Introduction to Jenkins CI Server

Prakash Badhe
Prakash.badhe@vishwasoft.in

Application Build Process

- Developers in the team write code and verify the functional behavior.
- 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 ?

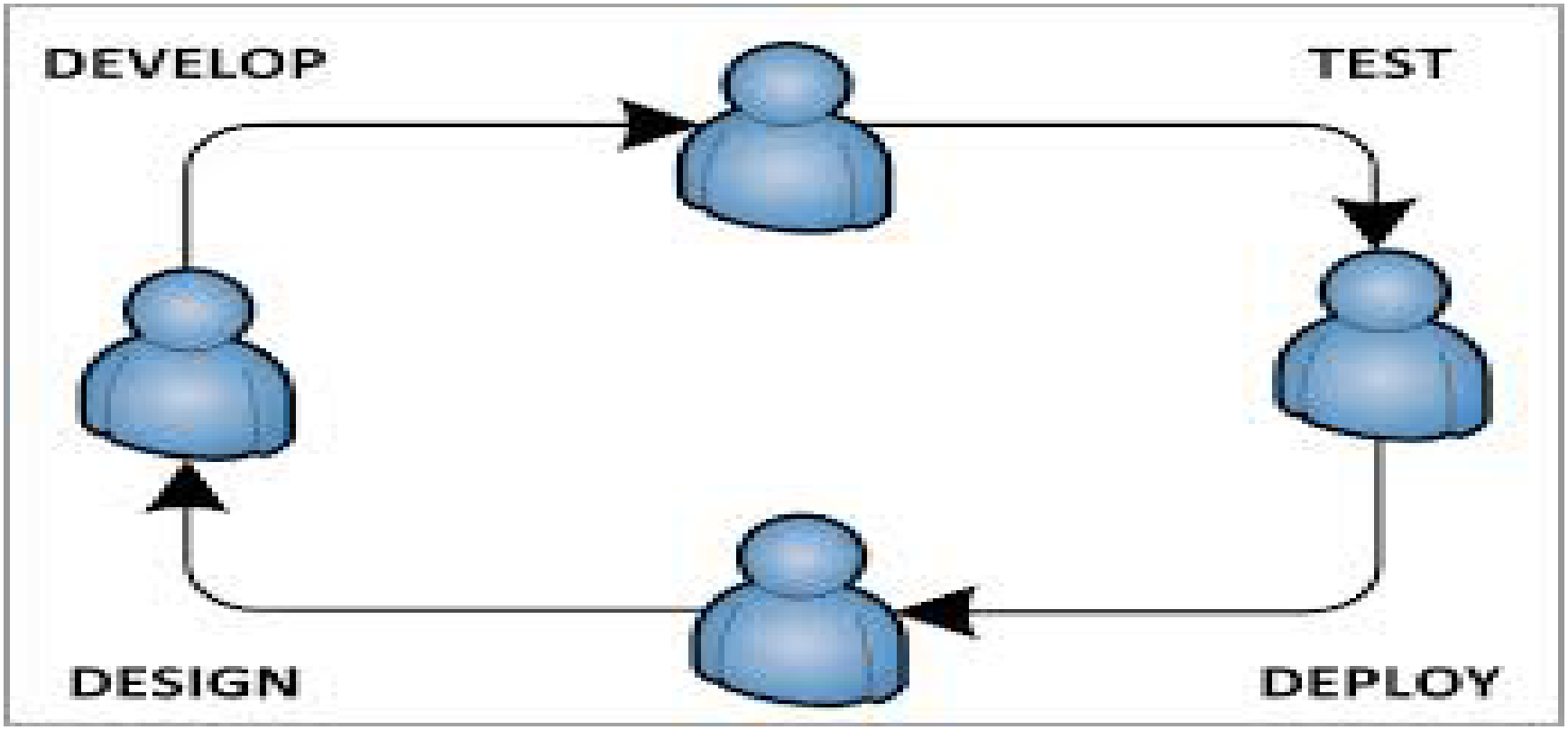
Traditional Project Cycle

— 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

— 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



Build Automation with Apache Ant and Maven

- 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

Development Process

— **T**est **D**riven **D**evelopment.
— 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

— 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

— 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

— 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.

SCM with GIT/SVN/CC

- 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

- 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.

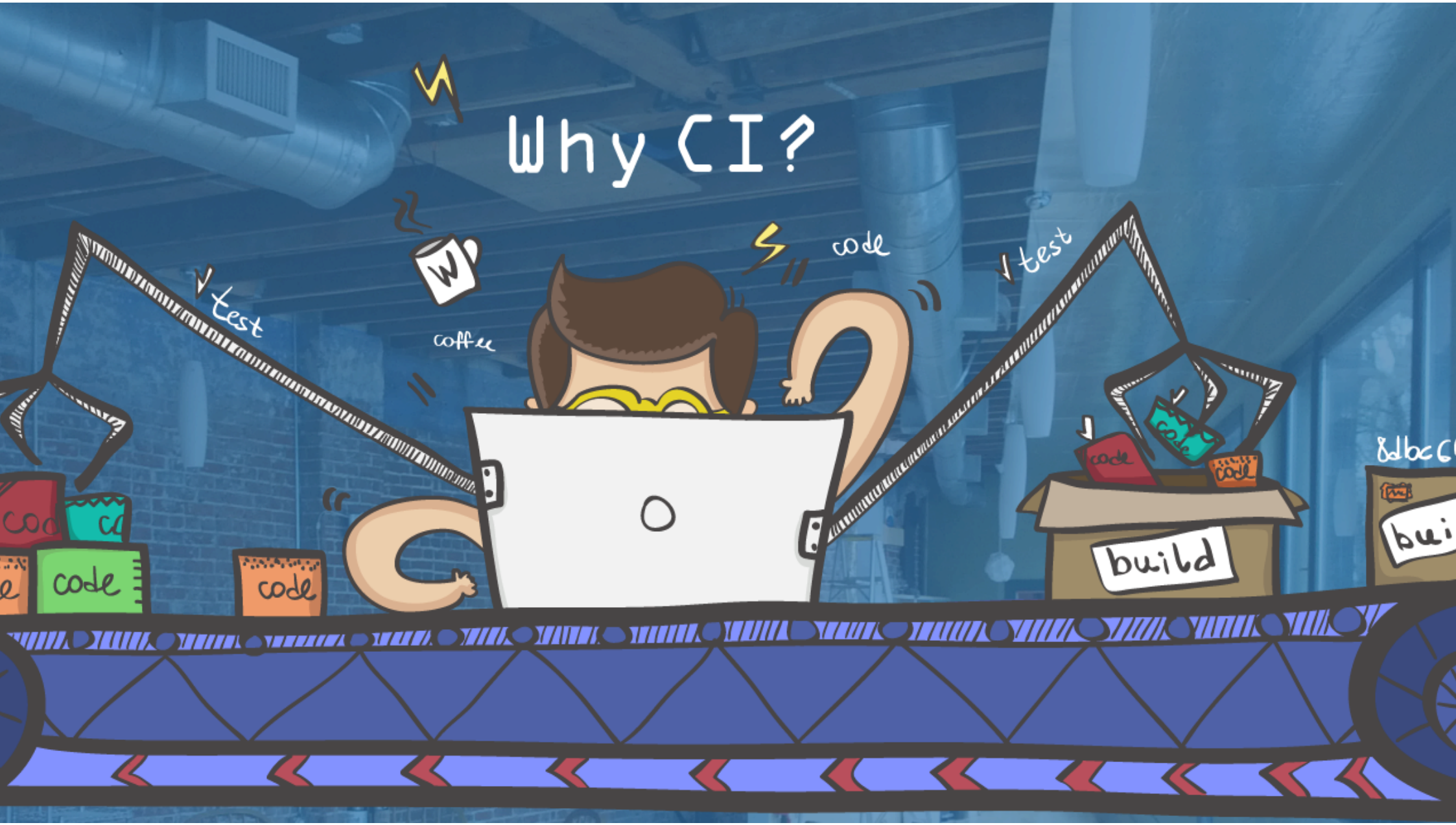
Build Automation Continuously..

- 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

The CI Process

- The **Continuous integration** (**CI**) in software build is the process of merging all developer working copies with a shared mainline several time 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.

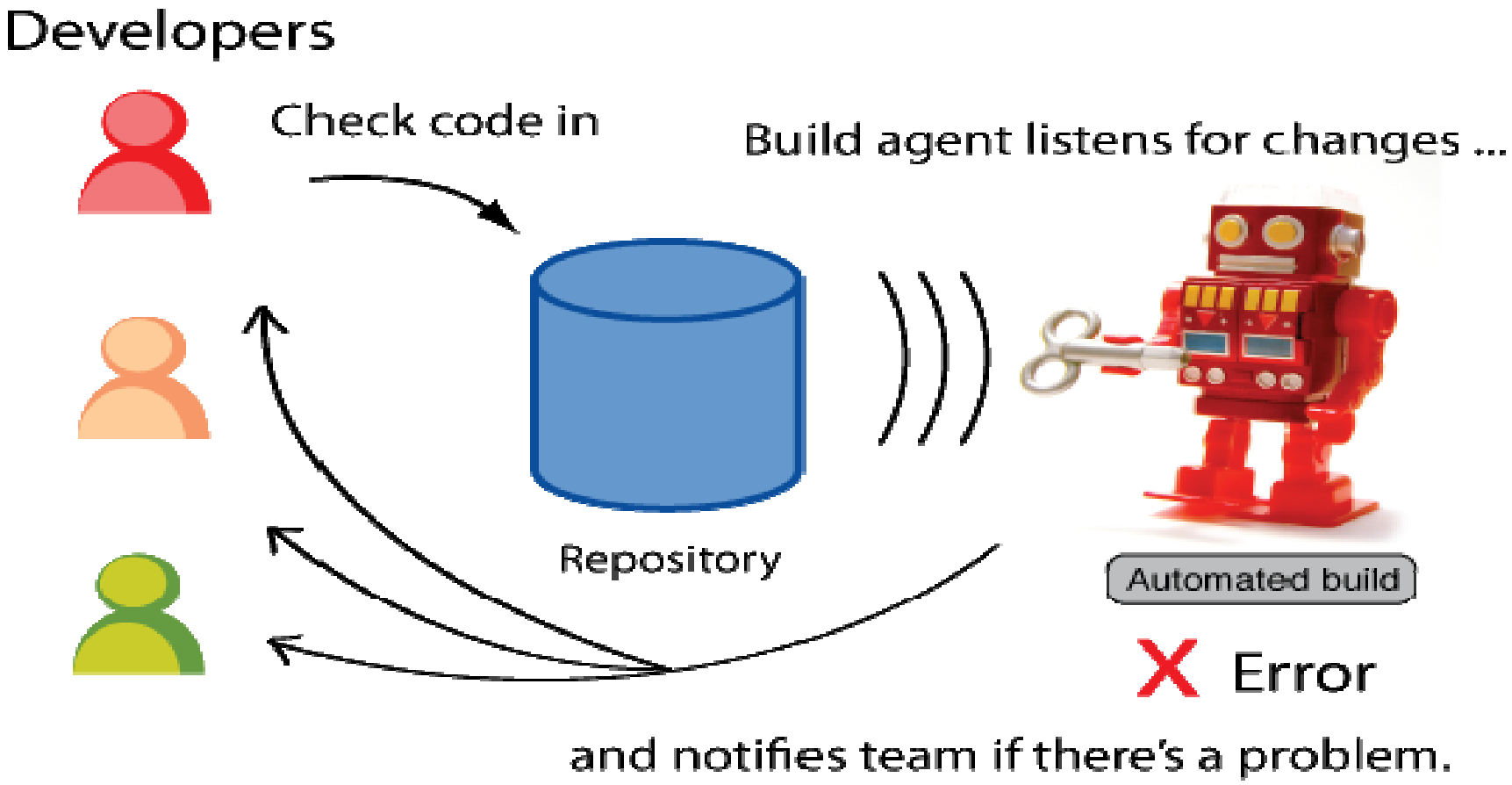

Why CI?
code
test
coffee
test
code
code
code
build

Process Automation Continuously..

