fands™
A Fast AND Steady Approach

Spring Integration - 2

Presented by
VAISHALI TAPASWI

FANDS INFONET Pvt.Ltd.

www.fandsindia.com

Ground Rules

- **Turn off cell phone. If you cannot, please keep it on silent mode. You can go out and attend your call.**
- **If you have questions or issues, please let me know immediately.**
- **Let us be punctual.**

Agenda

fands™

A Fast AND Steady Approach

Spring Data

Spring Data

- Spring Data's mission is to provide a familiar and consistent, Spring-based programming model for data access while still retaining the special traits of the underlying data store.
- It makes it easy to use data access technologies, relational and non-relational databases, map-reduce frameworks, and cloud-based data services

Spring Data Features

- Powerful repository and custom object-mapping abstractions
- Dynamic query derivation from repository method names
- Implementation domain base classes providing basic properties
- Support for transparent auditing (created, last changed)
- Possibility to integrate custom repository code
- Easy Spring integration via JavaConfig and custom XML namespaces
- Advanced integration with Spring MVC controllers
- Experimental support for cross-store persistence

Main Modules

- Spring Data Commons
  - Core Spring concepts underpinning every Spring Data project.
- Spring Data Gemfire
  - Provides easy configuration and access to GemFire from Spring applications.
- Spring Data JPA
  - Makes it easy to implement JPA-based repositories.
- Spring Data JDBC
  - JDBC-based repositories.
- Spring Data KeyValue
  - Map-based repositories and SPIs to easily build a Spring Data module for key-value stores.

Main Modules

- Spring Data LDAP
  - Provides Spring Data repository support for Spring LDAP.
- Spring Data MongoDB
  - Spring based, object-document support and repositories for MongoDB.
- Spring Data REST
  - Exports Spring Data repositories as hypermedia-driven RESTful resources.
- Spring Data Redis
  - Provides easy configuration and access to Redis from Spring applications.
- Spring Data for Apache Cassandra
  - Spring Data module for Apache Cassandra.
- Spring Data for Apache Solr
  - Spring Data module for Apache Solr.

Spring Data JPA

- Spring Data JPA, part of the larger Spring Data family, makes it easy to easily implement JPA based repositories. This module deals with enhanced support for JPA based data access layers. It makes it easier to build Spring-powered applications that use data access technologies.

Features

- Sophisticated support to build repositories based on Spring and JPA
- Support for Querydsl predicates and thus type-safe JPA queries
- Transparent auditing of domain class
- Pagination support, dynamic query execution, ability to integrate custom data access code
- Validation of @Query annotated queries at bootstrap time
- Support for XML based entity mapping
- JavaConfig based repository configuration by introducing @EnableJpaRepositories.

Hibernate -> Spring ORM-> Data JPA

```
public void create(Dept dept) {
          Session session = null;
          Transaction tx = null;
          try {
             session = sf.openSession();
             tx = session.beginTransaction();
            session.save(dept);
            tx.commit();
            } catch (Exception e) {
                   ….
          } finally {
                         session.close();
                   }
          }
```

```
void create(Dept d){
  template.save(d);
}
```

```
public interface
DeptRepository extends
CrudRepository<Dept,
Integer> {
..
}
```

m

Lab 1 – CRUD

- Create basic crud example using hsqldb embedded database with Spring Boot
- Create application.properties file to communicate with external hsqldb instance
- Create application.yaml

fands™

CRUDRepository Vs JPARepository

```
public interface CrudRepository<T, ID extends
Serializable>
  extends Repository<T, ID> {
  <S extends T> S save(S entity);
  Optional<T> findById(ID primaryKey);
  Iterable<T> findAll();
  long count();
  void delete(T entity);
  boolean existsById(ID primaryKey);
…..
}
```

Defining Query Methods

- Two ways to derive a store-specific query
  - Query from the method name directly
  - Using a manually defined query

Strategies

- CREATE
  - attempts to construct a store-specific query from the query method name.
- USE_DECLARED_QUERY
  - tries to find a declared query and will throw an exception in case it can’t find one. The query can be defined by an annotation somewhere or declared by other means.
- CREATE_IF_NOT_FOUND
  - (default) combines CREATE and USE_DECLARED_QUERY.
  - It looks up a declared query first, and if no declared query is found, it creates a custom method name-based query.

Query Creation

- The mechanism strips the prefixes find…By, read…By, query…By, count…By, and get…By from the method and starts parsing the rest of it.
- The introducing clause can contain further expressions such as a Distinct to set a distinct flag on the query to be created. However, the first By acts as delimiter to indicate the start of the actual criteria. At a very basic level you can define conditions on entity properties and concatenate them with And and Or.

Query Examples

```
interface PersonRepository extends Repository<User, Long> {
  List<Person> findByEmailAddressAndLastname(EmailAddress
emailAddress, String lastname);
  // Enables the distinct flag for the query
  List<Person> findDistinctPeopleByLastnameOrFirstname(String
lastname, String firstname);
  List<Person> findPeopleDistinctByLastnameOrFirstname(String
lastname, String firstname);
  // Enabling ignoring case for an individual property
  List<Person> findByLastnameIgnoreCase(String lastname);
  // Enabling ignoring case for all suitable properties
  List<Person> findByLastnameAndFirstnameAllIgnoreCase(String
lastname, String firstname);
  // Enabling static ORDER BY for a query
  List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
  List<Person> findByLastnameOrderByFirstnameDesc(String lastname);
```

Special Parameter Handling

- To handle parameters in your query you simply define method parameters as already seen in the examples above.
- Besides that the infrastructure will recognize certain specific types like Pageable and Sort to apply pagination and sorting to your queries dynamically

Special Parameter Handling

- A Page knows about the total number of elements and pages available. It does so by the infrastructure triggering a count query to calculate the overall number.
- As this might be expensive depending on the store used, Slice can be used as return instead. A Slice only knows about whether there’s a next Slice available which might be just sufficient when walking through a larger result set.

Examples

- Page<User> findByLastname(String lastname, Pageable pageable);
- Slice<User> findByLastname(String lastname, Pageable pageable);
- List<User> findByLastname(String lastname, Sort sort);
- List<User> findByLastname(String lastname, Pageable pageable);

Limiting query results

- The results of query methods can be limited via the keywords first or top, which can be used interchangeably. An optional numeric value can be appended to top/first to specify the maximum result size to be returned. If the number is left out, a result size of 1 is assumed.

Examples

- User findFirstByOrderByLastnameAsc();
- User findTopByOrderByAgeDesc();
- Page<User> queryFirst10ByLastname(String lastname, Pageable pageable);
- Slice<User> findTop3ByLastname(String lastname, Pageable pageable);
- List<User> findFirst10ByLastname(String lastname, Sort sort);
- List<User> findTop10ByLastname(String lastname, Pageable pageable);

Streaming query results

- The results of query methods can be processed incrementally by using a Java 8 Stream<T> as return type.
- Instead of simply wrapping the query results in a Stream data store specific methods are used to perform the streaming.

Example

```
@Query("select u from User u")
Stream<User>
findAllByCustomQueryAndStream();

Stream<User> readAllByFirstnameNotNull();

@Query("select u from User u")
Stream<User> streamAllPaged(Pageable
pageable);
```

Async Query Results

- Repository queries can be executed asynchronously using Spring’s asynchronous method execution capability. This means the method will return immediately upon invocation and the actual query execution will occur in a task that has been submitted to a Spring TaskExecutor

@ASync

```
@Async
Future<User> findByFirstname(String firstname);

@Async
CompletableFuture<User> findOneByFirstname(String
firstname);

@Async
ListenableFuture<User> findOneByLastname(String
lastname);
```

Spring Micro Services

www.fandsindia.com

What are Micro Services?

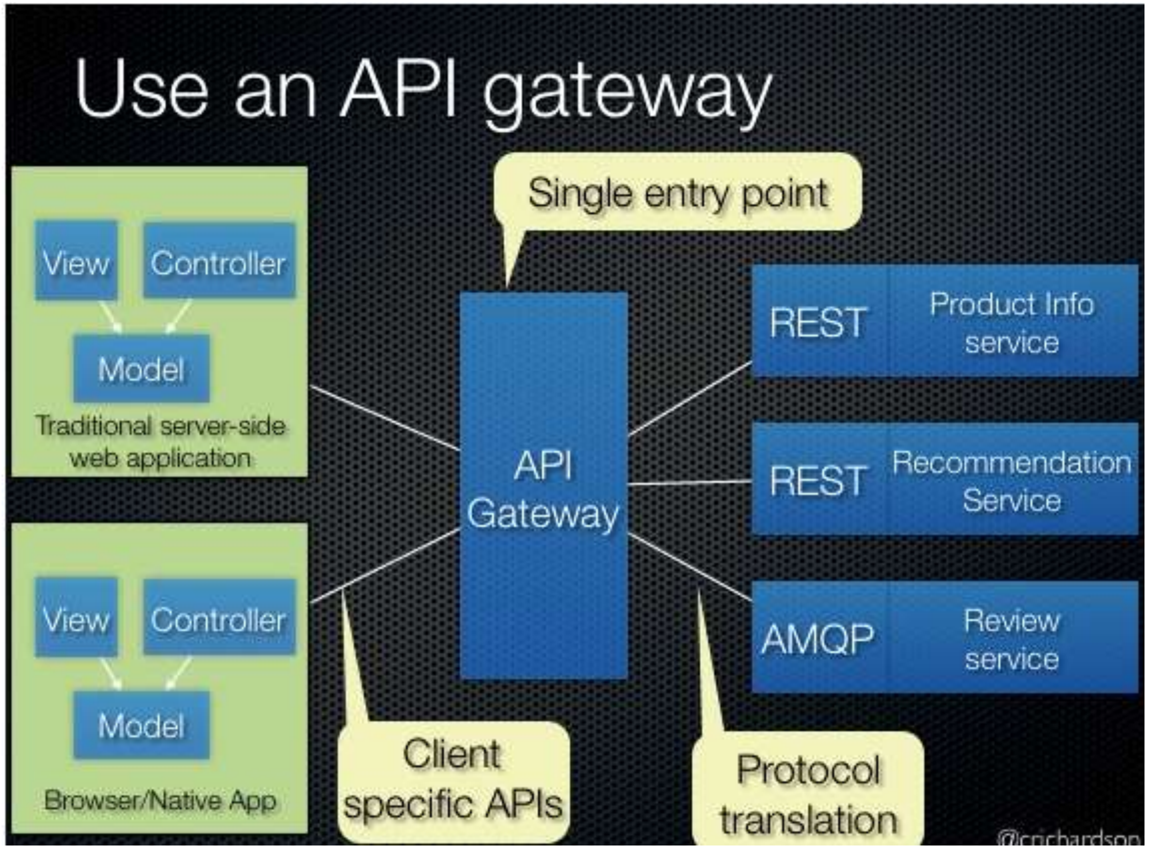
- known as the microservice architecture - is an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities. The microservice architecture enables the continuous delivery/deployment of large, complex applications. It also enables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

Patterns

- API Gateway
- Service Registry
- Service Discovery

fands™

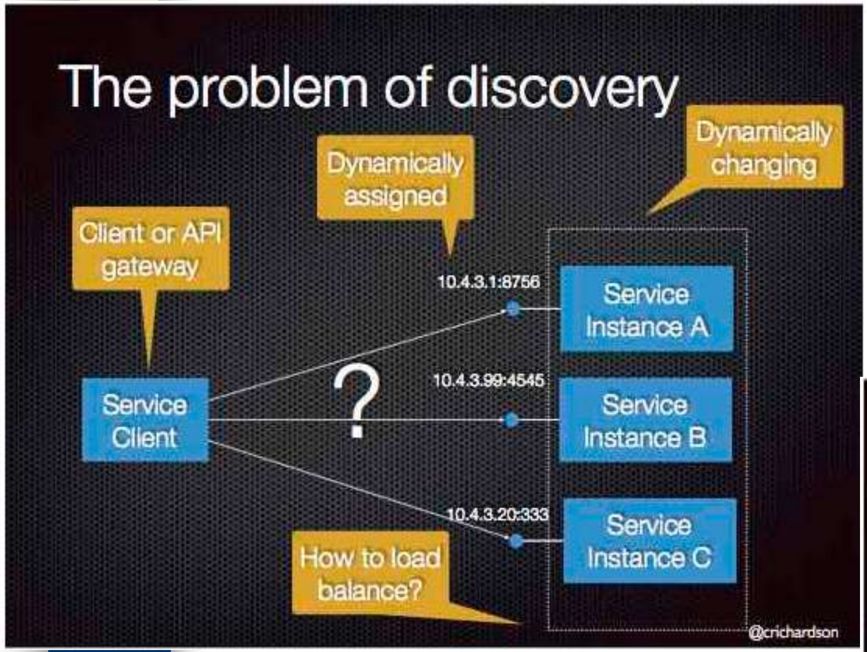
API Gateway

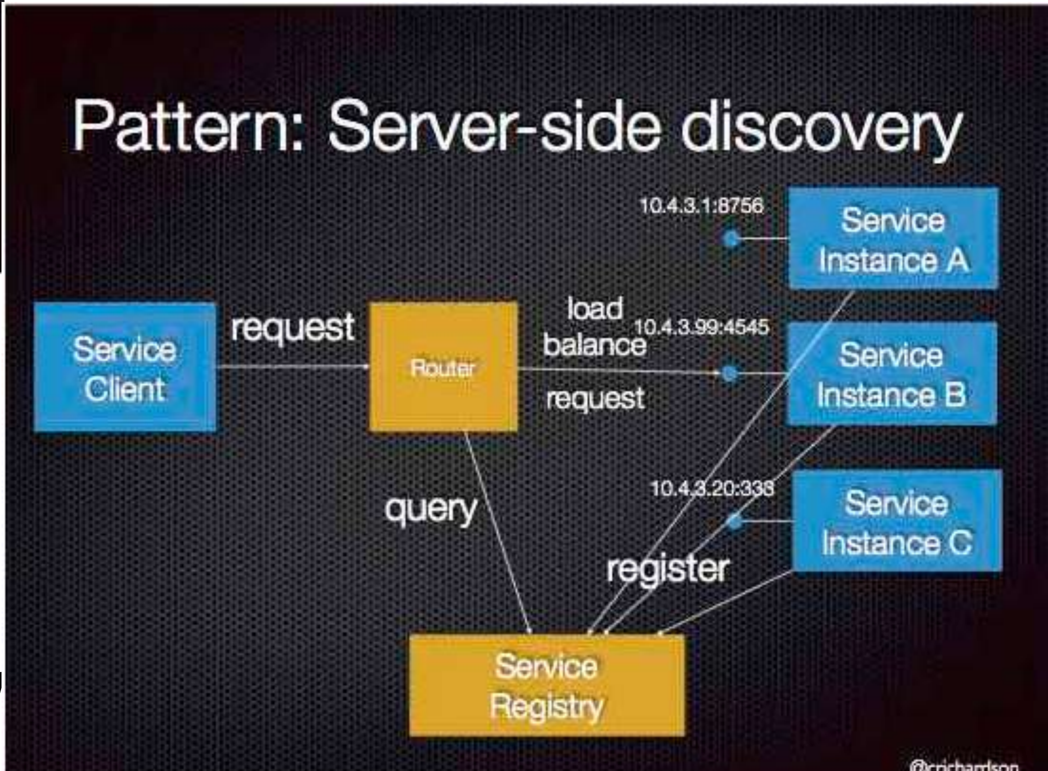


Service Registry

- 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





W

Netflix Micro Services

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

A Fast AND Steady Approach

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.

