Go_Team01-1

Развилка из недоступного проекта

Обновление с 5
Администратор создал 5 месяцев назад

Имя	Последний коммит	Последнее обновление
образцы кода	Инициировать фиксацию	1 год назад
образцы данных	Инициировать фиксацию	1 год назад
наборы данных	Инициировать фиксацию	1 год назад
материалы	Обновление с 5	5 месяцев назад
разное	Инициировать фиксацию	1 год назад
источник	Инициировать фиксацию	1 год назад
.gitignore	Обновление с 3	1 год назад
ИЗМЕНЕНИЙ	Инициировать фиксацию	1 год назад
ЛИЦЕНЗИЯ	Инициировать фиксацию	1 год назад
README.md	Обновление с 4	7 месяцев назад

README.md

Команда 01 - Перейти в учебный лагерь

Warehouse 13

Contents

Chapter I

General rules

- Your programs should not quit unexpectedly (giving an error on a valid input). If this happens, your project will be considered non functional and will receive a 0 during the evaluation.

- We encourage you to create test programs for your project even though this work won't have to be submitted and won't be graded. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded.
- If your code is using external dependencies, it should use Go Modules for managing them

Chapter II

Rules of the day

- You should only turn ink `\*.go` , `\*_test.go` files and (in case of external dependencies) `go.mod + go.sum`
- Your code for this task should be buildable with just `go build`
- All your tests should be runnable by calling standard `go test ./...`

Глава III

вступление

— Ой, Арти, это же древнее! - Латтимер собирался дернуть себя за волосы. - На дворе 21 век, никто больше не пользуется ручкой и бумагой для каталогизации вещей!

— Что ты хочешь, чтобы я сделал, Пит? Так было всегда!

— Ну, у нас есть компьютер, не так ли? Он довольно древний, но мы можем установить...

— Нет, мы не можем! Вы же знаете, что Хранилище должно оставаться совершенно секретным, не так ли? Мы не загружаем и не устанавливаем здесь никакого программного обеспечения.

— Итак, вы хотите, чтобы мы написали собственную реализацию базы данных? Будет ли это работать?

— Хм, это МОЖЕТ сработать...

- Идеальный! Итак, я попрошу Майку реализовать один для нас!

— Подождите, я думал, вы говорите о том, чтобы реализовать это самостоятельно...

— Нет, я плохо разбираюсь в кодировании. В любом случае, давайте спроектируем его! Какую информацию мы должны хранить?

— Каждому артефакту присваивается свой уникальный идентификатор, после чего мы должны хранить некоторые метаданные о нем в структурированном формате. Кроме того, все должно быть доступно через интерфейс командной строки, так как у меня нет этих современных графических интерфейсов...

Час спустя...

- Что? Вы хотите, чтобы я написал полнофункциональное хранилище ключей и значений для работы с документами JSON? С нуля?

- Знаю, знаю! Но вы не одиноки! Вот, я приготовил вам кофе в кофейной кружке Энди Уорхола, так что это довольно много и неограниченный ресурс суперсилы кофеина.

— Но, Пит! Мы должны решить несколько вопросов! Что делать, если данные повреждены? Что, если мы не сможем получить доступ к некоторым данным артефакта, когда они нам нужны больше всего?

— Не волнуйся, Мика, ты сможешь! Я тоже буду сидеть здесь и помогать. Давайте просто рассмотрим вопросы один за другим.

Глава IV

Задача 00: Масштабируемость

Через некоторое время доска была исписана.

«Доступ через командную строку — должно быть отдельное приложение, которое предоставит REPL и будет подключаться к работающему экземпляру по сети, даже если это просто локальный хост и порт».

«Мы должны иметь возможность убить любой экземпляр (процесс) базы данных, и он должен продолжать работать и предоставлять ответы на запросы. Это означает, что одним из настраиваемых параметров, например, должен быть коэффициент репликации, то есть количество копий одного и того же документа. мы храним. Для целей тестирования 2, вероятно, достаточно"

«Клиент должен периодически проверять, доступен ли текущий экземпляр базы данных. Если он перестает отвечать на запросы, он должен автоматически переключиться на другой экземпляр».

«Кроме того, для простоты давайте предположим, что на данный момент масштабируемость означает, что клиент должен знать обо всех других узлах. Каждый ответ пульса от текущего узла должен включать все известные в настоящее время адреса и порты экземпляров вместе с текущим коэффициентом репликации».

Итак, здесь нам нужно реализовать две программы — одну клиентскую, а другую — экземпляр базы данных. Всякий раз, когда вы запускаете новый экземпляр, вы должны иметь возможность указать его на существующий экземпляр, поэтому после получения пульса он отправит свой хост и порт всем другим работающим узлам, и все узнают нового парня.

Если узел экземпляра запущен с коэффициентом репликации, отличным от существующих узлов, он должен определить это и автоматически завершить работу без присоединения к кластеру. Это означает, что фактор репликации, вероятно, также должен быть включен в сердцебиение.

