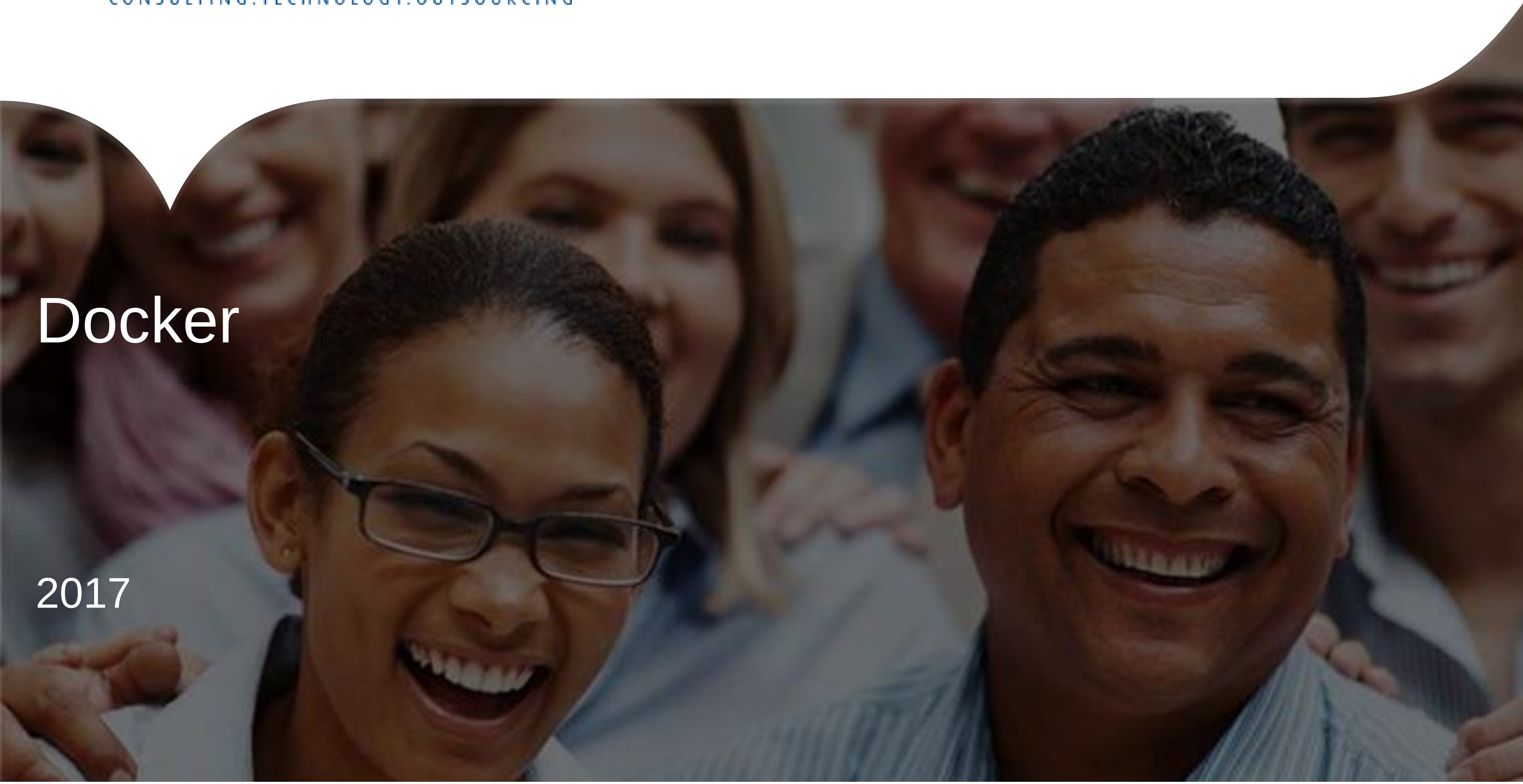
Capgemini

CONSULTING.TECHNOLOGY.OUTSOURCING

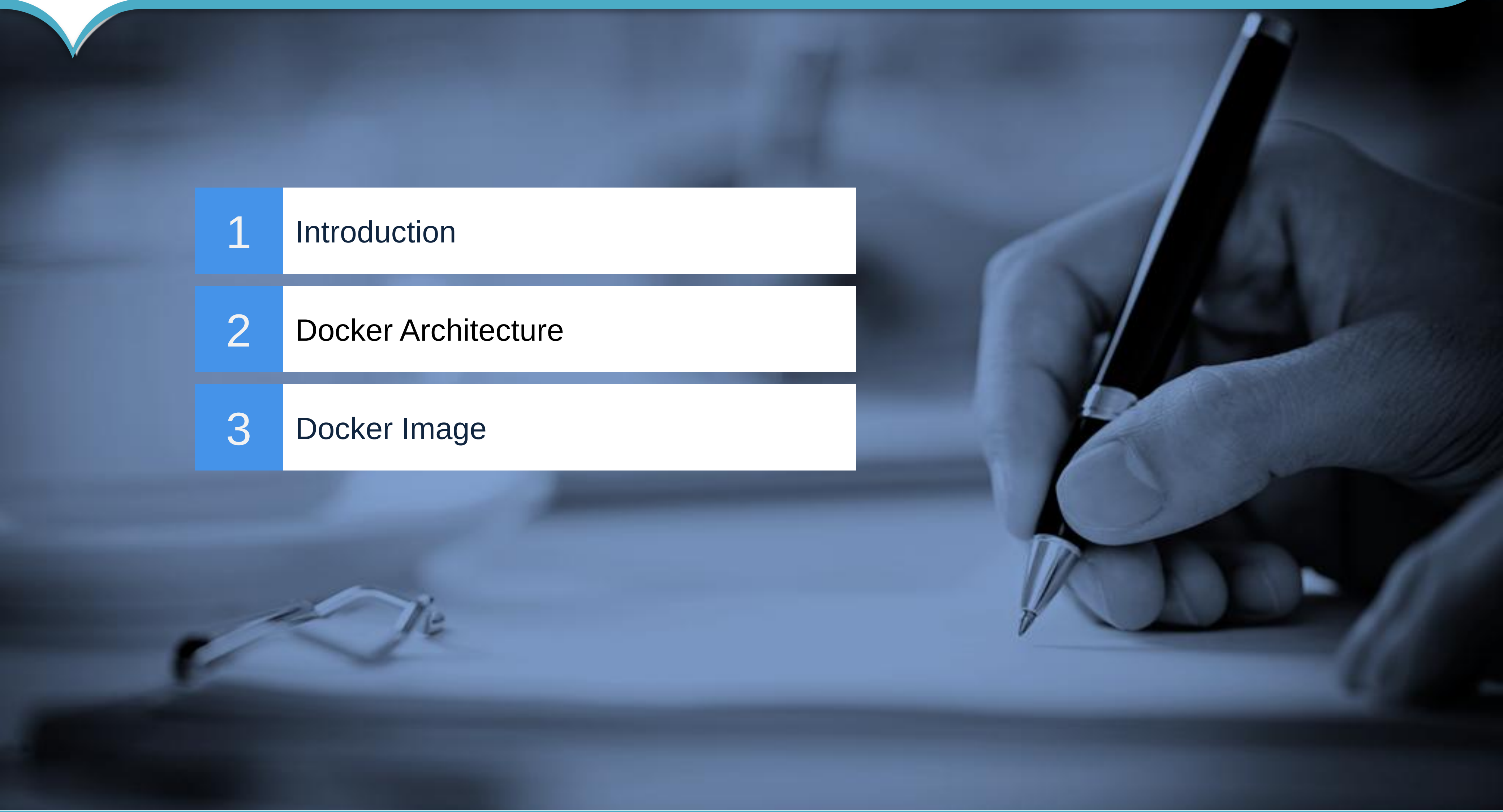


Docker

2017

People matter, results count.

