Unit 5

Testing Tools and automation

As we know, **software testing** is a process of analyzing an application's functionality as per the customer prerequisite.

If we want to ensure that our software is bug-free or stable, we must perform the various types of software testing because testing is the only method that makes our application bug-free.

Various types of testing

The categorization of software testing is a part of diverse testing activities, such as **test strategy, test deliverables, a defined test objective, etc**. And software testing is the execution of the software to find defects.

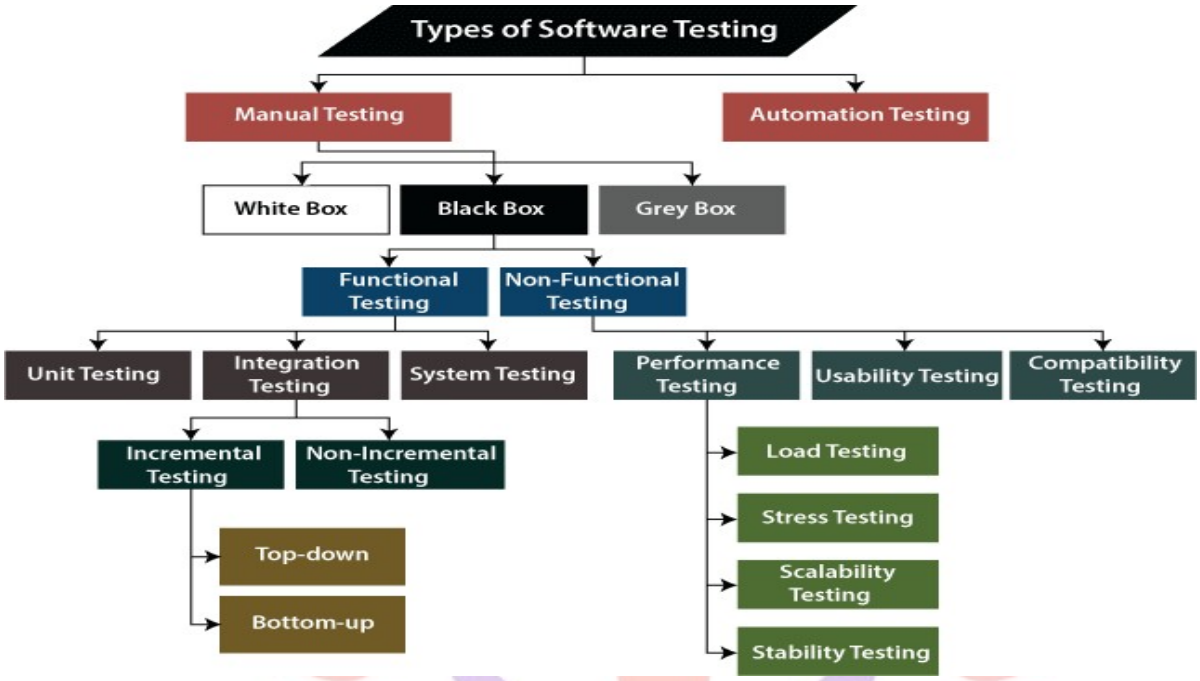
The purpose of having a testing type is to confirm the **AUT** (Application Under Test).

To start testing, we should have a **requirement, application-ready, necessary resources available**. To maintain accountability, we should assign a respective module to different test engineers.

The software testing mainly divided into two parts, which are as follows:

- **Manual Testing**
- **Automation Testing**

NRCM
your roots to success...



What is Manual Testing?

Testing any software or an application according to the client's needs without using any automation tool is known as **manual testing**.

In other words, we can say that it is a procedure of **verification and validation**. Manual testing is used to verify the behavior of an application or software in contradiction of requirements specification.

We do not require any precise knowledge of any testing tool to execute the manual test cases. We can easily prepare the test document while performing manual testing on any application.

To get in-detail information about manual testing, click on the following link: https://www.javatpoint.com/manual-testing.

Classification of Manual Testing

In software testing, manual testing can be further classified into **three different types of testing**, which are as follows:

- **White Box Testing**
- **Black Box Testing**
- **Grey Box Testing**

NRCM your roots to success...

For our better understanding let's see them one by one:

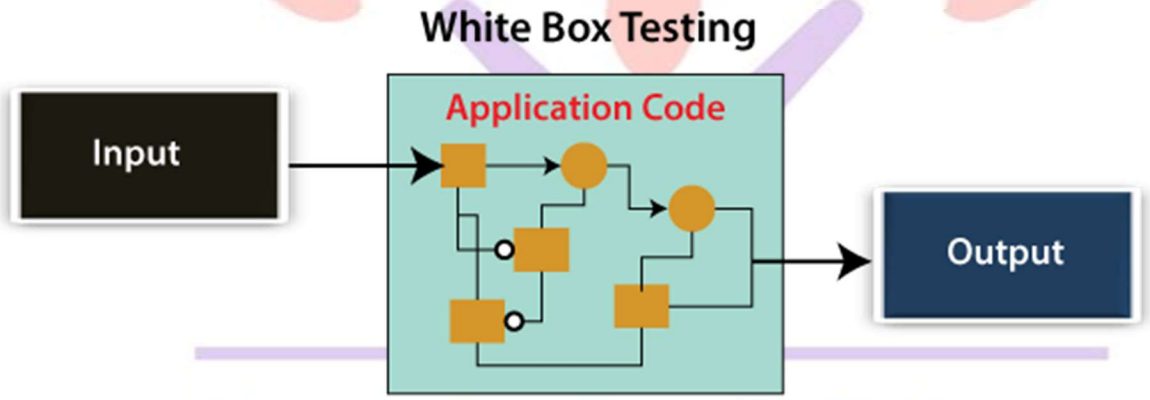
White Box Testing

In white-box testing, the developer will inspect every line of code before handing it over to the testing team or the concerned test engineers.

Subsequently, the code is noticeable for developers throughout testing; that's why this process is known as **WBT (White Box Testing)**.

In other words, we can say that the **developer** will execute the complete white-box testing for the particular software and send the specific application to the testing team.

The purpose of implementing the white box testing is to emphasize the flow of inputs and outputs over the software and enhance the security of an application.



White box testing is also known as **open box testing, glass box testing, structural testing, clear box testing, and transparent box testing**.

Black Box Testing

Another type of manual testing is **black-box testing**. In this testing, the test engineer will analyze the software against requirements, identify the defects or bug, and sends it back to the development team.

Then, the developers will fix those defects, do one round of White box testing, and send it to the testing team.

Here, fixing the bugs means the defect is resolved, and the particular feature is working according to the given requirement.

The main objective of implementing the black box testing is to specify the business needs or the customer's requirements.

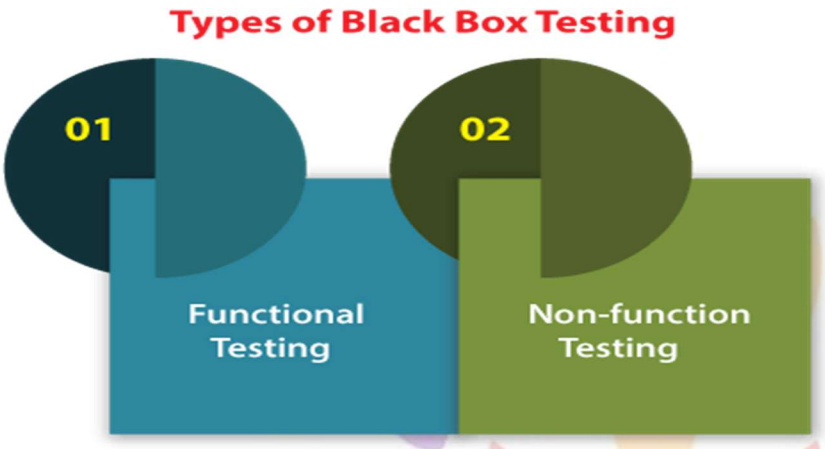
In other words, we can say that black box testing is a process of checking the functionality of an application as per the customer requirement. The source code is not visible in this testing; that's why it is known as **black-box testing**.

Types of Black Box Testing

Black box testing further categorizes into two parts, which are as discussed below:

- **Functional Testing**
- **Non-function Testing**

NRCM
your roots to success...



Functional Testing

The test engineer will check all the components systematically against requirement specifications is known as **functional testing**. Functional testing is also known as **Component testing**.

In functional testing, all the components are tested by giving the value, defining the output, and validating the actual output with the expected value.

Functional testing is a part of black-box testing as its emphases on application requirement rather than actual code. The test engineer has to test only the program instead of the system.

Types of Functional Testing

Just like another type of testing is divided into several parts, functional testing is also classified into various categories.

The diverse **types of Functional Testing** contain the following:

- **Unit Testing**
- **Integration Testing**
- **System Testing**

Now, Let's understand them one by one:

NRCM
your roots to success...

1. Unit Testing

Unit testing is the first level of functional testing in order to test any software. In this, the test engineer will test the module of an application independently or test all the module functionality is called **unit testing**.

The primary objective of executing the unit testing is to confirm the unit components with their performance. Here, a unit is defined as a single testable function of a software or an application. And it is verified throughout the specified application development phase.

2. Integration Testing

Once we are successfully implementing the unit testing, we will go integration testing. It is the second level of functional testing, where we test the data flow between dependent modules or interface between two features is called **integration testing**.

The purpose of executing the integration testing is to test the statement's accuracy between each module.

Types of Integration Testing

Integration testing is also further divided into the following parts:

- **Incremental Testing**
- **Non-Incremental Testing**

Incremental Integration Testing

Whenever there is a clear relationship between modules, we go for incremental integration testing. Suppose, we take two modules and analysis the data flow between them if they are working fine or not.

If these modules are working fine, then we can add one more module and test again. And we can continue with the same process to get better results.

In other words, we can say that incrementally adding up the modules and test the data flow between the modules is known as **Incremental integration testing**.

Types of Incremental Integration Testing

NRCM your roots to success...

Incremental integration testing can further classify into two parts, which are as follows:

1. **Top-down Incremental Integration Testing**
2. **Bottom-up Incremental Integration Testing**

Let's see a brief introduction of these types of integration testing:

1. Top-down Incremental Integration Testing

In this approach, we will add the modules step by step or incrementally and test the data flow between them. We have to ensure that the modules we are adding are the **child of the earlier ones**.

2. Bottom-up Incremental Integration Testing

In the bottom-up approach, we will add the modules incrementally and check the data flow between modules. And also, ensure that the module we are adding is the **parent of the earlier ones**.

Non-Incremental Integration Testing/ Big Bang Method

Whenever the data flow is complex and very difficult to classify a parent and a child, we will go for the non-incremental integration approach. The non-incremental method is also known as **the Big Bang method**.

