**Week 11**

**Application Testing**

**What is Application Testing?**

Application Testing is defined as a software testing type conducted through scripts with the motive of finding errors in software. It deals with tests for the entire application.

It helps to enhance the quality of your applications while reducing costs, maximizing ROI, and saving development time.

**What is Manual Testing?**

Manual testing is testing of the software where tests are executed manually by a QA Analyst. It is performed to discover bugs in software under development.

## What is Automation Testing?

In Automated Software Testing, testers write code/test scripts to automate test execution. Testers use appropriate automation tools to develop the test scripts and validate the software. The goal is to complete test execution in a less amount of time.

**Manual Testing Vs. Automation Testing**

| **Parameter** | **Automation Testing** | **Manual Testing** |
| --- | --- | --- |
| Definition | Automation Testing uses automation tools to execute test cases. | In manual testing, test cases are executed by a human tester and software. |
| Processing time | Automated testing is significantly faster than a manual approach. | Manual testing is time-consuming and takes up human resources. |
| Reliability | Automated testing is a reliable method, as it is performed by tools and scripts. There is no testing Fatigue. | Manual testing is not as accurate because of the possibility of the human errors. |
| Investment | Investment is required for testing tools as well as automation engineers | Investment is needed for human resources. |
| Cost-effective | Not cost effective for low volume regression | Not cost effective for high volume regression. |
| Test Report Visibility | With automation testing, all stakeholders can login into the automation system and check test execution results | Manual Tests are usually recorded in an Excel or Word, and test results are not readily/ readily available. |
| Batch testing | You can Batch multiple Test Scripts for nightly execution. | Manual tests cannot be batched. |
| Programming knowledge | Programming knowledge is a must in automation testing. | No need for programming in Manual Testing. |
| Deadlines | Automated Tests have zero risks of missing out a pre-decided test. | Manual Testing has a higher risk of missing out the pre-decided test deadline. |
| When to Use? | Automated Testing is suited for Regression Testing, Performance Testing, Load Testing or highly repeatable functional test cases. | Manual Testing is suitable for Exploratory, Usability and Adhoc Testing. It should also be used where the AUT changes frequently. |

**Application testing tools**

There are various testing tools for Application Testing. Selection of tools depends on what type of testing you want to perform. For different platform different tools are recommended. Application testing tools ensure performance, usability and functionality of applications across a variety of devices.

* [Selenium](https://www.guru99.com/selenium-tutorial.html)
* IBM Rational Robot
* RFT (Rational Functional Tester)
* [Load Runner](https://www.guru99.com/loadrunner-v12-tutorials.html) ( HP Performance Tester)
* [Apache Jmeter](https://www.guru99.com/jmeter-tutorials.html)

**FUNCTIONAL TESTING**

Functional testing  is a type of software testing that validates the software system against the functional requirements/specifications. The purpose of Functional tests is to test each function of the software application, by providing appropriate input, verifying the output against the Functional requirements.

**How to do Functional Testing** :

* Understand the Functional Requirements
* Identify test input or test data based on requirements
* Compute the expected outcomes with selected test input values
* Execute test cases
* Compare actual and computed expected results

## What is GUI Testing?

**GUI Testing** is a software testing type that checks the Graphical User Interface of the Software. The purpose of Graphical User Interface (GUI) Testing is to ensure the functionalities of software application work as per specifications by checking screens and controls like menus, buttons, icons, etc.

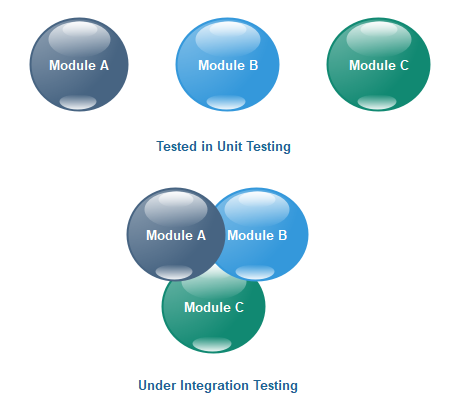
## What do you Check-in GUI Testing?

