

# Continuous Integration and Continuous Delivery with Jenkins Server

# Traditional Project Cycle

2

- Waterfall model with sequential process flow.
- Software delivery at the end
- No iterative development
- Surprises at end.
- Debugging, troubleshooting difficult with water fall model.

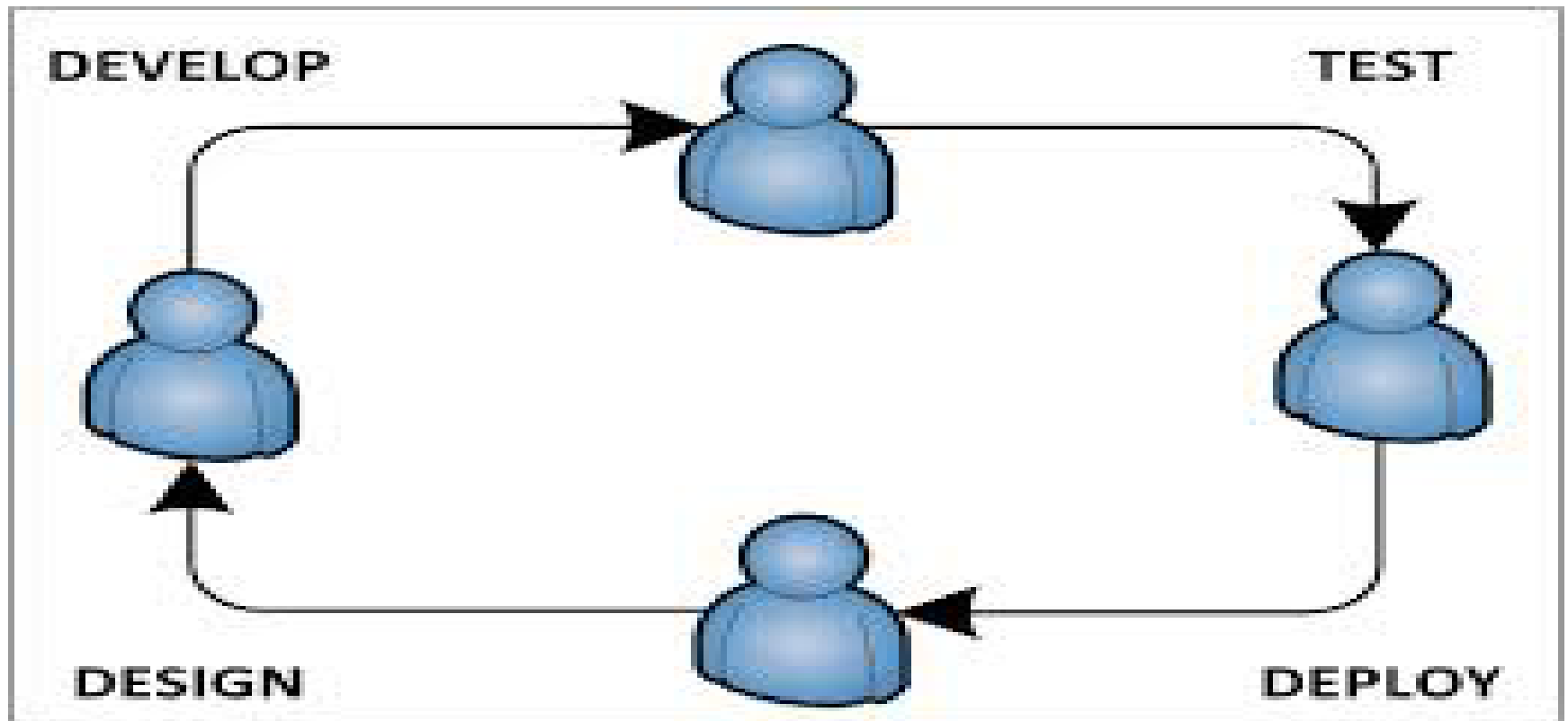
# Agile practices

3

- Entire team is involved.
- Parallel work and collaboration between the teams.
- Smaller release cycles.
- Iterative and incremental builds.
- Modular builds
- Client involvement.
- Developer confidence
- Change management and flexibilities.
- Agile methodologies such as Scrum and XP

# Continuous Process

4



# Application Build Process

5

- Developers in the team write application code and verify the functional behavior by unit testing.
- The QA team verifies the logic.
- The build team prepare the build and release environment.
- Finally package the release by applying various manual processes such as testing, build-release, bug tracking, documentation etc.
- Can this entire process be **automated** ?

# Build Automation with Apache Ant and Maven <sub>6</sub>

- Apache **Ant**
  - Build xml scripts
  - Large size script, but can be reusable
  - Limited portability
  - Build automation with existing code base
  - Cross platform (OS) support
- Apache **Maven**
  - Small XML configuration
  - Cross platform support(OS)
  - Automatic dependency manage/download
  - Centralized code base management in the repository.
  - Sharing the dependencies

Other automation tools such as Nant,MSBuild,TFS etc. are available.

# Development Process

7

- Test **D**riven **D**evelopment.(TDD)
- Test first approach.
- Test Case -> Implementation ->Refactor
- Unit testing
- Integration and acceptance testing
- Continuous Testing
- Minimum and optimum code as per the requirement specifications.

# TDD advantage

8

- Fewer defects
- The reduced maintenance costs
- Higher code quality
- Easier to understand
- More flexible code.
- More focused and effective tests.
- Incremental delivery.



# Java Application Testing

9

- JUnit and extended xUnit frameworks.
- Test automation with Apache Ant.
- Test specifications in BDD format
- BDD Frameworks: JBehave, EasyB, Cucumber
- Test Coverage measurement : Cobertura
- Application code coverage measurement
- Static code Analysis: Sonar, FindBugs
- Report generation

# Source code Repository

10

- The shared centralized code base for all the team.
- Manages automatic versioning of application code.
- Each team member can checkout/in the code with this repository.
- Maintains the history of code.
- Allows to clone, import and export of code modules.
- Replication of code across repositories.
- Detects the change in code base and triggers the build process.
- Example repositories: Git, GitHub, GitLab, SubVersion, CVS etc.

# SCM with GIT/SVN/CVS

11

- SCM version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.
- SVN, ClearCase are supporting distributed version control system.
- The SVN clients don't just check out the latest snapshot of the files, they fully mirror the repository.
- In case of server failures, any of the client repositories can be copied back up to the server to restore the backup.
- Every checkout is really a full backup of all the data.

# SCM Terminologies

12

- Branching : Make a copy of the code
- Branching isolates from other work
- Head : make sub branching of code
- Merge : Extra complexity code with hard integration can be merged.
- Master : The master copy of code base.
- Checkout : Get a snapshot of code from the code repository.
- Check in: Submit the modified version of code to the cod repository.

# Build Automation Continuously..

13

- Teams integrate their work multiple times per day.
- Each integration is verified by an automated build
- Significantly reduces integration problems
- Develop cohesive software more rapidly in an incremental way