3. System Testing

Whenever we are done with the unit and integration testing, we can proceed with the system testing.

In system testing, the test environment is parallel to the production environment. It is also known as **end-to-end** testing.

In this type of testing, we will undergo each attribute of the software and test if the end feature works according to the business requirement. And analysis the software product as a complete system.

your roots to success...

Non-function Testing

The next part of black-box testing is **non-functional testing**. It provides detailed information on software product performance and used technologies.

Non-functional testing will help us minimize the risk of production and related costs of the software.

Non-functional testing is a combination of **performance, load, stress, usability and, compatibility testing**.

Types of Non-functional Testing

Non-functional testing categorized into different parts of testing, which we are going to discuss further:

- **Performance Testing**
- **Usability Testing**
- **Compatibility Testing**

1. Performance Testing

In performance testing, the test engineer will test the working of an application by applying some load.

In this type of non-functional testing, the test engineer will only focus on several aspects, such as **Response time, Load, scalability, and Stability** of the software or an application.

Classification of Performance Testing

Performance testing includes the various types of testing, which are as follows:

- **Load Testing**
- **Stress Testing**
- **Scalability Testing**
- **Stability Testing**
- **Load Testing**

While executing the performance testing, we will apply some load on the particular application to check the application's performance, known as **load testing**. Here, the load could be less than or equal to the desired load.

your roots to success...

It will help us to detect the highest operating volume of the software and bottlenecks.

- **Stress Testing**

It is used to analyze the user-friendliness and robustness of the software beyond the common functional limits.

Primarily, stress testing is used for critical software, but it can also be used for all types of software applications.

- **Scalability Testing**

To analysis, the application's performance by enhancing or reducing the load in particular balances is known as **scalability testing**.

In scalability testing, we can also check the **system, processes, or database's ability** to meet an upward need. And in this, the **Test Cases** are designed and implemented efficiently.

- **Stability Testing**

Stability testing is a procedure where we evaluate the application's performance by applying the load for a precise time.

It mainly checks the constancy problems of the application and the efficiency of a developed product. In this type of testing, we can rapidly find the system's defect even in a stressful situation.

2. Usability Testing

Another type of **non-functional testing** is **usability testing**. In usability testing, we will analyze the user-friendliness of an application and detect the bugs in the software's end-user interface.

Here, the term **user-friendliness** defines the following aspects of an application:

- The application should be easy to understand, which means that all the features must be visible to end-users.
- The application's look and feel should be good that means the application should be pleasant looking and make a feel to the end-user to use it.

3. Compatibility Testing

In compatibility testing, we will check the functionality of an application in specific hardware and software environments. Once the application is functionally stable then only, we go for **compatibility testing**.

Here, **software** means we can test the application on the different operating systems and other browsers, and **hardware** means we can test the application on different sizes.

Grey Box Testing

Another part of **manual testing** is **Grey box testing**. It is a **collaboration of black box and white box testing**.

Since, the grey box testing includes access to internal coding for designing test cases. Grey box testing is performed by a person who knows coding as well as testing.

In other words, we can say that if a single-person team done both **white box and black-box testing**, it is considered **grey box testing**.

Automation Testing

The most significant part of Software testing is Automation testing. It uses specific tools to automate manual design test cases without any human interference.

Automation testing is the best way to enhance the efficiency, productivity, and coverage of Software testing.

It is used to re-run the test scenarios, which were executed manually, quickly, and repeatedly.

In other words, we can say that whenever we are testing an application by using some tools is known as **automation testing**.

your roots to success...

We will go for automation testing when various releases or several regression cycles goes on the application or software. We cannot write the test script or perform the automation testing without understanding the programming language.

Automation of testing Pros and cons

Some other types of Software Testing

In software testing, we also have some other types of testing that are not part of any above discussed testing, but those testing are required while testing any software or an application.

- **Smoke Testing**
- **Sanity Testing**
- **Regression Testing**
- **User Acceptance Testing**
- **Exploratory Testing**
- **Adhoc Testing**
- **Security Testing**
- **Globalization Testing**

Let's understand those types of testing one by one:

In **smoke testing**, we will test an application's basic and critical features before doing one round of deep and rigorous testing.

Or before checking all possible positive and negative values is known as **smoke testing**. Analyzing the workflow of the application's core and main functions is the main objective of performing the smoke testing.

Sanity Testing

It is used to ensure that all the bugs have been fixed and no added issues come into existence due to these changes. Sanity testing is unscripted, which means we cannot documented it. It checks the correctness of the newly added features and components.

Regression Testing

Regression testing is the most commonly used type of software testing. Here, the term **regression** implies that we have to re-test those parts of an unaffected application.

Regression testing is the most suitable testing for automation tools. As per the project type and accessibility of resources, regression testing can be similar to **Retesting**.

NRCM your roots to success

Whenever a bug is fixed by the developers and then testing the other features of the applications that might be simulated because of the bug fixing is known as **regression testing**.

In other words, we can say that whenever there is a new release for some project, then we can perform Regression Testing, and due to a new feature may affect the old features in the earlier releases.

User Acceptance Testing

The User acceptance testing (UAT) is done by the individual team known as domain expert/customer or the client. And knowing the application before accepting the final product is called as **user acceptance testing**.

In user acceptance testing, we analyze the business scenarios, and real-time scenarios on the distinct environment called the **UAT environment**. In this testing, we will test the application before UAI for customer approval.

Exploratory Testing

Whenever the requirement is missing, early iteration is required, and the testing team has experienced testers when we have a critical application. New test engineer entered into the team then we go for the **exploratory testing**.

To execute the exploratory testing, we will first go through the application in all possible ways, make a test document, understand the flow of the application, and then test the application.

Adhoc Testing

Testing the application randomly as soon as the build is in the checked sequence is known as **Adhoc testing**.

It is also called **Monkey testing and Gorilla testing**. In Adhoc testing, we will check the application in contradiction of the client's requirements; that's why it is also known as **negative testing**.

When the end-user using the application casually, and he/she may detect a bug. Still, the specialized test engineer uses the software thoroughly, so he/she may not identify a similar detection.

Security Testing

NRCM your roots to success...

It is an essential part of software testing, used to determine the weakness, risks, or threats in the software application.

The execution of security testing will help us to avoid the nasty attack from outsiders and ensure our software applications' security.

In other words, we can say that security testing is mainly used to define that the data will be safe and endure the software's working process.

Globalization Testing

Another type of software testing is **Globalization testing.** Globalization testing is used to check the developed software for multiple languages or not. Here, the words **globalization** means enlightening the application or software for various languages.

Globalization testing is used to make sure that the application will support multiple languages and multiple features.

In present scenarios, we can see the enhancement in several technologies as the applications are prepared to be used globally.

Conclusion

In the tutorial, we have discussed various types of software testing. But there is still a list of more than 100+ categories of testing. However, each kind of testing is not used in all types of projects.

We have discussed the most commonly used types of Software Testing like **black-box testing, white box testing, functional testing, non-functional testing, regression testing, Adhoc testing, etc**.

Also, there are alternate classifications or processes used in diverse organizations, but the general concept is similar all over the place.

These testing types, processes, and execution approaches keep changing when the project, requirements, and scope change.

Automation of testing Pros and cons

Pros of Automated Testing:
Automated Testing has the following advantages:

your roots to success...

1. Automated testing improves the coverage of testing as automated execution of test cases is faster than manual execution.
2. Automated testing reduces the dependability of testing on the availability of the test engineers.
3. Automated testing provides round the clock coverage as automated tests can be run all time in 24*7 environment.
4. Automated testing takes far less resources in execution as compared to manual testing.
5. It helps to train the test engineers to increase their knowledge by producing a repository of different tests.
6. It helps in testing which is not possible without automation such as reliability testing, stress testing, load and performance testing.
7. It includes all other activities like selecting the right product build, generating the right test data and analyzing the results.
8. It acts as test data generator and produces maximum test data to cover a large number of input and expected output for result comparison.
9. Automated testing has less chances of error hence more reliable.
10. As with automated testing test engineers have free time and can focus on other creative tasks.

Cons of Automated Testing : Automated Testing has the following disadvantages:

1. Automated testing is very much expensive than the manual testing.
2. It also becomes inconvenient and burdensome as to decide who would automate and who would train.
3. It has limited to some organisations as many organisations not prefer test automation.
4. Automated testing would also require additionally trained and skilled people.
5. Automated testing only removes the mechanical execution of testing process, but creation of test cases still required testing professionals.

Selenium

Introduction

Selenium is one of the most widely used open source Web UI (User Interface) automation testing suite.It was originally developed by Jason Huggins in 2004 as an internal tool at Thought Works. Selenium supports automation across different browsers, platforms and programming languages.

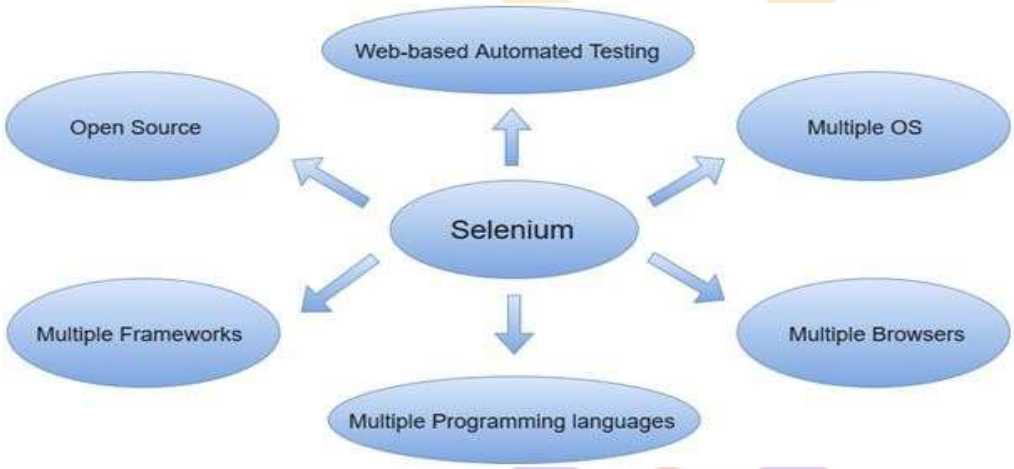
Selenium can be easily deployed on platforms such as Windows, Linux, Solaris and Macintosh. Moreover, it supports OS (Operating System) for mobile applications like iOS, windows mobile and android.

Selenium supports a variety of programming languages through the use of drivers specific to each Language.

NRCM your roots to success...

Languages supported by Selenium include C#, Java, Perl, PHP, Python and Ruby.

