fands™

A Fast AND Steady Approach

# Security

Presented by
VAISHALI TAPASWI

**FANDS INFONET Pvt.Ltd.**

**www.fandsindia.com**

# Spring Data

fands™

A Fast AND Steady Approach

# Spring Micro Services

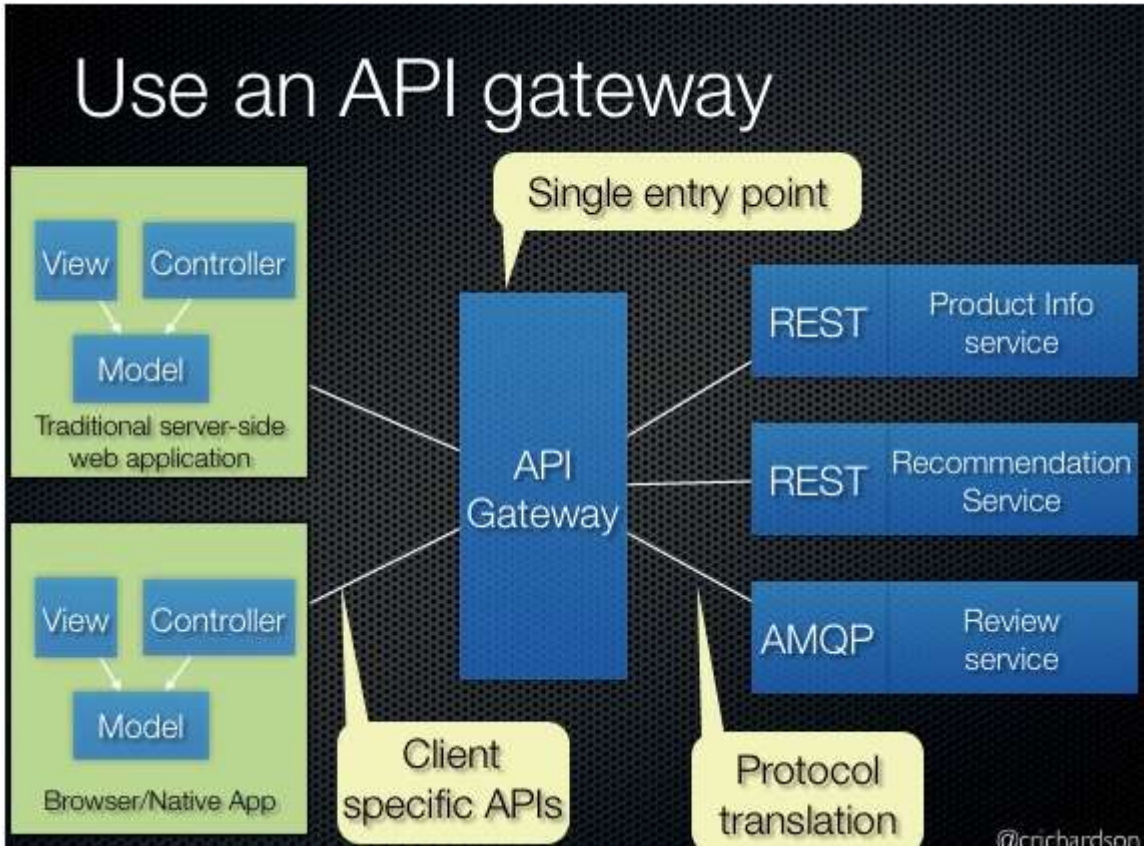
# What are Micro Services?

- known as the microservice architecture - is an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities. The microservice architecture enables the continuous delivery/deployment of large, complex applications. It also enables an organization to evolve its technology stack.

# Patterns

- API Gateway
- Service Registry
- Service Discovery
- Circuit Breaker

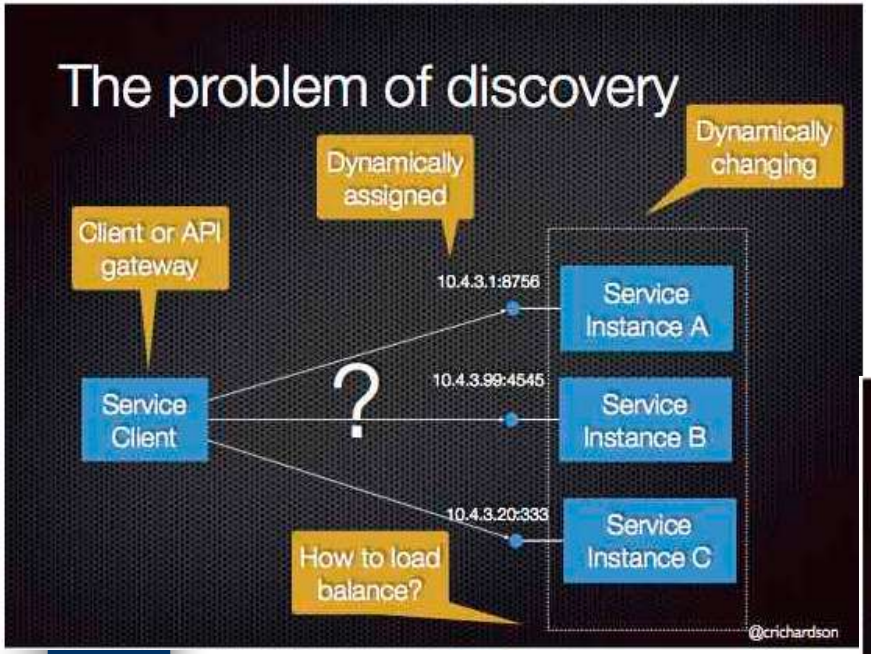
# API Gateway

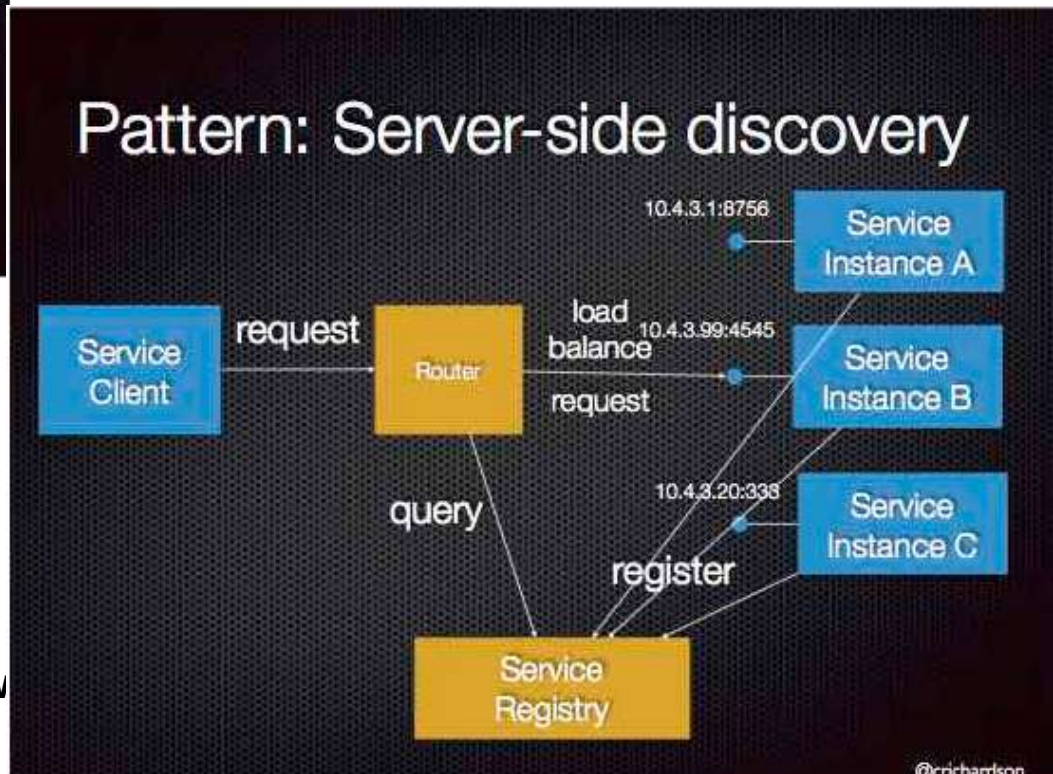

Use an API gateway
Single entry point
View
Controller
Model
Traditional server-side web application
View
Controller
Model
Browser/Native App
API Gateway
REST
Product Info service
REST
Recommendation Service
AMQP
Review service
Client specific APIs
Protocol translation

# Service Registry

- Clients of a service use either Client-side discovery or Server-side discovery to determine the location of a service instance to which to send requests.