- 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

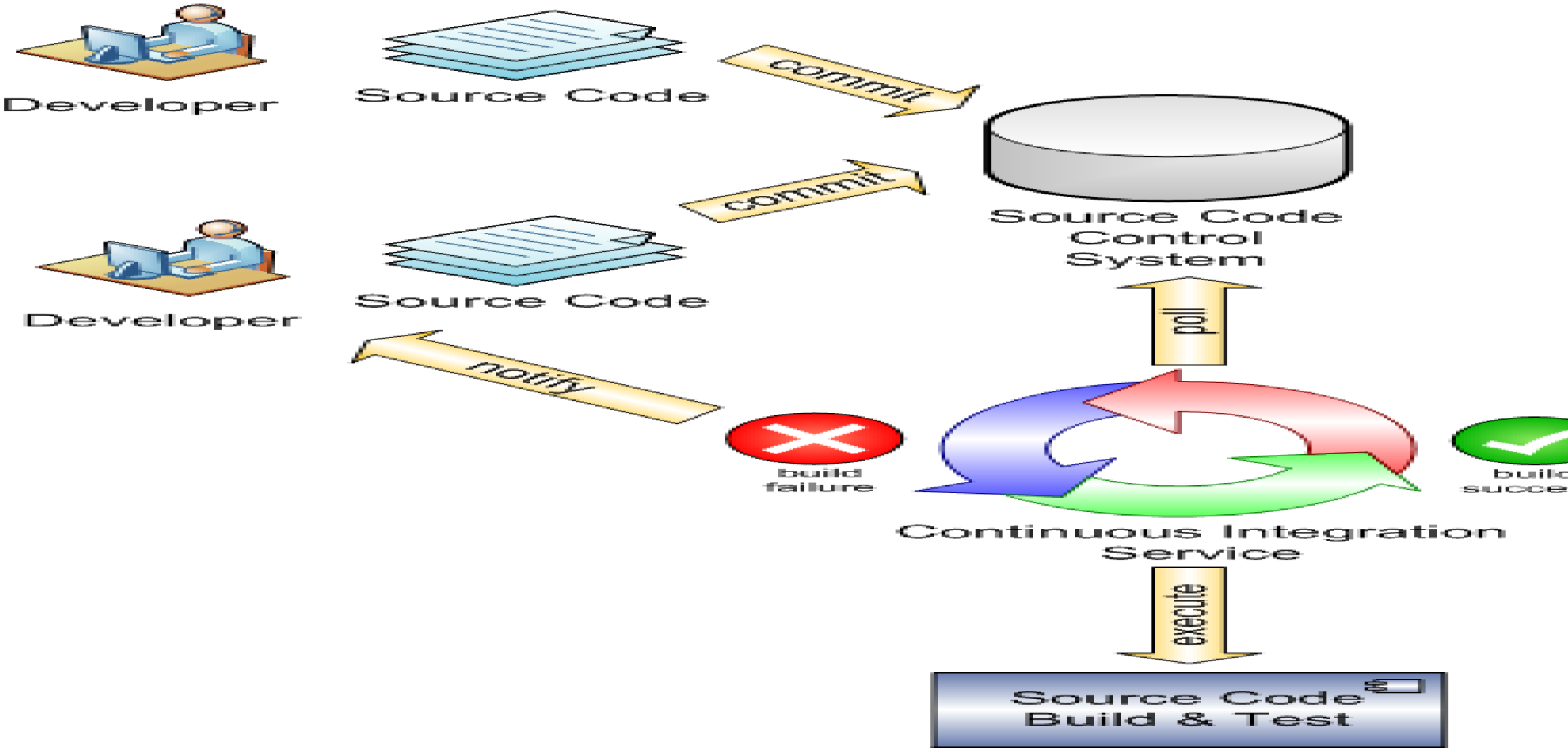
The CI Process…



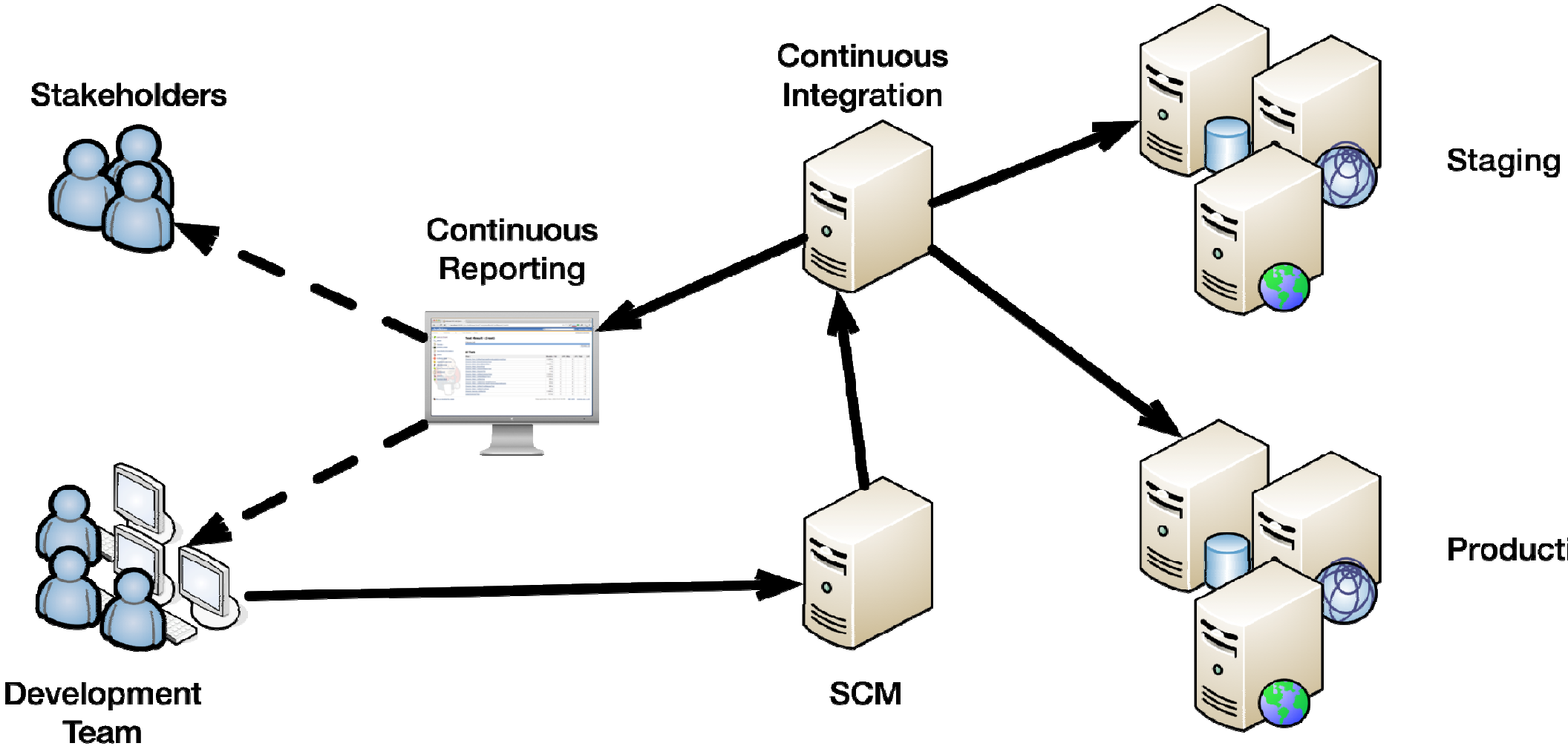
Continuous Integration

- 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



Continuous Integration Workflow



CI Features

- 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

— 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

— 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

— 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

- 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

- 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

- 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

- 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

- **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

— 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

- 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

- 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

- 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 ?

— **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 test code in a repeatable manner on **developer machine!!**..

Private Builds ?

- **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 ?

- **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

- **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

- **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.