**The following checklist will ensure detailed GUI Testing in Software Testing.**

* Check all the GUI elements for size, position, width, length, and acceptance of characters or numbers. For instance, you must be able to provide inputs to the input fields.
* Check you can execute the intended functionality of the application using the GUI
* Check Error Messages are displayed correctly
* Check for Clear demarcation of different sections on screen
* Check Font used in an application is readable
* Check the alignment of the text is proper
* Check the Color of the font and warning messages is aesthetically pleasing
* Check that the images have good clarity
* Check that the images are properly aligned
* Check the positioning of GUI elements for different screen resolution.

## What is Integration Testing?

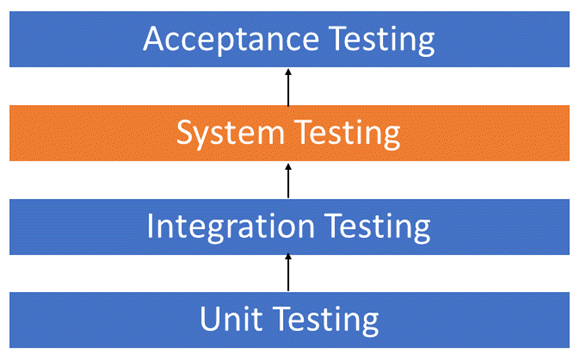
**Integration Testing** is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated



**System testing**

**System Testing** is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications.

Integrate the work of each group and carry out integration testing



**Acceptance testing**

Acceptance Testing is the last phase of software testing performed after System Testing and before making the system available for actual use.

**Acceptance tests and test plan**

The acceptance test plan or system test plan is based on the requirement specifications and is required for a formal test environment.

Acceptance evaluates the functionality and performance of the entire application and consists of a variety of tests like.

a) Performance Tests

b) Usability Tests  
c) Stress Tests  
d) Documentation Tests  
e) Security Tests  
f) Volume Tests  
g) Recovery Tests etc.

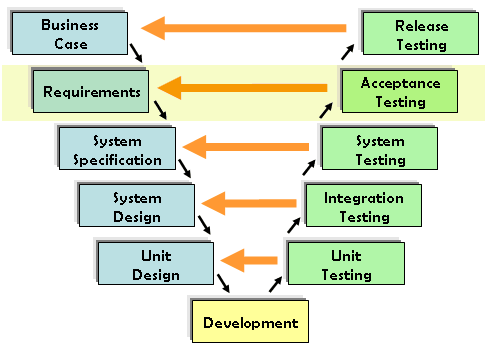
**User acceptance testing**

**User Acceptance Testing (UAT)** is a type of testing performed by the end user or the client to verify/accept the software system before moving the software application to the production environment. UAT is done in the final phase of testing after functional, integration and system testing is done.

UAT is performed by –

 Client

 End users



**Bug tracking – using Jira or similar tools**

Bug and issue tracking tools help software teams find, record, and track bugs in their software. It's critical that everyone on your team is able to find and record bugs, and even more important, assign them to the right team members at the right time.

### Step 1 - Capture and track bugs in your software

### Creating issue

### Step 2 - Assign and prioritize with ease

### Assigning and prioritizing tickets

### Step 3 - Track bugs from backlog to done

### Moving tickets in a kanban board

### Step 4 - Stay up to date with notifications

### Preview of tickets assigned to one person

## Step 5 - Go from backlog to release with ease

**Deployment process**

* Software deployment includes all of the steps, processes, and activities that are required to make a software system or update available to its intended users.
* Today, most IT organizations and software developers deploy software updates, patches and new applications with a combination of manual and automated processes.

Different types of application or software deployment strategies are:

* Basic
* Multi-service
* Rolling
* [Blue Green Deployment](https://www.sumologic.com/glossary/blue-green-deployment/)
* Canary
* A/B Testing

**Manual deployment**

Manual configuration is intended for advanced users when more control of the application server environment is required. When the configuration utility is run with the manual deployment mode selected, it makes necessary changes to the database, but your application server is not changed. Deployable application files, installation scripts, and properties files are generated in the <repository installation directory>/toDeploy/<timestamp> directory.

**Automated deployment**

Deployment automation is the use of automated processes to move your software between testing and production environments. This process is both reliable and repeatable across the [software delivery lifecycle (SDLC)](https://www.bmc.com/blogs/sdlc-software-development-lifecycle/).