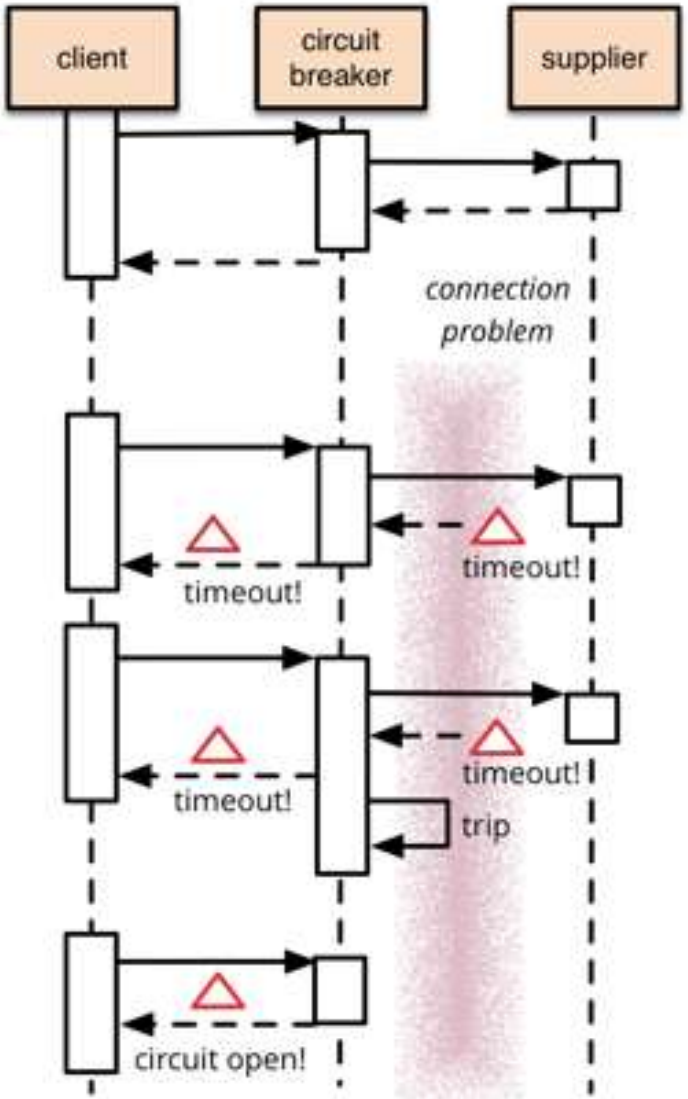
# Service Discovery



# Circuit Breaker

- One of the big differences between in-memory calls and remote calls is that remote calls can fail, or hang without a response until some timeout limi is reached. What's worse if you have many callers on a unresponsive supplier, then you can run out of critical resources leading to cascading failures across multiple systems



# Netflix Micro Services

- QConSF-MicroServices-IPC-Netflix-Sudhir-2014.pptx

# Spring Cloud Config

# Spring Cloud Config

- Spring Cloud Config provides server and client-side support for externalized configuration in a distributed system.
- With the Config Server you have a central place to manage external properties for applications across all environments.
- The concepts on both client and server map identically to the Spring Environment and PropertySource abstractions, so they fit very well with Spring applications, but can be used with any application running in any language.

# Spring Cloud Config

- As an application moves through the deployment pipeline from dev to test and into production you can manage the configuration between those environments and be certain that applications have everything they need to run when they migrate.
- The default implementation of the server storage backend uses git so it easily supports labelled versions of configuration environments, as well as being accessible to a wide range of tooling for managing the content. It is easy to add alternative implementations and plug them in with Spring configuration.

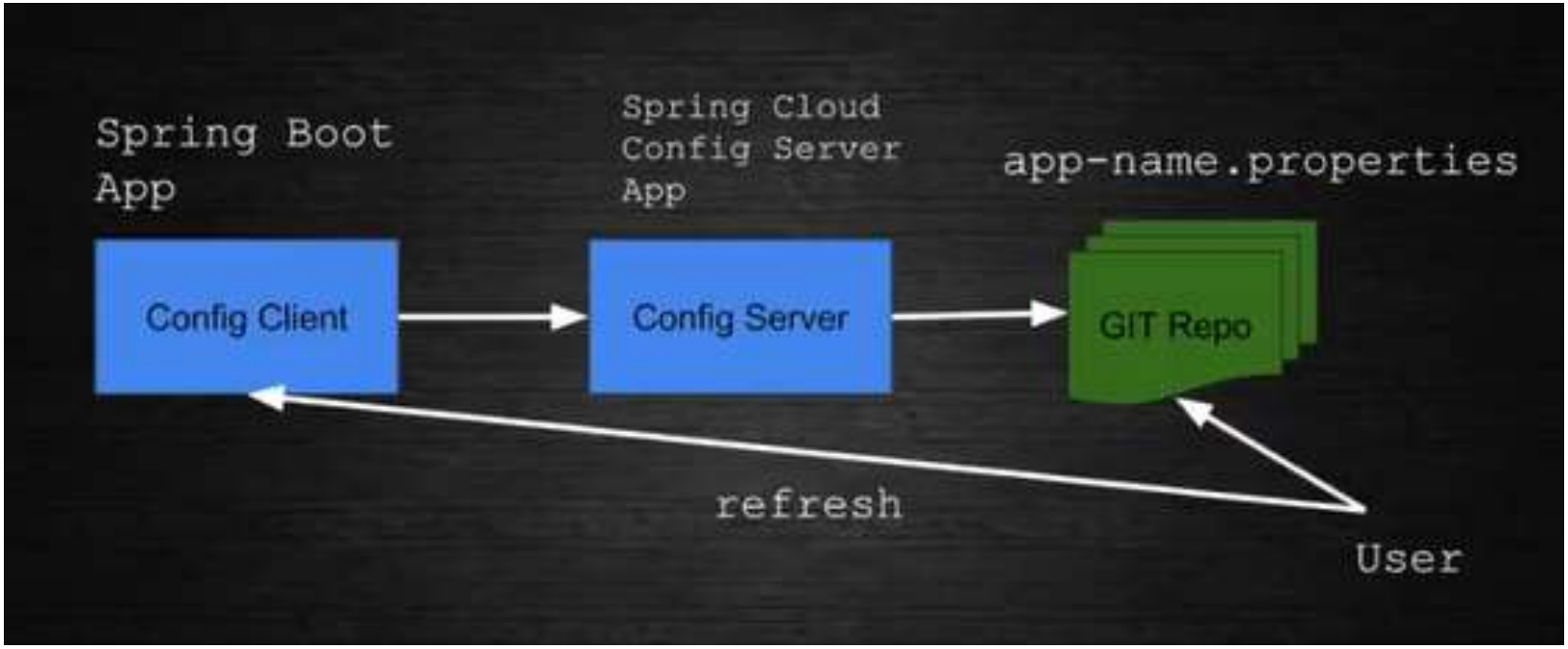
# Features

- Spring Cloud Config Server features:
  - HTTP, resource-based API for external configuration (name-value pairs, or equivalent YAML content)
  - Encrypt and decrypt property values (symmetric or asymmetric)
  - Embeddable easily in a Spring Boot application using @EnableConfigServer

# Features

- Config Client features (for Spring applications):
  - Bind to the Config Server and initialize Spring Environment with remote property sources
  - Encrypt and decrypt property values (symmetric or asymmetric)

# Lab - Create Config Server and Client



# Feign

- Feign is a java to http client binder inspired by Retrofit, JAXRS-2.0, and WebSocket. Feign's first goal was reducing the complexity of binding Denominator uniformly to http apis regardless of restfulness.
- Feign is also a declarative web service client

# Security

# Security

- Security is a crucial aspect of most applications
- Security is a concern that transcends an application's functionality
- An application should play no part in securing itself
- It is better to keep security concerns separate from application concerns

