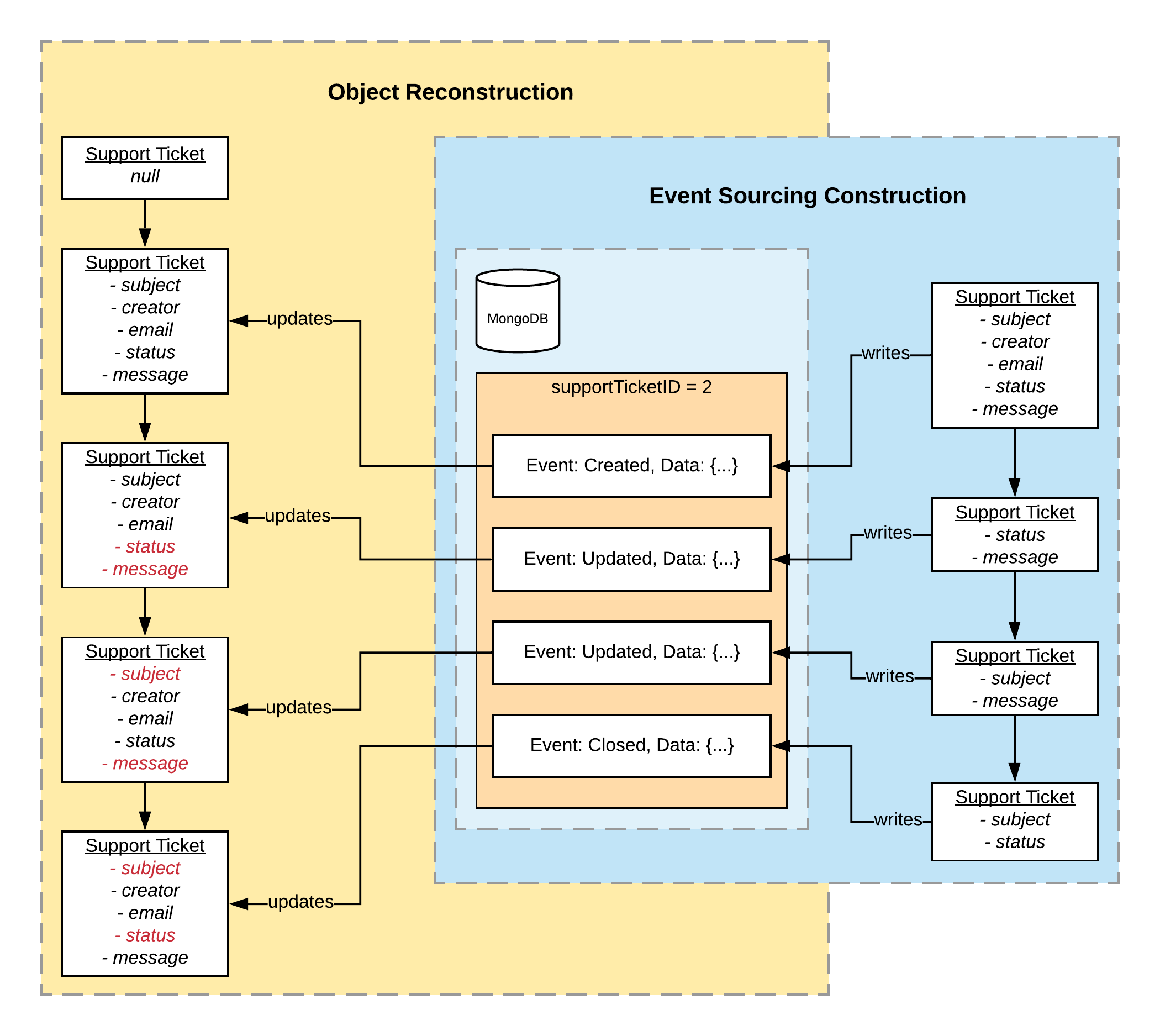
# Event Sourcing

Event Sourcing in de applicatie komt voor in de Support Service. In deze service vindt de afhandeling van support tickets plaats. Een klant of bezoeker van een support ticket indienen. Naar verloop van tijd kijkt een medewerker naar deze ticket en bepaald aan de hand daarvan de actie die ondernomen moet worden. Hij update hiervoor de ticket zodat de klant weet wat de laatste status van zijn ticket is. Bij het ‘verwijderen’ van een ticket wordt zijn status op ‘closed’ gezet, zodat de klant of indiener van de support ticket later nog kan terugkijken.

