuous Testing Pipeline

Refer the above picture to get an idea about the Continuous testing Pipeline.

Continuous testing is about finding the defects as early as possible. Below tests are some of the example tests that can be executed as part of the Pipeline

Pre-Deployment :

−Functional Suite

− Test suites against an environment

− UI or API driven

− Load Test

Staged (last step before customer)

− Load Test

− Integration Test

− Performance and Security

1.3.4.1 Deploying and Promoting your Application

The key to deploying any application in a reliable, consistent manner is constant practice

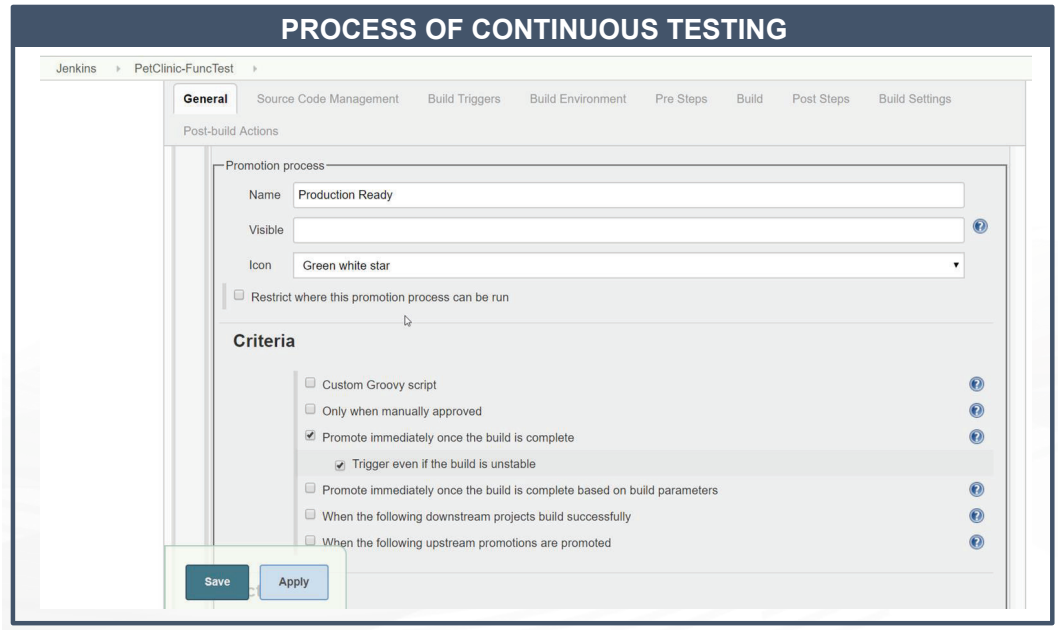
- ⇥ Use the same process to deploy to every environment, including production.
- ⇥ Automating the deployment should start with the very first deployment to a testing environment.
- ⇥ Instead of manually pulling the pieces of software into shape, write a simple script that does the job.

Facilitator Notes:

Give the participants a brief about deploying and promoting your application.

The strategies and plans listed above are fairly generic. They are worth considering for all projects, even if, after some consideration, you decide to only use a few of the sections.

You do not want the QA team to review an application until it has been automatically tested. To achieve this, you can use the promotion plugin in Jenkins build tool.



Facilitator Notes:

Give the participants a brief about deploying and promoting your application.

Promoting builds:

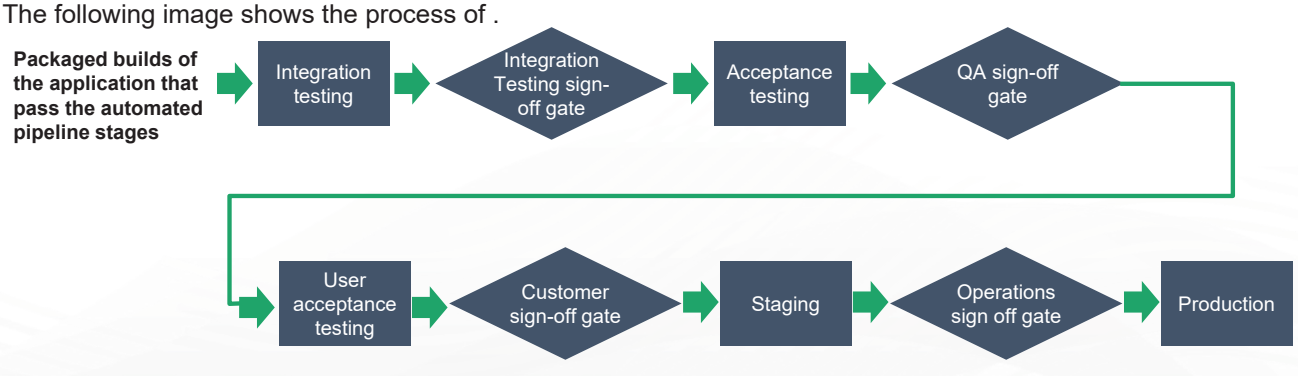
It is essential to differentiate builds from each other, based on verification or passing certain stages.

Let us see how to promoted a build in Jenkins (Build tool). Refer the picture in this slide for an example.

- Install the promoted builds plugin.
- Go to project configuration and check Promote builds when
- This build is promoted immediately once the build is complete, irrespective of its status.
- Builds can be promoted with manual approval too.