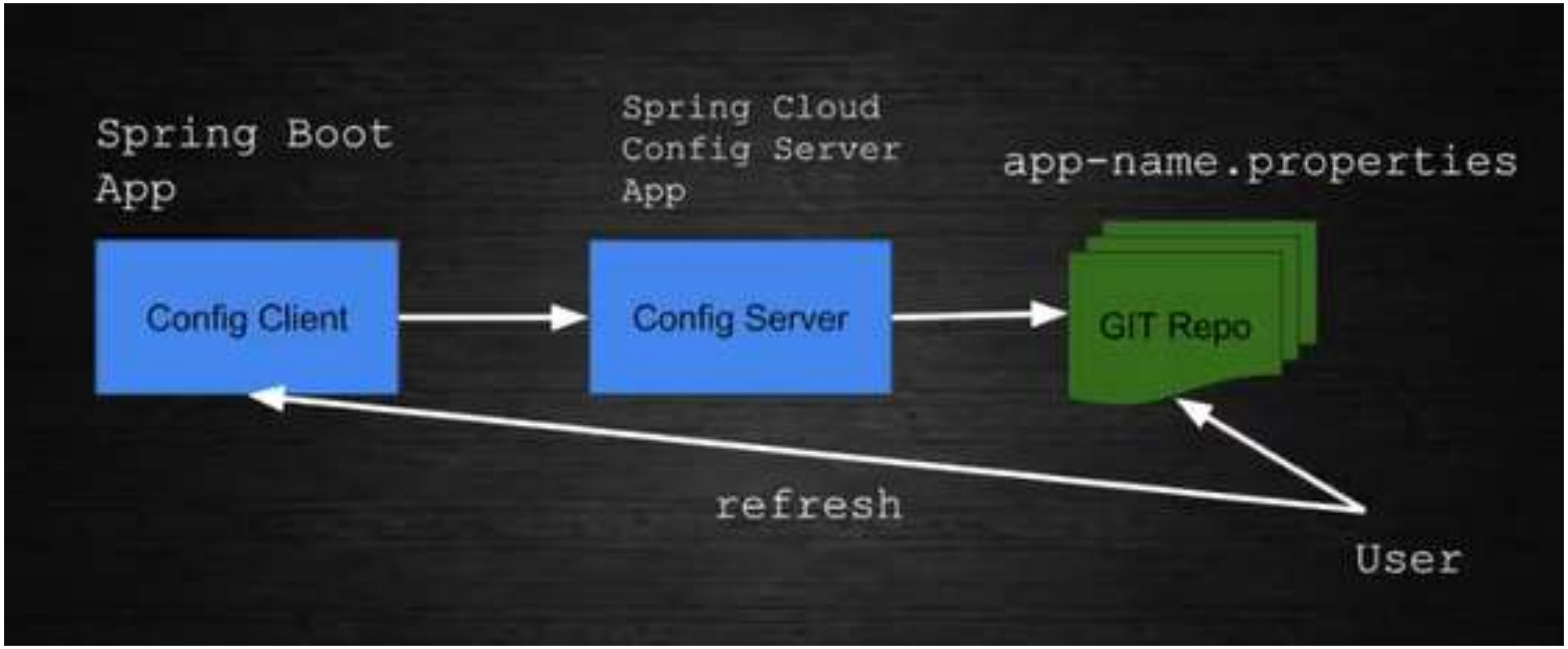
Features

- 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



Lab 1 – Create Config Server

- Spring Boot Application
- Dependencies
  - Spring Cloud Config Server
- ConfigServiceApplication Class
  - @EnableConfigServer , @SpringBootApplication
- Application.properties/yaml file
  - server.port=8888
  - spring.cloud.config.server.git.uri=..

Lab 2 – Create Config Client

- Spring Boot Application
- Dependencies
  - Spring Cloud Config Client, actuator, web
- Bootstrap.properties
  - spring.application.name=…
  - spring.cloud.config.uri=http://localhost:8888
  - management.security.enabled=false
  - spring.cloud.config.fail-fast=true
- A rest service to display properties

```
@RefreshScope
@RestController
class MessageRestController
{


@Value("${message:Hello
default}")
   private String message;


@RequestMapping("/messag
e")
   String getMessage() {
      return this.message;
   }
}
```

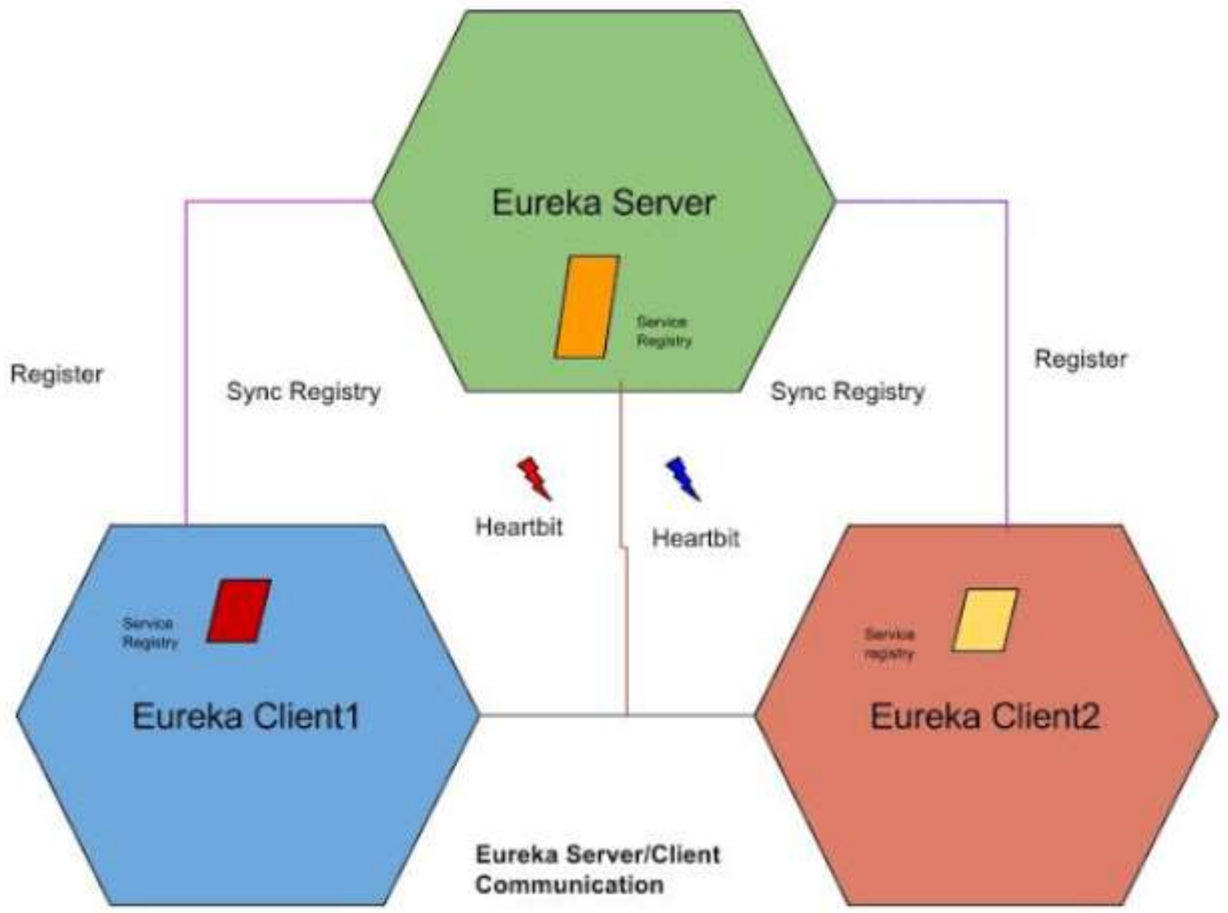
Spring Cloud Netflix

- Spring Cloud Netflix provides Netflix OSS integrations for Spring Boot apps through autoconfiguration and binding to the Spring Environment and other Spring programming model idioms. With a few simple annotations you can quickly enable and configure the common patterns inside your application and build large distributed systems with battle-tested Netflix components.
- The patterns provided include
  - Service Discovery (Eureka),
  - Intelligent Routing (Zuul)
  - Client Side Load Balancing (Ribbon)
  - Circuit Breaker (Hystrix),

Features

- Service Discovery: Eureka instances can be registered and clients can discover the instances using Spring-managed beans
- Service Discovery: an embedded **Eureka** server can be created with declarative Java configuration
- Circuit Breaker: Hystrix clients can be built with a simple annotation-driven method decorator
- Circuit Breaker: embedded **Hystrix** dashboard with declarative Java configuration
- Declarative REST Client: **Feign** creates a dynamic implementation of an interface decorated with JAX-RS or Spring MVC annotations
- Client Side Load Balancer: **Ribbon**
- External Configuration: a bridge from the Spring Environment to **Archaius** (enables native configuration of Netflix components using Spring Boot conventions)
- Router and Filter: automatic regsitration of **Zuul** filters, and a simple convention over configuration approach to reverse proxy creation

Lab - Eureka


Eureka Server
Service Registry
Register
Sync Registry
Sync Registry
Register
Heartbit
Heartbit
Service Registry
Eureka Client1
Service registry
Eureka Client2
Eureka Server/Client Communication

Lab 1 – Create Eureka Server
(Guides)

- Create application class
  - @EnableEurekaServer
- Application.properties

```
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
logging.level.com.netflix.eureka=OFF
logging.level.com.netflix.discovery=OFF
```

Lab 2 – Create Eureka Client

(Guides)

- Application - @EnableDiscoveryClient
- Bootstrap.properties - spring.application.name=a-bootiful-client
- Rest Service

```
@RestController
class ServiceInstanceRestController
{
@Autowired
private DiscoveryClient discoveryClient;
@RequestMapping("/service-instances/{applicationName}")
public List<ServiceInstance> serviceInstancesByApplicationName( @PathVariabl
applicationName)
{ return this.discoveryClient.getInstances(applicationName);
}
}
```

Lab 3

- Create DB service
- Register to Eureka Service
- Use the same in Eureka Client

```
ResponseEntity<List<String>> quoteResponse =
restTemplate.exchange("http://db-service/rest/db/" +
userName, HttpMethod.GET,
                null, new
ParameterizedTypeReference<List<String>>() {
                });
```

Lab 4

- Zuul

```
zuul:
#Service will be mapped under the /api URI
 prefix: /api
 routes:
  db-service:
   path: /db-service/**
   url: http://localhost:8300
  stock-service:
   path: /stock-service/**
   url: http://localhost:8301
```

Lab 5

- Ribbon

```
say-hello:
  ribbon:
    eureka:
      enabled: false
    listOfServers: 
localhost:8090,localhost:9092,localhost:9999
    ServerListRefreshInterval: 15000
```

Hystrix

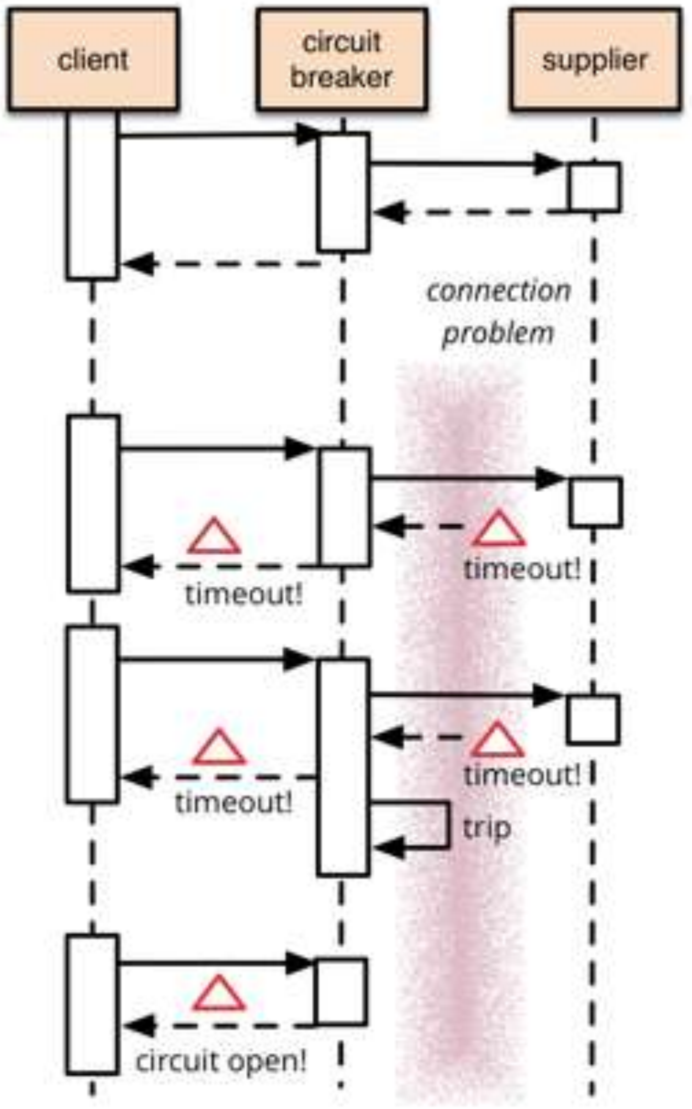
- In a distributed environment, inevitably some of the many service dependencies will fail. Hystrix is a library that helps you control the interactions between these distributed services by adding latency tolerance and fault tolerance logic. Hystrix does this by isolating points of access between the services, stopping cascading failures across them, and providing fallback options, all of which improve your system’s overall resiliency.

What Is Hystrix For?

- Give protection from and control over latency and failure from dependencies accessed (typically over the network) via third-party client libraries.
- Stop cascading failures in a complex distributed system.
- Fail fast and rapidly recover.
- Fallback and gracefully degrade when possible.
- Enable near real-time monitoring, alerting, and operational control.

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’s Hystrix

- Netflix’s Hystrix library provides an implementation of the Circuit Breaker pattern: when we apply a circuit breaker to a method, Hystrix watches for failing calls to that method, and if failures build up to a threshold, Hystrix opens the circuit so that subsequent calls automatically fail. While the circuit is open, Hystrix redirects calls to the method, and they’re passed on to our specified fallback method.