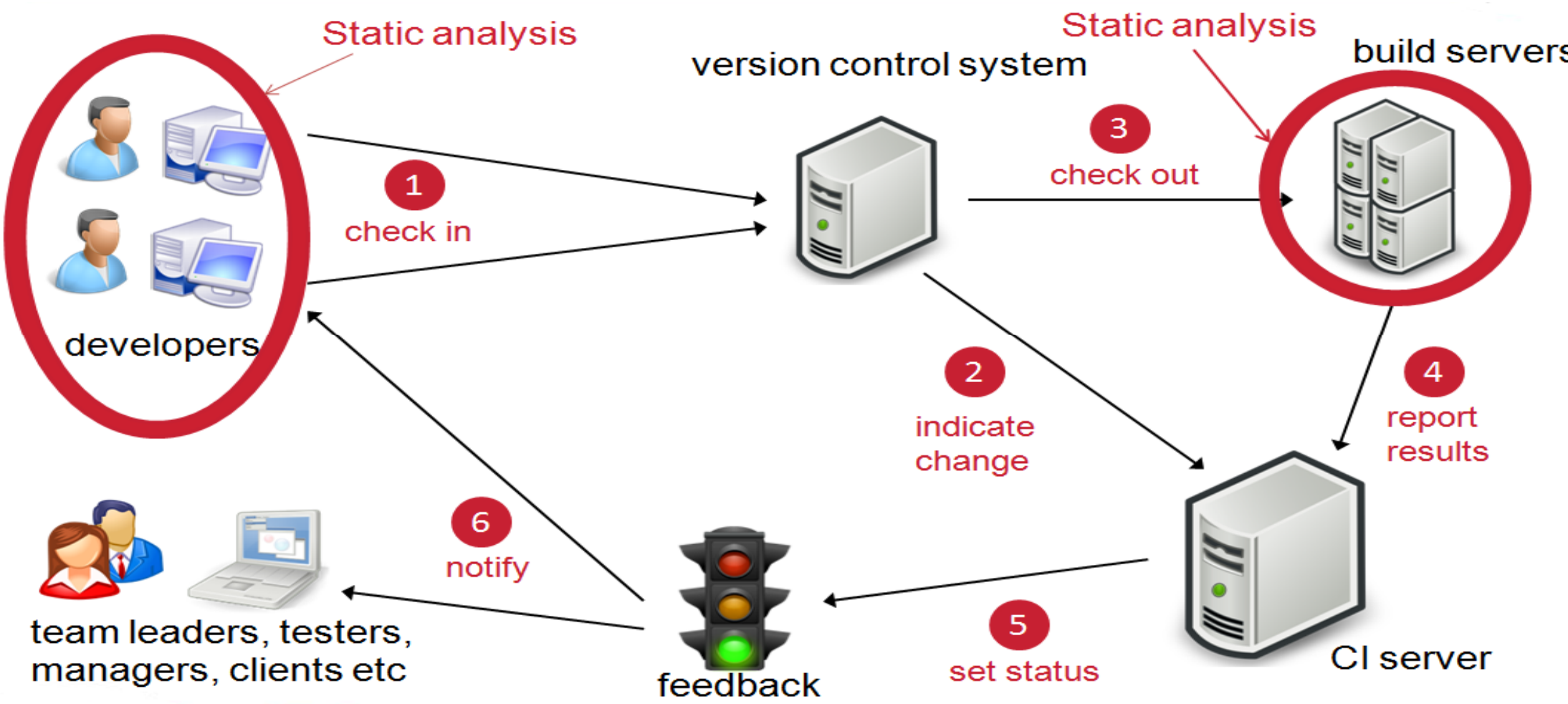
Code Ownership

- **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

- 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 Continuous Build Process



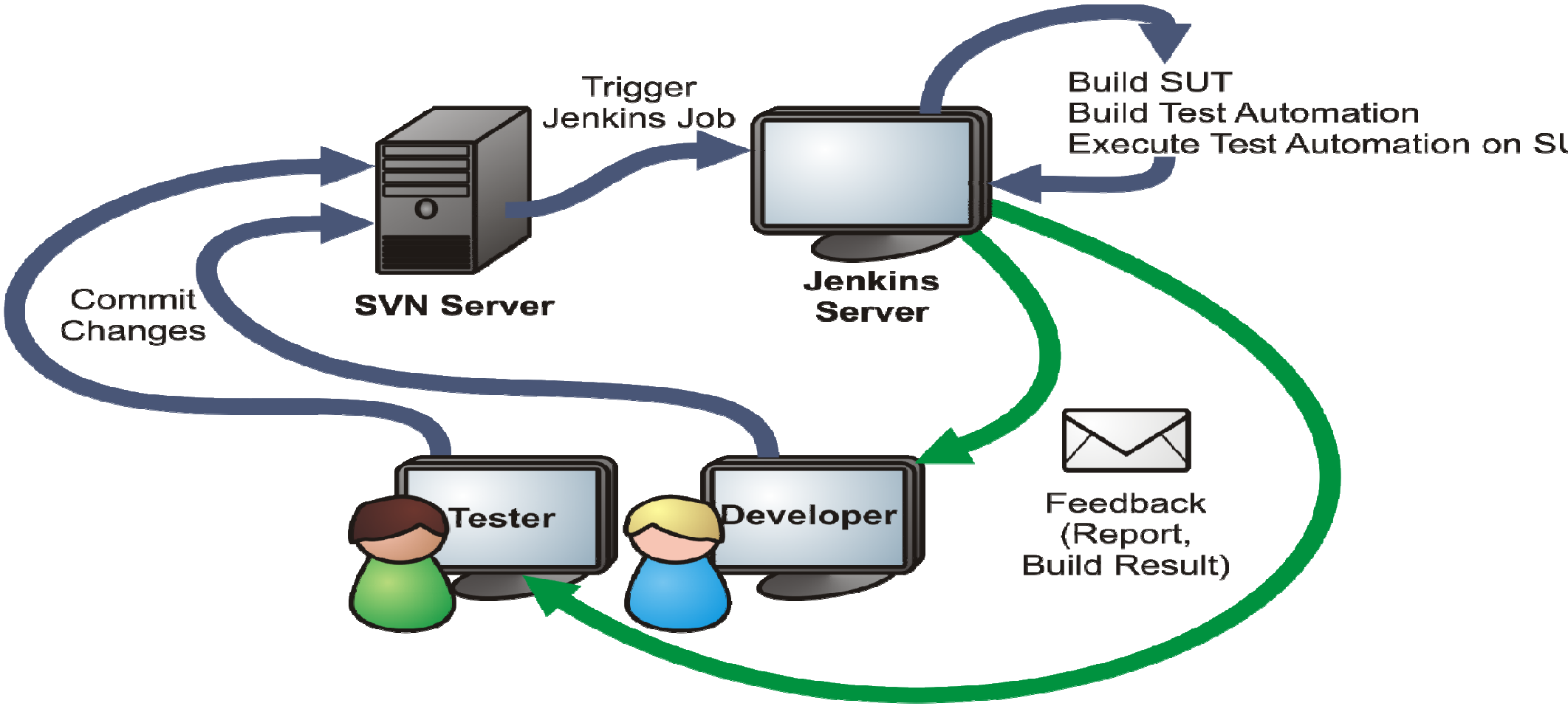
The CI build server

- 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

— Cruize Control
— Hudson
— Jenkins
— Bamboo
— Microsoft Team Foundation
— ThoughtWorks GO!
— Apache Continuum