1.3.4.2 Modeling Your Release Process and Promoting Builds

The following image shows the process of .



Facilitator Notes:

Give the participants a brief about modeling Your Release Process and Promoting Builds.

As your application grows and becomes more complex, so will your deployment pipeline implementation. Since your deployment pipeline should model your test and release process, you need first to work out what this process is. While this is often expressed in terms of promoting builds between environments, there are more details that we care about. In particular, it is important to capture

- What stages a build has to go through in order to be released (for example, integration testing, QA acceptance testing, user acceptance testing, staging, production).
- What the required gates or approval are.
- In each stage, who has the approving authority of a build passing through that stage.

What did You Grasp?

1. From which of the following environment you should configure auto deployment?
   - **A) Testing**
   - **B) Staging**
   - **C) UAT**
   - **D) Production**

2. _____ is performed before User acceptance tests
   - **A) Regression tests**
   - **B) Functional tests**
   - **C) Integration tests**
   - **D) None of the above**

3. Approval is required to deploy the build to following environment.
   - **A) Staging**
   - **B) UAT**
   - **C) Pre-Prod**
   - **D) Production**

Facilitator Notes:

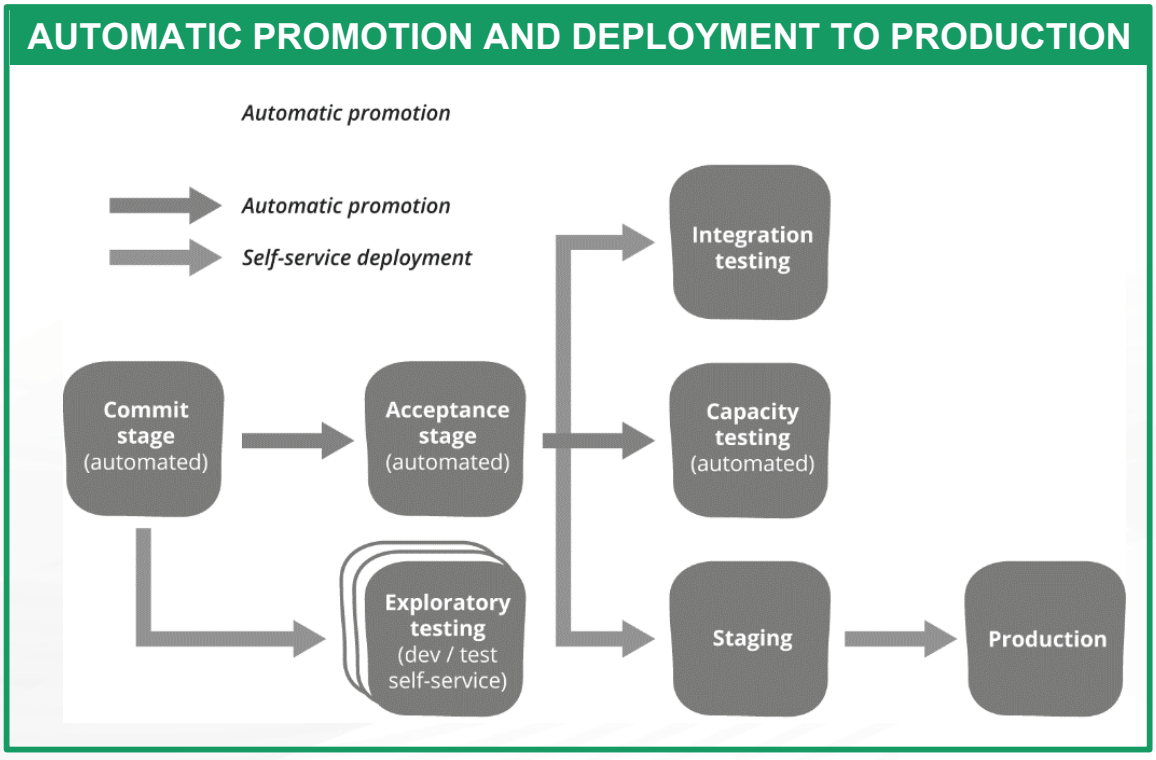
Ans:

1. A. Testing

2. C. Integration tests

3. D. Production

1.3.5 Continuous Deployment to Successive Environments before Production



Facilitator Notes:

Give a brief to the participants about continuous deployment to test environments.

Refer the picture given on the slide. Before the code is deployed into production acceptance tests, functional tests, integrations tests, capacity tests etc and executed and it is deployed to staging environment. If there are no critical bugs found in the staging environment that a build is ready to be deployed into production.

In general, a production-like environment has the following characteristics.

- It should run the same operating system as the production system.
- It should have the same software installed as the production system —and in particular, none of the development toolchain (such as compilers or IDEs) should be installed on it.
- This environment should, as far as is reasonable, be managed the same way as the production environment
- In the case of client-installed software, your UAT environment should be representative of your clients' hardware statistics, or at least someone else's real-world statistics.

1.3.6 Continuous Monitoring

Monitoring targets fall into several primary categories, such as

- ⇥ Complete operation monitoring (e.g. Up-time, Availability, Concurrency, Jobs, etc.)
- ⇥ Aggregated and searchable alert and error log files
- ⇥ Host level utilization trends (e.g. CPU, Memory, Storage)
- ⇥ Pre and Post deployment baselines and comparisons
- ⇥ Establish key performance metrics (KPI's)

Facilitator Notes:

Give the participants a brief about the continuous monitoring of a delivery pipeline.