*Source: Martin Fowler*

— This process is termed as **Continuous Integration**

# The Continuous Integration Process

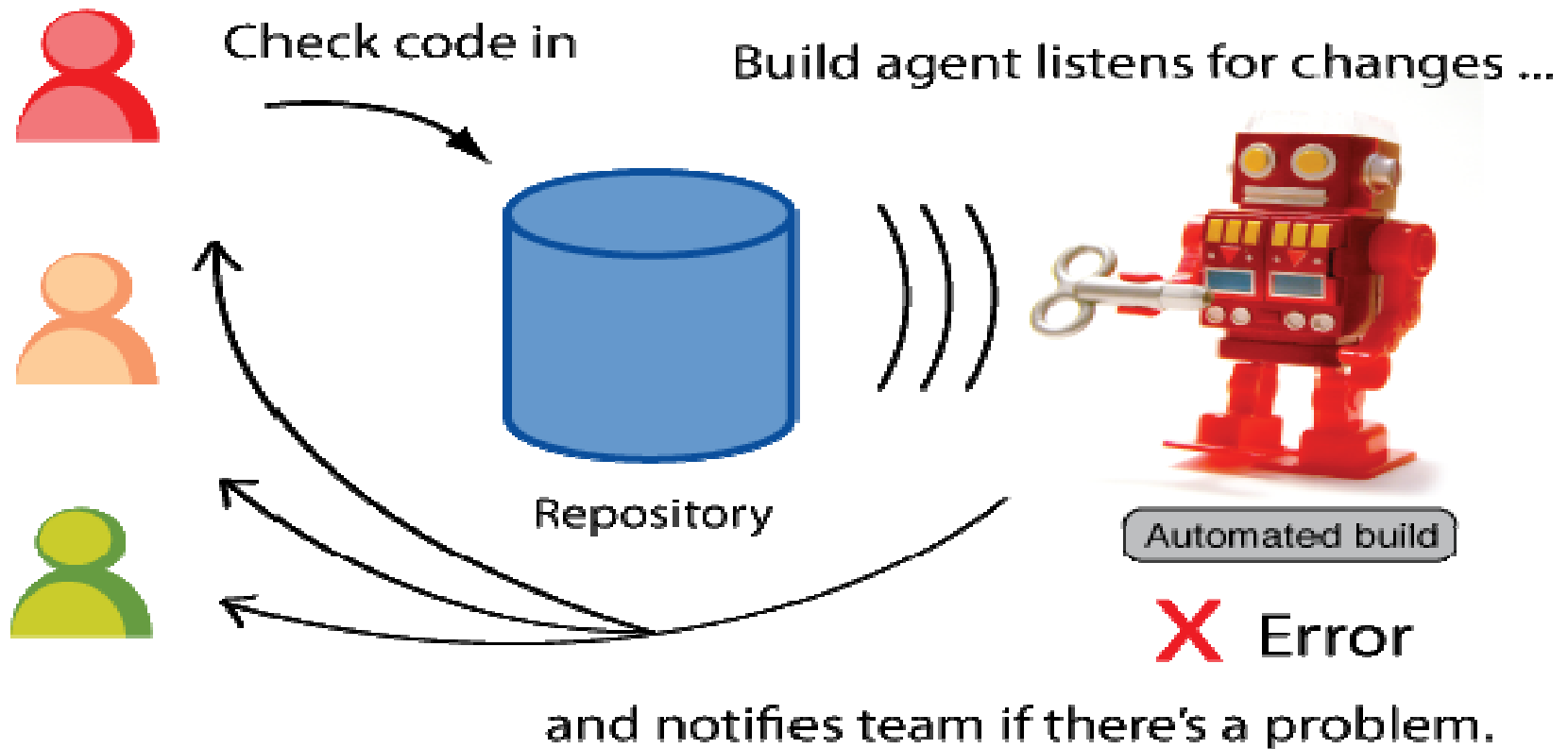
14

- The **Continuous integration (CI)** in software build is the process of merging all developer working copies with a shared mainline several times a day.
- The CI was originally intended to be used in combination with automated unit tests written through the practices of test-driven development.
- This was conceived as running all unit tests and verifying they all passed before committing to the mainline automatically.
- This helps avoid one developer's work in progress breaking another developer's copy.

# The Process...

15

## Developers



# Continuous Integration

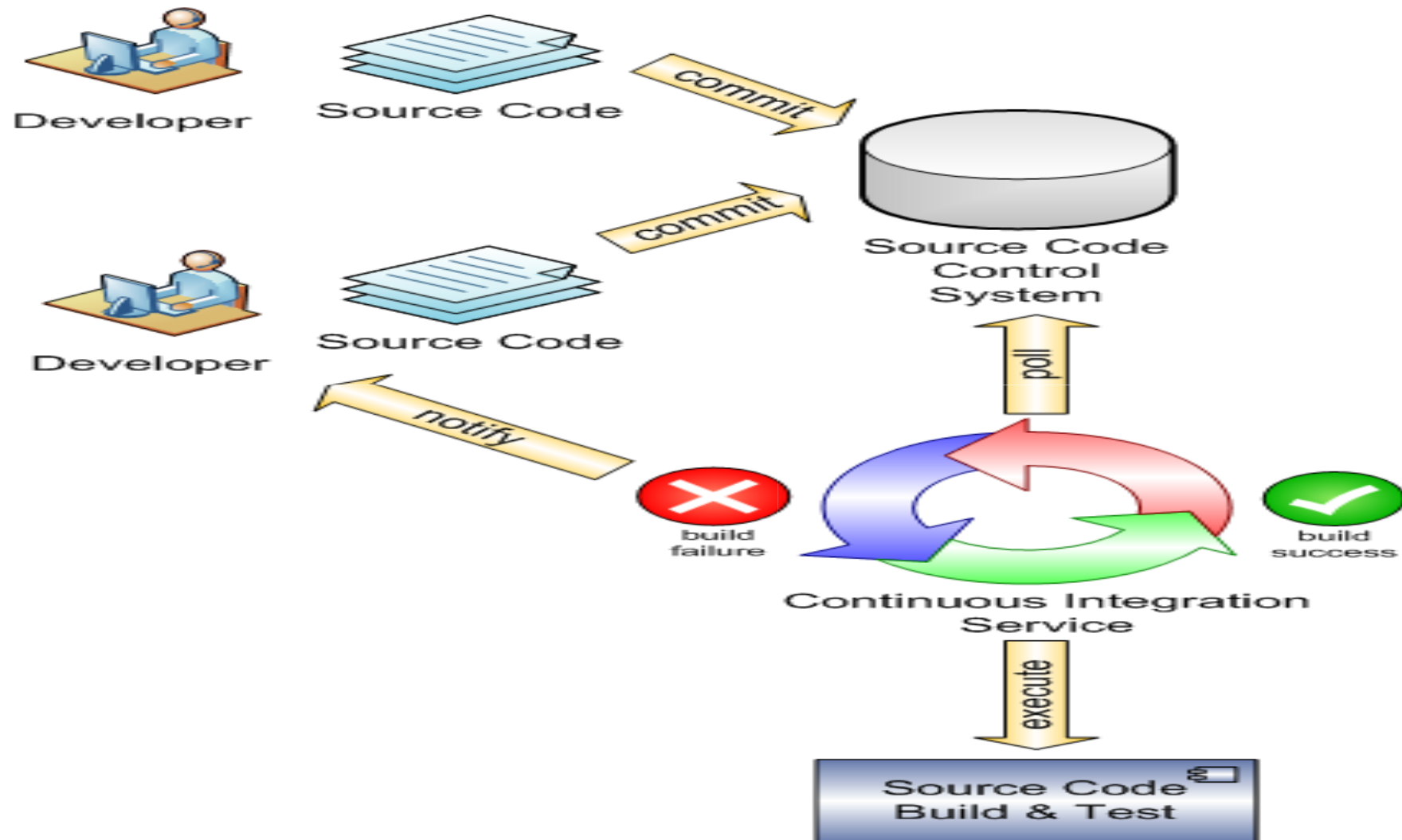
16

- Environments based on stability
- Maintain a code repository
- Commit frequently and build every commit
- Make the build self-testing
- Store every build