The Jenkins CI Server

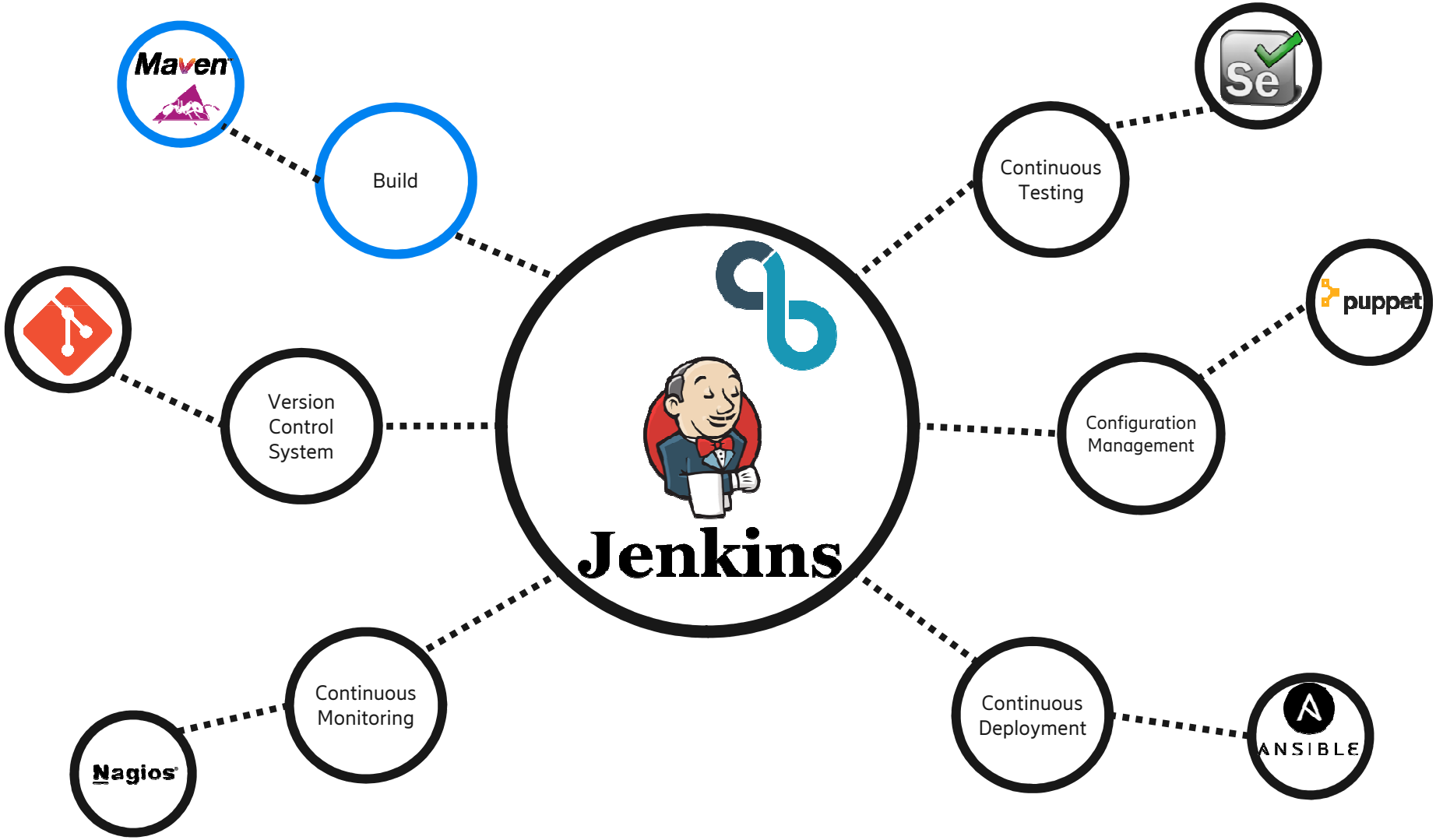


Jenkins Features

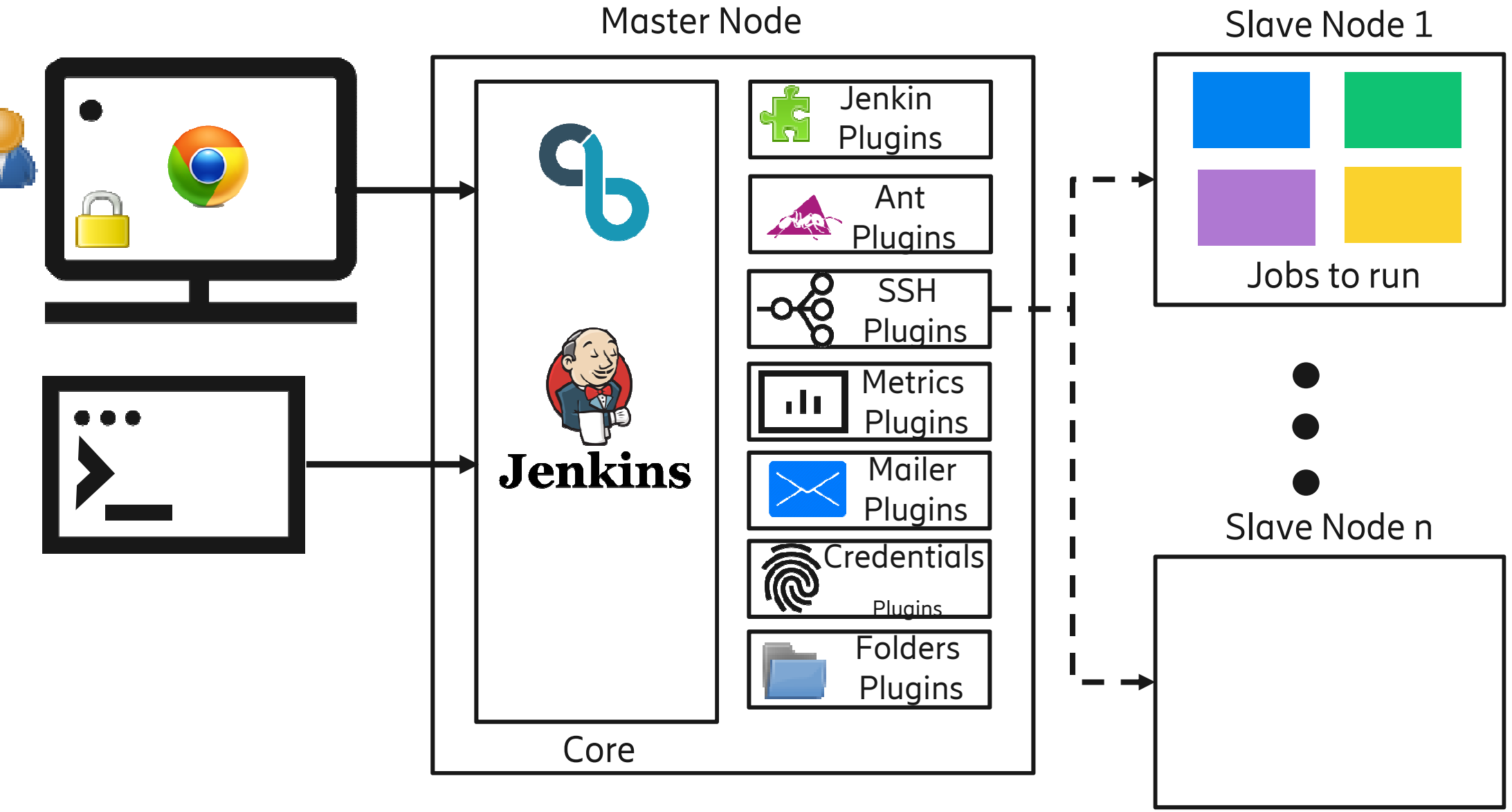
— Open source cloudBees platform.
— Forked from Hudson.
— Easier integration with other tools.
— More extensions available.
— Useful for Non java builds.

About Jenkins

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



Jenkins Architecture



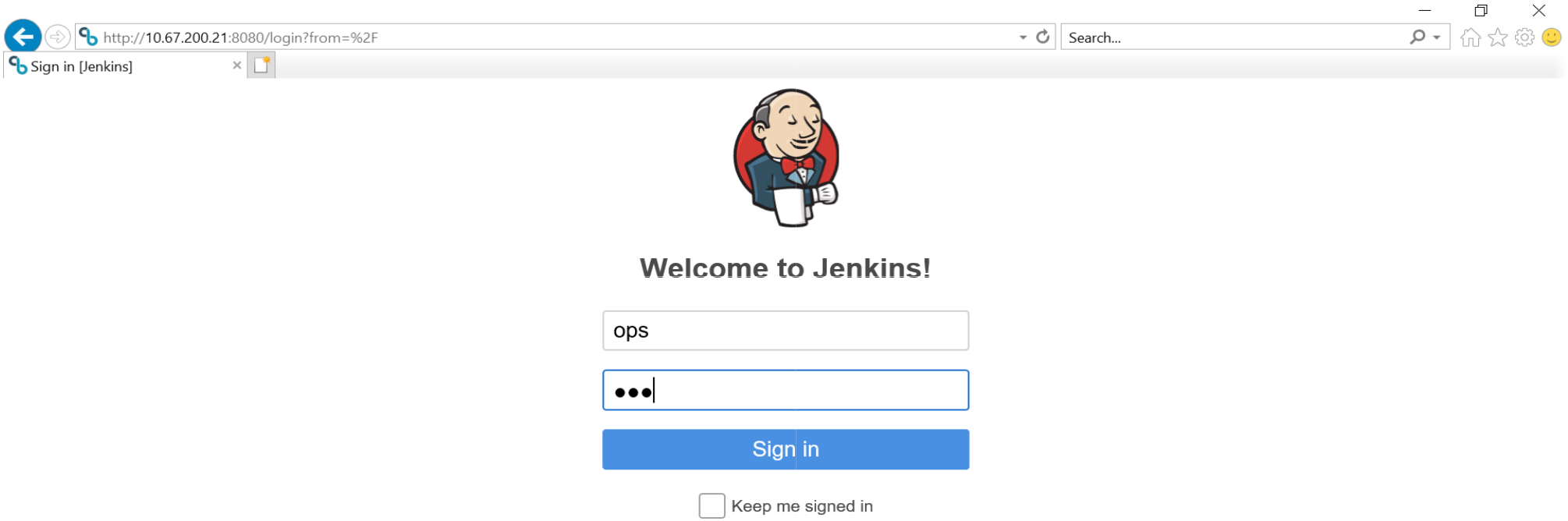
Jenkins Usage

- 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.

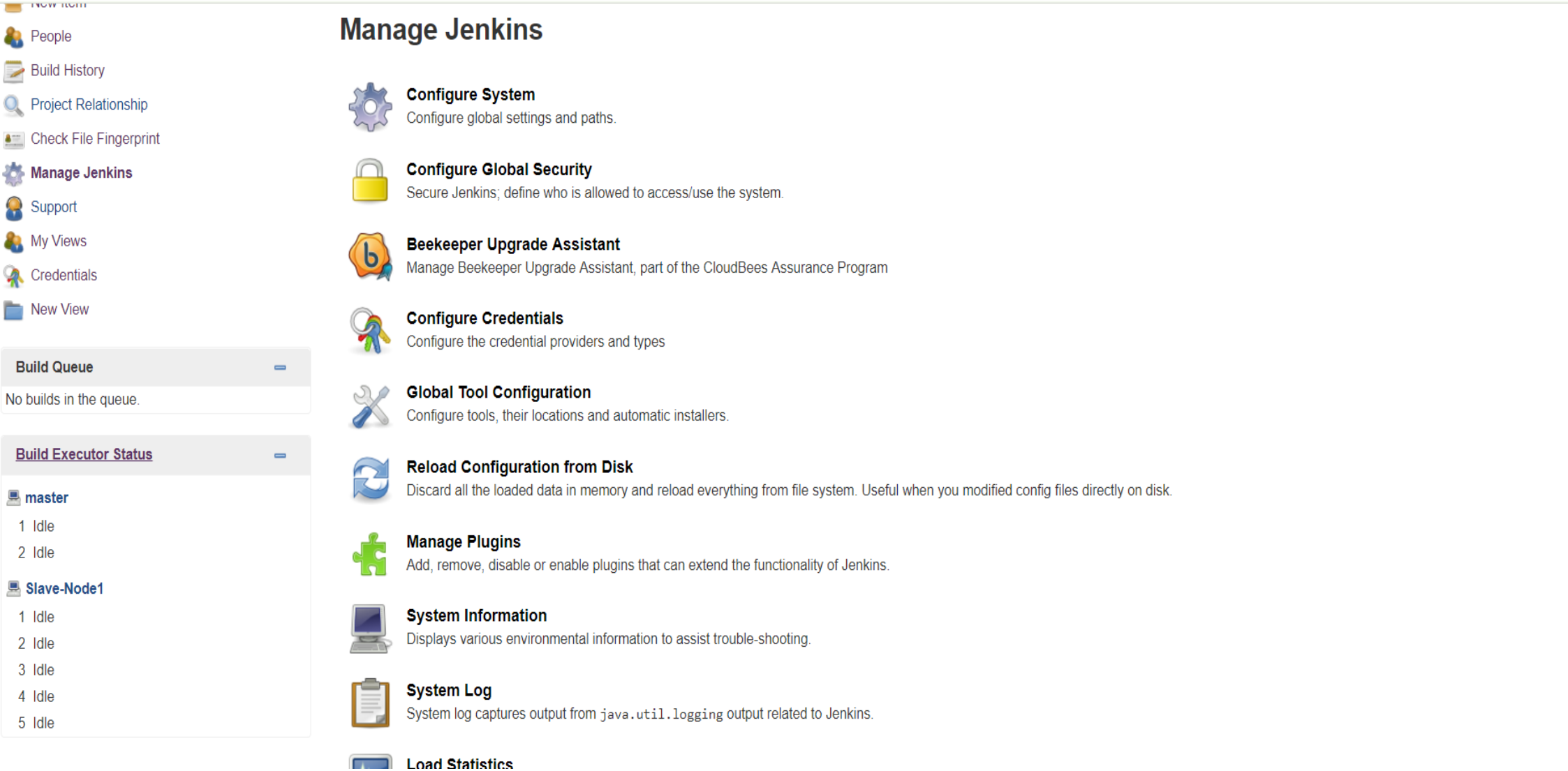
Jenkins configuration

- 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.