Currently, Selenium Web driver is most popular with Java and C#. Selenium test scripts can be coded in any of the supported programming languages and can be run directly in most modern web browsers. Browsers supported by Selenium include Internet Explorer, Mozilla Firefox, Google Chrome and Safari.



Selenium can be used to automate functional tests and can be integrated with automation test tools such as **Maven**, **Jenkins**, **& Docker** to achieve continuous testing. It can also be integrated with tools such as **TestNG**, & **JUnit** for managing test cases and generating reports.

Selenium Features

- Selenium is an open source and portable Web testing Framework.
- Selenium IDE provides a playback and record feature for authoring tests without the need to learn a test scripting language.
- It can be considered as the leading cloud-based testing platform which helps testers to record their actions and export them as a reusable script with a simple-to-understand and easy-to-use interface.
- Selenium supports various operating systems, browsers and programming languages. Following is the list:
    - Programming Languages: C#, Java, Python, PHP, Ruby, Perl, and JavaScript
    - Operating Systems: Android, iOS, Windows, Linux, Mac, Solaris.
    - Browsers: Google Chrome, Mozilla Firefox, Internet Explorer, Edge, Opera, Safari, etc.
- It also supports parallel test execution which reduces time and increases the efficiency of tests.

NRCM your roots to success...

- Selenium can be integrated with frameworks like Ant and Maven for source code compilation.
- Selenium can also be integrated with testing frameworks like TestNG for application testing and generating rseports.
- Selenium requires fewer resources as compared to other automation test tools.
- WebDriver API has been indulged in selenium whichis one of the most important modifications done to selenium.
- Selenium web driver does not require server installation, test scripts interact directly with the browser.
- Selenium commands are categorized in terms of different classes which make it easier to understand and implement.

JavaScript testing

JavaScript testing is a crucial part of the software development process that helps ensure the quality and reliability of code. The following are the key components of JavaScript testing:

Test frameworks: A test framework provides a structure for writing and organizing tests. Some popular JavaScript test frameworks include Jest, Mocha, and Jasmine.

Assertion libraries: An assertion library provides a set of functions that allow developers to write assertions about the expected behavior of the code. For example, an assertion might check that a certain function returns the expected result.

Test suites: A test suite is a collection of related tests that are grouped together. The purpose of a test suite is to test a specific aspect of the code in isolation.

Test cases: A test case is a single test that verifies a specific aspect of the code. For example, a test case might check that a function behaves correctly when given a certain input.

Test runners: A test runner is a tool that runs the tests and provides feedback on the results. Test runners typically provide a report on which tests passed and which tests failed.

Continuous Integration (CI): CI is a software development practice where developers integrate code into a shared repository frequently. By using CI, developers can catch issues early and avoid integration problems.

The goal of JavaScript testing is to catch bugs and defects early in the development cycle, before they become bigger problems and impact the quality of the software. Testing also helps to ensure that the code behaves as expected, even when changes are made in the future.

There are different types of tests that can be performed in JavaScript, including unit tests, integration tests, and end-to-end tests. The choice of which tests to write depends on the specific requirements and goals of the project.

Testing backend integration points

The term backend generally refers to **server-side deployment.** Here the process is entirely happening in the backend which is not shown to the user only the expected results will be shown to the user. In every web application, there will be a backend language to accomplish the task.
For Example, while uploading the details of the students in the database, the database will store all the details. When there is a need to display the details of the students, it will simply fetch all the details and display them. Here, it will show only the result, not the process and how it fetches the details.

What is Backend Testing?

Backend Testing is a testing method that checks the database or server-side of the web application. The main purpose of backend testing is to check the application layer and the database layer. It will find an error or bug in the database or server-side.

For implementing backend testing, the backend test engineer should also have some knowledge about that particular server-side or database language. It is also known as **Database Testing.**
Importance of Backend Testing: Backend testing is a must because anything wrong or error happens at the server-side, it will not further proceed with that task or the output will get differed or sometimes it will also cause problems such as data loss, deadlock, etc.,

Types of Backend Testing

The following are the different types of backend testing:

1. **Structural Testing**
2. **Functional Testing**
3. **Non-Functional Testing**

Let's discuss each of these types of backend testing.

NRCM your roots to success...

1. Structural Testing

Structural testing is the process of validating all the elements that are present inside the data repository and are primarily used for data storage. It involves checking the objects of front-end developments with the database mapping objects.

Types of Structural Testing: The following are the different types of structural testing:

a) Schema Testing: In this Schema Testing, the tester will check for the correctly mapped objects. This is also known as **mapping testing.** It ensures whether the objects of the front-end and the objects of the back-end are correctly matched or mapped. It will mainly focus on schema objects such as a table, view, indexes, clusters, etc., In this testing, the tester will find the issues of mapped objects like table, view, etc.,

b) Table and Column Testing: In this, it ensures that the table and column properties are correctly mapped.

- It ensures whether the table and the column names are correctly mapped on both the front-end side and server-side.
- It validates the datatype of the column is correctly mentioned.
- It ensures the correct naming of the column values of the database.
- It detects the unused tables and columns.
- It validates whether the users are able to give the correct input as per the requirement.

For example, if we mention the wrong datatype for the column on the server-side which is different from the front-end then it will raise an error.

c) Key and Indexes Testing: In this, it validates the key and indexes of the columns.

- It ensures whether the mentioned key constraints are correctly provided. For example, Primary Key for the column is correctly mentioned as per the given requirement.
- It ensures the correct references of Foreign Key with the parent table.
- It checks the length and size of the indexes.
- It ensures the creation of clustered and non-clustered indexes for the table as per the requirement.
- It validates the naming conventions of the Keys.

d) Trigger Testing: It ensures that the executed triggers are fulfilling the required conditions of the DML transactions.

- It validates whether the triggers make the data updates correctly when we have executed them.
- It checks the coding conventions are followed correctly during the coding phase of the triggers.
- It ensures that the trigger functionalities of update, delete, and insert.

NRCM your roots to success...

e) Stored Procedures Testing: In this, the tester checks for the correctness of the stored procedure results.

- It checks whether the stored procedure contains the valid conditions for looping and conditional statements as per the requirement.
- It validates the exception and error handling in the stored procedure.
- It detects the unused stored procedure.
- It validates the cursor operations.
- It validates whether the TRIM operations are correctly applied or not.
- It ensures that the required triggers are implicitly invoked by executing the stored procedures.

f) Database Server Validation Testing: It validates the database configuration details as per the requirements.

- It validates that the transactions of the data are made as per the requirements.
- It validates the user's authentication and authorization.

For Example, If wrong user authentication is given, it will raise an error.

2. Functional Testing

Functional Testing is the process of validating that the transactions and operations made by the end-users meet the requirements.

Types of Functional Testing: The following are the different types of functional testing:

a) Black Box Testing:

- Black Box Testing is the process of checking the functionalities of the integration of the database.
- This testing is carried out at the early stage of development and hence It is very helpful to reduce errors.
- It consists of various techniques such as boundary analysis, equivalent partitioning, and cause-effect graphing.
- These techniques are helpful in checking the functionality of the database.
- The best example is the User login page. If the entered username and password are correct, It will allow the user and redirect to the next page.

b) White Box Testing:

- White Box Testing is the process of validating the internal structure of the database.
- Here, the specified details are hidden from the user.
- The database triggers, functions, views, queries, and cursors will be checked in this testing.

NRCM Your roots to success...

- It validates the database schema, database table, etc.,
- Here the coding errors in the triggers can be easily found.
- Errors in the queries can also be handled in this white box testing and hence internal errors are easily eliminated.

3. Non-Functional Testing

Non-functional testing is the process of performing load testing, stress testing, and checking minimum system requirements are required to meet the requirements. It will also detect risks, and errors and optimize the performance of the database.

a) Load Testing:

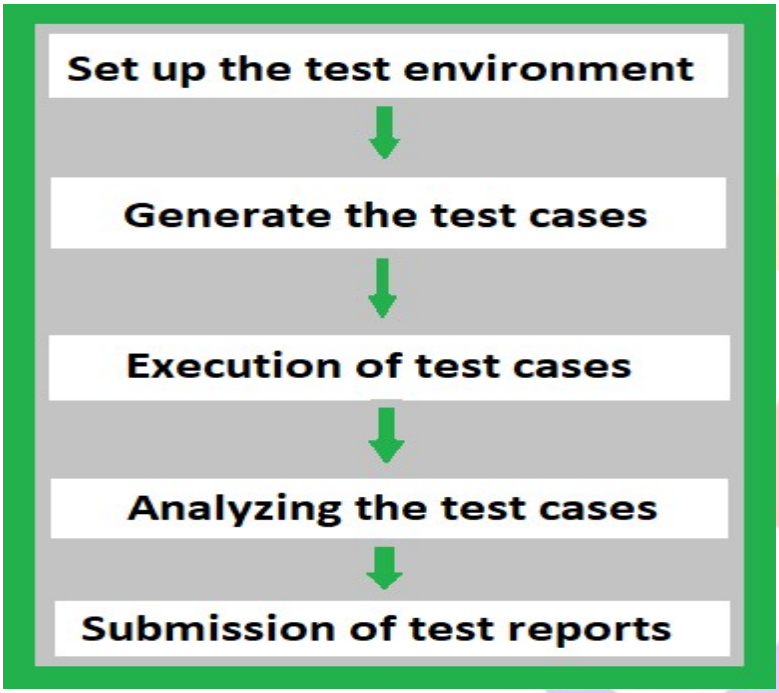
- Load testing involves testing the performance and scalability of the database.
- It determines how the software behaves when it is been used by many users simultaneously.
- It focuses on good load management.
- For example, if the web application is accessed by multiple users at the same time and it does not create any traffic problems then the load testing is successfully completed.

b) Stress Testing:

- Stress Testing is also known as endurance testing. Stress testing is a testing process that is performed to identify the breakpoint of the system.
- In this testing, an application is loaded till the stage the system fails.
- This point is known as a breakpoint of the database system.
- It evaluates and analyzes the software after the breakage of system failure. In case of error detection, It will display the error messages.
- For example, if users enter the wrong login information then it will throw an error message.

NRCM
your roots to success...

Backend Testing Process



1. Set up the Test Environment: When the coding process is done for the application, set up the test environment by choosing a proper testing tool for back-end testing. It includes choosing the right team to test the entire back-end environment with a proper schedule. Record all the testing processes in the documents or update them in software to keep track of all the processes.

2. Generate the Test Cases: Once the tool and the team are ready for the testing process, generate the test cases as per the business requirements. The automation tool itself will analyze the code and generate all possible test cases for developed code. If the process is manual then the tester will have to write the possible test cases in the testing tool to ensure the correctness of the code.

