Advanced topics

1. Messaging systems

Intro

- Messaging provides a mechanism for loosely-coupled integration of systems
- The central unit of processing in a message is a message which typically contains a **body** and a **header**
- Use cases include:
  - Log aggregation between systems
  - Event propagation between systems - някакви събития се fire-ват
  - Offloading log-running tasks to worker nodes - the result of the task then to be sent to a third systems for example
- Messaging solutions implement different protocols for transferring of messages such as **AMQP** (binary protocol), XMPP, MQTT and many more like XML, JSON, etc.
- The variety of protocols implies vendor lock-in when using a particular messaging solution (also called a messaging broker) - ако е специфичен протокола лошо. Т.е. протокола е добре да е такъв, че да може да се използва от различни message broker systems

- Message brokers
  - ActiveMQ - using JMS (Java Messaging System) Java EE
  - RabbitMQ
  - Qpid
  - TIBCO
  - WebSphere MQ
  - Msmq

- Messaging solutions provide means for:
  - Securing message transfer, authenticating and authorizing messaging endpoints
  - Routing messages between endpoints
  - Subscribing to the broker

- An **enterprise service bus (ESB)** is one layer of abstraction above a messaging solution that further provides:
  - Adapters for different messaging protocols
  - Translation of messages between the different types of protocols

I. RabbitMQ

Info

- An open source message broker written in Erlang
  - Заема малко ранм памет и процесор
  - при Erlang няма context switching като при JVM
  - reliability - дете ако не се изпълни, то родителят му го пуска за изпълнение наново
- **Implements the AMQP protocol** (Advanced Message Queueing Protocol)
- Has a pluggable architecture and provides extension for other protocols such as HTTP, STOMP and MQTT

- AMQP is a binary protocol that aims to standardize middleware communication
- The AMQP protocol derives its origins form the financial industry - processing of large volumes of financial data between different systems is a classic use case of messaging
- The AMQP protocol defines:
    - **Exchanges** - the message broker endpoints that receive messages
    - **Queues** - the message broker endpoints that store messages from exchanges and are used by subscribers for retrieval of messages. The Queue can also be persistent - messages can be saved.
    - **Bindings** - rules that bind exchanges and queues
- The AMQP protocol is programmable - which means that the above entities can be created/ modified/ deleted by applications

- The AMQP protocol defines multiple connection channels inside a single TCP connection in order to remove the overhead of opening a large number of TCP connections to the message broker

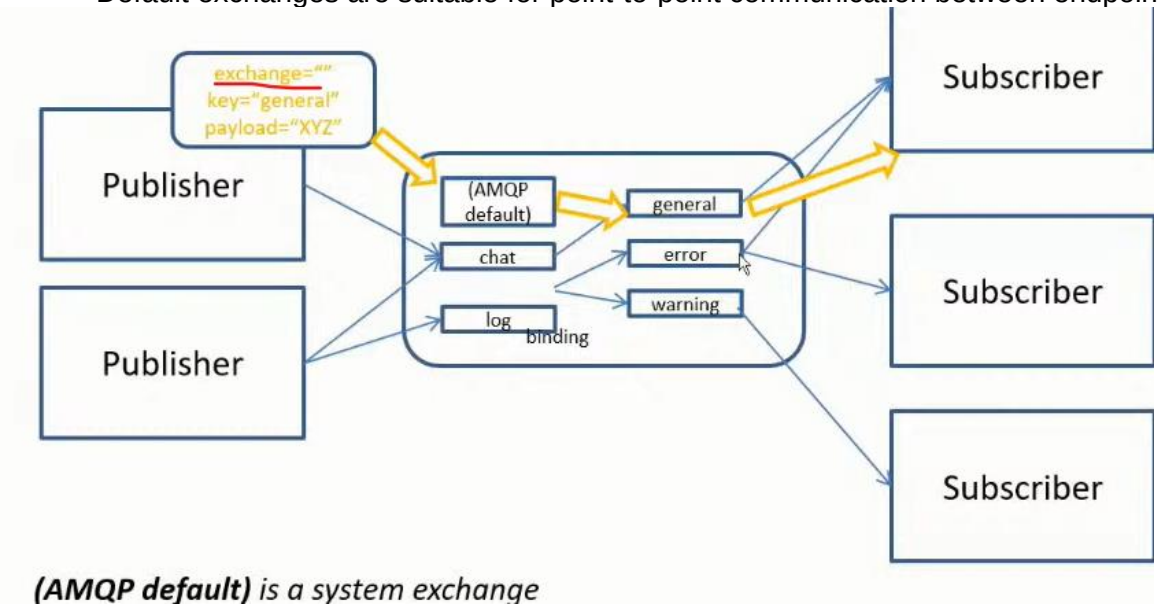
- Each message can be published with a **routing key**
- Each binding between an exchange and a queue has a **binding key**
- Routing of messages is determined based on matching between the **routing key** and the **binding key**

Messaging patterns with RabbitMQ

- Different types of messaging patterns are implemented by means of different types of exchanges
- RabbitMQ provides the following types of exchanges:
    - default - без име като търси съвпадение на **routing key** с **binding key**
    - direct - има име като търси съвпадение на **routing key** с **binding key**
    - fanout
    - topic
    - headers- на база мачинг по хедъри също

Default exchange

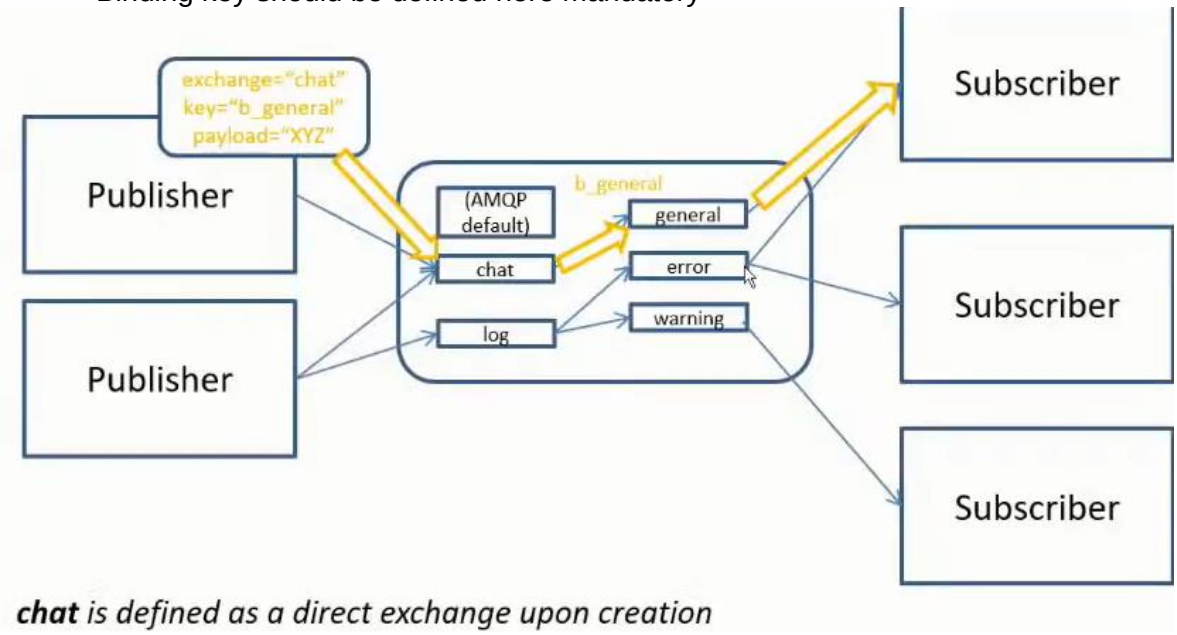
- A default exchange has **the empty string as a name** and routes messages to a queue if the routing key of the message matches the queue name (no binding needs to be declared between a default exchange and a queue)
- Default exchanges are suitable for point-to-point communication between endpoints



***(AMQP default)** is a system exchange*

Direct exchange

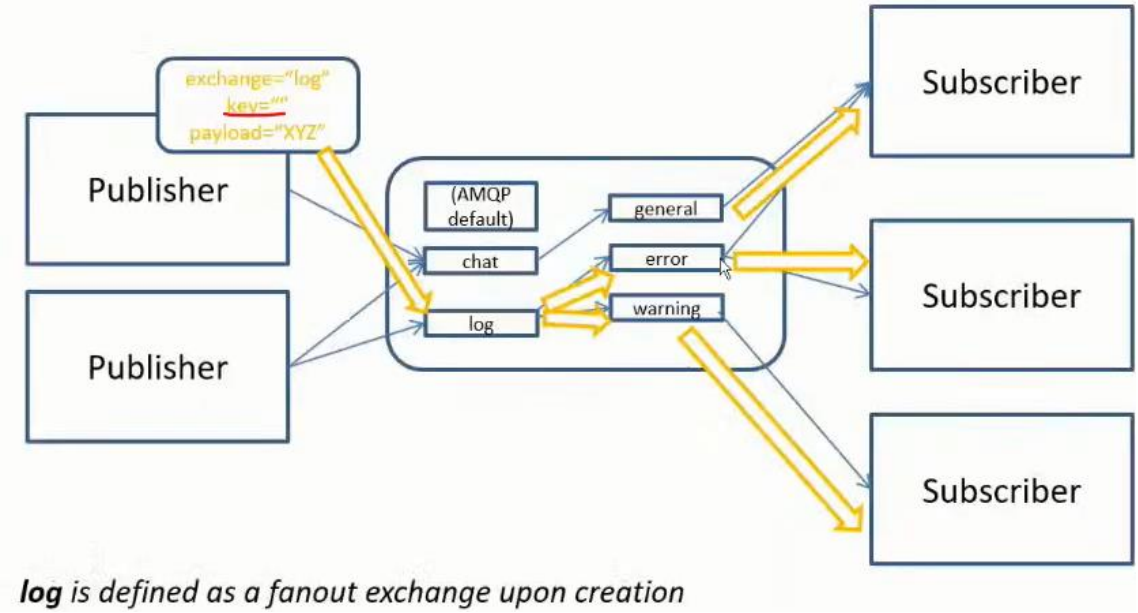
- A direct exchange routes messages to a queue if the routing key of the message matches the binding key between the direct exchange and the queue
- Direct exchanges are suitable for point-to-point communication between endpoints
- Binding key should be defined here mandatory



***chat** is defined as a direct exchange upon creation*

Fanout exchange

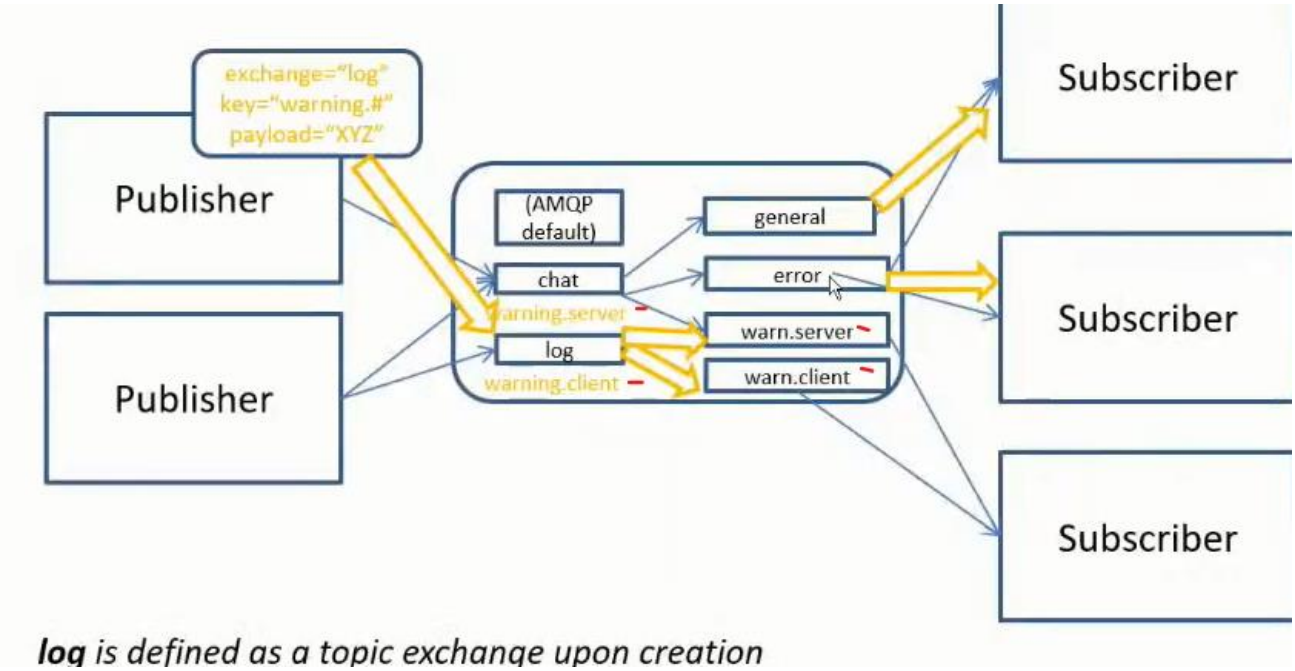
- A fanout exchange routes (broadcasts) messages to all queues that are bound to it (the binding key is not used)
- Fanout exchanges are suitable for publish-subscribe communication between endpoints



***log** is defined as a fanout exchange upon creation*

Topic exchange

- A topic exchange routes (multicasts) messages to all queues that have a binding key (can be a pattern) that matches the routing key of the message
- Topic exchanges are suitable for routing messages to different queues based on the type of message
- Диеса след *warning.* може да е всякаква дума

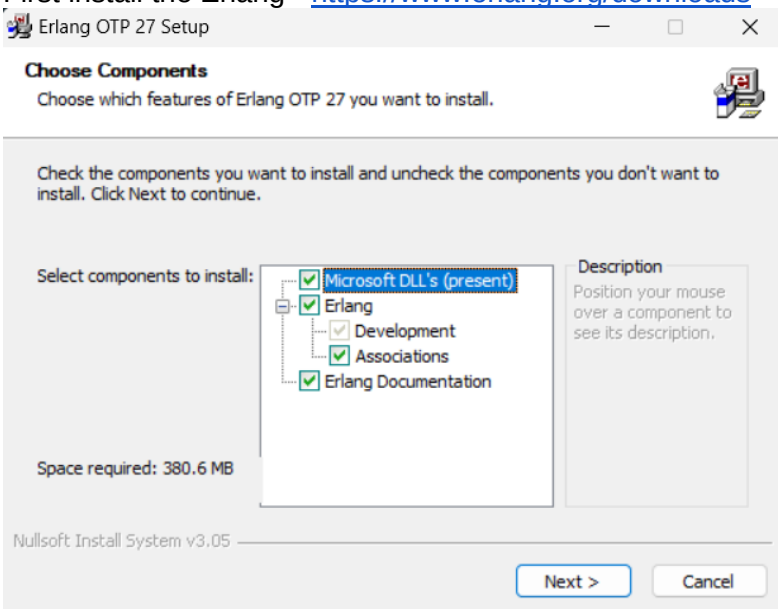


Headers exchange

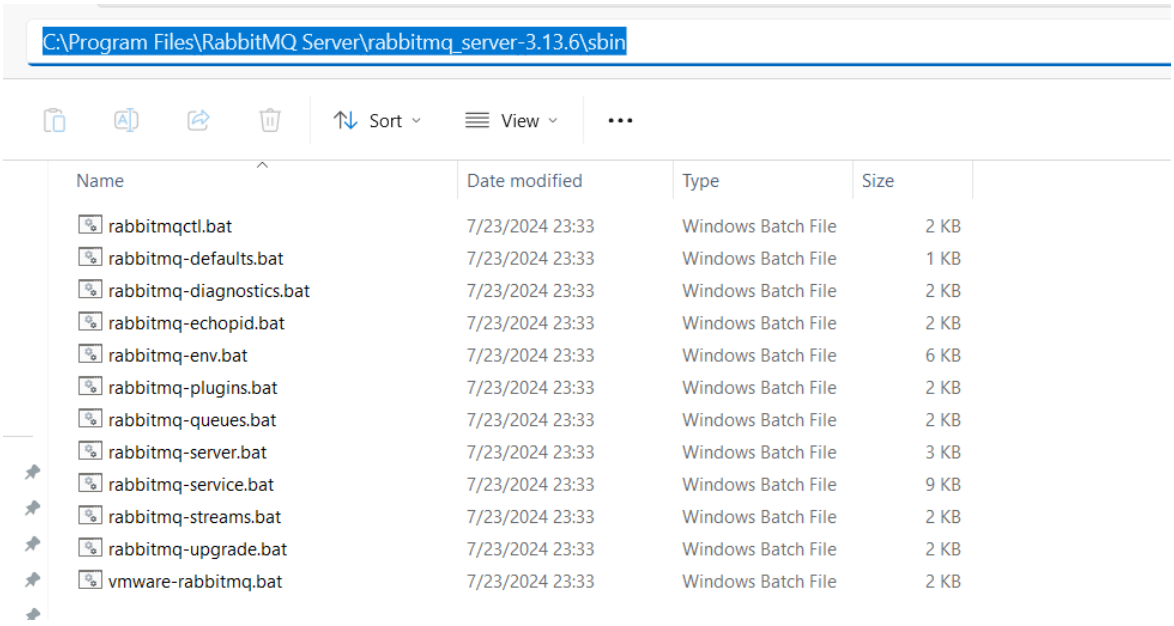
- A headers exchange routes messages based on a custom message header
- Header exchanges are suitable for routing messages to different queues based on more than one attribute

Installation of the RabbitMQ server

First install the Erlang - https://www.erlang.org/downloads



Then install the RabbitMQ server - https://www.rabbitmq.com/docs/download
rabbitmq-service.bat
rabbitmq-service.bat



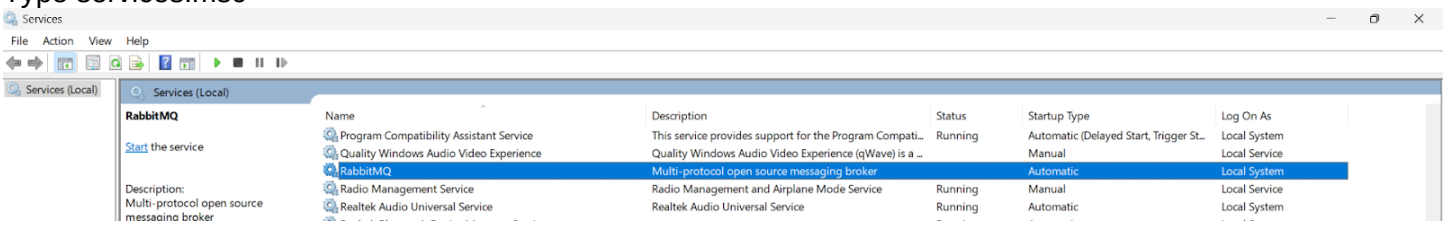
C:\Program Files\RabbitMQ Server\rabbitmq_server-3.13.6\sbin>**rabbitmq-plugins.bat enable** rabbitmq_management
Enabling plugins on node rabbit@SVILKATA:
rabbitmq_management
The following plugins have been configured:
  rabbitmq_management
  rabbitmq_management_agent
  rabbitmq_web_dispatch
Applying plugin configuration to rabbit@SVILKATA...
The following plugins have been enabled:
  rabbitmq_management
  rabbitmq_management_agent
  rabbitmq_web_dispatch

set 3 plugins.
Offline change; changes will take effect at broker restart.

През CommandPrompt като администратор:
C:\Program Files\RabbitMQ Server\rabbitmq_server-3.13.6\sbin>rabbitmq-plugins.bat list
Listing plugins with pattern ".*" ...
 Configured: E = explicitly enabled; e = implicitly enabled
 | Status: [failed to contact rabbit@SVILKATA - status not shown]
 |/
[ ] rabbitmq_amqp1_0 3.13.6
[ ] rabbitmq_auth_backend_cache 3.13.6
[ ] rabbitmq_auth_backend_http 3.13.6
[ ] rabbitmq_auth_backend_ldap 3.13.6
[ ] rabbitmq_auth_backend_oauth2 3.13.6
[ ] rabbitmq_auth_mechanism_ssl 3.13.6
[ ] rabbitmq_consistent_hash_exchange 3.13.6
[ ] rabbitmq_event_exchange 3.13.6
[ ] rabbitmq_federation 3.13.6
[ ] rabbitmq_federation_management 3.13.6
[ ] rabbitmq_jms_topic_exchange 3.13.6
[*] rabbitmq_management 3.13.6
[*] rabbitmq_management_agent 3.13.6
[ ] rabbitmq_mqtt 3.13.6

```
[ ] rabbitmq_peer_discovery_aws     3.13.6
[ ] rabbitmq_peer_discovery_common          3.13.6
[ ] rabbitmq_peer_discovery_consul 3.13.6
[ ] rabbitmq_peer_discovery_etcd    3.13.6
[ ] rabbitmq_peer_discovery_k8s     3.13.6
[ ] rabbitmq_prometheus             3.13.6
[ ] rabbitmq_random_exchange        3.13.6
[ ] rabbitmq_recent_history_exchange  3.13.6
[ ] rabbitmq_sharding               3.13.6
[ ] rabbitmq_shovel             3.13.6
[ ] rabbitmq_shovel_management    3.13.6
[ ] rabbitmq_stomp              3.13.6
[ ] rabbitmq_stream             3.13.6
[ ] rabbitmq_stream_management              3.13.6
[ ] rabbitmq_top                3.13.6
[ ] rabbitmq_tracing            3.13.6
[ ] rabbitmq_trust_store            3.13.6
[*] rabbitmq_web_dispatch           3.13.6
[ ] rabbitmq_web_mqtt               3.13.6
[ ] rabbitmq_web_mqtt_examples    3.13.6
[ ] rabbitmq_web_stomp              3.13.6
[ ] rabbitmq_web_stomp_examples             3.13.6
```