Om de geschiedenis van een support ticket te kunnen bewaren, en om de staat van een support ticket te kunnen opbouwen naar een bepaald moment in de tijd, is er gekozen om het principe ‘Event

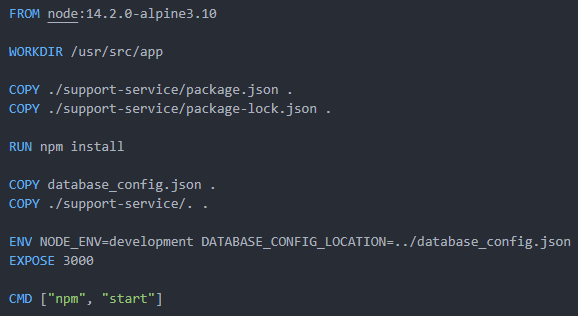
Sourcing’ toe te passen. In Afbeelding 1 is te zien hoe dit in zijn werk gaat. Een nieuwe support ticket kan worden aangemaakt via de POST/support endpoint. Deze nieuwe ticket wordt opgeslagen in de MongoDB, waarbij het supportTicketID gelijk staat aan de door MongoDB gegenereede ‘*\_id’* waarde. In dit document worden de verschillende events opgeslagen, met als eerste event het creëren van de support ticket. Nieuwe events worden gekoppeld in dit document. Bij het ophalen van het meest actuele support ticket object worden alle events uit een support ticket gelezen. De waardes die per event veranderen worden geüpdate, en dit uiteindelijke object wordt verstuurd. Elk event heeft een timestamp, dus de staat van een support ticket kan worden hersteld naar elk gewenst punt in tijd



Afbeelding : Schematische weergave Event Sourcing

# Containerization

Elke microservice draait in zijn eigen Docker container. Deze Docker container wordt opgebouwd met een eigen Dockerfile. Een fragment van zo’n Dockerfile is te vinden in Afbeelding 2. De context van de Dockerimage ligt in de root van het project (1 map boven de map van de microservice). Dit is vanwege de database\_config.json die elke microservice nodig heeft. Naast deze 4 microservices die hun eigen Dockerfile hebben, draaien er nog 3 andere containers: 1 voor de MongoDB server; 1 voor de MySQL server en 1 voor de RabbitMQ server. Om te voorkomen dat de images van Docker te groot worden, worden de node\_modules niet gekopieerd middels een .dockerignore bestand.

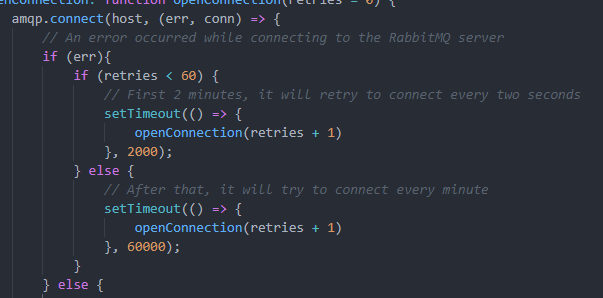


Afbeelding : Dockerfile van Support Service

Om deze containers met elkaar te laten communiceren (zoals met de MongoDB server bijvoorbeeld) is gebruik gemaakt van Docker Compose. Docker compose is een tool van Docker om meerdere containers op dezelfde host te laten draaien. Hierbij wordt gebruik gemaakt van een speciaal intern netwerk waarmee containers onderling kunnen communiceren. De structuur van deze containers en het onderlinge netwerk wordt gedefinieerd in de docker-compose.yaml. Een fragment van dit bestand is te zien in Afbeelding 3. De containers kunnen onderling met elkaar communiceren aan de hand van de ‘service name’ (de eerste regel van de afbeelding, mongodb, bijvoorbeeld).

Afbeelding : Fragment docker-compose.yaml

Het kan voorkomen dat een aantal microservices eerder zijn opgestart dan zijn afhankelijkheden, zoals de RabbitMQ server. Gebleken is dat de “depends\_on” waarde van de docker-compose wel werkt als het gaat om wachten tot de container draait, maar deze waarde wacht niet tot de server zelf daadwerkelijk is opgestart. Hier is een retry-mechanisme voor geschreven, die een aantal keer probeert verbinding te maken met de RabbitMQ container zonder de applicatie te laten crashen. Een uitwerking hiervan is te vinden in Afbeelding 4



Afbeelding : Retry mechanisme