Some of the best deployment automation tools available include:

* [Jenkins](https://www.jenkins.io/)
* [Visual Studio](https://visualstudio.microsoft.com/)
* [AWS CodeDeploy](https://aws.amazon.com/codedeploy/)

# Deployment automation benefits

* Anyone can deploy
* Faster, more efficient deployment
* Increased productivity
* Fewer errors
* More frequent releases
* Immediate feedback

**How to Implement Automated Deployment**

Follow these steps to set up a basic deployment pipeline:

1. **Set up a Continuous Integration server**
   * Choose an existing server, such as [Jenkins](https://jenkins.io/), or [go.cd](https://www.gocd.org/).
   * Set up your first build (a link to your source code, for example, on GitHub and a command to build the application)
   * Ensure that it runs every time a change is checked into source control.
2. **Set up a few test suites**
   * Set up a separate build for each test suit and link the builds like dominos, so that each is triggered by the one before it. This way you see very quickly if there is some problem - the problematic code change stops at the first test that recognizes it and does not go any further.
3. **Add a deployment step**
   * On your Continuous Integration server, set a build that runs your script to deploy your application to a testing or staging environment. Ensure that it runs only when all the tests pass.

**Top Deployment tools and their features**

### 1. [Bamboo](https://www.atlassian.com/software/bamboo)

Key Features:

* Language agnostic integration with many platforms
* Dedicated agents focus on specific build projects
* Full reporting of upcoming code changes and issues

### 2. [TeamCity](https://www.jetbrains.com/teamcity/" \t "_blank)

Key Features:

* Start within minutes from installation
* Support for third-party tools without plugins or modifications
* Comprehensive version control system
* Customization and extension of the server

### 3. [AWS CodeDeploy](https://aws.amazon.com/codedeploy/)

Key Features:

* Automated and repeatable deployments
* Centralized control
* Applications still accessible during deployments

### 4. [Octopus Deploy](https://octopus.com/)

Key Features:

* Dashboard showing deployment statuses
* Installation and set up completed quickly
* Automatic deployments for virtual machines on site or in cloud services

### 5. [ElectricFlow](http://electric-cloud.com/products/electricflow/" \t "_blank)

Key Features:

* Build and deploy microservices
* View release status and milestone dates
* Automate and track cloud-native, container, and legacy application deployments

### 6. [PDQ Deploy](https://www.pdq.com/pdq-deploy/)

Key Features:

* Customize installs
* Installs to multiple computers simultaneously
* Real-time status monitoring

### 7. [Ansible Tower](https://www.ansible.com/tower" \t "_blank)

Key Features:

* Visual dashboard includes statuses, job activity, and job snapshots
* Playbooks allow multiple workflows to be chained together
* API and command line interface tools
* Role-based access control
* Job scheduling
* Integrated notifications
* Graphical inventory management

### 8. [Chef](https://www.chef.io/chef/)

Key Features:

* Manage data centers and cloud environments as one infrastructure
* Migrate workloads to the cloud quickly
* Cloud agnostic — promises integration with any provider

### 9. [Codeship](https://codeship.com/" \t "_blank)

Key Features:

* Docker support
* Parallel deployments and testing
* Debugging via SSH access and command line interface

### 10. [AppVeyor](https://www.appveyor.com/" \t "_blank)

Key Features:

* No setup or maintenance required
* Developer-friendly build configs using UI or YAML
* Fast builds on dedicated hardware with SSD drives
* Multi-branch builds
* Staff and community support
* Pay-as-you-go pricing

**Best Deployment practices**

### [Argonaut](https://www.plutora.com/ci-cd-tools/deployment-tools/argonaut)

Deploy apps and infrastructure on your cloud in minutes. Support for custom and third-party app deployments on Kubernetes and Lambda environments.

### [Flosum](https://www.plutora.com/ci-cd-tools/deployment-tools/flosum)

Flosum is a native DevOps solution that handles the development lifecycle including merging components, version control, continuous deployments, static code analysis, user story management, and regression testing. Flosum works together with Salesforce to identify changes in a particular component.