Type services.msc



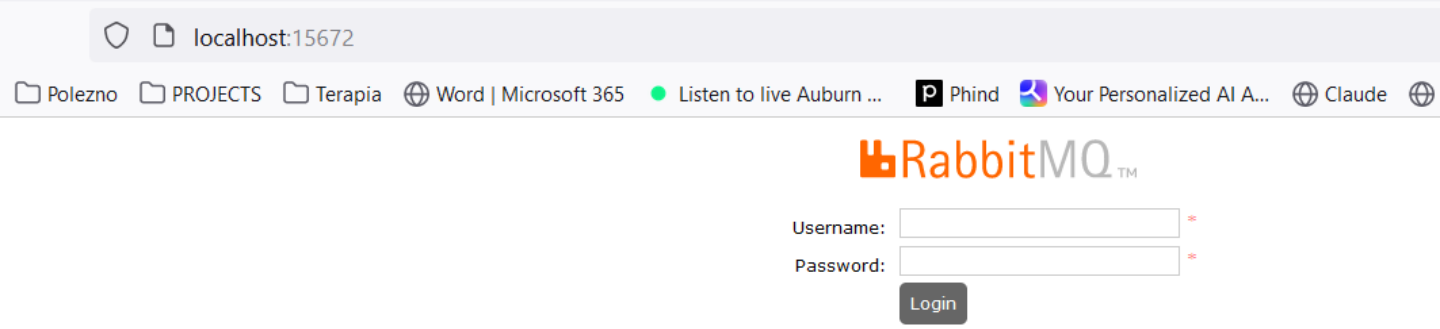
C:\Program Files\RabbitMQ Server\rabbitmq_server-3.13.6\sbin>**rabbitmq-server.bat**
2024-07-31 11:52:47.571000+03:00 [warning] <0.134.0> Using RABBITMQ_ADVANCED_CONFIG_FILE: c:/Users/svilk/AppData/Roaming/RabbitMQ/advanced.config
2024-07-31 11:52:52.145000+03:00 [notice] <0.45.0> Application syslog exited with reason: stopped
2024-07-31 11:52:52.145000+03:00 [notice] <0.213.0> Logging: switching to configured handler(s); following messages may not be visible in this log output

```
  ##  ##      RabbitMQ 3.13.6
  ##  ##
  ##########  Copyright (c) 2007-2024 Broadcom Inc and/or its subsidiaries
  ######  ##
  ##########  Licensed under the MPL 2.0. Website: https://rabbitmq.com

  Erlang:      27.0.1 [jit]
  TLS Library: OpenSSL - OpenSSL 3.1.0 14 Mar 2023
  Release series support status: see https://www.rabbitmq.com/release-information

  Doc guides:  https://www.rabbitmq.com/docs
  Support:     https://www.rabbitmq.com/docs/contact
  Tutorials:   https://www.rabbitmq.com/tutorials
  Monitoring:  https://www.rabbitmq.com/docs/monitoring
  Upgrading:   https://www.rabbitmq.com/docs/upgrade

  Logs: <stdout>
```

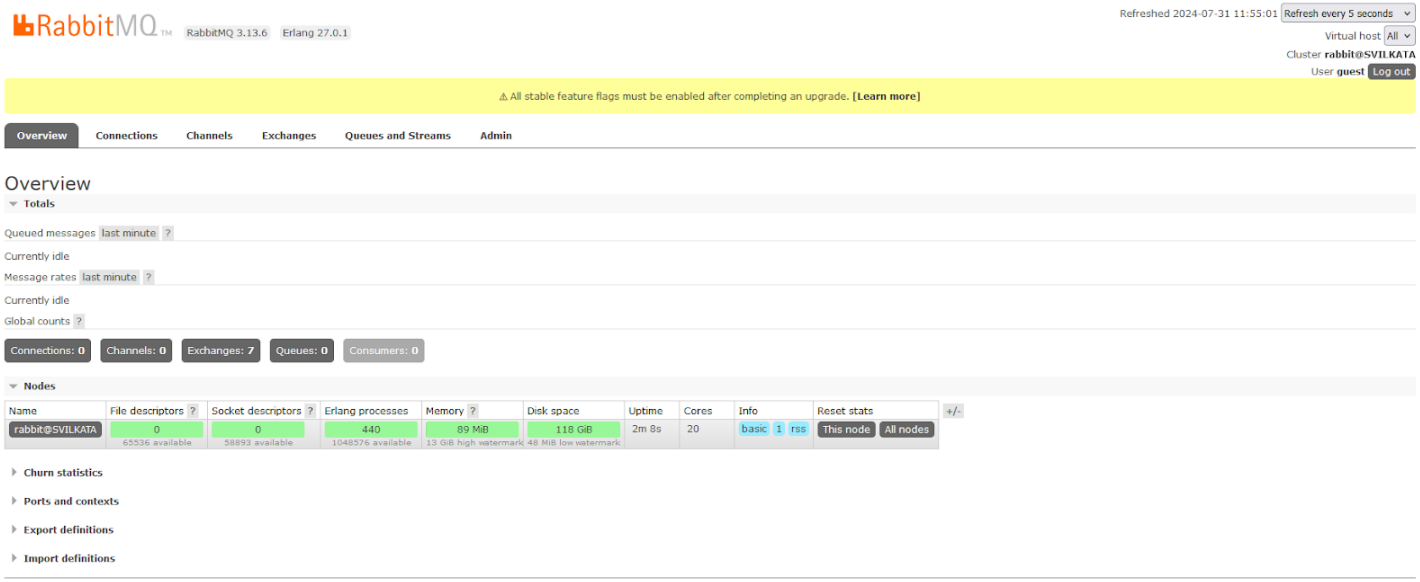
c:/Users/svilk/AppData/Roaming/RabbitMQ/log/rabbit@SVILKATA.log

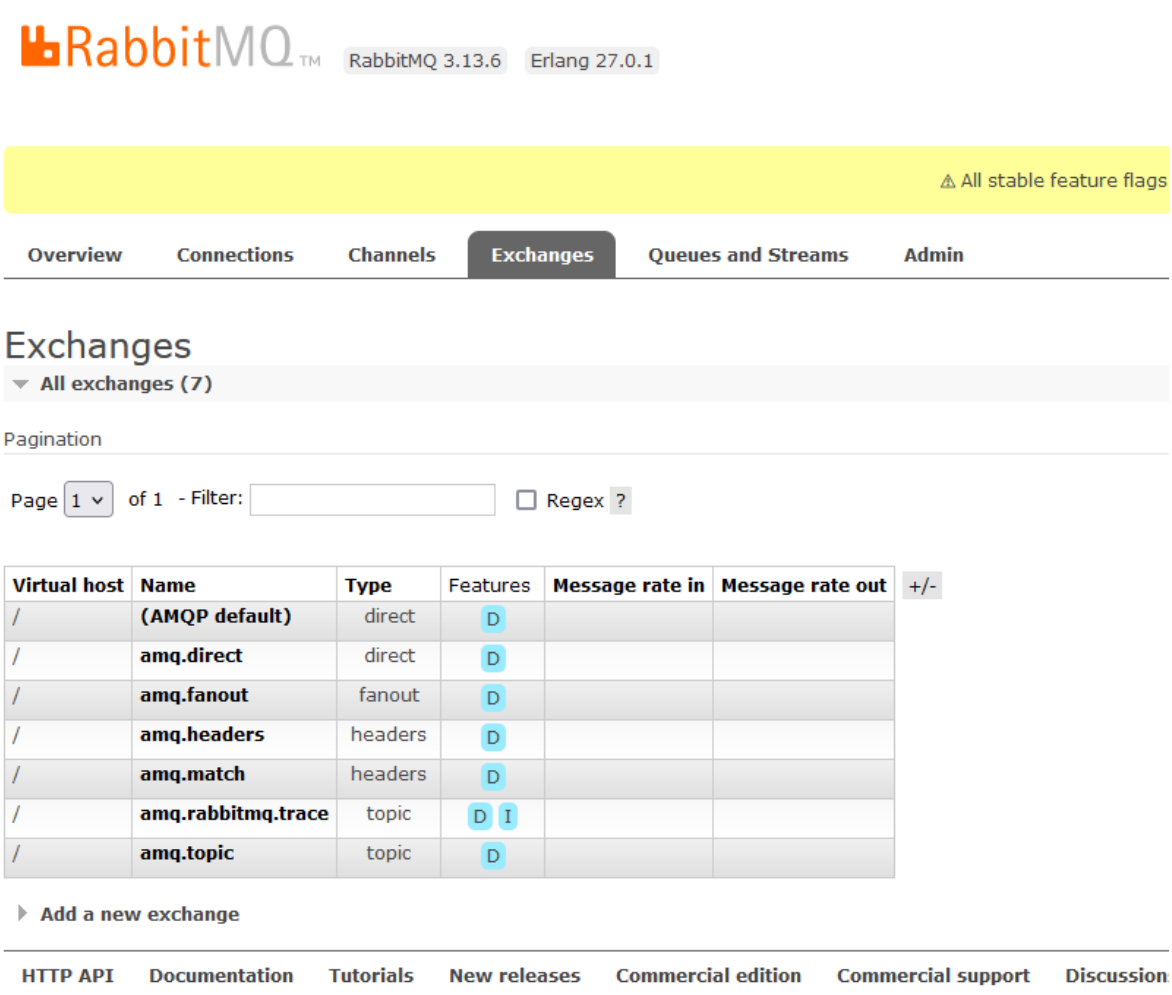
Config file(s): c:/Users/svilk/AppData/Roaming/RabbitMQ/advanced.config

Starting broker... **completed with 3 plugins.**



username: guest
password: guest



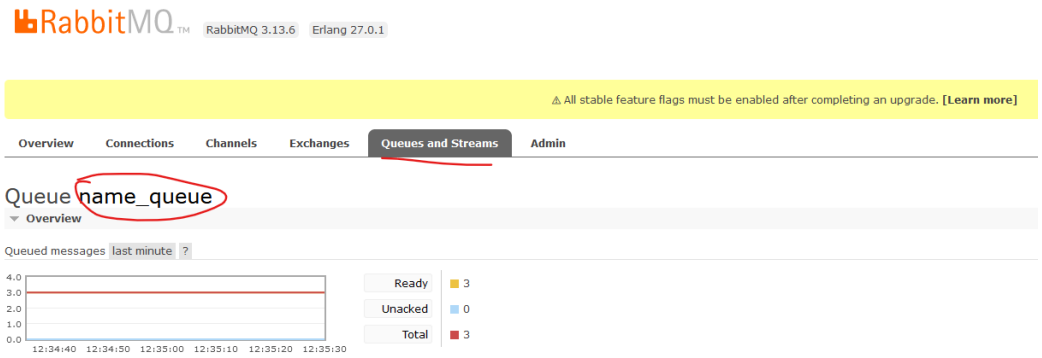


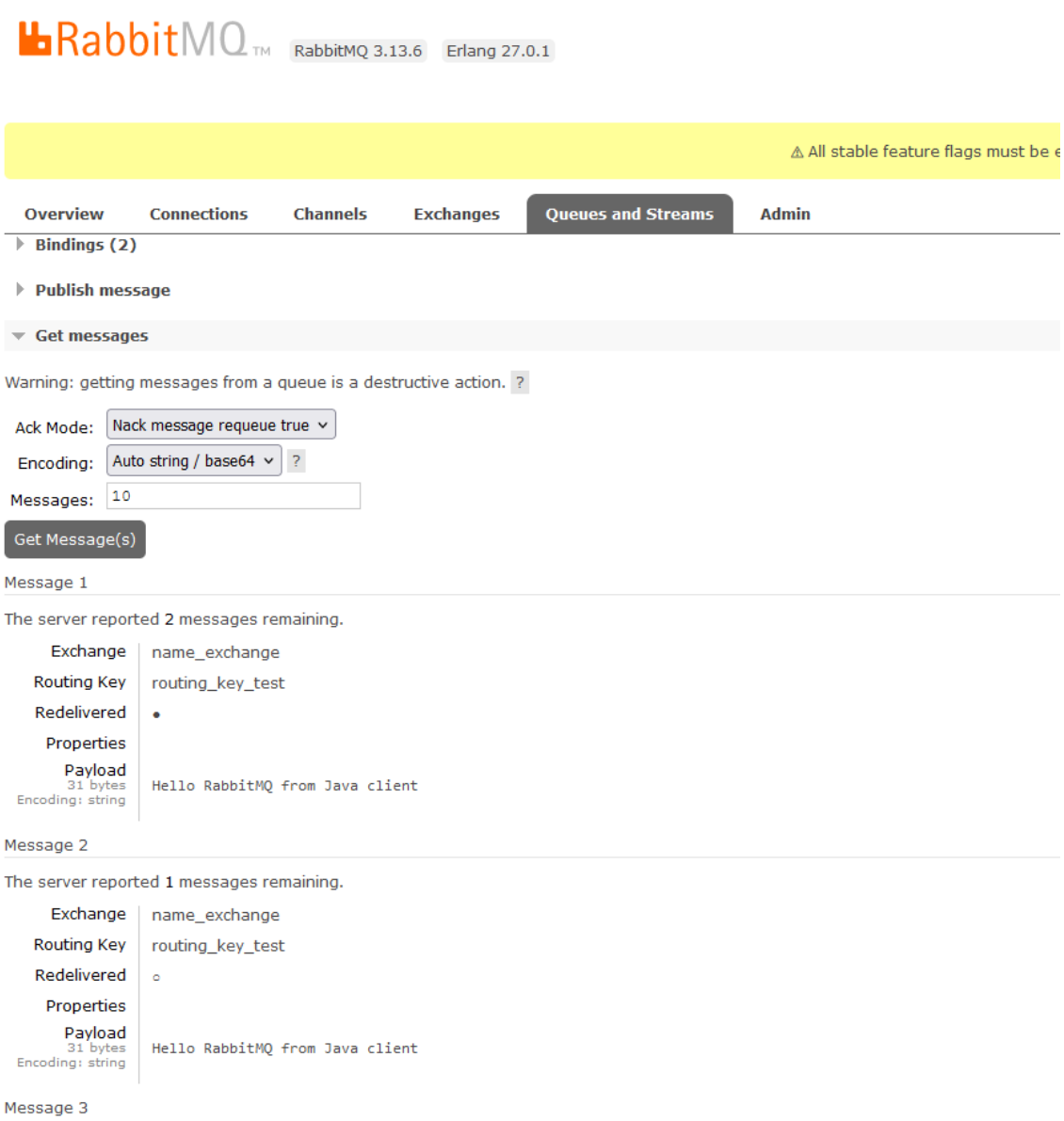
Using the Java Client

```
<dependency>
 <groupId>com.rabbitmq</groupId>
 <artifactId>amqp-client</artifactId>
 <version>5.20.0</version>
</dependency>
```

```
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;

import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.concurrent.TimeoutException;

public class Publisher {
   public static void main(String[] args) throws IOException, TimeoutException {
       Connection connection = null;
       Channel channel = null;

       try {
           ConnectionFactory connectionFactory = new ConnectionFactory();
           connectionFactory.setHost("localhost"); //by default on port 15672
           connection = connectionFactory.newConnection();
           channel = connection.createChannel();
```

```
            channel.exchangeDeclare("name_exchange", "direct"); //created only once on
the RabbitMQ server
            channel.queueDeclare("name_queue", false, false, false, null); //created
only once on the RabbitMQ server
            channel.queueBind("name_queue", "name_exchange", "routing_key_test");

            channel.basicPublish("name_exchange", "routing_key_test", null,
                    "Hello RabbitMQ from Java
client".getBytes(StandardCharsets.UTF_8));
        } finally {
            if (channel != null) {
                channel.close();
            }
            if (connection != null) {
                connection.close();
            }
        }
    }
}
```






```java
import com.rabbitmq.client.AMQP;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.DefaultConsumer;
import com.rabbitmq.client.Envelope;

import java.io.IOException;
import java.util.concurrent.TimeoutException;

public class Subscriber {
   public static void main(String[] args) throws IOException, TimeoutException,
InterruptedException {
       Connection connection = null;
       Channel channel = null;

       try {
           ConnectionFactory connectionFactory = new ConnectionFactory();
           connectionFactory.setHost("localhost"); //by default on port 15672
           connection = connectionFactory.newConnection();
           channel = connection.createChannel();
```

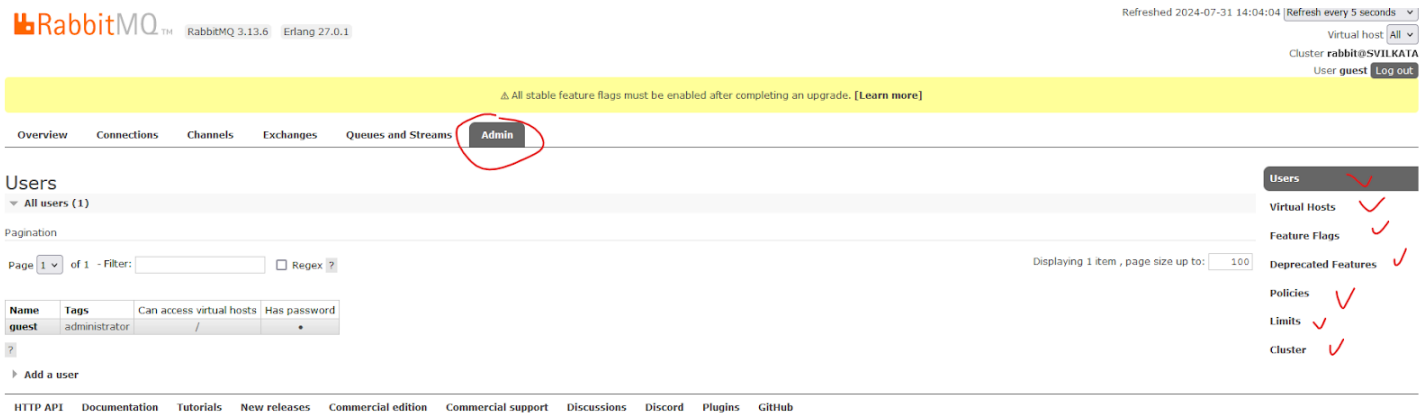
```
            channel.exchangeDeclare("name_exchange", "direct"); //created only once on
the RabbitMQ server
            channel.queueDeclare("name_queue", false, false, false, null); //created
only once on the RabbitMQ server
            channel.queueBind("name_queue", "name_exchange", "routing_key_test");

            while (true) {
                channel.basicConsume("name_queue", true, new DefaultConsumer(channel)
{
                    @Override
                    public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
                        // super.handleDelivery(consumerTag, envelope, properties,
body);  no-op no work to do
                        System.out.println(new String(body));
                    }
                });

                Thread.sleep(3000);
            }
        } finally {
            if (channel != null) {
                channel.close();
            }
            if (connection != null) {
                connection.close();
            }
        }
    }
}
```

Administration

- Administration of the broker includes a number of activities such as:
  - Updating the broker
  - Backing up the broker database
  - installing/uninstalling and configuring plug-ins
  - Configuring the various components of the broker

- Apart from queues, exchanges and bindings we can also manage the following types of components:
  - vhosts (virtual hosts) - for logical separation of broker components
  - users
  - Parameters - defining upstream links to another brokers
  - Policies - for queue mirroring

- Administration of single instance or an entire cluster can be performed in several ways:
  - Using the management Web interface



- Using the management HTTP API - rest API
- Using the **rabbitmq-admin.py /** **rabbitmqadmin.py** script - written on Python
- Using the **rabbitmqctl** utility

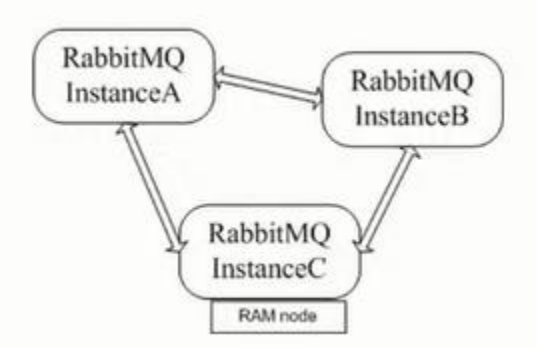
Scalability and High Availability in RabbitMQ

Basic default configuration

- RabbitMQ provides clustering support that allows new RabbitMQ nodes to be added on the fly
- Clustering by default does not guarantee that message loss may or may not occur - ако дадена инстанция примерно падне

- Nodes in a RabbitMQ cluster can be:
  - DISK - data is persisted in the node database
  - RAM - data is buffered only in-memory - когато не е критично да се запазват данните след рестарт например
- **Nodes share only broker metadata - messages are not replicated among nodes!! - съобщението не се репликира в останалите node-ве**

Example:
A и B са DISK nodes, а C е Ram node.