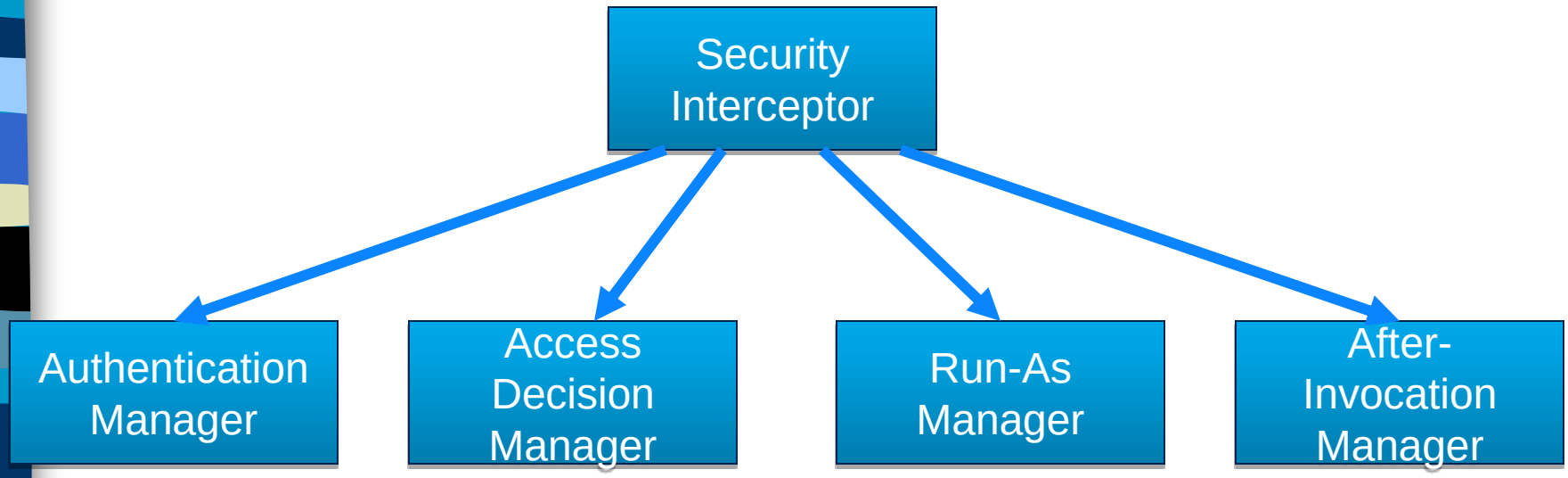
# Acegi Security


- Started in 2003
- Became extremely popular
- Security Services for the Spring framework
- From version 1.1.0, Acegi becomes a Spring Module

# Key concepts

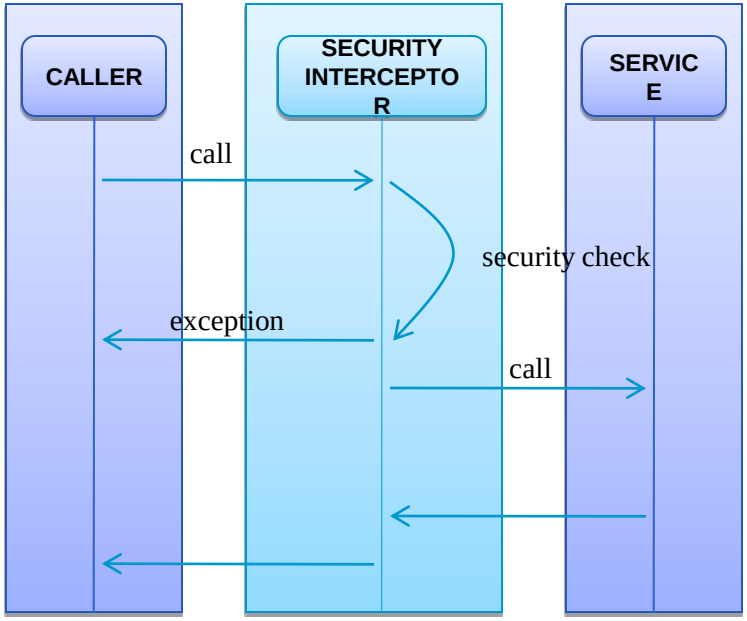
- Filters (Security Interceptor)
- Authentication
- Authorization
- Web authorization
- Method authorization

# Fundamental Elements

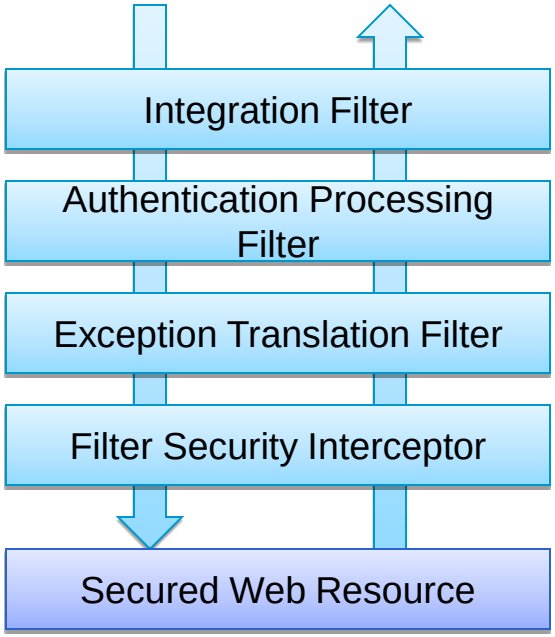


# Security Interceptor

- A latch that protects secured resources, to get past users typically enter a username and password
- Implementation depends on resource being secured
  - URLs - Servlet Filter
  - Methods - Aspects
- Delegates the
- responsibilities to the
- various managers



# Spring Security Filters




| Filter | What it does |
|---|---|
| Integration Filter | responsible for retrieving a previously stored authentication (most likely stored in the HTTP session) so that it will be ready for Spring Security's other filters to Process |
| Authentication Processing Filter | determine if the request is an authentication request. If so, the user information (typically a username/ password pair) is retrieved from the request and passed on to the authentication manager |
| Exception Translation Filter | translates exceptions, for AuthenticationException request will be sent to a login screen, for AccessDeniedException returns HTTP 403 to the browser |
| Filter Security Interceptor | examine the request and determine whether the user has the necessary privileges to access the secured resource. It leans heavily on the authentication manager and the access decision manager |

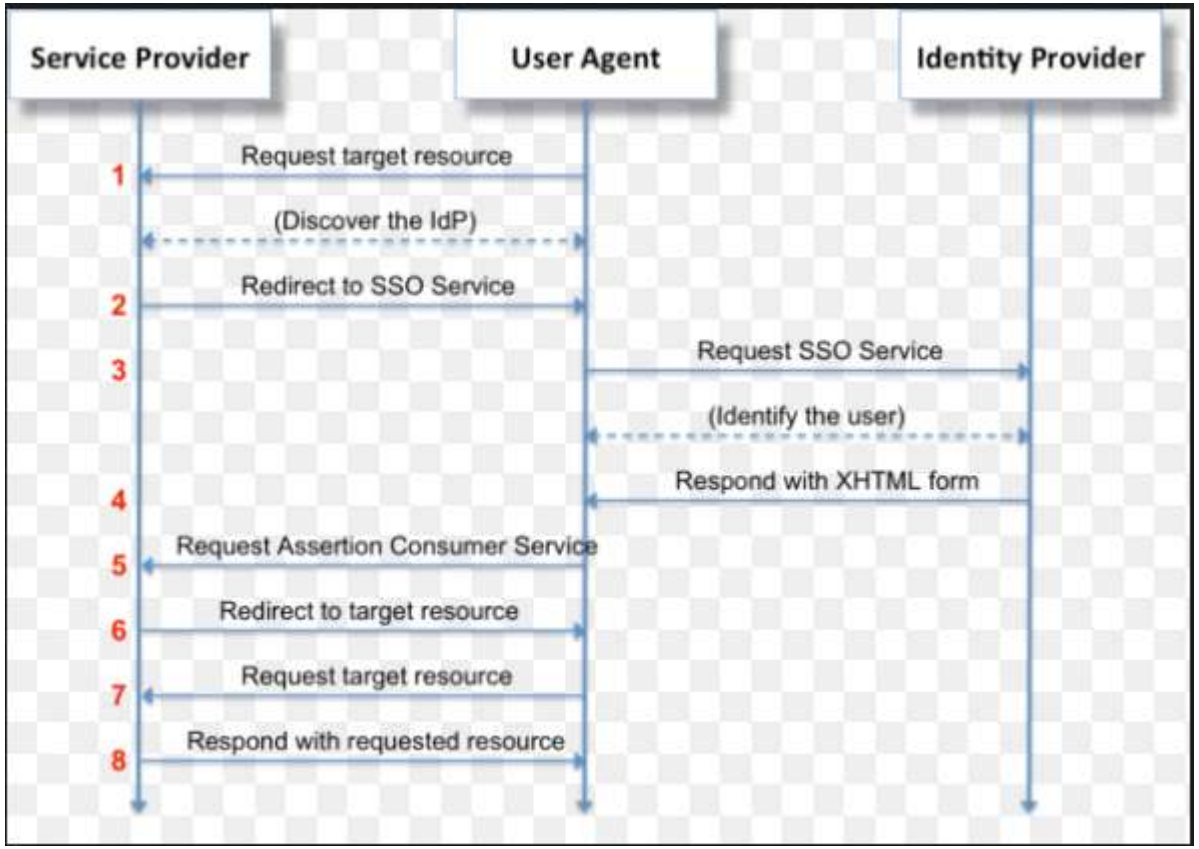
# OAuth

- OAuth is an open standard for access delegation, commonly used as a way for internet users to grant websites or applications access to their information on other websites but without giving them the passwords.