Agenda



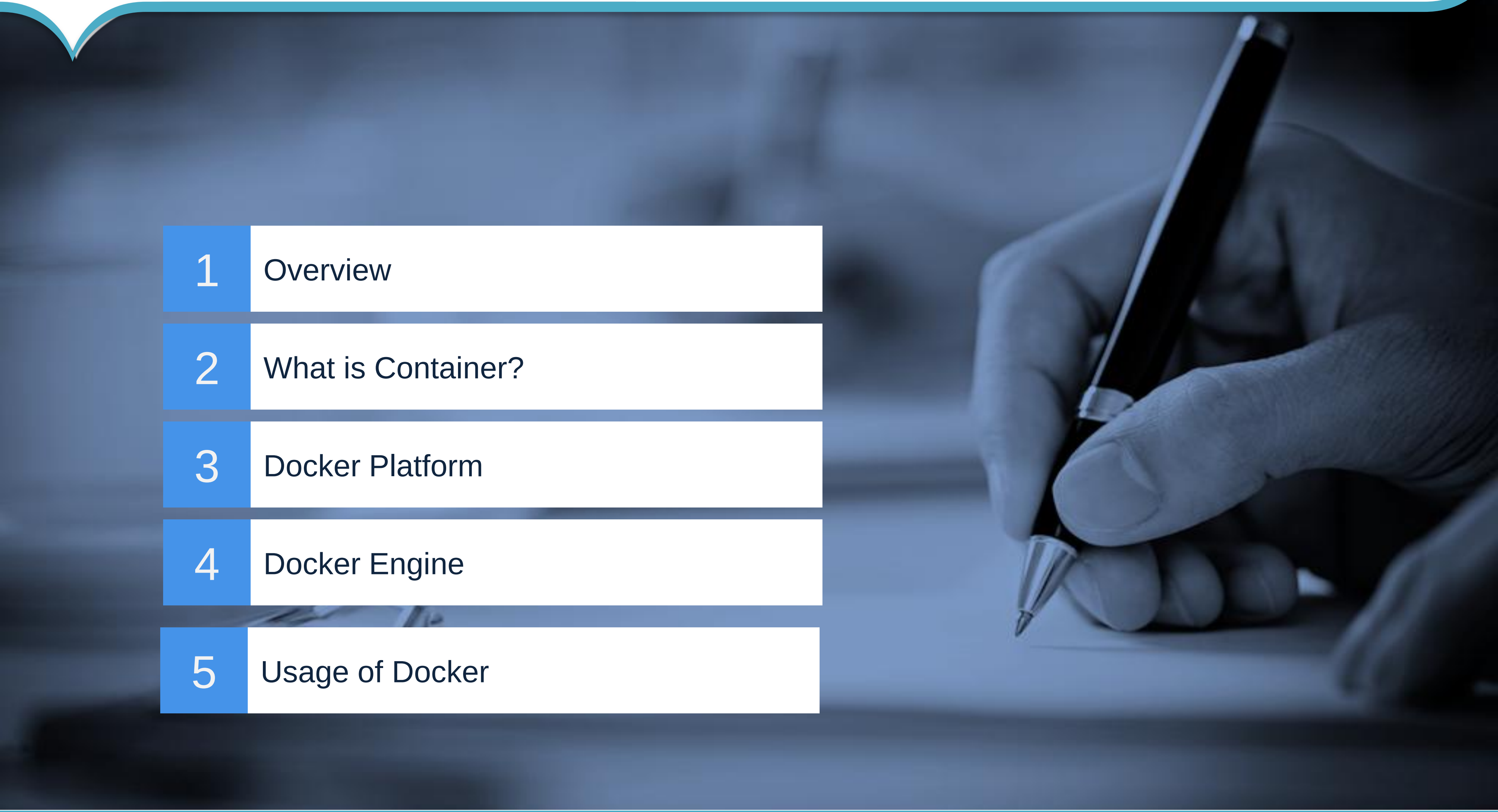
1 Introduction

2 Docker Architecture

3 Docker Image

Docker – Introduction

Outline



1 Overview

2 What is Container?

3 Docker Platform

4 Docker Engine

5 Usage of Docker

Overview

- **Docker**
  - World’s leading software container platform.
  - Developers use Docker to eliminate “works on my machine” problems when collaborating on code with co-workers.
  - Operators use Docker to run and manage apps side-by-side in isolated containers to get better compute density.
  - Enterprises use Docker to build agile software delivery pipelines to ship new features faster, more securely and with confidence for both Linux and Windows Server apps.
  - Separates our applications from our infrastructure so we can deliver software quickly.
  - Methodologies for shipping, testing, and deploying code quickly.
  - Significantly reduce the delay between writing code and running it in production.

What is Container?

- To make a piece of software run is packaged into isolated containers.
- Unlike VMs, containers do not bundle a full operating system - only libraries and settings required to make the software work are needed.
- Making efficient, lightweight, self-contained systems that guarantees software will always run the same, regardless of where it’s deployed.

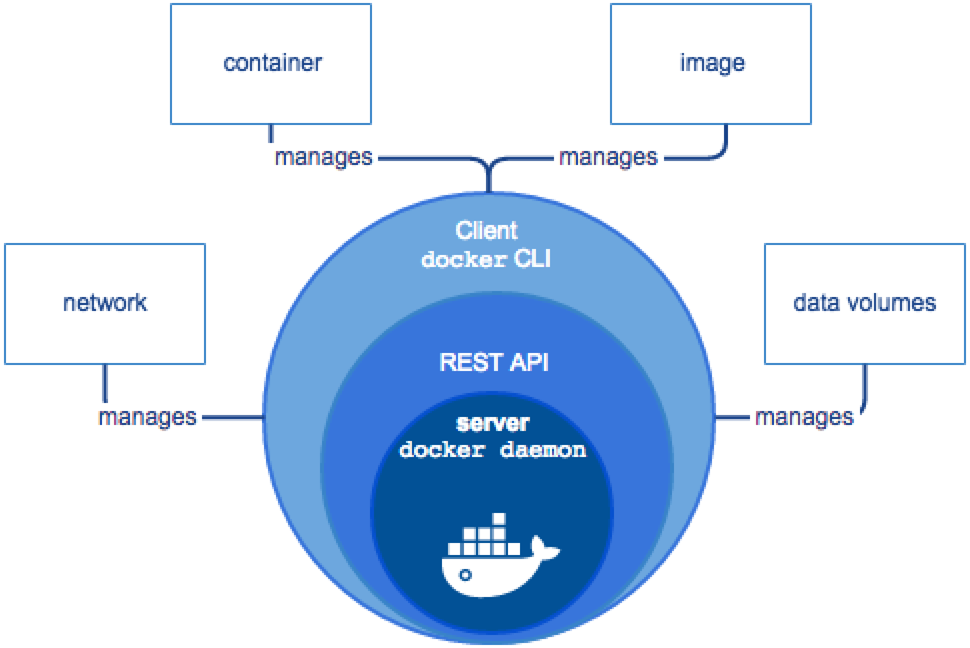
Docker Platform

- Docker provides the ability to package and run an application in a loosely isolated environment called a container.
- Allows us to run many containers simultaneously on a given host.
- Containers are light weight in nature.
- Without the extra load of a hypervisor, we can run more containers on a given hardware combination than if we were using virtual machines.

- Docker provides tooling and a platform to manage the lifecycle of your containers:
  - Encapsulate your applications (and supporting components) into Docker containers
  - Distribute and ship those containers to your teams for further development and testing
  - Deploy those applications to your production environment, whether it is in a local data center or the Cloud

Docker Engine

- *Docker Engine* is a client-server application with these major components:
  - A server which is a type of long-running program called a daemon process.
  - A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
  - A command line interface (CLI) client.



Usage of Docker

- *Fast, consistent delivery of your applications*
  - Streamline the development lifecycle
  - Integrate Docker into our continuous integration and continuous deployment (CI/CD) workflow.
- *Responsive deployment and scaling*
  - Container-based platform allows for highly portable workloads
  - Docker containers can run on a developer’s local host, on physical or virtual machines in a data center, in the Cloud, or in a mixture of environments.
  - Docker’s portability and lightweight nature also make it easy to dynamically manage workloads, scaling up or tearing down applications and services as business needs dictate, in near real time.
- *Running more workloads on the same hardware*
  - Lightweight and fast.
  - Viable, cost-effective alternative to hypervisor-based virtual machines
  - Useful in high density environments and for small and medium deployments where you need to do more with fewer resources.