Instance A node DISK

```
set RABBITMQ_NODENAME=instanceA &
set RABBITMQ_NODE_PORT=5770 &
set RABBITMQ_SERVER_START_ARGS=
      -rabbitmq_management listener [{port,33333}] &
rabbitmq-server.bat -detached
```

Instance B node DISK

Пускаме инстанция B, спираме я, присъединяваме я след това

```
set RABBITMQ_NODENAME=instanceB &
set RABBITMQ_NODE_PORT=5771 &
rabbitmq-server.bat -detached
rabbitmqctl.bat -n instanceB stop_app
rabbitmqctl.bat -n instanceB join_cluster instanceA@MARTIN
rabbitmqctl.bat -n instanceB start_app
```

Instance C node RAM

Пускаме инстанция C, спираме я, присъединяваме я след това

```
set RABBITMQ_NODENAME=instanceC &
set RABBITMQ_NODE_PORT=5772 &
rabbitmq-server.bat -detached
rabbitmqctl.bat -n instanceC stop_app
rabbitmqctl.bat -n instanceC join_cluster -ram instanceA@MARTIN
rabbitmqctl.bat -n instanceC start_app
```

- If a node that hosts a queue buffers unprocessed messages goes down, then messages are lost
- Default clustering mechanism provides scalability in terms of queues rather than high availability

Mirrored queues

- **Mirrored queues** are an extension to the default clustering mechanism that can be used to establish **high availability** at the broker level
- Mirrored queues provide queue replication over different nodes that allows a message to survive node failure
- Queue mirroring is establishing by means of a mirroring policy that specifies:
  - Number of nodes to use for queue replication
  - Particular nodes designated by name for queue replication
  - All nodes for queue replication

- The node where the queue is created is the master node - all other nodes are slaves
- A new master node can be promoted in case the original one goes down
- A slave node is promoted to/as the new master in case it is fully synchronized with the old master

Example:

Let's define the test queue in the cluster and mirror it over all other nodes:

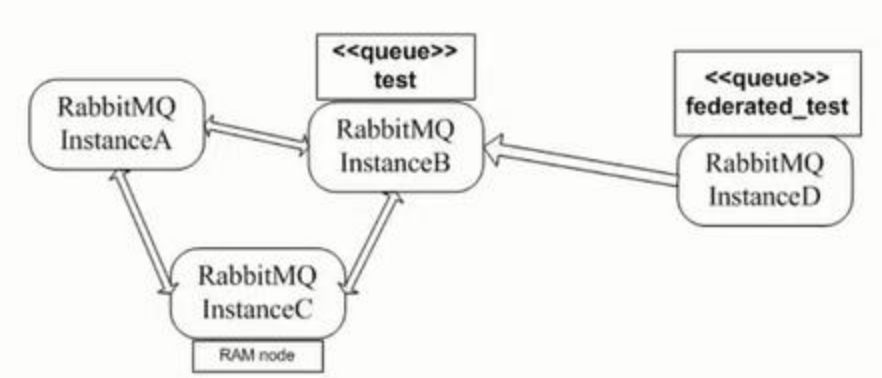
```
rabbitmqadmin.py -N instanceA declare queue name=test
durable=false
rabbitmqctl -n instanceA set_policy ha-all "test" "{""ha-
mode"":""all""}"
```

Federation and Shovel plugins

- The RabbitMQ clustering mechanism uses Erlang message passing along with a message cookie in order to establish communication between the nodes…….. which is **not reliable** over the Wide Area Networks!!
- In order to establish high availability among nodes in different geographic locations you can use the **federation, federation_management** and **shovel** plug-ins
- The shovel plug-in works at a lower level than the federation plug-in

Federation

- Ако искаме да репликираме опашката на отдалечена инстанция D:




```
set RABBITMQ_NODENAME=instanceD &
set RABBITMQ_NODE_PORT=6001 &
set RABBITMQ_SERVER_START_ARGS=
      -rabbitmq_management listener [{port,44444}] &
rabbitmq-server.bat -detached

rabbitmq-plugins -n instanceD enable rabbitmq_federation
rabbitmq-plugins -n instanceD enable
rabbitmq_federation_management
```

- Declare the **federated_test** queue

```
rabbitmqadmin.py -N instanceD -P 44444 declare queue
name=federated_test durable=false
```

Declare the upstream to the initial cluster and set a federation link to the **test** queue:

```
rabbitmqctl -n instanceD set_parameter federation-upstream
upstream
"{""uri"":""amqp://localhost:5770"",""expires"":3600000,
""queue"":""test""}"

rabbitmqctl -n instanceD set_policy federate-queue
--apply-to queues "federated_test"
"{""federation-upstream"":""upstream""}"
```

Shovel

The shovel plug-in provides two variants:

- **static** - all links between the source/destination nodes/clusters are defined statically in the RabbitMQ configuration file
- **dynamic** - all links between the source/destination nodes/clusters are defined dynamically via the RabbitMQ parameters

source \ destination	exchange	queue
exchange	federation dynamic shovel	dynamic shovel
queue	static shovel dynamic shovel	federation dynamic shovel

Integrations

Info

- RabbitMQ provides integrations with other protocols such as STOMP, MQTT and LDAP by means of RabbitMQ plug-ins
- Using the Java Client - already discussed above
- The Spring framework provides integration with AMQP protocol and RabbitMQ in particular
- The **Spring AMQP framework** provides:
  - **RabbitAdmin** class for automatically declaring queues, exchanges and bindings
  - **Listener container** for asynchronous processing of inbound messages
  - **RabbitTemplate** class for sending and receiving messages

- Utilities of the Spring AMQP framework can be used directly in Java or preconfigured in the Spring configuration
- The **Spring Integration framework** to **Spring Boot** provides adapters for the AMQP protocol

- Integration with Quarkus framework

Spring AMQP framework

```
<dependencies>
    <dependency>
        <groupId>org.springframework.amqp</groupId>
        <artifactId>spring-rabbit</artifactId>
        <version>1.4.5.RELEASE</version>
    </dependency>
</dependencies>
```

The **RabbitAdmin** class:

```
CachingConnectionFactory factory = new
       CachingConnectionFactory("localhost");
RabbitAdmin admin = new RabbitAdmin(factory);
Queue queue = new Queue("sample-queue");
admin.declareQueue(queue);
TopicExchange exchange = new TopicExchange("sample-topic-
       exchange");
admin.declareExchange(exchange);
admin.declareBinding(BindingBuilder.bind(queue).to(exchange)
       .with("sample-key"));
factory.destroy();
```

Listener container

```
CachingConnectionFactory factory =
       new CachingConnectionFactory(
"localhost");
SimpleMessageListenerContainer container = new
SimpleMessageListenerContainer(
       factory);
Object listener = new Object() {
    public void handleMessage(String message) {
    System.out.println("Message received: " +
    message);
}};
MessageListenerAdapter adapter = new
       MessageListenerAdapter(listener);
container.setMessageListener(adapter);
container.setQueueNames("sample-queue");
container.start();
```

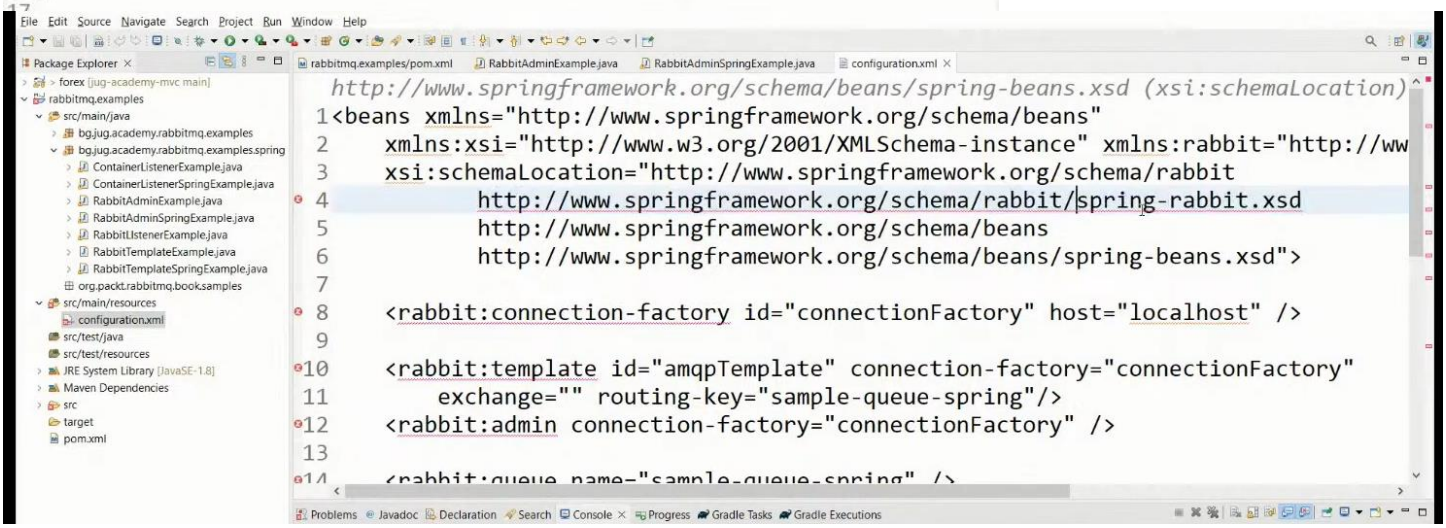
The **RabbitTemplate** class:

```
CachingConnectionFactory factory =
       new CachingConnectionFactory("localhost");
RabbitTemplate template = new
RabbitTemplate(factory);
template.convertAndSend("", "sample-queue",
        "sample-queue test message!");
```

- All of the above Spring AMQP framework examples can be configured using the Spring configuration - so that to be cleaner and to decouple RabbitMQ configuration from the business logic/Java code

For example within a **configuration.xml** file:

```
7 public class RabbitAdminSpringExample {
8
9     public static void main(String[] args) {
10
11         AbstractApplicationContext context = new ClassPathXmlApplicationContext(
12                 "configuration.xml");
13         RabbitAdmin admin = context.getBean(RabbitAdmin.class);
14     }
15
16 }
```



Spring Boot Starter AMQP - Spring Integration framework

In gradle

implementation 'org.springframework.boot:**spring-boot-starter-amqp**'

Предоставя ни бийнове за **RabbitAdmin** class, **Listener container** and **RabbitTemplate** class.

Посредством дефиниране на бийнове - можем да си декларираме **exchange, queue** или **queueBinding**

```
@Component
public class EventingServiceImpl implements EventingService {

    private final RabbitTemplate template;

    public EventingServiceImpl(RabbitTemplate template) {
        this.template = template;
    }

    @Bean
    public Queue createExchangeQueue() {
        return new Queue("exchange_rate_queue");
    @Override
    public void publish(String message) {
        template.convertAndSend("exchange_rate_queue", message);
    }

}
```

Quarkus framework

```
<dependency>
 <groupId>io.quarkus</groupId>
 <artifactId>quarkus-smallrye-reactive-messaging-rabbitmq</artifactId>
</dependency>
```

In application.properties file

```
mp.messaging.outgoing.bi.use-ssl=true
mp.messaging.outgoing.bi.connector=smallrye-rabbitmq
mp.messaging.outgoing.bi.exchange.type=direct
mp.messaging.outgoing.bi.port=5672
```

```
import io.smallrye.reactive.messaging.rabbitmq.OutgoingRabbitMQMetadata;

@Inject
@Channel("asd")  //from Microprofile
Emitter<String> emitter;  //from Microprofile

public void emmitMessage(RabbitMqPayload payload) {
   String messagePayload = JsonUtils.toJsonString(List.of(payload));
   LOGGER.debugf("Emitting Bi message to RabbitMQ %s", messagePayload);
   OutgoingRabbitMQMetadata metadata = new OutgoingRabbitMQMetadata.Builder()
           .withRoutingKey(payload.routingKey())
           .build();

   Message<String> message = Message.of(messagePayload, Metadata.of(metadata));
   biEmitter.send(message); //from Microprofile
   LOGGER.infof("Bi message emitted to route %s", payload.routingKey());
}
```

Security

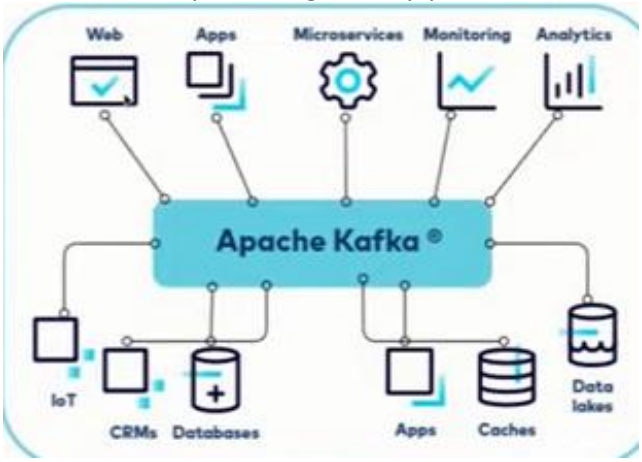
- RabbitMQ uses SASL Simple Authentication Security Layer for authentication (SASL PLAIN used by default)
- RabbitMQ uses access control lists (permissions) for authorization
- SSL/TLS support can be enabled for the AMQP communication channels
- SSL/TLS support can be enabled for node communication between nodes in a cluster
- SSL/TLS support can be enabled for the federation and shovel plug-ins

II. Apache Kafka

Book
Kafka – The Definitive Guide – Real-Time Data and Stream Processing at Scale

Use cases

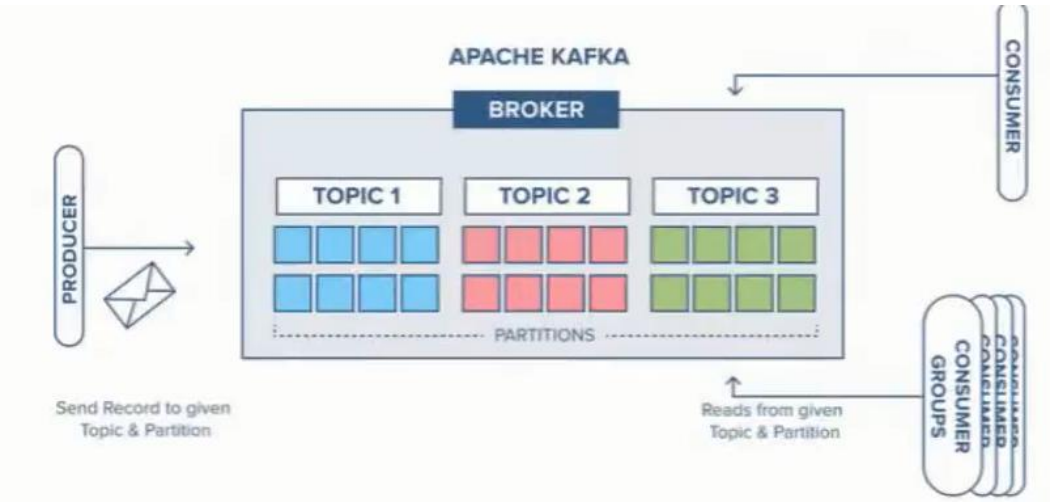
- For activity tracking – например LinkedIn да знае кой профил гледате/се гледа най-много
- Messaging
- Metrics and logging
- Commit log
- Stream processing – data pipeline – за един ден колко човека са кликнали на моята страница



Brokers and Clusters

A single Kafka server/instance is called a ***broker***. The broker receives messages from producers, assigns offsets to them, and writes the messages to storage on disk. It also services consumers, responding to fetch requests for partitions and responding with the messages that have been published. Depending on the specific hardware and its performance characteristics, a single broker can easily handle thousands and millions of messages per second.

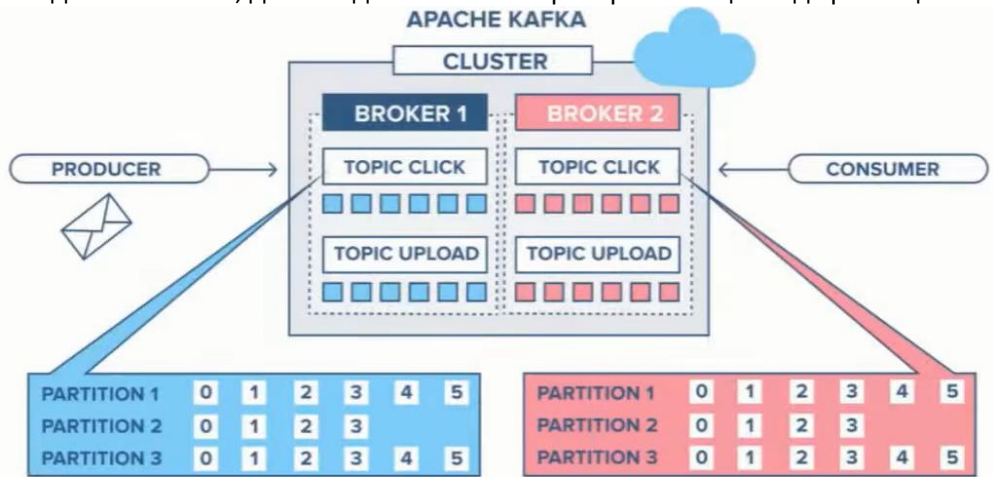
В един Kafka инстанция може да има 1 или повече топици, а във всеки топик има определен брой partitions.



Cluster – комбинация от няколко брокера (поне 3), т.е. няколко брокера които работят заедно и се координират заедно.

С други думи да има Fault tolerance, и когато един broker instance падне/или целият cluster падне, то да има опция да се репликират данните.

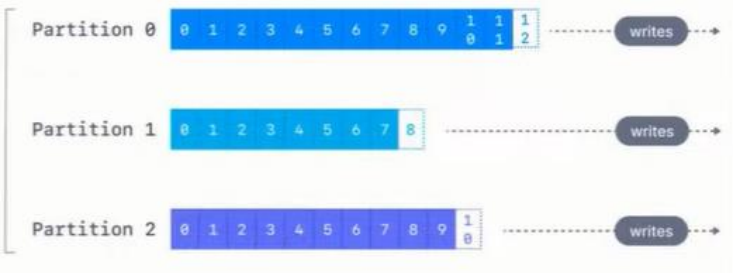
Като си правим cluster, то самите broker instances да бъдат географски на различни места – че ако падне нета на единия instance, да не паднат всичките брокер инстанции и дефакто целия клъстер.



Topics and Partitions

- With key – Messages are appended in the same partition if they have the same key
- Without key – messages are appended to the next partition in a round-robin fashion
- The order is guaranteed only in a partition and only if we have/use **key**

Kafka topic – основната единица



Колкото partitions има в даден топик, то толкова паралелни consumers може да имаме.

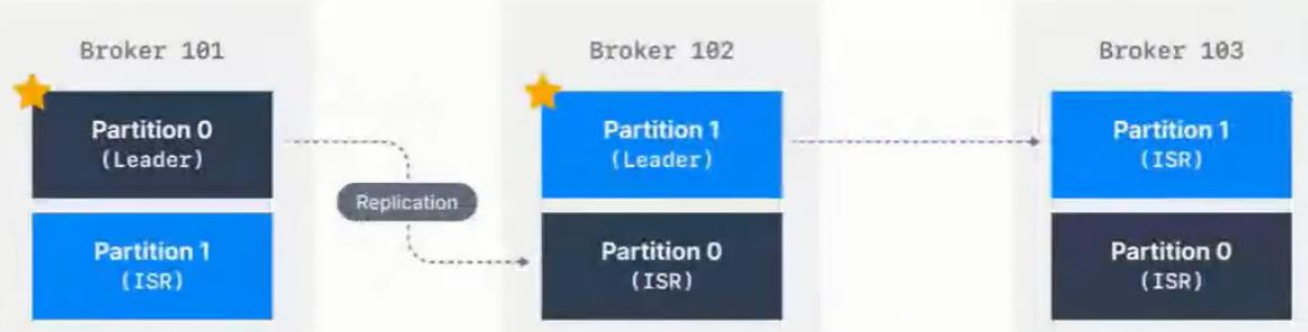
Ако имаме повече от 1 partition в даден топик, то нямаме гаранция за подредба. Освен разбира се ако не използваме **key!**

Partition представлява множество съобщения/елементи едно след друго, с определен индекс който наричаме **offset!**

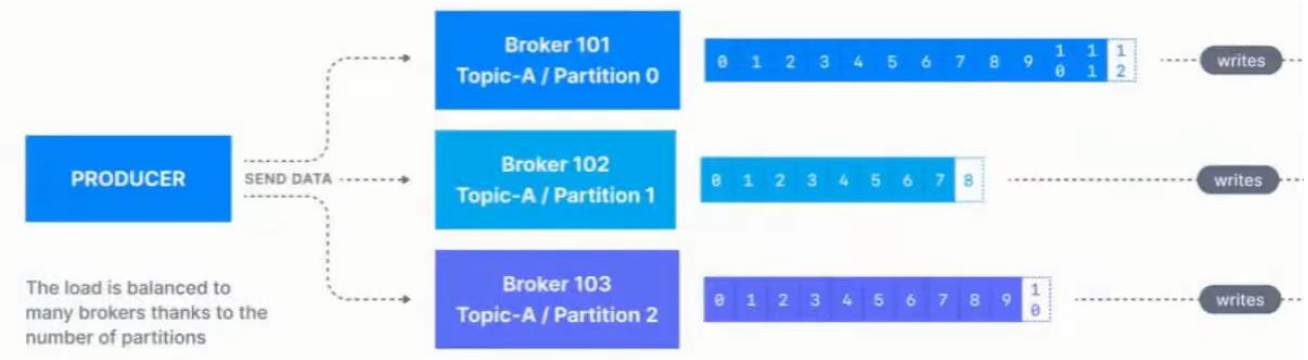
Topic replication

- **min.insync.replicas** – how many brokers (including the leader) should have the data before it is considered stored and ready to serve. Recommended is **replication.factor -1** (можем да настроим да я има тази информация на само 1 брокер, на още 2 брокера, и т.н. – зависи от use case-а и от наличните брокери)

Each partition is replicated separately, has its own broker leader and ISR.
Обикновено единия брокер се води Leader на partition-а.

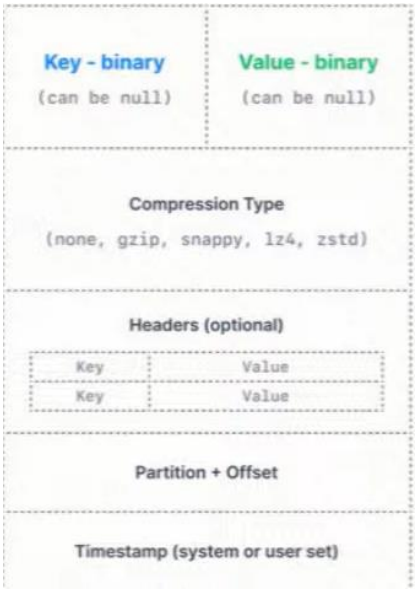


Когато изпращаме данни, ние изпращаме данни към съответния Leader на съответния partition.



What is a Kafka message

И key и value са **binary**! За Кафка всичко е byte-ове.
Възможност за компресия. Запазват се sequential – на следващия сегмент на диска.
Headers – например информация през кои сървиси е минало
Информация на кой partition е, и на кой офсет индекс е даденото съобщение.
Timestamp by default се слага от producer-а, а не вътрешно от Кафка брокера!

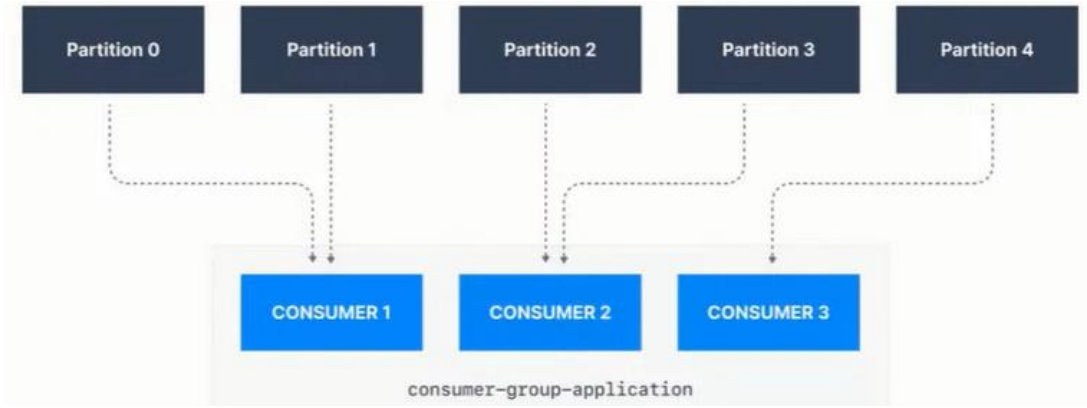


Consumers

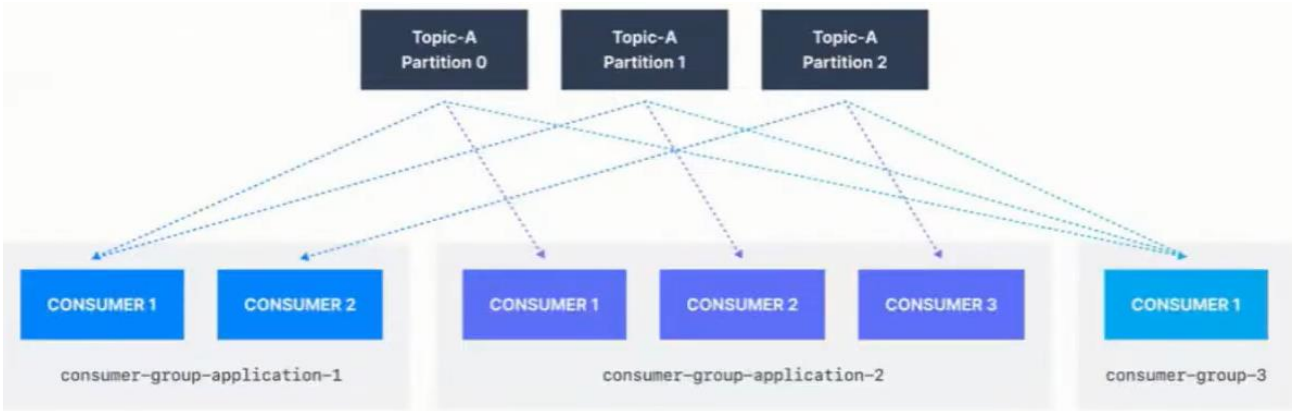
Kafka скалира не на базата на топици, а на базата на partitions!!!