# OAuth 2.0

- OAuth 2.0 is the industry-standard protocol for authorization. OAuth 2.0 supersedes the work done on the original OAuth protocol created in 2006. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices

# SAML (SSO)


Service Provider
User Agent
Identity Provider
1
Request target resource
(Discover the IdP)
2
Redirect to SSO Service
3
Request SSO Service
(Identify the user)
4
Respond with XHTML form
5
Request Assertion Consumer Service
6
Redirect to target resource
7
Request target resource
8
Respond with requested resource

# SAML & OAuth

- SAML (Security Assertion Mark-up Language) is an umbrella standard that covers federation, identity management and single sign-on (SSO). In contrast, the OAuth (Open Authorisation) is a standard for, authorisation of resources. Unlike SAML, it doesn't deal with authentication.

# SAML Vs OAuth

| Use case type | Standard to use |
|---|---|
| Access to applications from a portal | SAML |
| Centralised identity source | SAML |
| Enterprise SSO | SAML |
| Mobile use cases | OAuth (preferably with Bearer Tokens) |
| Permanent or temporary access to resources such as accounts, files | OAuth |

| Term in SAML | Term in OAuth | Description |
|---|---|---|
| Client | Client | For example a web browser that an end user uses to access a web application |
| Identity Provider (IdP) | Authorisation Server | Server that owns the user identities and credentials |
| Service Provider (SP) | Resource Server | The protected application |

# OpenID

- OpenID is an open standard for authentication, promoted by the non-profit OpenID Foundation. As of March 2016, there are over a billion OpenID-enabled accounts on the internet, and organizations such as Google, WordPress, Yahoo, and PayPal use OpenId
- A user must obtain an OpenID account through an OpenID identity provider (for example, Google). The user will then use that account to sign into any website (the relying party) that accepts OpenID authentication (think YouTube or another site that accepts a Google account as a login).
- The OpenID standard provides a framework for the communication that must take place between the identity provider and the relying party.

| | OAuth2 | OpenId | SAML |
|---|---|---|---|
| **Token (or assertion) format** | JSON or SAML2 | JSON | XML |
| **Authorization?** | Yes | No | Yes |
| **Authentication?** | Pseudo-authentication | Yes | Yes |
| **Year created** | 2005 | 2006 | 2001 |
| **Current version** | OAuth2 | OpenID Connect | SAML 2.0 |
| **Transport** | HTTP | HTTP GET and HTTP POST | HTTP Redirect (GET) binding, SAML SOAP binding, HTTP POST binding, and others |
| **Security Risks** | Phishing<br>OAuth 2.0 does not support signature, encryption, channel binding, or client verification. Instead, it relies completely on TLS for confidentiality. | Phishing<br>Identity providers have a log of OpenID logins, making a compromised account a bigger privacy breach | XML Signature Wrapping to impersonate any user |
| **Best suited for** | API authorization | Single sign-on for consumer apps | Single sign-on for enterprise<br>Note: not well suited for mobile |

# Choosing an SSO Strategy: SAML vs OAuth2

- https://www.mutuallyhuman.com/blog/2013/05/09/choosing-an-sso-strategy-saml-vs-oauth2/

# Docker

# The Challenge

Multiplicity of Stacks

Static website

nginx 1.5 + modsecurity + openssl + bootstrap 2

User DB

postgresql + pgv8 + v8

Queue

Redis + redis-sentinel

Analytics DB

hadoop + hive + thrift + Ope

ackground workers

thon 3.0 + celery + pyredis + libcurl + ffmpeg + libopencv + nodejs + phantomjs

Web frontend

Ruby + Rails + sass + Unicorn

API endpoint

Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client

Do services and apps interact appropriately?

of hardware environmen ts

Development VM

QA server

Customer Data Center

Public Cloud

Disaster recovery

Production Servers

Production Cluster

Contributor's laptop

Can I migrate smoothly and quickly?

# The Matrix From Hell

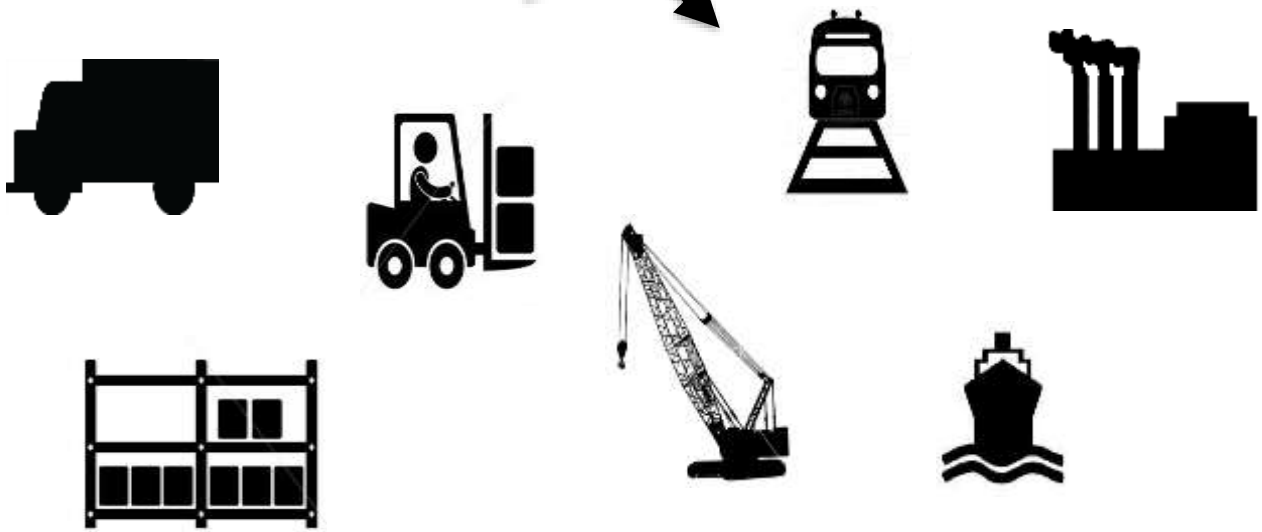
| | Development VM | QA Server | Single Prod Server | Onsite Cluster | Public Cloud | Contributor's laptop | Customer Servers |
|---|---|---|---|---|---|---|---|
| Static website | ? | ? | ? | ? | ? | ? | ? |
| Web frontend | ? | ? | ? | ? | ? | ? | ? |
| Background workers | ? | ? | ? | ? | ? | ? | ? |
| User DB | ? | ? | ? | ? | ? | ? | ? |
| Analytics DB | ? | ? | ? | ? | ? | ? | ? |
| Queue | ? | ? | ? | ? | ? | ? | ? |

# Cargo Transport Pre-1960

Multiplicity of Goods

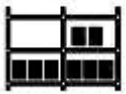
about how goods interact (e.g. coffee beans next to spices)

Multiplicity of methods for transporting/storing



quickly and smoothly (e.g. from boat to train to truck)

# Matrix Management

| | | | | | | | |
|---|---|---|---|---|---|---|---|
|  | ? | ? | ? | ? | ? | ? | ? |
|  | ? | ? | ? | ? | ? | ? | ? |
|  | ? | ? | ? | ? | ? | ? | ? |
|  | ? | ? | ? | ? | ? | ? | ? |
|  | ? | ? | ? | ? | ? | ? | ? |
|  | ? | ? | ? | ? | ? | ? | ? |
| |  |  |  |  |  |  |  |

# Solution: Intermodal Shipping Container

Multiplicity of Goods

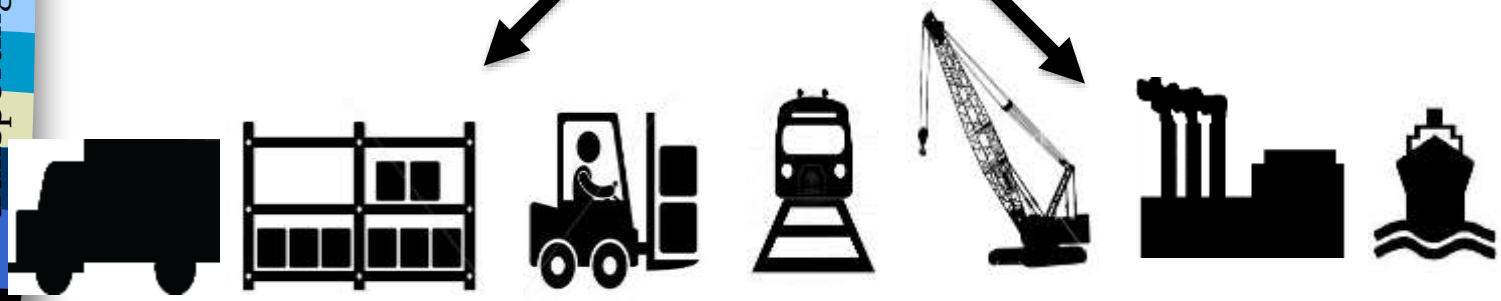
Do I worry about how goods interact (e.g. coffee beans next to spices)

A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.

