@[Entity|Document] public class A {  
 @Id [@GeneratedValue(strategy=GenerationType.$)] private long $;  
 @Temporal(TemporalType.$) private Date $;  
 @OneToMany(cascade = CascadeType.$, mappedBy=”attribuut\_B”, fetch = FetchType.$)   
 private List<B> $;   
 public $ (params){} # lege constructor VERPLICHT bij elke @Entity of @Document  
@[Entity|Document] public class B {  
 @ManyToOne(fetch = FetchType.$, optional = [true|false])  
 @JoinColumn(name = “A\_id”, nullable = [true|false])  
 @OnDelete(action = OnDeleteAction.$) private A a; }  
public interface $ extends [MongoRepository|CrudRepository]<T, V>{ # T = sleuteltype, V = klasse  
 @Query(“SELECT x from y x WHERE x.attribuut = :parameter”)  
 [List|Optional]<Klasse> $ (@Param(“parameter”) [int|String|…] parameter);  
 // Bij enum heel het pad vermelden: “WHERE x.attribuut = package.enumklasse.keuze” }  
@Component public class $Service {  
 @Autowired private Repository repository;  
 public $ $ { return repository.$ } }   
[application.properties] server.port=$ # $ vervangen door poortnummer  
@RestController public class $Controller {  
 @Autowired private $Service;  
 @RequestMapping(value=”/path/{abc}”, method=RequestMethod.$)  
 1) public $ $ () -> /path  
 2) public $ $(@PathVariable(“abc”) $ $) -> /path/{abc} (ook aanpassen in value)  
 3) public $ $ (@RequestBody Object o) -> verwacht JSON of gelijkaardig  
 4) public $ $ (@RequestParam(“$”) $ $) -> form-encoded data  
 { logica }   
 public void performResponse(params) { logica }}  
[terminal] bin/zookeeper-server-start.sh config/zookeeper.properties  
[terminal] bin/kafka-server-start.sh config/server.properties  
public interface [ProducerChannels|ConsumerChannels] {  
 String EVENT = “EVENT”;  
 @[Output|Input]( EVENT) [MessageChannel|SubscribableChannel] doSomething(); }  
@EnableBinding(value={ProducerChannels.class, ConsumerChannels.class}) <- IN MAIN APPLICATIE  
[application.properties] spring.cloud.stream.bindings.EVENT.destination=EVENT  
[application.properties] spring.cloud.stream.bindings.EVENT.contentType=application/json  
[application.properties] spring.cloud.stream.bindings.EVENT.group=EVENT\_group <- bij consumers  
@MessagingGateway public interface MessageChannelGateway {  
 @Gateway(requestChannel = ProducerChannels.EVENT) void doSomething(params) }  
@Component public class $Saga {   
 @Autowired private MessageChannelGateway gateway; # doorsturen van berichten  
 @Autowired private $RestController commandHandler; # voor berichten terug te geven  
 # nu komen er verschillende methoden die elk een stap in de saga voorstellen}  
[terminal] bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic EVENT @Component public class [CommandHandler|EventConsumer] { # gewoon naamgeving dat anders is  
 @Autowired private $Service service; # instantie van service  
 @StreamListener(ConsumerChannels.EVENT) # ontvangen van berichten van een kanaal  
 @SendTo(ProducerChannels.EVENT) # op kanaal zetten, niet nodig bij void methoden  
 public … (params) { logica }}  
@Bean public RouteLocator locator(RouteLocatorBuilder b) { # bij api gateway  
 return b.routes().route(r -> r.host(“\*”).and().path(“/x/\*\*”).uri(<http://localhost:xxxx>).build()}  
**docker ps -a** # alle containers tonen  
**docker inspect** *<docker object>* # informatie over een image, container, … tonen  
**docker run** [-d] [-p 80:80/tcp | -P] [--link *<container>*] [--name *<naam>*] [-e var1=’x’] [-e var2=’y’] *<image>* # start container vanuit image  
**docker start** [--attach] *<container>* # STDOUT/STDERR uitprinten van container  
**docker container logs** *<container>* # logs van een bepaalde container  
**docker** [**rm** *<container>* | **container** **prune**] # [1 | alle] GESTOPTE container(s) verwijderen  
**docker** **pull** *<image>* # bestaande image van de Docker hub halen  
**docker** **exec -i -t** *<container>* /bin/bash # interactie BINNEN een bepaalde container  
**docker** **build** **-t** *<eigen\_image\_naam>* . 🡨 punt niet vergeten op het einde  
[DockerFile] FROM <bestaande\_image>”\n”WORKDIR <directory in container>”\n”  
COPY <directory\_op\_systeem> <directory\_in\_container>”\n”RUN cmd1 && cmd2 && … && cmdn  
CMD <uitvoeren\_jar\_ofzo> # bv met 'java -jar file.jar’, best geen [“ “, “ “] zoals op docs   
[application.properties] spring.jpa.hibernate.ddl-auto=create  
[application.properties] spring.datasource.url=jdbc:mysql://<mysql\_container>:3306/table  
[application.properties] spring.datasource.[username|password]=…  
[application.prop…]spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect   
[ComposerFile] version: ‘3’”\n”services: #lijst van services. Belangrijke attributen per service:   
 build, depends\_on, image, command, environment, ports, volumes. Zoek gewoon op in documentatie  
**docker-compose -f docker-compose.yml build** # builden of rebuilden van services  
**docker-compose up** [--build] # aanmaken en starten van containers  
**docker swarm init** # Initialisatie van de master swarm. Dit geeft een token terug.  
**docker swarm join-token worker** # Token ophalen van vorig commando.  
**docker service create** [-e var1=’x’] [-e var2=’y’] [--replicas <*aantal>*] [--name *<naam>*]  
**docker service** [ls | [ps|rm] *<naam>*] # services tonen | nodes van service bekijken | service verwijderen  
**docker stack deploy** [–compose-file *<naam>*] *<stacknaam>* # aanmaken van stack  
**docker stack rm** *<stacknaam>*  
implementatie CIRCUIT BREAKER in API GATEWAY: voeg @RestController toe boven de klasse   
@Bean public RouteLocator locator(RouteLocatorBuilder b) { # bij api gateway  
 return b.routes().route(r -> r.host(“\*”).and().path(“/x/\*\*”).filters(f -> f.hystrix(config -> config.setName(“config1”).setFallbackUri(“forward:/fallback”))) .uri(<http://localhost:xxxx>).build()}  
@RequestMapping(“/fallback”) public Mono<String> fallback() { return Mono.just(“fout”);}   
[application.properties] management.security.enabled=false  
[application.properties] management.endpoint.web.expouse.include=\*  
[application.properties] management.endpoint.metrics.enabled=true  
[application.properties] management.endpoint.prometheus.enabled=true  
[application.properties] management.metrics.export.prometheus.enabled=true  
(niet vergeten Prometheus.yml file te maken => zoek op docker docs)