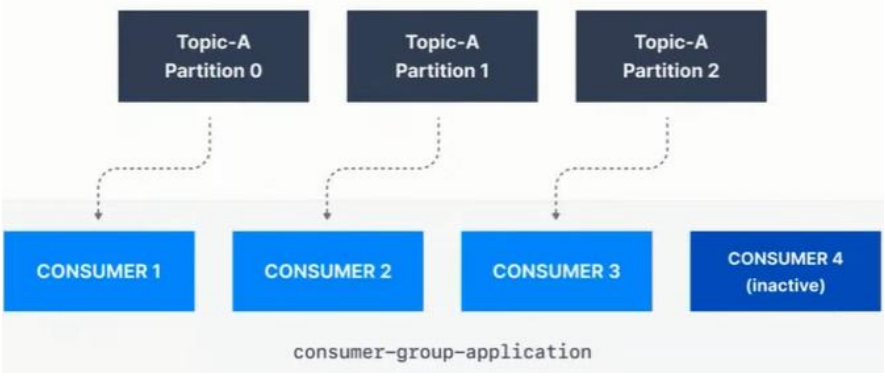
Consumer group

Пример: Имаме 5 партишъни. И разпределяме партишъните на различни consumers.
В текущият пример можем да сложим общо макс 5 consumers!!! Или с други думи правилото е че можем да скалираме consumers до броя на partitions!



Many consumers groups

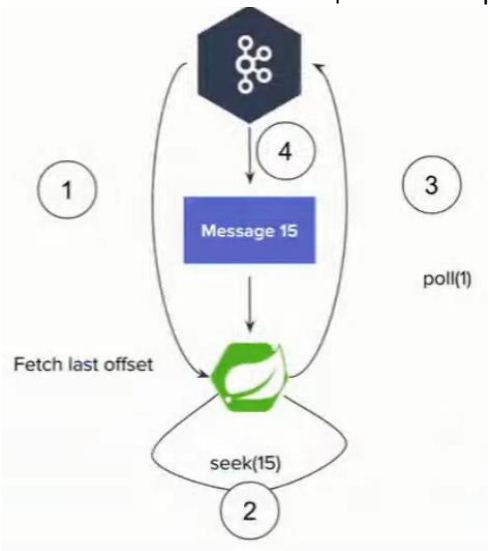




Commit

- Commit is the process of telling Kafka to remember that you have processed **(both produced and consumed)** the messages of a partition to a specific offset
- When first started, the consumer looks up where it left off via the kafka commit facility
- A call to **seek()** is made to initialize an internal counter which determines the messages that will be poll()-ed for the lifetime of the **consumer**

Poll означава вземи съобщението от брокера, и го процесни при теб (при Spring framework-а например)



Producer

Batching

spring.kafka.producer.batch-size=100
Sets the maximum number of records to be sent om one request. Sends the batch as soon as it is filled.

spring.kafka.producer.properties[linger.ms]
Gives the batch that much ms to be filled and sent automatically (see above). After the duration, if the batch is still not filled, sends it as it is.

spring.kafka.producer.buffer-memory
If a batch cannot be sent due to broker being down or whatever other recoverable issue, fills new batches in the buffer. After the buffer is filled, the **send()** method blocks for **spring.kafka.producer.properties[max.block.ms]**. This buffer **must be large enough** to contain at least one full batch or such batches will be lost.

spring.kafka.producer.compression-type=lz4 avro gzip snappy
The batch will be compressed before sending. This directly increases throughput and latency (not much latency)
Следните неща ни интересуват като изпращаме съобщения:

- **acks=0**

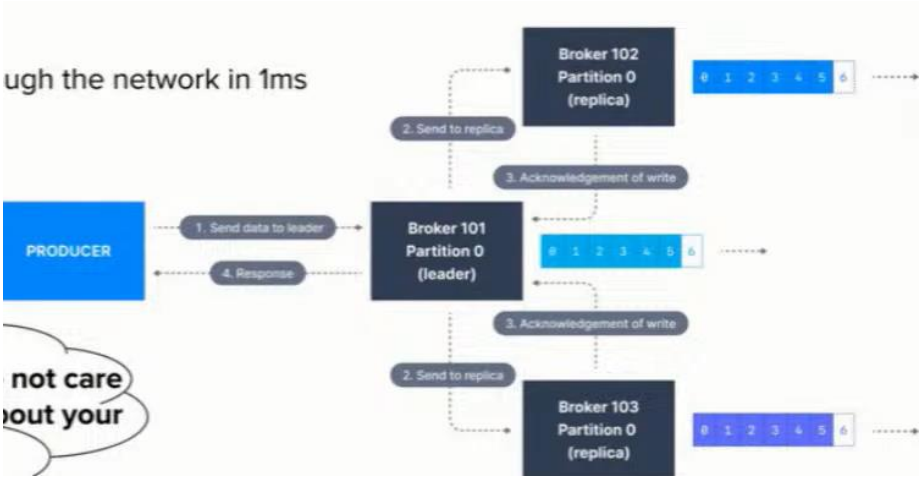
Does not wait for a response to consider the sent successful

- **acks=1**

Wait for the leader to commit the message/s but do not wait for the replicas to commit!!
Изчакай поне Leader да ти върне, че е записано съобщението, но не чакаме да се случи репликацията. Проблем би бил ако leader-а умре/падне, и тогава съобщението се губи

- **acks=all**

Wait for the leader and all replicas(или еди колко си на брой реплики) to commit, so that to consider the message/s sent.
Note: all == -1



x2 slower – докато се получат повтържденията за репликите, и се забавя
Assuming that a message/s is sent through the network in 1ms. This leads to 2.5ms latency.
1000 / 2ms = 2micro s per message

Consumer

Batching again

spring.kafka.consumer.max-poll-records=500
Sets the maximum number of records that will be returned by a poll()

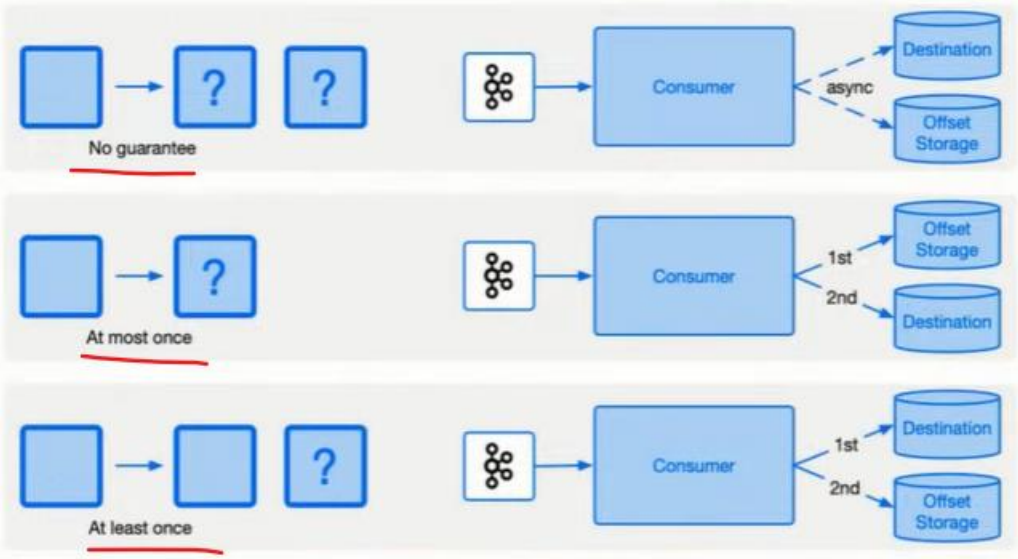
spring.kafka.listener.ack-mode=batch
Auto commit is mostly evil. Don't use it by default unless actually verifiably ok for your data
Като процеснеш batch-а, то commit-ни ръчно тогава offset-а. Ако се закача наново, то да не го процесна същото съобщение.
spring.kafka.consumer.enable-auto-commit=false

max.poll.interval.ms=5min
You must be able to process a full batch in that time or a rebalance will be triggered. На колко време да вземаме съобщенията обратно.

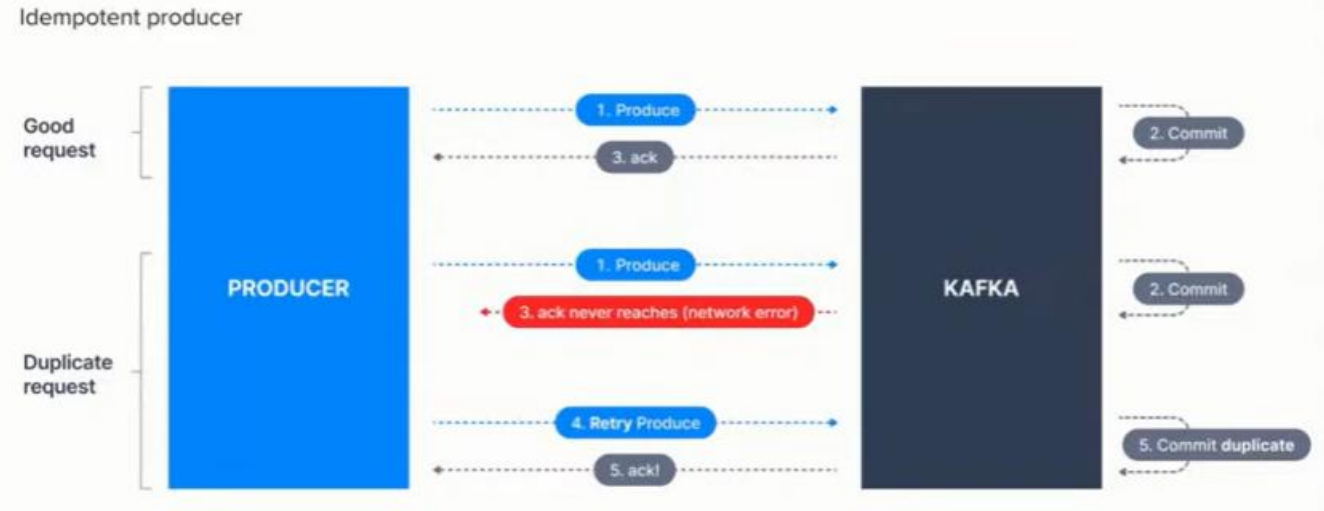
Delivery guarantees

Това съобщение защо сме го процеснали 2 пъти.
Или това съобщение защо изобщо не сме го процеснали.

At **least once** means not that a single message might be duplicated but a whole batch it is scary with big batches 😊



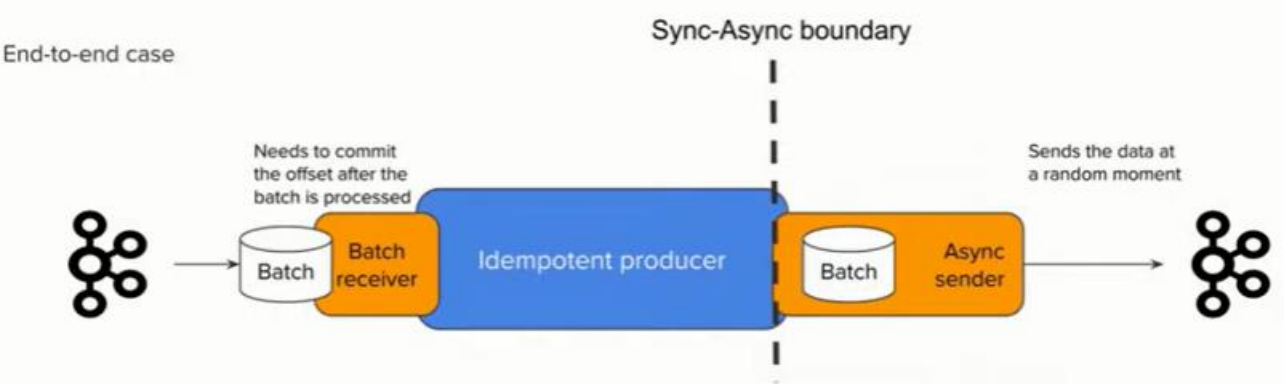
Idempotent producer – получаване на дуплицирани съобщения
Поради network проблем може Кафка брокера да е приел съобщението и да не отговори, и ние да изпратим съобщението наново.



Idempotent producer
spring.kafka.producer.properties[max.in.flight.requests.per.connection]=5
spring.kafka.producer.properties[enable.idempotence]=true
It works for the lifespan of the producer. Required but not enough for proper idempotency across restarts.

Ако е Network проблем, това решава дуплицирането.

Consumer – producer problem
Ако чета в Кафка, и пиша в Кафка – имам проблем. Защото трябва да имаме/използваме нещо, което се казва трансакции (трансакции в Кафка).



kafkaTemplate.send(msg)
This is fully async. You don't know when the data is actually going to be sent. The method returns a future that can be waited but to do it for each message breaks the batching and is extremely slow.

Demo with kafka-clients

```
Docker-compose file for Zookeeper, Kafka and RedPanda
and also logback.xml configuration. - see BGJUG public repo.
```

build.gradle – Groovy style

```
dependencies {
    implementation group: 'org.apache.kafka', name: 'kafka-clients', version: '3.6.1'
    //SLF4J Api
    implementation 'org.slf4j:slf4j-api:1.7.32' //Use the latest version available

    //Logback (SLF$J implementation)
    implementation 'ch.qos.logback:logback-classic:1.2.6' //Use the latest version
available

}
```

```
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;

import java.util.Properties;
```

```java
public class ProducerExample {
    public static void main(String[] args) {
        Properties properties = new Properties();
        properties.put("bootstrap.servers", "localhost:29092");
        properties.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        properties.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");

        Producer<String, String> producer = new KafkaProducer<>(properties);

        producer.send(new ProducerRecord<>("bgjug","KAFKA","Welcome to RabbitMQ"), (metadata,
exception) -> {
            if (exception == null) {
                System.out.println("Message sent successfully - Topic: " +
                        metadata.topic() + ", Partition: " + metadata.partition() +
                        ", Offset: " + metadata.offset());
            } else {
                System.err.println("Error sending message: " + exception.getMessage());
            }
        });

        producer.close();
    }
}
```

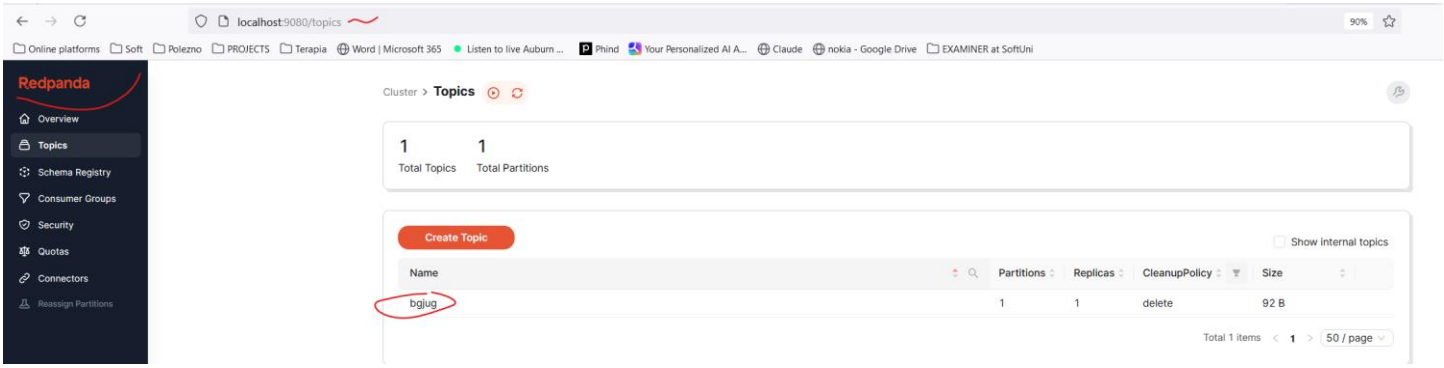
След като пуснем Producer-а, получаваме следния лог:

```
13:19:15.064 [main] INFO  o.a.k.c.producer.ProducerConfig - ProducerConfig values:
        acks = -1
        auto.include.jmx.reporter = true
        batch.size = 16384
        bootstrap.servers = [localhost:29092]
        buffer.memory = 33554432
        client.dns.lookup = use_all_dns_ips
        client.id = producer-1
        compression.type = none
        connections.max.idle.ms = 540000
        delivery.timeout.ms = 120000
        enable.idempotence = true
        interceptor.classes = []
        key.serializer = class org.apache.kafka.common.serialization.StringSerializer
        linger.ms = 0
        max.block.ms = 60000
        max.in.flight.requests.per.connection = 5
        max.request.size = 1048576
        metadata.max.age.ms = 300000
        metadata.max.idle.ms = 300000
        metric.reporters = []
        metrics.num.samples = 2
        metrics.recording.level = INFO
        metrics.sample.window.ms = 30000
        partitioner.adaptive.partitioning.enable = true
        partitioner.availability.timeout.ms = 0
        partitioner.class = null
        partitioner.ignore.keys = false
        receive.buffer.bytes = 32768
        reconnect.backoff.max.ms = 1000
        reconnect.backoff.ms = 50
        request.timeout.ms = 30000
        retries = 2147483647
        retry.backoff.ms = 100
        sasl.client.callback.handler.class = null
        sasl.jaas.config = null
        sasl.kerberos.kinit.cmd = /usr/bin/kinit
        sasl.kerberos.min.time.before.relogin = 60000
        sasl.kerberos.service.name = null
        sasl.kerberos.ticket.renew.jitter = 0.05
        sasl.kerberos.ticket.renew.window.factor = 0.8
        sasl.login.callback.handler.class = null
        sasl.login.class = null
```

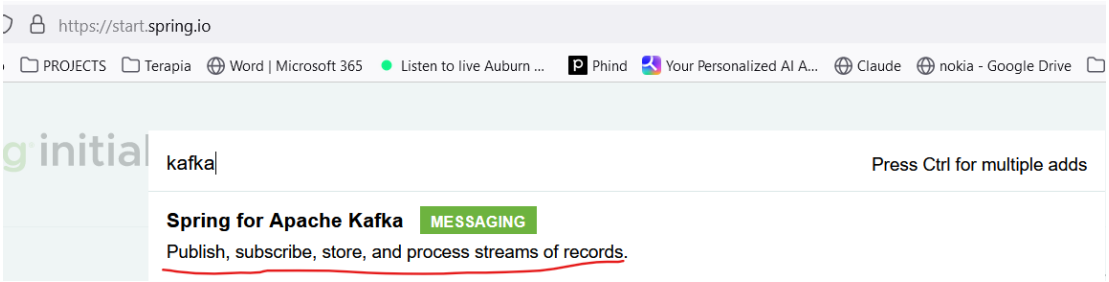
```
        sasl.login.connect.timeout.ms = null
        sasl.login.read.timeout.ms = null
        sasl.login.refresh.buffer.seconds = 300
        sasl.login.refresh.min.period.seconds = 60
        sasl.login.refresh.window.factor = 0.8
        sasl.login.refresh.window.jitter = 0.05
        sasl.login.retry.backoff.max.ms = 10000
        sasl.login.retry.backoff.ms = 100
        sasl.mechanism = GSSAPI
        sasl.oauthbearer.clock.skew.seconds = 30
        sasl.oauthbearer.expected.audience = null
        sasl.oauthbearer.expected.issuer = null
        sasl.oauthbearer.jwks.endpoint.refresh.ms = 3600000
        sasl.oauthbearer.jwks.endpoint.retry.backoff.max.ms = 10000
        sasl.oauthbearer.jwks.endpoint.retry.backoff.ms = 100
        sasl.oauthbearer.jwks.endpoint.url = null
        sasl.oauthbearer.scope.claim.name = scope
        sasl.oauthbearer.sub.claim.name = sub
        sasl.oauthbearer.token.endpoint.url = null
        security.protocol = PLAINTEXT
        security.providers = null
        send.buffer.bytes = 131072
        socket.connection.setup.timeout.max.ms = 30000
        socket.connection.setup.timeout.ms = 10000
        ssl.cipher.suites = null
        ssl.enabled.protocols = [TLSv1.2, TLSv1.3]
        ssl.endpoint.identification.algorithm = https
        ssl.engine.factory.class = null
        ssl.key.password = null
        ssl.keymanager.algorithm = SunX509
        ssl.keystore.certificate.chain = null
        ssl.keystore.key = null
        ssl.keystore.location = null
        ssl.keystore.password = null
        ssl.keystore.type = JKS
        ssl.protocol = TLSv1.3
        ssl.provider = null
        ssl.secure.random.implementation = null
        ssl.trustmanager.algorithm = PKIX
        ssl.truststore.certificates = null
        ssl.truststore.location = null
        ssl.truststore.password = null
        ssl.truststore.type = JKS
        transaction.timeout.ms = 60000
        transactional.id = null
        value.serializer = class org.apache.kafka.common.serialization.StringSerializer
```

```
import org.apache.kafka.clients.consumer.Consumer;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.time.Duration;
import java.util.Collections;
import java.util.Properties;

public class ConsumerExample {

    private static final Logger logger = LoggerFactory.getLogger(ConsumerExample.class);

    public static void main(String[] args) {
        Properties properties = new Properties();
        properties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:29092");
//        properties.put(ConsumerConfig.GROUP_ID_CONFIG, "your.group.id");
        properties.put(ConsumerConfig.GROUP_ID_CONFIG, "svilen");
```

```
        properties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
        properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
        properties.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

        Consumer<String, String> consumer = new KafkaConsumer<>(properties);
        consumer.subscribe(Collections.singletonList("bgjug"));

        while (true) {
            ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));

            records.forEach( record -> {
                logger.info("Consumed message - Topic: {}, Partition: {}, Offset: {}, Key: {},
Value: {}",
                        record.topic(), record.partition(), record.offset(), record.key(),
record.value());
            });
        }

    }
}
```



Demo with Spring




```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.kafka:spring-kafka'

    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testImplementation 'org.springframework.kafka:spring-kafka-test'
    testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
}
```

https://spring.io/projects/spring-kafka
https://spring.io/projects/spring-kafka#learn
https://docs.spring.io/spring-kafka/reference/index.html

https://docs.spring.io/spring-kafka/reference/kafka/receiving-messages/listener-annotation.html
https://docs.spring.io/spring-kafka/reference/kafka/sending-messages.html

Не е добре на production Кафка сама да си прави топиците – тази опция трябва да бъде настроена подходящо!!!