Summary

Introduction

Container

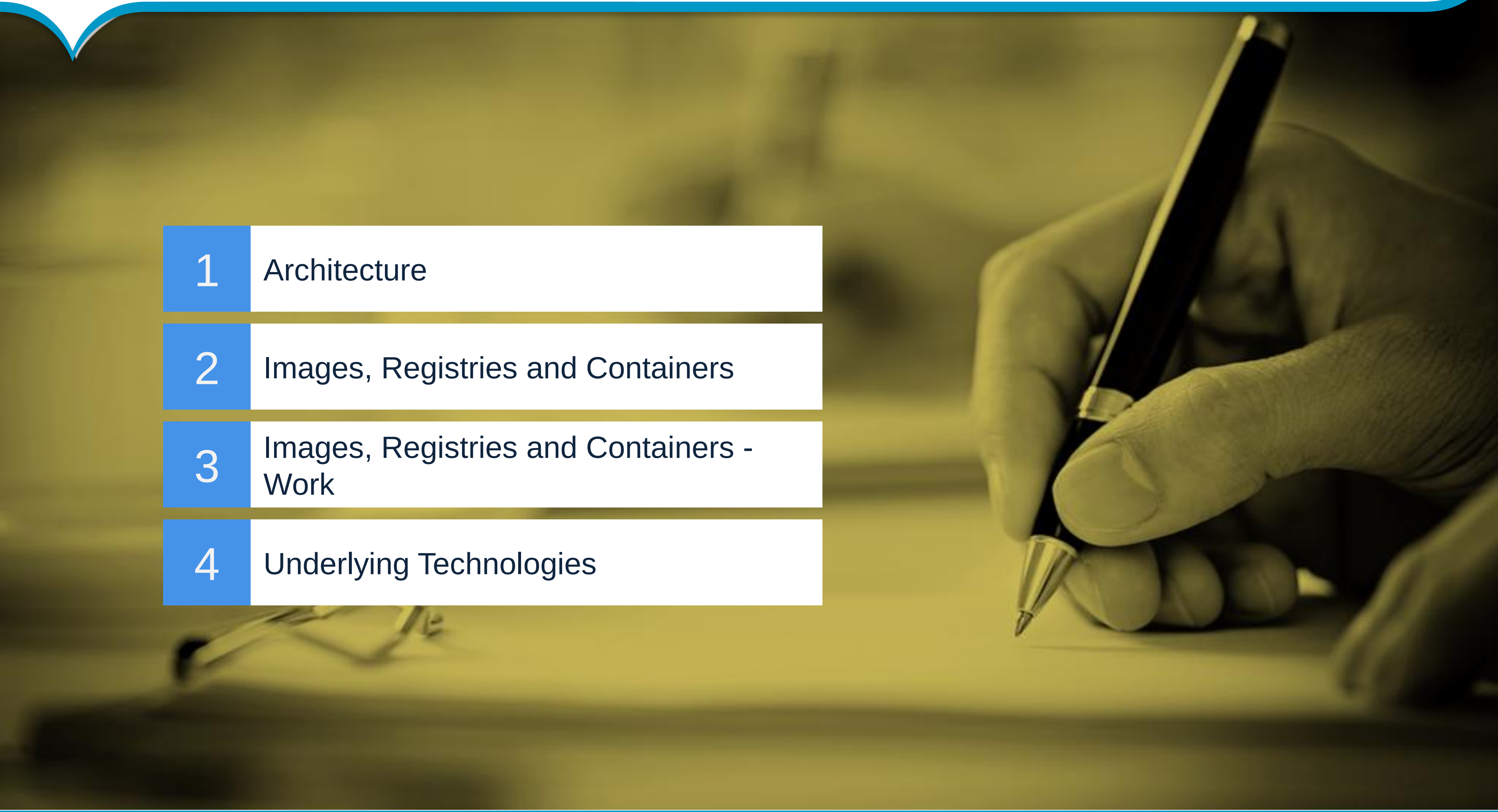
Docker Platform

Docker Engine

Usage

Docker – Architecture

Outline



1 Architecture

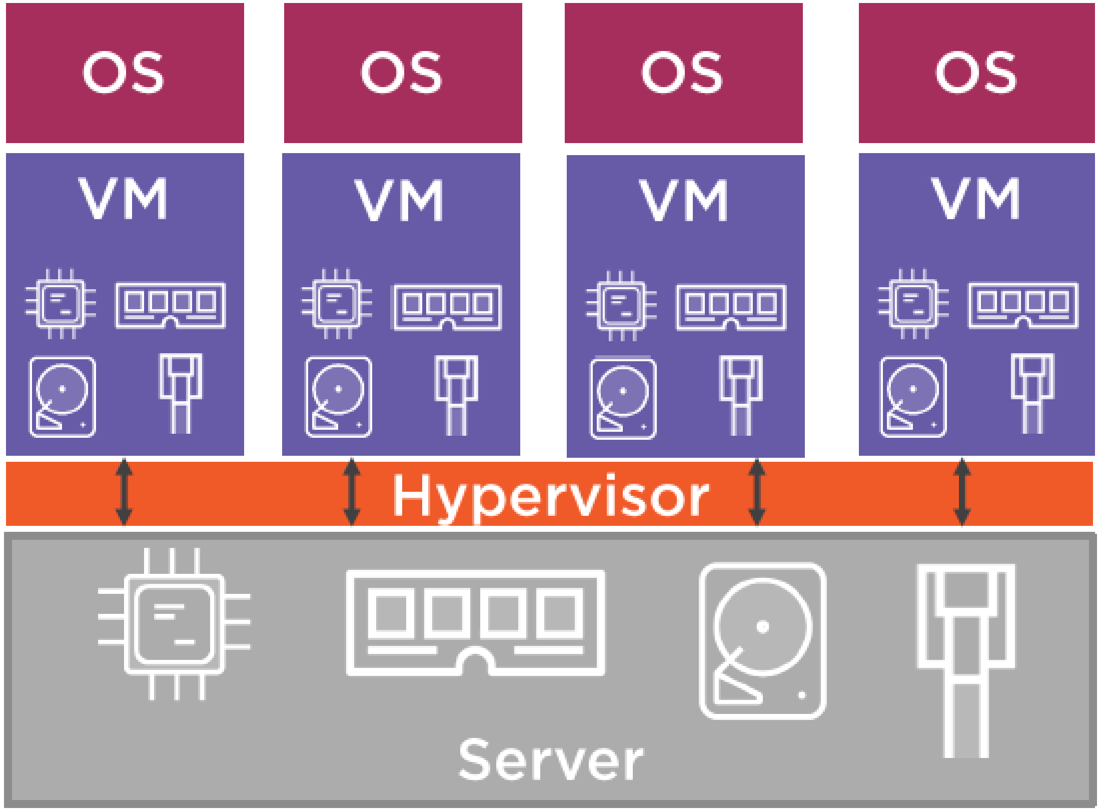
2 Images, Registries and Containers

3 Images, Registries and Containers - Work

4 Underlying Technologies

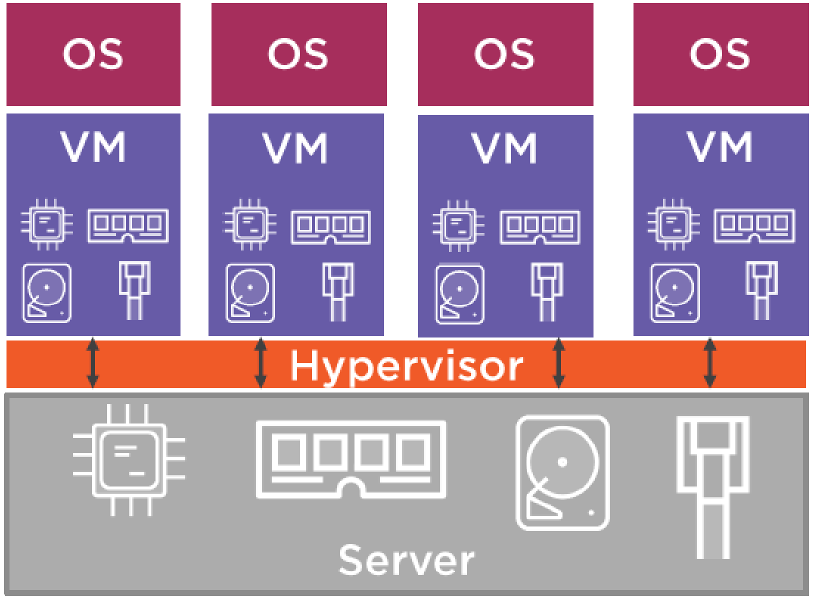
Docker Architecture

VM Model

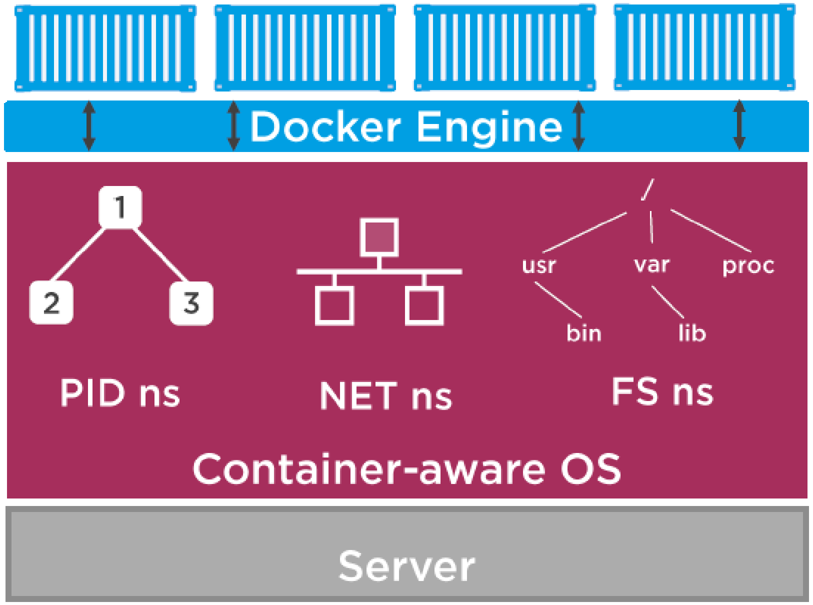


Docker Architecture

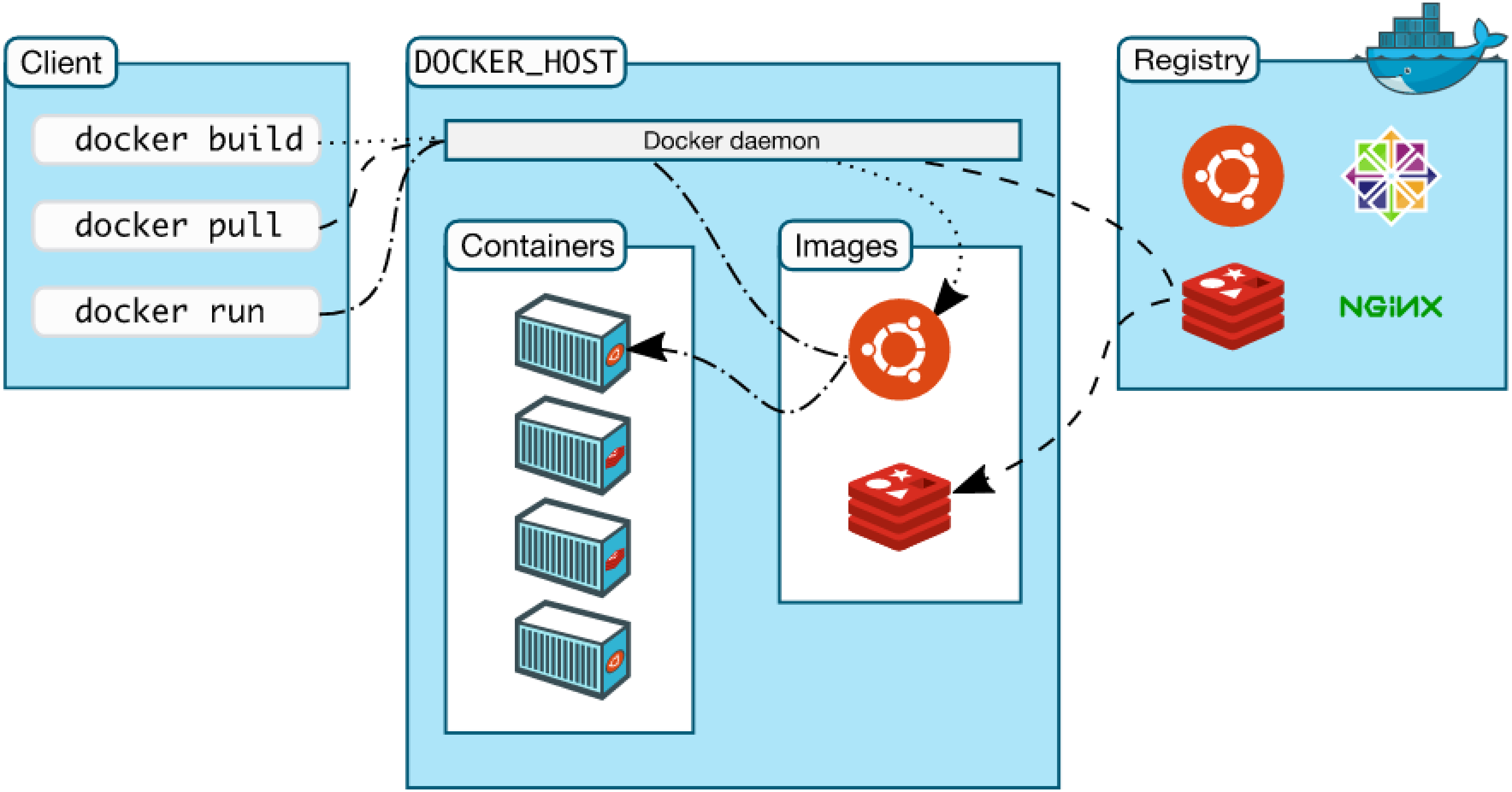
VM Model



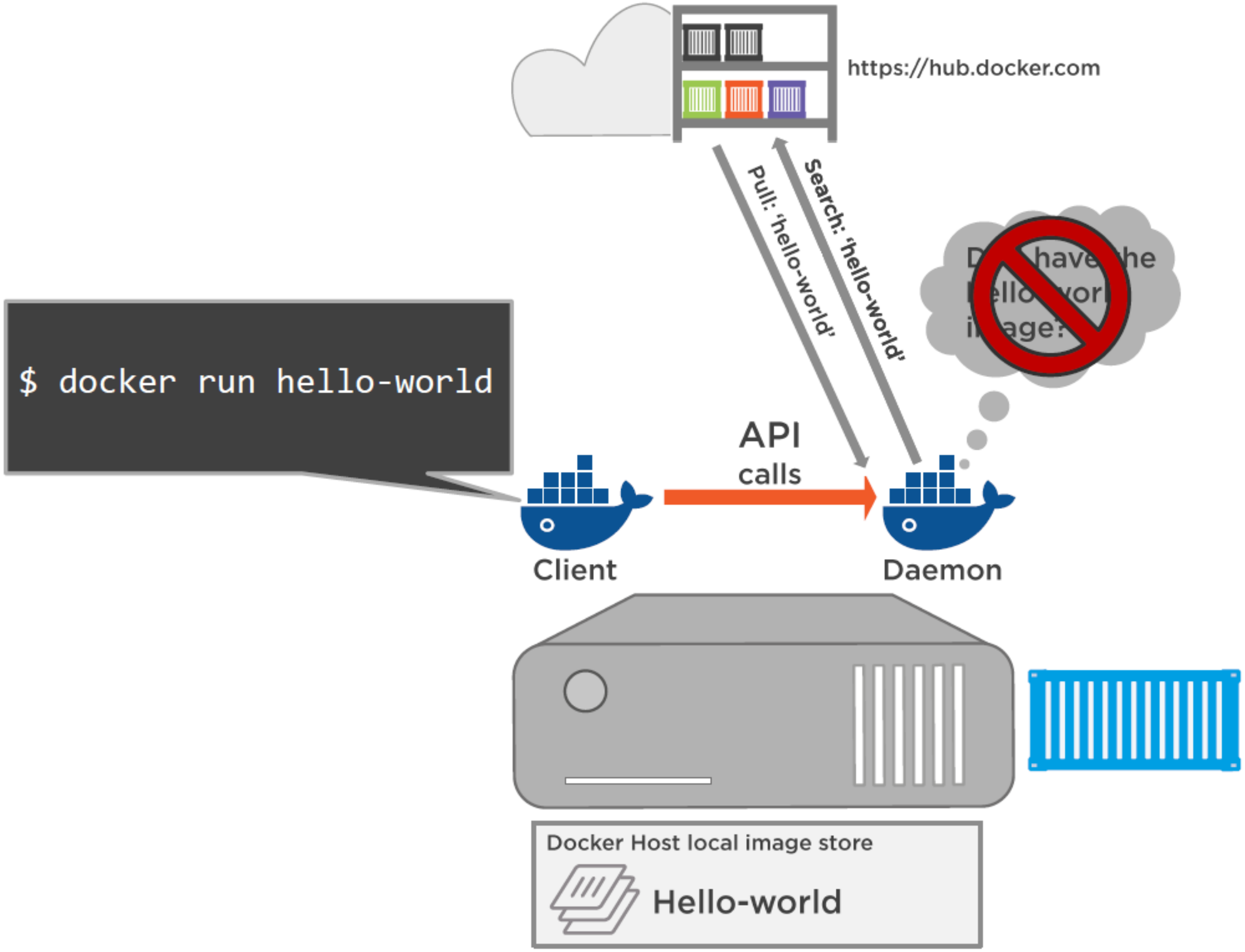
Container Model



Docker Architecture

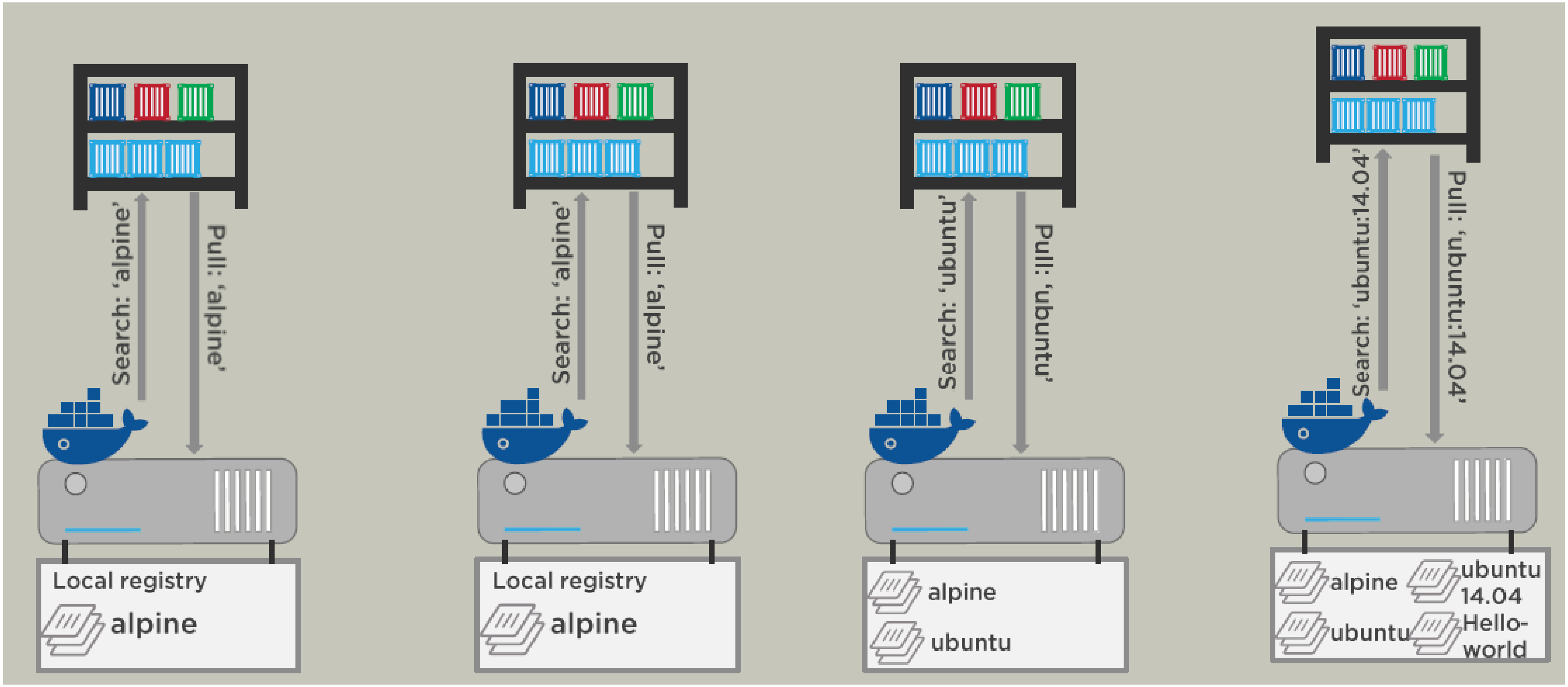


Docker Architecture



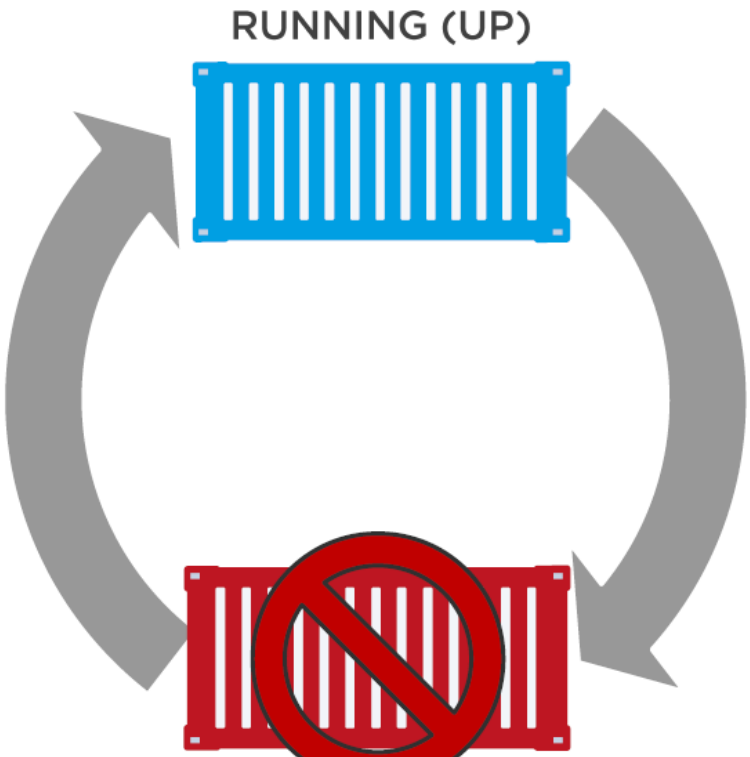
```
$ docker run hello-world
```

Docker Architecture



Docker Architecture

Container Life Cycle



RUNNING (UP)

docker start <container>

docker stop <container>

STOPPED
(EXITED)

docker rm <container>

Docker Architecture

- Client - Server Architecture.
- Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers.
- Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon.
- Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.

- **The Docker daemon**
  - The Docker daemon runs on a host machine. The user uses the Docker client to interact with the daemon.
