

# Table of Contents

Solutions.....	1
1    Editing the hello.go program .....	1
2    Execution of updated hello.go program.....	1
3    Solutions to the theoretical questions .....	2
3.1    What do “pets” and “cattle” refer to?.....	2
3.2    Why do we need to use sudo when executing the script setup_vm.sh? .....	2
3.3    When we pull docker images, where on the internet do the images come from?..	3
3.4    What is the difference between a docker image and a container? .....	4

## List of Figures

Figure 1: Edited code in the hello.go file to display current local time.	1
Figure 2: Updated Program displaying the necessary details.	1

## Solutions

### 1 Editing the hello.go program

```
package main

import (
    "fmt"
    "time"
)

func main() {
    fmt.Println("Hello world!")

    now := time.Now()
    gmtMinus5 := now.Add(-5 * time.Hour)

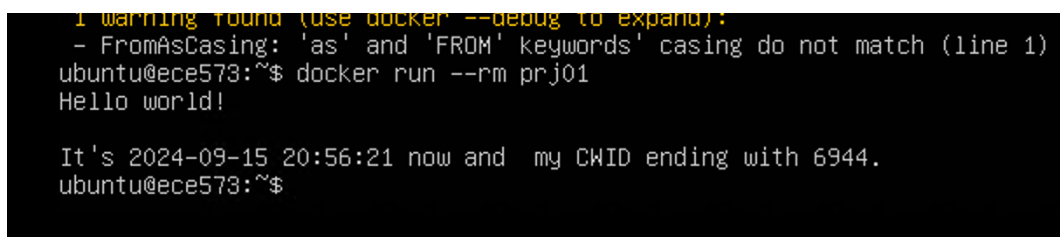
    fmt.Printf("It's %s now and my CWID ending in 6944.\n",
        gmtMinus5.Format("2006-01-02 15:04:05"))
}
```

Figure 1: Edited code in the hello.go file to display current local time.

The above Figure 1 represents the edited hello.go file with the code to display the current time of Chicago (GMT - 5) along with the last four digits from my CWID.

It is to note that since I use an instance from a stand-alone server based in the United Kingdom, the local time for Chicago needs to be calibrated as per the code in Figure 1. That is why a variable named “gmtMinus5” has been introduced to facilitate the same.

### 2 Execution of updated hello.go program



```
1 warning found (use docker --debug to expand):
- FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 1)
ubuntu@ece573:~$ docker run --rm prj01
Hello world!

It's 2024-09-15 20:56:21 now and my CWID ending with 6944.
ubuntu@ece573:~$
```

Figure 2: Updated Program displaying the necessary details.

Figure 2 showcases the output from the changes made to the hello.go file when the docker file is run. It is to be observed that the “Hello World!” text is retained from the previously created document and the new addition of the current time in Chicago along with the last 4 digits of my CWID is made to display.

## 3 Solutions to the theoretical questions

### 3.1 What do “pets” and “cattle” refer to?

Pets and Cattle refer to the traditional server systems to the modern-day cloud systems. Pets are individual, nurtured, specially cared-for and cherished individuals and thus the standalone traditional server management system. However, on the other hand, Cattle represent the disposable and easily replaceable units compared with the servers in the large-scale cloud environment.

Pets receive a unique identity, they are irreplicable, they are manually cared for and each one is different. This exactly represents a stand-alone individual server maintained by a household or a small organization.

Whereas Cattle meant for farming are numbered not named, they are easily replaced if something were to go wrong, the process of handling them is highly systematic and automated, and there's consistency maintained among the lot. There can be a straight line that can be drawn to connect the similarity with this and cloud-based servers that are industrially maintained.

### 3.2 Why do we need to use sudo when executing the script setup\_vm.sh?

Sudo stands for Super user, and it is generally used for the following:

1. Installation of Software Packages.
2. System Configuration.
3. Creating/Modifying directories.
4. Consistency in Execution.
5. Security.