2024-08-12T13:36:36.054+03:00 INFO 9752 --- [live-demo] [main] o.a.k.clients.consumer.ConsumerConfig : ConsumerConfig values:

```
	allow.auto.create.topics = true
	auto.commit.interval.ms = 5000
	auto.include.jmx.reporter = true
	auto.offset.reset = earliest
	bootstrap.servers = [localhost:29092]
	check.crcs = true
	client.dns.lookup = use_all_dns_ips
	client.id = consumer-application-3
	client.rack =
	connections.max.idle.ms = 540000
	default.api.timeout.ms = 60000
	enable.auto.commit = false
	exclude.internal.topics = true
	fetch.max.bytes = 52428800
	fetch.max.wait.ms = 500
	fetch.min.bytes = 1
	group.id = application
	group.instance.id = null
	heartbeat.interval.ms = 3000
	interceptor.classes = []
	internal.leave.group.on.close = true
	internal.throw.on.fetch.stable.offset.unsupported = false
	isolation.level = read_uncommitted
	key.deserializer = class org.apache.kafka.common.serialization.StringDeserializer
	max.partition.fetch.bytes = 1048576
	max.poll.interval.ms = 300000
	max.poll.records = 500
	metadata.max.age.ms = 300000
	metric.reporters = []
	metrics.num.samples = 2
	metrics.recording.level = INFO
	metrics.sample.window.ms = 30000
	partition.assignment.strategy = [class org.apache.kafka.clients.consumer.RangeAssignor, class org.apache.kafka.clients.consumer.CooperativeStickyAssignor]
	receive.buffer.bytes = 65536
	reconnect.backoff.max.ms = 1000
	reconnect.backoff.ms = 50
	request.timeout.ms = 30000
	retry.backoff.ms = 100
	sasl.client.callback.handler.class = null
	sasl.jaas.config = null
	sasl.kerberos.kinit.cmd = /usr/bin/kinit
	sasl.kerberos.min.time.before.relogin = 60000
	sasl.kerberos.service.name = null
```

```
sasl.kerberos.ticket.renew.jitter = 0.05
sasl.kerberos.ticket.renew.window.factor = 0.8
sasl.login.callback.handler.class = null
sasl.login.class = null
sasl.login.connect.timeout.ms = null
sasl.login.read.timeout.ms = null
sasl.login.refresh.buffer.seconds = 300
sasl.login.refresh.min.period.seconds = 60
sasl.login.refresh.window.factor = 0.8
sasl.login.refresh.window.jitter = 0.05
sasl.login.retry.backoff.max.ms = 10000
sasl.login.retry.backoff.ms = 100
sasl.mechanism = GSSAPI
sasl.oauthbearer.clock.skew.seconds = 30
sasl.oauthbearer.expected.audience = null
sasl.oauthbearer.expected.issuer = null
sasl.oauthbearer.jwks.endpoint.refresh.ms = 3600000
sasl.oauthbearer.jwks.endpoint.retry.backoff.max.ms = 10000
sasl.oauthbearer.jwks.endpoint.retry.backoff.ms = 100
sasl.oauthbearer.jwks.endpoint.url = null
sasl.oauthbearer.scope.claim.name = scope
sasl.oauthbearer.sub.claim.name = sub
sasl.oauthbearer.token.endpoint.url = null
security.protocol = PLAINTEXT
security.providers = null
send.buffer.bytes = 131072
session.timeout.ms = 45000
socket.connection.setup.timeout.max.ms = 30000
socket.connection.setup.timeout.ms = 10000
ssl.cipher.suites = null
ssl.enabled.protocols = [TLSv1.2, TLSv1.3]
ssl.endpoint.identification.algorithm = https
ssl.engine.factory.class = null
ssl.key.password = null
ssl.keymanager.algorithm = SunX509
ssl.keystore.certificate.chain = null
ssl.keystore.key = null
ssl.keystore.location = null
ssl.keystore.password = null
ssl.keystore.type = JKS
ssl.protocol = TLSv1.3
ssl.provider = null
ssl.secure.random.implementation = null
ssl.trustmanager.algorithm = PKIX
ssl.truststore.certificates = null
ssl.truststore.location = null
ssl.truststore.password = null
ssl.truststore.type = JKS
value.deserializer = class org.apache.kafka.common.serialization.StringDeserializer
```

```
import lombok.RequiredArgsConstructor;
import org.springframework.boot.CommandLineRunner;
```

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.kafka.config.KafkaListenerEndpointRegistry;
import org.springframework.kafka.core.KafkaTemplate;

@SpringBootApplication
@RequiredArgsConstructor
public class LiveDemoApplication implements CommandLineRunner {

    private final KafkaTemplate<String, String> kafkaTemplate; //injecting it

    private final KafkaListenerEndpointRegistry kafkaListenerEndpointRegistry; //injecting it

    public static void main(String[] args) {
       SpringApplication.run(LiveDemoApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
//      for (int i = 0; i < 25_000_000; i++) {
//         kafkaTemplate.send("welcometokafka", "When is next Java beer in Plovdiv?");
//      }
       kafkaListenerEndpointRegistry.getListenerContainer("bgjug").start(); //How to dynamically
start/stop Kafka Listener
    }
}

---------------------

import lombok.extern.slf4j.Slf4j;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Component;

@Slf4j
@Component
public class KafkaConsumer {

//    @KafkaListener(topics = "welcometokafka", groupId = "application", concurrency = "3")
//    @KafkaListener(topics = "welcometokafka", groupId = "application")
    @KafkaListener(id = "bgjug", topics = "welcometokafka", groupId = "application", autoStartup =
"false")
    public void listen(String data) {
        log.info(data);
    }
}

------------------------

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.config.ConcurrentKafkaListenerContainerFactory;
import org.springframework.kafka.core.ConsumerFactory;
import org.springframework.kafka.core.DefaultKafkaConsumerFactory;

import java.util.HashMap;
import java.util.Map;

@Configuration
public class KafkaConsumerConfig {

    @Bean
    public ConsumerFactory<String, String> consumerFactory() {
        return new DefaultKafkaConsumerFactory<>(consumerConfigs());
    }

    @Bean
    public Map<String, Object> consumerConfigs() {
        Map<String, Object> props = new HashMap<>();
```

```
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:29092");
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);

        props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest"); //when app goes down and
then restart

        props.put(ConsumerConfig.MAX_POLL_INTERVAL_MS_CONFIG, 1); //
        props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 100_000); //

        return props;
    }

    @Bean
    ConcurrentKafkaListenerContainerFactory<String, String>
kafkaListenerContainerFactory(ConsumerFactory<String, String> consumerFactory) {
        ConcurrentKafkaListenerContainerFactory<String, String> factory = new
ConcurrentKafkaListenerContainerFactory<>();
        factory.setConsumerFactory(consumerFactory);

        return factory;
    }
}
```

```
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.StringSerializer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.core.DefaultKafkaProducerFactory;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.core.ProducerFactory;

import java.util.HashMap;
import java.util.Map;

@Configuration
public class KafkaProducerConfig {
    @Bean
    public KafkaTemplate<String, String> kafkaTemplate(ProducerFactory<String, String>
producerFactory) { //Injecting the producer factory
        return new KafkaTemplate<String, String>(producerFactory);
    }

    @Bean
    public ProducerFactory<String, String> producerFactory() {
        return new DefaultKafkaProducerFactory<>(producerConfigs());
    }

    @Bean
    public Map<String, Object> producerConfigs() {
        Map<String, Object> props = new HashMap<>();
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:29092");
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        props.put(ProducerConfig.COMPRESSION_TYPE_CONFIG, "gzip");
        // See https://kafka.apache.org/documentation/#producerconfigs for more properties

        return props;
    }
}
```

Quarkus integration

```
<dependency>
  <groupId>io.quarkus</groupId>
```

```
  <artifactId>quarkus-smallrye-reactive-messaging-kafka</artifactId>
</dependency>
```

---

application.properties

```
########################
## Logging configuration
########################
quarkus.log.category."io.zerodt.service.KafkaEventLogProducer".level=INFO
%dev.quarkus.log.category."io.zerodt.service.KafkaEventLogProducer".level=DEBUG
%dev.quarkus.log.category."org.apache.kafka.clients".level=ERROR



########################
## Kafka configuration
########################
%dev.kafka.bootstrap.servers=localhost:9092
%prod.kafka.sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule
required username="${KAFKA_USER}" password="${KAFKA_PASSWORD}";

mp.messaging.incoming.analytics-in.connector=smallrye-kafka
mp.messaging.incoming.analytics-in.topic=analytics
mp.messaging.incoming.analytics-
in.key.serializer=org.apache.kafka.common.serialization.StringSerializer
mp.messaging.incoming.analytics-
in.value.serializer=org.apache.kafka.common.serialization.StringSerializer
mp.messaging.incoming.analytics-in.group.id=wallet-service

mp.messaging.outgoing.analytics.connector=smallrye-kafka
mp.messaging.outgoing.analytics.topic=analytics
mp.messaging.outgoing.analytics.key.serializer=org.apache.kafka.common.serialization.
StringSerializer
mp.messaging.outgoing.analytics.value.serializer=org.apache.kafka.common.serializatio
n.StringSerializer
```

```
import io.smallrye.reactive.messaging.annotations.Blocking;
import io.smallrye.reactive.messaging.kafka.KafkaRecord;
import io.zerodt.bonus.dto.CancelBonusDTO;
import io.zerodt.dto.AccountStatusChangedDTO;
import io.zerodt.dto.UserProfileDTO;
import io.zerodt.exception.BadDataException;
import io.zerodt.service.WalletService;
import io.zerodt.util.ExcludeCodeCoverageGenerated;
import io.zerodt.util.JsonUtils;
import org.apache.kafka.common.header.Header;
import org.eclipse.microprofile.reactive.messaging.Incoming;
import org.jboss.logging.Logger;
import org.jboss.logging.MDC;

import jakarta.enterprise.context.ApplicationScoped;
import jakarta.inject.Inject;
import jakarta.transaction.Transactional;
import java.nio.charset.StandardCharsets;
import java.util.concurrent.CompletionStage;

@ApplicationScoped
public class KafkaEventListener {

    private static final Logger LOGGER = Logger.getLogger(KafkaEventListener.class);

    @Inject
```

```
    WalletService walletService;


    @Incoming("analytics-in")
    @Blocking
    @Transactional
    public CompletionStage<Void> updateProfileStatus(KafkaRecord<String, String> message) {
        final String eventType = getHeaderAsString(message, "eventType");
        if (!"ACCOUNT_STATUS_CHANGED".equals(eventType)) {
            LOGGER.debugf("Received different type of event - %s. Skipping", eventType);
            return message.ack();
        }

        LOGGER.infof("Received new user status changed event. Updating wallet status");
        final AccountStatusChangedDTO statusChange =
JsonUtils.readJsonFromString(message.getPayload(), AccountStatusChangedDTO.class);
        if (statusChange == null) {
            LOGGER.warnf("Received invalid profile JSON: %s", message.getPayload());
            return message.ack();
        }

        MDC.put("profileId", statusChange.playerId());
        try {
            walletService.updateProfileWalletsStatus(statusChange.playerId(),
statusChange.newStatus(), statusChange.previousStatus());
        } finally {
            MDC.clear();
        }
        return message.ack();
    }
}
```

```
import io.smallrye.reactive.messaging.kafka.OutgoingKafkaRecord;
import org.eclipse.microprofile.reactive.messaging.Channel;
import org.eclipse.microprofile.reactive.messaging.Emitter;

import jakarta.enterprise.context.ApplicationScoped;
import jakarta.inject.Inject;

@ApplicationScoped
public class KafkaEventLogEmitter {

    @Inject
    @Channel("analytics")
    Emitter<String> analyticsEmitter;

    public void emmitAnalyticsEvent(OutgoingKafkaRecord<String, String> message) {
        analyticsEmitter.send(message);
    }
}
```

```
public class OutgoingKafkaRecord<K, T> implements KafkaRecord<K, T>


public interface KafkaRecord<K, T> extends Message<T>, ContextAwareMessage<T> {

    static <K, T> OutgoingKafkaRecord<K, T> from(Message<T> message) {
        return OutgoingKafkaRecord.from(message);
    }

    /**
     * Creates a new outgoing Kafka record.
     *
     * @param key the key, can be {@code null}
     * @param value the value / payload, must not be {@code null}
```

```
     * @param <K> the type of the key
     * @param <T> the type of the value
     * @return the new outgoing Kafka record
     */
    static <K, T> OutgoingKafkaRecord<K, T> of(K key, T value)
```

```
import io.smallrye.reactive.messaging.kafka.KafkaRecord;
import io.smallrye.reactive.messaging.kafka.OutgoingKafkaRecord;
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.enterprise.event.Observes;

import jakarta.inject.Inject;
import static jakarta.enterprise.event.TransactionPhase.AFTER_COMPLETION;


@ApplicationScoped
@ExcludeCodeCoverageGenerated
public class KafkaEventLogProducer {

    private static final Logger LOGGER = Logger.getLogger(KafkaEventLogProducer.class);

    @Inject
    KafkaEventLogEmitter emitter;

    public void publishLoyaltyLevelChangeEvent(@Observes(during = AFTER_COMPLETION)
LoyaltyStatusChangeDTO loyaltyStatusChange) {
        Long profileId = loyaltyStatusChange.profileId();

        final String loyaltyStatusChangeMessage = JsonUtils.toJsonString(loyaltyStatusChange);
        final var loyaltyStatusChangeRecord = KafkaRecord.of(profileId.toString(),
loyaltyStatusChangeMessage)
                .withHeader("id", UUID.randomUUID().toString())
                .withHeader("eventType", "LOYALTY_STATUS_CHANGE");

        LOGGER.debugf("Publishing the following message to analytics topic: %s",
loyaltyStatusChangeRecord);
        emitter.emmitAnalyticsEvent(loyaltyStatusChangeRecord);
    }
```

Conclusion

Всеки път/за всяка задача трябват да настройваме ръчно Kafka producer и kafka consumer настройките! Никога да не разчитаме на дефаултните конфигурации!

2. Docker and containerization

2.1. Containers, Docker, Images

Containerization and images

OS-level virtualization refers to an operating system paradigm in which the kernel allows the existence of **multiple isolated** user **space instances** known as **containers, zones, jails, …**

Работят в изолация /Standalone/.
Не може единия апп да източи паметта на другия апп. Ако единия апп му свърши паметта, то другия апп продължава да работи.

Цял сървър е доста скъпо също.
Затова използваме контейнеризация.

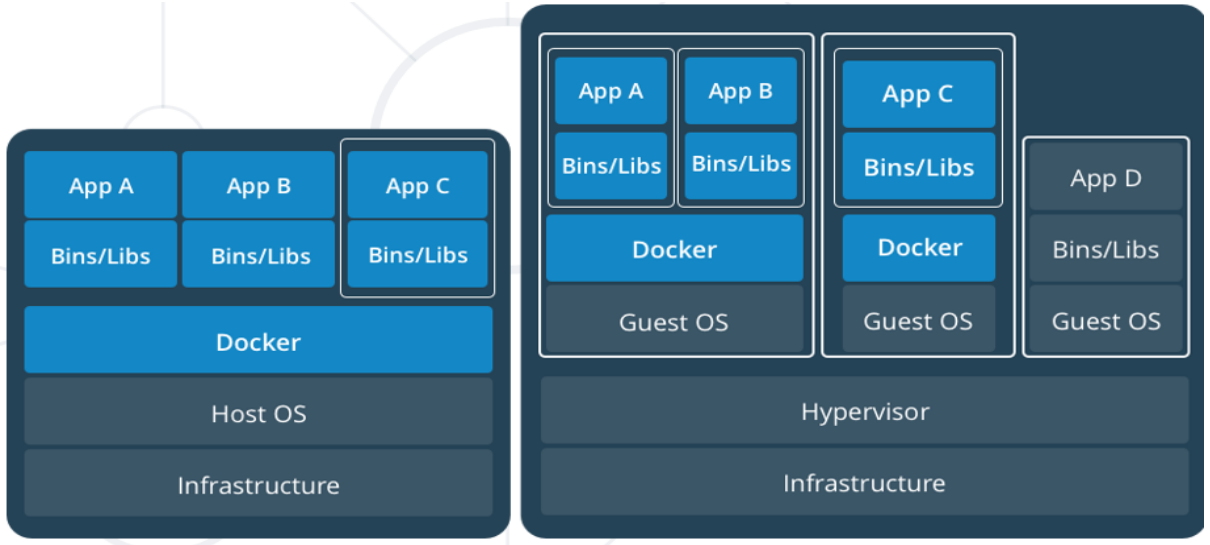
Container:

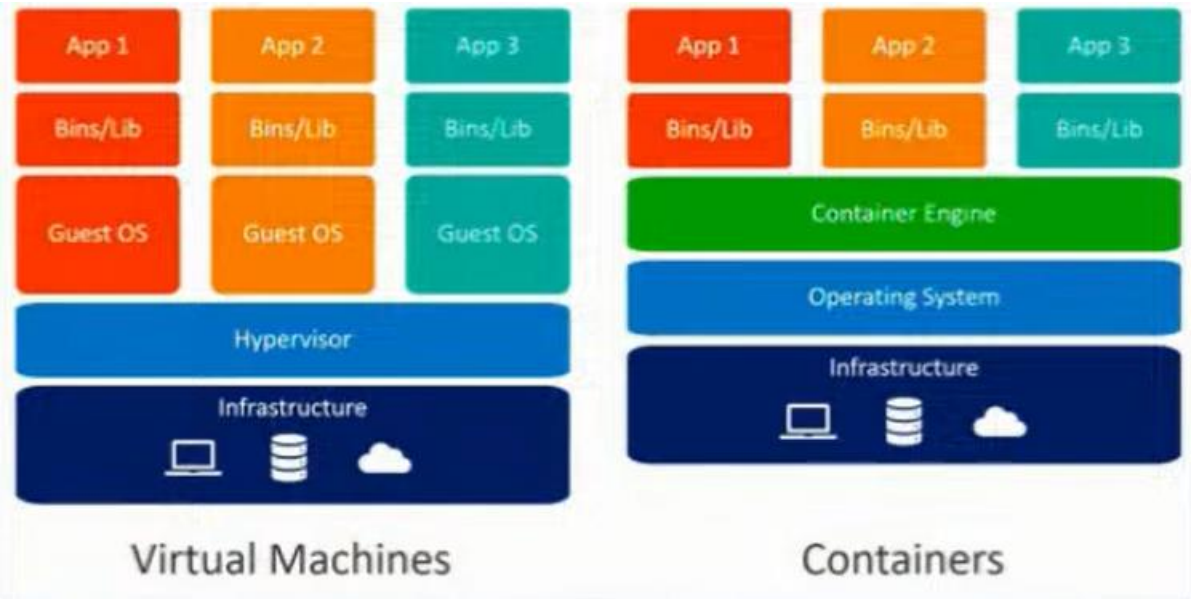
- A sandboxed process that is isolated from all other processes on a host machine
- Should contain only a single application and its dependencies
- **Portable and lightweight**

Image:

- Standalone executable package
- The specification of a container

VMs (виртуални машини) vs Containers





Първоначално е имало само виртуални машини.
Виртуална машина означава на даден хост машина/сървър да има изолирана операционна система, която можем да я ползваме без достъп до някакви други неща.

Идеята, ако има тази хост машина досттъчно ресурси, то да пуснем две незасивими изолирани виртуално операционни системи. Идеята е на всяка виртуална машина да се пуска един application.

Hypervisor – да дистрибутира и менъджира всички виртуални операционни системи и техните ресурси.

Един Docker container е процес, а не цяла виртуална машина! Една хост машина, една операционна система, различни процеси като един от тях е Docker cointainer!

В самият Docker container се казва също че има инсталирана малка операционна система – това всъщност не е съвсем вярно! Има файлова операционна система.

Docker Daemon(Docker engine) менъджира отделните контейнери.

- VMs virtualize the hardware
- Complete isolation
- Complete OS installation. Requires more resources
- Runs almost any OS

- Containers virtualize the OS
- Lightweight isolation
- Shared kernel among all the containers. Requires fewer resources
- Runs on the same OS == Linux

Solutions

- **rkt** by CoreOS
  - Application container engine
  - https://coreos.com/rkt
- **Docker** by Docker Inc
  - Application container engine
  - https://www.docker.com/

Containers Concepts (Docker View)

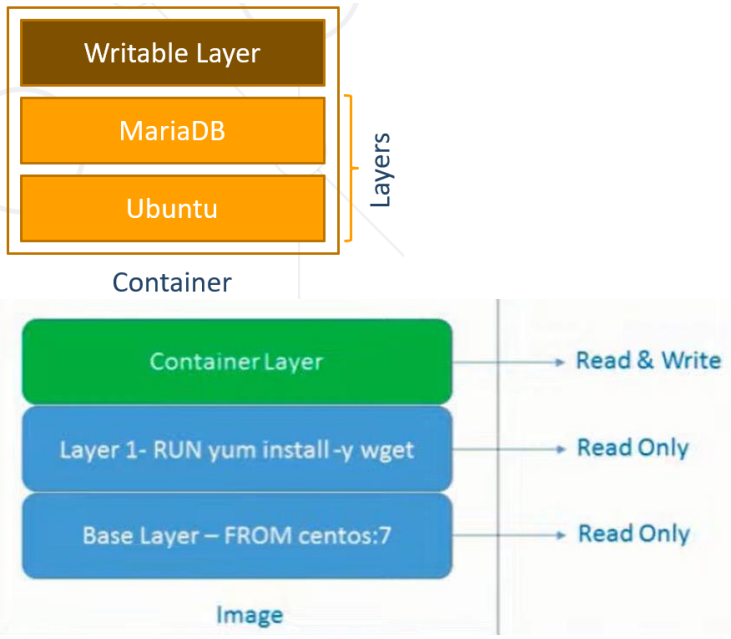
- **Container image** shows the state of a container, including registry or file system changes = все едно **container image** е нещо клас, а самия **контейнер** е нещо като клас. Самият container image е работещ container.
- **Container OS image** is the first layer of potentially many image layers that make up a container = основния image, върху който се гради контейнера
- **Container repository** stores container images and their dependencies = мястото, където се пазят тези container images

Definitions

- Container
  - Containers are processes with much more isolation

- Image
  - Images provide a way for simpler software distribution

Container layers



docker pull nginx

```
❯ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
25d3892798f8: Pull complete
42de7275c085: Pull complete
c459a9332e03: Pull complete
48882f13d668: Pull complete
49180167b771: Pull complete
da4abc2b066c: Pull complete
20dc44ab57ab: Pull complete
Digest: sha256:84c52dfd55c467e12ef85cad6a252c0990564f03c4850799bf41dd738738691f
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

Като дръпнем един докер image, той сваля няколко слоя/layers с няколко id-та, като всеки layer е една от следните команди: FROM, MAINTAINER, RUN, COPY, ADD, EXPOSE, ENTRYPOINT, CMD.

