Практическая работа с NoSQL базой данной

В процессе выполнения практической работы вы научитесь развёртывать NoSQL решение MongoDB на компьютер под управлением ОС Ubuntu 20.04, а затем напишите код, взаимодействующий с базой данных на языке программирования Python с использованием библиотеки PyMongo.

1. Установка MongoDB в Ubuntu 20.04

Введение

MongoDB (или Mongo) — это документоориентированная база данных с открытым исходным кодом, используемая во многих современных веб-приложениях. Она относится к базам данных типа NoSQL, поскольку не опирается на традиционную табличную структуру реляционных баз данных.

Вместо этого она хранит документы типа JSON с помощью динамических схем, т. е., в отличие от реляционных баз данных, MongoDB не требует наличия предопределенной схемы перед добавлением данных в базу. Вы можете менять схему в любое время и так часто, как вам нужно, без необходимости настройки новой базы данных с обновленной схемой.

В этом руководстве мы установим MongoDB на сервере Ubuntu 20.04, протестируем ее и узнаем, как управлять ею как службой `systemd`.

Предварительные требования

Для данного обучающего модуля вам потребуется следующее:

- Один сервер Ubuntu 20.04. На сервере должен быть пользователь без привилегий root с правами администратора и брандмауэр, настроенный с помощью UFW.

Шаг 1 — Установка MongoDB

Официальные репозитории пакетов Ubuntu включают стабильную версию MongoDB. Однако на момент написания этого документа версия MongoDB, доступная в репозиториях Ubuntu по умолчанию, — это версия 3.6, в то время как последний стабильный выпуск — это версия 4.4.
Чтобы получить самую последнюю версию этого программного обеспечения, вам нужно добавить выделенный репозиторий пакетов MongoDB в ваши источники APT. Теперь вы сможете установить `mongodb-org`, метапакет, который всегда указывает на последнюю версию MongoDB.

Для начала импортируйте публичный ключ GPG для последней стабильной версии MongoDB. Вы можете найти соответствующий файл ключа на сервере ключей MongoDB. Вам нужно найти файл, который включает номер последней стабильной версии и заканчивается на `.asc`. Например, если вы хотите установить MongoDB версии 4.4, необходимо искать файл с именем server-4.4.asc.

Нажмите правой кнопкой мыши на файл и выберите опцию link address (Скопировать адрес ссылки). Затем вставьте эту ссылку в следующую команду `curl`, заменив выделенный URL:

- `curl -fsSL https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add -`

cURL — это инструмент командной строки, который доступен во многих операционных системах и используется для передачи данных. Он читает все данные, хранящиеся в переданном URL, и выводит содержание в вывод системы. В следующем примере cURL выводит содержание файла ключа GPG, а затем вводит его в команду `sudo apt-key add -`, добавляя ключ GPG в ваш список доверенных ключей.

Также обратите внимание, что команда `curl` использует опции `-fsSL`, которые вместе указывают cURL не выполнять скрипт без каких-либо обязательств. Это означает, что, если по какой-либо причине cURL не может связаться с сервером GPG, либо сервер GPG не работает, он не добавит полученный код ошибки к вашему списку доверенных ключей случайно.

Эта команда покажет вывод `OK`, если ключ добавлен успешно:

```
Output

OK
```

Если вы хотите еще раз убедиться, что ключ добавлен корректно, вы можете сделать это, выполнив следующую команду:

- `apt-key list`

Эта команда возвращает ключ MongoDB в выводе:

```
Output

/etc/apt/trusted.gpg

--------------------

pub   rsa4096 2019-05-28 [SC] [expires: 2024-05-26]

      2069 1EEC 3521 6C63 CAF6  6CE1 6564 08E3 90CF B1F5

uid           [ unknown] MongoDB 4.4 Release Signing Key <packaging@mongodb.com>

. . .
```

На этом этапе ваша система APT все еще не знает, где искать пакет `mongodb-org`, который необходим для установки последней версии MongoDB.

На вашем сервере есть два места, где APT ищет онлайн-источники пакетов для загрузки и установки: файл `sources.list` и каталог `sources.list.d`. `sources.list` — это файл, который перечисляет активные источники данных APT (по одному источнику в строке, наиболее предпочтительные указываются первыми). Каталог `sources.list.d` позволяет добавлять такие записи `sources.list` в качестве отдельных файлов.