- **The Docker client**
  - The Docker client, in the form of the docker binary, is the primary user interface to Docker. It accepts commands and configuration flags from the user and communicates with a Docker daemon. One client can even communicate with multiple unrelated daemons.

Images, Registries and Containers

- *Images*
- *Registries*
- *Containers*

- **DOCKER IMAGES**
  - Read-only template with instructions for creating a Docker container.
    - For example, an image might contain an Ubuntu operating system with Apache web server and your web application installed.
  - can build or update images from scratch
  - download and use images created by others.
  - Image may be based on, or may extend, one or more other images

- A docker image is described in text file called a *Dockerfile*, which has a simple, well-defined syntax.

Images, Registries and Containers

- **DOCKER CONTAINERS**
  - Docker container is a runnable instance of a Docker image.
  - Can run, start, stop, move, or delete a container using Docker API or CLI commands.
  - When you run a container, you can provide configuration metadata such as networking information or environment variables.
  - Each container is an isolated and secure application platform
  - But can be given access to resources running in a different host or container, as well as persistent storage or databases.
- **DOCKER REGISTRIES**
  - A docker registry is a library of images.
  - A registry can be public or private.
  - Can be on the same server as the Docker daemon or Docker client, or on a totally separate server.
  - Docker registries are the **distribution** component of Docker.

Images, Registries and Containers - works

- **How does a Docker image work?**
  - Each image consists of a series of layers.
  - Docker uses union file systems to combine these layers into a single image.
  - Union file systems allow files and directories of separate file systems, known as branches, to be transparently overlaid, forming a single coherent file system.
  - These layers are one of the reasons Docker is so lightweight.
  - When you change a Docker image, such as when you update an application to a new version, a new layer is built and replaces only the layer it updates. The other layers remain intact.
  - To distribute the update, we only need to transfer the updated layer.
  - Layering speeds up distribution of Docker images.
  - Docker determines which layers need to be updated at runtime.