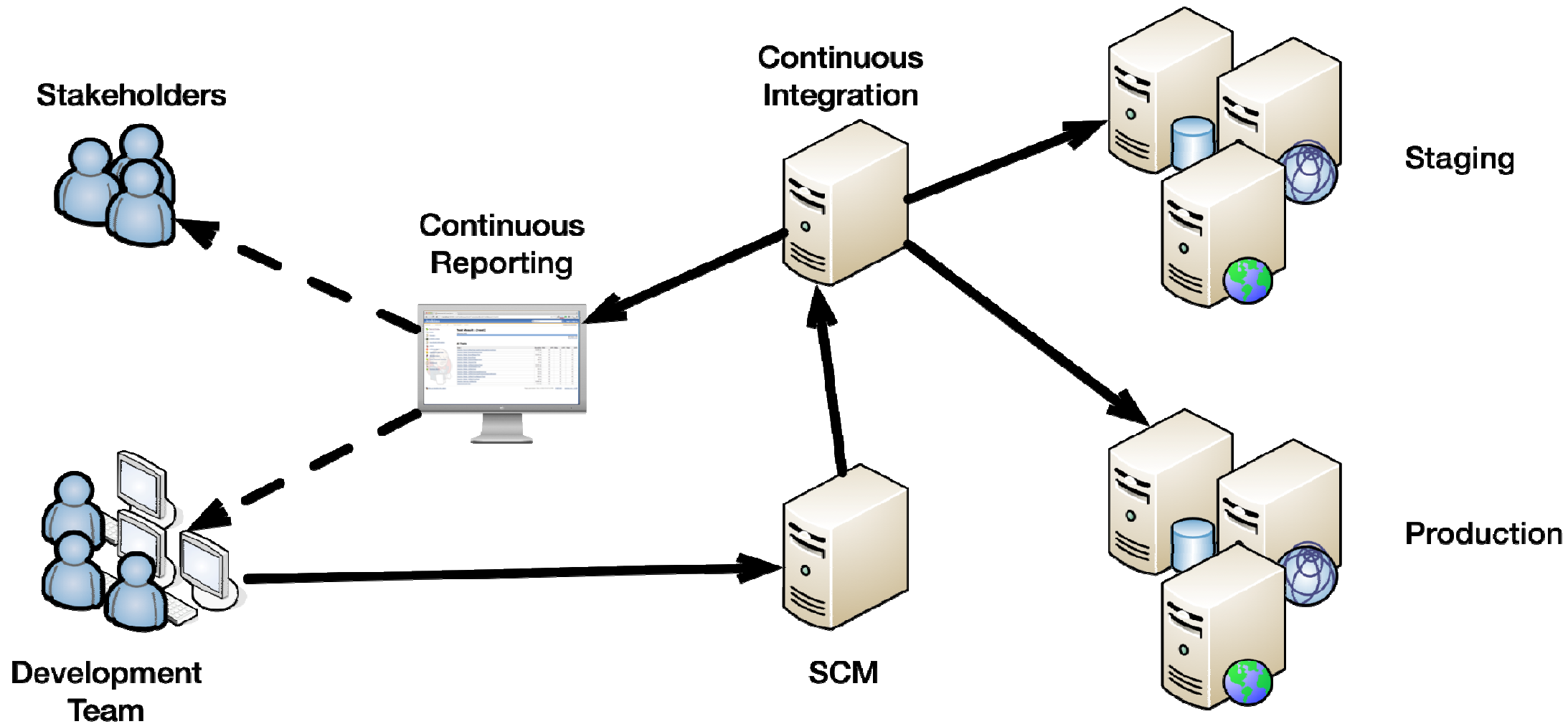
# The automatic integration

17



# Continuous Integration Workflow

18



# CI Features

19

- When unit tests fail or a bug emerges, the developers has the provision revert the codebase to a previous bug-free state, without wasting time in debugging
- Developers detect and fix integration problems continuously — avoiding last-minute chaos at release dates.
- Early warning of broken/incompatible code
- Early warning of conflicting changes
- Immediate unit testing of all changes

# The CI output

20

- The CI makes sure that the software built is always in a state that can be deployed to users and makes the actual deployment process very rapid.
- This process runs continuously and automatically!

# CI Automation

21

- Reducing repetitive processes saves time, costs, and effort.
- These are project activities such as code compilation, database deployments, testing, inspection, deployment, and feedback.

# CI to Developers

22

- Metrics generated from automated testing and CI such as metrics for code coverage, code complexity, and features completely focus the developers more on developing functional, quality code
- The CI helps to develop momentum in a team.

# Risk reductions

23

- The CI helps to mitigate the risks as
  - Lack of functional working software
  - The late discovery of the defects
  - The software does not fit into quality metrics.
  - Lack of project visibility.(when the final module/demo/release is available ?)

# The CI advantage

24

- The CI Process applies small pieces of effort, applied frequently.
- The CI process facilitate QA processes.
- This continuous application of quality control aims to improve the quality of software
- The CI Process reduces the time taken to deliver the software by replacing the traditional practice of applying quality control *after* completing all development.



# CI to reduce the Risk

25

- When we make assumptions in software development, we waste time and increase risks.
- Continuous Integration helps to reduce assumptions on a project by rebuilding software *whenever a change occurs* in a version control system.

# CI Assurance

26

- The CI automation ensures the following..
  - The process runs the same way *every time*.
  - An ordered process is followed. For example, the inspections may run on the code.
  - The static analysis of the code before the running of tests—in the build scripts.
  - The processes will run every time a commit occurs in the version control repository.

# Continuous Integration System

27

- **Identify**—identify a process that requires automation.
- **Design Build** : Creating a build script makes the automation repeatable and consistent.
- **Share** : By using a version control system such as Subversion, Git or CVS and share these scripts/programs with others to use.
- **Make it continuous** : Ensure that the automated process is run with every change applied, using a CI server.

# CI Principles

28

- Maintain a code repository
- Automate the build
- Make the build self-testing
- Everyone commits to the baseline every day
- Every commit to baseline should be built
- Keep the build fast and small
- Test in the production environment
- Make it easy to get the latest deliverables
- Automate the deployment process
- Make the results of the latest build available to everyone

# CI Disciplines

29

- Developers must commit code more frequently
- Make it a priority to fix broken builds
- Write automated builds with tests that pass 100% of the time
- Should never get or commit broken code from/to the version control repository.

# CI Commit Practice

30

- Commit code frequently
- Make small changes
- Commit after each task
- When to commit : Avoid having everyone commit at the same time every day
- Why commit attitude : unless it is 'complete'
- Committing code frequently to the version control repository is the only effective way to implement CI.

# Build every commit

31

- Why build frequently?
- Why not integrate frequently?
- Agile principles
  - If it hurts, do it more often.
  - Many difficult activities can be made much more straightforward by doing them more frequently.
  - Reduce time between defect introduction and removal
- Automate the build
  - Key to continuous integration

# Broken code to CI ?

32

- **Don't commit broken code**
- Make sure not to commit the code that doesn't work to the version control repository.
- To avoid this risk is having a well-factored build script that compiles and tests the code in a reproducible manner on **developer machine!!..**



# Private Builds ?

33

- **Run Private Builds FIRST!**
- To prevent broken builds, developers should run or simulate an integration build on their local workstation IDE after completing their unit tests.
- The simulation can be a mock testing environment!

# Broken from the CI ?

34

- **Avoid Getting Broken Builds**
- When the build is broken, don't check out the latest code from the version control repository.
- This can be monitored by looking at the commit comments or the latest build failure notification over email/sms/build history.
- If you check out the broken code time is wasted on something wrongly done!

# CI Test Automation

35

- **Write automated unit and integration tests.**
- A build should be fully automated. In order to run tests for a CI system, the tests must be automated.
- Writing the tests in an xUnit framework such as NUnit or JUnit provides the capability of running these tests in an automated fashion with ant or certain build scripts.
- **Automate As Much As Possible**

# Ensure Success at every stage

36

- **All tests and inspections must pass**
- In a CI environment, 100% of a project's automated tests must pass for the build to pass.
- Automated tests are as important as the compilation. The code that does not compile does not work similarly the code that has test errors will not work either.
- The code coverage tools assist in pinpointing source code that does not have a corresponding test. You can run a code coverage tool as part of an integration build.

# Code Ownership

37

- **Collective Ownership of the Code**
- The CI allows to share the code ownership.
- Any developer can work on any part of the software system.
- This prevents “knowledge islands,” where there is only one person who has knowledge of a particular area of the system.
- The practice of CI enables for everyone to have the collective ownership by ensuring adherence to coding standards and the running of regression tests on a continual basis.