Ако имаме едно и също начало на два докер файла за изпичане на image то при pull-ване на 2 различни Image-а, то те ще се преизползват – няма да ги дърпа два пъти! Т.е. всеки един layer се кешира поотделно и представлява определена част от тази lightweight файлова система. Това също означава, че image-те се оптимизират. Всички layer-и се кешират, с изключение на последния Read & Write layer.

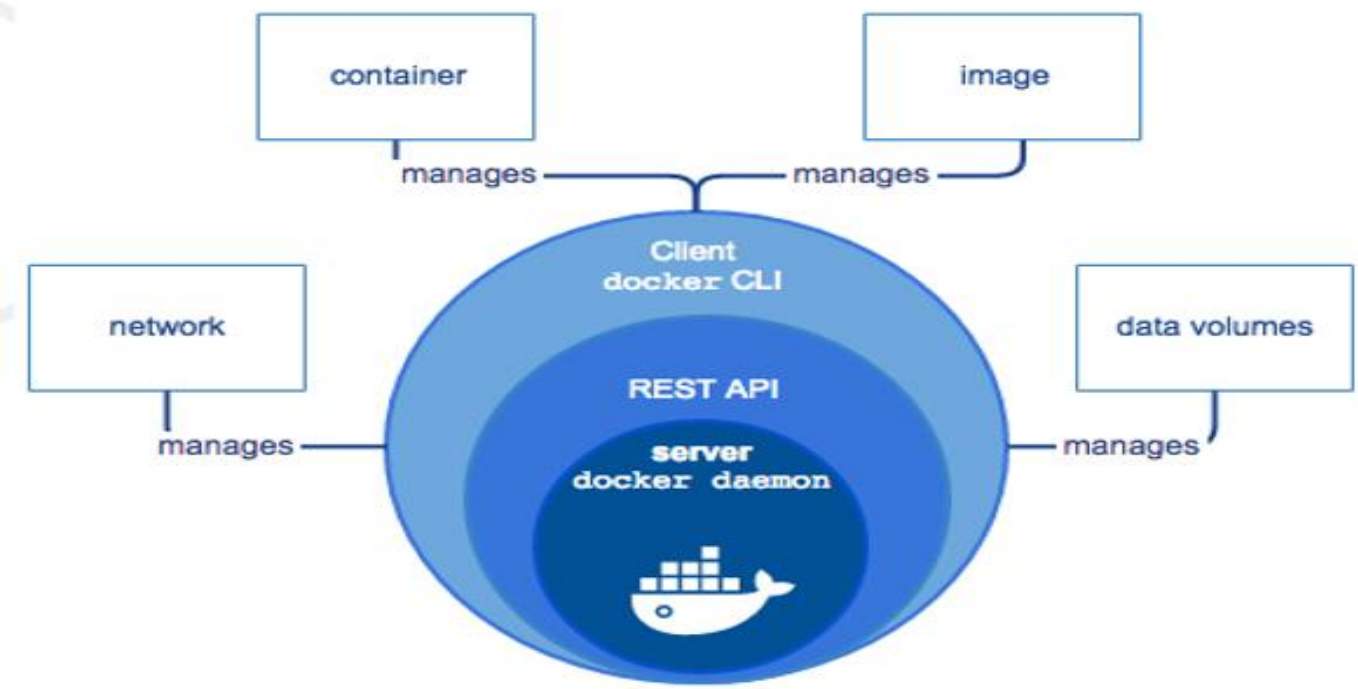
Когато създадем нов контейнер от image, то само тогава се създава и то **наново Read & Write layer-а**.

```
❯ docker pull httpd
Using default tag: latest
latest: Pulling from library/httpd
25d3892798f8: Already exists
7a5857226826: Pull complete
4f4fb700ef54: Pull complete
ec9858119e90: Pull complete
54e79f99f33b: Pull complete
e6520ad62ebb: Pull complete
Digest: sha256:5ee9ec089bab71ffcb85734e2f7018171bcb2d6707f402779d3f5b28190bb1af
Status: Downloaded newer image for httpd:latest
docker.io/library/httpd:latest
```

2.2. Docker

Docker Engine

- Docker Mission – **Build**, **Ship**, **Deploy**
- Client-server application
- Components
    - dockerd daemon
    - REST API
    - docker CLI

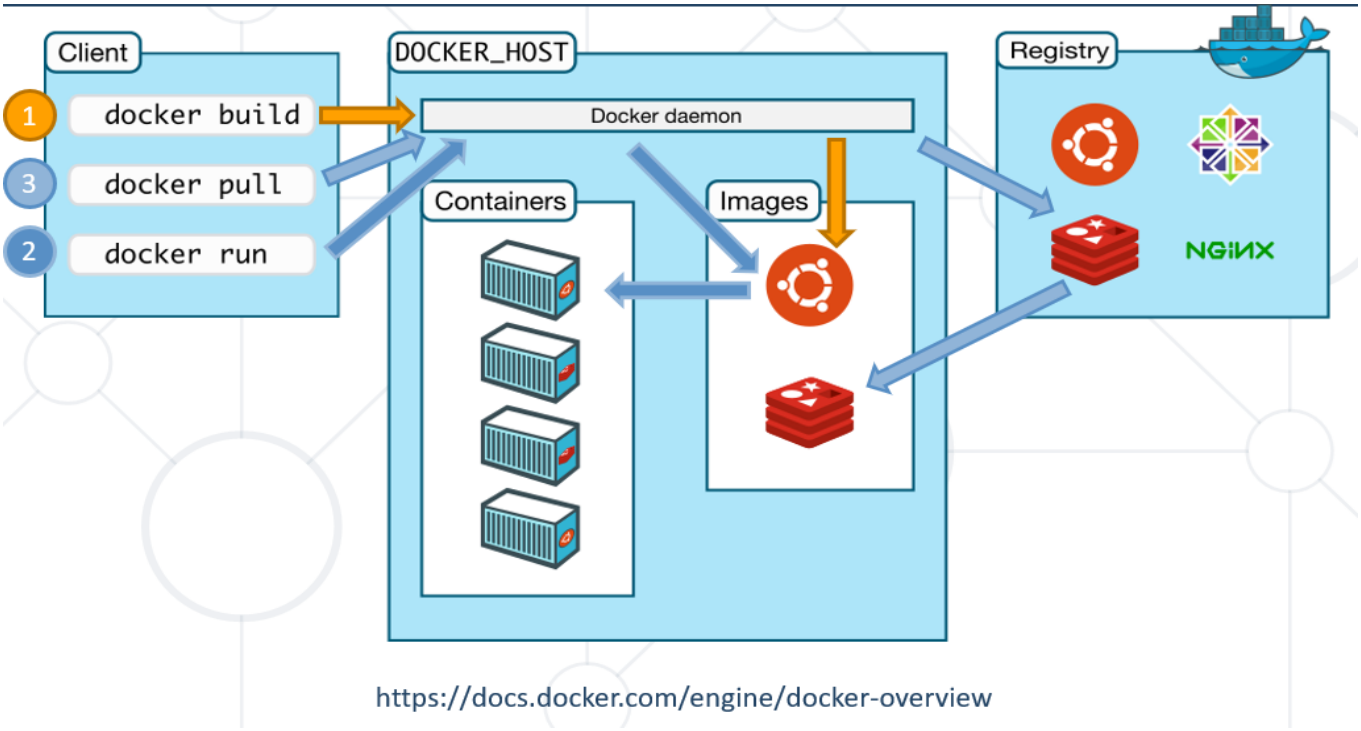


Registries – мястото, където се пазят тези images

- Provided by Docker
    - Cloud
        - Docker Hub (https://hub.docker.com/explore/)
        - Docker Store (https://store.docker.com/)
    - On-premise
- Provided by 3rd parties
    - Quay.io, Artifactory, Google Container Registry

Workflow

docker build – създаване на docker image
docker pull – дърпане на image от регистри
docker run – стартирай ми контейнер
docker push – ако локално ние направим update и искаме нашия image да го дистрибутираме към регистрито.



https://docs.docker.com/engine/docker-overview

Client – Docker CLI (Docker Command Line Interface)
Docker host – това реално е Docker engine-a == Docker daemon
Не е задължително CLI и Docker host да са на една и съща машина. Но в прости дема – това е една и съща машина.

Docker платформата върви само на Линукс. С други думи на Windows и на MacOS пуска виртуална машина на Линукс, за да може да запали Докера.

Registry – място където се споделят images – нещо като MVN repository
https://hub.docker.com/

2.3. Docker Installation

What We Need to Know?

- **Two Editions** (Community and Enterprise)
- **Native Options**
  - Docker for Linux – без Desktop 😊
  - Docker Desktop for MAC
  - Docker Desktop for Windows
- **Docker Toolbox** (deprecated) - All-in-one solution
  - For Mac and Windows

- **Native Options**
  - Docker for Linux
  - Docker for MAC
  - Docker for Windows

→ Deployment via package system (three channels – **stable**, **nightly**, and **test**), script, or archive

Specific requirements: OS version, Hypervisor, etc.

2.4. Working with Docker

Docker е оптимизиран откъм бързина, но не и откъм дисково пространство

Pull / Image Pull

- Purpose
  - Pull an image or a repository from a registry
- Old syntax

docker pull [OPTIONS] NAME[:TAG|@DIGEST]

- New syntax

docker image pull [OPTIONS] NAME[:TAG|@DIGEST]

- Example

docker image pull ubuntu:latest

Run / Container Run

- Purpose
  - Run a command in a new container
- Old syntax

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

docker run -it ubuntu :latest //-it идва от interactive TTY – да имаме достъп отвън до контейнера отвътре
docker run -it ubuntu :latest /bin/bash стартирай ми контейнера с имидж Ubuntu, и стартирай отвътре bash-а
docker run -it nginx самият nginx може да пуска web приложение
docker run -p 8080:80 nginx **-p** опция за forwarding of traffic (port forwarding)

Лявата страна на порта "8080:8080" е на нашата линукс сървърна машина, а дясната страна на порта е каквото е вътре в пуснатия един или няколко контейнера от image nginx.

docker run -d -p 8080:80 nginx **-d** опция за пускане на процеса в background/long run – като се изпълни процеса, то процеса остава и не се самоизчиства!
docker run -d -p 8081:80 nginx

```
❯ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                  NAMES
2b171cfb6d1d   nginx     "/docker-entrypoint.…"   17 seconds ago   Up 16 seconds   0.0.0.0:8081->80/tcp   heuristic_wozniak
44f014f68fe6   nginx     "/docker-entrypoint.…"   About a minute ago   Up About a minute   80/tcp                 heuristic_tharp
5bd2b56352b7   nginx     "/docker-entrypoint.…"   3 minutes ago    Up 3 minutes    0.0.0.0:8080->80/tcp   hardcore_torvalds
```

- New syntax

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

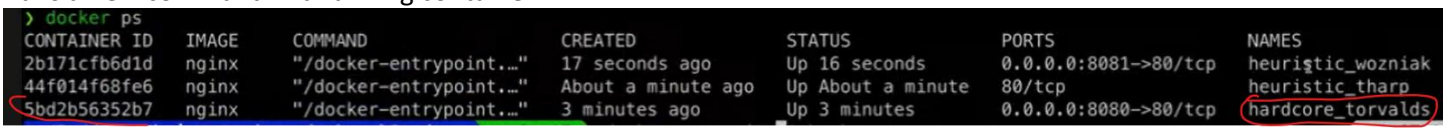
- Example

docker container run -it ubuntu

Exec

Ако имаме няколко контейнера активни, то в нов терминал да си достъпим отвътре някой действащ контейнер примерно.

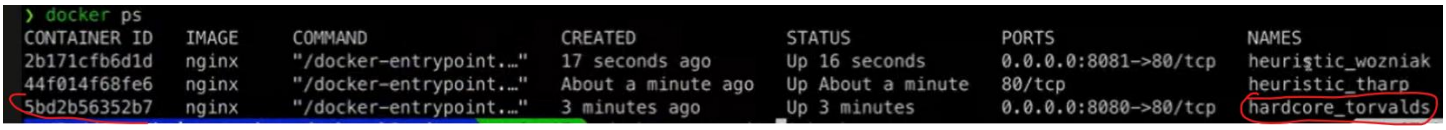
Runs a new command in a running container.



docker exec -it hardcore_torvalds bash

docker exec -it 5bd bash само с част от container id-то стават доста от докер командите

Kill

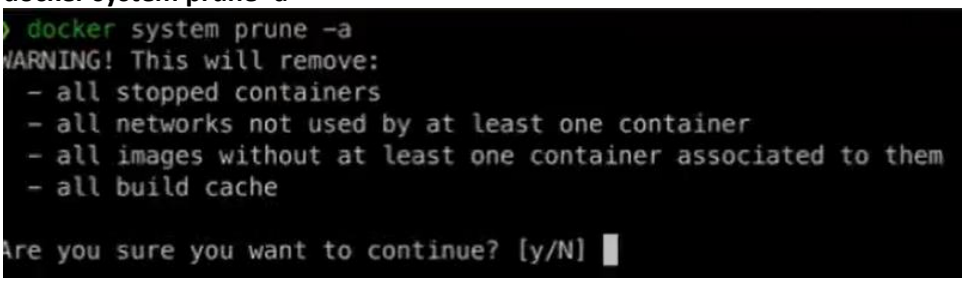


docker kill 5bd само с част от container id-то стават доста от докер командите

Prune

Deletes both containers and images

docker system prune -a



Run container with a specific volume / Bind Mount

docker run -d -p 8080:80 **-v** /tmp/nginx:/user/share/nginx/html nginx

Рънни контейнер от nginx имидж, като копирай всичко от папка /tmp/nginx вътре в контейнера на път /user/share/nginx/html – като **-V** указва, че един вид това е volume. В случая този ред означава **Bind Mount** – директна връзка между външния свят и контейнра вътре, и е по-различно от volume. Volume заема произволно място на хост машината, докато в нашия случай ние сме си направили папка отвън, която да се копира вътре в контейнера.

Images / Image Ls

- Purpose
  - List locally available images
- Old syntax

docker images [OPTIONS] [REPOSITORY[:TAG]]

- New syntax

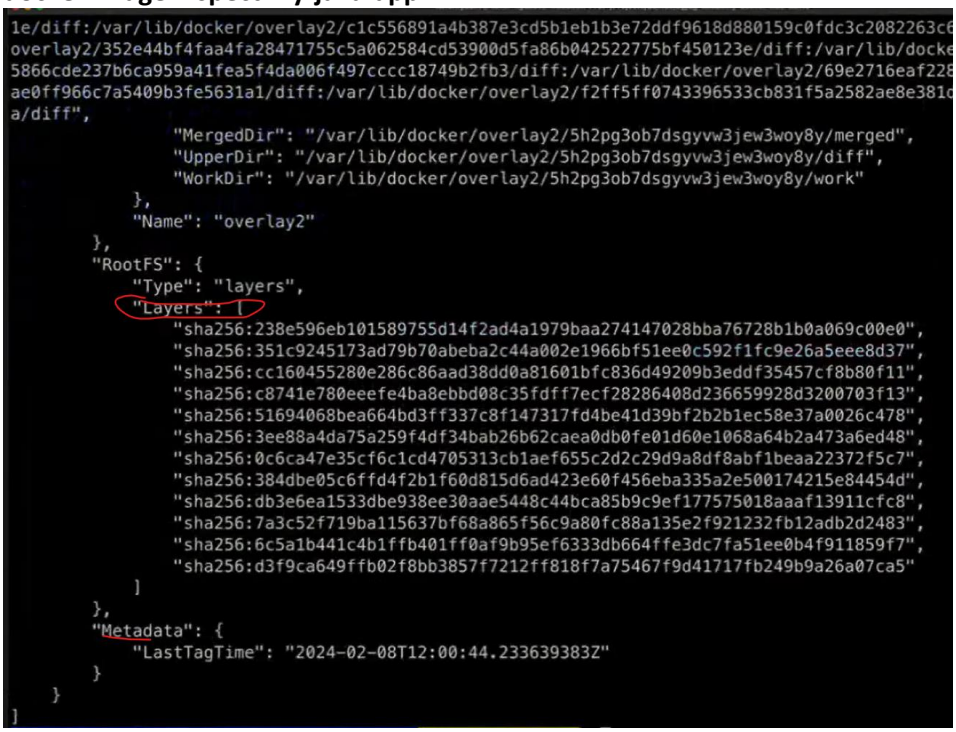
docker image ls [OPTIONS] [REPOSITORY[:TAG]]

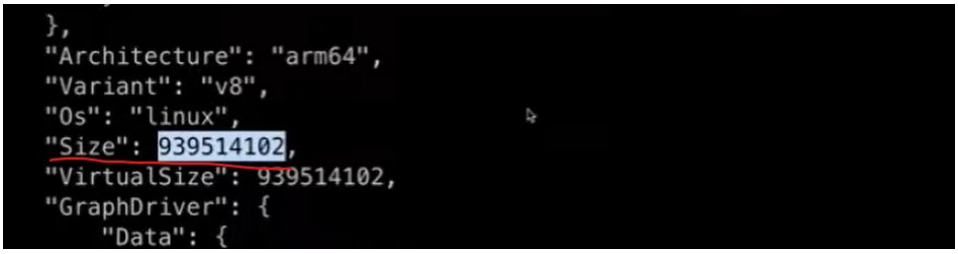
- Example

docker image ls fedora

Image inspect

docker image inspect my-java-app





Ps / Container Ls

- Purpose
  - List containers
- Old syntax

docker ps [OPTIONS]

- New syntax

docker container ls [OPTIONS]

- Example

docker container ls –a -q

Rm / Container Rm

- Purpose
  - Remove one or more containers
- Old syntax

```
docker rm [OPTIONS] CONTAINER [CONTAINER]
```

- New syntax

```
docker container rm [OPTIONS] CONTAINER [CONTAINER]
```

- Example

```
docker container rm weezy_snake
```

Rmi / Image Rm

- Purpose
  - Remove one or more images
- Old syntax

```
docker rmi [OPTIONS] IMAGE [IMAGE]
```

- New syntax

```
docker image rm [OPTIONS] IMAGE [IMAGE]
```

- Example

```
docker image rm ubuntu fedora
```

Start / Container Start

- Purpose
  - Start one or more stopped containers
- Old syntax

```
docker start [OPTIONS] CONTAINER [CONTAINER]
```

- New syntax

```
docker container start [OPTIONS] CONTAINER [CONTAINER]
```

- Example

```
docker container start -a -i 0cbf27183
```

Restart / Container Restart

- Purpose
  - Restart a one or more containers
- Old syntax

```
docker restart [OPTIONS] CONTAINER [CONTAINER]
```

- New syntax

```
docker container restart [OPTIONS] CONTAINER [CONTAINER]
```

- Example

```
docker container restart 0cbf27183
```

Stop / Container Stop

- Purpose
  - Stop one or more running containers
- Old syntax

docker stop [OPTIONS] CONTAINER [CONTAINER]

- New syntax

docker container stop [OPTIONS] CONTAINER [CONTAINER]

- Example

docker container stop 0cbf27183

Unpause / Container Unpause

- Purpose
  - Unpause all processes within one or more containers
- Old syntax

docker unpause CONTAINER [CONTAINER]

- New syntax

docker container unpause CONTAINER [CONTAINER]

- Example

docker container unpause 0cbf27183

Attach / Container Attach

- Purpose
  - Attach to a running container
- Old syntax

docker attach [OPTIONS] CONTAINER

- New syntax

docker container attach [OPTIONS] CONTAINER

- Example

docker container attach 0cbf27183

docker -exec ……

Push / Image Push

- Purpose
  - Push an image or repository to a registry
- Old syntax

docker push [OPTIONS] NAME[:TAG]

- New syntax

docker image push [OPTIONS] NAME[:TAG]

- Example

docker image push repo-name/test:latest

Login to Docker registry

- Purpose
  - Log into a Docker registry
- Old syntax

docker login [OPTIONS] [SERVER]

- New syntax

docker login [OPTIONS] [SERVER] като стария синтаксис

- Example

docker login

Logout from Docker registry

- Purpose
  - Log out from a Docker registry
- Old syntax

docker logout [SERVER]

- New syntax

docker logout [SERVER] като стария синтаксис

- Example

docker logout

2.5. Image from DockerFile – изпичане на docker файл

General Structure (Dockerfile)

- Script, composed of commands and arguments
- Always begins with FROM instruction

 # Set the base image
FROM nginx Comment

 # Set the maintainer
MAINTAINER John Smith Command (Instruction)

 # Copy files
COPY index.html /usr/share/nginx/html/

FROM

- Purpose
  - Defines the base image to use to start the build process
- Syntax

FROM <image>[:<tag>] [AS <name>]

- Example

```
# it is a good practice to state a version (tag)
FROM ubuntu:18.04

# for the latest version the tag could be skipped
FROM ubuntu
```

MAINTAINER

- Purpose
  - Sets the author field of the image. It is deprecated
- Syntax

```
MAINTAINER <name>
```

- Example

```
# deprecated
MAINTAINER John Smith
# newer variant is this:
LABEL maintainer="John Smith"
```

RUN

- Purpose
  - Used during build process to add software (forms another layer)
- Syntax

```
RUN <command>
```

- Example

```
# single command
RUN apt-get -y update
# more than one command
RUN apt-get -y update && apt-get -y upgrade
```

COPY

- Purpose
  - Copy files between the host and the container
- Syntax

```
COPY [--chown=<user>:<group>] <src>... <dest>
```

- Example

```
# Copy single file
COPY readme.txt /root

# Copy multiple files
COPY *.html /var/www/html/my-web-app
```

ADD

- Purpose
  - Copy files to the image

- Syntax

```
ADD [--chown=<user>:<group>] <src>... <dest>
```

- Example

```
# Add single file from URL
ADD https://softuni.bg/favicon.ico  /www/favicon.ico

# Add tar file content
ADD web-app.tar /var/www/html/my-web-app
```

EXPOSE

- Purpose
  - Informs Docker that the container listens on the specified ports
- Syntax

```
EXPOSE <port> [<port>/<protocol>...]
```

- Example

```
# single port
EXPOSE 80

# multiple ports
EXPOSE 80 8080
```

ENTRYPOINT

- Purpose
  - Allows configuration of container that will run as an executable
  - **Задължителната команда, която ще се изпълни в началото**
- Syntax

```
# exec form, this is the preferred form
ENTRYPOINT ["executable", "param1", "param2"]

# shell form
ENTRYPOINT command param1 param2
```

CMD

- Purpose
  - Main purpose is to provide defaults for an executing container
  - **Дефаултната команда, която ще се изпълни в началото**
- Syntax

```
# exec form, this is the preferred form
CMD ["executable","param1","param2"]

# as default parameters to ENTRYPOINT
CMD ["param1","param2"]

# shell form
CMD command param1 param2
```

CMD vs ENTRYPOINT

- Both **define** what **command** gets **executed** when **running** a container
- **Dockerfile** should specify at **least one** of them
- **ENTRYPOINT** should be defined when using the **container** as an **executable**
- **CMD** should be used as a way of defining **default arguments** for an **ENTRYPOINT** command or for **executing** an **ad-hoc** command in a container
- **CMD** will be overridden when **running** the container with **alternative** arguments

- Both have **exec** and shell **form**
- When used **together always** use their **exec** form

Когато се използват заедно или поотделно:
CMD командата може да заема няколко аргумента

		ENTRYPOINT		
		N/A	**exec_entry p1_entry**	**["exec_entry", "p1_entry"]**
CMD	**N/A**	Error	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry
	["exec_cmd", "p1_cmd"]	exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	**exec_entry p1_entry exec_cmd p1_cmd**
	["p1_cmd", "p2_cmd"]	p1_cmd p2_cmd	/bin/sh -c exec_entry p1_entry	**exec_entry p1_entry p1_cmd p2_cmd**
	exec_cmd p1_cmd	/bin/sh -c exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd

Build / Image Build

- Purpose
  - Build an image from a Dockerfile
- Old syntax

docker build [OPTIONS] PATH | URL | -

- New syntax

docker image build [OPTIONS] PATH | URL | -

- Example - **-t** означава сложи му таг, а точката е текущата директория

docker image build -t new-image .