### [Codefresh](https://www.plutora.com/ci-cd-tools/deployment-tools/codefresh)

Codefresh is an enterprise continuous delivery platform for cloud-native applications. It automates GitOps workflows and advanced deployments such as Canary and Blue/Green.

### [Launch Darkly](https://www.plutora.com/ci-cd-tools/deployment-tools/launch-darkly)

LaunchDarkly is a feature management application that enables teams to deploy code at any time, even if a feature isn’t ready to be released to all users.

### [Harness](https://www.plutora.com/ci-cd-tools/deployment-tools/harness)

Harness Continuous Delivery is a software delivery solution that automatically deploys, verifies, and rolls back artifacts. You can leverage automated canary and blue-green deployments for faster, safer, and more efficient rollouts.

### [Juju](https://www.plutora.com/ci-cd-tools/deployment-tools/juju)

Juju is an open source application and service modeling tool from Ubuntu that helps you deploy, manage and scale your applications on any cloud. With Juju, different authors are able to create service formulas, called charms, independently, and make those services coordinate their communication and configuration through a simple protocol. Juju can also be used together with Orchestra for physical deployments.

### [Rundeck](https://www.plutora.com/ci-cd-tools/deployment-tools/rundeck)

Rundeck is an open source operations management platform that helps to automate routine operational procedures in data center or cloud environments. It enables you to run tasks on any number of nodes from a web-based or command-line interface.

**Setup deployment pipeline**

In software development, a **deployment pipeline** is a system of automated processes designed to quickly and accurately move new code additions and updates from version control to production.

1. Step 1: Create a deployment environment. ...
2. Step 2: Get a copy of the sample code. ...
3. Step 3: Create your pipeline. ...
4. Step 4: Activate your pipeline to deploy your code. ...
5. Step 5: Commit a change and then update your app. ...
6. Step 6: Clean up your resources.

Refer : https://aws.amazon.com/getting-started/hands-on/continuous-deployment-pipeline/

**Continuous deployment**

Continuous deployment is a strategy for software releases wherein any code commit that passes the automated testing phase is automatically released into the production environment, making changes that are visible to the software's users.

Continuous deployment eliminates the human safeguards against unproven code in live software. It should only be implemented when the development and IT teams rigorously adhere to production-ready development practices and thorough testing, and when they apply sophisticated, real-time monitoring in production to discover any issues with new releases.

**Static code analysis**

Static analysis is a method of debugging that is done by automatically examining the source code without having to execute the program. This provides developers with an understanding of their code base and helps ensure that it is compliant, safe, and secure.

## When Is Static Code Analysis Performed?

Static code analysis is performed early in development, before [software testing](https://www.perforce.com/resources/alm/types-of-software-testing)  begins. For organizations practicing [DevOps](https://www.perforce.com/solutions/devops" \t "_blank), static code analysis takes place during the “Create” phase.

**Practice – code analysis using tools**

**Containers**

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

**Why containers?**

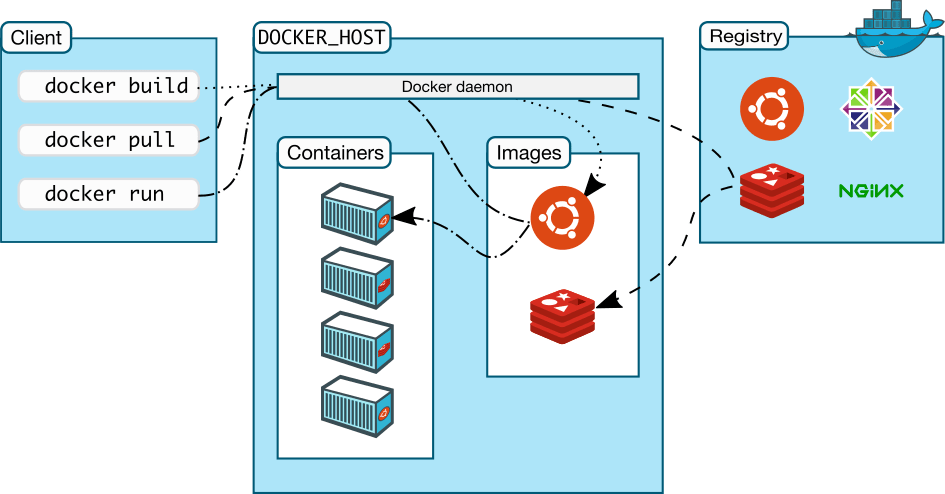
Containers require less system resources than traditional or hardware virtual machine environments because they don't include operating system images. Applications running in containers can be deployed easily to multiple different operating systems and hardware platforms.