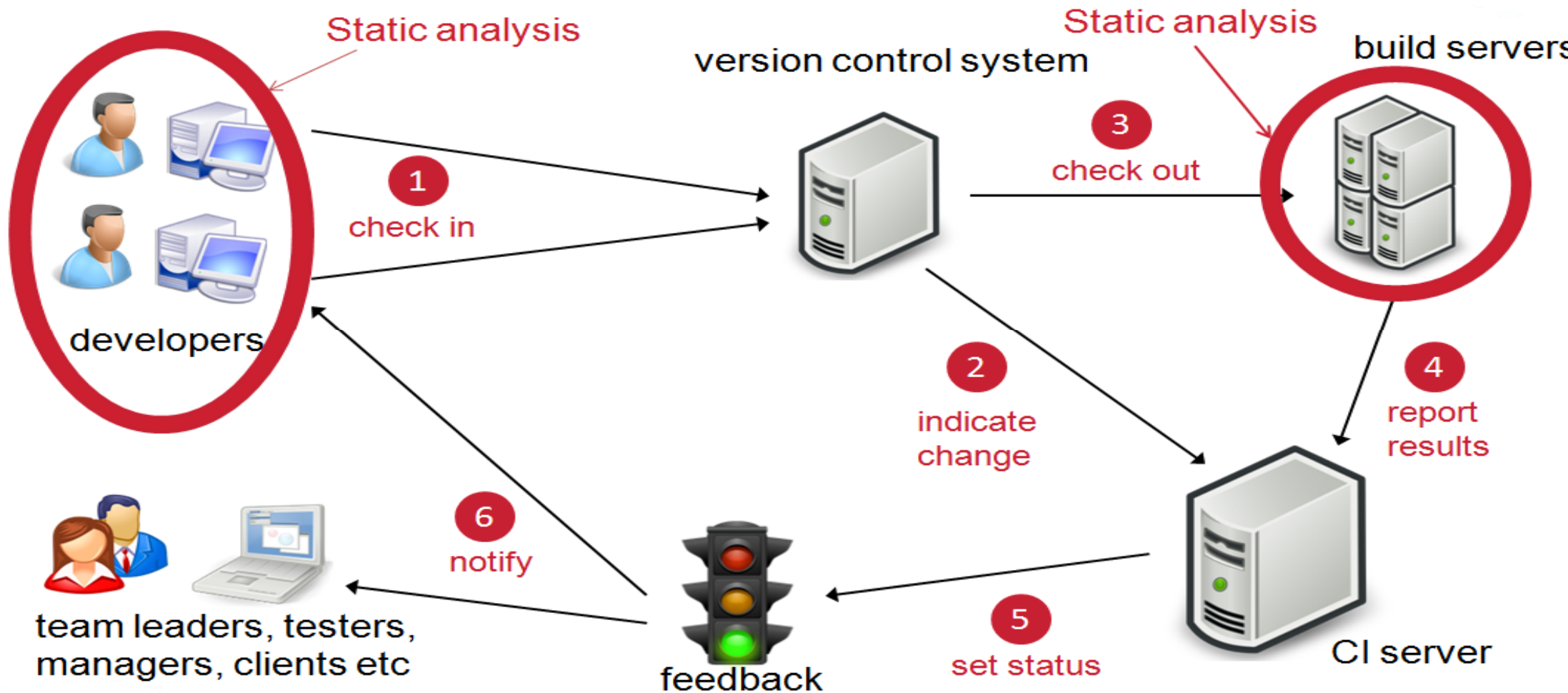
# Build with self-tests

38

- System Tests
  - End-to-end test
  - Often take minutes to hours to run
- Unit tests
  - Fast
    - No database or file system
  - Focused
    - Pinpoint problems
  - Best method for verifying builds

# The CI Phases

39



# The CI build server

40

- The build server can automatically run the unit tests periodically or even after every commit and report the results to the developers.
- The use of build servers (not necessarily running unit tests) had already been practised by some teams from the XP community.
- The CI server apart from the automated testing also integrates other aspects of software such as code analysis, report generation, measure and profile performance, extract and format documentation from the source code etc.



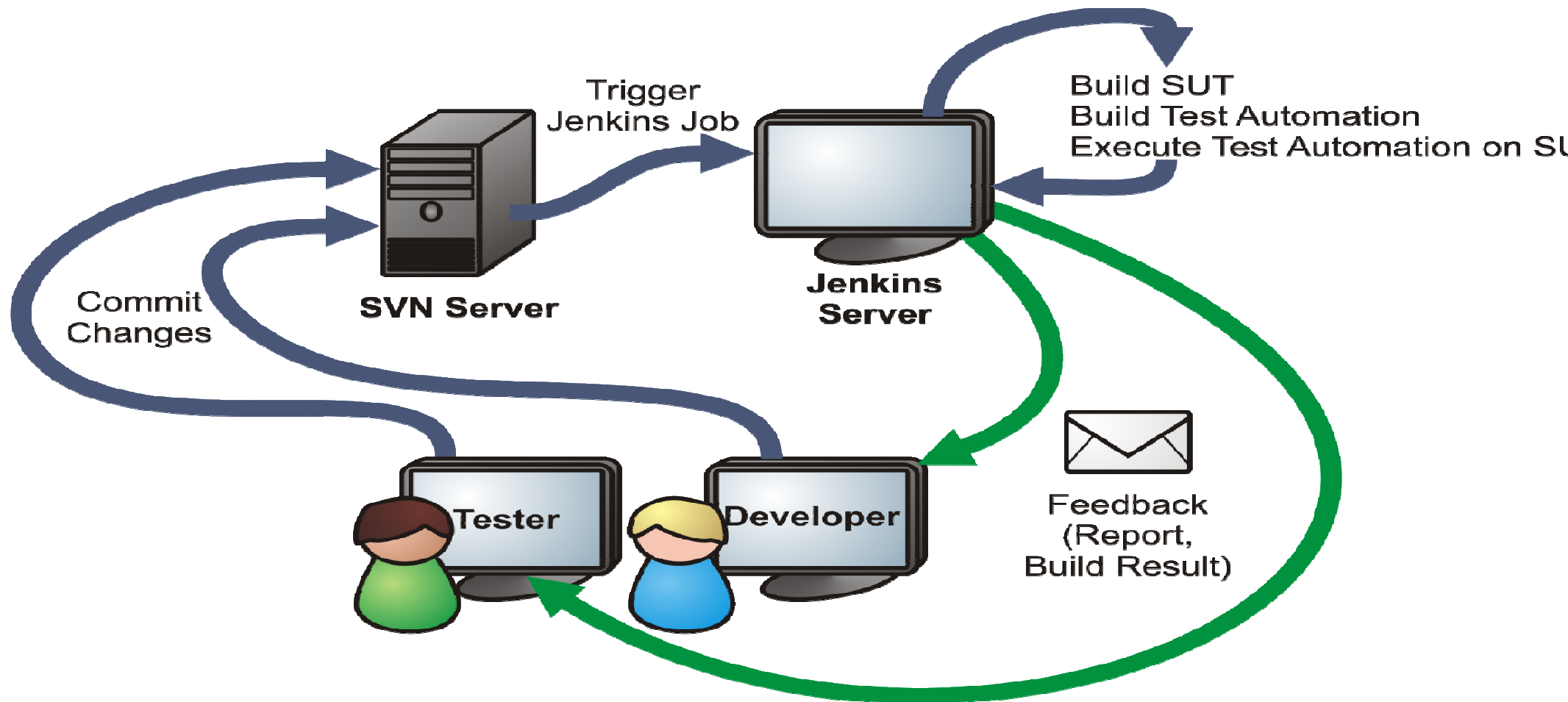
# The CI servers

41

- Cruise Control
- Hudson
- Jenkins
- Bamboo
- Microsoft Team Foundation
- ThoughtWorks GO!
- Apache Continuum