Lab - Hystrix

- Hytrix dependency
- @HystrixCommand(fallbackMethod = "reliable") public String readingList() {
- @EnableCircuitBreaker @RestController @SpringBootApplication public class ReadingApplication

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

Lab – Feign Client

- Write a feign client to test reqres.in
  - Invoke get request
  - RequestHeader

fands™
A Fast AND Steady Approach

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

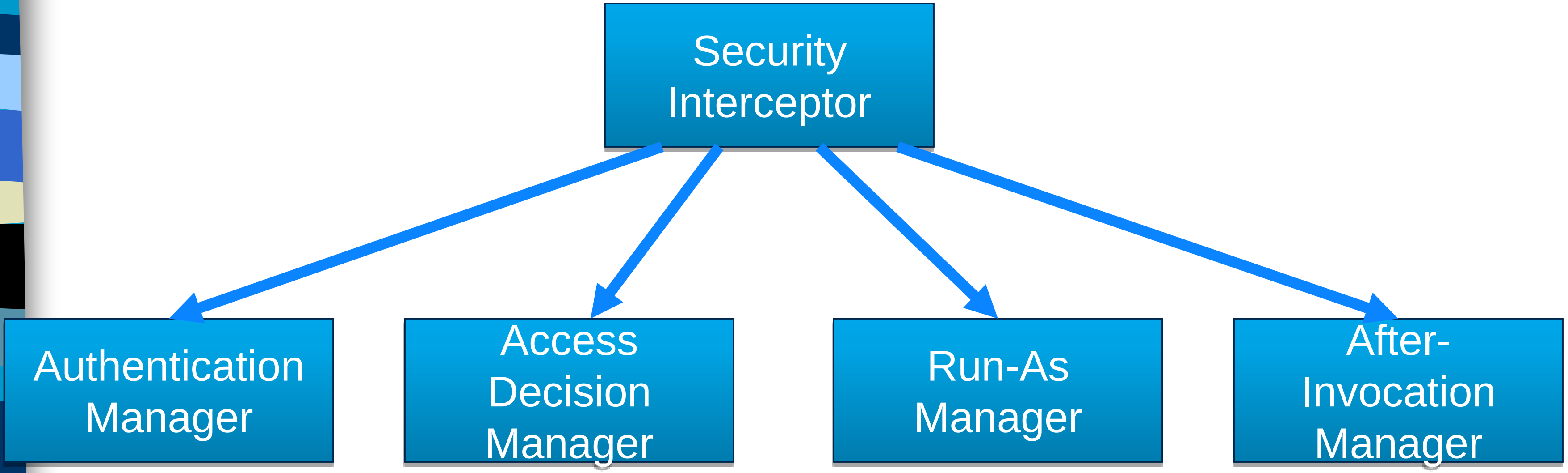
Acegi Security

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

Key concepts

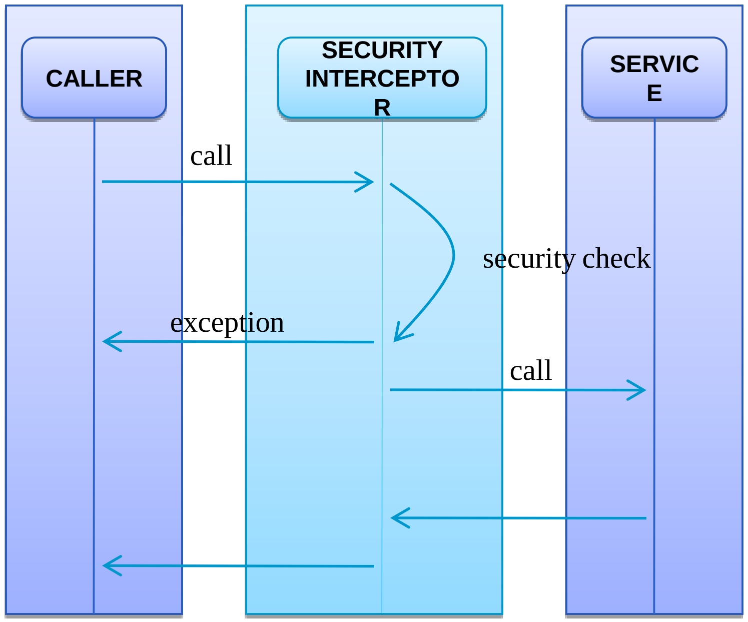
- Filters (Security Interceptor)
- Authentication
- Authorization
- Web authorization
- Method authorization

Fundamental Elements

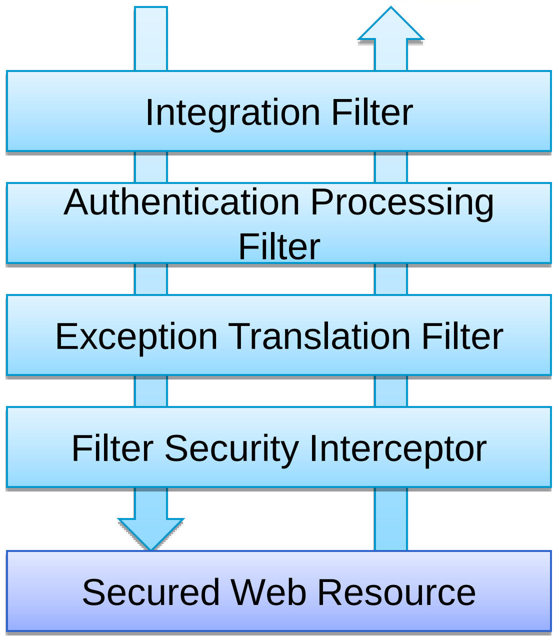


Security Interceptor

- 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


Integration Filter
Authentication Processing Filter
Exception Translation Filter
Filter Security Interceptor
Secured Web Resource

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

Secure Application

```
<sec:authentication-manager>
    <sec:authentication-provider>
        <sec:user-service>
            <sec:user name="user1" password="abc" authorities="ROLE_admin" />
            <sec:user name="user2" password="abc" authorities="ROLE_stduser" />
            <sec:user name="user3" password="abc" authorities="ROLE_admin" />
        </sec:user-service>
    </sec:authentication-provider>
</sec:authentication-manager>
<sec:global-method-security secured-annotations="enabled" />
```

Secure Application

```
<bean id="hello" class="demo.HelloWorld"
    scope="singleton">
        <sec:intercept-methods>
                <sec:protect access="ROLE_admin"
    method="method2" />
                <sec:protect access="ROLE_stduser"
    method="method1" />
        </sec:intercept-methods>
    </bean>
```

Pass Security Details

```
SecurityContextImpl scimpl = new
    SecurityContextImpl();
Authentication auth = new
    UsernamePasswordAuthenticationToken("user1",
    "abc");
scimpl.setAuthentication(auth);
SecurityContextHolder.setContext(scimpl);
hello.method1();
```

Security with tables

```
<sec:global-method-security secured-annotations="enabled" />

<sec:authentication-manager>
    <sec:authentication-provider>
        <sec:jdbc-user-service   data-source-ref='myds'
    users-by-username-query="select username,password, 'true' as
    enabled from users where username=?"
    authorities-by-username-query=
    'select users.username , roles.role as role from users, roles
        where users.username = ? AND
    roles.username=users.username ' />
    </sec:authentication-provider>
</sec:authentication-manager>
```

Annotations

```
@Secured("ROLE_stduser")
   public void method1(){
         System.out.println("method1 called");
         }
```

SAML

- SAML is an open standard for exchanging authentication and authorization data between parties, in particular, between an identity provider and a service provider. As its name implies, SAML is an XML-based markup language for security assertions (statements that service providers use to make access-control decisions). SAML is also:
  - A set of XML-based protocol messages
  - A set of protocol message bindings
  - A set of profiles (utilizing all of the above)

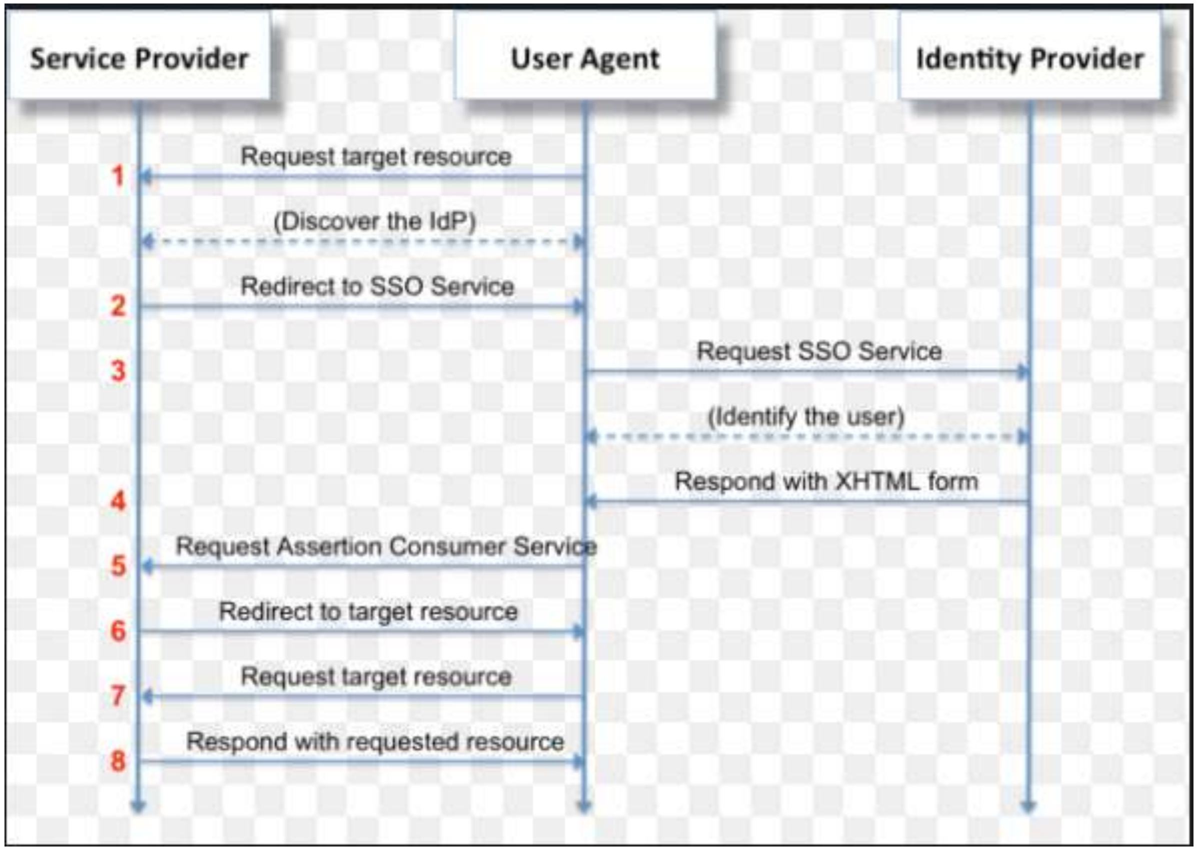
OAuth

- 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)

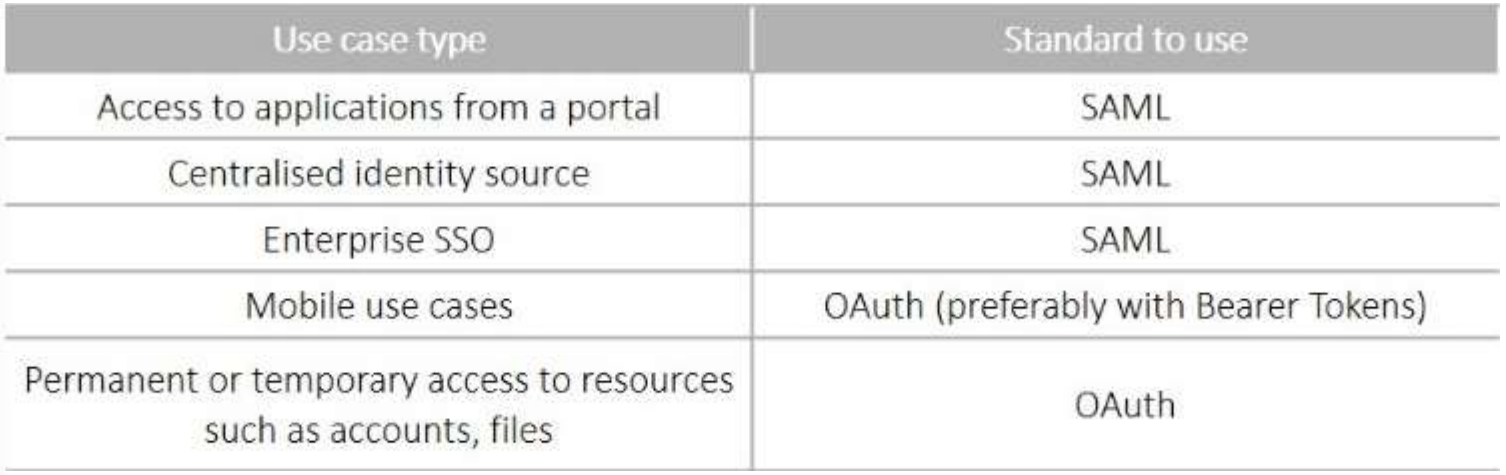


Service Provider
User Agent
Identity Provider
1
Request target resource
(Discover the IdP)
2
Redirect to SSO Service
3
Request SSO Service
(Identify the user)
4
Respond with XHTML form
5
Request Assertion Consumer Service
6
Redirect to target resource
7
Request target resource
8
Respond with requested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