3. Execution of Test Cases: Once the test cases are generated, the tester or Quality Analyst needs to execute those test cases in the developed code. If the tool is automated, it will generate and execute the test cases by itself. Otherwise, the tester needs to write and execute those test cases. It will highlight whether the execution of test cases is executed successfully or not.

4. Analyzing the Test Cases: After the execution of test cases, it highlights the result of all the test cases whether it has been executed successfully or not. If an error occurs in the test cases, it will highlight where the particular error is formed or raised, and in some cases, the automation tool will give hints regarding the issues to solve the error. The

tester or Quality Analyst should analyze the code again and fix the issues if an error occurred.

5. Submission of Test Reports: This is the last stage in the testing process. Here, all the details such as who is responsible for testing, the tool used in the testing process, number of test cases generated, number of test cases executed successfully or not, time is taken to execute each test case, number of times test cases failed, number of times errors occurred. These details are either documented or updated in the software. The report will be submitted to the respective team.

Backend Testing Validation

The following are some of the factors for backend testing validation:

- **Performance Check:** It validates the performance of each individual test and the system behavior.
- **Sequence Testing:** Backend testing validates that the tests are distributed according to the priority.
- **Database Server Validations:** In this, ensures that the data fed through for the tests is correct or not.
- **Functions Testing:** In this, the test validates the consistency in transactions of the database.
- **Key and Indexes:** In this, the test ensures that the accurate constraint and the rules of constraints and indexes are followed properly.
- **Data Integrity Testing:** It is a technique in which data is verified in the database whether it is accurate and functions as per requirements.
- **Database Tables:** It ensures that the created table and the queries for the output are providing the expected result.
- **Database Triggers:** Backend Testing validates the correctness of the functionality of triggers.
- **Stored Procedures:** Backend testing validates the functions, return statements, calling the other events, etc., are correctly mentioned as per the requirements,
- **Schema:** Backend testing validates that the data is organized in a correct way as per the business requirement and confirms the outcome.

NRCM your roots to success...

Tools For Backend Testing

The following are some of the tools for backend testing:

1. LoadRunner:

- It is a stress testing tool.
- It is an automated performance and testing automation tool for analyzing system behavior and the performance of the system while generating the actual load.

2. Empirix-TEST Suite:

- It is acquired by Oracle from Empirix. It is a load testing tool.
- It validates the scalability along with the functionality of the application under heavy test.
- Acquisition with the Empirix -Test suite may be proven effective to deliver the application with improved quality.

3. Stored Procedure Testing Tools – LINQ:

- It is a powerful tool that allows the user to show the projects.
- It tracks all the ORM calls and database queries from the ORM.
- It enables to see the performance of the data access code and easily determine performance.

4. Unit Testing Tools – SQL Unit, DBFit, NDbUnit:

- **SQL UNIT:** SQLUnit is a Unit Testing Framework for Regression and Unit Testing of database stored procedures.
- **DBFit:** It is a part of FitNesse and manages stored procedures and custom procedures. Accomplishes database testing either through Java or .NET and runs from the command line.
- **NDbUnit:** It performs the database unit test for the system either before or after execution or compiled the other parts of the system.

5. Data Factory Tools:

- These tools work as data managers and data generators for backend database testing.
- It is used to validate the queries with a huge set of data.
- It allows performing both stress and load testing.

6. SQLMap:

- It is an open-source tool.
- It is used for performing Penetration Testing to automate the process of detection.
- Powerful detection of errors will lead to efficient testing and result in the expected behavior of the requirements.

7. phpMyadmin:

- This is the software tool and it is written in PHP.
- It is developed to handle the databases and we can execute test queries to ensure the correctness of the result as a whole and even for a separate table.

NRCM your roots to success...

8. Automatic Efficient Test Generator (AETG):

- It mechanically generates the possible tests from user-defined requirements.
- It is based on algorithms that use ideas from statistical experimental design theory to reduce the number of tests needed for a specific level of test coverage of the input test space.

9. Hammer DB:

- It is an open-source tool for load testing.
- It validates the activity replay functionality for the oracle database.
- It is based on industry standards like TPC-C and TPC-H Benchmarks.

10. SQL Test:

- SQL Test uses an open-source tSQLt framework, views, stored procedures, and functions.
- This tool stores database object in a separate schema and if changes occur there is no need for clearing the process.
- It allows running the unit test cases for the SQL server database.

Advantages of Backend Testing

The following are some of the benefits of backend testing:

- Errors are easily detectable at the earlier stage.
- It avoids deadlock creation on the server-side.
- Web load management is easily achieved.
- The functionality of the database is maintained properly.
- It reduces data loss.
- Enhances the functioning of the system.
- It ensures the security and protection of the system.
- While doing the backend testing, the errors in the UI parts can also be detected and replaced.
- Coverage of all possible test cases.

Disadvantages of Backend Testing

The following are some of the disadvantages of backend testing:

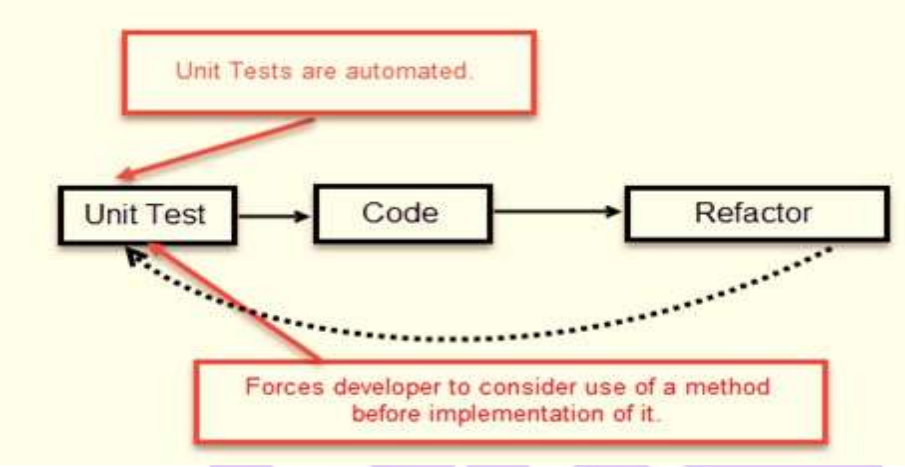
- Good domain knowledge is required.
- Providing test cases for testing requires special attention.

NRCM your roots to success...

- Investment in Organizational costs is higher.
- It takes more time to test.
- If more testing becomes fails then It will lead to a crash on the server-side in some cases.
- Errors or Unexpected results from one test case scenario will affect the other system results also.

Test-driven development

Test Driven Development (TDD) is software development approach in which test cases are developed to specify and validate what the code will do. In simple terms, test cases for each functionality are created and tested first and if the test fails then the new code is written in order to pass the test and making code simple and bug-free.
Test-Driven Development starts with designing and developing tests for every small functionality of an application. TDD framework instructs developers to write new code only if an automated test has failed. This avoids duplication of code. The TDD full form is Test-driven development.



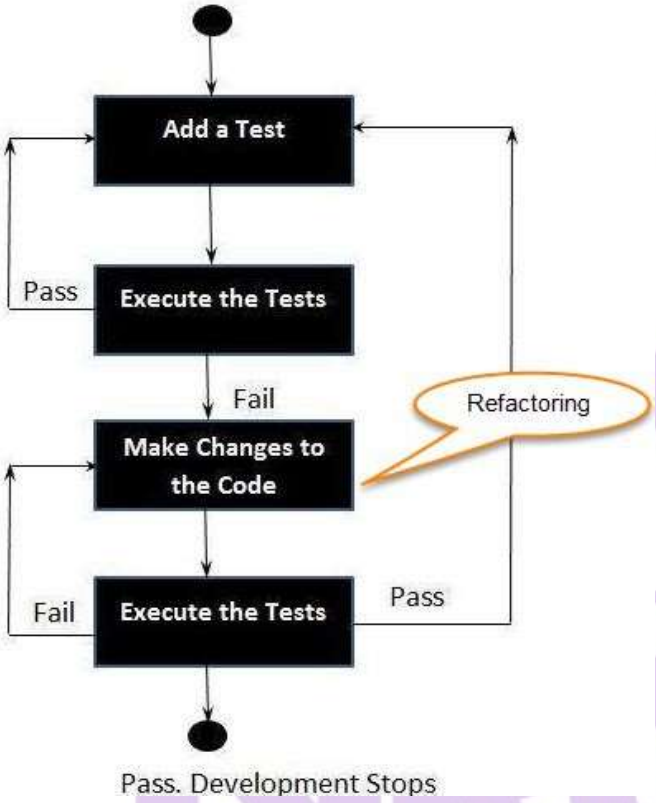
The simple concept of TDD is to write and correct the failed tests before writing new code (before development). This helps to avoid duplication of code as we write a small amount of code at a time in order to pass tests. (Tests are nothing but requirement conditions that we need to test to fulfill them).

Test-Driven development is a process of developing and running automated test before actual development of the application. Hence, TDD sometimes also called as **Test First Development.**

How to perform TDD Test

Following steps define how to perform TDD test,

1. Add a test.
2. Run all tests and see if any new test fails.
3. Write some code.
4. Run tests and Refactor code.
5. Repeat



TDD Vs. Traditional Testing

Below is the main difference between Test driven development and traditional testing:

TDD approach is primarily a specification technique. It ensures that your source code is thoroughly tested at confirmatory level.