# The Jenkins CI Server

42



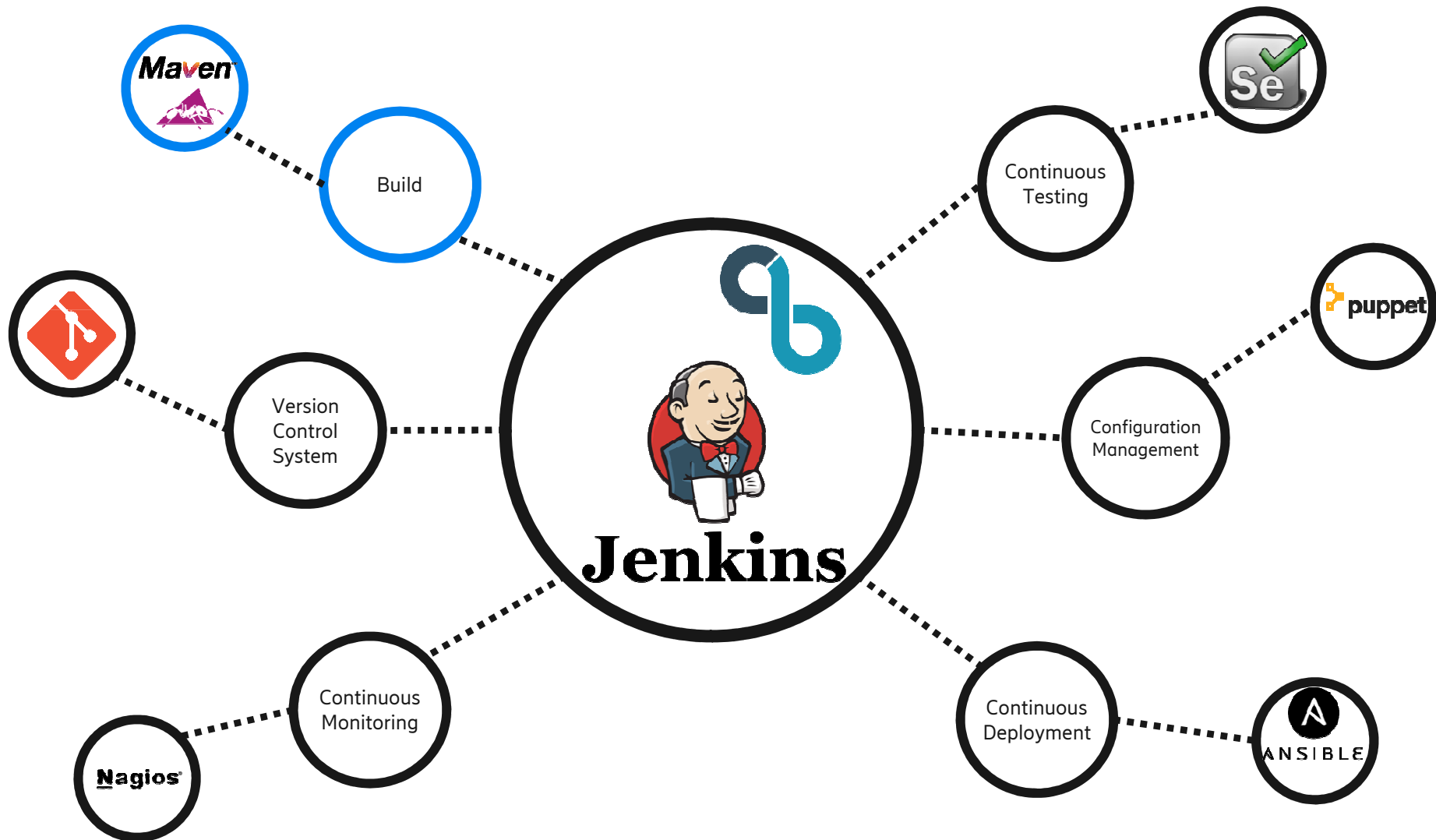
# Jenkins Features

43

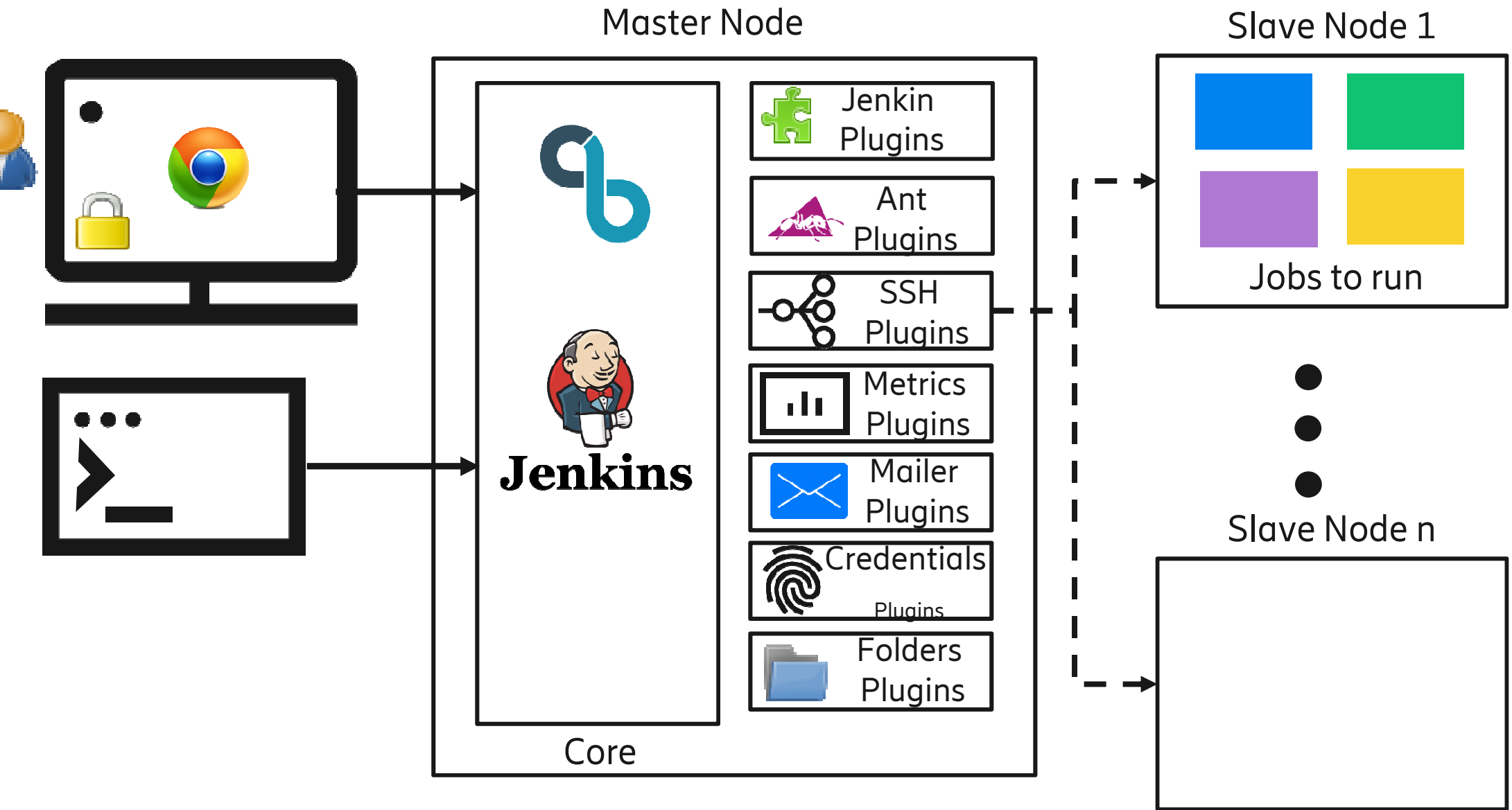
- Open source cloudBees platform.
- Forked from Hudson.
- Easier integration with other tools.
- More extensions available.
- Applicable for Non java builds also.

# About Jenkins

Jenkins is an automation server, that is mainly used in CI/CD cycles.



# Jenkins Architecture



# Jenkins Usage

46

- The Jenkins comes as a war file which be deployed on any J2EE compatible Web Server.(i.e. Tomcat)
- The Jenkins war can be executed as standalone web server.(The Jenkins own server process)
- The Jenkins can also be deployed as service on windows/Unix systems.

# Jenkins configuration

47

- The Jenkins is managed from the web console in the browser.
- Add plug-ins
- Configure global tools JDK ,ant, Maven etc.
- Configure security
- Configure email notifications and settings
- Create and Configure build jobs with Jenkins in console.
- The every jobs is configured with set of options in build stage along with notifications and report generations.

# Login Page

48



**Welcome to Jenkins!**

Sign in

☐ Keep me signed in



# Manage Jenkins

New Item

People

Build History

Project Relationship

Check File Fingerprint

**Manage Jenkins**

Support

My Views

Credentials

New View

**Build Queue**

No builds in the queue.

**Build Executor Status**

**master**

1 Idle

2 Idle

**Slave-Node1**

1 Idle

2 Idle

3 Idle

4 Idle

5 Idle

## Manage Jenkins



### Configure System

Configure global settings and paths.



### Configure Global Security

Secure Jenkins; define who is allowed to access/use the system.



### Beekeeper Upgrade Assistant

Manage Beekeeper Upgrade Assistant, part of the CloudBees Assurance Program



### Configure Credentials

Configure the credential providers and types



### Global Tool Configuration

Configure tools, their locations and automatic installers.



### Reload Configuration from Disk

Discard all the loaded data in memory and reload everything from file system. Useful when you modified config files directly on disk.



### Manage Plugins

Add, remove, disable or enable plugins that can extend the functionality of Jenkins.



### System Information

Displays various environmental information to assist trouble-shooting.