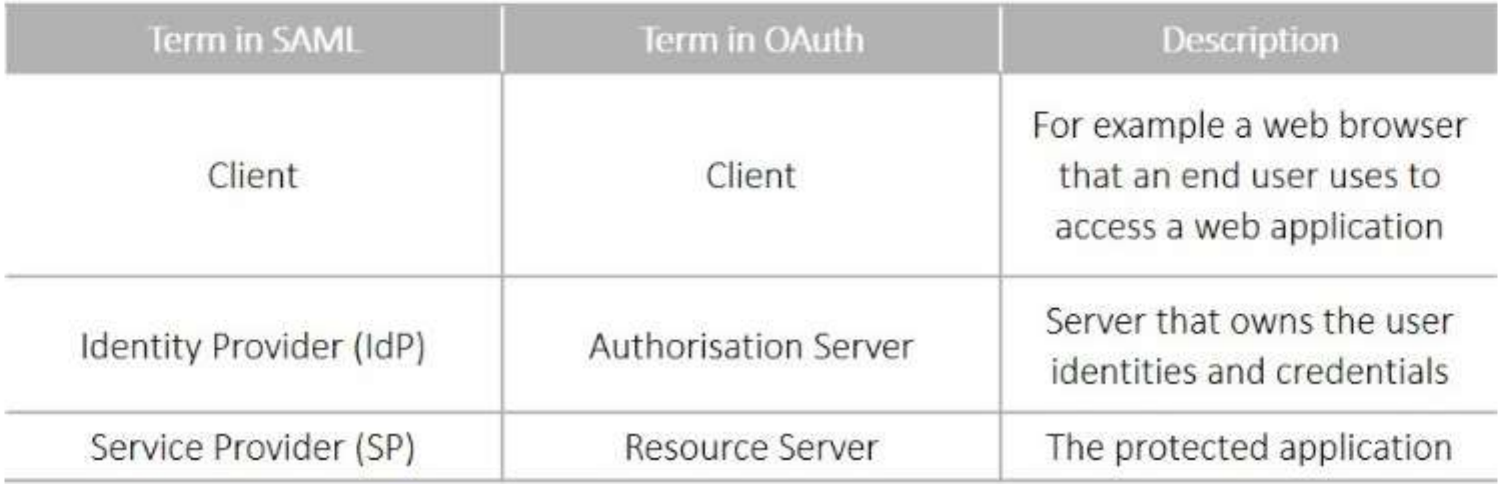
SAML & OAuth

- 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



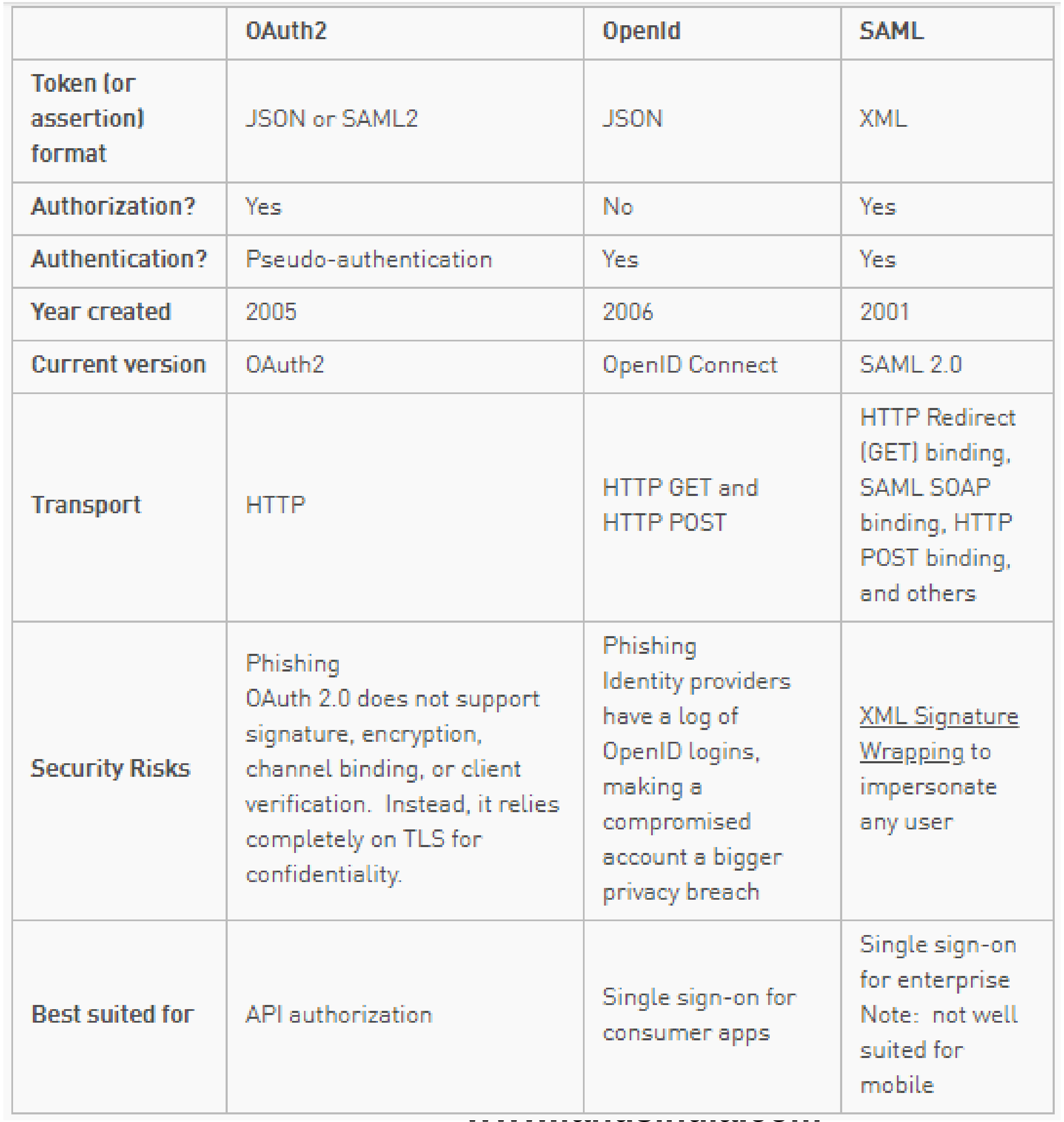
Use case type	Standard to use
Access to applications from a portal	SAML
Centralised identity source	SAML
Enterprise SSO	SAML
Mobile use cases	OAuth (preferably with Bearer Tokens)
Permanent or temporary access to resources such as accounts, files	OAuth



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 OAuth 2.0 does not support signature, encryption, channel binding, or client verification. Instead, it relies completely on TLS for confidentiality.	Phishing 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 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/

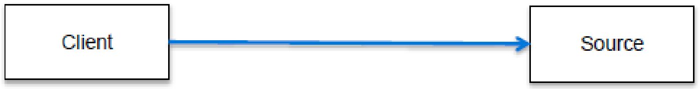
Lab

- Create a client to connect GitHub
- Write a Auth Server and modify client to connect to our Auth Server

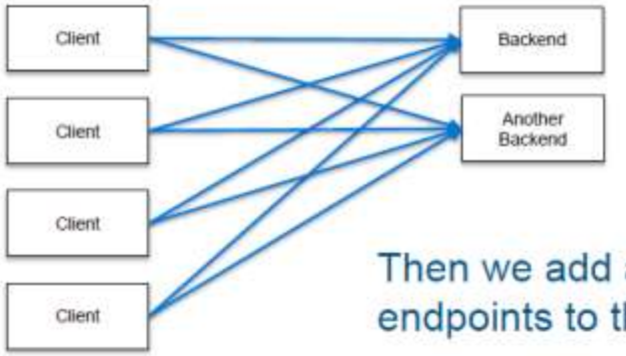
fands™
A Fast AND Steady Approach

Kafka

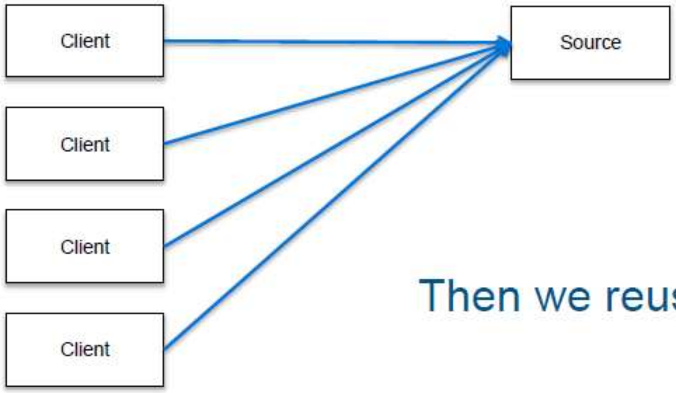
Why Kafka?



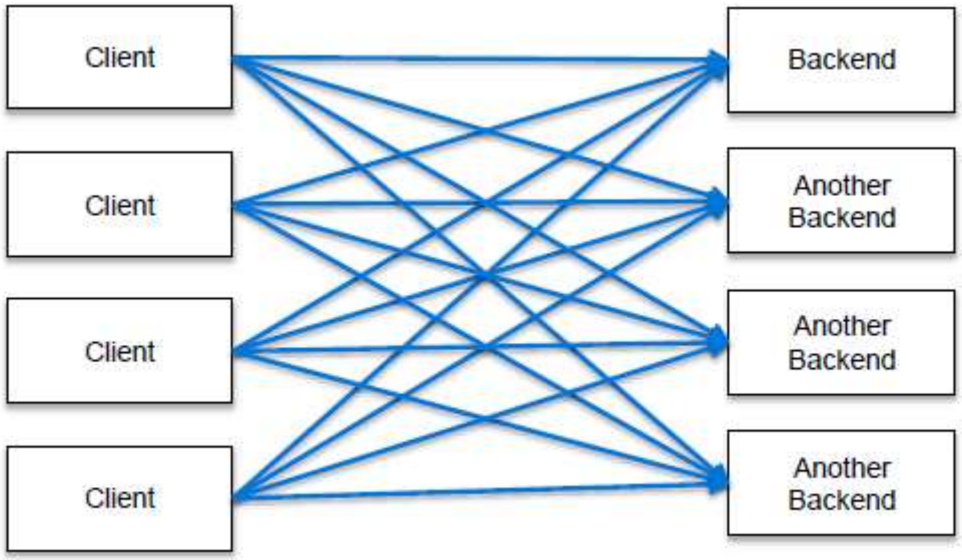
Data Pipelines Start like this.



Then we add additional endpoints to the existing sources

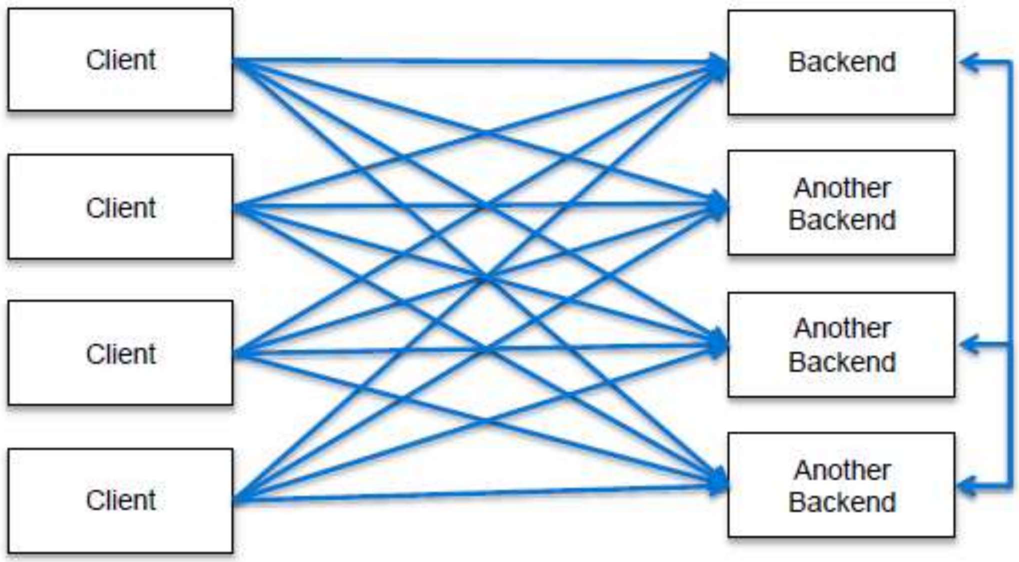


Then we reuse them



Then it starts to look like this

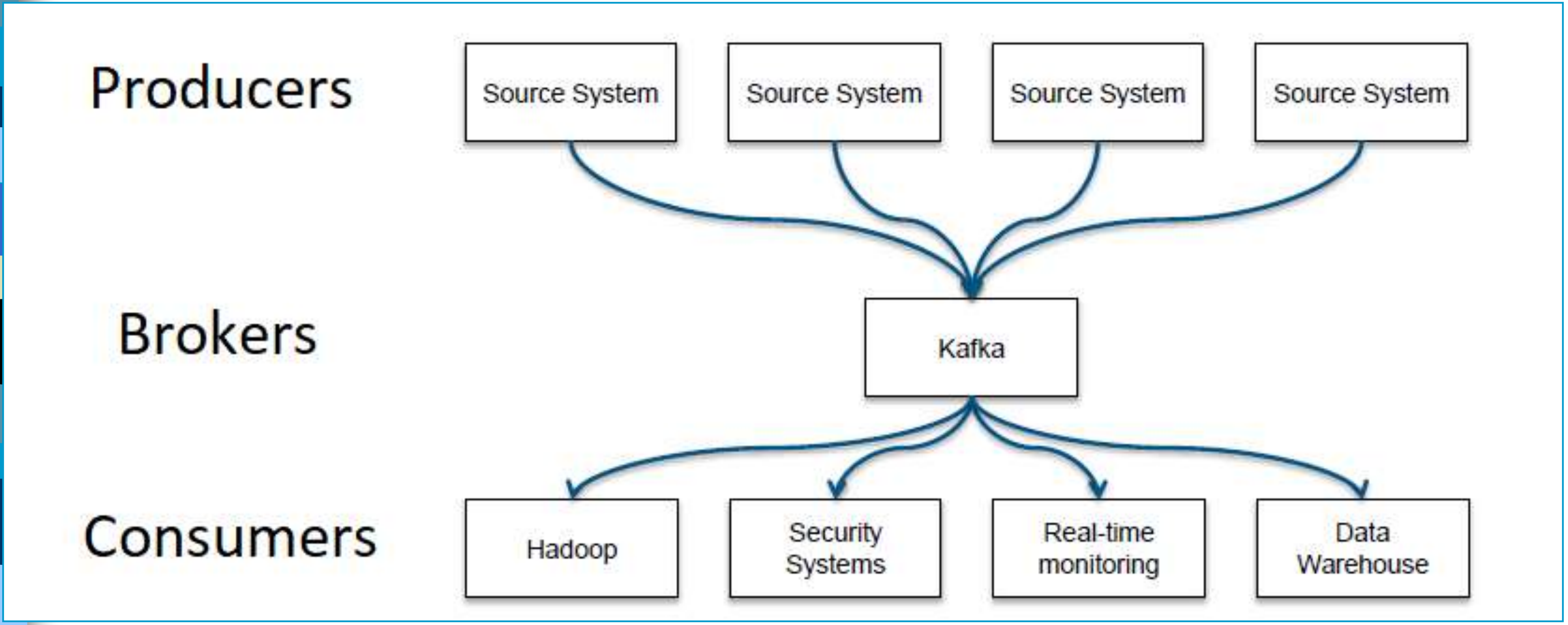
Why Kafka?