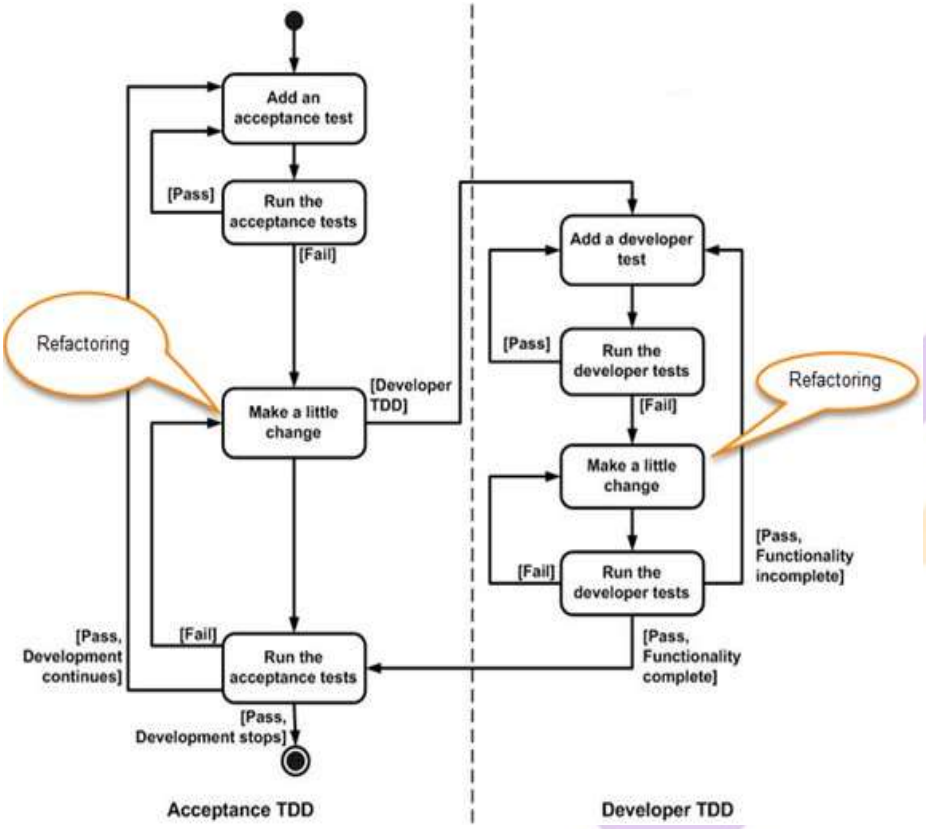
- With traditional testing, a successful test finds one or more defects. It is same as TDD. When a test fails, you have made progress because you know that you need to resolve the problem.
- TDD ensures that your system actually meets requirements defined for it. It helps to build your confidence about your system.
- In TDD more focus is on production code that verifies whether testing will work properly. In traditional testing, more focus is on test case design. Whether the test will show the proper/improper execution of the application in order to fulfill requirements.
- In TDD, you achieve 100% coverage test. Every single line of code is tested, unlike traditional testing.
- The combination of both traditional testing and TDD leads to the importance of testing the system rather than perfection of the system.
- In Agile Modeling (AM), you should "test with a purpose". You should know why you are testing something and what level its need to be tested.

What is acceptance TDD and Developer TDD

There are two levels of TDD

1. **Acceptance TDD (ATDD):** With ATDD you write a single acceptance test. This test fulfills the requirement of the specification or satisfies the behavior of the system. After that write just enough production/functionality code to fulfill that acceptance test. Acceptance test focuses on the overall behavior of the system. ATDD also was known as **Behavioral Driven Development (BDD).**
2. **Developer TDD:** With Developer TDD you write single developer test i.e. unit test and then just enough production code to fulfill that test. The unit test focuses on every small functionality of the system. Developer TDD is simply called as **TDD.**The main goal of ATDD and TDD is to specify detailed, executable requirements for your solution on a just in time (JIT) basis. JIT means taking only those requirements in consideration that are needed in the system. So increase efficiency.

NRCM
your roots to success...



REPL-driven development

REPL-driven development (Read-Eval-Print Loop) is an interactive programming approach that allows developers to execute code snippets and see their results immediately. This enables developers to test their code quickly and iteratively, and helps them to understand the behavior of their code as they work.

In a REPL environment, developers can type in code snippets, and the environment will immediately evaluate the code and return the results. This allows developers to test small bits of code and quickly see the results, without having to create a full-fledged application.

REPL-driven development is commonly used in dynamic programming languages such as Python, JavaScript, and Ruby. Some popular REPL environments include the Python REPL, Node.js REPL, and IRB (Interactive Ruby).

Benefits of REPL-driven development include:

Increased efficiency: The immediate feedback provided by a REPL environment allows developers to test and modify their code quickly, without having to run a full-fledged application.

Improved understanding: By being able to see the results of code snippets immediately, developers can better understand how the code works and identify any issues early on.

Increased collaboration: REPL-driven development makes it easy for developers to share code snippets and collaborate on projects, as they can demonstrate the behavior of the code quickly and easily.

Overall, REPL-driven development is a useful tool for developers looking to improve their workflow and increase their understanding of their code. By providing an interactive environment for testing and exploring code, REPL-driven development can help developers to be more productive and efficient.

Deployment of the system:

In DevOps, deployment systems are responsible for automating the release of software updates and applications from development to production. Some popular deployment systems include:

Jenkins: an open-source automation server that provides plugins to support building, deploying, and automating any project.

Ansible: an open-source platform that provides a simple way to automate software provisioning, configuration management, and application deployment.

Docker: a platform that enables developers to create, deploy, and run applications in containers.

Kubernetes: an open-source system for automating deployment, scaling, and management of containerized applications.

AWS Code Deploy: a fully managed deployment service that automates software deployments to a variety of compute services such as Amazon EC2, AWS Fargate, and on-premises servers.

Azure DevOps: a Microsoft product that provides an end-to-end DevOps solution for developing, delivering, and deploying applications on multiple platforms.

Virtualization stacks

In DevOps, virtualization refers to the creation of virtual machines, containers, or environments that allow multiple operating systems to run on a single physical machine. The following are some of the commonly used virtualization stacks in DevOps:

Docker: An open-source platform for automating the deployment, scaling, and management of containerized applications.

Kubernetes: An open-source platform for automating the deployment, scaling, and management of containerized applications, commonly used in conjunction with Docker.

VirtualBox: An open-source virtualization software that allows multiple operating systems to run on a single physical machine.

VMware: A commercial virtualization software that provides a comprehensive suite of tools for virtualization, cloud computing, and network and security management.

Hyper-V: Microsoft's hypervisor technology that enables virtualization on Windows-based systems.

These virtualization stacks play a crucial role in DevOps by allowing developers to build, test, and deploy applications in isolated, consistent environments, while reducing the costs and complexities associated with physical infrastructure.

code execution at the client

In DevOps, code execution at the client refers to the process of executing code or scripts on client devices or machines. This can be accomplished in several ways, including:

Client-side scripting languages: JavaScript, HTML, and CSS are commonly used client-side scripting languages that run in a web browser and allow developers to create dynamic, interactive web pages.

Remote execution tools: Tools such as SSH, Telnet, or Remote Desktop Protocol (RDP) allow developers to remotely execute commands and scripts on client devices.

Configuration management tools: Tools such as Ansible, Puppet, or Chef use agent-based or agentless architectures to manage and configure client devices, allowing developers to execute code and scripts remotely.

Mobile apps: Mobile applications can also run code on client devices, allowing developers to create dynamic, interactive experiences for users.

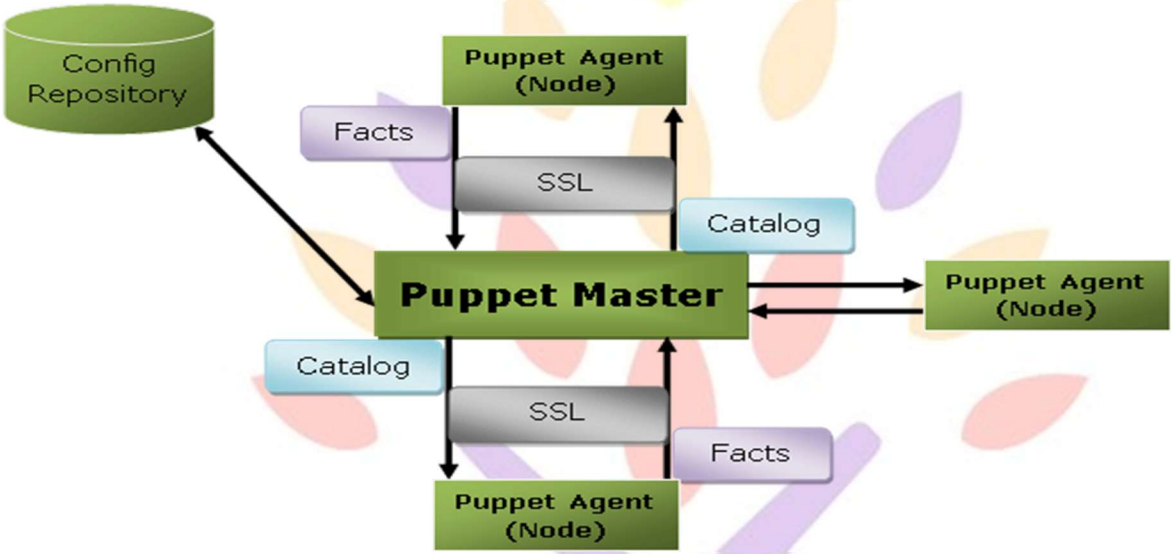
These methods are used in DevOps to automate various tasks, such as application deployment, software updates, or system configuration, on client devices. By executing code on the client side, DevOps teams can improve the speed, reliability, and security of their software delivery process.

your roots to success...

Puppet master and agents:

Puppet Architecture

Puppet uses master-slave or client-server architecture. Puppet client and server interconnected by SSL, which is a secure socket layer. It is a model-driven system.



Here, the client is referred to as a Puppet agent/slave/node, and the server is referred to as a Puppet master.

Let's see the components of Puppet architecture:

Puppet Master

Puppet master handles all the configuration related process in the form of puppet codes. It is a Linux based system in which puppet master software is installed. The puppet master must be in Linux. It uses the puppet agent to apply the configuration to nodes.

This is the place where SSL certificates are checked and marked.

Puppet Slave or Agent

Puppet agents are the real working systems and used by the Client. It is installed on the client machine and maintained and managed by the puppet master. They have a puppet agent service running inside them.

The agent machine can be configured on any operating system such as Windows, Linux, Solaris, or Mac OS.

Config Repository

Config repository is the storage area where all the servers and nodes related configurations are stored, and we can pull these configurations as per requirements.

Facts

Facts are the key-value data pair. It contains information about the node or the master machine. It represents a puppet client states such as operating system, network interface, IP address, uptime, and whether the client machine is virtual or not.

These facts are used for determining the present state of any agent. Changes on any target machine are made based on facts. Puppet's facts are predefined and customized.

Catalog

The entire configuration and manifest files that are written in Puppet are changed into a compiled format. This compiled format is known as a catalog, and then we can apply this catalog to the target machine.

The above image performs the following functions:

- First of all, an agent node sends facts to the master or server and requests for a catalog.
- The master or server compiles and returns the catalog of a node with the help of some information accessed by the master.
- Then the agent applies the catalog to the node by checking every resource mentioned in the catalog. If it identifies resources that are not in their desired state, then makes the necessary adjustments to fix them. Or, it determines in no-op mode, the adjustments would be required to reconcile the catalog.
- And finally, the agent sends a report back to the master.

Puppet Master-Slave Communication

Puppet master-slave communicates via a secure encrypted channel through the SSL (Secure Socket Layer). Let's see the below diagram to understand the communication between the master and slave with this channel:

your roots to success...