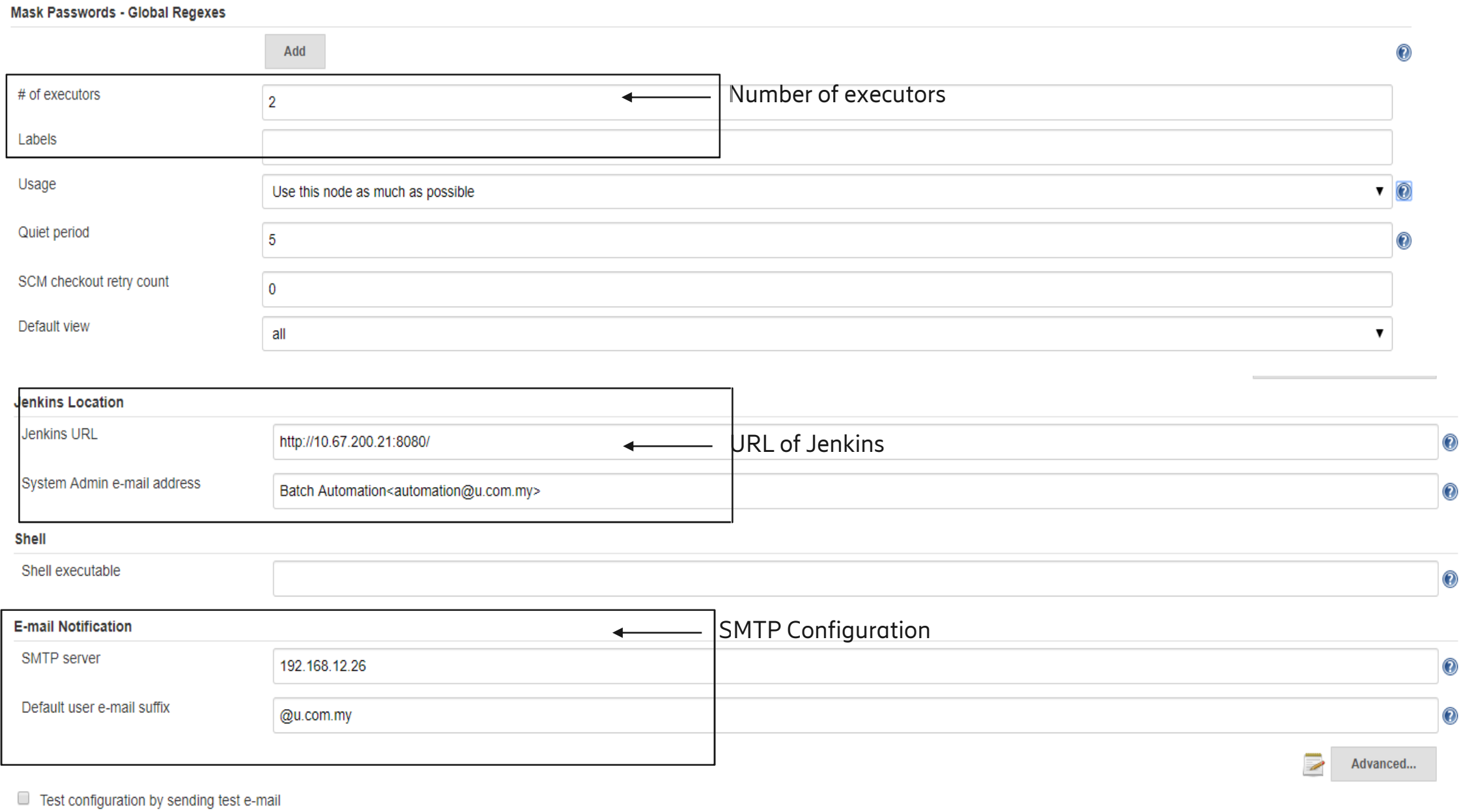
Login Page


http://10.67.200.21:8080/login?from=%2F
Search...
Sign in [Jenkins]
Welcome to Jenkins!
ops
Sign in
Keep me signed in

Manage Jenkins



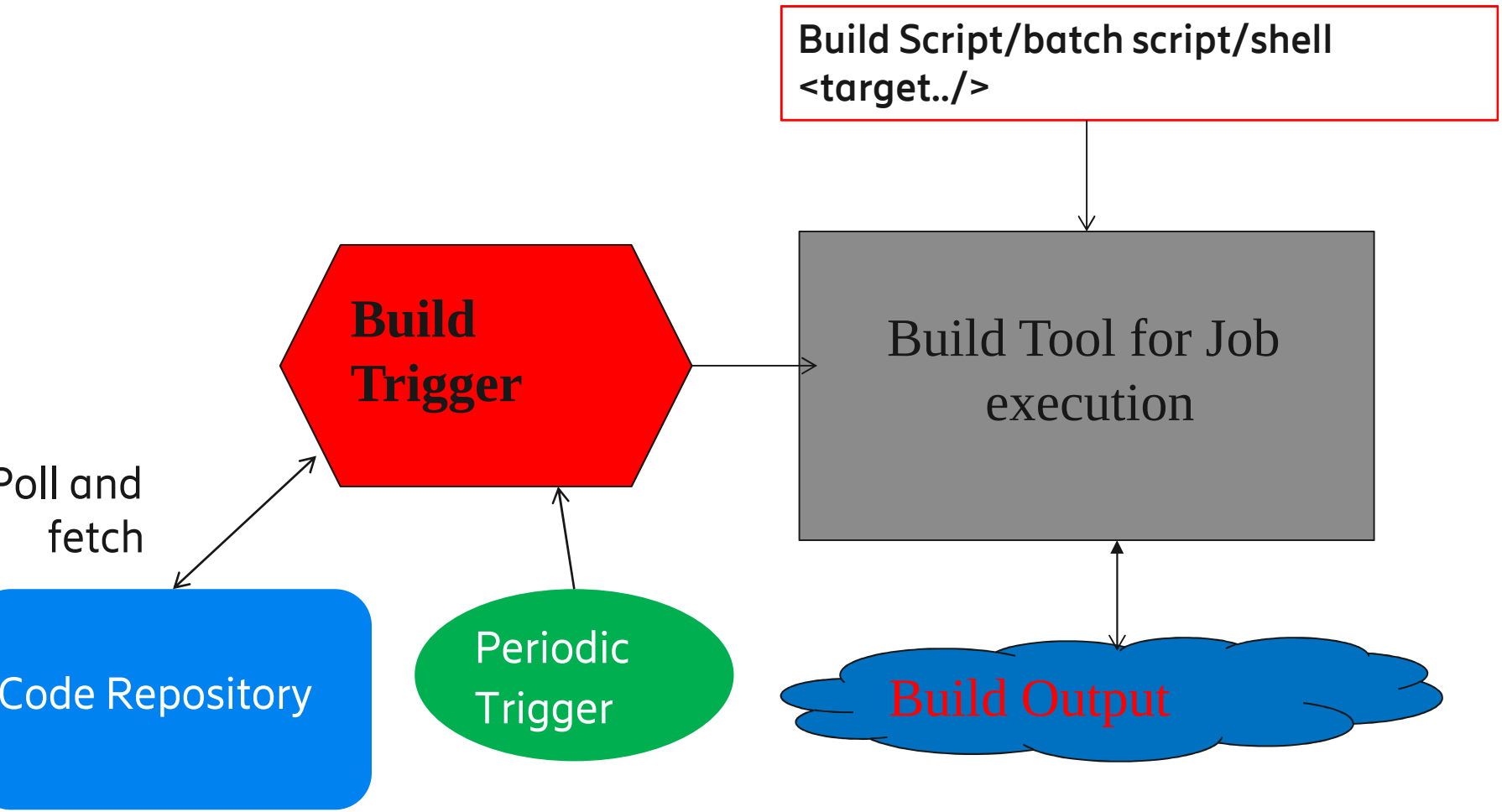
Configure Jenkins



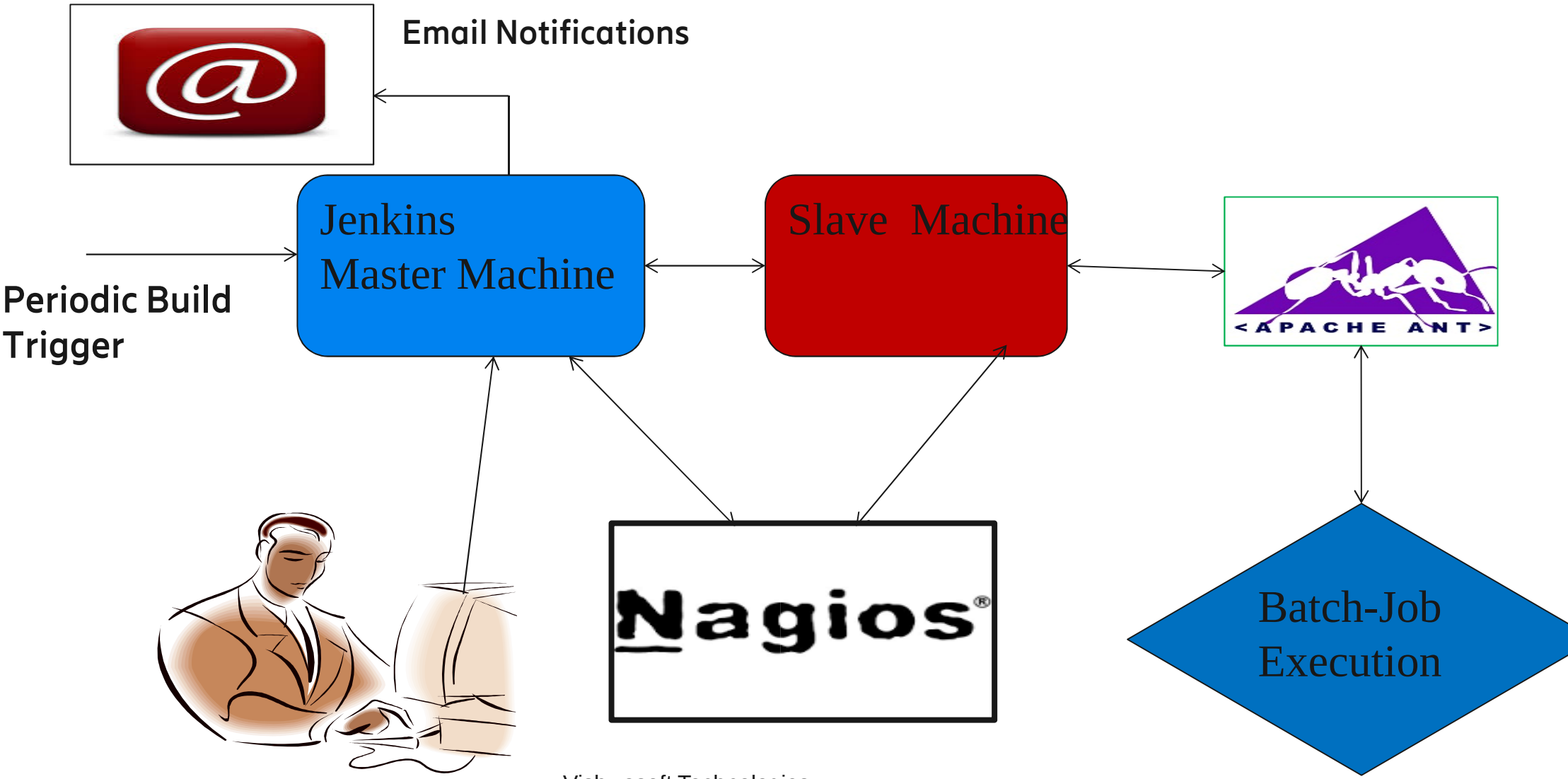
Jenkins Jobs

- The job is an execution 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 Build Automation Example