Images, Registries and Containers - works

- **Dockerfile:**
  - An image is defined in a Dockerfile.
  - Every image starts from a base image
    - **ubuntu, fedora, Apache**
  - The base image is defined using the FROM keyword in the dockerfile.
  - Some examples of Dockerfile instructions are:
    - Specify the base image (FROM)
    - Specify image metadata (LABEL)
    - Run a command (RUN)
    - Add a file or directory (ADD)
    - Create an environment variable (ENV)
    - What process to run when launching a container from this image (CMD)

Images, Registries and Containers - works

- **How does a Docker registry work?**
  - A Docker registry stores Docker images.
  - *Push* it to a public registry such as **Docker Hub** or to a private registry running behind your firewall.
  - Search for existing images and pull them from the registry to a host.

- **Docker Hub** is a public Docker registry which serves a huge collection of existing images and allows you to contribute your own.
- **Docker store** allows you to buy and sell Docker images
  - We can buy a Docker image containing an application or service from the software vendor
  - Use the image to deploy the application into our testing, staging, and production environments
  - Upgrade the application by pulling the new version of the image and redeploying the containers
  - Docker Store is currently in private beta.

Images, Registries and Containers - works

- **How does a container work?**
  - A container uses the host machine’s Linux kernel, and consists of any extra files you add when the image is created, along with metadata associated with the container at creation or when the container is started.
  - Each container is built from an image.
  - The image defines the container’s contents, which process to run when the container is launched, and a variety of other configuration details.
  - Docker image is read-only.
  - When Docker runs a container from an image, it adds a read-write layer on top of the image (using a UnionFS as we saw earlier) in which your application runs.

Images, Registries and Containers - works

- **WHAT HAPPENS WHEN YOU RUN A CONTAINER?**
  - **EXAMPLE**

```
$ docker run -i -t ubuntu /bin/bash
```

- **Pulls the ubuntu image:** Docker Engine checks for the presence of the ubuntu image. If the image already exists locally, Docker Engine uses it for the new container. Otherwise, then Docker Engine pulls it from **Docker Hub.**
- **Creates a new container:** Docker uses the image to create a container.
- **Allocates a filesystem and mounts a read-write *layer*:** The container is created in the file system and a read-write layer is added to the image.
- **Allocates a network / bridge interface:** Creates a network interface that allows the Docker container to talk to the local host.
- **Sets up an IP address:** Finds and attaches an available IP address from a pool.
- **Executes a process that you specify:** Executes the /bin/bash executable.
- **Captures and provides application output:** Connects and logs standard input, outputs and errors for you to see how your application is running, because you requested interactive mode.

Underlying Technologies

- Docker is written in **Go** and takes advantage of several features of the **Linux kernel** to deliver its functionality.
- **Namespaces**
  - 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.
    - **The pid namespace:** Process isolation (PID: Process ID).
    - **The net namespace:** Managing network interfaces (NET: Networking).
    - **The ipc namespace:** Managing access to IPC resources (IPC: InterProcess Communication).
    - **The mnt namespace:** Managing file system mount points (MNT: Mount).
    - **The uts namespace:** Isolating kernel and version identifiers. (UTS: Unix Timesharing System).

Underlying Technologies

- **Control groups**
  - Docker Engine on Linux also relies on another technology called *control groups* (cgroups). A cgroup limits an application to a specific set of resources. Control groups allow Docker Engine to share available hardware resources to containers and optionally enforce limits and constraints. For example, you can limit the memory available to a specific container.
- **Union file systems**
  - Union file systems, or UnionFS, are file systems that operate by creating layers, making them very lightweight and fast. Docker Engine uses UnionFS to provide the building blocks for containers. Docker Engine can use multiple UnionFS variants, including *AUFS, btrfs, vfs, and DeviceMapper.*
- **Container format**
  - Docker Engine combines the namespaces, control groups, and UnionFS into a wrapper called a container format. The default container format is **libcontainer**. In the future, Docker may support other container formats by integrating with technologies such as ***BSD Jails or Solaris Zones.***