The above diagram depicts the following:

- Puppet slave requests for Puppet Master Certificate.
- Puppet master sends the Master Certificate to the puppet slave in response to the client request.
- Puppet master requests to the Puppet slave for the slave certificate.
- Puppet slave sends the requested slave certificate to the puppet master.
- Puppet slave sends a request for data to the puppet master.
- Finally, the master sends the data to the puppet slave as per the request.

Puppet Blocks

Puppet provides the flexibility to integrate Reports with third-party tools using Puppet APIs.

Four types of Puppet building blocks are

1. Resources
2. Classes
3. Manifest
4. Modules

Puppet Resources:

Puppet Resources are the building blocks of Puppet.

Resources are the **inbuilt functions** that run at the back end to perform the required operations in puppet.

Puppet Classes:

A combination of different resources can be grouped together into a single unit called class.

Puppet Manifest:

Manifest is a directory containing puppet DSL files. Those files have a .pp extension. The .pp extension stands for puppet program. The puppet code consists of definitions or declarations of Puppet Classes.

Puppet Modules:

Modules are a collection of files and directories such as Manifests, Class definitions. They are the re-usable and sharable units in Puppet.

For example, the MySQL module to install and configure MySQL or the Jenkins module to manage Jenkins, etc.

Ansible:

Ansible is simple open source IT engine which automates application deployment, intra service orchestration, cloud provisioning and many other IT tools.

Ansible is easy to deploy because it does not use any agents or custom security infrastructure.

Ansible uses playbook to describe automation jobs, and playbook uses very simple language i.e. **YAML** (It's a human-readable data serialization language & is commonly used for configuration files, but could be used in many applications where data is being stored) which is very easy for humans to understand, read and write. Hence the advantage is that even the IT infrastructure support guys can read and understand the playbook and debug if needed (YAML – It is in human readable form).

Ansible is designed for multi-tier deployment. Ansible does not manage one system at time; it models IT infrastructure by describing all of your systems are interrelated. Ansible is completely agentless which means Ansible works by connecting your nodes through ssh (by default). But if you want other method for connection like Kerberos, Ansible gives that option to you.

After connecting to your nodes, Ansible pushes small programs called as "Ansible Modules". Ansible runs that modules on your nodes and removes them when finished. Ansible manages your inventory in simple text files (These are the hosts file). Ansible uses the hosts file where one can group the hosts and can control the actions on a specific group in the playbooks.

Sample Hosts File

This is the content of hosts file −

```
#File name: hosts
#Description: Inventory file for your application. Defines machine type abc
node to deploy specific artifacts
# Defines machine type def node to upload
metadata.

[abc-node]
#server1 ansible_host = <target machine for DU deployment> ansible_user = <Ansible
user> ansible_connection = ssh
server1 ansible_host = <your host name> ansible_user = <your unix user>
ansible_connection = ssh

[def-node]
#server2 ansible_host = <target machine for artifact upload>
ansible_user = <Ansible user> ansible_connection = ssh
server2 ansible_host = <host> ansible_user = <user> ansible_connection = ssh
```

your roots to success...

What is Configuration Management

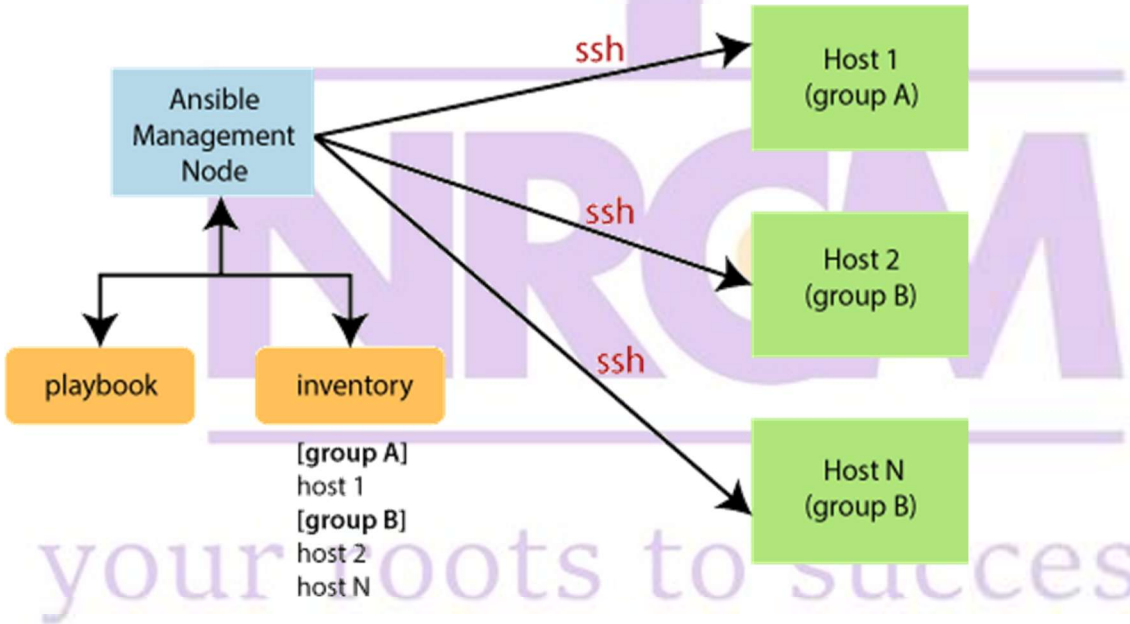
Configuration management in terms of Ansible means that it maintains configuration of the product performance by keeping a record and updating detailed information which describes an enterprise's hardware and software.

Such information typically includes the exact versions and updates that have been applied to installed software packages and the locations and network addresses of hardware devices. For e.g. If you want to install the new version of WebLogic/WebSphere server on all of the machines present in your enterprise, it is not feasible for you to manually go and update each and every machine.

You can install WebLogic/WebSphere in one go on all of your machines with Ansible playbooks and inventory written in the most simple way. All you have to do is list out the IP addresses of your nodes in the inventory and write a playbook to install WebLogic/WebSphere. Run the playbook from your control machine & it will be installed on all your nodes.

Ansible Workflow

Ansible works by connecting to your nodes and pushing out a small program called **Ansible modules** to them. Then Ansible executed these modules and removed them after finished. The library of modules can reside on any machine, and there are no daemons, **servers,** or **databases** required.



In the above image, the **Management Node** is the controlling node that controls the entire execution of the playbook. The **inventory** file provides the list of hosts where the Ansible

modules need to be run. The **Management Node** makes an **SSH** connection and executes the small modules on the host's machine and install the software.

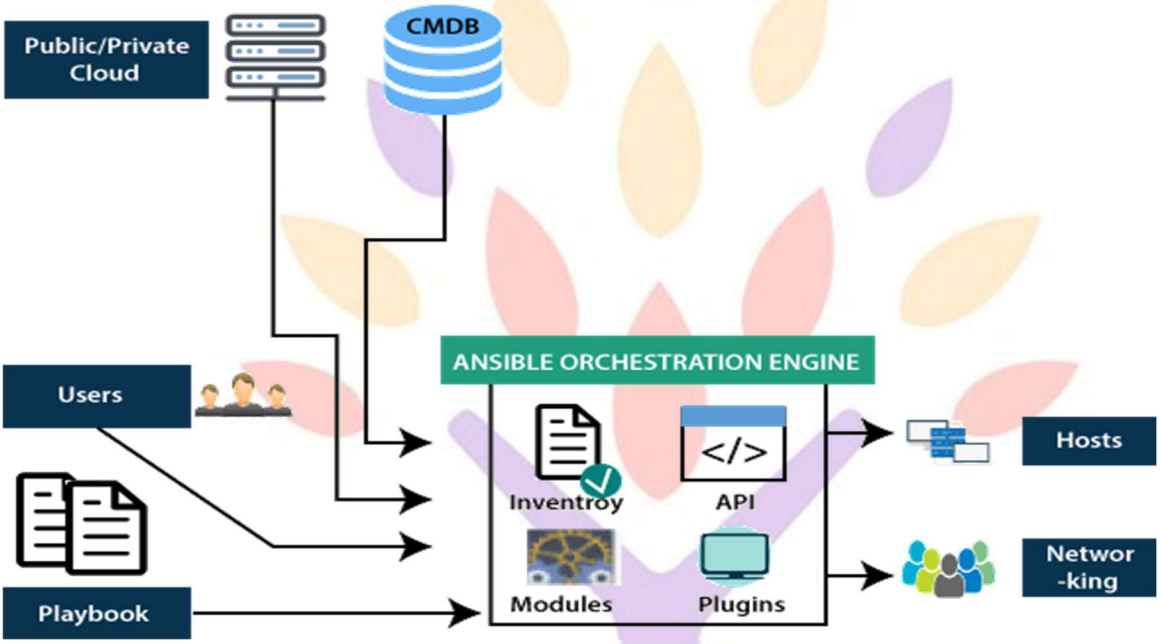
Ansible removes the modules once those are installed so expertly. It connects to the host machine executes the instructions, and if it is successfully installed, then remove that code in which one was copied on the host machine.

Terms used in Ansible

Terms	Explanation
Ansible Server	It is a machine where Ansible is installed and from which all tasks and playbooks will be executed.
Modules	The module is a command or set of similar commands which is executed on the client-side.
Task	A task is a section which consists of a single procedure to be completed.
Role	It is a way of organizing tasks and related files to be later called in a playbook.
Fact	The information fetched from the client system from the global variables with the gather facts operation.
Inventory	A file containing the data regarding the Ansible client-server.
Play	It is the execution of the playbook.
Handler	The task is called only if a notifier is present.
Notifier	The section attributed to a task which calls a handler if the output is changed.
Tag	It is a name set to a task that can be used later on to issue just that specific task or group of jobs.

Ansible Architecture

The Ansible orchestration engine interacts with a user who is writing the Ansible playbook to execute the Ansible orchestration and interact along with the services of private or public cloud and configuration management database. You can show in the below diagram, such as:



Inventory

Inventory is lists of nodes or hosts having their IP addresses, databases, servers, etc. which are need to be managed.

API's

The Ansible API's works as the transport for the public or private cloud services.

Modules

Ansible connected the nodes and spread out the Ansible modules programs. Ansible executes the modules and removed after finished. These modules can reside on any machine; no database or servers are required here. You can work with the chose text editor or a terminal or version control system to keep track of the changes in the content.

NRCM your roots to success...

Plugins

Plugins is a piece of code that expends the core functionality of Ansible. There are many useful plugins, and you also can write your own.

Playbooks

Playbooks consist of your written code, and they are written in YAML format, which describes the tasks and executes through the Ansible. Also, you can launch the tasks synchronously and asynchronously with playbooks.

Hosts

In the Ansible architecture, hosts are the node systems, which are automated by Ansible, and any machine such as RedHat, Linux, Windows, etc.