Ако създаваме локално docker image на M1, M2, M3 MacOS компютър (ARM или ARM64 процесорна архитектура), то би имало проблем този имидж като го качим в регистрито и други потребители си го изтеглят и пуснат контейнер на обикновен Линукс. Затова ако сме на MacOS пускаме следната команда и тогава е доста голям шанса да работи:
docker build buildx –platform linux/amd64 my-java-app .

Доста често в практиката тези docker images се билдват от Jenkins/GitHub actions агенти, и тъй като се билдват (чрез buildAaA.yml) на Линукс сървър, тогава го нямаме този проблем с ARM64! Т.е. преместваме създаването/изпичането на докер image в CI/CD pipeline-а.

Dockerfile example workflow

Example 1:
Dockerfile-build

```
# We start from openjdk17 base image to have JDK installed
FROM eclipse-temurin:17-jdk
WORKDIR /app

COPY build/libs/petclinic-0.0.1-SNAPSHOT.jar app.jar   //копирай jar файла като
app.jar вътре в контейнера

CMD [”java”, ”-jar”, ”app.jar”]     //Read & Write layer
```

In the terminal:
docker build -t my-java-app . **-t** означава сложи му таг, а точката е текущата директория

docker run -p -d 8080:8080 **my-java-app**

Example 2:

Dockerfile-build

```
# We start from openjdk17 base image to have JDK installed
FROM eclipse-temurin:17-jdk
WORKDIR /app      //same as RUN mkdir app && cd app

//организираме layers по честота на промяна – т.е. src ще се променя често, тогава
измести копиранията за gradle по-горе – това ще ги сложи в кеша за следващ image
build ще се изпълнява по-бързо
COPY gradlew .          //копирай gradlew файла вътре в контейнера
COPY gradle gradle/      //копирай съдържанието на gradle папката вътре в контейнера
в директория gradle/
COPY build.gradle.kts .
COPY settings.gradle.kts .

COPY src src/            //копирай съдържанието на src папката вътре в контейнера в
директория src/
RUN ./gradlew bootJar

EXPOSE 8080

CMD sh -c 'java -jar build/libs/*.jar'     //Read & Write layer
```

In the terminal:
docker build -t my-java-app . **-t** означава сложи му таг, а точката е текущата директория

След като image-а е изпечен/създаден, то може да пуснем container от него:
docker run -it my-java-app /bin/bash

Example 3 – with maven offline:

Dockerfile-build

```
# We start from openjdk17 base image to have JDK installed
FROM eclipse-temurin:17-jdk
WORKDIR /app      //same as RUN mkdir app && cd app

COPY maven???*** .

# 3 layers runs
```

```
RUN apt update
RUN apt install java
RUN apt install python

# preferable to be only 1 layer instead of 3 layers!!!
RUN apt update && apt install java && apt install python

RUN mvn dependencies:go-offline

COPY src src/

RUN ./mvnw package

EXPOSE 8080

CMD sh -c 'java -jar build/libs/*.jar'
```

Example 4 – multistage

Заема по-малко място самият готов изпечен image

Dockerfile-multistage

```
# phase 1
FROM eclipse-temurin:17-jdk AS build
WORKDIR /app

COPY gradlew .
COPY gradle gradle/
COPY build.gradle.kts .
COPY settings.gradle.kts .

COPY src src/
RUN ./gradlew bootJar

# phase 2 - run the jar file
FROM eclipse-temurin:17-jre
WORKDIR /app

COPY --from=build /app/build/libs/*.jar app.jar

EXPOSE 8080
CMD sh -c 'java -jar app.jar'
```

docker build -t "petclinic" -f Dockerfile-multistage .

Recommendations

- Don’t create large images
- Don’t use only the “latest” tag
- Don’t run more than one process in a single container
- Don’t rely on IP addresses
- Put information about the author

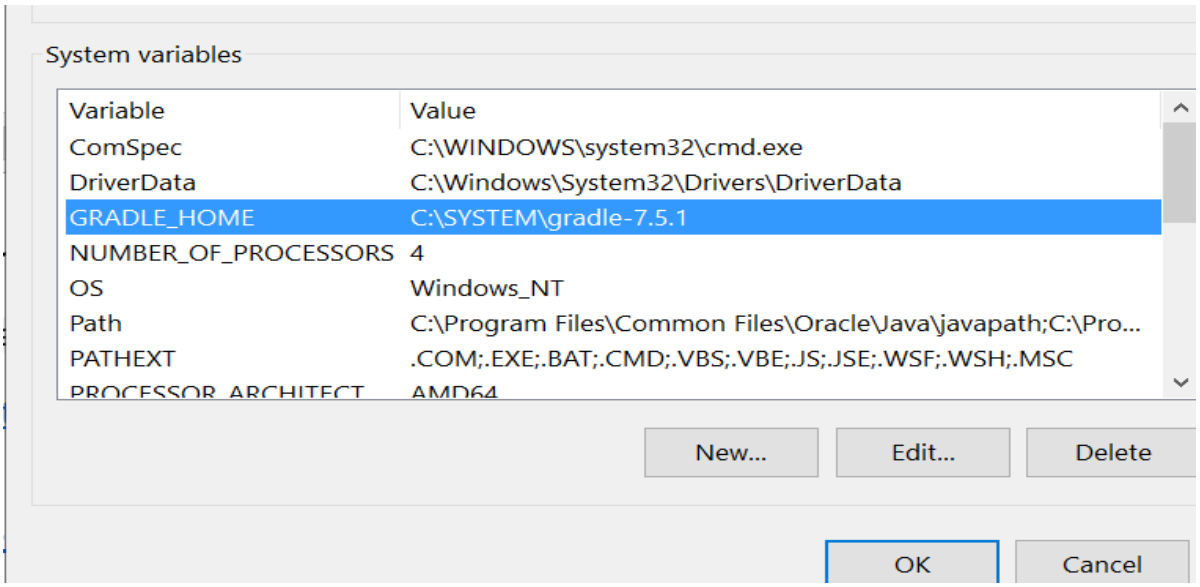
2.6. Demo summary + Gradle Wrapper and Gradle for Windows

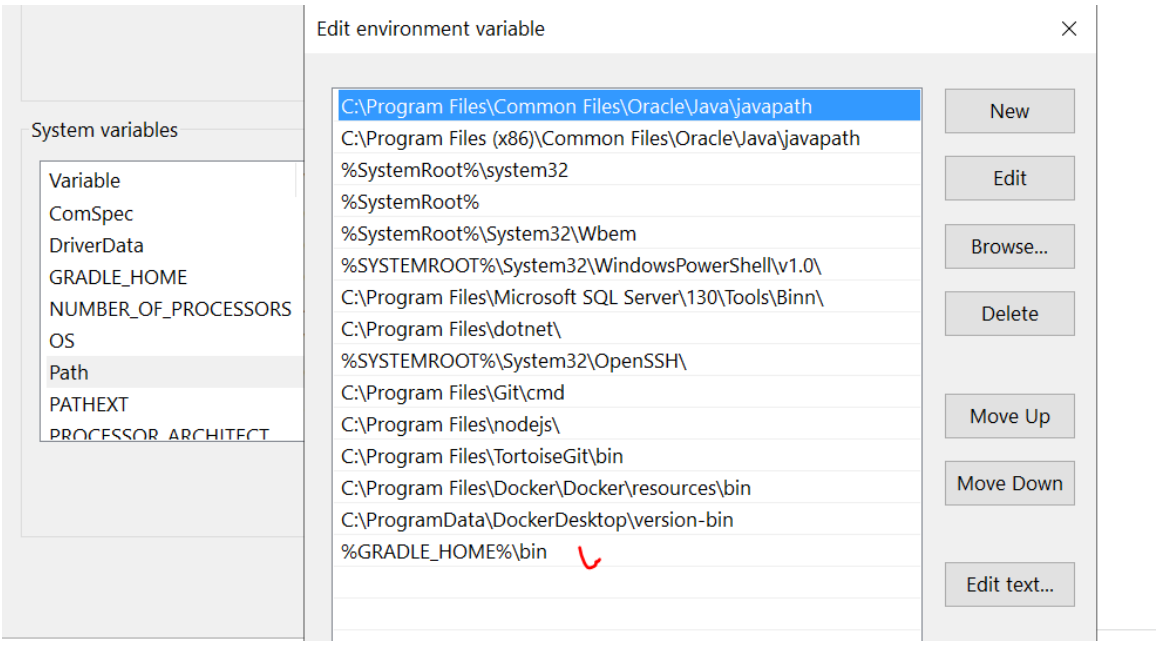
Gradle

https://tomgregory.com/what-is-the-gradle-wrapper-and-why-should-you-use-it/

The Gradle wrapper is a script you add to your Gradle project and use to execute your build. The advantages are:

you don’t need to have Gradle installed on your machine to build the project
the wrapper guarantees you’ll be using the version of Gradle required by the project
you can easily update the project to a newer version of Gradle, and push those changes to version control so other team members use the newer version



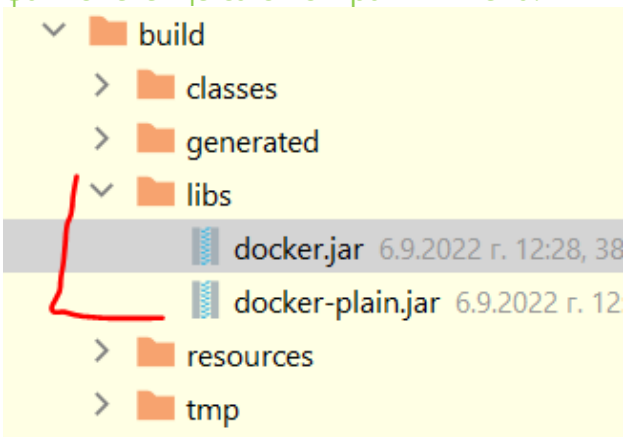



```
gradle init
```

Running a build

`gradle build`
`gradle clean build`

When you run the **build** command you'll see output like this.
This means that your application was assembled and any tests passed successfully. Gradle will have created a build directory if it didn't already exist, containing some useful outputs you should be aware of.

1. **the *classes* directory** contains compiled .class files, as a result of running the Java compiler against your application Java source files
2. **the *libs* directory** contains a generated jar file, an archive with all your compiled classes inside **ready to be executed or published** – ако махнем версията от build.gradle, то .jar файловете ще са с по-кратки имена.
3. 

4. **the *reports* directory** contains an HTML report summarising your test results. If your build fails, then consult this report to see what went wrong.

You now know that running the Gradle build command is equivalent to running the build task, which you do in Windows with `gradlew build` and on Linux/Mac with `./gradlew build`. The *build* task depends on other tasks, and in some situations it may make sense to run a more specific task.

След което можем да го пуснем/run-нем тези jar
java -jar ./build/libs/docker.jar

За да създадем .jar файл с Gradle, то тестовете трябва да минават

Maven
За да създадем .jar файл с Maven, то тестовете трябва да минават

Файлът /deployment/Dockerfile:


```
FROM openjdk:11-jre

VOLUME /tmp
COPY build/libs/docker.jar app.jar
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom -Xms256m -Xmx512m", "-jar","/app.jar"]
```

Ако използваме Maven, тези думички били различни
build/libs

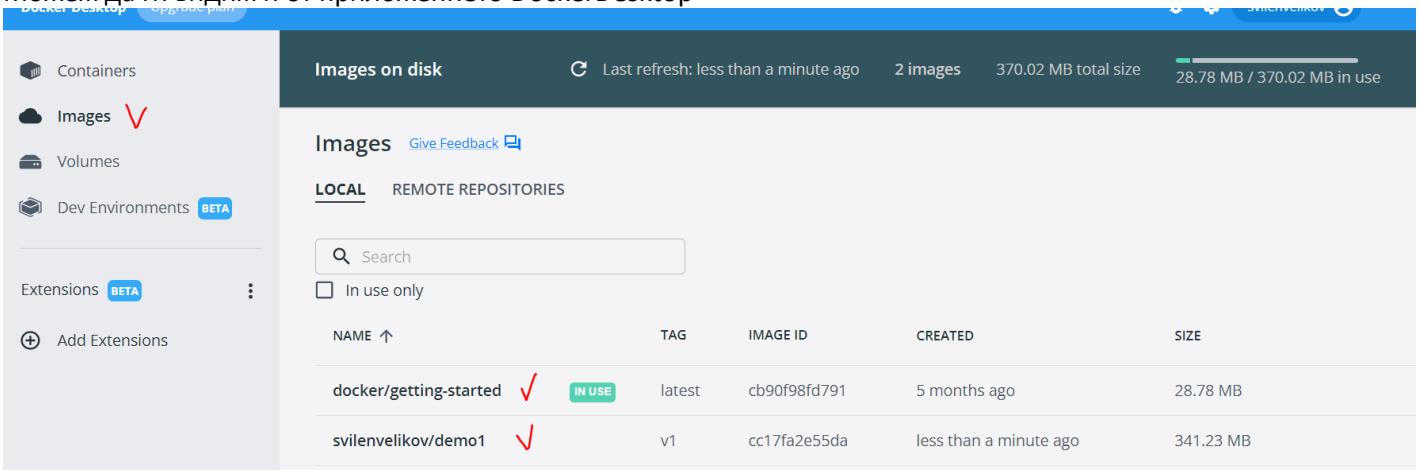
От основната директория на проекта ни, под CMD на Windows пускаме следната команда
C:\Temp projects\Spring Advanced\12 Containerization & Documentation\docker_example>
docker build -t svilenvelikov/demo1:v1 -f deployment/Dockerfile .
създай от .jar файла **image** с име demo1 версия 1 на user svilenvelikov и от текущата директория която е точка, изпълни Dockerfile.

docker images

в терминала показва всички имиджи качени на докер към момента

```
C:\Temp projects\Spring Advanced\12 Containerization & Documentation\docker_example>docker images
REPOSITORY                TAG       IMAGE ID       CREATED         SIZE
svilenvelikov/demo1       v1        cc17fa2e55da   5 minutes ago   341MB
docker/getting-started    latest    cb90f98fd791   4 months ago    28.8MB
```

Можем да ги видим и от приложението DockerDesktop



I) Пускане на docker image-а

C:\Temp projects\Spring Advanced\12 Containerization & Documentation\docker_example>
docker run -p 8080:8080 svilenvelikov/demo1:v1

Когато image-а се run-не, то той става контейнер.

docker ps

покажи текущо активни контейнери

Качи image-а на облака docker repositories

docker push svilenvelikov/demo2:v2

останалите images са част от jdk и няма нужда да се пушват

```
PS C:\Temp projects\Spring Advanced\12 Containerization & Documentation\docker_example> docker push svilenvelikov/demo2:v2
The push refers to repository [docker.io/svilenvelikov/demo2]
4ced9a8f798d: Pushed
5a7e7a880634: Mounted from library/openjdk
3dccaa93bb0e: Mounted from library/openjdk
5c384ea5f752: Mounted from library/openjdk
293d5db30c9f: Mounted from library/openjdk
03127cdb479b: Mounted from library/openjdk
9c742cd6c7a5: Mounted from library/openjdk
v2: digest: sha256:3232e698c894f814aea10bb0e5ecbd61af3562e9b8bdea73dab5cb4d49acc547 size: 1794
```

Remove-ване на спрян контейнер

При опит да изтрием docker image:
Error response from daemon: conflict: unable to delete **5d4cfe7396db** (must be forced) - image is being used by stopped container **d61b2cca0024**
PS C:\Temp projects\Spring Advanced\12 Containerization & Documentation\docker_example> **docker rm d61b2cca0024**

Remove-ване на самия docker image

PS C:\Temp projects\Spring Advanced\12 Containerization & Documentation\docker_example> **docker rmi 5d4cfe7396db**

Untagged: svilenvelikov/demo2:v2
Untagged: svilenvelikov/demo2@sha256:3232e698c894f814aea10bb0e5ecbd61af3562e9b8bdea73dab5cb4d49acc547
Deleted: sha256:5d4cfe7396db1e26c28c46c664427ad0041b8afa23c23aeb5c8f4b28091e9ac6

Търси първо локално, след това търси от облака

Ако пуснем команда за рънване на изтрития локално image, то първо търси локално, след това търси на облака docker repositories
docker run -p 8080:8080 svilenvelikov/demo2:v2

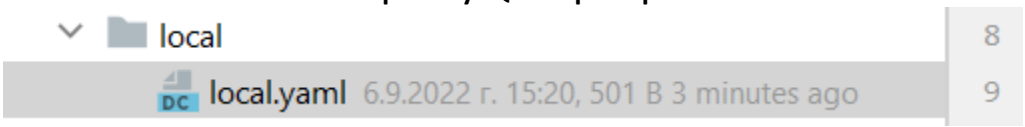
PS C:\Temp projects\Spring Advanced\12 Containerization & Documentation\docker_example> docker run -p 8080:8080 svilenvelikov/demo2:v2
Unable to find image 'svilenvelikov/demo2:v2' locally
v2: Pulling from svilenvelikov/demo2
001c52e26ad5: Already exists
d9d4b9b6e964: Already exists
2068746827ec: Already exists
8510da692cda: Already exists
b6d84395b34d: Already exists
bf03fea6c3ad: Already exists

7ac2c538825b: Already exists
Digest: sha256:3232e698c894f814aea10bb0e5ecbd61af3562e9b8bdea73dab5cb4d49acc547
Status: Downloaded newer image for svilenvelikov/demo2:v2

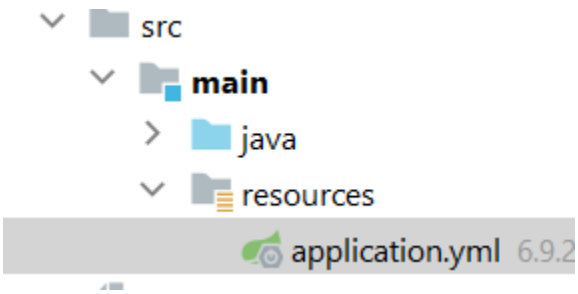
II) Пускане на повече docker images - как 2 image-а си говорят когато са качени на Docker

PS C:\Temp projects\Spring Advanced\12 Containerization & Documentation\docker_example>
Docker Compose is **a tool that was developed to help define and share multi-container applications**. With Compose, we can create a YAML file to define the services and with a single command, can spin everything up or tear it all down.
docker-compose -f ./local/local.yaml up
docker-compose -f ./local/local.yaml down

Ако нямаме локално инсталиран MySQL например




```yaml
version: '3.3'
services:
  db:
    image: arm64v8/mysql:oracle
    ports:
      - "3306:3306"
    command: ['--character-set-server=utf8mb4', '--collation-server=utf8mb4_bin', '--default-authentication-plugin=mysql_native_password']
    environment:
      - MYSQL_ALLOW_EMPTY_PASSWORD="yes"
      - MYSQL_DATABASE=intro
  docker-demo:
    image: svilenvelikov/docker-demo:v3
    ports:
      - "8080:8080" # web ui
    environment:
      - MYSQL_HOST=db
      - MYSQL_USER=root
      - MYSQL_PASSWORD=
```



application.yml

```yaml
spring:
  datasource:
    driverClassName: com.mysql.cj.jdbc.Driver
    url: "jdbc:mysql://${MYSQL_HOST:localhost}:3306/demodocker?useSSL=false&createDatabaseIfNotExist=true&serverTimezone=UTC"
```

```
username: "${MYSQL_USER:root}"
password: "${MYSQL_PASSWORD:}"
```

III) Пускане на docker image с CMD опция

При стартиране на docker image-а да прави ping
При първата команда ping-ва от локалния сървър localhost/от локалния компютър.
При втората команда овъррайдваме CMD-то да бъде google.com и сега ping-ва от google.com
ENTRYPOINT не може да се override-не

```
FROM debian:wheezy
ENTRYPOINT ["/bin/ping"]
CMD ["localhost"]

$ docker run -it test
PING localhost (127.0.0.1): 48 data bytes

$ docker run -it test google.com
PING google.com (173.194.45.70): 48 data bytes
```

3. Kubernetes and orchestration

History Overview

- At the beginning software was mostly installed **on prem** (on physical hardware)
- Then the cloud paradigm started emerging and **hypervisors** like VirtualBox and HyperV became important players in that area
- However virtualization posed significant limitations due to the strict isolation of VMs in terms of resources (CPU and memory)

- The evolution of hardware and operating systems provided the better possibilities to run applications as separate isolated processes
- An example of such possibilities are **namespaces** and **cgroups** in the Linux Kernel – изолираме ресурсите като памет и процесор за всеки процес без да е нужно външно приложение като **hypervisor**
- This enabled the implementation of container solutions like Docker

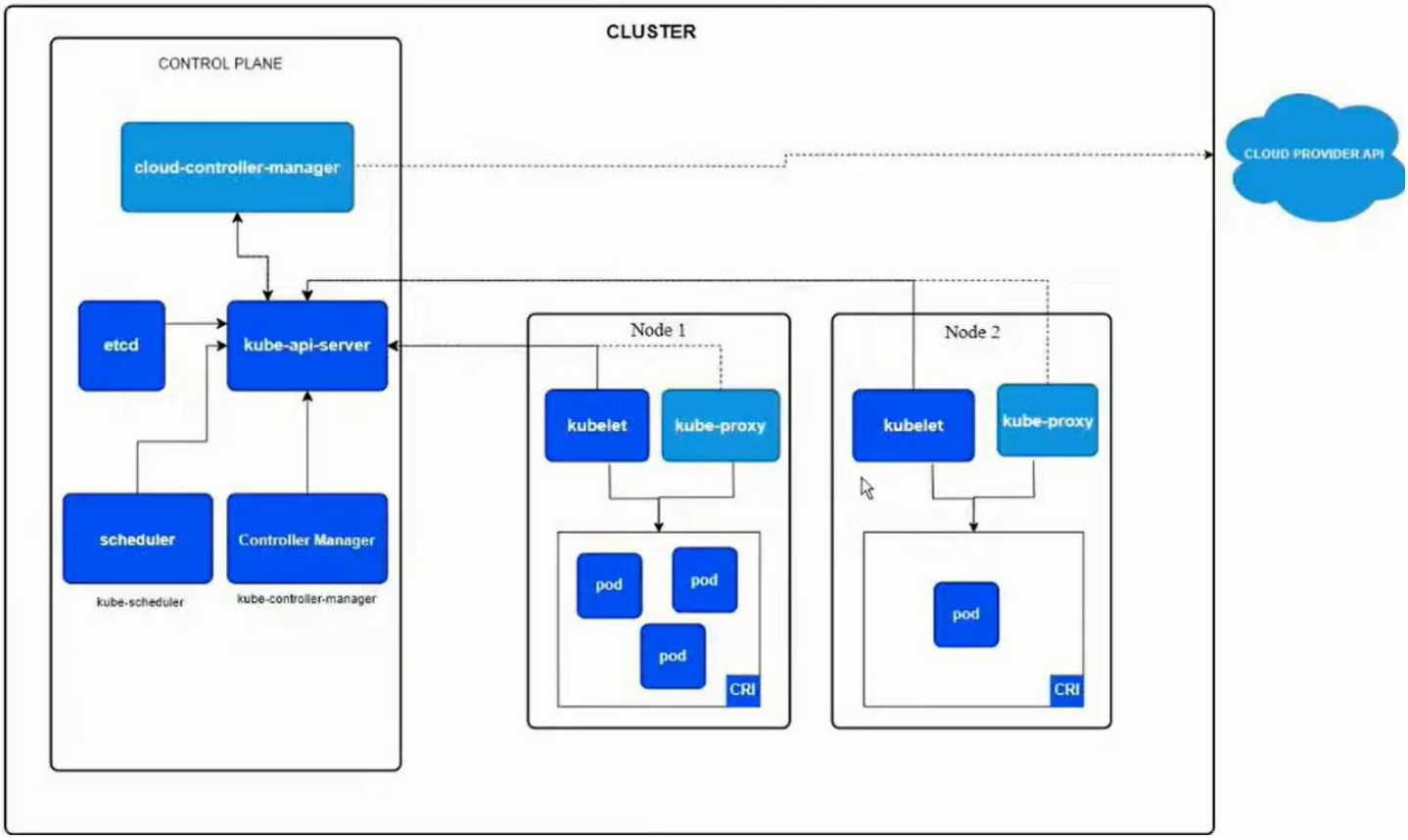
- However, the use of container technology like Docker still implied a lack of good tooling to automate the process of management and orchestration of the containers:
    - How do we manage configuration of containers?
    - How do we scale up/down using containers? (i.e. Docker Swarm)
    - How do we automate deployment of a group of containers? (i.e. Ansible scripts or Docker compose)
    - How do we manage the administrative aspects of a container (i.e. networking, traffic redirection, security, resource allocation, etc…)

- Due to the increasing needs to optimize resources utilization and management aspects of containers Google developed Borg as an internal large-scale cluster management system
- Borg allowed Google to run thousands of jobs from a number of applications
- Borg was later succeeded by Omega as a second generation cluster management system which emerged at about the same time as Docker

- Thus Kubernetes was born out of the idea to combine the possibilities of a container technology like Docker with a scalable cluster management system like Borg….