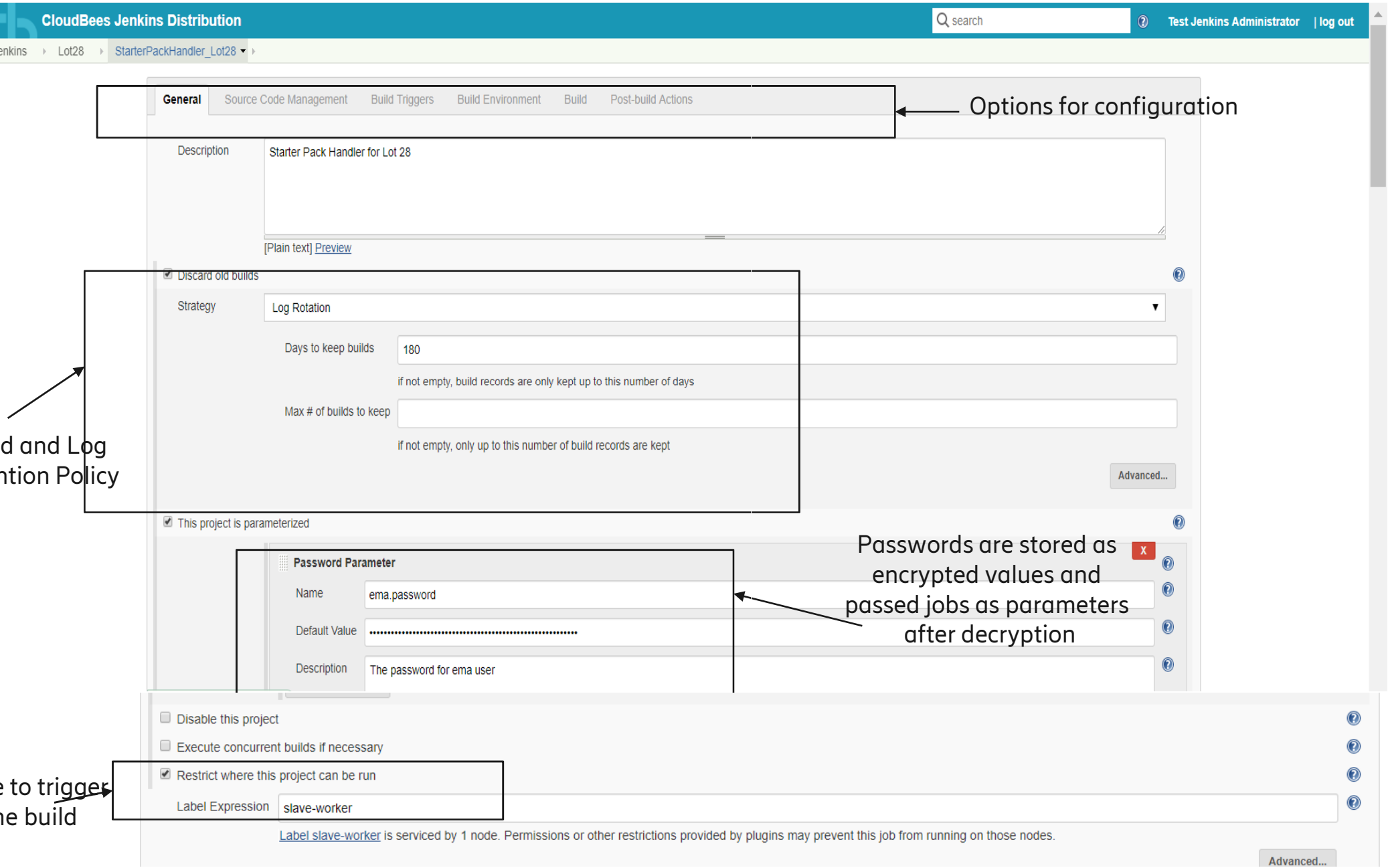
Jenkins Jobs

- 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.
  - Pipeline job
  - Copy job

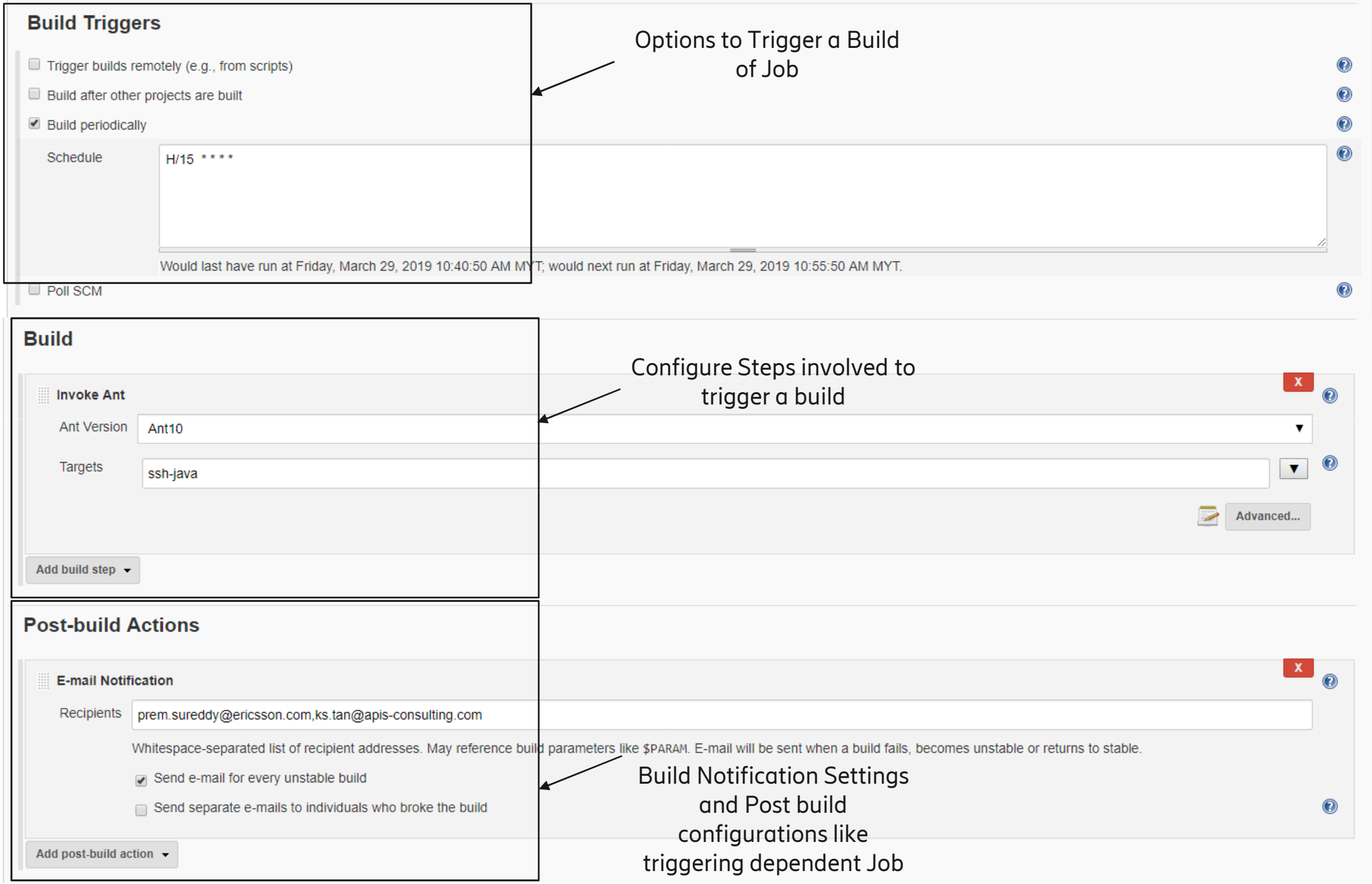
Job Configuration

- The SCM (code repository)code base location/URL
- The build process trigger—auto/repository/job in upstream
- Set of pre-build activities processes.
  - Unit testing
  - Bath script/Shell Script execution
  - Release package
  - Deploy package
- Set of post-build activities
  - Static code analysis
  - test report generation
  - Notification of build process via email/SMS
- Job execution and monitoring