You then need to work out what to actually monitor, and how to display that information. Many companies use Nagios for Continuous monitoring. Nagios, a powerful and widely used IT monitoring and management software for problem -solving. It detects problems related to your organization's infrastructure and helps in resolving the issue before it impacts the business. In Nagios the events are classified as:

Green means all of the following are true:

- All expected events have occurred.
- No abnormal events have occurred.
- All metrics are nominal (within two standard deviations for this time period).
- All states are fully operational.

Amber means at least one of the following is true:

- An expected event has not occurred.
- At least one abnormal event, with a medium severity, has occurred.
- One or more parameters are above or below the nominal values.
- A noncritical state is not fully operational (for example, a circuit breaker has cut off a noncritical feature).

Red means at least one of the following is true:

- A required event has not occurred.
- At least one abnormal event, with a high severity, has occurred.
- One or more parameters are far above or below the nominal values.
- A critical state is not fully operational (for example, “accepting requests” is false where it should be true).

1.3.6.1 How does Continuous Monitoring Supports DevOPS

The following points can be some of the factors that need monitoring

- ⇥ Static code analysis – New code do not pass quality gates
- ⇥ Failure of the CI process
- ⇥ Infrastructure issues where an application needs to be deployed
- ⇥ Application issues

Facilitator Notes:

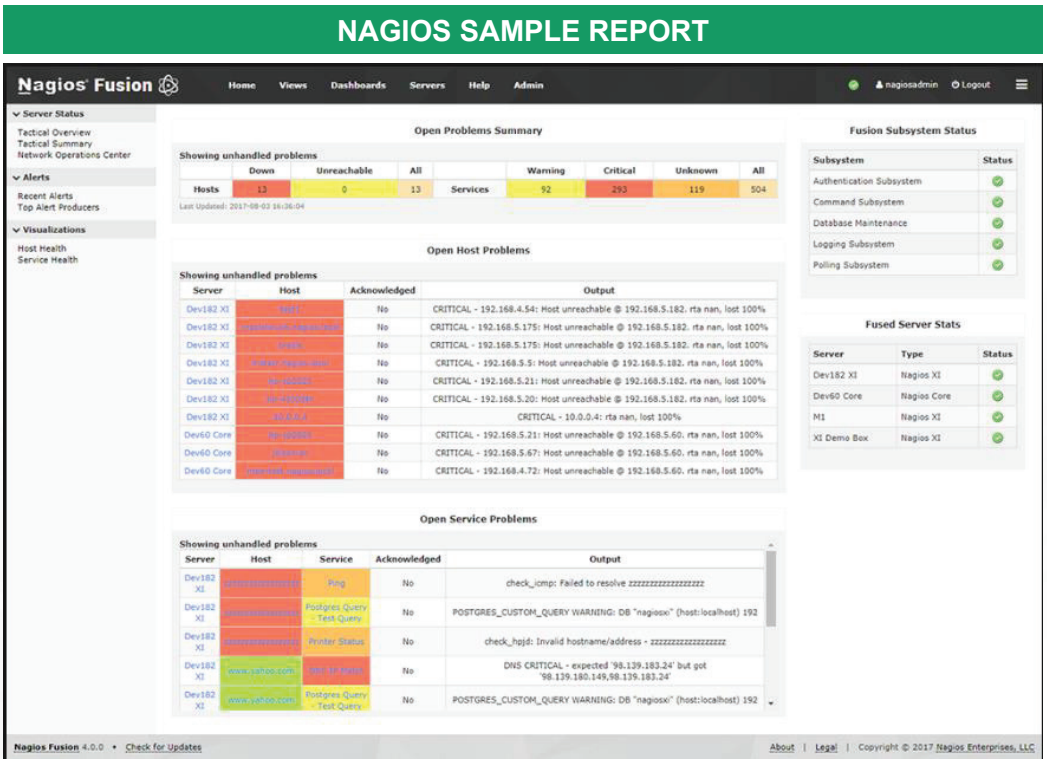
Give the participants a brief about how the continuous monitoring support DevOPS.

The process and technology used to identify compliance and risk problems connected with the economic and operational environment of an organization is continuous monitoring. The economic and operational environment comprises of working together individuals, procedures and systems to promote efficient and effective activities.

It is important to monitor the below points as part of a DevOPS pipeline:

- Static code analysis – New code do not pass quality gates
- Failure of the CI process
- Infrastructure issues where an application needs to be deployed
- Application issues

1.3.6.2 Nagios Sample Report

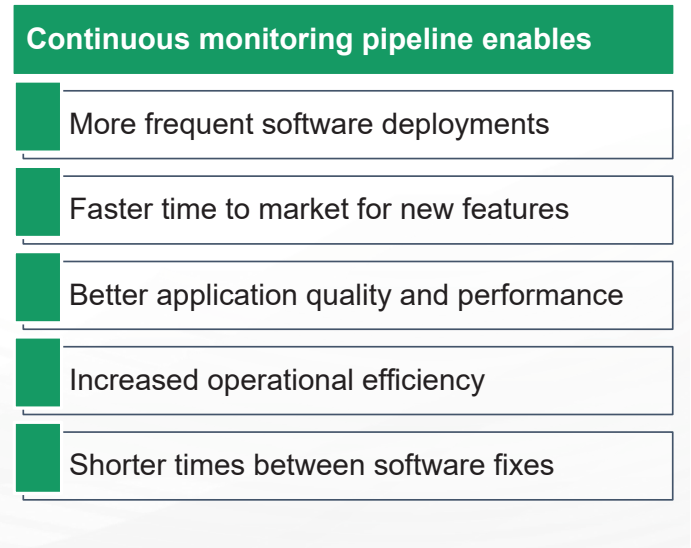


Facilitator Notes:

Give the participants a brief about the monitoring tools (Nagios) used for continuous monitoring of a delivery pipeline.

