Ansible-playbook [playbook-name] –check

Will show you what happen in case of playbook execution

**Configuration Testing**: is a type of software testing that verifies the performance of the system under development against various combinations of software and hardware to find the best configuration under which the system can work without any flaws or issues while matching its functional requirements.

Different configuration of software and hardware mean the multiple OS versions, various browsers, various supported drivers, distinct memory sizes, different hard drive types, various types of CPU, etc.

**Objectives:**

1. Adaptability to different configurations
2. Evaluation of stability
3. Testing the user experience
4. Security throughout configurations
5. Compatibility of networks
6. Data compatibility

**Types:**

1. Software configuration testing

When the build is released, software configuration begins after passing through the unit test and integration test.

1. Hardware configuration testing

**Another classification:**

1. Client level testing

is associated with the usability and functionality testing. This test is node from the point of view of its direct interest of the users.

1. Server level testing

This type of configuration testing is carried out to determine the communication between the software and the external environment when it is planned to be integrated after the release.

Source: <https://www.geeksforgeeks.org/software-testing-configuration-testing/>

**Managing configuration diversity**

One of the key challenge in configuration testing in software engineering is the management of configuration diversity.

As software applications need to support a wide range of configurations, it can be challenging to ensure that all configurations are adequately tested.

This requires careful planning, prioritization, and the allocation of resources to ensure that the most critical configurations are thoroughly tested.

**Dealing with configuration interactions**

Configurations within a software system are often interconnected, and changes to one configuration can have unintended consequences on others.

Testing the interactions between configurations can be complex and time-consuming.

It is crucial to identify and test specific configuration combinations that are likely to have interdependencies to reduce the risk of compatibility issues.

Source: <https://www.institutedata.com/blog/configuration-testing-in-software-engineering/>

**Software Configuration Testing**

Software configuration testing can typically begin when:

* Configurability requirements to be tested are specified
* Test environment is ready
* Testing team is well trained in configuration testing
* Build release is unit and integration test passed

Some applications are going to pass the unit and integration testing in the test lab where all the prerequisites are present, but once the same application is installed in a client place and the machines are missing some software’s updates or the versions on which the application is dependent directly or indirectly there is a change that the application might fail. To avoid this kind of situation, it’s always suggested to fail the tests manually be removing some of the configurability requirements and then proceed with the testing.

**Hardware Configuration Testing**

It is the duty of tester to analyze what hardware is mostly used by the users and try to make the testing based on prioritization.

Source: <https://www.guru99.com/configuration-testing.html>

Configuration refers to any setting used within software.

Types:

1. Static Configuration

These configurations are set once and generally remain unchanged.

1. Customer Specific Configuration

These configurations are managed regardless if whether the software architecture is single-tenancy or multi-tenancy.

1. Environment Specific Configuration

These configurations vary depending on the environment in which the application operates. These environments are primarily created to isolate customers for different environments.

1. Feature Toggles

When developing a new feature or enhancing an existing one in a product, deploying directly to production can be risky duo to the large customer base. In the CD approach, product changes are initially deployed to production for a limited set of users, known as the pilot phase. Feedback is collected from these users before a full-scale release to ensure stability in the production environment.

Additionally, there are cases where customers request to temporarily disable specific changes. In response, developers must perform rollbacks.

To address these scenarios, we utilize feature toggles, which involve keeping configurations specific to application features.

**Handling Static Configuration:**

For one-time activities can be managed be the DevOps team during the initial stages of creating the production infrastructure

**Handling Customer-specific Configuration:**

To manage customer-specific configuration in both single-tenant and multi-tenancy systems, a tenant configuration table can be employed.

Environment files and third-party configuration services are avoided where they load configurations at application startup, making it challenging to manage dynamic updates.

Using these approaches requires involving DevOps to restart the application which may impact business operations.

Source: <https://medium.com/@nikhilag/simplifying-configuration-management-to-reduce-stress-for-developers-devops-7e95ef3f2691>

Application Configuration Tables (Also Solves Feature Toggles)

By categorizing configurations into two types:

1. Infrastructure Related Configurations
2. Feature Related Configurations

Source: <https://medium.com/@nikhilag/configuration-management-environment-specific-configuration-and-feature-toggles-abedebb32f44>