Konfigurationsvalg

Til denne opgave har vi valgt en topic-exchange. Dette er valgt for at gøre udvidelse nemmere. For eksempel hvis der pludseligt skal kunne reserveres en tennisbane. Således er vores nuværende topic for reservations *"hotel.room.reservation"* og ved udvidelse med reservationer af en tennisbane kunne det blive *“hotel.tenniscourt.reservation”*. Dette ville tillade at samle reservations i en kø, på trods af at der er forskellige typer i denne kø. Det vil også være nemmere i senere at binde tennis routingen til en anden kø, hvis reservationkøen bliver for stor / travl.

Vores strategi med oprettelse af køer er at en “admin” (eller CI/CD pipeline) skal gøre dette. Således kan scriptet *“RabbitMQSetup.js”* oprette alle exchanges og køer. Dette gør at producers og consumers kun skal kende til navnet på den exchange de publisher til, eller den kø de consumer fra, hvilket giver en høj afkobling.

Konfigurationen af exchanges og køer er med “*durable: true, autoDelete: false”*. Hertil er alle publishede beskeder *persistent*. Durable og persistent er for at sikre at ingen beskeder går tabt når først de er sendt. *autoDelete* er slået fra, da det ville være en admins ansvar at slette køer og exchanges når de ikke længere er i brug, eventuelt via RabbitMQ management interfacet.
Udover dette er consumers også sat til at prefetch’e maximalt 1 besked fra køen ad gangen, for at undgå diverse concurrency problemer, samt manuelle ACK’e for at sikre at der først ACK’es når et køelement er håndteret.

OBS: Vi er opmærksomme på at det nuværende system ikke tåler 2 instanser af “Handle Reservation”-servicen da vi så har concurrency problemer ved tilgang til databasen og tjek af frie reservation-tidspunkter. Dette kunne man fikse på flere måder, e.g. ved at gøre denne service *Exclusive* consumer til køen fra Front-enden (men dette ville begrænse i andre aspekter). Dette ses dog som out-of-scope og er dermed ikke forsøgt fikset.