…in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another

Can I transport quickly and smoothly (e.g. from boat to train to truck)

Multiplicity of methods for transporting/storing



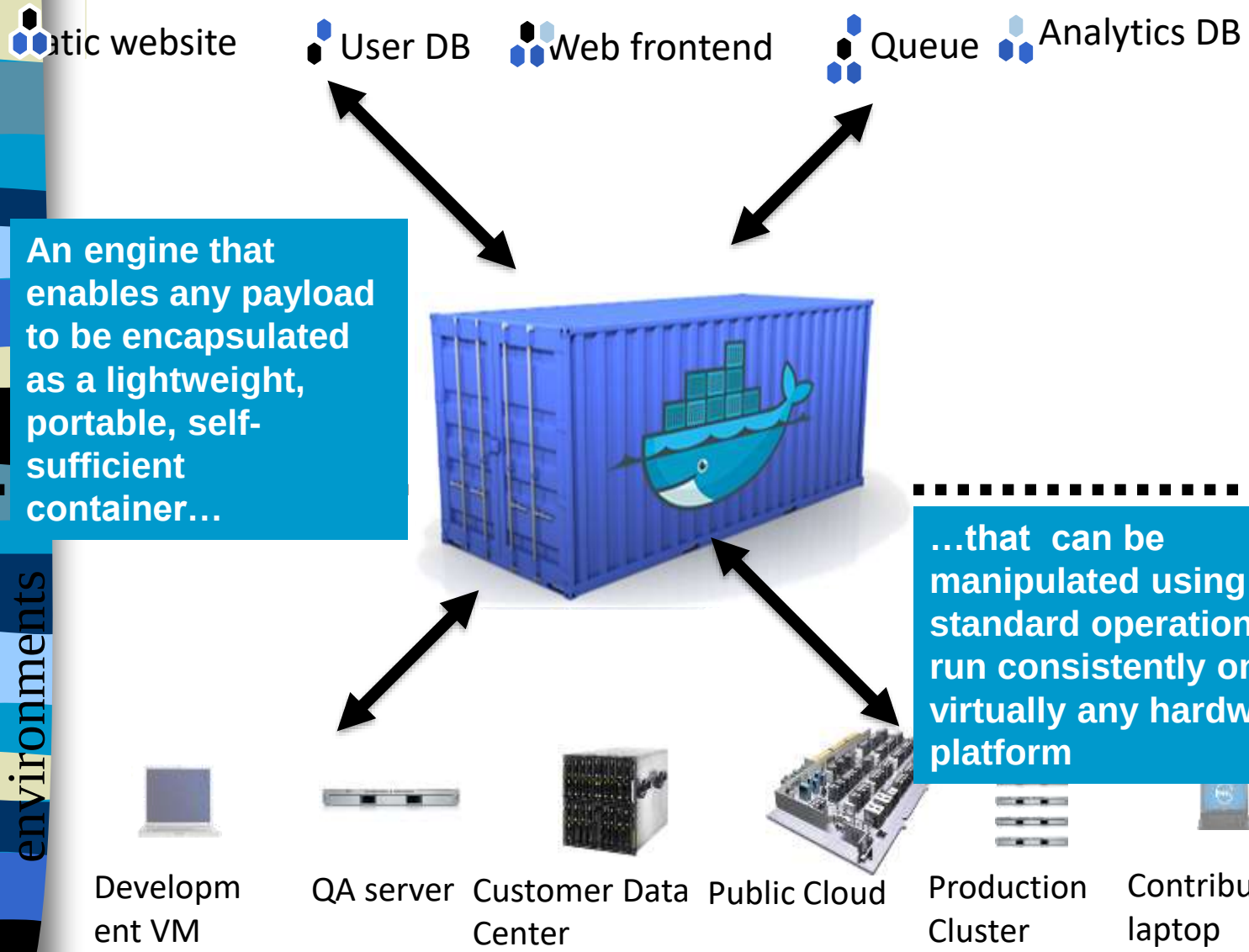
# Docker is a shipping container system for code

Multiplicity of Stacks

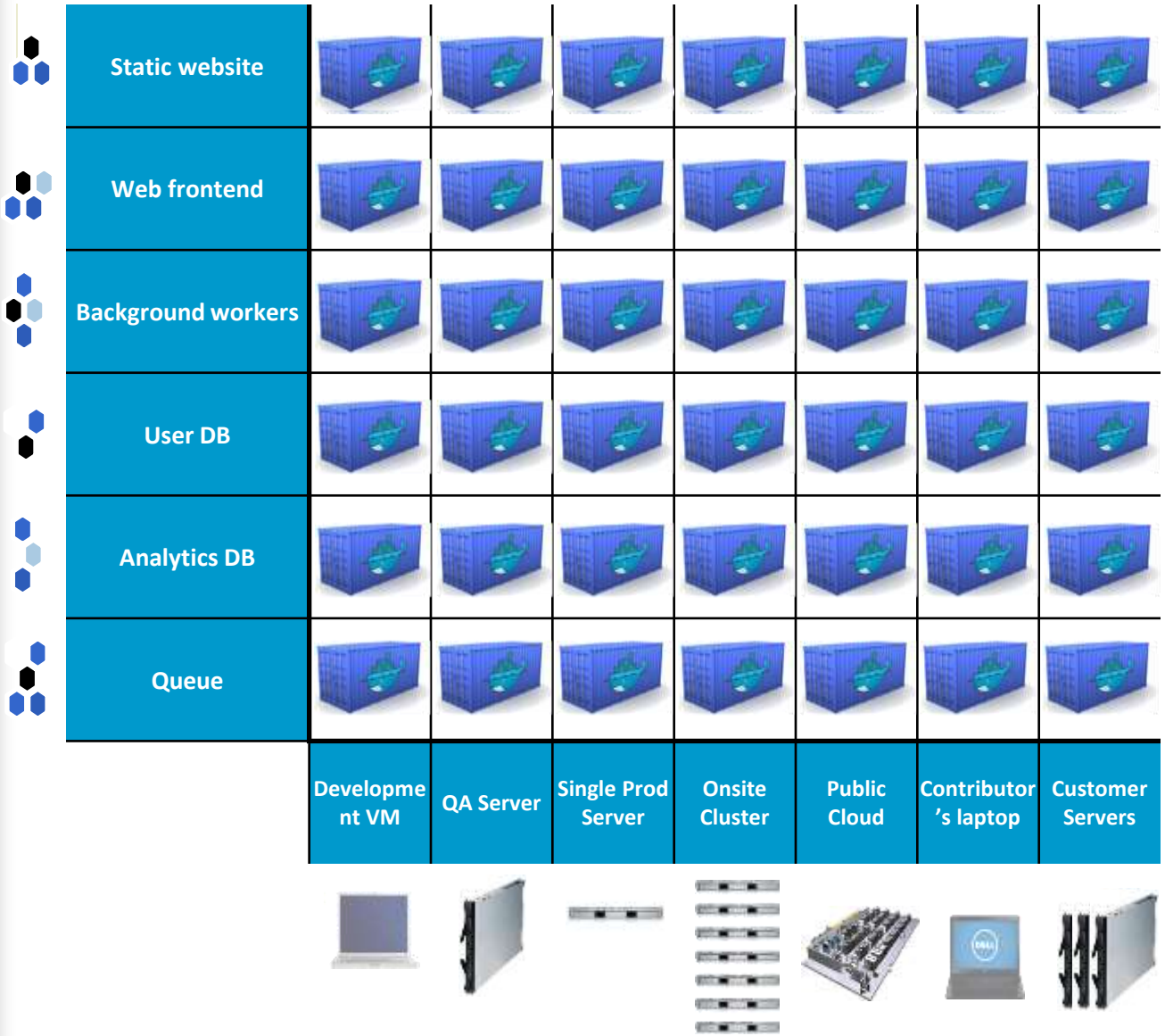
Multiplicity of hardware environments

Do services and apps interact appropriately?

