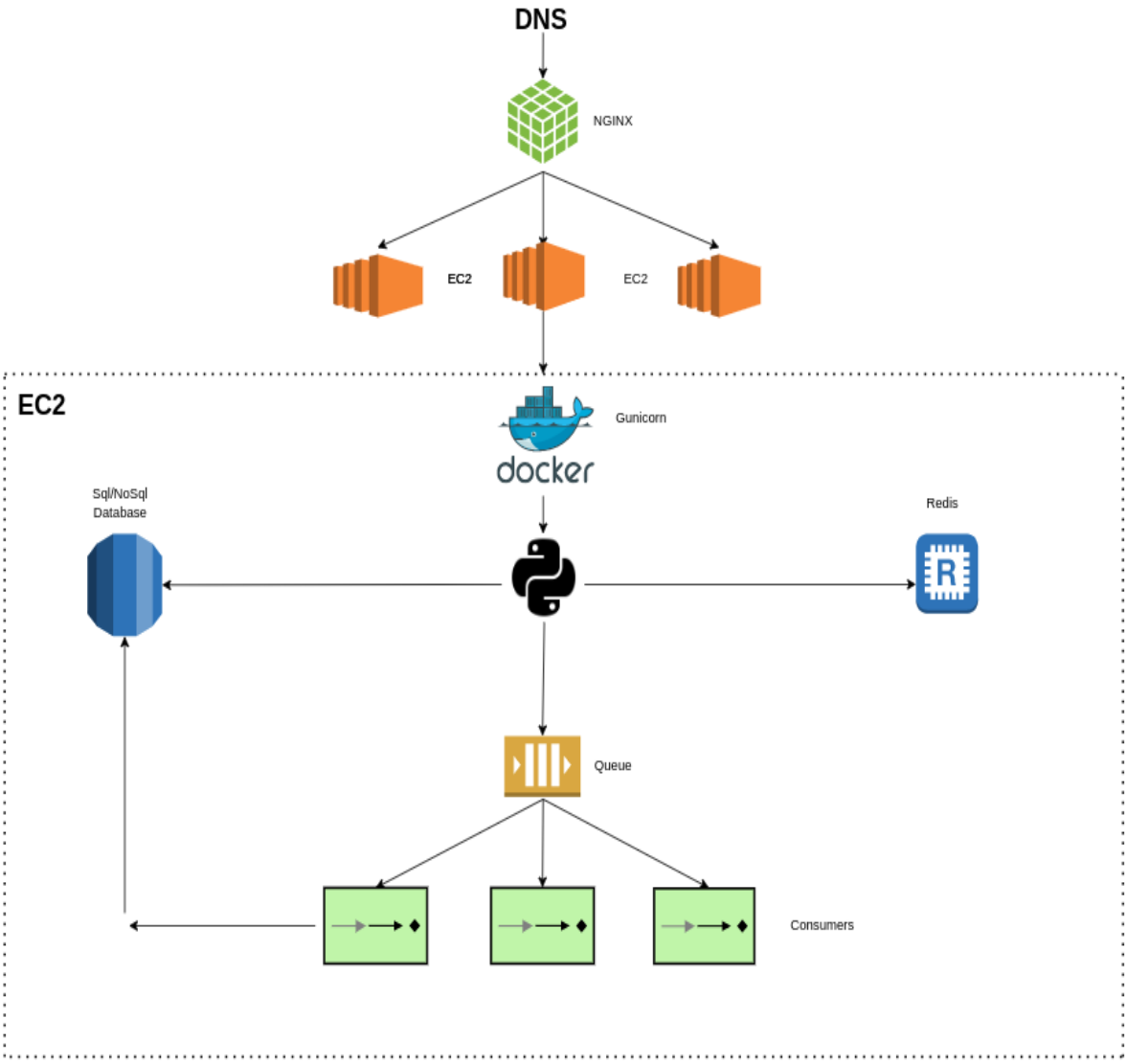
Architecture For Scalable Movie Database



Components:

- NGINX
    - Using Nginx as an Web Server for our application in order to handle the following:
        - Concurrent Connections
        - Static Files (Current Application may include Image, Audio, Video related to Movies)
        - Load Balancer : Efficient redirection of traffic onto lower level Ec2 instances.
        - No Downtime during reload in case of route change.

- Ec2 Instance
    - Ec2 instance to host and run the application itself.

- Docker
    - Docker to run, deploy applications with ease.

- Gunicorn
    - Gunicorn to be used as standard WSGI to forward requests to the application.
    - We can set up multiple worker/thread combinations to distribute incoming requests.

- Python Application
    - Current Application is written in Flask.
    - Although flask < 2.0 does not support async calls, we can use celery with flask to publish long running tasks onto the queue in an async manner.
    - In our application, we have decoupled the `create/update/delete` operations to celery.

- Sql Database
    - A relational database is preferred since we know that the entities to be stored are related to each other.(eg. Actors, Movies, Shows, Genres etc).
    - Scaling can be done Vertically in order to handle increasing load.
    - Schemas are consistent and hence structure is defined well in advance.

- Redis
    - User and token level details can be stored onto Redis for authentication/authorization purposes.

- RabbitMq
    - RabbitMq is used as a Message Broker for queuing messages and their eventual consumption.

- Celery Worker
    - Multiple Celery workers can be spawned for asynchronous publishing and consumption of messages.