Запустите следующую команду, которая создает файл в каталоге `sources.list.d` под именем `mongodb-org-4.4.list`. В этом файле содержится только одна строка: `deb [`

`arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 multiverse`:

```
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.4.list
```

Эта единственная строка указывает APT все, что необходимо знать об источнике, и где его найти:

- `deb`: означает, что источник ссылается на обычную архитектуру Debian. В других случаях эта часть строки может выглядеть как `deb-src`. Это означает, что источник представляет исходный код дистрибутива Debian.
- `[ arch=amd64,arm64 ]`: указывает, в какие архитектуры загружать данные APT. В данном случае это архитектуры `amd64` и `arm64`.
- `https://repo.mongodb.org/apt/ubuntu`: это URI, представляющий местоположение данных APT. В данном случае URI указывает на адрес HTTPS, где находится официальный репозиторий MongoDB.
- `focal/mongodb-org/4.4`: репозитории Ubuntu могут содержать несколько разных выпусков. Это означает, что вам нужна только версия `4.4` пакета `mongodb-org`, доступная для выпуска Ubuntu `focal` («Focal Fossa» — это кодовое название Ubuntu 20.04).
- `multiverse`: эта часть указывает APT на один из четырех основных репозиториев Ubuntu. В данном случае — на репозиторий `multiverse`.

После запуска этой команды обновите локальный индекс пакетов вашего сервера, чтобы APT знал, где найти пакет `mongodb-org`:

```
sudo apt update
```

После этого вы можете установить MongoDB:

```
sudo apt install mongodb-org
```

При появлении запроса нажмите `Y`, а затем `ENTER`, чтобы подтвердить, что вы хотите установить пакет.

После завершения выполнения команды MongoDB будет установлена на вашей системе. Однако она еще не готова к использованию. На следующем шаге мы запустим MongoDB и убедимся, что она работает корректно.

Шаг 2 — Начало работы службы MongoDB и тестирование базы данных

Процесс установки, описанный на предыдущем шаге, автоматически настраивает MongoDB для запуска в качестве демона с помощью `systemd`, а это означает, что вы можете управлять MongoDB, используя различные команды `systemctl`. Но данная процедура установки не запускает службу автоматически.

Выполните следующую команду `systemctl`, чтобы запустить службу MongoDB:

```
sudo systemctl start mongod.service
```

Затем проверьте статус службы. Обратите внимание, что эта команда не включает `.service` в определение служебного файла. `systemctl` будет автоматически добавлять этот суффикс для любого аргумента, который вы передаете, если он еще не присутствует, поэтому нет необходимости включать его:

```
sudo systemctl status mongod
```

Эта команда возвращает вывод, аналогичный следующему, указывая на то, что служба запущена и работает:

```
Output
● mongod.service - MongoDB Database Server
     Loaded: loaded (/lib/systemd/system/mongod.service; disabled; vendor preset: enabled)
     Active: active (running) since Tue 2020-06-09 12:57:06 UTC; 2s ago
       Docs: https://docs.mongodb.org/manual
   Main PID: 37128 (mongod)
     Memory: 64.8M
     CGroup: /system.slice/mongod.service
             └─37128 /usr/bin/mongod --config /etc/mongod.conf
```

После подтверждения того, что служба работает нормально, установите активацию службы MongoDB при загрузке:

```
sudo systemctl enable mongod
```

Теперь вы можете убедиться, что база данных работает, подключившись к серверу базы данных и выполнив диагностическую команду. Следующая команда подключает к базе данных и выводит текущую версию, адрес сервера и порт. Также она выводит результат внутренней команды MongoDB `connectionStatus`:

```
mongo --eval 'db.runCommand({ connectionStatus: 1 })'
```

`connectionStatus` проверяет и возвращает статус подключения базы данных. Значение `1` поля `ok` в ответе означает, что сервер работает нормально:

```
Output
MongoDB shell version v4.4.0
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("1dc7d67a-0af5-4394-b9c4-8a6db3ff7e64") }
```

```
MongoDB server version: 4.4.0
{
    "authInfo" : {
        "authenticatedUsers" : [ ],
        "authenticatedUserRoles" : [ ]
    },
    "ok" : 1
}
```

Также обратите внимание, что база данных работает на порту `27017` на `127.0.0.1`, локальном циклическом адресе, представляющем локальный хост localhost. Это номер порта MongoDB по умолчанию.