Can I migrate smoothly and quickly



# Docker eliminates the matrix management



# Introduction

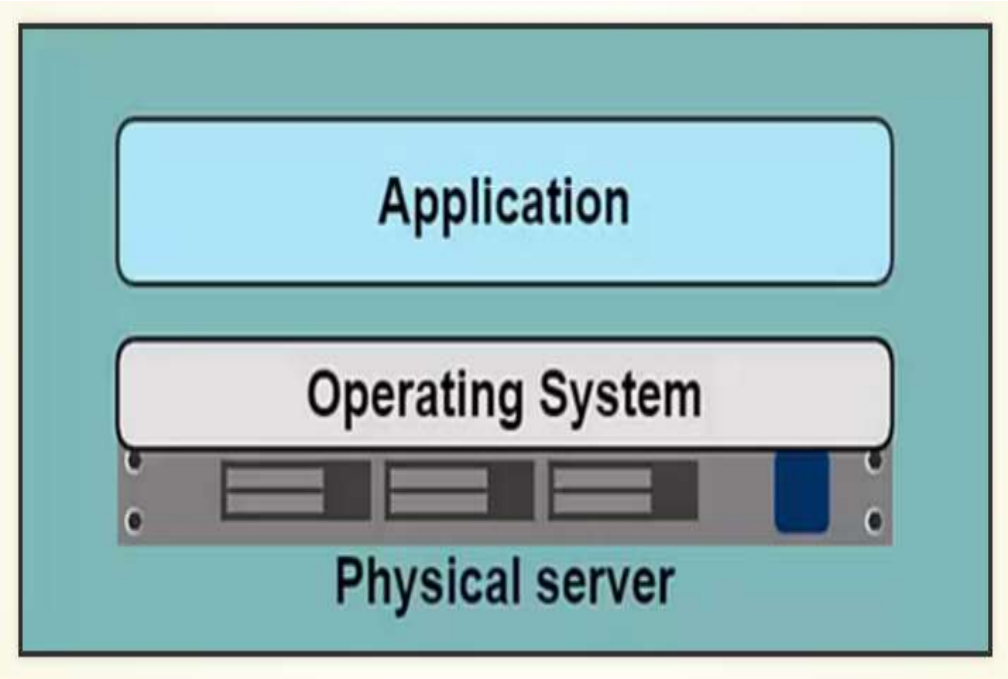
# What is Docker?

- Docker is a platform for developing, shipping and running applications using container virtualization technology.
- The Docker Platform consists of multiple tools.
  - Docker Engine
  - Docker Hub
  - Docker Machine
  - Docker Swarm
  - Docker Compose
  - Kitematic

# What is Docker?

- Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux, Mac OS and Windows.
  - Wikipedia

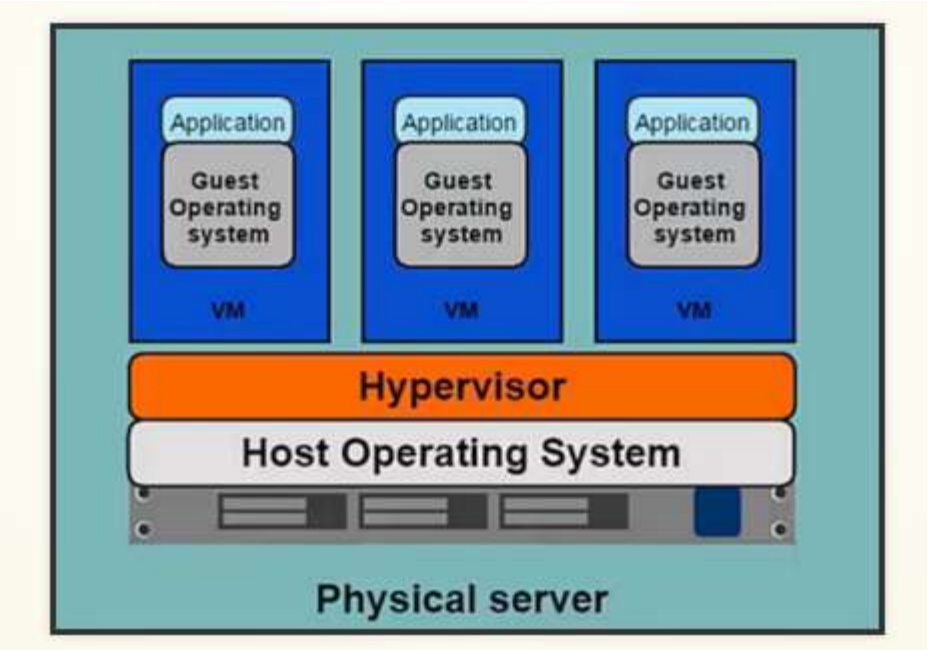
# Basic Application Hosting




- Problems
- Slow deployment times
- Huge costs
- Wasted resources
- Difficult to scale or migrate
- Vendor lock in

# Hypervisor-based Virtualization

- One physical server can contain multiple applications
- Each application runs in a virtual machine



# Pros and Cons

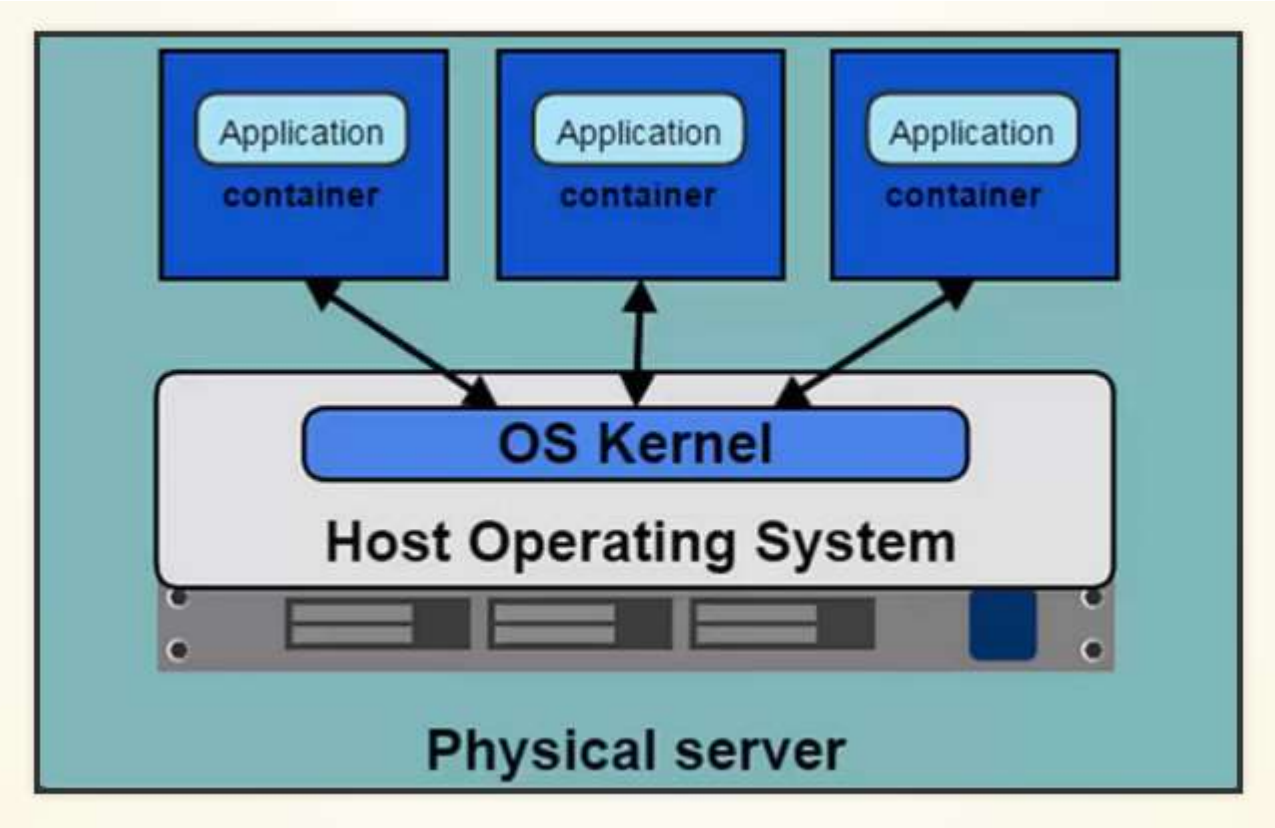
- Better resource pooling
  - one physical machine divided into multiple VM
- Easier to scale
- VM's in the cloud
  - Pay as you go
- Each VM stills requires
  - CPU allocation
  - storage
  - RAM
  - An entire guest operation system
- More VM's you run, the more resources you need
- Guest OS means wasted resources

# Introducing Containers

*Container based virtualization uses the kernel on the host's operating system to run multiple guest instances*

- Each guest instance is called a container
- Each container has its own
  - Root filesystem
  - Processes
  - Memory
  - Network ports