**What is a docker?**

Docker is an open source platform for building, deploying, and managing containerized applications.

**How docker works?**

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.



**Components of docker**

- Docker container

Docker containers are the live, running instances of Docker images. While Docker images are read-only files, containers are life, ephemeral, executable content. Users can interact with them, and administrators can adjust their settings and conditions using Docker commands.

- Docker client

[Docker client](https://docs.docker.com/desktop/) is an application for Mac or Windows that includes Docker Engine, Docker CLI client, Docker Compose, Kubernetes, and others. It also includes access to Docker Hub.

- Docker daemon

Docker daemon is a service that creates and manages Docker images, using the commands from the client. Essentially Docker daemon serves as the control center of your Docker implementation**.**The server on which Docker daemon runs is called the Docker host.

-Docker image

Docker images contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.

- Docker registry

A Docker registry is a scalable open-source storage and distribution system for Docker images. The registry enables you to track image versions in repositories, using tagging for identification. This is accomplished using git, a version control tool.

**Install docker on desktop and start the docker tool.**

a – Download the Docker Toolbox executable

b – Run the Docker Toolbox installer for Windows.

Docker Toolbox for Windows installer step one


Click on **Next**.

For the second screen, choose a location where to store your Docker files, and click on Next.

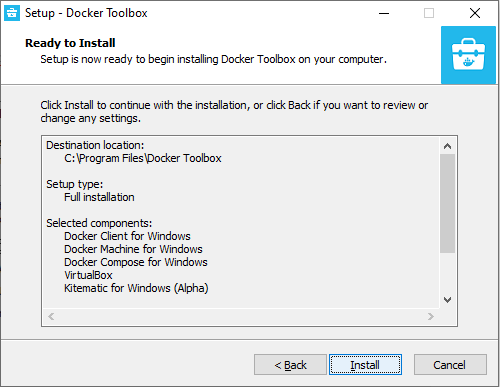
Docker Toolbox installer for Windows step two


Click on **Next**.

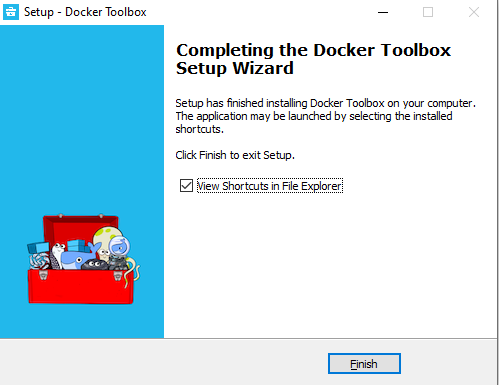
For the additional tasks, you can let the default options and click on Next.

Docker Toolbox for Windows installer step four


Finally, click on Install to start the Docker Toolbox installation.



After a short while, your installation should complete. Leave the checkbox checked in order to see the shortcuts created by Docker.



To verify your current Docker version, run the following command

$ docker version

**Docker file**

Every Docker container starts with a simple text file containing instructions for how to build the Docker container image. DockerFile automates the process of Docker image creation. It’s essentially a list of command-line interface (CLI) instructions that Docker Engine will run in order to assemble the image. The list of Docker commands is huge, but standardized: Docker operations work the same regardless of contents, infrastructure, or other environment variables.

**Docker image**

Docker images contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.