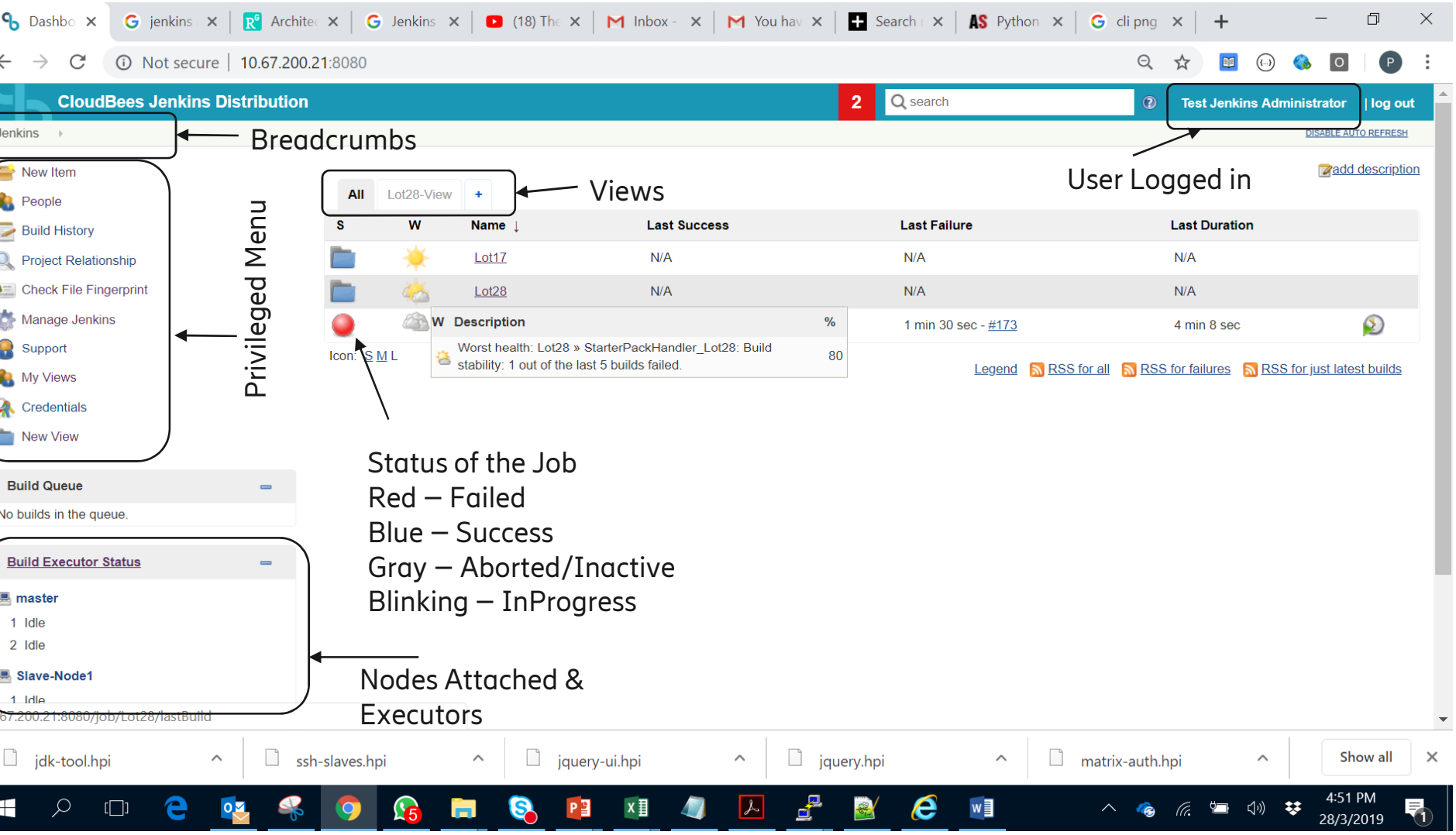
Job Configuration



Job Configuration Continued



Jenkins Jobs Dashboard



5 builds success

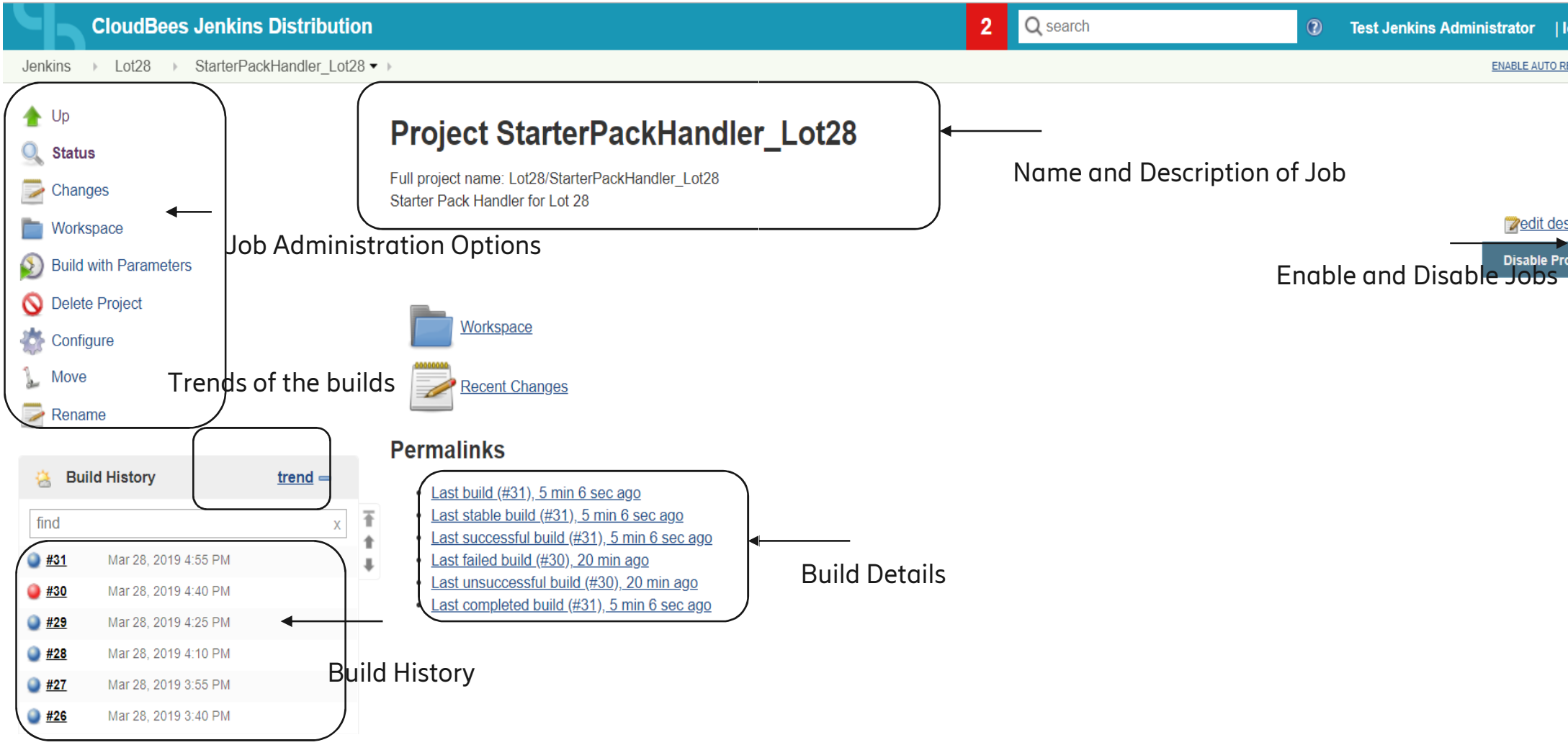
1 of las builds f

2 of las builds f

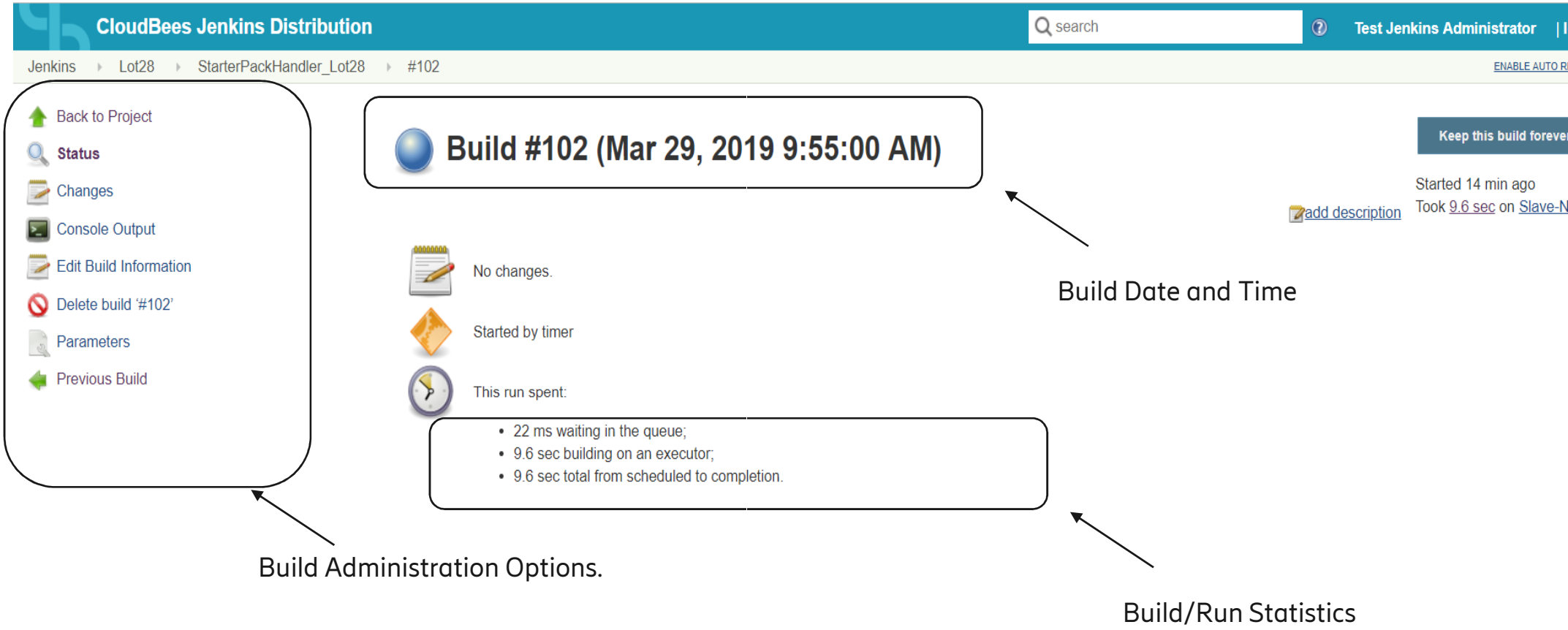
3 of las builds f

All Rece builds failed

Job Summary View



Job Build Details



Job build activities

- 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

- 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

- 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

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

Sonar with Jenkins

- 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

- 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

Continuous Delivery

Delivery/deployment

As part of build job pipeline activities

- Release management
- Packaging
- Deployment
- Start/stop remote server
- Execute acceptance testing on production server
- Test report generation
- Tracking the delivery stage
- Deploy to docker container
- Publish the image