Networking

Ansible is used to automate different networks, and it uses the simple, secure, and powerful agentless automation framework for IT operations and development. It uses a type of data model which separated from the Ansible automation engine that spans the different hardware quite easily.

Cloud

A cloud is a network of remote servers on which you can store, manage, and process the data. These servers are hosted on the internet and storing the data remotely rather than the local server. It just launches the resources and instances on the cloud, connect them to the servers, and you have good knowledge of operating your tasks remotely.

CMDB

CMDB is a type of repository which acts as a data warehouse for the IT installations.

Puppet Components

Following are the key components of Puppet:

- Manifests
- Module
- Resource

your roots to success...

- Factor
- M-collective
- Catalogs
- Class
- Nodes

Let's understand these components in detail:

Manifests

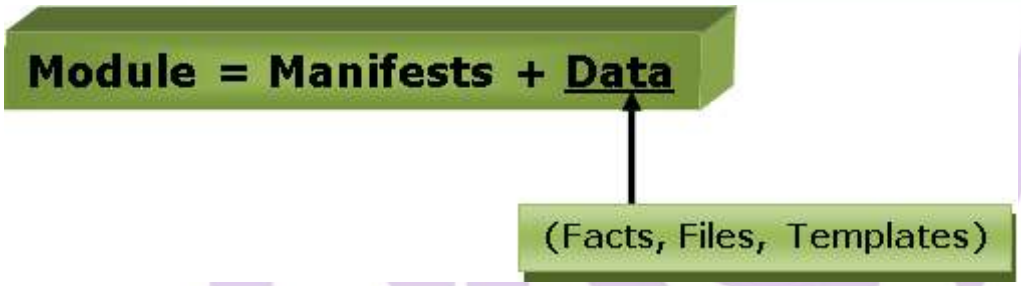
Puppet Master contains the Puppet Slave's configuration details, and these are written in Puppet's native language.

Manifest is nothing but the files specifying the configuration details for Puppet slave. The extension for manifest files is .pp, which means Puppet Policy. These files consist of puppet scripts describing the configuration for the slave.

Module

The puppet module is a set of manifests and data. Here data is file, facts, or templates. The module follows a specific directory structure. These modules allow the puppet program to split into multiple manifests. Modules are simply self-contained bundles of data or code.

Let's understand the module by the following image:



Resource

Resources are a basic unit of system configuration modeling. These are the predefined functions that run at the backend to perform the necessary operations in the puppet.

Each puppet resource defines certain elements of the system, such as some particular service or package.

Factor

The factor collects facts or important information about the puppet slave. Facts are the key-value data pair. It contains information about the node or the master machine. It represents a puppet client state such as operating system, network interface, IP address, uptime, and whether the client machine is virtual or not.

These facts are used for determining the present state of any agent. Changes on any target machine are made based on facts. Puppet's facts are predefined and customized.

M-Collective

M-collective is a framework that enables parallel execution of several jobs on multiple Slaves. This framework performs several functions, such as:

- This is used to interact with clusters of puppet slaves; they can be in small groups or very large deployments.
- To transmit demands, use a broadcast model. All Slaves receive all requests at the same time, requests have filters attached, and only Slaves matching the filter can act on requests.
- This is used to call remote slaves with the help of simple command-line tools.
- This is used to write custom reports about your infrastructure.

Catalogs

The entire configuration and manifest files that are written in Puppet are changed into a compiled format. This compiled format is known as a catalog, and then we can apply this catalog to the target machine.

All the required states of slave resources are described in the catalog.

Class

Like other programming languages, the puppet also supports a class to organize the code in a better way. Puppet class is a collection of various resources that are grouped into a single unit.

Nodes

The nodes are the location where the puppet slaves are installed used to manage all the clients and servers.

NRCM your roots to success...

Deployment tools

Chef

Chef is an open source technology developed by Opscode. Adam Jacob, co-founder of Opscode is known as the founder of Chef. This technology uses Ruby encoding to develop basic building blocks like recipe and cookbooks. Chef is used in infrastructure automation and helps in reducing manual and repetitive tasks for infrastructure management.

Chef have got its own convention for different building blocks, which are required to manage and automate infrastructure.

Why Chef?

Chef is a configuration management technology used to automate the infrastructure provisioning. It is developed on the basis of Ruby DSL language. It is used to streamline the task of configuration and managing the company's server. It has the capability to get integrated with any of the cloud technology.

In DevOps, we use Chef to deploy and manage servers and applications in-house and on the cloud.

Features of Chef

Following are the most prominent features of Chef −

- Chef uses popular Ruby language to create a domain-specific language.
- Chef does not make assumptions on the current status of a node. It uses its mechanisms to get the current status of machine.
- Chef is ideal for deploying and managing the cloud server, storage, and software.

Advantages of Chef

Chef offers the following advantages −

- **Lower barrier for entry** − As Chef uses native Ruby language for configuration, a standard configuration language it can be easily picked up by anyone having some development experience.
- **Excellent integration with cloud** − Using the knife utility, it can be easily integrated with any of the cloud technologies. It is the best tool for an organization that wishes to distribute its infrastructure on multi-cloud environment.

Disadvantages of Chef

Some of the major drawbacks of Chef are as follows −

NRCM your roots to success...

- One of the huge disadvantages of Chef is the way cookbooks are controlled. It needs constant babying so that people who are working should not mess up with others cookbooks.
- Only Chef solo is available.
- In the current situation, it is only a good fit for AWS cloud.
- It is not very easy to learn if the person is not familiar with Ruby.
- Documentation is still lacking.

Key Building Blocks of Chef

Recipe

It can be defined as a collection of attributes which are used to manage the infrastructure. These attributes which are present in the recipe are used to change the existing state or setting a particular infrastructure node. They are loaded during Chef client run and comparted with the existing attribute of the node (machine). It then gets to the status which is defined in the node resource of the recipe. It is the main workhorse of the cookbook.

Cookbook

A cookbook is a collection of recipes. They are the basic building blocks which get uploaded to Chef server. When Chef run takes place, it ensures that the recipes present inside it gets a given infrastructure to the desired state as listed in the recipe.

Resource

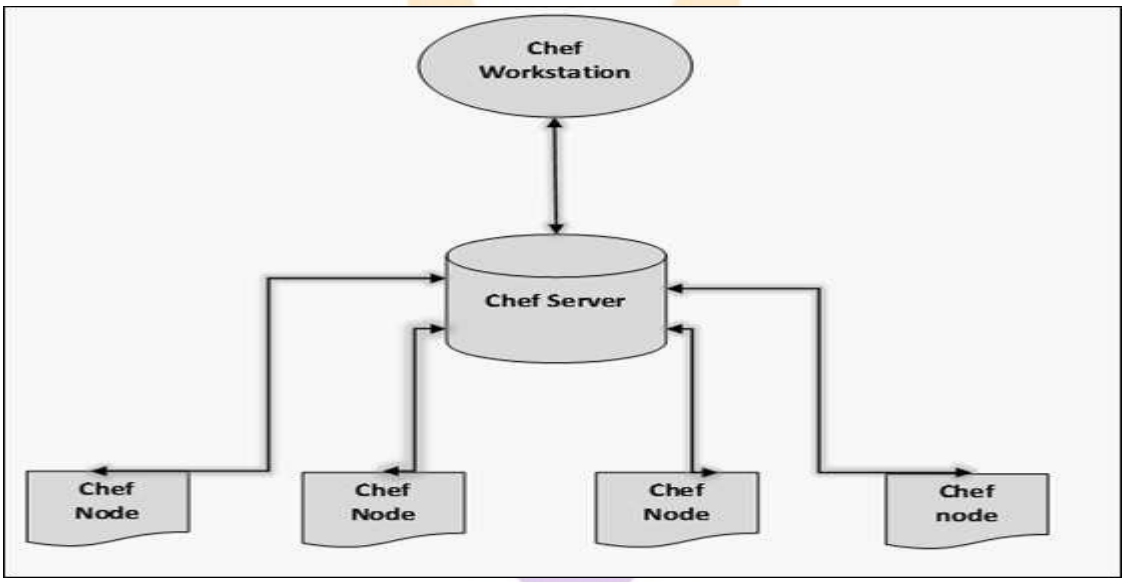
It is the basic component of a recipe used to manage the infrastructure with different kind of states. There can be multiple resources in a recipe, which will help in configuring and managing the infrastructure. For example −

- **package** − Manages the packages on a node
- **service** − Manages the services on a node
- **user** − Manages the users on the node
- **group** − Manages groups
- **template** − Manages the files with embedded Ruby template
- **cookbook_file** − Transfers the files from the files subdirectory in the cookbook to a location on the node
- **file** − Manages the contents of a file on the node
- **directory** − Manages the directories on the node
- **execute** − Executes a command on the node
- **cron** − Edits an existing cron file on the node

NRCM your roots to success...

Chef - Architecture

- Chef works on a three-tier client server model wherein the working units such as cookbooks are developed on the Chef workstation. From the command line utilities such as knife, they are uploaded to the Chef server and all the nodes which are present in the architecture are registered with the Chef server.



- In order to get the working Chef infrastructure in place, we need to set up multiple things in sequence.
- In the above setup, we have the following components.
- Chef Workstation
- This is the location where all the configurations are developed. Chef workstation is installed on the local machine. Detailed configuration structure is discussed in the later chapters of this tutorial.
- Chef Server
- This works as a centralized working unit of Chef setup, where all the configuration files are uploaded post development. There are different kinds of Chef server, some are hosted Chef server whereas some are built-in premise.
- Chef Nodes
- They are the actual machines which are going to be managed by the Chef server. All the nodes can have different kinds of setup as per requirement. Chef client is the key component of all the nodes, which helps in setting up the communication between the Chef server and Chef node. The other components of Chef node is Ohai, which helps in getting the current state of any node at a given point of time.

NRCM your roots to success...

Salt Stack

Salt Stack is an open-source configuration management software and remote execution engine. Salt is a command-line tool. While written in Python, SaltStack configuration management is language agnostic and simple. Salt platform uses the push model for executing commands via the SSH protocol. The default configuration system is **YAML** and **Jinja templates**. Salt is primarily competing with **Puppet**, **Chef** and **Ansible**.

Salt provides many features when compared to other competing tools. Some of these important features are listed below.

- **Fault tolerance** − Salt minions can connect to multiple masters at one time by configuring the master configuration parameter as a YAML list of all the available masters. Any master can direct commands to the Salt infrastructure.
- **Flexible** − The entire management approach of Salt is very flexible. It can be implemented to follow the most popular systems management models such as Agent and Server, Agent-only, Server-only or all of the above in the same environment.
- **Scalable Configuration Management** − SaltStack is designed to handle ten thousand minions per master.
- **Parallel Execution model** − Salt can enable commands to execute remote systems in a parallel manner.
- **Python API** − Salt provides a simple programming interface and it was designed to be modular and easily extensible, to make it easy to mold to diverse applications.
- **Easy to Setup** − Salt is easy to setup and provides a single remote execution architecture that can manage the diverse requirements of any number of servers.
- **Language Agnostic** − Salt state configuration files, templating engine or file type supports any type of language.