With maybe some of this

As distributed systems and services increasingly become part of a modern architecture, this makes for a fragile system

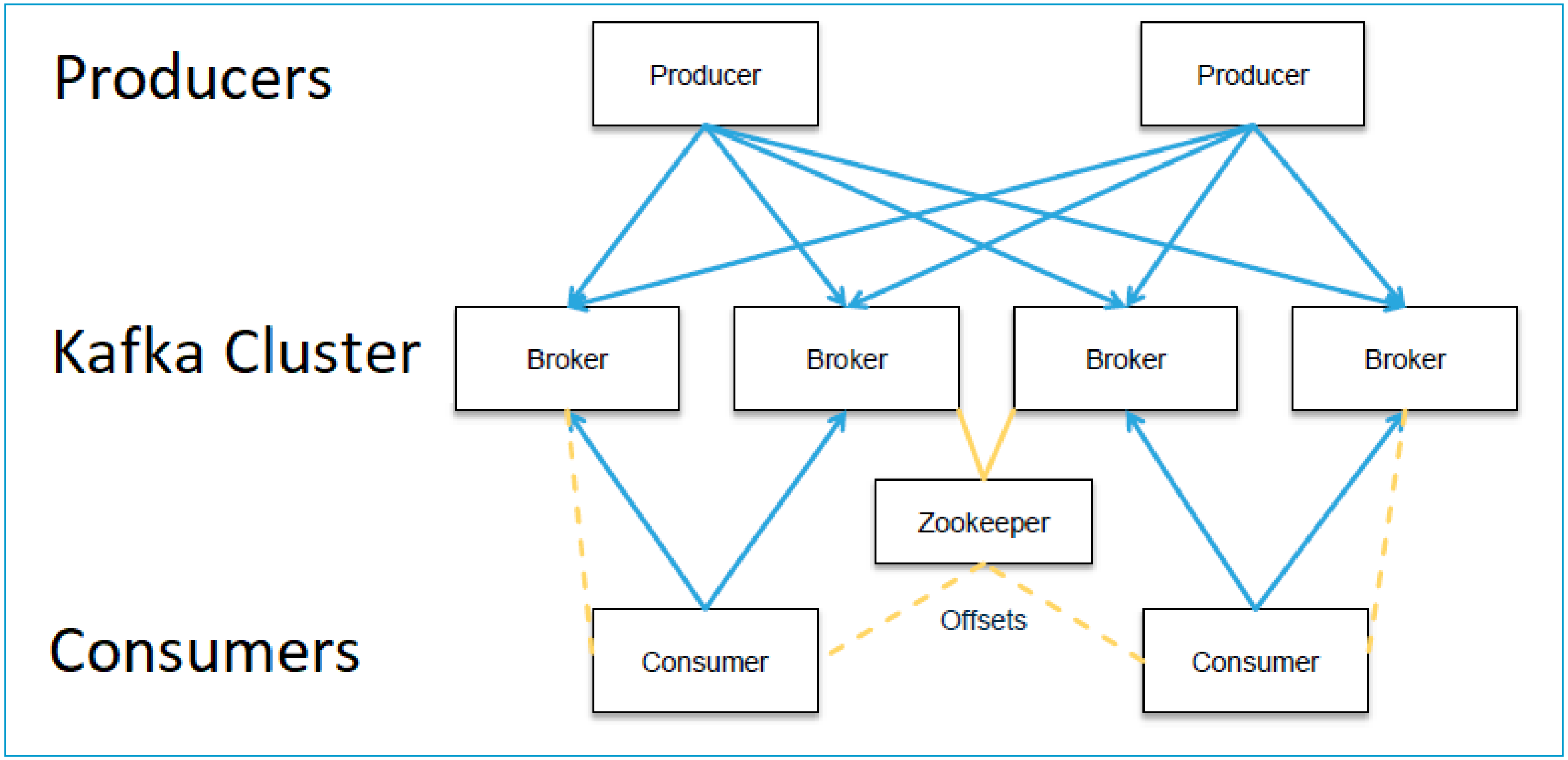
Kafka Decouples Data Pipelines



Basic Terminology

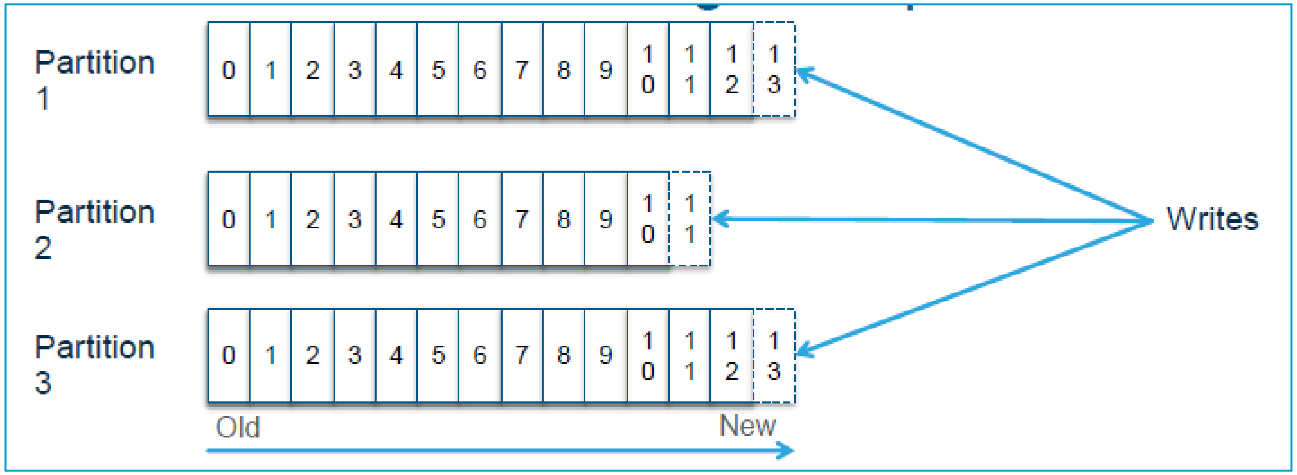
- Kafka maintains feeds of messages in categories called topics.
- Processes that publish messages to a Kafka topic are called producers.
- Processes that subscribe to topics and process the feed of published messages are called consumers.
- Kafka is run as a cluster comprised of one or more servers each of which is called a broker.
- Communication between all components is done via a high performance simple binary API over TCP protocol

Architecture



Topics - Partitions

- Topics are broken up into ordered commit logs called partitions.
  - Each message in a partition is assigned a sequential id called an offset.
  - Data is retained for a configurable period of time*



Topics

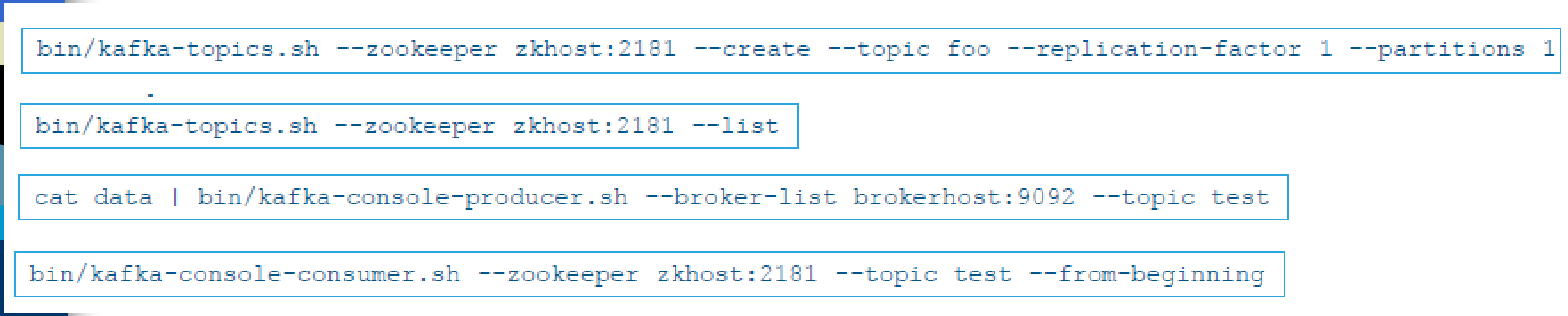
- Message Ordering
- Gurantees
- Replication
- Durable Writes

Should I use Kafka ?

- For really large file transfers?
- Probably not, it’s designed for “messages” not really for files.
- As a replacement for MQ/Rabbit/Tibco
  - Probably. Performance Numbers are drastically superior. Also gives the ability for transient consumers. Handles failures pretty well.
- To do transformations of data?
  - • Not really by itself

Lab

- Installation of Kafka
- Kafka Operations



```
bin/kafka-topics.sh --zookeeper zkhost:2181 --create --topic foo --replication-factor 1 --partitions 1
```

```
bin/kafka-topics.sh --zookeeper zkhost:2181 --list
```

```
cat data | bin/kafka-console-producer.sh --broker-list brokerhost:9092 --topic test
```

```
bin/kafka-console-consumer.sh --zookeeper zkhost:2181 --topic test --from-beginning
```

Lab

- Spring Cloud Stream with Kafka

fands™
A Fast AND Steady Approach

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

Background 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?

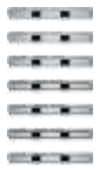
of hardware environmen ts

Development VM

QA server

Public Cloud

Production Cluster



Disaster recovery

Customer Data Center

Contributor's laptop

Production Servers

Can I migrate smoothly and quickly?

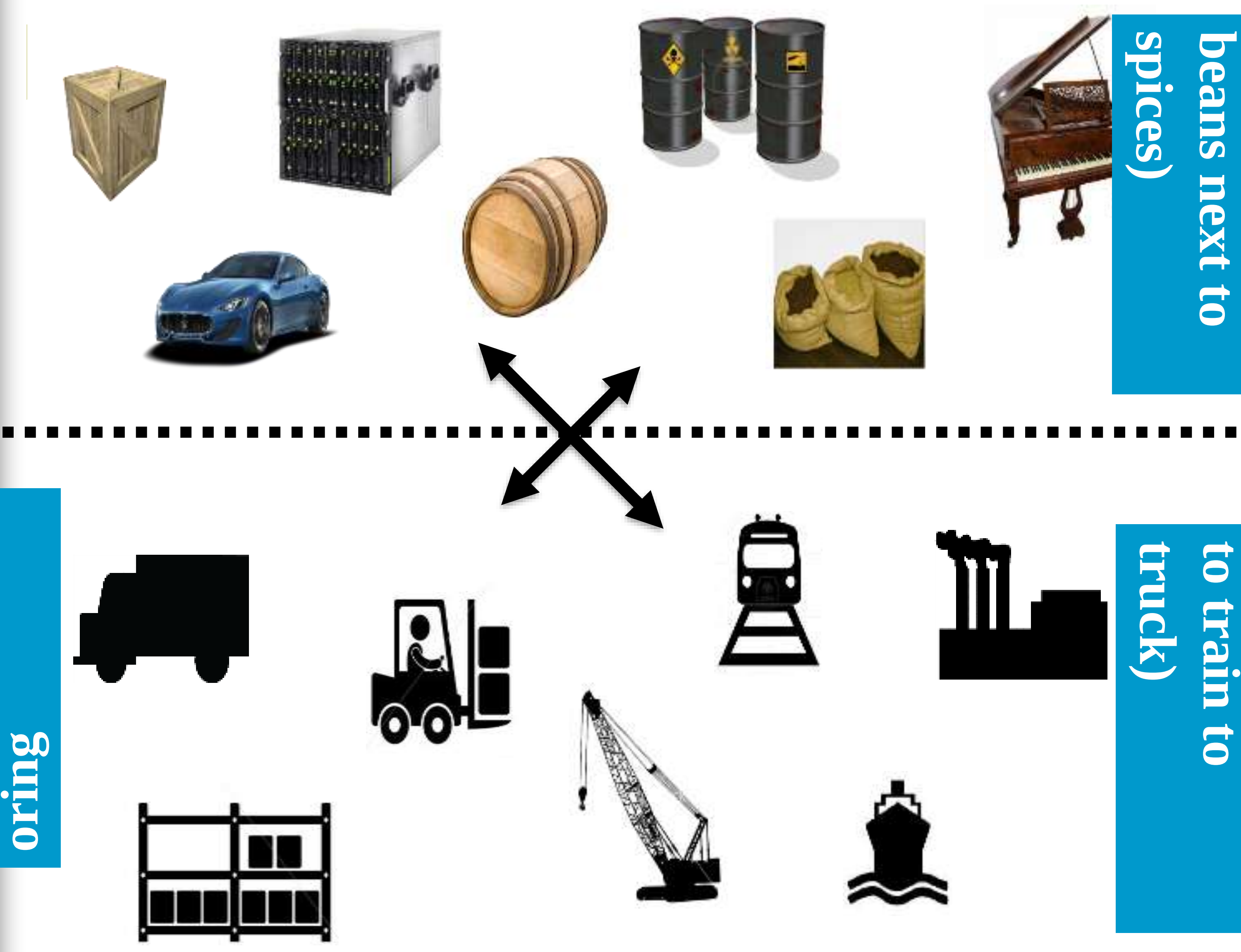
The Matrix From Hell

	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

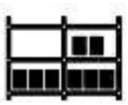
about how goods interact (e.g. coffee beans next to spices)