Jenkins Build Pipeline

Jenkins pipeline plug-in

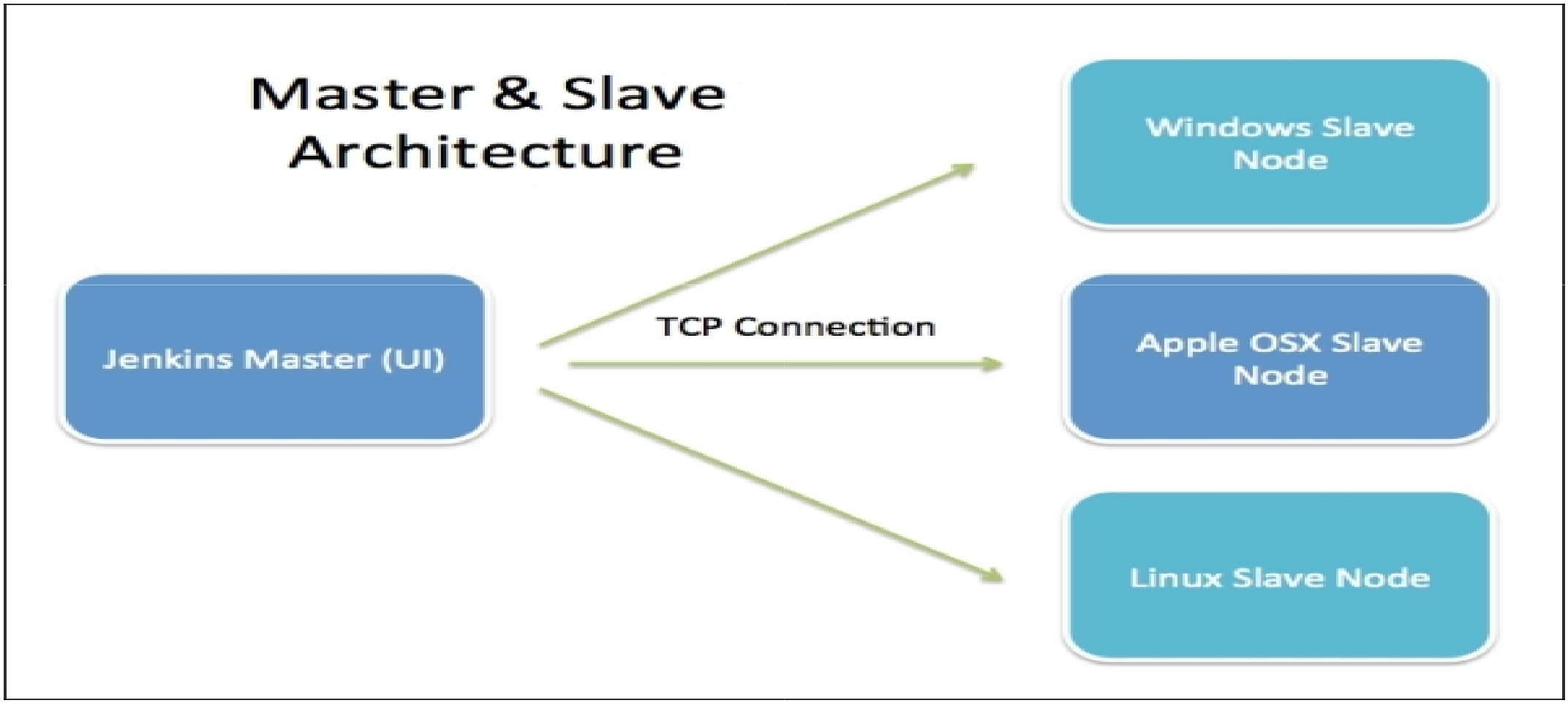
Enables create job with pipeline graphics

- Simple to configure the job with UI
- Reusable blocks
- Editable as text
- The configuration is saved in file
- The configuration file can be managed with repository
- The pipeline code is Infrastructure as code

Distributed build

— 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 salve node.
— The overhead on the master is reduced.

Jenkins in Distributed Mode



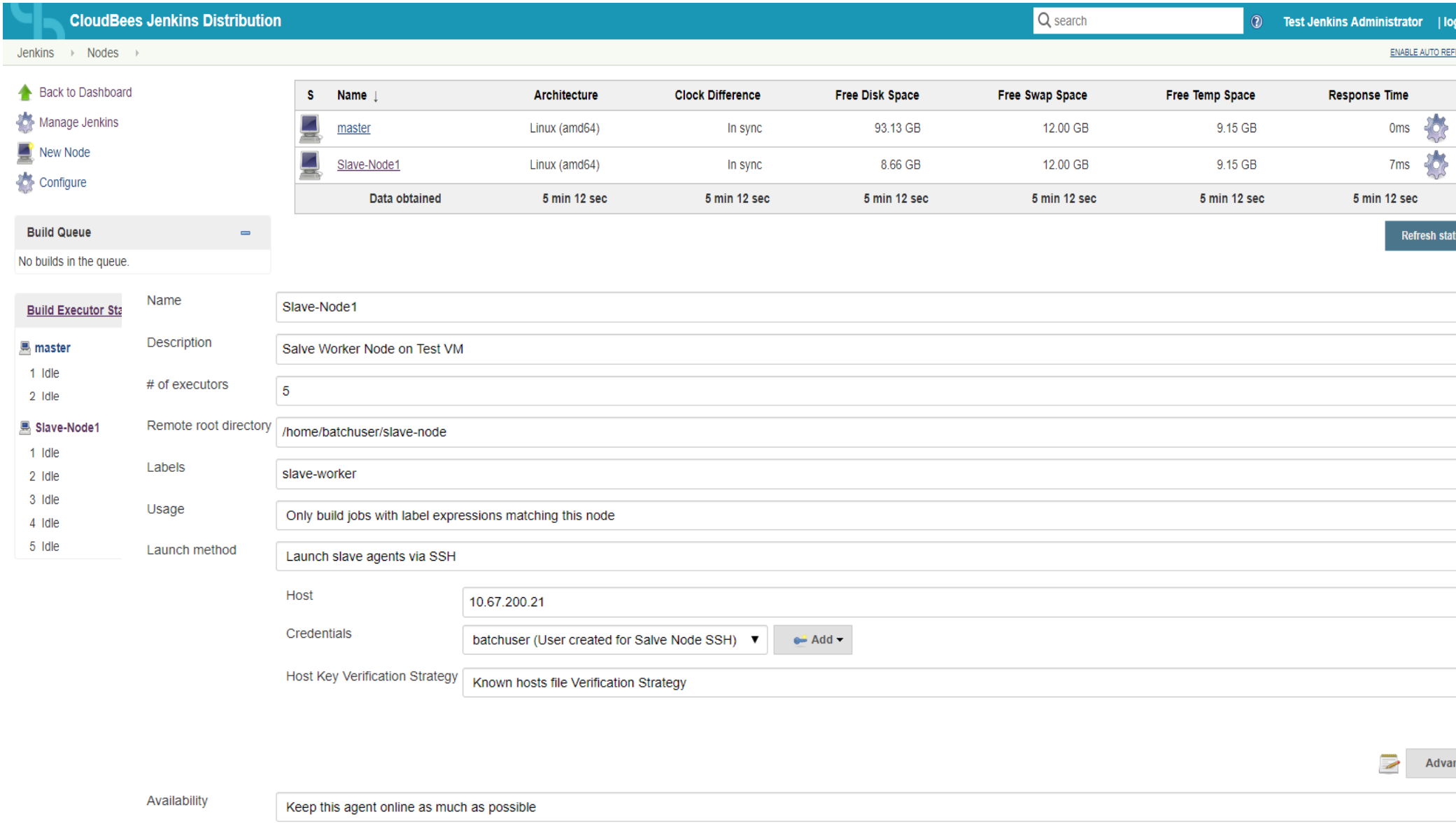
Master and Slave Nodes

— The Master node has entire configuration of 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.

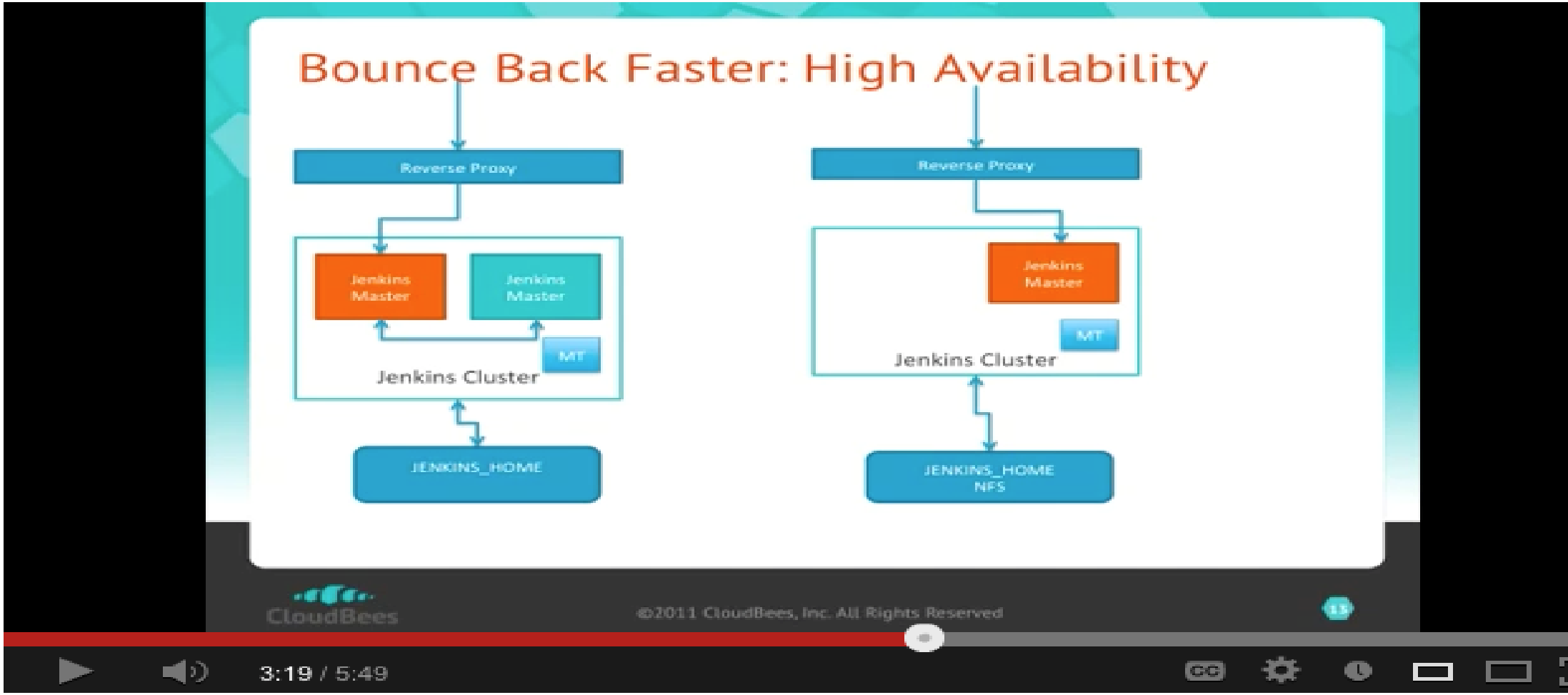
Distributed Jobs

— 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



Jenkins High Availability



Jenkins Cluster

- 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.

Jenkins scripting with groovy

Jenkins Management with groovy script

- Execute the groovy script as batch file
- Execute the groovy script as shell script
- Execute the groovy script with groovy plug-in.
- Execute the groovy script in script console.
- Customized plug-ins

Jenkins Script console

Execute the groovy script

Remotely Manage Jenkins

Manage Jenkins Remotely

Jenkins Remote Management

— Jenkins CLI from command line
— Jenkins web console
— Jenkins REST API

Jenkins with Docker

Jenkins-docker plug-in

— Build the docker image
— Create the docker container
— Deploy to docker
— Publish the latest built image to repository

Thank You!