Вы можете использовать для этого любой сетевой протокол — HTTP, gRPC и т. д.

Всякий раз, когда коэффициент репликации превышает количество работающих узлов, информация об этой проблеме должна включаться в пульс и явно отображаться на каждом подключенном клиенте. Вы можете увидеть пример сеанса пользователя в Задаче 01.

Реальная работа с документами будет реализована в следующем задании.

Глава V

Задача 01: Балансировка и запросы

— Хорошо, давайте использовать строки UUID4 в качестве ключей артефактов. Нам также нужно реализовать некоторую балансировку, чтобы обеспечить отказоустойчивость...

Наша простая база данных должна поддерживать только три операции — GET, SET и DELETE.

Вот как должен выглядеть типичный сеанс с комментариями (начиная с #):

```
~$ ./warehouse-cli -H 127.0.0.1 -P 8765
Connected to a database of Warehouse 13 at 127.0.0.1:8765
Known nodes:
127.0.0.1:8765
127.0.0.1:9876
127.0.0.1:8697
> SET 12345 '{"name": "Chapayev's Mustache comb"}'
Error: Key is not a proper UUID4
> SET 0d5d3807-5fbf-4228-a657-5a091c4e497f '{"name": "Chapayev's Mustache comb"}'
Created (2 replicas)
> GET 0d5d3807-5fbf-4228-a657-5a091c4e497f
'{"name": "Chapayev's Mustache comb"}'
> DELETE 0d5d3807-5fbf-4228-a657-5a091c4e497f
Deleted (2 replicas)
> GET 0d5d3807-5fbf-4228-a657-5a091c4e497f
Not found
>
# if current instance is stopped in the background
Reconnected to a database of Warehouse 13 at 127.0.0.1:8697
Known nodes:
127.0.0.1:9876
127.0.0.1:8697
>
# if another current instance is stopped in the background
Reconnected to a database of Warehouse 13 at 127.0.0.1:9876
Known nodes:
127.0.0.1:9876
WARNING: cluster size (1) is smaller than a replication factor (2)!
>
```

If a key specified in SET already exists in a database the value should be overwritten. If it doesn't, then SET operation should provide read-after-write consistency, meaning immediate reading should give you proper value.

When updating an existing value or deleting it, an eventual consistency should be implemented, meaning immediate (dirty) reads can (but not "should"!) give you old results, but after a couple of seconds the data should be updated to a proper new state.

You can implement key-hash-based balancing so your client could explicitly calculate for every entry the list of nodes where it should be stored according to a replication factor. This will also come in handy for deletion.

If a current node is killed during writing, your client should automatically perform another request to another available node. The only case when user should see the error like "Failed to write/read an entry" is when ALL instances are dead.

Chapter VI

Task 02: Long Live the King

Let's upgrade the logic from Tasks 00/01. Now, we introduce concepts of a Leader and a Follower nodes. This leads us to a list of important changes:

- from now on, client ONLY interacts with a Leader node. The hashing function to determine where to write replicas is now *on Leader*, *not* in client
- all nodes (Leader and Followers) keep sending each other heartbeats with a full list of nodes. If node doesn't respond to heartbeats for some specific configurable timeout (for testing purposes you should set it to 10 seconds by default)
- if the Leader is stopped, remaining Followers should be able to choose a new Leader among them. For simplicity, each of them can just order the list of nodes by some other unique identifier (numeric id, port etc.) and pick the topmost one. From that moment all heartbeats will include a new elected Leader
- if not able to connect to a known Leader, a client should try and connect to Followers to receive a heartbeat from them. If a Leader is killed, this heartbeat will include a new elected Leader

Chapter VII

Task 03: Consensus

NOTE: this task is completely optional. It is only graded as a bonus part

You may have noticed that a lot of things could go wrong in a schema provided above, specifically race conditions and ability to lose some data due to replicas not being re-synced automatically between instances after some of them die.

You can try and solve that for some extra credit by either using an existing solution or writing some workaround yourself. Here are some options:

- Using existing Raft implementation (https://github.com/hashicorp/raft) or writing minimal implementation by yourself (https://www.youtube.com/watch?v=64Zp3tzNbpE)
- Utilizing external tools, like Zookeeper (https://zookeeper.apache.org/) or Etcd (https://etcd.io/)
- Выбор другого пути, например, Paxos (https://github.com/kkdai/paxos), какой-либо реализации блокчейна (например, https://tendermint.com/) или ваших собственных хаков.

...Надеюсь, теперь Пит и Майка не будут копаться в куче бумаг каждый раз, когда им нужно что-то найти. Наверное, Арти все равно это сделает, потому что иногда очень трудно бросить вызов силе привычки.

Но я думаю, что это было интересное путешествие, во время которого мы нашли по пути несколько крутых артефактов. Ты?

Глава VIII

Чтение

Распределенные системы MIT — RaftIntro Распределенные системы MIT — Raft1 Распределенные системы MIT — Raft2