- As with continuous integration for the development team, it's essential that the operations team has a big visible display where they can see at a high level if there are any incidents.
- They then need to be able to dive into the detail when things go wrong to work out what the problem is. All the open source and commercial tools offer this kind of facility, including the ability to view historical trends and do some kind of reporting.
- A screenshot from Nagios is shown in above slide. It's also extremely useful to know which version of each application is in which environment, and that will require some additional instrumentation and integration work.

1.3.6.1 Benefits of Continuous Monitoring in a DevOps Pipeline



Facilitator Notes:

Give the participants a brief about how the continuous monitoring benefits in DevOps pipeline.

Continuous monitoring is an end-to-end distribution pipeline backbone, and open source monitoring tools are like ice cream scoop toppings. In order to have transparency about all the procedures, it is desirable to have monitoring at almost every point. It also enables us to rapidly resolve problems. Monitoring should be a well-considered plan application.

Continuous monitoring pipeline enables:

- More frequent software deployments
- Faster time to market for new features
- Better application quality and performance
- Increased operational efficiency
- Shorter times between software fixes

What did You Grasp?

1. Which of the following metrics should be monitored from a continuous delivery pipeline?
   A) Vulnerabilities
   B) User activity
   C) Application Log output
   D) All of the above

2. Fill in the blank.
   As per nagios event classification if an expected event do not occur, then it is classified as……..:.....
   A) Amber
   B) Red
   C) Green
   D) None of the above

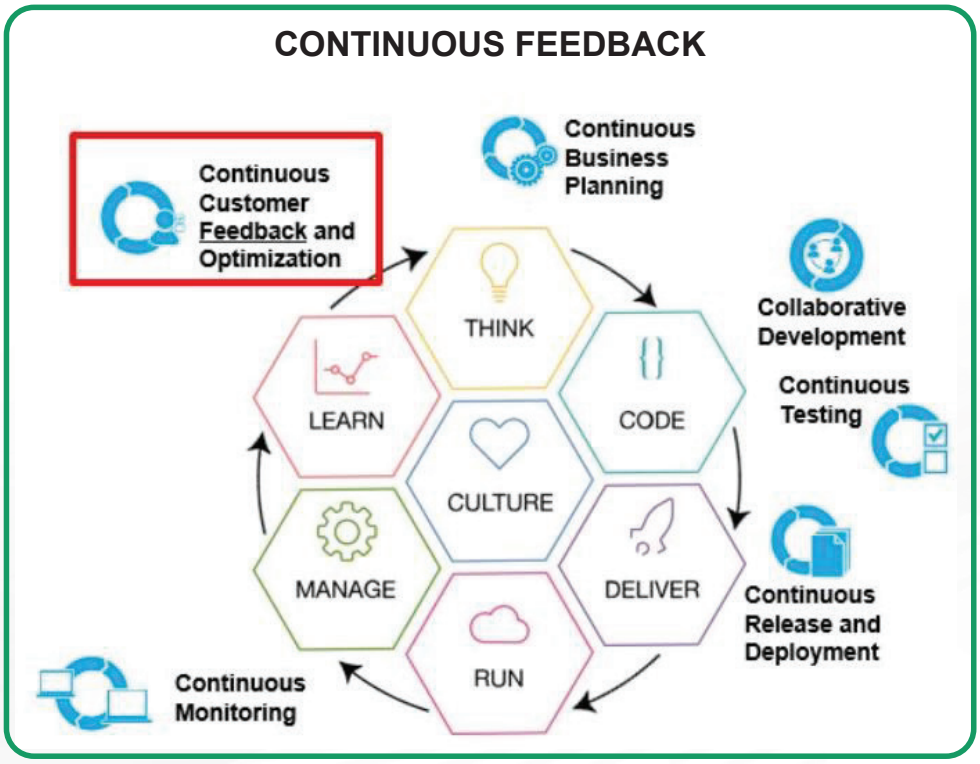
Facilitator Notes:

Ans:

1. D. All of the above

2. A. Amber

Continuous Feedback



Facilitator Notes:

Give a brief to the participants about Continuous feedback.