### System Log

System log captures output from `java.util.logging` output related to Jenkins.



### Load Statistics

# Configure Jenkins

## Mask Passwords - Global Regexes

Add



# of executors	2	←	Number of executors
Labels			
Usage	Use this node as much as possible		▼ ?
Quiet period	5		?
SCM checkout retry count	0		
Default view	all		▼

Jenkins Location	
Jenkins URL	http://10.67.200.21:8080/ ← URL of Jenkins
System Admin e-mail address	Batch Automation<automation@u.com.my> ?

Shell	
Shell executable	

E-mail Notification ← SMTP Configuration	
SMTP server	192.168.12.26 ?
Default user e-mail suffix	@u.com.my ?



Advanced...

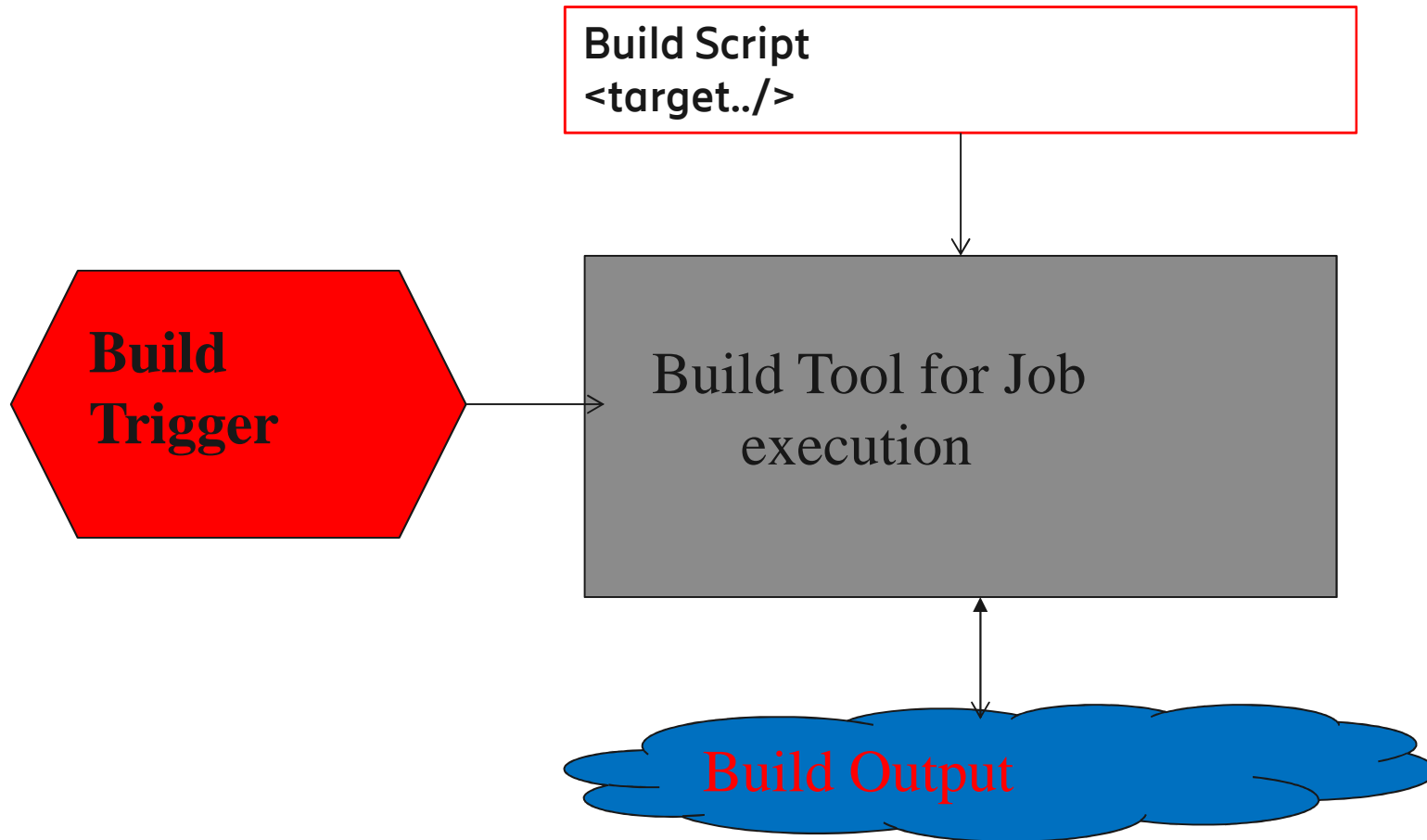
☐ Test configuration by sending test e-mail

# Jenkins Jobs

51

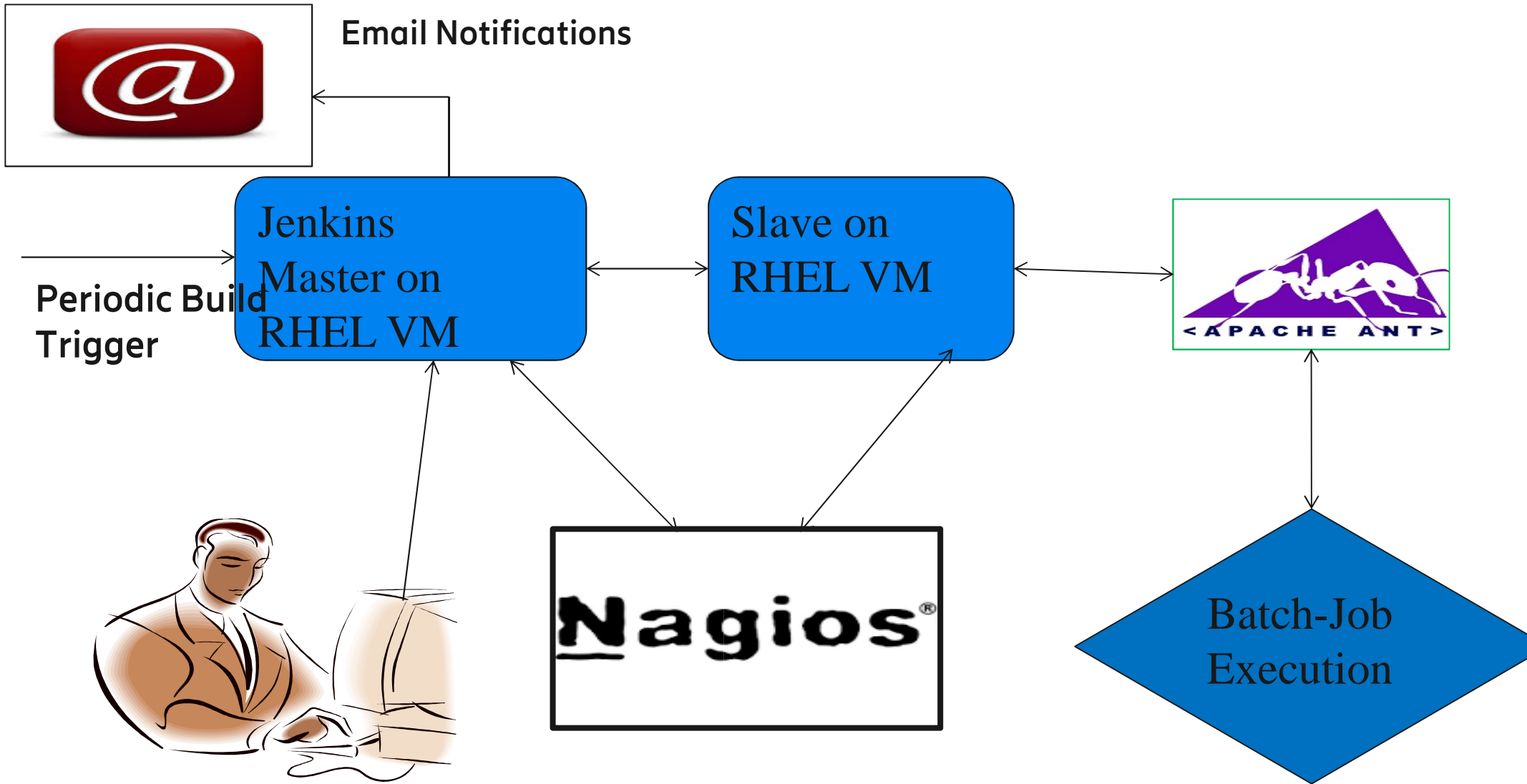
- The jobs are scheduling units for continuous execution process and monitoring in the Jenkins web console.
- Every job is configured with set of options in build stages along with notifications.