Sudo helps for the above in requesting for access as a primary user. Generally, after using sudo command, there is a prompt required for typing in the password that grants access to all the above critical requirements.

The usage of sudo during the execution of setup\_vm.sh is for the factors such as: Installing Docker and Go that requires system-wide access, during installation, there may be requirement in altering configurations and settings, creating and modifying directories is also another possibility that is required during setup\_vm.sh, to avoid uneven installations and multiple prompts to the user each time requesting permission to perform a single task, sudo will be user.

Once password is entered and sudo access is granted, this is equivalent to administrator access into the system.

### **3.3 When we pull docker images, where on the internet do the images come from?**

Usually, we get our Docker images from a variety of online sources, the most well-known of which being Docker Hub. The public registry that Docker, Inc. maintains by default for Docker images is called Docker Hub.

It houses a sizable assortment of public domain photos, user-contributed photos, and private archives. Docker scans Docker Hub automatically when you execute a docker pull command without providing a registry.

But there are other places to find Docker images outside Docker Hub. With the increasing use of containerization in software development and deployment, a multitude of substitutes have surfaced to meet varying requirements:

1. **Cloud Provider Registries:** As part of their services, major cloud providers have their own container registries. Google Container Registry (GCR), Azure Container Registry (ACR), and Amazon Elastic Container Registry (ECR) are a few of them. The deployment pipelines inside these registries' ecosystems are smooth since they are frequently connected with other cloud services.
2. **Private Registries:** To house confidential or proprietary photos, a lot of firms create their own private registries. These could be cloud-based services or self-hosted programs. Greater control over compliance, security, and access is provided by private registries.
3. **Third-Party Registry Services:** Specialized container registry services are offered by organizations such as Quay.io, GitLab Container Registry, and JFrog Artifactory. These frequently have extra functionality like interaction with CI/CD pipelines, access control, and vulnerability scanning.
4. **Open-Source Registries:** Initiatives such as Harbor offer self-managing, self-deployable open-source registry solutions to enterprises.
5. **GitHub Container Registry:** Developers can store and manage their Docker images in addition to their code repositories using GitHub's proprietary container registry service.

Developers and businesses may select the best registry for their requirements by considering aspects like security, performance, and interaction with current processes thanks to the flexibility of image sources.

It's important to remember that registry selection can have a big impact on performance and security. Public registries such as Docker Hub are useful, but if not thoroughly screened, they might provide security problems. Although they need more administrative work, private or company registers give users greater control.

Furthermore, complex image management techniques have been developed because of the capacity to extract photos from several sources. Before deploying images, organizations frequently examine them for vulnerabilities, set up policies to restrict which registries may be used, and put in place caching methods to speed up pull requests and save bandwidth.

In conclusion, even while Docker Hub is still the most popular place to find Docker images, the ecosystem has grown to include a variety of solutions that meet various requirements and use cases. The growing popularity of Docker and containerization in contemporary software development and deployment processes may be attributed in large part to its flexibility.

### **3.4 What is the difference between a docker image and a container?**

In containerization technology, a Docker image and a Docker container are two different but closely related concepts:

An executable, independent, and lightweight package that contains all the components required to run a piece of software is called a Docker image. The runtime, libraries, settings, system tools, and application code are all contained in it. Images are simply a snapshot or template that are constructed using instructions included in a Docker file. They are unchangeable, which means that once an image is made, it never changes.

Conversely, a Docker container is an instance of a Docker image that is currently in use. The program operates in this context during real execution. Containers can be started, halted, transferred, or removed. They are produced from pictures. In contrast to pictures, containers can have states and can be modified. To enable it to alter the filesystem's contents, a running container adds a writable layer on top of the immutable image layers.

Their goals and states are where the main differences are found. Containers are the dynamic, operating instances of images, while images are the static templates used to build them. In object-oriented programming, an image is like a class, and an object created from that class is an object. This is a common analogy. One image can be used to construct several containers, each of which can operate independently and store changes to its filesystem and state.