methods for transporting/st oring

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

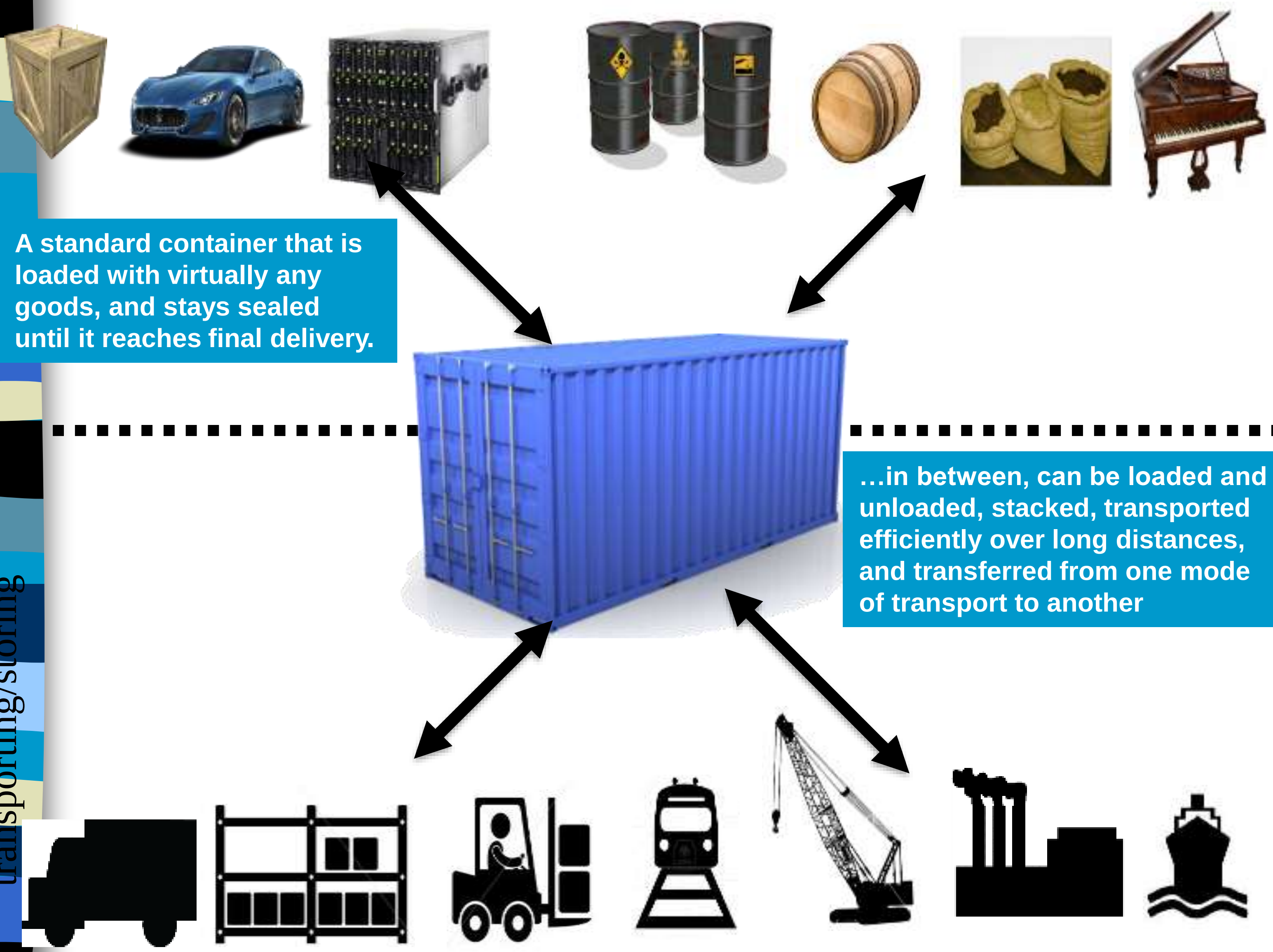
Matrix Management

	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

Solution: Intermodal Shipping Container

Multiplicity of Goods

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



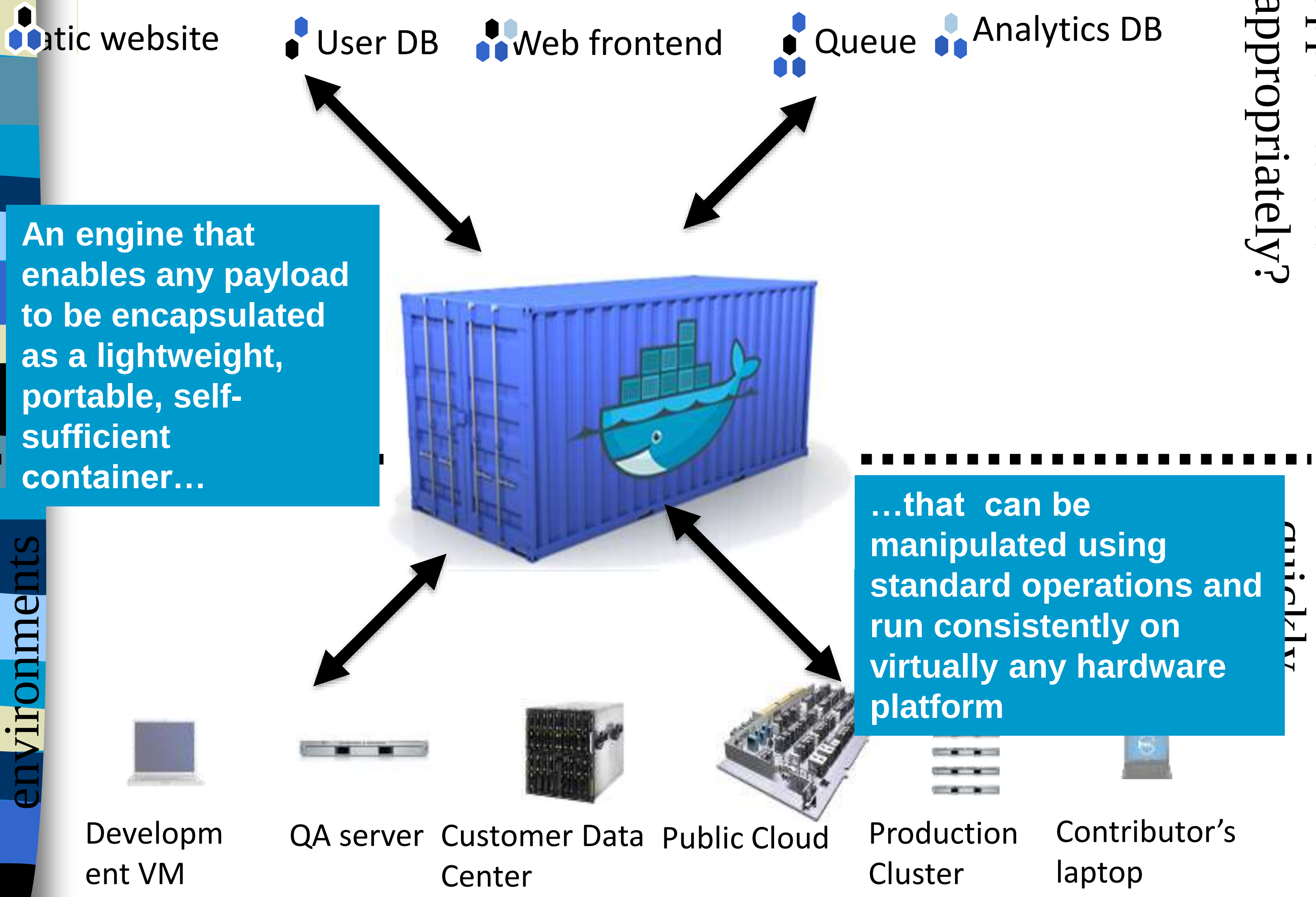
A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.

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

Multiplicity of methods for transporting/storing

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

Docker is a shipping container system for code



An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container…

…that can be manipulated using standard operations and run consistently on virtually any hardware platform

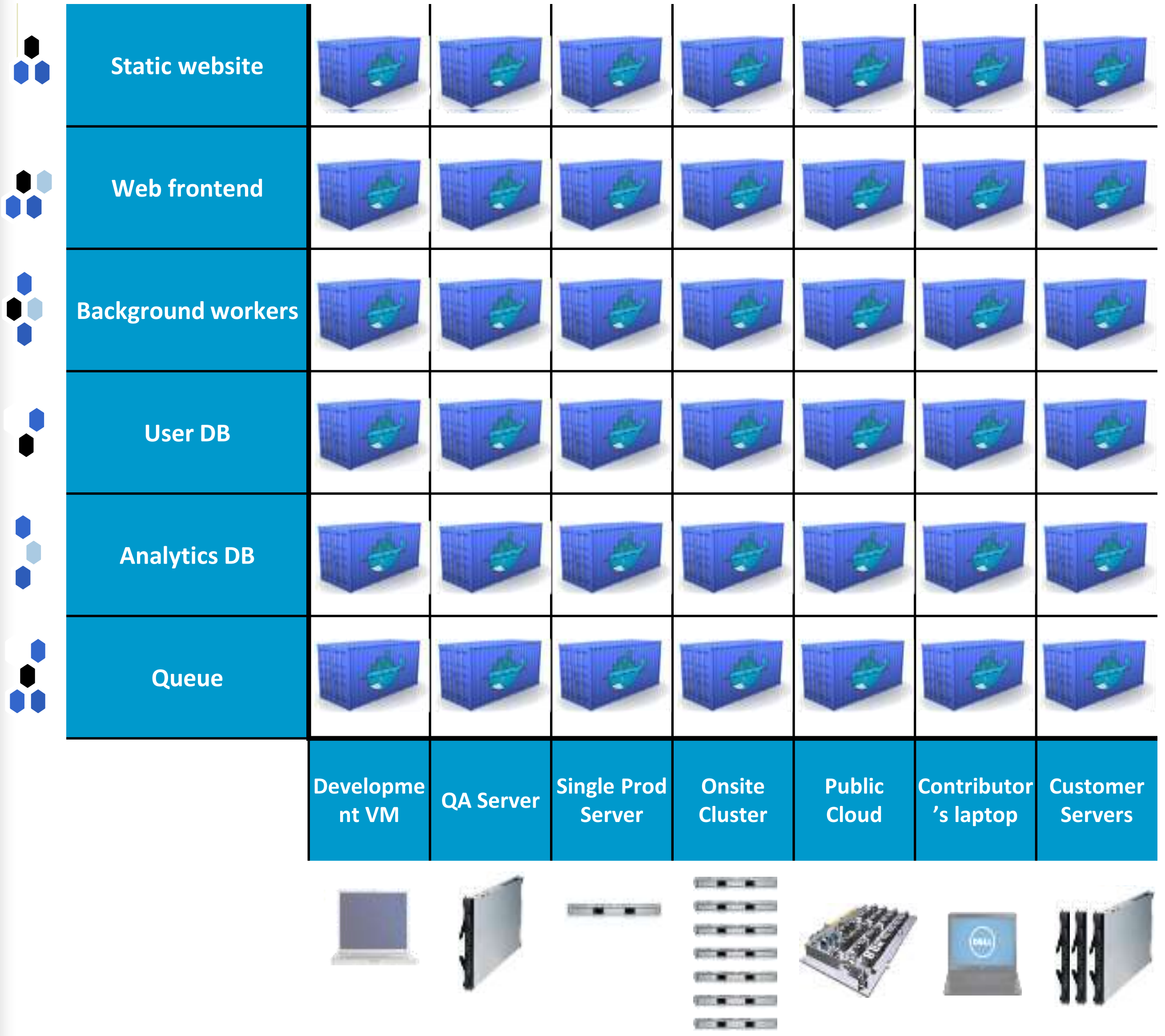
Multiplicity of Stacks

Multiplicity of hardware environments

Do services and apps interact appropriately?

Can I migrate smoothly and quickly

Docker eliminates the matrix management



fands™
A Fast AND Steady Approach

Introduction

What is Docker?

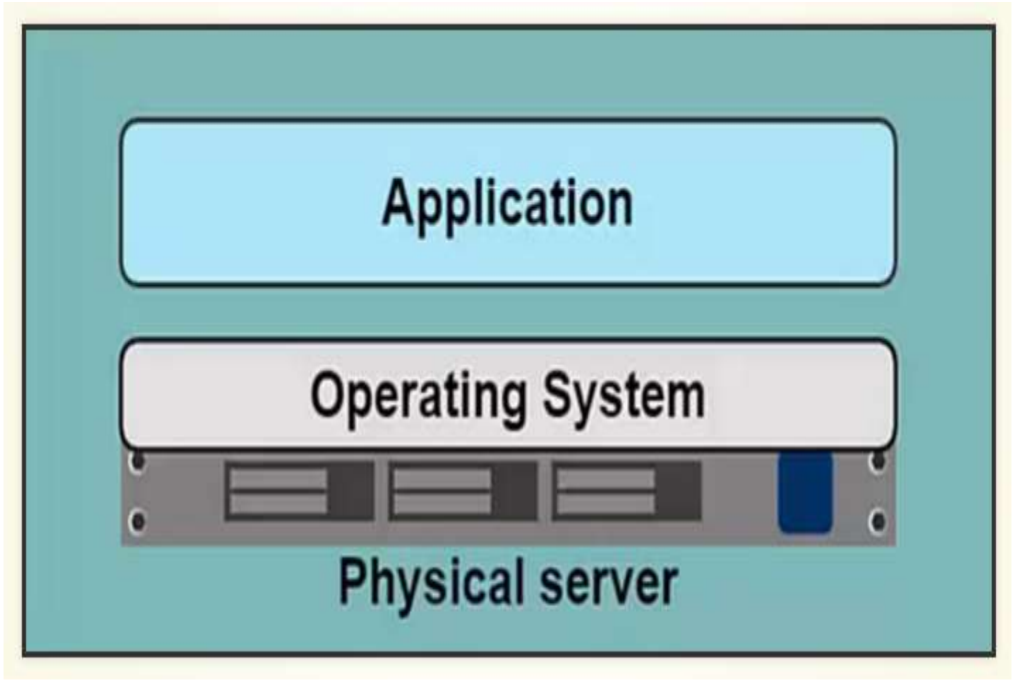
- Docker is a platform for developing, shipping and running applications using container virtualization technology.
- The Docker Platform consists of multiple tools.
  - Docker Engine
  - Docker Hub
  - Docker Machine
  - Docker Swarm
  - Docker Compose
  - Kitematic

fands™

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

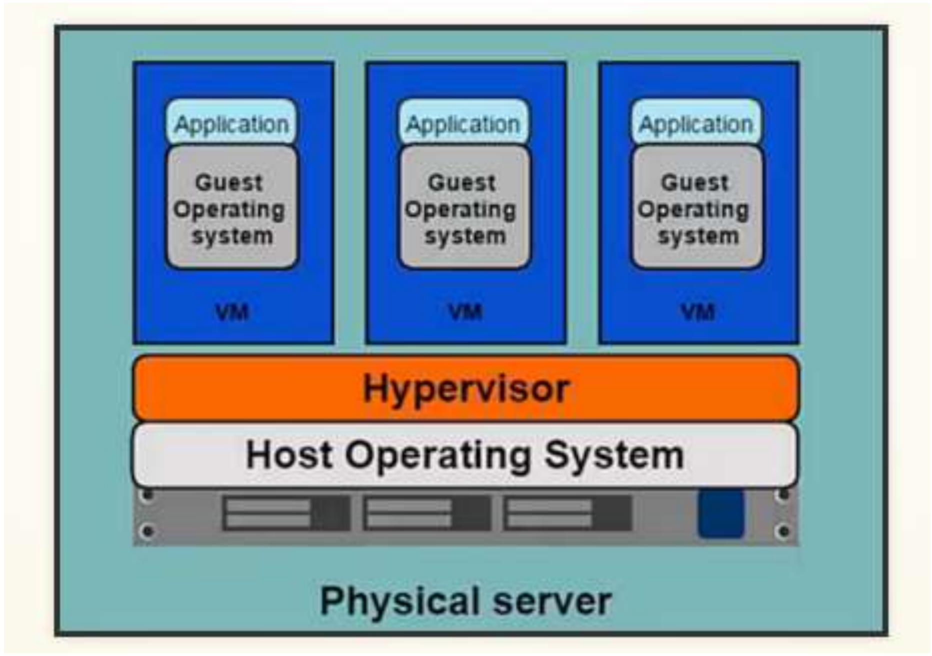
Basic Application Hosting



- 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

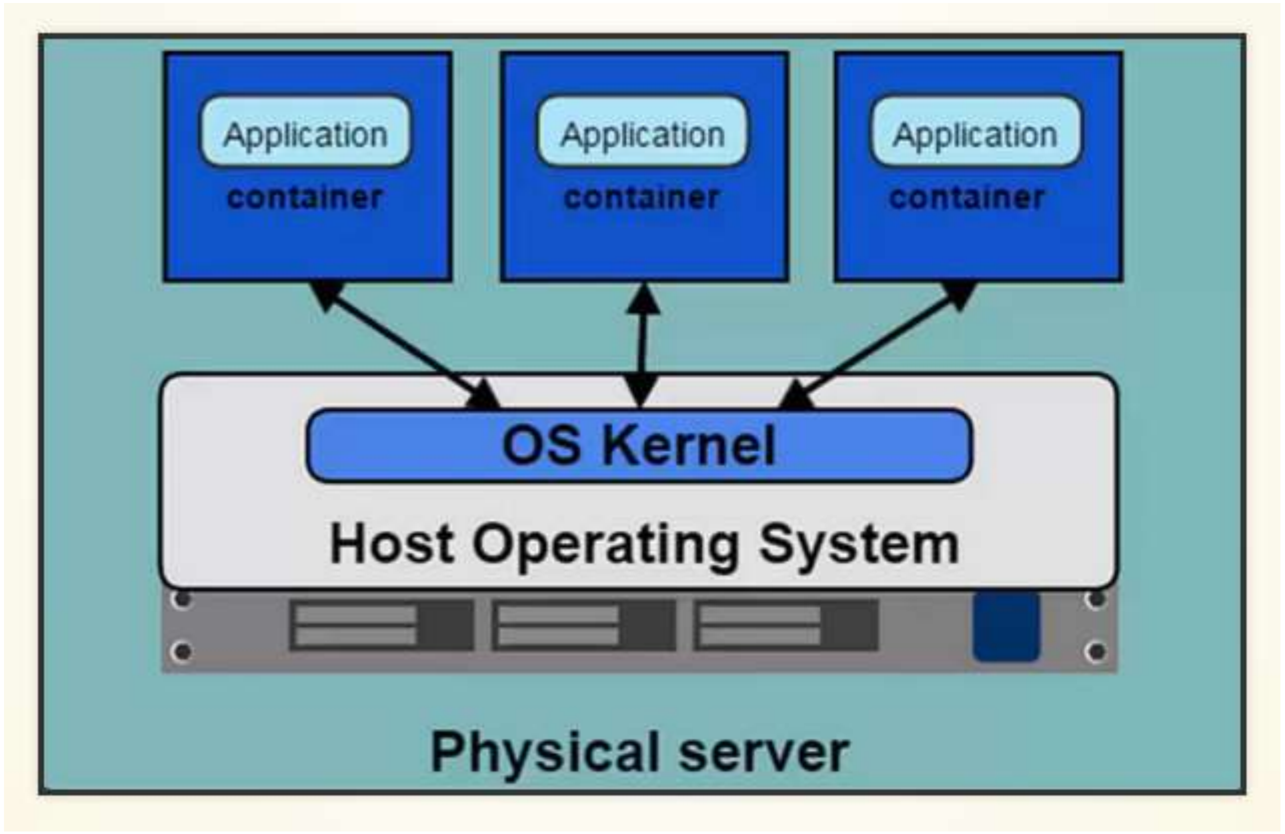
- Each VM stills requires
  - CPU allocation
  - storage
  - RAM
  - An entire guest operation system
- More VM's you run, the more resources you need
- Guest OS means wasted resources