# A Job in Jenkins



# Job Automation

53



# Jenkins Jobs

54

- The jobs are execution units for continuous build process and monitoring in the Jenkins web console.
- Type of jobs
  - Freestyle job
  - Maven job
  - Multi-configuration job
  - External job
  - Parameterized job
  - Non-java build such as MS.Net, C+++ etc.

# Job Configuration

55

- The SCM code base location
- The build process trigger
- Set of pre-build testing processes.
- Set of post-build activities for report generation, code analysis etc.
- Notification of build process via email/SMS
- Job execution and monitoring

# Job Configuration

CloudBees Jenkins Distribution

Test Jenkins Administrator | log out

StarterPackHandler\_Lot28

General | Source Code Management | Build Triggers | Build Environment | Build | Post-build Actions

Options for configuration

Description: Starter Pack Handler for Lot 28

[Plain text] [Preview](#)

☒ Discard old builds

Strategy: Log Rotation

Days to keep builds: 180

if not empty, build records are only kept up to this number of days

Max # of builds to keep:

if not empty, only up to this number of build records are kept

[Advanced...](#)

☒ This project is parameterized

**Password Parameter**

Name: ema.password

Default Value: .....

Description: The password for ema user

Passwords are stored as encrypted values and passed jobs as parameters after decryption

☐ Disable this project

☐ Execute concurrent builds if necessary

☒ Restrict where this project can be run

Label Expression: slave-worker

[Label slave-worker](#) is serviced by 1 node. Permissions or other restrictions provided by plugins may prevent this job from running on those nodes.

[Advanced...](#)

Build and Log Retention Policy

to trigger the build



# Job Configuration Continued

## Build Triggers

- ☐ Trigger builds remotely (e.g., from scripts)
- ☐ Build after other projects are built
- ☒ Build periodically

Schedule

H/15 \* \* \* \*

Would last have run at Friday, March 29, 2019 10:40:50 AM MYT; would next run at Friday, March 29, 2019 10:55:50 AM MYT.

☐ Poll SCM

Options to Trigger a Build of Job

## Build

### Invoke Ant

Ant Version

Ant10

Targets

ssh-java

Add build step

Configure Steps involved to trigger a build

## Post-build Actions

### E-mail Notification

Recipients

prem.sureddy@ericsson.com,ks.tan@apis-consulting.com

Whitespace-separated list of recipient addresses. May reference build parameters like \$PARAM. E-mail will be sent when a build fails, becomes unstable or returns to stable.

- ☒ Send e-mail for every unstable build
- ☐ Send separate e-mails to individuals who broke the build

Add post-build action

Build Notification Settings and Post build configurations like triggering dependent Job

# Jenkins Jobs Dashboard

CloudBees Jenkins Distribution 2

Test Jenkins Administrator | log out

Breadcrumbs

Privileged Menu

Views

S	W	Name ↓	Last Success	Last Failure	Last Duration
		Lot17	N/A	N/A	N/A
		Lot28	N/A	N/A	N/A
		W Description	%	1 min 30 sec - #173	4 min 8 sec

Icon: S M L

Worst health: Lot28 » StarterPackHandler\_Lot28: Build stability: 1 out of the last 5 builds failed.

Legend RSS for all RSS for failures RSS for just latest builds

Build Queue

Build Executor Status

Nodes Attached & Executors

Status of the Job

- Red – Failed
- Blue – Success
- Gray – Aborted/Inactive
- Blinking – InProgress

4:51 PM 28/3/2019



5 builds  
success



1 of las  
builds f



2 of las  
builds f



3 of las  
builds f



All Rece  
builds  
failed

# Job Summary View

The screenshot displays the Jenkins Job Summary View for the project 'StarterPackHandler\_Lot28'. The interface includes a top navigation bar with the 'CloudBees Jenkins Distribution' logo, a search bar, and the user 'Test Jenkins Administrator'. The breadcrumb trail shows the path: Jenkins > Lot28 > StarterPackHandler\_Lot28.

**Annotations:**

- Project StarterPackHandler\_Lot28**: Points to the job title and description box.
- Job Administration Options**: Points to the left sidebar menu containing options like Up, Status, Changes, Workspace, Build with Parameters, Delete Project, Configure, Move, and Rename.
- Trends of the builds**: Points to the 'trend' button in the Build History section.
- Build History**: Points to the list of recent builds.
- Permalinks**: Points to the list of links for build details.
- Build Details**: Points to the 'Last build (#31), 5 min 6 sec ago' link.
- Name and Description of Job**: Points to the job title and description box.
- Enable and Disable Jobs**: Points to the 'Disable Project' button on the right.

**Build History Table:**

Build Number	Timestamp
#31	Mar 28, 2019 4:55 PM
#30	Mar 28, 2019 4:40 PM
#29	Mar 28, 2019 4:25 PM
#28	Mar 28, 2019 4:10 PM
#27	Mar 28, 2019 3:55 PM
#26	Mar 28, 2019 3:40 PM

**Permalinks List:**

- Last build (#31), 5 min 6 sec ago
- Last stable build (#31), 5 min 6 sec ago
- Last successful build (#31), 5 min 6 sec ago
- Last failed build (#30), 20 min ago
- Last unsuccessful build (#30), 20 min ago
- Last completed build (#31), 5 min 6 sec ago

# Job Build Details

The screenshot shows the Jenkins interface for a specific build. At the top is a blue header with the Jenkins logo and the text 'CloudBees Jenkins Distribution'. To the right of the header is a search bar and a user profile for 'Test Jenkins Administrator'. Below the header is a breadcrumb trail: 'Jenkins > Lot28 > StarterPackHandler\_Lot28 > #102'. On the left side, there is a sidebar with a list of links: 'Back to Project', 'Status', 'Changes', 'Console Output', 'Edit Build Information', 'Delete build '#102'', 'Parameters', and 'Previous Build'. The main content area displays 'Build #102 (Mar 29, 2019 9:55:00 AM)' in a large box. Below this, there are three status indicators: 'No changes.' (with a notepad icon), 'Started by timer' (with a timer icon), and 'This run spent:' (with a clock icon). The 'This run spent:' section is expanded, showing a list of statistics: '22 ms waiting in the queue;', '9.6 sec building on an executor;', and '9.6 sec total from scheduled to completion.'. On the right side, there is a button 'Keep this build forever' and a section showing 'Started 14 min ago' and 'Took 9.6 sec on Slave-N'. There are three arrows pointing to specific parts of the interface: one from the text 'Build Administration Options.' pointing to the sidebar, one from 'Build Date and Time' pointing to the build title box, and one from 'Build/Run Statistics' pointing to the expanded run time statistics box.

CloudBees Jenkins Distribution

search

Test Jenkins Administrator

Jenkins > Lot28 > StarterPackHandler\_Lot28 > #102

ENABLE AUTO-RUN

Back to Project

Status

Changes

Console Output

Edit Build Information

Delete build '#102'

Parameters

Previous Build

**Build #102 (Mar 29, 2019 9:55:00 AM)**

Keep this build forever

Started 14 min ago

Took 9.6 sec on Slave-N

[add description](#)

No changes.

Started by timer

This run spent:

- 22 ms waiting in the queue;
- 9.6 sec building on an executor;
- 9.6 sec total from scheduled to completion.

Build Administration Options.

Build Date and Time

Build/Run Statistics

# Job build activities

61

- Ant build.xml is executed in Jenkins environment.
- Maven job is built with maven dependencies managed.
- Shell executable run in Jenkins environment.
- MSBuild is triggered for MS.Net projects.
- Unit tests are executed.
- Code analysis tools are executed.
- Application is built and deployed.
- Groovy Script is executed as part of build process.
- Application is deployed.
- Integration test tools like selenium is invoked.
- Email is generated and sent as part of build result.
- Code quality and test report generation is done.
- Any custom task is executed.

# Application Deployments

62

- Deploy the successfully built code only, no developer builds
- One click deploy from Jenkins
- Deploy code first to staging environment then production
- Few deployment defects since adopting this method.

