Репорт

Подготовил студент: Жамауов Камбар

Архитектура

Платформа для спортивного программирования - место для студентов, школьников, да и не только, испытать свои силы в программировании, математике и логике. Она предоставляет доступ к разного родам алгоритмическим задачам. Под капотом каждая задача должна содержать в себе тесты, валидаторы, чекеры, которые предназначены для сопоставления решения пользователя с оригинальным решением автора. Помимо этого, на сайте могут проходить онлайн контесты, на которые могут быть записаны от сотен до десятков тысяч пользователей. И для сайта, основной обязанностью является быстрая проверка пользовательского решения, и отправки информации о статусе задачи.

Так как нагрузка с каждым годом будет только увеличиваться, необходимо будет наращивать мощь серверов, а также увеличивать объемы памяти баз данных. Эту проблему лучше всего решает микросервисная архитектура. В моём проекте имеются следующие сервисы:

1) Сервис компиляции решений
2) Сервис пользователя
3) Сервис сообщений
4) Сервис соревнований
5) Сервис обучающих курсов
6) Сервис авторизации

Разбивая большое приложение каждый микросервис можно разработать силами одной команды, чтобы увеличить скорость создания программного продукта.

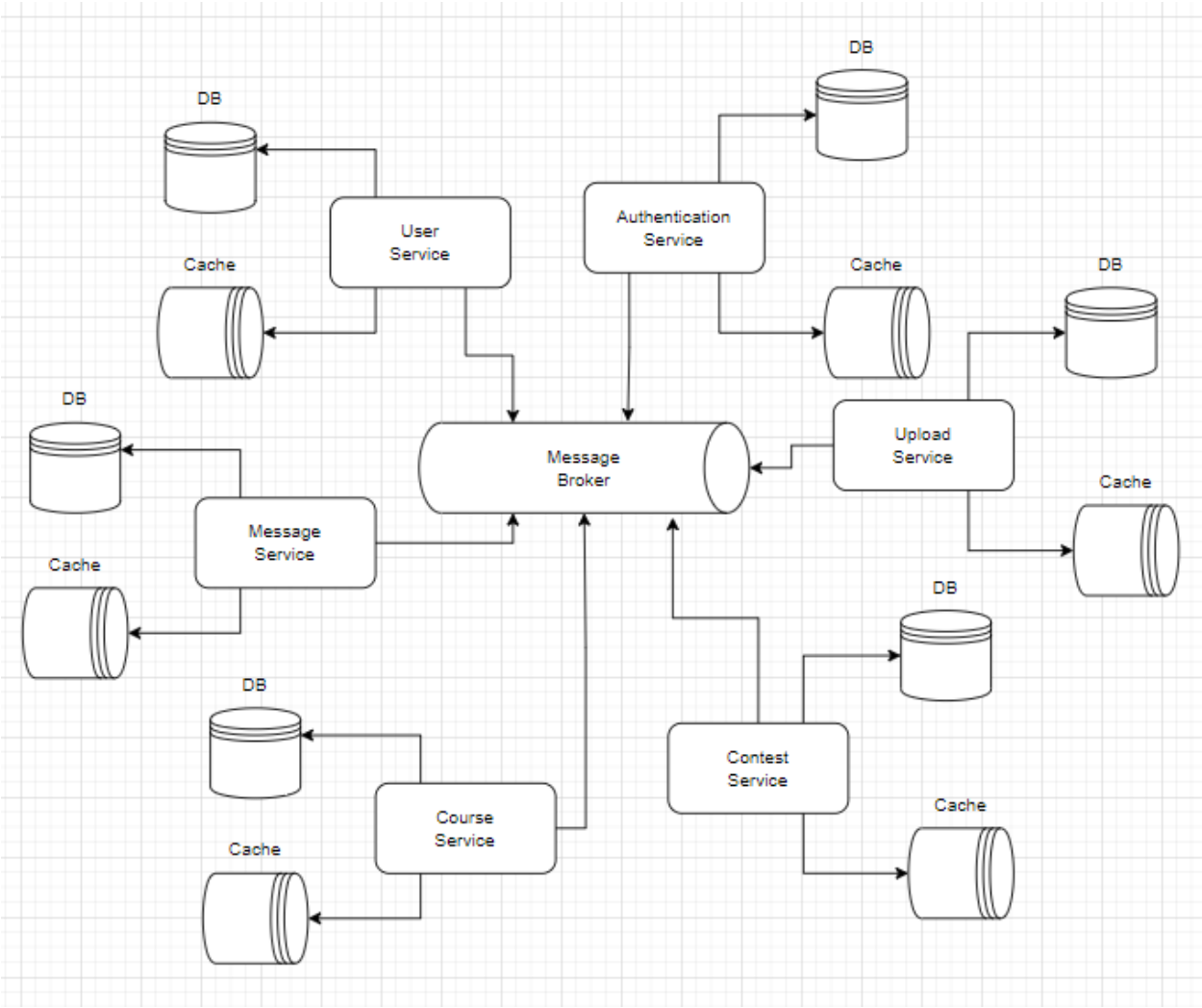
Гибкость: Микросервисы общаются между собой различными путями через REST, gRPC или же SOAP. Определив API им не нужно заботиться о том какая логика будет инкапсулирована внутри конкретного сервиса. Следовательно разные сервисы могут использовать разные базы данных (PostrgeSQL, MySQL, MongoDB, Aerospike и м.д.), разные хранилища кэша (Redis, ElasticSearch) и свой язык программирования. Каждая команда вольна выбирать свой язык и технологию реализацию микросервисов.