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

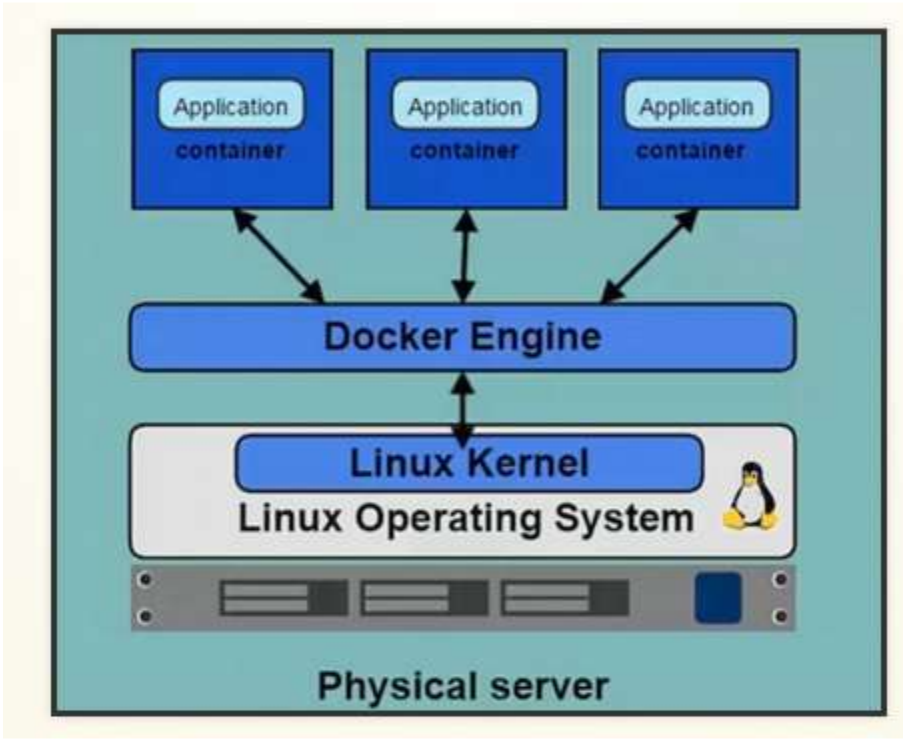

Application
container
Application
container
Application
container
OS Kernel
Host Operating System
Physical server

Containers Vs VMs

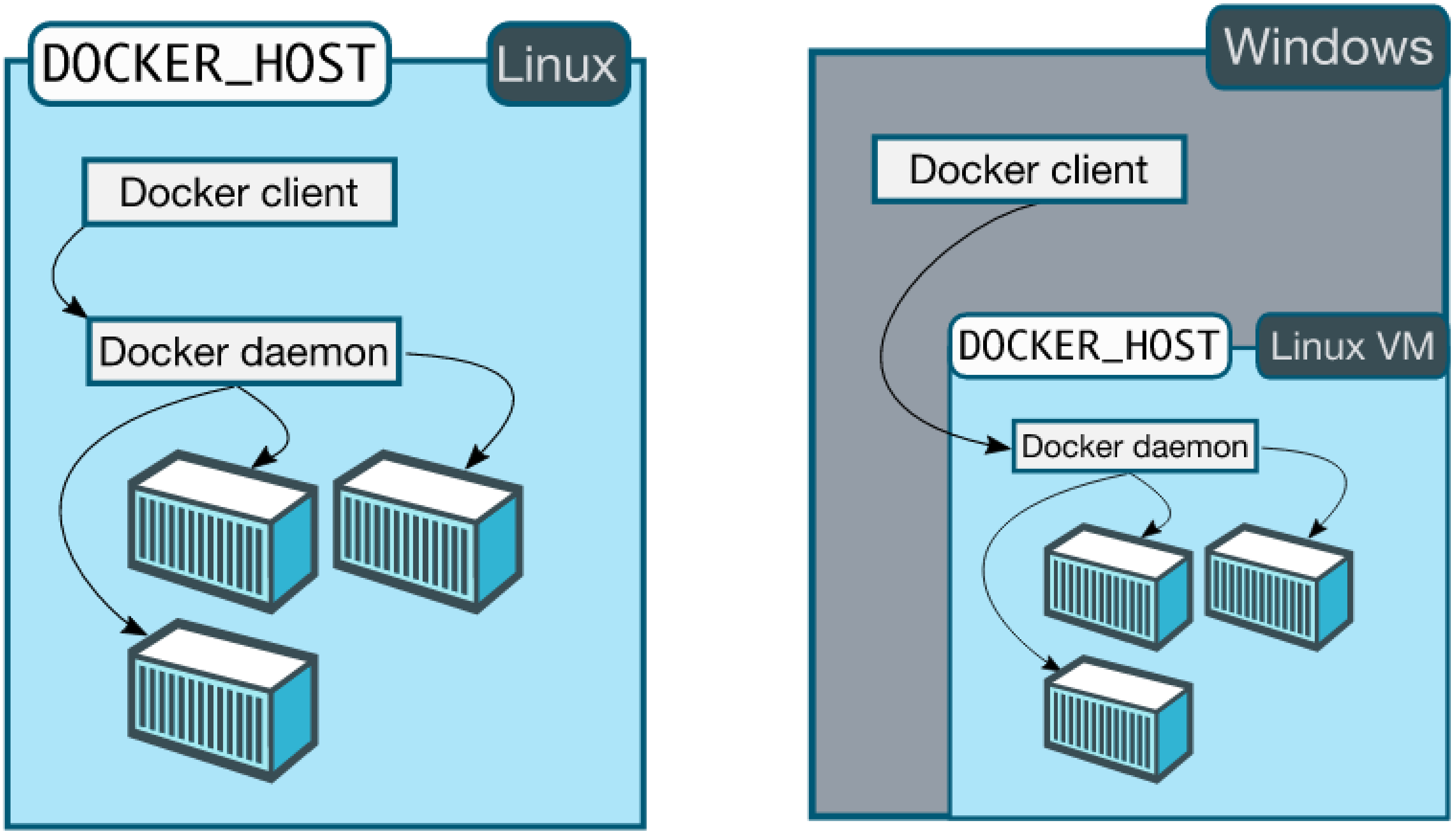
- Containers are more lightweight
- No need to install guest OS
- Less CPU, RAM, storage space required
- More containers per machine than VMs
- Greater portability

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



Docker Engine on Different OS

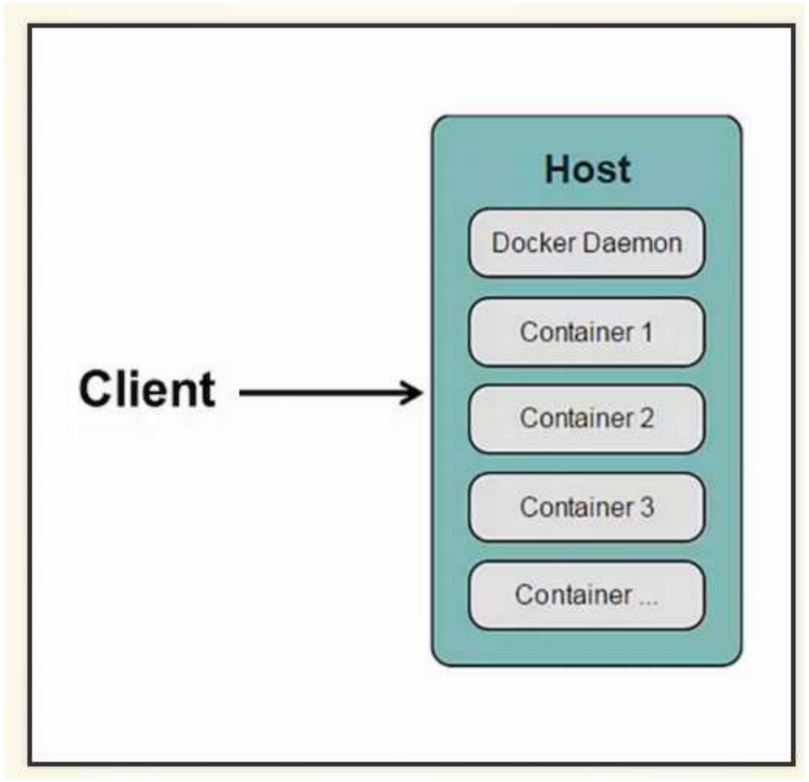


Installation

- WINDOWS AND MAC OS X:
  - Docker need linux kernel... so you will need a linux Virtual Machine. Docker toolbox contains everything needed.
- VERIFY DOCKER INSTALLATION
  - $ docker run hello-world
    - This command will download(if required) and run an hello world container

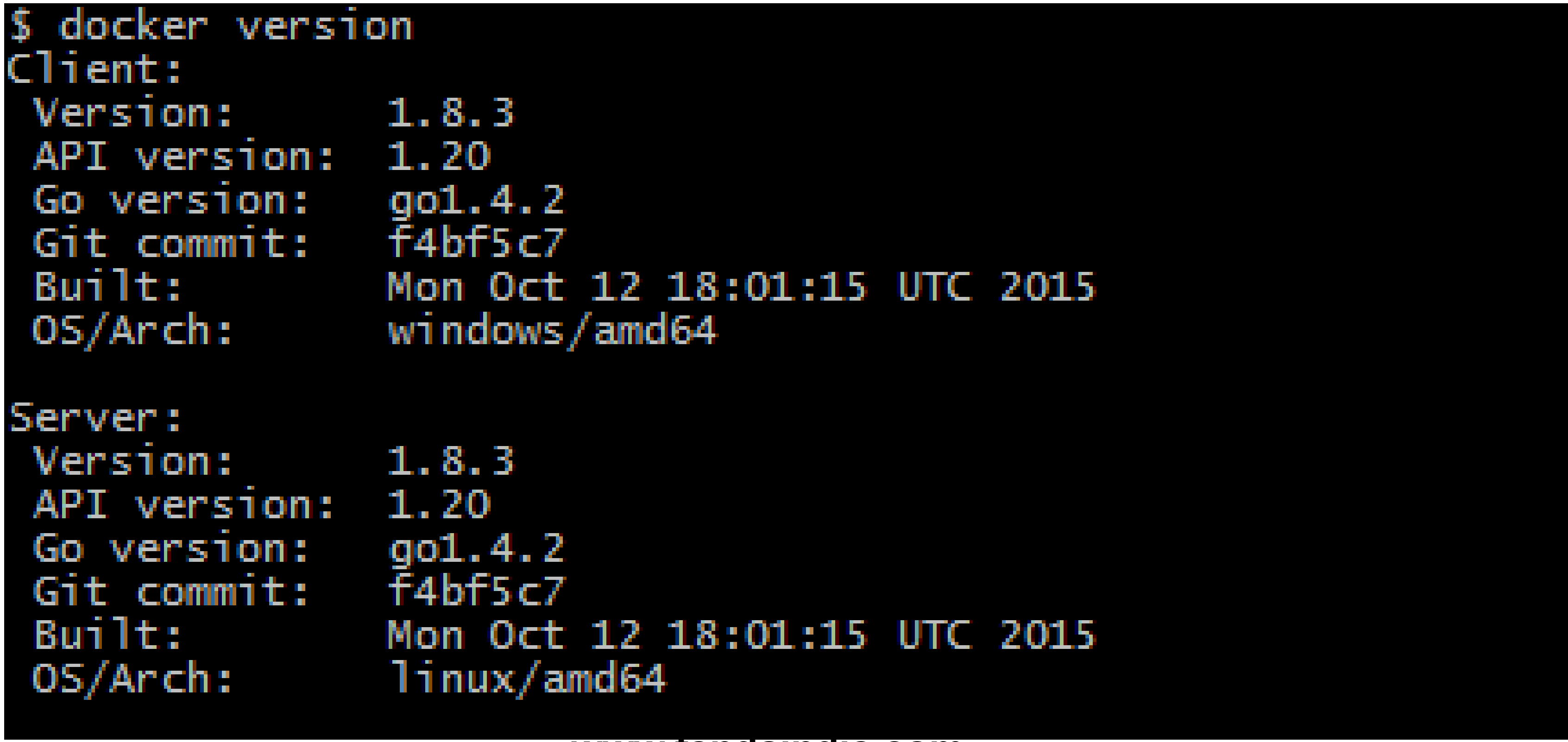
Docker Client and Daemon

- Client / Server Architecture
- Client takes user inputs and send them to the daemon
- Daemon builds, runs, and distributes containers
- Client and daemon can run on same or different hosts



Checking Client and Daemon Version

- $ docker version



```
$ docker version
Client:
 Version:      1.8.3
 API version:  1.20
 Go version:   go1.4.2
 Git commit:   f4bf5c7
 Built:        Mon Oct 12 18:01:15 UTC 2015
 OS/Arch:      windows/amd64

Server:
 Version:      1.8.3
 API version:  1.20
 Go version:   go1.4.2
 Git commit:   f4bf5c7
 Built:        Mon Oct 12 18:01:15 UTC 2015
 OS/Arch:      linux/amd64
```