# Code maintenance

63

- Individual programmers <50% efficient at finding their own bugs
- Multiple quality methods = more defects discovered
  - Use 3 or more methods for >90% defect removal
- Most effective methods
  - design inspections
  - code inspections
  - Testing

# Code analysis tools for java

64

- Checkstyle
- PMD
- FindBugs
- Sonar
- Application code coverage with Cobertura
- Test coverage tools



# Sonar with Jenkins

65

- Jenkins builds the code
- SONAR runs after each build
- SONAR alert thresholds can 'break' the build
- Automate quality improvement processes
- SonarQube server in standalone mode
- SonarRunner
- Sonar Jenkins plugin

# Sonar Code metrics

66

- Sonar is the shared central repository for quality management with code metrics for the following...
  - Code Duplications
  - Coding standards
  - Unit tests coverage
  - Complex code
  - Potential bugs
  - Comments
  - Design and architecture

# Jenkins High Availability

67

## Bounce Back Faster: High Availability



3:19 / 5:49



# Jenkins Cluster

68

- No of Jenkins node machines are configured in the network.
- The build job work is shared/distributed across the nodes by load balancer, depending upon the availability of the node machine.
- Whenever one of the current job executing node fails, other node takes over the job and build process continues.

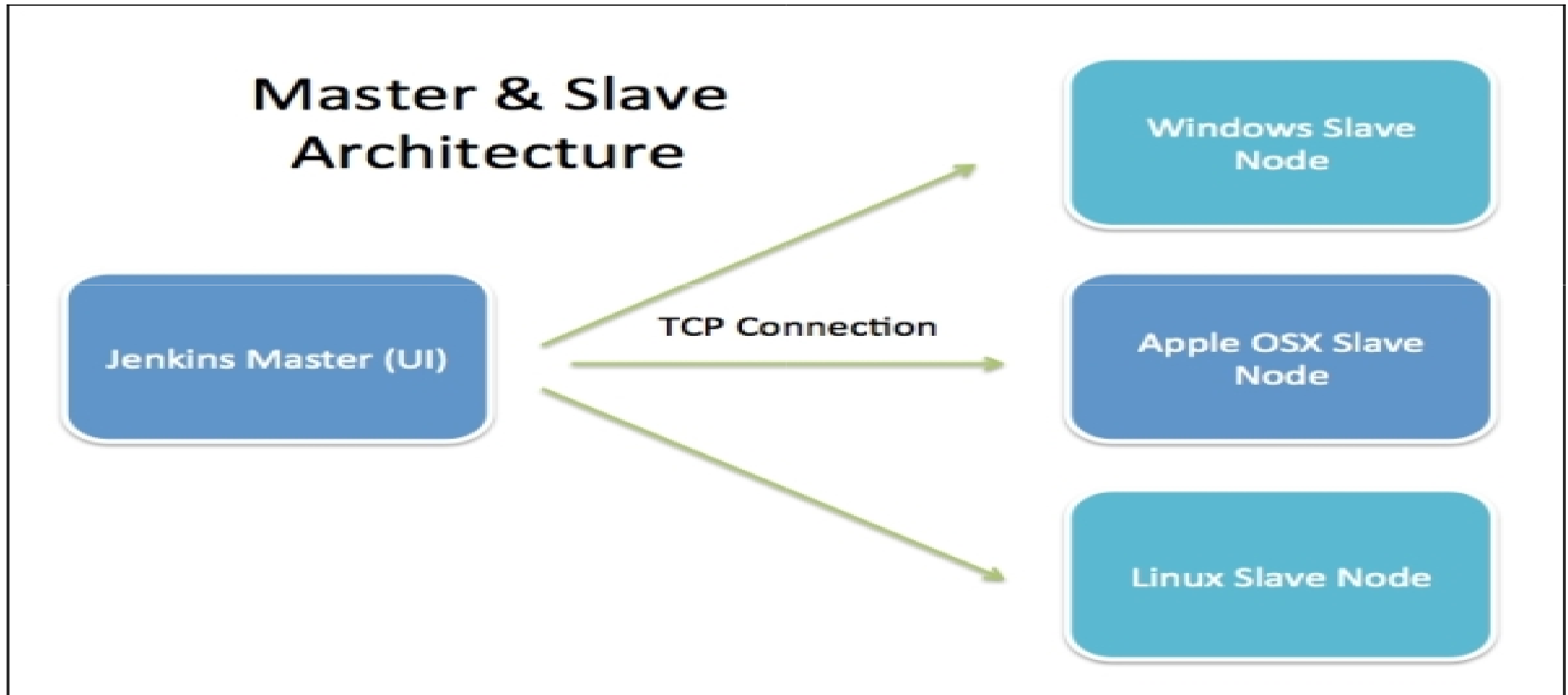
# Distributed build

69

- The master node is the main configuration node having Jenkins instance running.
- The build is distributed/restricted across slave nodes.
- The slave node doesn't need Jenkins to be installed, only java is needed.
- The master node delegates the build to slave node.
- The overhead on the master is reduced.

# Jenkins in Distributed Mode

70



# Master and Slave Nodes

- The Master node has entire configuration of the Jenkins and jobs.
- The Master node schedules the job build to execute on Slave node.
- The Master-slave job distribution reduces the burden on Jenkins Master.
- Multiple Slave nodes can be added
- Multiple jobs in parallel execute quicker.

# Distributed Jobs

72

- The overhead on the master is reduced.
- Whenever one of the current job executing node fails, job starts on other available node and job process continues.



# Configuring Slave Node

CloudBees Jenkins Distribution

Q search

Test Jenkins Administrator | log

Jenkins > Nodes >

Back to Dashboard

Manage Jenkins

New Node

Configure

Build Queue

No builds in the queue.

Build Executor Sta

master

1 Idle

2 Idle

Slave-Node1

1 Idle

2 Idle

3 Idle

4 Idle

5 Idle

S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	<a href="#">master</a>	Linux (amd64)	In sync	93.13 GB	12.00 GB	9.15 GB	0ms
	<a href="#">Slave-Node1</a>	Linux (amd64)	In sync	8.66 GB	12.00 GB	9.15 GB	7ms
Data obtained		5 min 12 sec	5 min 12 sec	5 min 12 sec	5 min 12 sec	5 min 12 sec	5 min 12 sec

Refresh stat

Name

Slave-Node1

Description

Salve Worker Node on Test VM

# of executors

5

Remote root directory

/home/batchuser/slave-node

Labels

slave-worker

Usage

Only build jobs with label expressions matching this node

Launch method

Launch slave agents via SSH

Host

10.67.200.21

Credentials

batchuser (User created for Salve Node SSH)

Host Key Verification Strategy

Known hosts file Verification Strategy

Availability

Keep this agent online as much as possible

Advanc

# Security Configuration

Jenkins ▾

Configure Global Security

Configure Global Security

☒ Enable security

Disable remember me ☐

Access Control

Security Realm

☐ Delegate to servlet container

☒ Jenkins' own user database

☐ Allow users to sign up

Authorization

☐ Anyone can do anything

☐ Legacy mode

☐ Logged-in users can do anything

☒ Matrix-based security

User/group	Overall	Support	Credentials	Agent	Job	Run	View	SCM	Metrics
	Administer	Read DownloadBundle	Create Delete Update ManageDomains View	Build Configure Connect Create Delete Disconnect	Cancel Configure Create Delete Discover Move Read Workspace	Delete Update Configure	Create Delete Read Tag	HealthCheck ThreadDump	View
Anonymous Users	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Authenticated Users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Test Jenkins Administrator	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Donny Lee Foo Yan	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Add user or group...

☐ Project-based Matrix Authorization Strategy

— Security strategies and Role based access to users.

# Continuous Delivery

- When the integration and build process is completed successfully, the final product release artifact is deployed on production server.
- This deployment and delivery process is automated with Jenkins.
- This is Continuous Delivery management.
- Automated with tools like Apache Ant, Maven , MSBuild etc.

# Jenkins with Docker and Kubernetes

- With plug-fins and other tools, Jenkins supports integration with Docker and Kubernetes environment.
- When the final build is successful, the product image is automatically built and pushed to shared registry like docker-hub and then the docker is instructed to create new containers based on the pulled updated images .from registry.
- The cluster with Kubernetes gets updated with new product delivery aspects.

*Thank You!*