Architecture

https://kubernetes.io/docs/concepts/architecture/



Control Plane – на отделна физическа машина

etcd – разпределена файлова система, която се използва за да persist-ва цялата информация в Kubernetes cluster-а.

kube-api-server – rest interface, с който можем да менъджираме различните елементи в Kubernetes клъстера. На Java, C#, Python, и т.н. Този kube-api-server менъджира също node-вете.

node – виртуална машина или физическа машина или набор от няколко мощни сървъра - машина на която да се деплойват **pod**-вете

kubelet – комуникира двустранно – с kube-api-server и с pod-вете. Като менъджира pod-вете. С други думи kube-api-server казва на kubelet-а стартирай ми pod с тези и тези контейнери на този node в клъстера.

kube-proxy – се грижи да ротира трафика към pod-овете в рамките на node-а.

scheduler – се грижи за стартирването на pod-вете в node-вете – като например кой node има достатъчно ресурси в момента да стартира дадения pod.

Controller Manager – се грижи да изпълнява различни процеси/задачи – като си минава по пътя kube-api-server -> kubelet:

- като например деплоймент на приложение – еднократни job-ве от рода на стартирай нов node и му сложи първоначални pod-ве.
- Ако даден node не е reachable, то да го маркира да не се деплойват на него pod-ве в момента.
- Replicaset-controller – да се погрижи да има поне една инстанция например за nginx **pod**-а

Cloud-controller-manager – API, което може да се имплементира от различни Cloud providers (GoogleCloudProvider, AmazonWebServices). Дава възможност да се менъджират различни дейности в рамките на съответния cloud provider, последния който деплойва Kubernetes cluster.

Pod

- The basic unit of operation in Kubernetes is a **pod**
- A **pod** may contain one or more containers (Docker containers for example)
- **Pods** are logically grouped in Kubernetes **namespaces** – не е задължително да използваме namespaces, но когато организацията е голяма определни хора или тиймове да имат определени права свързани с този namespace
- **Namespaces** provide isolation of resources, users and permissions
- За всеки един pod се алокира отделен IP address

Workload

- Workloads are a logical grouping of pods that form an application
- Workloads in Kubernetes can be different types – in yaml syntaxis:
    - ReplicaSets – как да репликираме даден pod
    - Deployments – описание как можем да деплойваме pod
    - StatefulSets
    - DaemonSets

Services

Контейнерите на всеки pod могат да стартират на определени портове. Но за всеки отделен pod се алокира отделен IP address.

Всички pod-s от даден node са организирани в т.нар. клъстерна мрежа.
За да може един pod да достъпи сървиси от друг pod, то не е видимо как да стане от самите контейнери! За да може това нещо да се реализира, то имаме т.нар. сървиси/services.

- As the name implies: provides the possibility to define a ‘service’ on top of pods
- The service has an IP address, port and a DNS name
- The service provides the possibility to route traffic to pods either from inside the Kubernetes cluster or from outside
- A service is also described in yaml file

Ingress

Ако искаме да ротираме трафик между подовете на базата на конкретни правила.

- Ingress elements provide the possibility to route traffic to pods
- It can be configured on a rule-basis specifying how traffic coming to the cluster can be routed to pods
- Ingress routes traffic to services – тези правила са вече декларирани в **service** yaml файловете.

ConfigMaps

- ConfigMaps are the main mechanism used to supply configuration to pods
- That configuration can be passed in either of two ways from a ConfigMap to a pod:
    - As environment variables
    - Mounted on a pod container's file system

Secrets

- Secrets like ConfigMaps can be used to store configuration data
- Secrets are intended to store **sensitive data (such as passwords and certificates)**
- Data stored as Kubernetes secrets is not encrypted-at-rest by default
- Secrets can also be derived from external access management applications like HashiCorp Vault or Keycloak, etc.

Имаме няколко различни механизма, с които можем да интегрираме Keycloak в Kubernetes:

- Ако сме на Spring/Quarkus проект – самият проект ни дава възможност да се интегрираме с Keycloak
- Kubernetes може да извлича данни от Keycloak и да ги записва като Kubernetes secrets

Volumes

- Containers provide by default temporary storage that is wiped out on container restart (in the case of Kubernetes – a pod restart)
- Kubernetes provides the possibility to create and manage persistent volumes used by the pods
- Volumes can be mounted to the file system on a pod container – например да пазим данни в база данни

Deployment and tools

One local instance of Kubernetes

- Deploying a Kubernetes cluster form scratch is not a trivial task…
- For that purpose for local development there are a number of options available like:
    - **minikube – една инстанция на една виртуална машина или на един докер контейнер**
    - Kind
    - K3s
    - MicroK8s
    - Kubeadm
    - DockerDesktop with installed also Kubernetes plugin run in a docker container

API clients

- **kubectl** – standard (and most used) CLI client for Kubernetes

kubectl create deployment nginx --image nginx //create deployment е по-скоро за локално тестване
kubectl apply -f deployment.yaml
kubectl get pods -n somenamespace
kubectl describe pod pod_id -n somenamespace какви контейнери и други неща в дадения под
kubectl logs pod_id -n somenamespace по подразбиране се листват логовете на първия контейнер от пода
kubectl logs pod_id -c containerid -n somenamespace

- language API clients (i.e. for Java)
- other tools – like **k9s**

Web UIs

Different third party web interfaces for managing Kubernetes:

- **Kubernetes Dashboard** (standard and most popular)
- Kubernetes Lens (IDE for Kubernetes)
- Octant
- others

Package management

- **Helm** is a widely used package manager for Kubernetes
- It is used to build distributable **charts** that are packages of Kubernetes resources
- Charts са група от Kubernetes ресурси – например deploy.yaml, който деплойва **pod** с nginx и Spring приложение с/на 3 инстанции, services, ConfigMaps, и т.н.
- Тези **charts** можем да ги създаваме, да ги качваме в централно Helm хранилище, и да бъдат преизползвани в различни Kubernetes клъстери.
- Charts can be stored in a central repository and used by applications

Инсталираме първо helm.
След това:
helm install petclinic k8s/charts

values.yaml

```yaml
namespace: petclinic
app:
  name: petclinic
  image: petclinic
  tag: latest
  pullPolicy: IfNotPresent
  port: 8080
  serviceType: LoadBalancer
  replicas: 4
db:
  name: postgres
  image: postgres
  tag: latest
  port: 5432
  serviceType: ClusterIP
  replicas: 1
  username: petclinic
  password: petclinic
  database: petclinic
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Values.app.name }}-deployment
  namespace: {{ .Values.namespace }}
spec:
  replicas: {{ .Values.app.replicas }}
  selector:
    matchLabels:
      app: {{ .Values.app.name }}
  template:
    metadata:
      labels:
        app: {{ .Values.app.name }}
    spec:
      containers:
        - image: "{{ .Values.app.image }}:{{ .Values.app.tag }}"
          imagePullPolicy: {{ .Values.app.pullPolicy }}
          name: {{ .Values.app.name }}
```

```
          env:
            - name: SPRING_PROFILES_ACTIVE
              value: {{ .Values.db.name }}
            - name: SPRING_DATASOURCE_URL
              value: jdbc:postgresql://{{ .Values.db.name }}-service:{{ .Values.db.port }}/{{
.Values.db.database }}
          ports:
            - name: http
              containerPort: {{ .Values.app.port }}
              protocol: TCP
      restartPolicy: Always
```

Installing

Combination 1

1) DockerEngine *Или* DockerDesktop
2) kubectl

https://kubernetes.io/docs/tasks/tools/

`kubectl version –client`

3) minikube

https://minikube.sigs.k8s.io/docs/start/

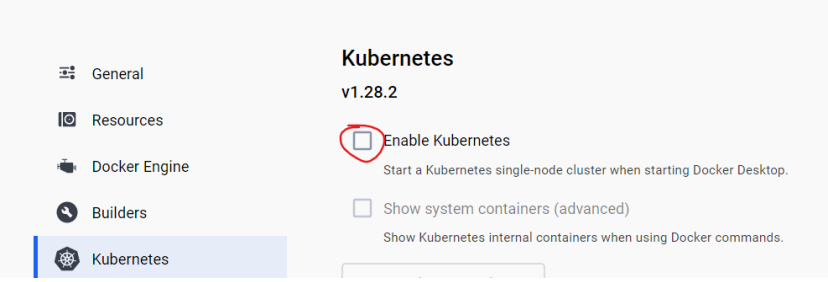
Add the minikube.exe binary to your PATH.
Make sure to run PowerShell as Administrator.

```
$oldPath = [Environment]::GetEnvironmentVariable('Path',
[EnvironmentVariableTarget]::Machine)
if ($oldPath.Split(';') -inotcontains 'C:\minikube'){
  [Environment]::SetEnvironmentVariable('Path', $('{0};C:\minikube' -f $oldPath),
[EnvironmentVariableTarget]::Machine)
}
```

K8S === Kubernetes //8 букви между K и S

Combination 2

Use DockerDesktop with Kubernetes plugin enabled



minikube not needed in this option

Some commands:

minikube start –driver=docker
docker ps
kubectl get pods
kubectl get pods -A all including system

kubectl create deployment nginx --image nginx

Основната разлика между Kubernetes и Docker(и Docker compose) е, че вече минаваме към декларативен начин на работа – чрез въпросните yaml файлове за Kubernetes:

C:\Windows\System32>**kubectl run nginx --image nginx** //създава си служебен yaml файл
pod/nginx created

C:\Windows\System32>**kubectl get pods**
NAME READY STATUS RESTARTS AGE
nginx 1/1 Running 0 84s

C:\Windows\System32>**kubectl delete pod nginx**
pod "nginx" deleted

C:\Windows\System32>**kubectl delete pod/nginx**
pod "nginx" deleted

C:\Windows\System32>**kubectl run nginx --image nginx -o yaml** //-o принтни служебно създадения yaml файл

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2024-08-15T11:38:32Z"
  labels:
    run: nginx
  name: nginx
  namespace: default
  resourceVersion: "14000"
  uid: 67bb5e2b-f0cd-482c-9663-196c6fef2a0e
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: nginx
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: kube-api-access-knh6t
      readOnly: true
  dnsPolicy: ClusterFirst
  enableServiceLinks: true
  preemptionPolicy: PreemptLowerPriority
  priority: 0
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext: {}
  serviceAccount: default
  serviceAccountName: default
```

```
  terminationGracePeriodSeconds: 30
  tolerations:
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
  volumes:
  - name: kube-api-access-knh6t
    projected:
      defaultMode: 420
      sources:
      - serviceAccountToken:
          expirationSeconds: 3607
          path: token
      - configMap:
          items:
          - key: ca.crt
            path: ca.crt
          name: kube-root-ca.crt
      - downwardAPI:
          items:
          - fieldRef:
              apiVersion: v1
              fieldPath: metadata.namespace
            path: namespace
status:
  phase: Pending
  qosClass: BestEffort
```

Доста от служебно създадените неща за yaml файла са излишни.

D:_GitHub locale\BGJUG-public\bgjug-academy-docker-k8s-demo-main> **kubectl apply -f k8s/app-service.yaml**

D:_GitHub locale\BGJUG-public\bgjug-academy-docker-k8s-demo-main> **kubectl get all**

```
NAME        READY   STATUS    RESTARTS   AGE
pod/nginx   1/1     Running   0          11m
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   3h3m
```

За следене на промени в терминала да се зареждат автоматично – този watch е допълнителен plug-in
watch -kubectl get all

app-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: petclinic-deployment
```

```
  namespace: petclinic
spec:
  replicas: 1    //една инстанция в случая - това нещо Docker и Docker Compose не могат да го
направят!
  selector:
    matchLabels:
      app: petclinic
  template:
    metadata:
      labels:
        app: petclinic
    spec:
      containers:
        - image: petclinic:latest
          name: petclinic
          imagePullPolicy: IfNotPresent
          env:
            - name: SPRING_PROFILES_ACTIVE
              value: postgres
            - name: SPRING_DATASOURCE_URL
              value: jdbc:postgresql://postgres-service:5432/petclinic
          ports:
            - name: http
              containerPort: 8080
              protocol: TCP
      restartPolicy: Always
```

app-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: petclinic-service
  namespace: petclinic
spec:
  ports:
    - name: http
      port: 8080
      targetPort: 8080
  selector:
    app: petclinic
  type: LoadBalancer  #ClusterIP   NodePort
```

C:\Windows\System32>**kubectl get pods**

NAME READY STATUS RESTARTS AGE

nginx 1/1 Running 0 22m

Логовете

C:\Windows\System32>**kubectl logs nginx**

/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/08/15 11:38:33 [notice] 1#1: using the "epoll" event method
2024/08/15 11:38:33 [notice] 1#1: nginx/1.27.0
2024/08/15 11:38:33 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/08/15 11:38:33 [notice] 1#1: OS: Linux 5.15.133.1-microsoft-standard-WSL2
2024/08/15 11:38:33 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576

2024/08/15 11:38:33 [notice] 1#1: start worker processes
2024/08/15 11:38:33 [notice] 1#1: start worker process 29
2024/08/15 11:38:33 [notice] 1#1: start worker process 30
2024/08/15 11:38:33 [notice] 1#1: start worker process 31
2024/08/15 11:38:33 [notice] 1#1: start worker process 32
2024/08/15 11:38:33 [notice] 1#1: start worker process 33
2024/08/15 11:38:33 [notice] 1#1: start worker process 34
2024/08/15 11:38:33 [notice] 1#1: start worker process 35
2024/08/15 11:38:33 [notice] 1#1: start worker process 36
2024/08/15 11:38:33 [notice] 1#1: start worker process 37
2024/08/15 11:38:33 [notice] 1#1: start worker process 38
2024/08/15 11:38:33 [notice] 1#1: start worker process 39
2024/08/15 11:38:33 [notice] 1#1: start worker process 40
2024/08/15 11:38:33 [notice] 1#1: start worker process 41
2024/08/15 11:38:33 [notice] 1#1: start worker process 42
2024/08/15 11:38:33 [notice] 1#1: start worker process 43
2024/08/15 11:38:33 [notice] 1#1: start worker process 44
2024/08/15 11:38:33 [notice] 1#1: start worker process 45
2024/08/15 11:38:33 [notice] 1#1: start worker process 46
2024/08/15 11:38:33 [notice] 1#1: start worker process 47
2024/08/15 11:38:33 [notice] 1#1: start worker process 48

kubectl delete deploy nginx-deployment
kubectl delete deployment nginx-deployment

Идеята на контекста, е че можем да достъпваме няколкок Kubernetes клъстъра – единият клъстър може да го менъджира нашата фирма, у нас локално друг клъстър, и т.н.

C:\Windows\System32>**kubectl config get-contexts**
CURRENT NAME CLUSTER AUTHINFO NAMESPACE
* docker-desktop docker-desktop docker-desktop

C:\Windows\System32>**minikube start**
W0815 15:24:53.997133 22260 main.go:291] Unable to resolve the current Docker CLI context "default": context "default": context not found: open C:\Users\svilk\.docker\contexts\meta\37a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f4133a33f0688f\meta.json: The system cannot find the path specified.
* minikube v1.33.1 on Microsoft Windows 11 Pro 10.0.22631.3880 Build 22631.3880
* Automatically selected the docker driver. Other choices: hyperv, ssh
* Using Docker Desktop driver with root privileges
* Starting "minikube" primary control-plane node in "minikube" cluster
* **Pulling base image v0.0.44 ...**
* Downloading Kubernetes v1.30.0 preload ...
> gcr.io/k8s-minikube/kicbase...: 481.58 MiB / 481.58 MiB 100.00% 3.43 Mi
> preloaded-images-k8s-v18-v1...: 342.90 MiB / 342.90 MiB 100.00% 2.44 Mi
* **Creating docker container** (CPUs=2, Memory=8000MB) ...
* Preparing Kubernetes v1.30.0 on Docker 26.1.1 ...
- Generating certificates and keys ...
- Booting up control plane ...
- Configuring RBAC rules ...
* Configuring bridge CNI (Container Networking Interface) ...
* Verifying Kubernetes components...
- Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass

* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default

Автоматично сменя контекста на kubectl да използва minikubea, не docker-desktop
C:\Windows\System32>**kubectl config get-contexts**
CURRENT NAME CLUSTER AUTHINFO NAMESPACE

```
        docker-desktop   docker-desktop   docker-desktop
*       minikube         minikube         minikube         default

C:\Windows\System32>kubectl config use-context docker-desktop
Switched to context "docker-desktop".
```

C:\Windows\System32>**kubectl config use-context minikube**
Switched to context "minikube".

C:\Windows\System32>**kubectl create namespace petclinic**
namespace/petclinic created

C:\Windows\System32>**kubectl config use-context minikube --namespace=petclinic**
Switched to context "minikube".

D:_GitHub locale\BGJUG-public\bgjug-academy-docker-k8s-demo-main> **kubectl apply -f k8s/app-deployment.yaml**
deployment.apps/petclinic-deployment created

D:_GitHub locale\BGJUG-public\bgjug-academy-docker-k8s-demo-main> **kubectl run petclinic --image petclinic**
pod/petclinic created

C:\Windows\System32>**minikube stop**
W0815 16:15:12.563473 4696 main.go:291] Unable to resolve the current Docker CLI context "default": context "default": context not found: open C:\Users\svilk\.docker\contexts\meta\37a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f4133a33f0688f\meta.json: The system cannot find the path specified.
* Stopping node "minikube" ...
* Powering off "minikube" via SSH ...
* 1 node stopped.

Minikube си има собствен контекст. Обаче за локални images, трябва да му се даде тази команда при MacOS (за Windows обаче не бачка). И след това наново билдваме image-а petclinic (D:_GitHub locale\BGJUG-public\bgjug-academy-docker-k8s-demo-main> **docker build -t "petclinic" -f Dockerfile-multistage .**) и текущия терминал го засича вече Image-а като локален такъв.

```
# Set docker env
eval $(minikube docker-env)              # Unix shells
minikube docker-env | Invoke-Expression # PowerShell
```

D:_GitHub locale\BGJUG-public\bgjug-academy-docker-k8s-demo-main> **kubectl get all**
NAME READY STATUS RESTARTS AGE
pod/petcliniccontainer 0/1 ImagePullBackOff 0 22s

D:_GitHub locale\BGJUG-public\bgjug-academy-docker-k8s-demo-main> **kubectl apply -f k8s цялата папка**
deployment.apps/petclinic-deployment created
service/petclinic-service created
deployment.apps/postgres-deployment created
service/postgres-service created

D:_GitHub locale\BGJUG-public\bgjug-academy-docker-k8s-demo-main> **kubectl get all**

```
NAME                                     READY   STATUS             RESTARTS      AGE
pod/petclinic-deployment-85b48d56b4-sldnb   1/1     Running            2 (33s ago)   42s
pod/petcliniccontainer                   0/1     ImagePullBackOff   0             2m32s
pod/postgres-deployment-6cdb694745-bskt8    1/1     Running            0             42s

NAME                        TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
service/petclinic-service   LoadBalancer   10.108.107.116   localhost     8080:31560/TCP   42s
service/postgres-service    ClusterIP      10.102.100.199   <none>        5432/TCP         42s

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/petclinic-deployment   1/1     1            1           42s
deployment.apps/postgres-deployment    1/1     1            1           42s

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/petclinic-deployment-85b48d56b4   1         1         1       42s
replicaset.apps/postgres-deployment-6cdb694745    1         1         1       42s
```

Kubernetes е предвиден да работи с много сървъри с много клъстери, най-често Cloud услуги. Като му зададем LoadBalancer, то трябва да има друг Load Balancer, към който Kubernetes да се кънектва!

Kubernetes изпълнява image-те по същият начин както Docker.

Kubernetes влиза в полза особено много когато проекта ни има микросървиси.