Lab

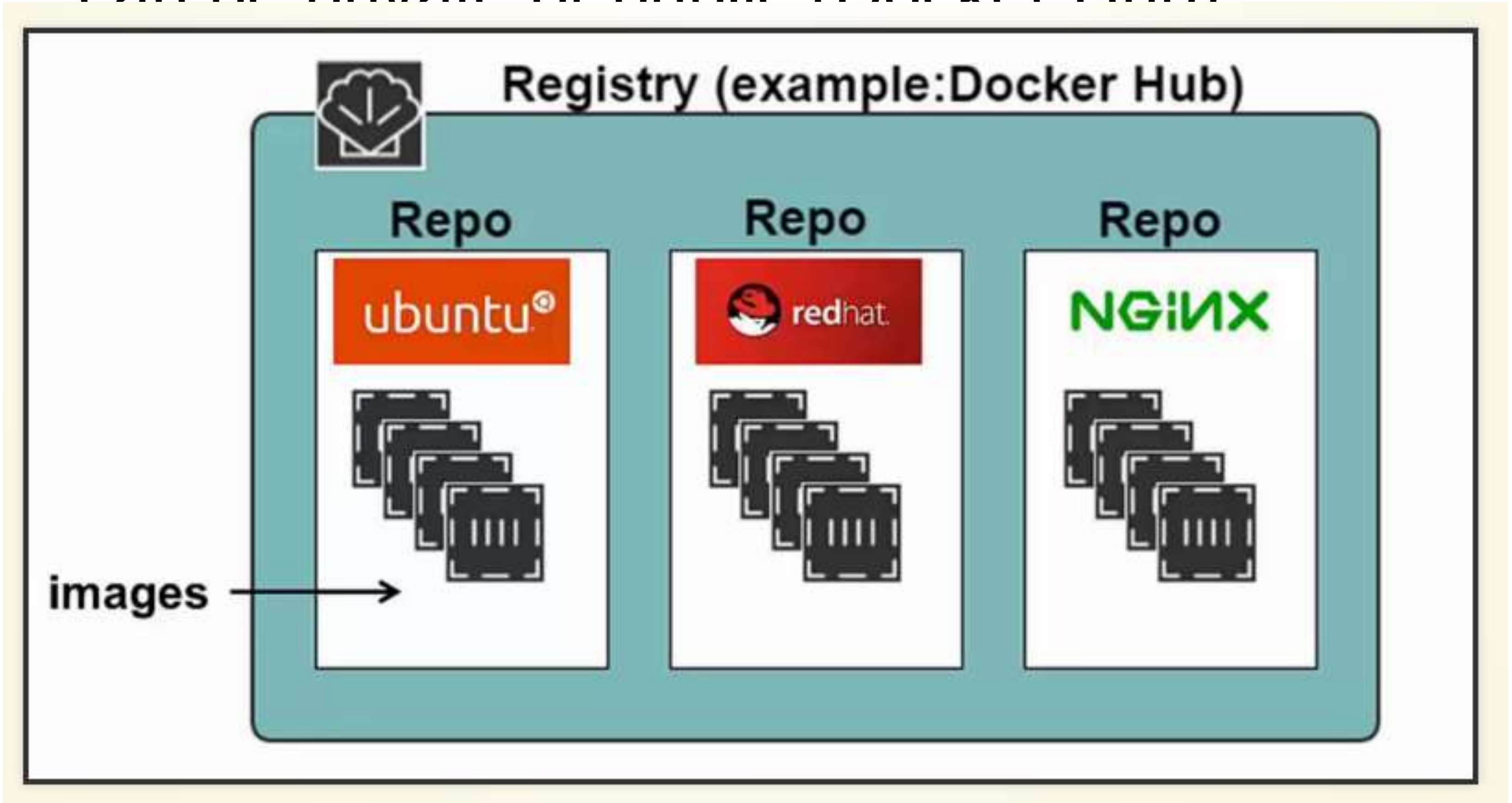
- Install Docker
- Check Docker version
- Connect to docker hub and work with hello world
- Check first hello-world image and container both

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

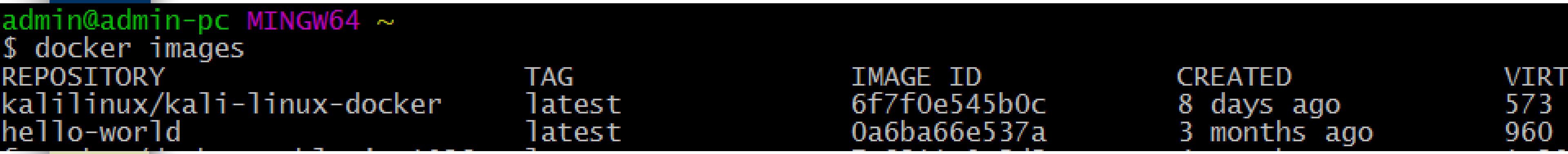
fands™

A Fast AND Steady Approach

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

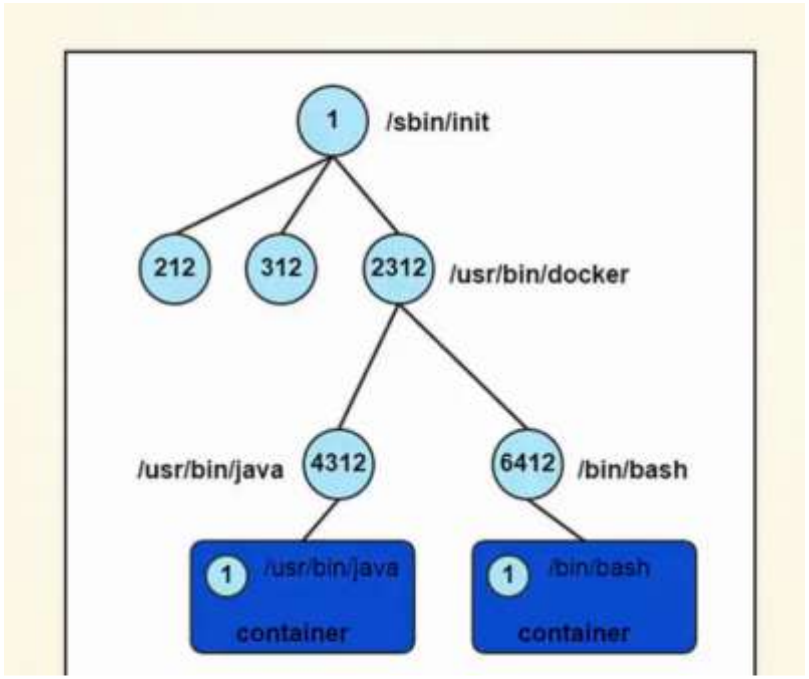
- Using docker run
  - Syntax:
  - $ docker run [options] [image] [command] [args]
    - image is specified with repository:tag
- examples:
  - docker run ubuntu:14.04 echo “HelloWorld”

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

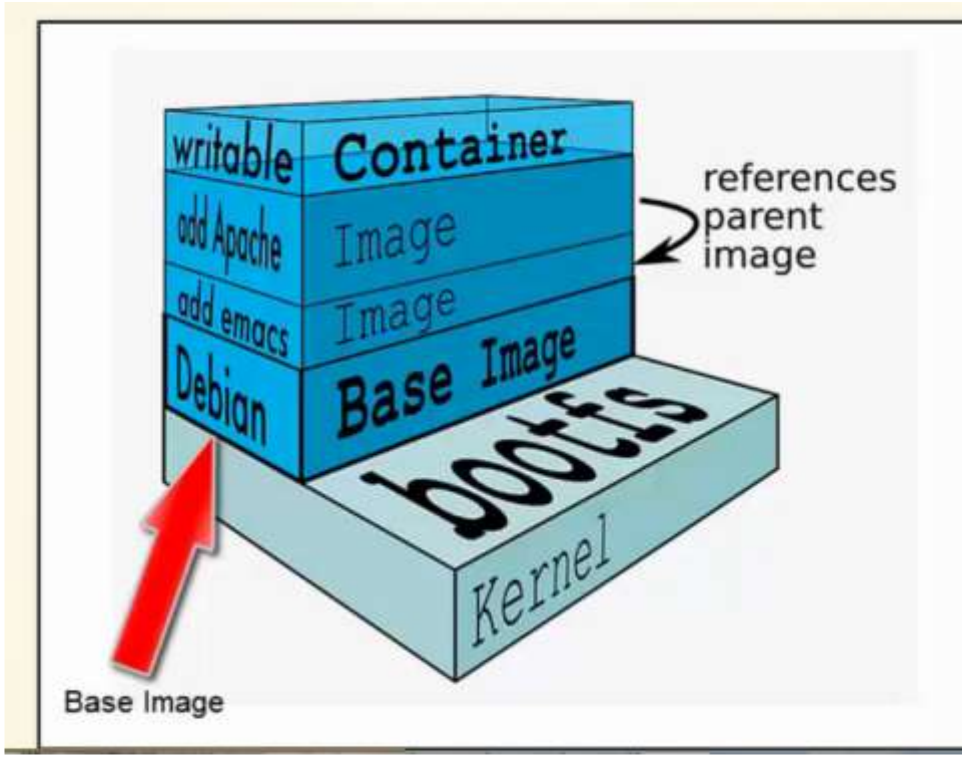
- Run a web app inside a container
- The –p flag to map container ports to host ports

fands™
A Fast AND Steady Approach

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

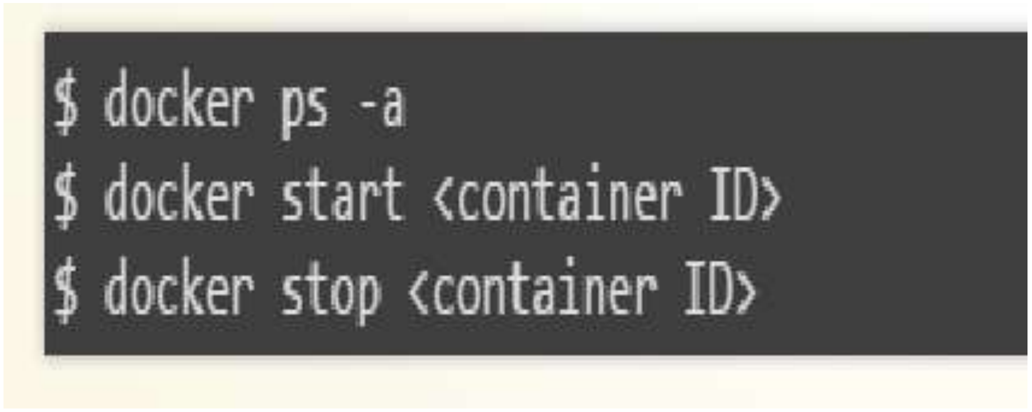
fands™

A Fast AND Steady Approach

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

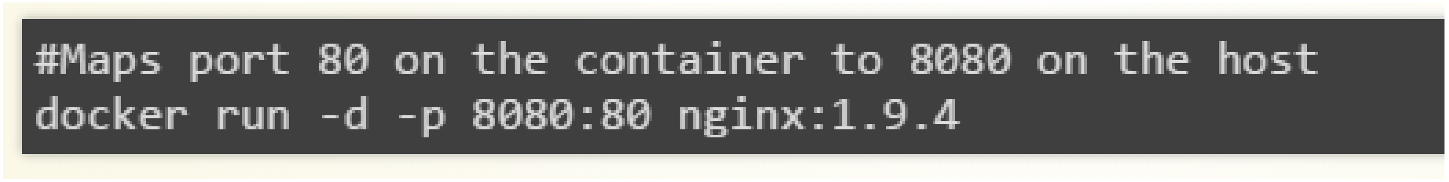
- Containers that have been stopped
  - docker rm containerid
- Images
  - docker rmi [image ID]
    or
    docker rmi [repo:tag]

A Fast AND Steady Approach

Container Networking Basics

Mapping Ports

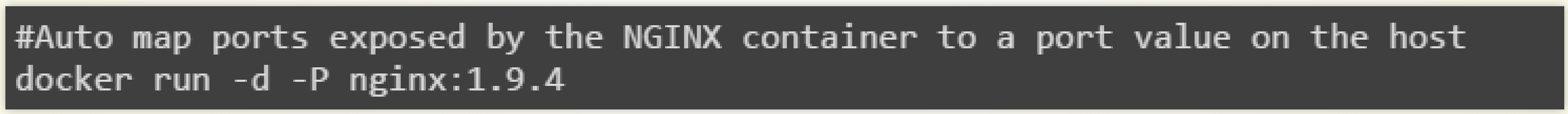
- **Recall:** containers have their own network and IP address
- Map exposed container ports to ports on the host machine
- Ports can be manually mapped or auto mapped
- Uses the -p and -P parameters in docker run



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

Automapping Ports

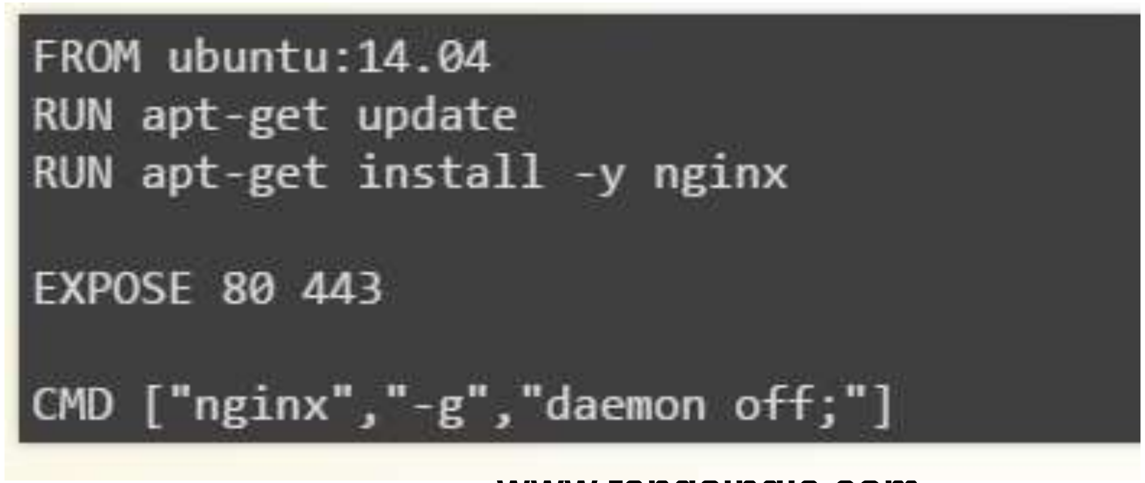
- Use the -P option in **docker run**
- Automatically maps exposed ports in the container to a port number in the host
- Host port numbers used go from 49153 to 65535
- Only work for ports defined in the EXPOSE instruction



```
#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

fands™
A Fast AND Steady Approach

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"]
```

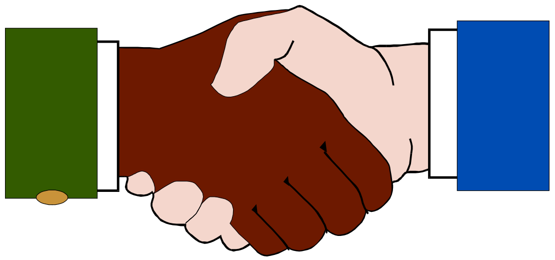
- Create a simple docker file to launch a Spring web application
- Create image from Dockerfile
  - docker build -t imageName .
- Create a container from image
  - Docker run ..
- Check from external browser

fands™

QUESTION / ANSWERS

www.fandsindia.com

THANKING YOU !



www.fandsindia.com