Benefits of SaltStack

Being simple as well as a feature-rich system, Salt provides many benefits and they can be summarized as below −

- **Robust** − Salt is powerful and robust configuration management framework and works around tens of thousands of systems.
- **Authentication** − Salt manages simple SSH key pairs for authentication.
- **Secure** − Salt manages secure data using an encrypted protocol.
- **Fast** − Salt is very fast, lightweight communication bus to provide the foundation for a remote execution engine.
- **Virtual Machine Automation** − The Salt Virt Cloud Controller capability is used for automation.

NRCM your roots to success...

- **Infrastructure as data, not code** − Salt provides a simple deployment, model driven configuration management and command execution framework.

Introduction to ZeroMQ

Salt is based on the **ZeroMQ** library and it is an embeddable networking library. It is lightweight and a fast messaging library. The basic implementation is in **C/C++** and native implementations for several languages including **Java** and **.Net** is available.

ZeroMQ is a broker-less peer-peer message processing. ZeroMQ allows you to design a complex communication system easily.

ZeroMQ comes with the following five basic patterns −

- **Synchronous Request/Response** − Used for sending a request and receiving subsequent replies for each one sent.
- **Asynchronous Request/Response** − Requestor initiates the conversation by sending a Request message and waits for a Response message. Provider waits for the incoming Request messages and replies with the Response messages.
- **Publish/Subscribe** − Used for distributing data from a single process (e.g. publisher) to multiple recipients (e.g. subscribers).
- **Push/Pull** − Used for distributing data to connected nodes.
- **Exclusive Pair** − Used for connecting two peers together, forming a pair.

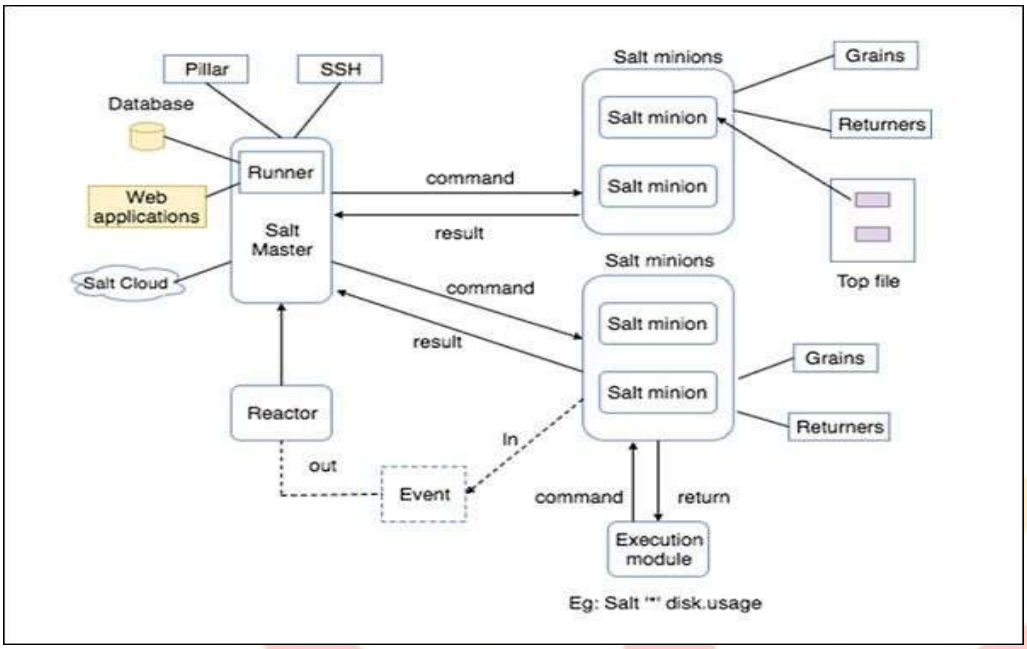
ZeroMQ is a highly flexible networking tool for exchanging messages among clusters, cloud and other multi system environments. ZeroMQ is the **default transport library** presented in SaltStack.

SaltStack – Architecture

The architecture of SaltStack is designed to work with any number of servers, from local network systems to other deployments across different data centers. Architecture is a simple server/client model with the needed functionality built into a single set of daemons.

Take a look at the following illustration. It shows the different components of SaltStack architecture.

NRCM your roots to success...



- **SaltMaster** − SaltMaster is the master daemon. A SaltMaster is used to send commands and configurations to the Salt slaves. A single master can manage multiple masters.
- **SaltMinions** − SaltMinion is the slave daemon. A Salt minion receives commands and configuration from the SaltMaster.
- **Execution** − Modules and Adhoc commands executed from the command line against one or more minions. It performs Real-time Monitoring.
- **Formulas** − Formulas are pre-written Salt States. They are as open-ended as Salt States themselves and can be used for tasks such as installing a package, configuring and starting a service, setting up users or permissions and many other common tasks.
- **Grains** − Grains is an interface that provides information specific to a minion. The information available through the grains interface is static. Grains get loaded when the Salt minion starts. This means that the information in grains is unchanging. Therefore, grains information could be about the running kernel or the operating system. It is case insensitive.
- **Pillar** − A pillar is an interface that generates and stores highly sensitive data specific to a particular minion, such as cryptographic keys and passwords. It stores data in a key/value pair and the data is managed in a similar way as the Salt State Tree.
- **Top File** − Matches Salt states and pillar data to Salt minions.
- **Runners** − It is a module located inside the SaltMaster and performs tasks such as job status, connection status, read data from external APIs, query connected salt minions and more.
- **Returners** − Returns data from Salt minions to another system.

your roots to success...

- **Reactor** − It is responsible for triggering reactions when events occur in your SaltStack environment.
- **SaltCloud** − Salt Cloud provides a powerful interface to interact with cloud hosts.
- **SaltSSH** − Run Salt commands over SSH on systems without using Salt minion.

Docker

Docker is a container management service. The keywords of Docker are **develop, ship** and **run** anywhere. The whole idea of Docker is for developers to easily develop applications, ship them into containers which can then be deployed anywhere.

The initial release of Docker was in March 2013 and since then, it has become the buzzword for modern world development, especially in the face of Agile-based projects.

Features of Docker

- Docker has the ability to reduce the size of development by providing a smaller footprint of the operating system via containers.
- With containers, it becomes easier for teams across different units, such as development, QA and Operations to work seamlessly across applications.
- You can deploy Docker containers anywhere, on any physical and virtual machines and even on the cloud.
- Since Docker containers are pretty lightweight, they are very easily scalable.

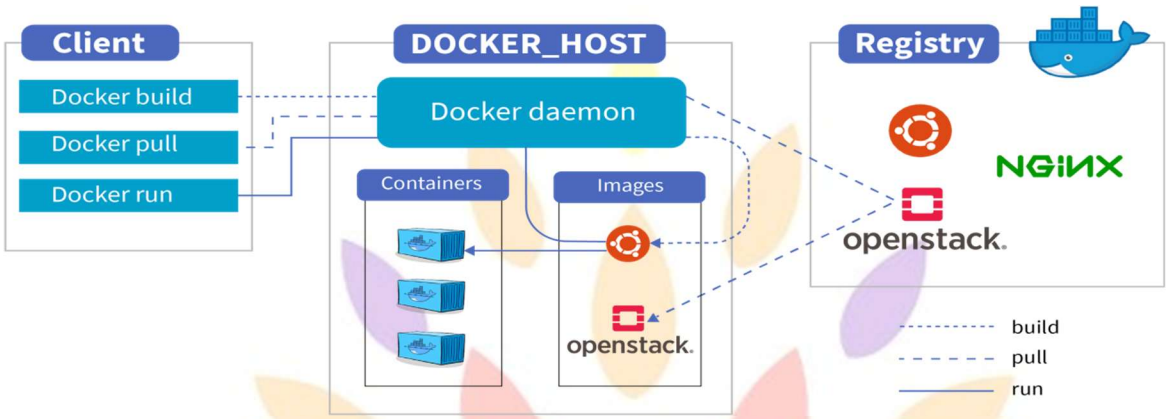
Components of Docker

Docker has the following components

- **Docker for Mac** − It allows one to run Docker containers on the Mac OS.
- **Docker for Linux** − It allows one to run Docker containers on the Linux OS.
- **Docker for Windows** − It allows one to run Docker containers on the Windows OS.
- **Docker Engine** − It is used for building Docker images and creating Docker containers.
- **Docker Hub** − This is the registry which is used to host various Docker images.
- **Docker Compose** − This is used to define applications using multiple Docker containers.

Docker architecture

- Docker uses a client-server architecture. The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.



The Docker daemon

The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

The Docker client

The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

Docker Desktop

Docker Desktop is an easy-to-install application for your Mac, Windows or Linux environment that enables you to build and share containerized applications and microservices. Docker Desktop includes the Docker daemon (dockerd), the Docker client (docker), Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper. For more information, see Docker Desktop.

Docker registries

A Docker *registry* stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

Images

An *image* is a read-only template with instructions for creating a Docker container. Often, an image is *based on* another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.
You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a *Dockerfile* with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

Containers

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

Example docker run command
The following command runs an ubuntu container, attaches interactively to your local command-line session, and runs /bin/bash.
$ docker run -i -t ubuntu /bin/bash

When you run this command, the following happens (assuming you are using the default registry configuration):

1. If you do not have the ubuntu image locally, Docker pulls it from your configured registry, as though you had run docker pull ubuntu manually.
2. Docker creates a new container, as though you had run a docker container create command manually.
3. Docker allocates a read-write filesystem to the container, as its final layer. This allows a running container to create or modify files and directories in its local filesystem.
4. Docker creates a network interface to connect the container to the default network, since you did not specify any networking options. This includes assigning an IP address to the container. By default, containers can connect to external networks using the host machine's network connection.
5. Docker starts the container and executes /bin/bash. Because the container is running interactively and attached to your terminal (due to the -i and -t flags), you can provide input using your keyboard while the output is logged to your terminal.
6. When you type exit to terminate the /bin/bash command, the container stops but is not removed. You can start it again or remove it.

The underlying technology

Docker is written in the Go programming language and takes advantage of several features of the Linux kernel to deliver its functionality. Docker uses a technology called namespaces to provide the isolated workspace called the container. When you run a container, Docker creates a set of namespaces for that container.

These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

NRCM
your roots to success...