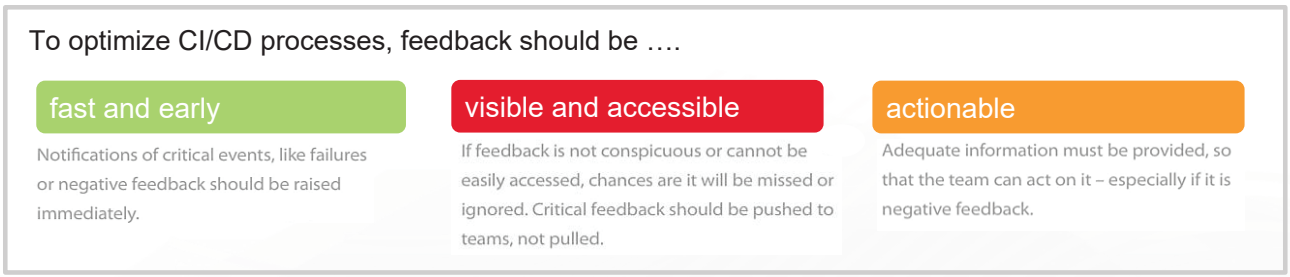
The feedback loop is very a important aspect of DevOps. It plays a very important role for the success of DevOps. The feedback loop consists of automated communication between an issue registered during development and the human element of the DevOps process. It should be managed by a tool, through which issues must be registered manually or via an

automated mechanism. The issue should also be tagged with an artifact that developers can use to trace what occurred, why it occurred, and where in the process it occurred. The tool should help to define the chain of communication with all automated and human elements in the loop.

Continuous feedback is essential to application release and deployment because it evaluates the effect of each release on the user experience and then reports that evaluation back to the DevOps team to improve future releases.

Stakeholders (customers, users, management) see results quickly and become more engaged in the project, offering more time, insight, and funding. The most energizing phenomenon on a software team is seeing early releases perform (or be demoed) in action. No amount of spec review can substitute for the value of working bits.Stakeholders (customers, users, management) see results quickly and become more engaged in the project, offering more time, insight, and funding. The most energizing phenomenon on a software team is seeing early releases perform (or be demoed) in action. No amount of spec review can substitute for the value of working bits.

1.3.7.1 Continuous Feedback

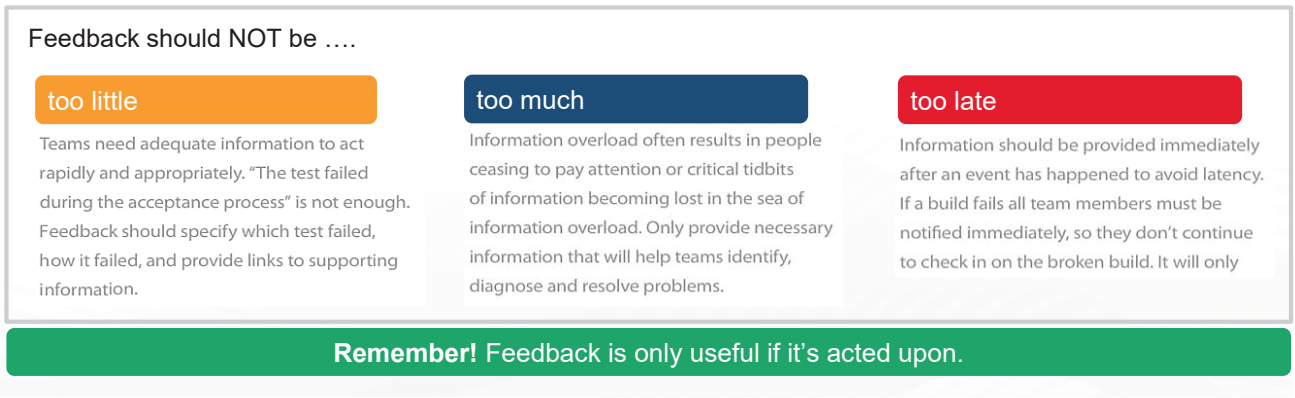


Facilitator Notes:

Give a brief to the participants about the continuous feedback that should be.

Feedback should be Fast and early, Visible and accessible and Actionable.