На следующем шаге мы рассмотрим, как управлять экземпляром сервера MongoDB с помощью `systemd`.

Шаг 3 — Управление службой MongoDB

Как уже упоминалось ранее, процесс установки, описанный в шаге 1, настраивает MongoDB для работы в качестве службы `systemd`. Это означает, что вы можете управлять ею, используя стандартные команды `systemctl`, используемые для работы с другими службами системы Ubuntu.

Как уже упоминалось ранее, команда `systemctl status` проверяет статус службы MongoDB:

- `sudo systemctl status mongod`

Вы можете остановить работу службы в любое время с помощью следующей команды:

- `sudo systemctl stop mongod`

Чтобы запустить службу после остановки, введите команду:

- `sudo systemctl start mongod`

Также вы можете перезапустить сервер, когда он уже работает:

- `sudo systemctl restart mongod`

На шаге 2 вы установили активацию MongoDB автоматически при загрузке сервера. Если вы хотите отключить автоматический запуск, введите:

- `sudo systemctl disable mongod`

Для повторной установки активации при загрузке запустите команду `enable` снова:

```
sudo systemctl enable mongod
```

Заключение

В ходе выполнения вы добавили официальный репозиторий MongoDB в ваш экземпляр APT и установили последнюю версию MongoDB. Затем вы протестировали функциональность Mongo и попробовали использовать некоторые команды `systemctl`.

2. Интеграция MongoDB с Python с использованием PyMongo

В этой части мы рассмотрим MongoDB как хранилище данных с точки зрения Python. Для этого мы напишем простой сценарий, демонстрирующий, чего мы можем достичь, и любые преимущества, которые мы можем извлечь из этого.

Веб-приложения, как и многие другие программные приложения, питаются от данных. Организация и хранение этих данных важны, поскольку они определяют, как мы взаимодействуем с различными приложениями, которые находятся в нашем распоряжении. Вид обрабатываемых данных также может влиять на то, как мы предпринимаем этот процесс.

Базы данных позволяют нам организовывать и хранить эти данные, а также контролировать, как мы храним, получаем доступ и защищаем информацию.

Базы данных NoSQL

Существует два основных типа баз данных - реляционные и нереляционные базы данных.

Реляционные базы данных позволяют нам хранить, получать доступ и манипулировать данными по отношению к другому фрагменту данных в базе данных. Данные хранятся в организованных таблицах со строками и столбцами со связями, связывающими информацию между таблицами. Для работы с этими базами данных мы используем язык структурированных запросов (SQL), например MySQL и PostgreSQL.

Нереляционные базы данных хранят данные ни в реляционной, ни в табличной форме, как в реляционных базах данных. Они также называются базами данных NoSQL, так как мы не используем SQL для взаимодействия с ними.

MongoDB и когда его использовать

MongoDB - это хранилище документов и нереляционная база данных. Это позволяет нам хранить данные в коллекциях, которые составлены из документов.

В MongoDB документ представляет собой просто JSON-подобный двоичный формат сериализации, называемый BSON или Binary-JSON, и имеет максимальный размер 16 мегабайт. Это ограничение размера используется для обеспечения эффективного использования памяти и полосы пропускания во время передачи.

MongoDB также предоставляет спецификацию GridFS в случае необходимости хранить файлы, размер которых превышает установленное ограничение.

Документы состоят из пар ключ-значение, как в обычных данных JSON. Однако этот формат BSON также может содержать больше типов данных, таких как `Date` и `Binary Data`. BSON был разработан, чтобы быть легким, легко проходимым и эффективным при кодировании и декодировании данных в и из BSON.

Будучи хранилищем данных NoSQL, MongoDB позволяет нам пользоваться преимуществами использования нереляционной базы данных по сравнению с реляционной. Одним из преимуществ является то, что он обеспечивает высокую масштабируемость за счет эффективного масштабирования по горизонтали за счет разделения данных и размещения их на нескольких компьютерах.