**Commands to create docker file.**

|  |  |
| --- | --- |
|  |  |
|  |  |
| **ADD** – Defines files to copy from the Host file system onto the Container   * + ADD ./local/config.file /etc/service/config.file * **CMD** – This is the command that will run when the Container starts   + CMD [“nginx”, “-g”, “daemon off;”] * **ENTRYPOINT** – Sets the default application used every time a Container is created from the Image. If used in conjunction with CMD, you can remove the application and just define the arguments there   + CMD Hello World!   + ENTRYPOINT echo * **ENV** – Set/modify the environment variables within Containers created from the Image.   + ENV VERSION 1.0 * **EXPOSE** – Define which Container ports to expose   + EXPOSE 80 * **FROM** – Select the base image to build the new image on top of   + FROM ubuntu:latest * **LABEL maintainer** – Optional field to let you identify yourself as the maintainer of this image. This is just a label (it used to be a dedicated Docker directive).   + LABEL maintainer=someone@xyz.xyz” * **RUN** – Specify commands to make changes to your Image and subsequently the Containers started from this Image. This includes updating packages, installing software, adding users, creating an initial database, setting up certificates, etc. These are the commands you would run at the command line to install and configure your application. This is one of the most important dockerfile directives.   + RUN apt-get update && apt-get upgrade -y && apt-get install -y nginx && rm -rf /var/lib/apt/lists/\* * **USER** – Define the default User all commands will be run as within any Container created from your Image. It can be either a UID or username   + USER docker * **VOLUME** – Creates a mount point within the Container linking it back to file systems accessible by the Docker Host. New Volumes get populated with the pre-existing contents of the specified location in the image. It is specially relevant to mention is that defining Volumes in a Dockerfile can lead to issues. Volumes should be managed with docker-compose or “docker run” commands. Volumes are optional. If your application does not have any state (and most web applications work like this) then you don’t need to use volumes.   + VOLUME /var/log * **WORKDIR** – Define the default working directory for the command defined in the “ENTRYPOINT” or “CMD” instructions   + WORKDIR /home |  |
|  |  |

**Build docker image with docker file**

1. Write a Dockerfile for your application.
2. Build the image with docker build command.
3. Host your Docker image on a registry.
4. Pull and run the image on the target machine.

**Refer : https://codefresh.io/blog/build-docker-image-dockerfiles/**

**create docker container from docker image**

## Step 1: Create a Base Container

The Docker create command will create a new container for us from the command line:

~ docker create --name nginx\_base -p 80:80 nginx:alpine

## Step 2: Inspect Images

## Step 3: Inspect Containers

➜ ~ docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

c365af6303e4 nginx:alpine "/docker-entrypoint.…" 6 minutes ago Created nginx

## Step 4: Start the Container

➜ ~ docker start nginx\_base

nginx\_base



## Step 5: Modify the Running Container

Let’s create a new index.html file and copy it onto the running container. Using an editor on your machine, create an index.html file in the same directory that you have been running Docker commands from.

➜ ~ docker cp index.html nginx\_base:/usr/share/nginx/html/index.html

## Step 6: Create an Image From a Container

➜ ~ docker commit nginx\_base

sha256:0c17f0798823c7febc5a67d5432b48f525320d671beb2e6f04303f3da2f10432

## Step 7: Tag the Image

➜ ~ docker tag 0c17f0798823 hello\_world\_nginx

## Step 8: Create Images With Tags

You can also tag the image as it is created by adding another argument to the end of the command like this:

➜ ~ docker commit nginx\_base hello\_world\_nginx

## Step 9: Delete the Original Container

➜ ~ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

c365af6303e4 nginx:alpine "/docker-entrypoint.…" 33 minutes ago Up 25 minutes 0.0.0.

## Step 10: Look at Running Containers

➜ ~ docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

7ca08e03862d hello\_world\_nginx "/docker-entrypoint.…" 2 minutes ago Up 2 minutes 0.0.0.0:80->

## Step 11: Consider Your Options

Option A: Set Authorship

Option B: Create Commit Messages

Option C: Commit Without Pause

Option D: Change Configuration

**Refer : https://www.dataset.com/blog/create-docker-image/**

**Run the docker container**

The basic syntax for the command is:

docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

To run a container, the only thing you need to include in the command is the image on which it is based:

docker run [docker\_image]

As an example, we used a [Dockerfile to create a sample Docker image](https://phoenixnap.com/kb/create-docker-images-with-dockerfile" \t "_blank) with the task to echo the message **Hello World**. For us, the image has the ID **e98b6ec72f51**. Your image name will differ depending on the container you want to run.

The command to run our sample container would be:

docker run e98b6ec72f51

**Refer : https://phoenixnap.com/kb/docker-run-command-with-examples**