1.3.7.2 Continuous Feedback

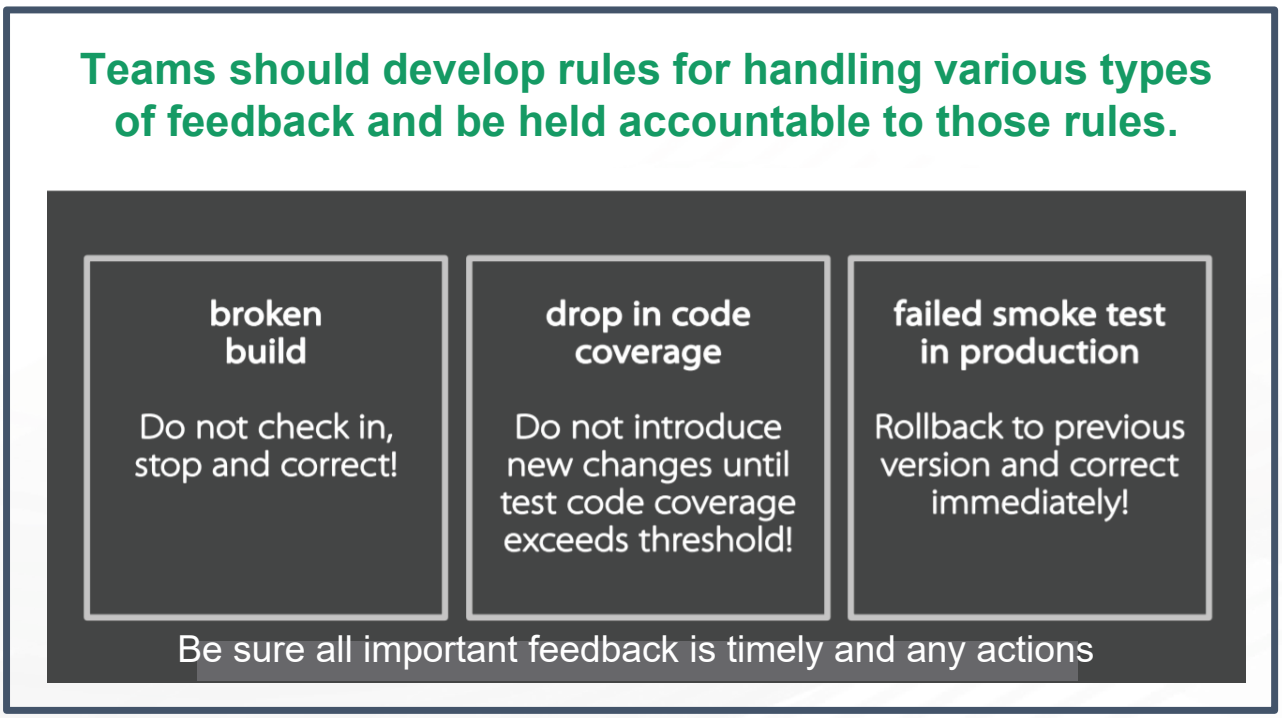


Facilitator Notes:

Give a brief to the participants about Continuous Feedback that should not be.

Feedback should not be Too little, Too much and Too late:

1.3.7.3 Continuous Feedback Rules



Facilitator Notes:

Explain the participants about continuous feedback rules.

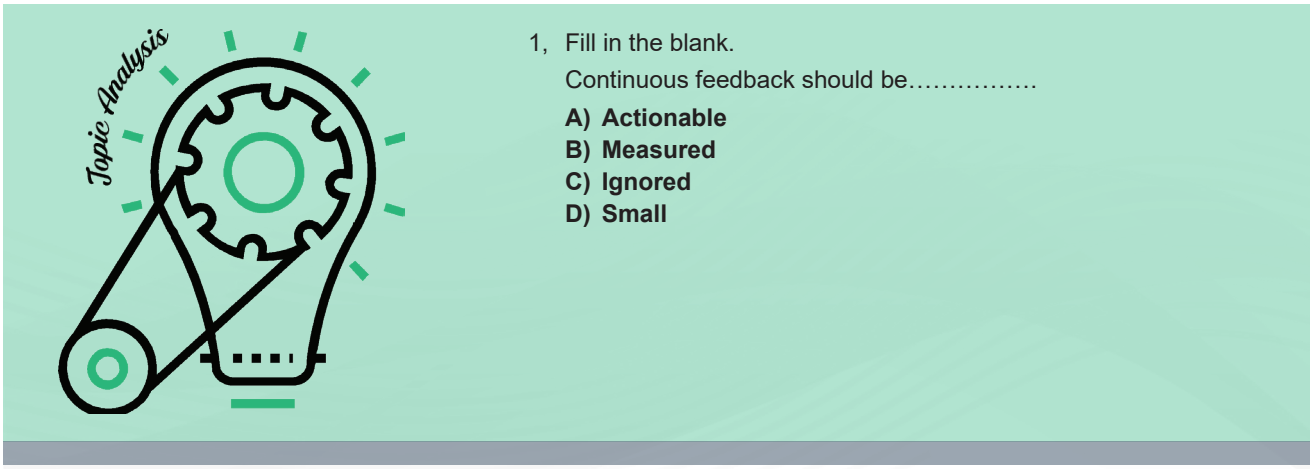
The various rules of Continuous Feedback.

Broken Builds: If any builds are failed, do not merge that code. Fix the code and then merge the code. So that only Quality code will be checked-in.

Drop in Code – Coverage: Code coverage is one of the important metrics for any product. As per the standard most of the companies set 80% coverage standard for new code

Failed Smoke-test: If smoke test fails. Stop the build immediately and rollback to the previous working version. Fix the issue and then perform the Smoke test.

What did You Grasp?



2. Fill in the blank.

_____ reports comes under continuous feedback

A) Number of resources
B) Failed smoke tests
C) Product Training
D) None of the above

3. State True or False.

Continuous feedback is essential to application release and deployment.

A) True
B) False

Facilitator Notes:

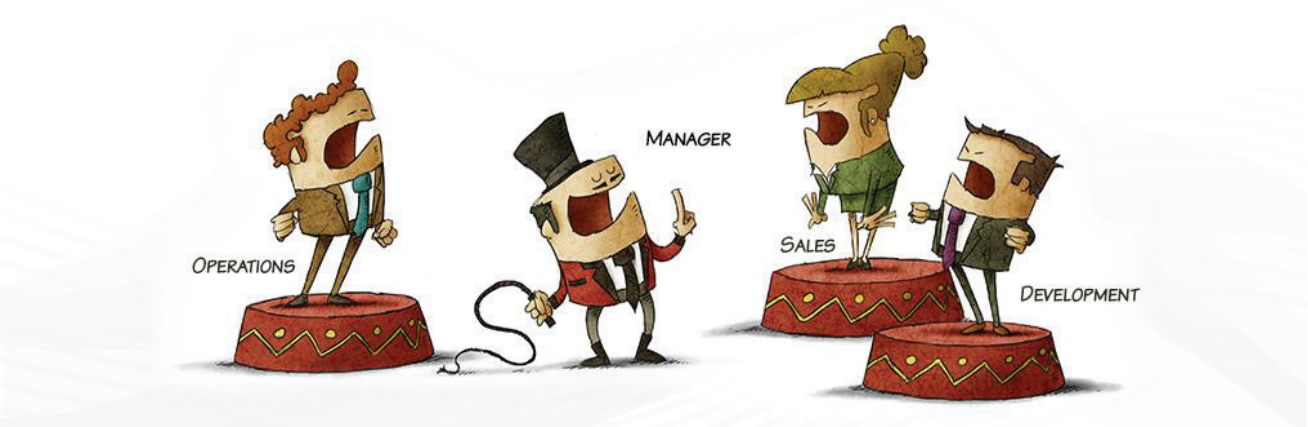
Ans:

1. Actionable

2. Failed smoke tests

3 1. True

Group Discussion



Facilitator Notes:

Divide the students in groups and discuss about releasing an application to the production, CD engineering best practices, Continuous Testing and Continuous Feedback.