MongoDB также позволяет нам хранить большие объемы структурированных, полуструктурированных и неструктурированных данных без необходимости поддерживать отношения между ними.

Как и у любого другого решения, у MongoDB есть свои недостатки. Первый заключается в том, что он не поддерживает отношения между сохраненными данными. Из-за этого трудно выполнять транзакции ACID, которые обеспечивают согласованность.

Сложность увеличивается при попытке поддержки транзакций ACID. MongoDB, как и другие хранилища данных NoSQL, не настолько развит, как реляционные базы данных, и это может затруднить поиск экспертов.

Нереляционная природа MongoDB делает его идеальным для хранения данных в определенных ситуациях по сравнению с его реляционными аналогами. Например, сценарий, где MongoDB является более подходящим, чем реляционная база данных, - это когда формат данных является гибким и не имеет отношений.

С гибкими / нереляционными данными нам не нужно поддерживать свойства ACID при хранении данных, в отличие от реляционных баз данных. MongoDB также позволяет нам легко масштабировать данные в новые узлы.

Однако, несмотря на все свои преимущества, MongoDB не идеален, когда наши данные носят реляционный характер. Например, если мы храним записи клиентов и их заказы.

В этой ситуации нам понадобится реляционная база данных для поддержания отношений между нашими данными, которые важны. Также не подходит использовать MongoDB, если нам нужно соблюдать свойства ACID.

Взаимодействие с MongoDB через Mongo Shell

MongoDB Server поставляется с оболочкой Mongo, которую мы можем использовать для взаимодействия с сервером через терминал.

Чтобы активировать оболочку, просто введите в терминале `mongo`. Вы получите информацию о настройке сервера MongoDB, включая версию MongoDB и Mongo Shell, а также URL-адрес сервера.

Например, наш сервер работает на:

```
mongodb://127.0.0.1:27017
```

В MongoDB база данных используется для хранения коллекций, содержащих документы. Через оболочку Mongo мы можем создать новую базу данных или переключиться на существующую, используя команду `use`:

```
> use SeriesDB
```

Каждая операция, которую мы выполняем после этого, будет выполняться в нашей базе данных `SeriesDB`. В базе данных мы будем хранить коллекции, которые похожи на таблицы в реляционных базах данных.

Например, для нашихцелей давайте добавим несколько рядов в базу данных:

```
> db.series.insertMany([
... { name: "Game of Thrones", year: 2012},
... { name: "House of Cards", year: 2013 },
... { name: "Suits", year: 2011}
... ])
```

И у нас получится:

```
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("5e300724c013a3b1a742c3b9"),
        ObjectId("5e300724c013a3b1a742c3ba"),
        ObjectId("5e300724c013a3b1a742c3bb")
    ]
}
```

Для извлечения всех документов, хранящихся в нашей коллекции series, мы используем db.inventory.find({}), эквивалент SQL SELECT * FROM series. Передача пустого запроса (т.е. {}) вернет все документы:

```
> db.series.find({})


{ "_id" : ObjectId("5e3006258c33209a674d1d1e"), "name" : "The Blacklist", "year" : 2013 }
{ "_id" : ObjectId("5e300724c013a3b1a742c3b9"), "name" : "Game of Thrones", "year" : 2012
}
{ "_id" : ObjectId("5e300724c013a3b1a742c3ba"), "name" : "House of Cards", "year" : 2013
}
{ "_id" : ObjectId("5e300724c013a3b1a742c3bb"), "name" : "Suits", "year" : 2011 }
```

Мы также можем запросить данные, используя условие равенства, например, чтобы вернуть все сериалы, которые были показаны в 2013 году:

```
> db.series.find({ year: 2013 })
{ "_id" : ObjectId("5e3006258c33209a674d1d1e"), "name" : "The Blacklist", "year" : 2013 }
{ "_id" : ObjectId("5e300724c013a3b1a742c3ba"), "name" : "House of Cards", "year" : 2013
}
```

SQL-эквивалент будет SELECT * FROM series WHERE year=2013.