Underlying Technologies

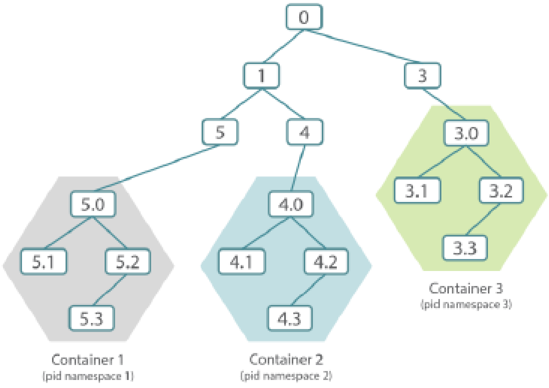
Kernel namespaces

The pid Namespace

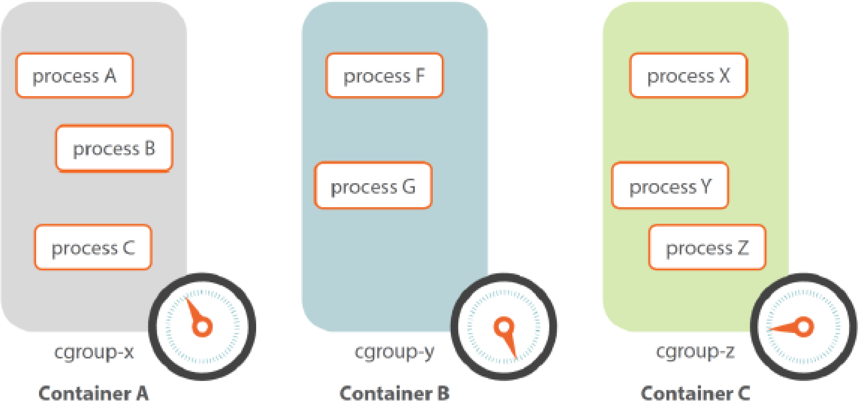
The net Namespace

The mnt Namespace

The user Namespace



cgroups



Capabilities

CAP_AUDIT_CONTROL ✓

CAP_CHOWN ✓

CAP_DAC_OVERRIDE ✓

CAP_KILL ✓

CAP_NET_BIND_SERVICE ✓

CAP_SETUID ✓

Summary

Architecture

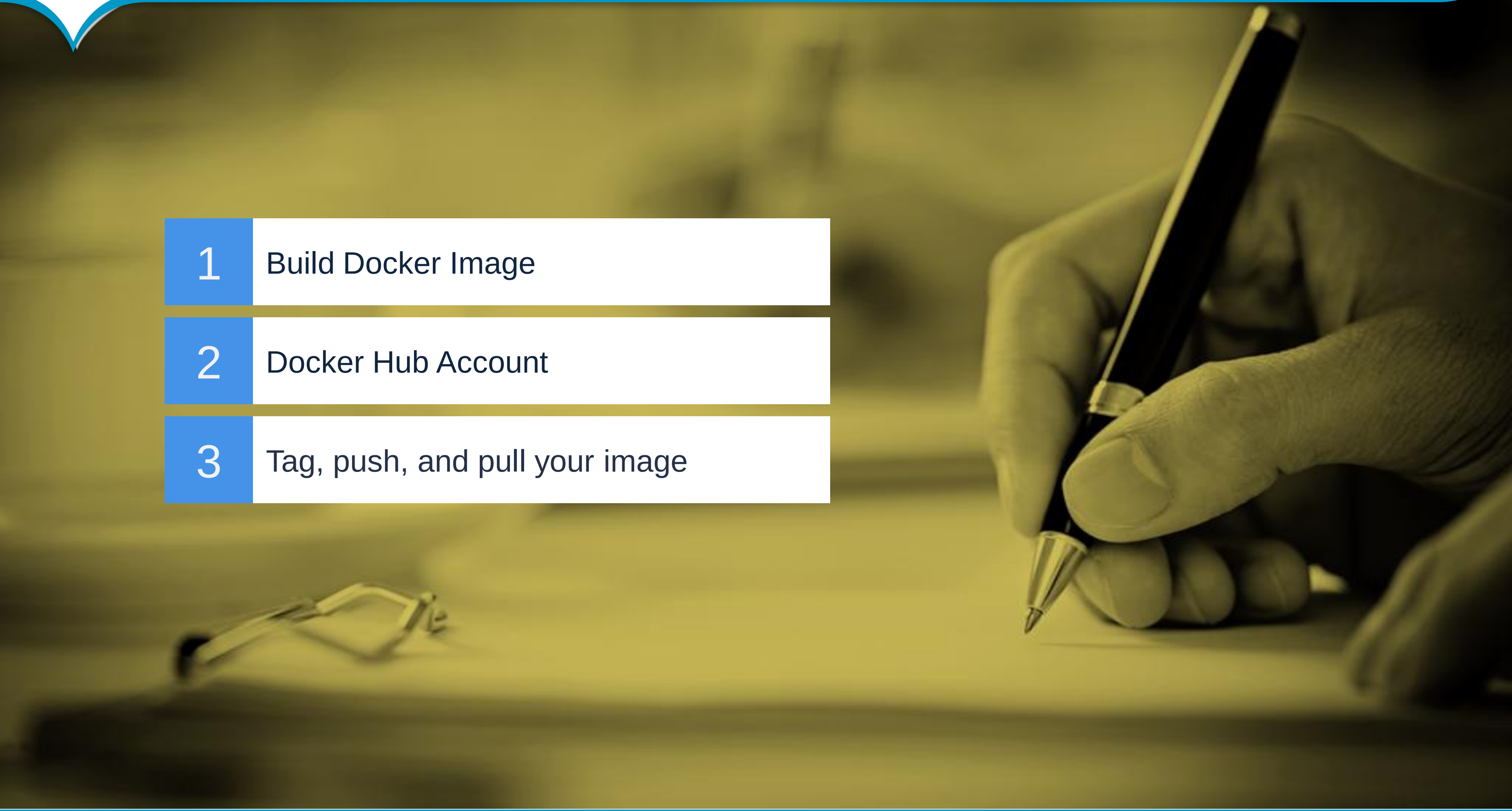
Images, Registries and Container

How it Works

Underlying Technologies

Docker Image

Outline



1 Build Docker Image

2 Docker Hub Account

3 Tag, push, and pull your image

Dockerfile commands

Command	**Description**
ADD	Copies a file from the host system onto the container
CMD	The command that runs when the container starts
ENTRYPOINT	
ENV	Sets an environment variable in the new container
EXPOSE	Opens a port for linked containers
FROM	The base image to use in the build. This is mandatory and must be the first command in the file.
MAINTAINER	An optional value for the maintainer of the script
ONBUILD	A command that is triggered when the image in the Dokerfile is used as a base for another Image
RUN	Executes a command and save the result as a new layer
USER	Sets the default user within the container
VOLUME	Creates a shared volume that can be shared among containers or by the host machine
WORKDIR	Set the default working directory for the container

Build Docker Image

docker **run** docker/whalesay cowsay boo-boo

- We can build images with the help of following 4 steps:

1. Write a Dockerfile
   - $ **mkdir** mydockerbuild
   - $ cd mydockerbuild
   - Edit a new text file named Dockerfile
     - **Linux or Mac**:
       - $ nano Dockerfile
     - **Windows**:
       - C:\> notepad Dockerfile.
   - **FROM** docker/whalesay:latest
   - **RUN** apt-get -y update && apt-get install -y fortunes
   - **CMD** /usr/games/fortune -a | cowsay

```
FROM docker/whalesay:latest
RUN apt-get -y update && apt-get install -y fortunes
CMD /usr/games/fortune -a | cowsay
```

Build Docker Image

2. Build an image from your Dockerfile

```
$ docker build -t docker-whale .

Sending build context to Docker daemon 2.048 kB
...snip...
Removing intermediate container cb53c9d09f3b
Successfully built c2c3152907b5
```

3. Learn about the build process
   - Docker checks to make sure it has everything it needs to build. This generates this message:
     - Sending build context to Docker daemon 2.048 kB
   - Docker checks to see whether it already has the whalesay image locally and pulls it from Docker hub if not. In this case, the image already exists locally because you pulled it in a previous task
   - Docker starts up a temporary container running the whalesay image

Build Docker Image

4. Run your new docker-whale

- $ docker images

```
REPOSITORY          TAG          IMAGE ID          CREATED            SIZE
docker-whale        latest       c2c3152907b5      4 minutes ago      275.1 MB
docker/whalesay     latest       fb434121fc77      4 hours ago        247 MB
hello-world         latest       91c95931e552      5 weeks ago        910 B
Run your new image by typing docker run docker-whale.
```

- $ docker run docker-whale

```
 ________________________________________
< You will be successful in your work. >
##        .
          ## ## ##       ==
        ## ## ## ##      ===
     /""""""""""""""""___/ ===
 ~~~ {~~ ~~~~ ~~~ ~~~~ ~~ ~ /  ===- ~~~
     \______ o          __/
      \    \        __/
        \____\______/
```

Docker Hub Account

- Sign up for an account
  - https://hub.docker.com/register/?utm_source=getting_started_guide&utm_medium=embedded_MacOSX&utm_campaign=create_docker_hub_account
- Verify your email and add a repository
  - Go to your email, and look for the email titled Please confirm email for your Docker ID.
  - If you don’t see the email, check your Spam folder or wait a moment for the email to arrive.
  - Open the email and click **Confirm Your Email**.
  - The browser opens Docker Hub to your profile page.
  - From that page, choose **Create Repository**.
  - Enter a Repository Name and Short Description.
  - Make sure the repo **Visibility** is set to **Public**.

Tag, push, and pull your image

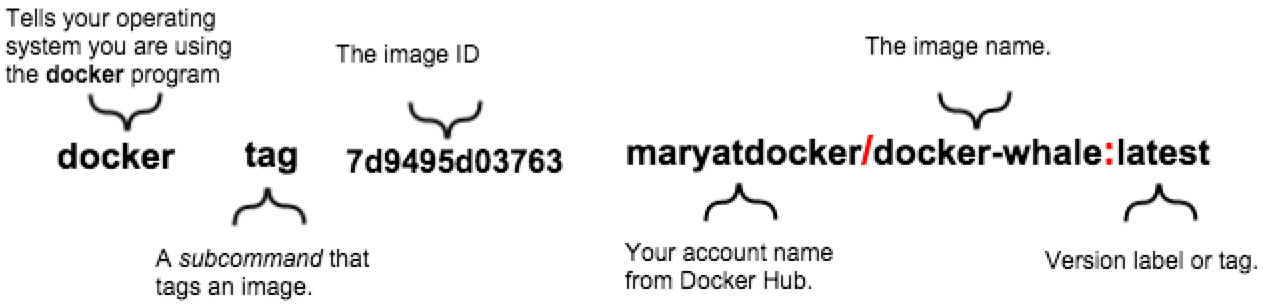
- $ docker login

```
Username: *****
 Password: *****
Login Succeeded
```

- Tag and push the image
  - $ docker images

```
REPOSITORY          TAG         IMAGE ID          CREATED             SIZE
docker-whale        latest      7d9495d03763      38 minutes ago      273.7 MB
<none>              <none>      5dac217f722c      45 minutes ago      273.7 MB
docker/whalesay     latest      fb434121fc77      4 hours ago         247 MB
hello-world         latest      91c95931e552      5 weeks ago         910 B
```

- Find the image ID for the docker-whale image
- Tag the docker-whale image using the docker tag command and the image ID.



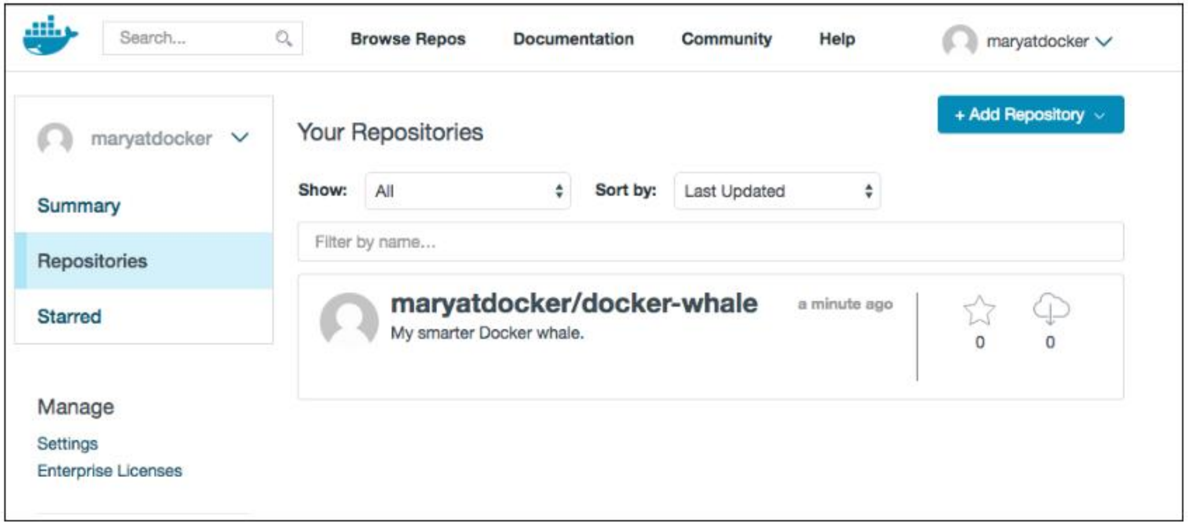
Tells your operating system you are using the **docker** program
The image ID
The image name.
docker
tag
7d9495d03763
maryatdocker/docker-whale:latest
A *subcommand* that tags an image.
Your account name from Docker Hub.
Version label or tag.

- Make sure to use your own Docker Hub account name.
- $ docker tag 7d9495d03763 maryatdocker/docker-whale:latest

Tag, push, and pull your image

- $ docker push maryatdocker/docker-whale