We've so far discussed about different aspects of releasing an application to the production, the best practices followed by CD engineering , Continuous testing and Continuous Feedback. Let's discuss Compuware DevOps case study and know how they got benefited by adopting DevOps model.

1.3.8 Compuware DevOps Transformation Case Study

Challenges

- Compuware was following Waterfall model for 40 years.
- They are a slow-moving development organization.
- They were trying to compete in the digital economy, which requires you to be fast, (fast beats slow).
- They needed to be innovative, and have lot of new ideas. Ideation is the key to success in the new economy.
- Being a software vendor they need to maintain quality.

Facilitator Notes:

Explain the participants about Compuware DevOps transformation case study.

For Compuware, the solution to above problems were:

Agility: They wanted to be fully agile so that they could have continuous code drops and fulfill business needs in a timely basis.

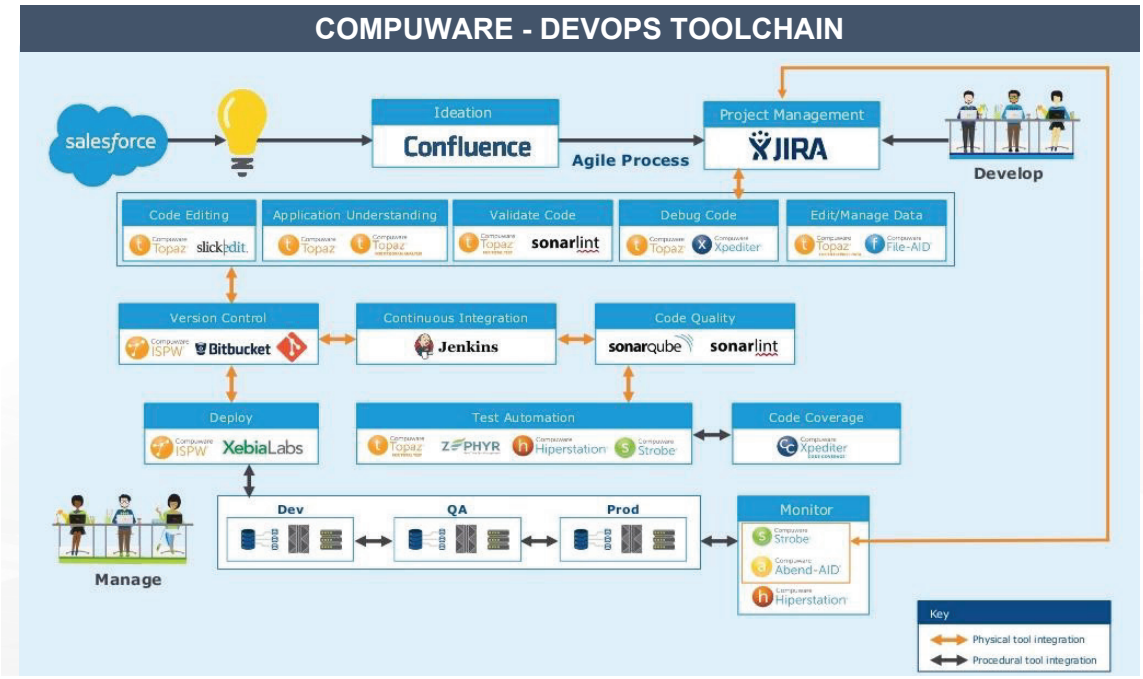
Confidence: They wanted to have the confidence to know that when changes are made, what they were delivering was going to work and was good, so we had to know that they could move at a fast pace to be able to deliver what they wanted.

Efficiency: They had to be efficient, and move across the entire enterprise.

Ease of use: Their tools had to be easy to empower the developers to be able to move at that pace, and to make updates and enhancements that would be valuable, provide business needs, and meet the business objectives.

Integrations: Being in the enterprise companies, the mainframe integrates across the entire enterprise, through the cloud, and all the way back to your mainframe system of record. So they had to make sure that those integrations would be available.

1.3.8.1 Compuware - DevOps Toolchain



Facilitator Notes:

Give a brief to the participants about Compuware DevOps toolchain.

Determining the Right tools:

What they did was to put in a toolchain, below you'll see an example of Compuware DevOps Toolchain that they are using to develop the software.

- First, in the upper left hand side, let's start with Salesforce, which is how they interact with the customers so that they can provide ideas to Compuware, and can communicate using that portal.
- Next tools is Confluence. When they have a new idea that comes in, they put it out in Confluence, and they get a lot of collaboration from the people around the world who are interacting with their customers and using Compuware products.
- Finally, they use JIRA for all of the project management, for all of the task management, for the agile boards, etc. so that they can keep track of everything they are doing.