# Containers

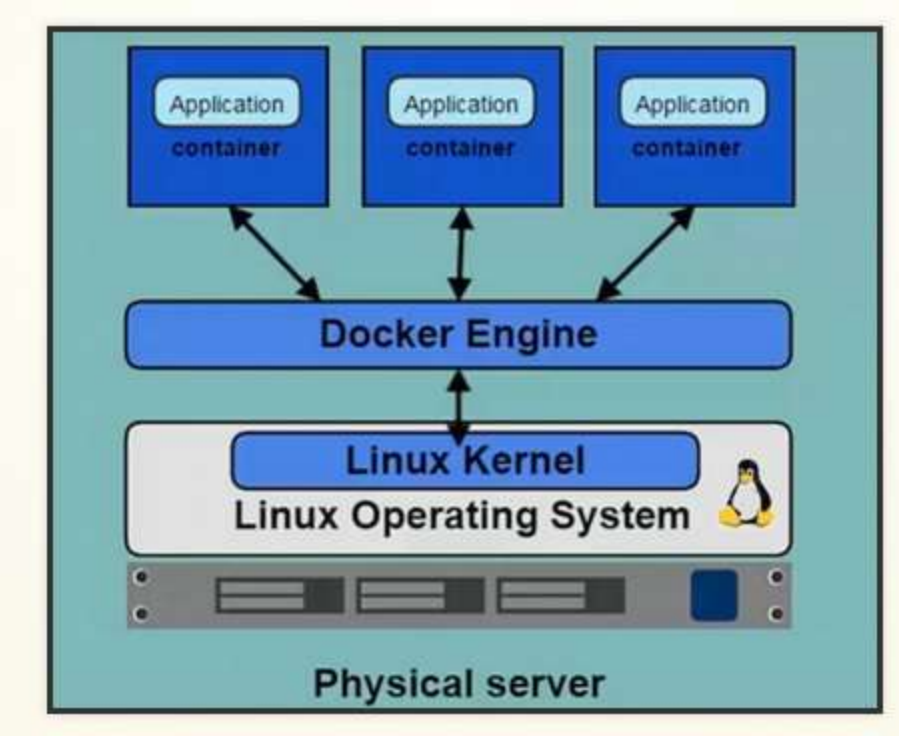

Application
container
Application
container
Application
container
OS Kernel
Host Operating System
Physical server

# Containers Vs VMs

- Containers are more lightweight
- No need to install guest OS
- Less CPU, RAM, storage space required
- More containers per machine than VMs
- Greater portability

# Docker Engine

- Docker Engine (deamon) is the program that enables containers to be built, shipped and run.
- It uses Linux Kernel namespaces and control groups
- Namespaces give us the isolated workspace

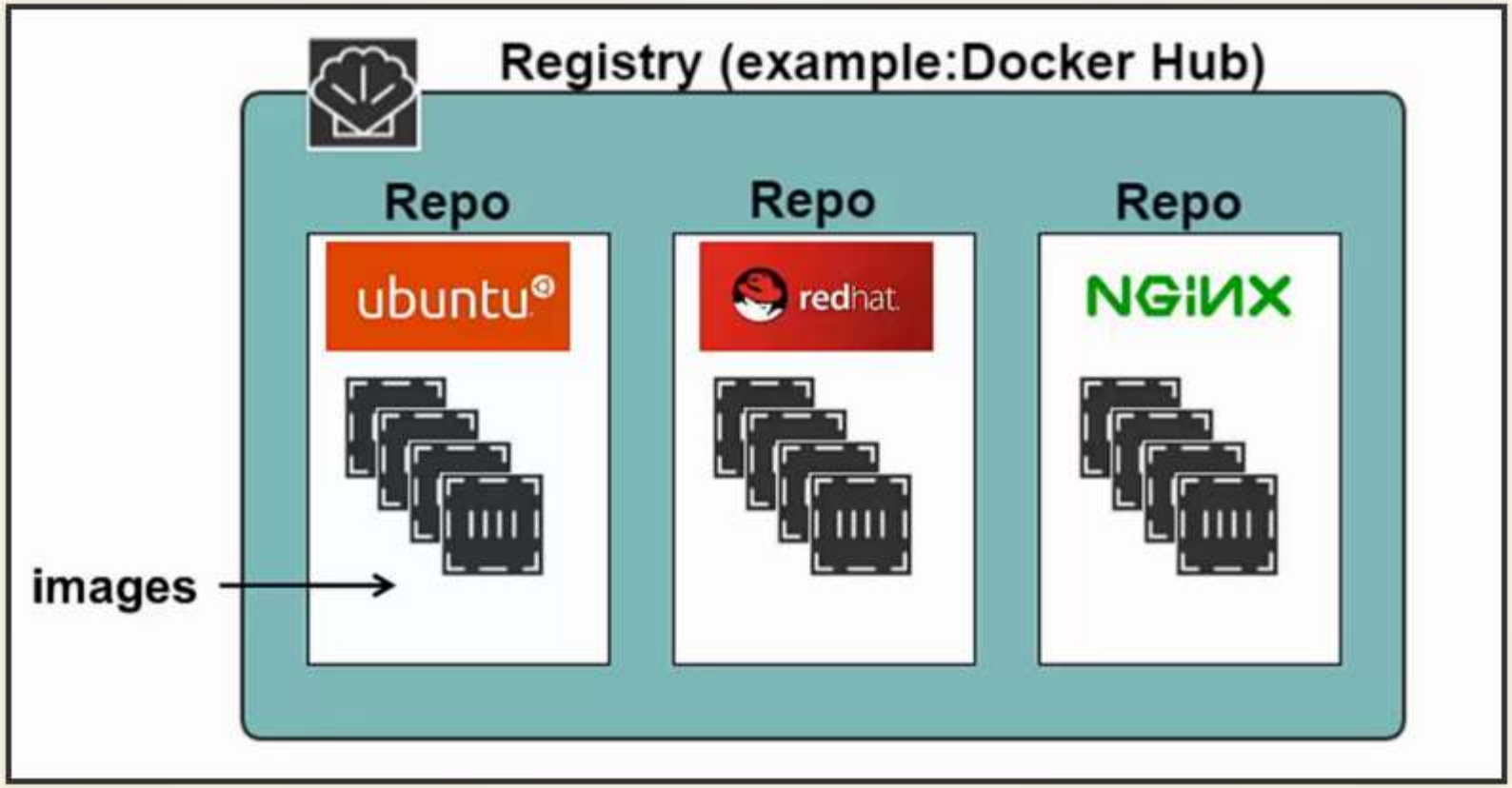


# Images and Containers

- Images
  - Read only template used to create containers
  - Built by you or other Docker users
  - Stored in the Docker Hub or your local Registry
- Containers
  - Isolated application platform
  - Contains everything needed to run your application
  - Based on images

# Registry & Repository

- Registry is where we store images. Registry can be private or public (Docker Hub)



# Docker Orchestration

- Three tools for orchestrating distributed applications with Docker
  - Docker Machine
    - Tool that provisions Docker hosts and installs the Docker Engine on them
  - Docker Swarm
    - Tool that clusters many Engines and schedules containers
  - Docker Compose
    - Tool to create and manage multi-container applications

# Benefits of Docker

- Separation of concerns
  - Developers focus on building their apps
  - System administrators focus on deployment
- Fast development cycle
- Application portability
  - Build in one environment, ship to another
- Scalability
  - Easily spin up new containers if needed
- Run more apps on one host machine

# Images

# Display Local Images

- When creating a container, docker will attempt to use a local image first
- If no local image is found, the Docker daemon will look in Docker Hub unless another registry is specified.

```
admin@admin-pc MINGW64 ~
$ docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED             VIRT
kalilinux/kali-linux-docker   latest              6f7f0e545b0c        8 days ago          573
hello-world                   latest              0a6ba66e537a        3 months ago        960
```