Отказоустойчивость: является также важной особенностью, так как, к примеру если сервис компиляции решений упадет, все сервисы помимо соревнований, которая прямо зависит от него, будут доступны, в то время как текущий сервис будет находится в стадии технического обслуживания.

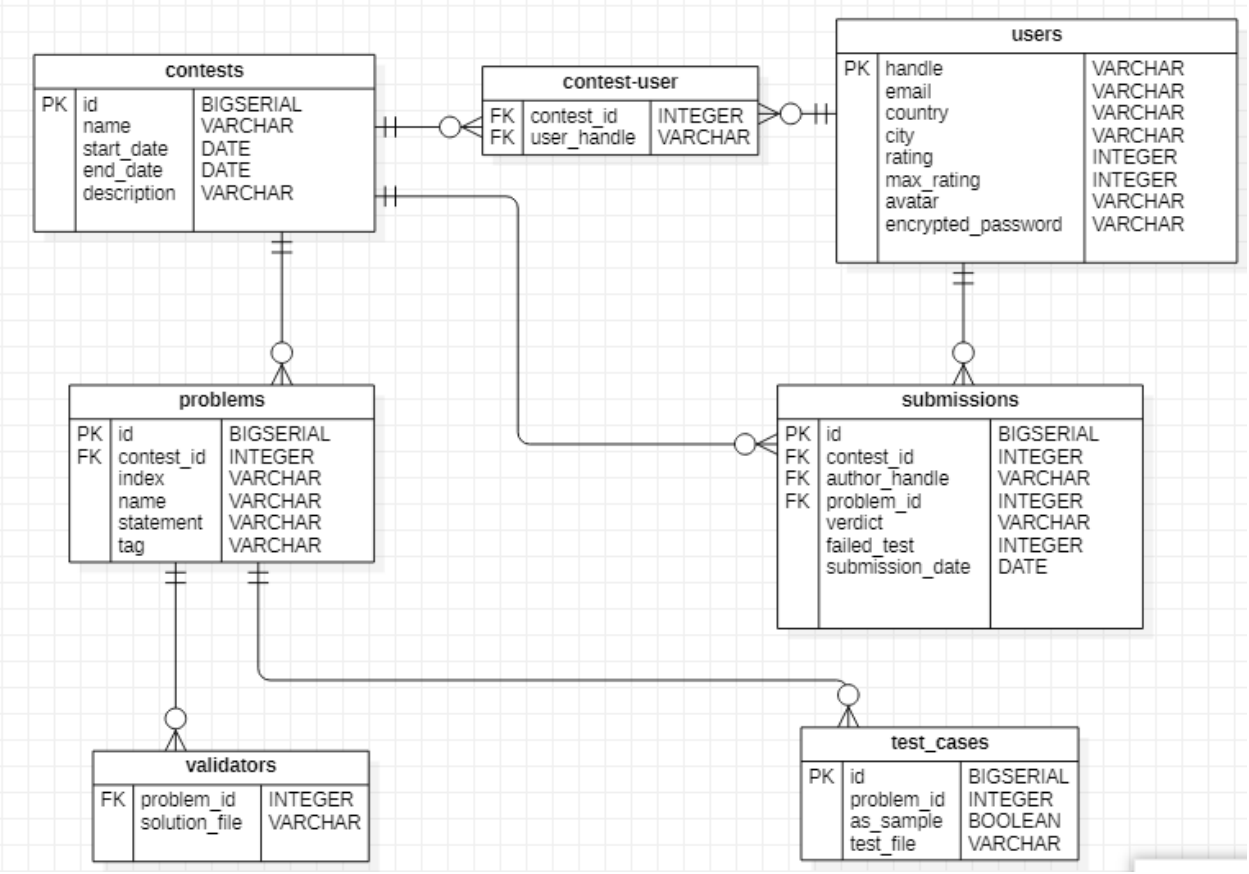
Масштабируемость: самые необходимые и нужные сервисы можно дополнить и расширить, когда появится такая необходимость. Вся система при этом остается прежней. Мы можем использовать систему управления контейнерами - Docker. В идеале любой компонент может быть развернут внутри контейнера.

UML диаграммы

Архитектура проекта будет примерно выглядеть следующим образом(каждый сервис обращается имеет свою базу данных и кэш, инвалидация которого происходит внутри мэссэдж брокера):



Entity-Relation диаграмма - описывает сущности проекта и их взаимосвязь при помощи указаний первичных/внешних ключей. Как мы можем видеть, имеются различны виды отношений «one-to-many» - элемент одного объекта может быть связан с несколькими элементами второго, но каждый элемент второго связан лишь с одним элементом первого объекта(contest-problems, user-submissions, problem-validators/test_cases), «many-to-many» - множественным записям из одной таблицы могут соответствовать множественные записи из другой (contests-users).



Sequence диаграмма – необходима для описания взаимодействия объектов с различными стереотипами**.** Самая ключевая бизнес логика проекта располагается в сервисе компиляции решений, поэтому важно понимать процесс взаимодействия и работы с ним.