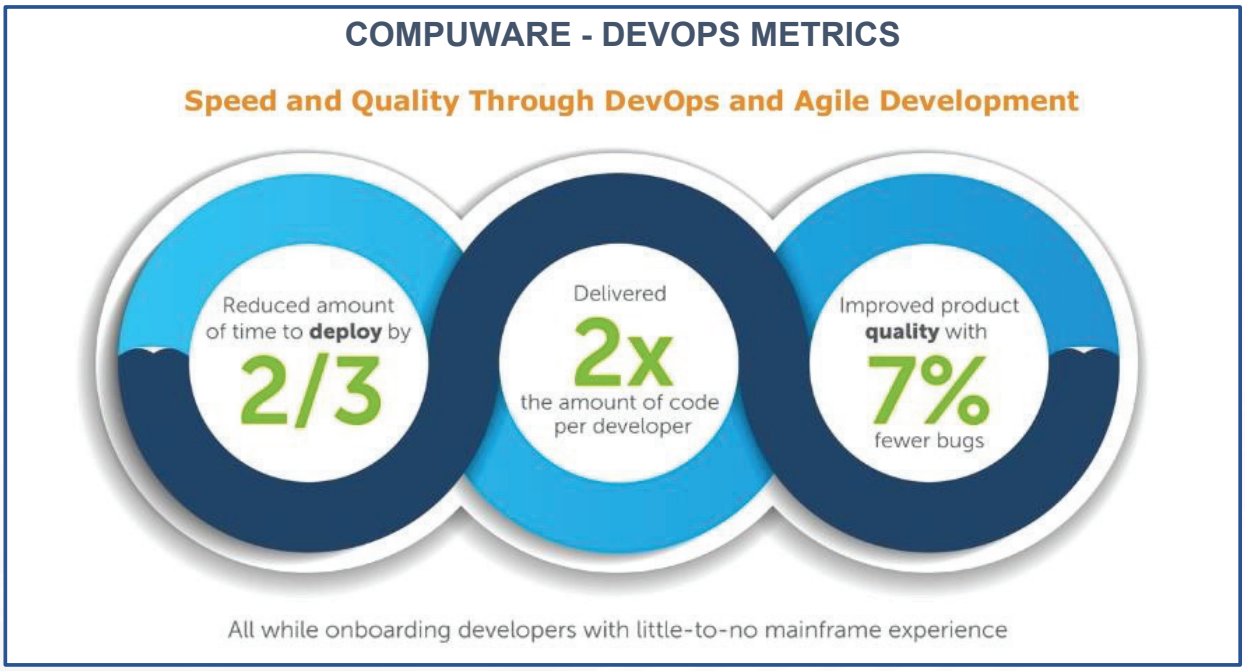
The next phase happens as they move through code editing, application understanding, to validate their code, debug the code, edit, and manage the code, etc.

For this stage, they use the Compuware Topaz base, which is an Eclipse Base ID which interacts with the mainframe tools. It works with the interactive debuggers, and it can integrate with all the different tools.

- They use two version control systems.
- They use ISPW, which is the mainframe source code management system; and they also use Git for the distributed code. They believe that the source code should be managed on the platform where it is going to be built and executed, so mainframe code stays on the mainframe, distributed code stays on distributed platform.
- They use Jenkins for the continuous integration into ISPW and Git to identify when code changes are made. Once those code changes are made, it automatically initiates automated testing; both mainframe, distributed platforms as well as builds.

- Then the results from the testing go into Sonar where they can use it to identify code quality, identify how the changes are made to their products, and how the code is impacting the quality of what they are delivering.
- For test automation, they use not only Topaz products, but also use Zephyr and Hiperstation Strobe for performance. So they are monitoring the continuous performance and continuous viability of all code changes. They like to ensure that they have at least 50% code coverage as they move along.
- And then, finally, they get to where they have to deploy. Again, they use ISPW and XebiaLabs XL Release to do the deployments across different platforms, and that allows them to move and confidently move code changes from development to QA to production.
- Then, once they are in production, they have to monitor if there's faults, errors, or any performance issues. They continuously monitor performance using Strobe, and have an integration back into JIRA. Within the environment, if an issue is found, JIRA issues can be opened and sent back to developers so that they can keep a continuous integration in the true spirit of being fully DevOps enabled.

1.3.8.2 Compuware - DevOps Metrics that matter



Facilitator Notes:

Explain the participants about Compuware DevOps metrics that is improved with CI/CD

- Through the DevOps toolchain, Compuware is able to cut their deployment time by two thirds, which means they were able to more rapidly deploy the software that they have.
- As per the estimate they do about two times more code delivery per developer per year, and that's not measured by lines of code.
- They actually measure that by stories, tasks, and enhancements that are delivered. In other words, they are measuring on the value that they are providing to the end users in applications that is being delivered, not just simply lines of code.
- Through automated testing and the use of continuous integration, year after year they have reduced the defects reported by the customers and end users by seven percent.

In a nutshell, we learnt

1. Releasing an application to production
2. CD engineering practices
3. Continuous Development & Integration
4. Continuous testing & promotion of builds
5. Continuous Deployment to successive environments until before Production
6. Continuous monitoring for the delivery pipeline
7. Continuous feedback

Facilitator Notes:

Share the module summary with the audience.

Ask the participants if they have any questions. They can ask their queries by raising their hands.