# Image Tags

- Images are specified by repository:tag
- The same image may have multiple tags
- Default tag is latest
- Classically Tags are version or tools
- Lookup the repository on Docker Hub to see what tags are available

fands™

A Fast AND Steady Approach

# Getting Started With Containers

# Creating a container

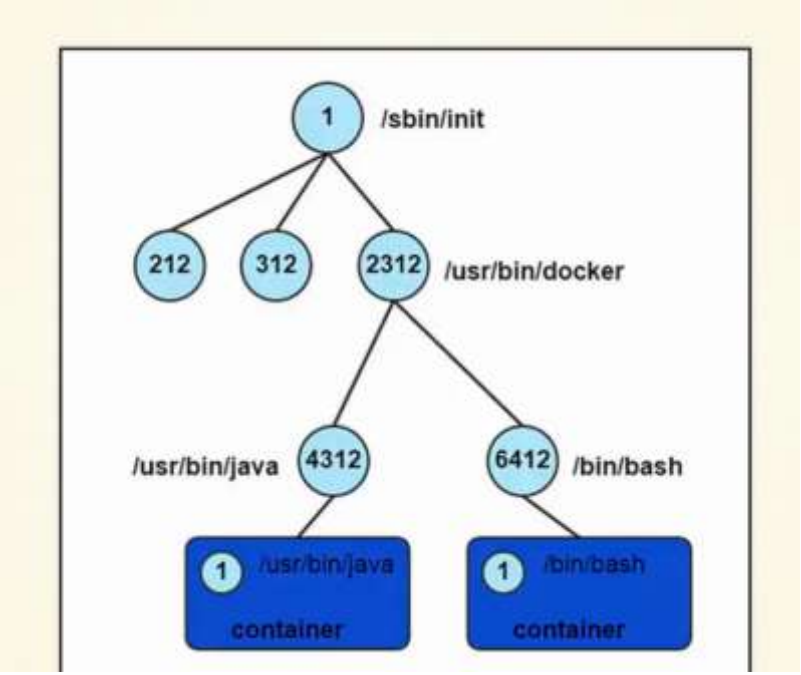
- Using docker run
  - Syntax:
  - $ docker run [options] [image] [command] [args]
    - image is specified with repository:tag
- examples:
  - docker run ubuntu:14.04 echo “HelloWorld”

# Container With Terminal

- Use some options:
  - -i flag tells docker to connect to STDIN on the container
  - -t flag specifies to get a pseudo-terminal
- EXAMPLE
  - docker run -i -t ubuntu /bin/bash

# Container Processes

- A container only runs as long as the process from your command is running
- Your command's process is always PID 1 inside the container



# Find Your Containers

- use docker ps to list running containers
- use docker ps -a to list all containers (includes containers that are stopped)

# Ports Mapping

- Run a web application inside a container
- The -P flag to map container ports to host ports

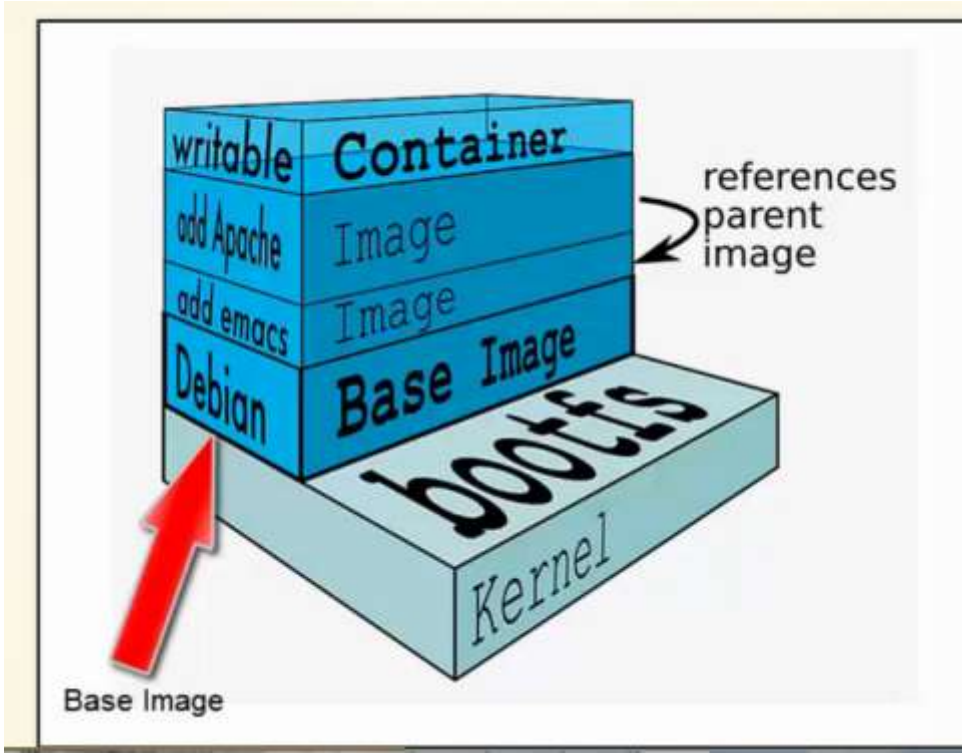
# Practical Container

- Run a web app inside a container
- The –p flag to map container ports to host ports

# Building Images

# Image Layers

- Images are comprised of multiple layers
- A layer is also just another image
- Every image contains a base layer
- Docker uses a copy on write system
- Layers are read only



# The Container Writable Layer

- Docker creates a top writable layer for containers
- Parent images are read only
- All changes are made at the writeable layer

# Managing Images And Containers

# Start And Stop Containers

- Find your containers first with docker ps and note the ID or name
- 'docker start' and 'docker stop'

```
$ docker ps -a
$ docker start <container ID>
$ docker stop <container ID>
```

# Getting Terminal Access

- Use docker exec command to **start another process** within a container
- Execute /bin/bash to get a bash shell
- docker exec -i -t [container ID] /bin/bash
- Exiting from the terminal will **not** terminate the container

# Deleting Containers and Images

- Containers that have been stopped
  - docker rm containerid
- Images
  - docker rmi [image ID]
    or
    docker rmi [repo:tag]

fands™

A Fast AND Steady Approach

# Container Networking Basics

# Mapping Ports

- **Recall:** containers have their own network and IP address
- Map exposed container ports to ports on the host machine
- Ports can be manually mapped or auto mapped
- Uses the -p and -P parameters in docker run

```
#Maps port 80 on the container to 8080 on the host
docker run -d -p 8080:80 nginx:1.9.4
```

# Automapping Ports

- Use the -P option in **docker run**
- Automatically maps exposed ports in the container to a port number in the host
- Host port numbers used go from 49153 to 65535
- Only work for ports defined in the EXPOSE instruction

```
#Auto map ports exposed by the NGINX container to a port value on the host
docker run -d -P nginx:1.9.4
```

# Expose Instruction

- Configures which ports a container will listen on at runtime
- Ports still need to be mapped when container is executed

```
FROM ubuntu:14.04
RUN apt-get update
RUN apt-get install -y nginx

EXPOSE 80 443

CMD ["nginx","-g","daemon off;"]
```

# Linking Containers

***Linking*** *is a communication method between containers which allows them to securely transfer data from one to another*

- Source and recipient containers
- Recipient containers have access to data on source containers
- Links are established based on container names

# Dockerfile

# Need of Dockerfile

- Docker can build images automatically by reading the instructions from a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession.

# Usage

- Docker build command builds an image from a Dockerfile and a context. The build's context is the files at a specified location PATH or URL. The PATH is a directory on your local filesystem. The URL is a the location of a Git repository.
- A context is processed recursively. So, a PATH includes any subdirectories and the URL includes the repository and its submodules. A simple build command that uses the current directory as context:

# Usage

- $ docker build .
  - Sending build context to Docker daemon 6.51 MB
    ...
- The build is run by the Docker daemon, not by the CLI. The first thing a build process does is send the entire context (recursively) to the daemon. In most cases, it’s best to start with an empty directory as context and keep your Dockerfile in that directory. Add only the files needed for building the Dockerfile.

# Implementation

- Traditionally, the Dockerfile is called Dockerfile and located in the root of the context. You use the -f flag with docker build to point to a Dockerfile anywhere in your file system
  - docker build -f /path/to/a/Dockerfile .
  - docker build -t shykes/myapp .
  - docker build -t shykes/myapp:1.0.2 -t shykes/myapp:latest .
- The Docker daemon runs the instructions in the Dockerfile one-by-one, committing the result of each instruction to a new image if necessary, before finally outputting the ID of your new image. The Docker daemon will automatically clean up the context you sent.

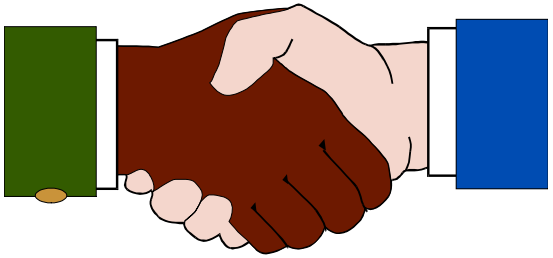
# Lab

```
FROM anapsix/alpine-java
MAINTAINER myNAME
COPY app.jar /home/app.jar
CMD ["java","-jar","/home/app.jar"]
```

- Create a simple docker file to launch a Spring web application
- Create image from Dockerfile
  - docker build -t imageName .
- Create a container from image
  - Docker run ..
- Check from external browser

# QUESTION / ANSWERS

# THANKING YOU !



www.fandsindia.com