```
The push refers to a repository [maryatdocker/docker-whale] (len: 1)
7d9495d03763: Image already exists
...
e9e06b06e14c: Image successfully pushed
Digest: sha256:ad89e88beb7dc73bf55d456e2c600e0a39dd6c9500d7cd8d1025626c4b985011
```



Your Repositories

Show: All   Sort by: Last Updated

Filter by name...

maryatdocker/docker-whale   a minute ago
My smarter Docker whale.
0   0

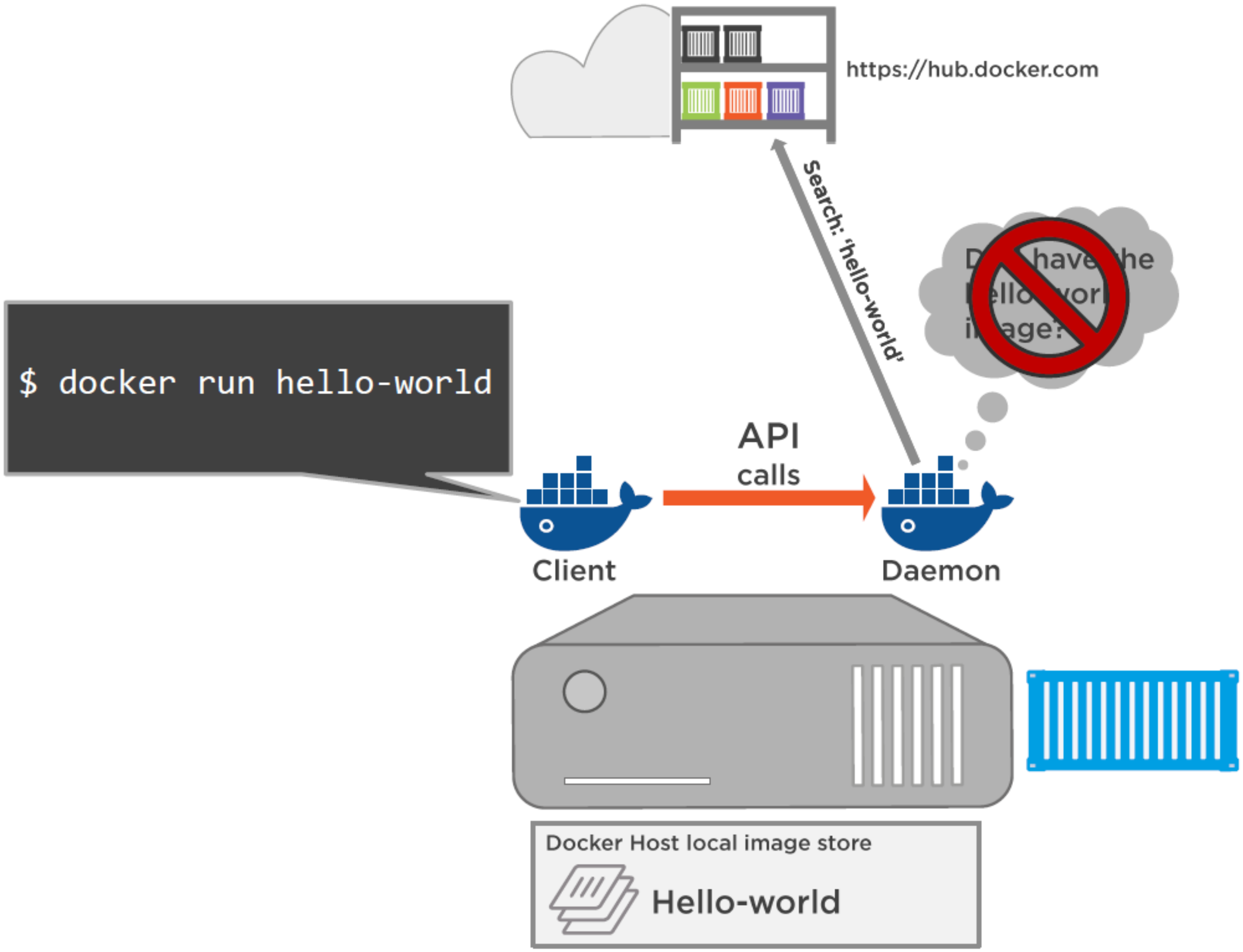
Tag, push, and pull your image

- Pull your new image
  - $ docker images

```
REPOSITORY                TAG       IMAGE ID        CREATED          SIZE
maryatdocker/docker-whale   latest    7d9495d03763    5 minutes ago    273.7 MB
docker-whale              latest    7d9495d03763    2 hours ago      273.7 MB
<none>                   <none>    5dac217f722c    5 hours ago      273.7 MB
docker/whalesay             latest    fb434121fc77    5 hours ago      247 MB
hello-world              latest    91c95931e552    5 weeks ago      910 B
```

  - $ docker rmi -f 7d9495d03763
  - $ docker **run** yourusername/docker-whale
    - Since the image is no longer available on your local system, Docker downloads it.

Tag, push, and pull your image



```
$ docker run hello-world
```

Summary

- Images
- Creating Images
- Docker Hub Account
- Tag, push, and pull your image
- Access image from Docker Hub

Capgemini

CONSULTING.TECHNOLOGY.OUTSOURCING

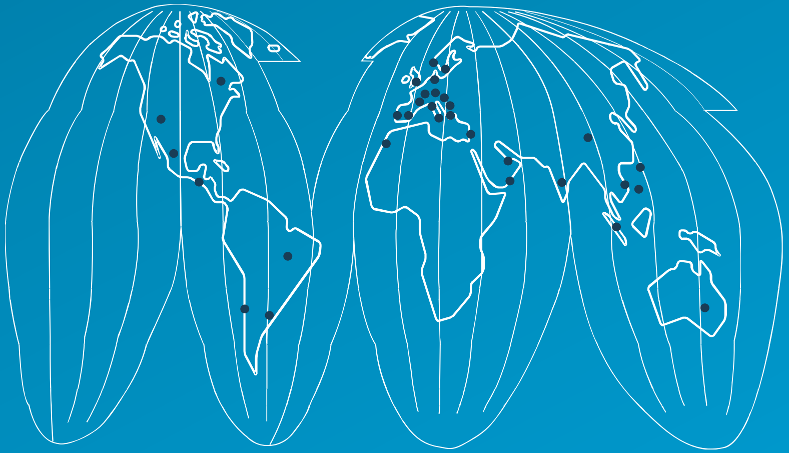
People matter, results count.

About Capgemini

With more than 145,000 people in 40 countries, Capgemini is one of the world's foremost providers of consulting, technology and outsourcing services. The Group reported 2014 global revenues of EUR 10.5 billion.

Together with its clients, Capgemini creates and delivers business and technology solutions that fit their needs and drive the results they want. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience™, and draws on Rightshore®, its worldwide delivery model.

Rightshore® is a trademark belonging to Capgemini



www.capgemini.com