MongoDB также позволяет нам обновлять отдельные документы с помощью db.collection.UpdateOne() или выполнять пакетное обновление с помощью db.collection.UpdateMany(). Например, чтобы обновить год выпуска для Suits:

```
> db.series.updateOne({ name: "Suits" }, {
    $set: { year: 2010 }
})


{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

Наконец, для удаления документов, Mongo Shell предлагает функции db.collection.deleteOne() и db.collection.deleteMany().

Например, чтобы удалить все премьеры 2012, мы запустили:

```
> db.series.deleteMany({ year: 2012 })
{ "acknowledged" : true, "deletedCount" : 2 }
```

Дополнительную информацию об операциях CRUD над MongoDB можно найти в онлайн-справке, включая дополнительные примеры, выполнение операций с условиями, атомарность и сопоставление понятий SQL с понятиями и терминологией MongoDB.

Интеграция Python с MongoDB

MongoDB предоставляет драйверы и инструменты для взаимодействия с хранилищем данных MongoDB с использованием различных языков программирования, включая Python, JavaScript, Java, Go и C#.

PyMongo является официальным драйвером MongoDB для Python, и мы будем использовать его для создания простого скрипта, который мы будем использовать для манипулирования данными, хранящимися в нашей базе данных SeriesDB.

С установленными Python 3.6 + и Virtualenv, давайте создадим виртуальную среду для приложения и установить PyMongo с помощью pip:

```
virtualenv --python=python3 env --no-site-packages

source env/bin/activate

pip install pymongo
```

Используя PyMongo, мы собираемся написать простой скрипт, который мы можем выполнить для выполнения различных операций с нашей базой данных MongoDB.

Подключение к MongoDB

Сначала мы импортируем pymongo в mongo_db_script.py и создаем клиент, подключенный к нашему локально работающему экземпляру MongoDB:

```
import pymongo


# Create the client

client = MongoClient('localhost', 27017)


# Connect to our database

db = client['SeriesDB']


# Fetch our series collection

series_collection = db['series']
```

Мы создали клиент, который подключается к нашему серверу MongoDB, и использовали его для получения нашей базы данных 'SeriesDB'. Затем мы получаем нашу коллекцию 'series' и сохраняем ее в переменной.

Создание документов

Чтобы сделать наш сценарий более удобным, мы напишем функции, которые позволяют нам легко манипулировать данными с PyMongo. Мы будем использовать Python словари для представления документов и передадим эти словари нашим функциям. Во-первых, давайте создадим функцию для вставки данных в нашу коллекцию 'series':

```
# Imports truncated for brevity


def insert_document(collection, data):
```

```
    """ Function to insert a document into a collection and
    return the document's id.
    """
    return collection.insert_one(data).inserted_id
```

Эта функция получает коллекцию и словарь данных и вставляет данные в предоставленную коллекцию. Затем функция возвращает идентификатор, который мы можем использовать для точного запроса отдельного объекта из базы данных.

Следует также отметить, что MongoDB добавляет дополнительный ключ _id к нашим документам, когда они не предоставляются, при создании данных.

Теперь давайте попробуем добавить шоу, используя нашу функцию:

```
new_show = {
    "name": "FRIENDS",
    "year": 1994
}
print(insert_document(series_collection, new_show))
```

В консоли получим:

```
5e4465cfdcbbdc68a6df233f
```

Когда мы запускаем наш скрипт, _id новой записи распечатывается в терминале, и мы можем использовать этот идентификатор для получения записи позже.

Мы можем предоставить значение _id вместо того, чтобы назначать его автоматически, что мы предоставили бы в словаре:

```
new_show = {
    "_id": "1",
    "name": "FRIENDS",
    "year": 1994
}
```

И если бы мы попытались сохранить документ с существующим _id, нас встретит ошибка, подобная следующей:

```
DuplicateKeyError: E11000 duplicate key error index: SeriesDB.series.$id dup key: { : 1}
```

Получение документов

Для извлечения документов из базы данных мы будем использовать find_document(), которая запрашивает нашу коллекцию для одного или нескольких документов. Наша функция получит словарь, который содержит элементы, по которым мы хотим фильтровать, и необязательный аргумент, чтобы указать, хотим ли мы один документ или несколько документов:

```
def find_document(collection, elements, multiple=False):
    """ Function to retrieve single or multiple documents from a provided
    Collection using a dictionary containing a document's elements.
    """
    if multiple:
        results = collection.find(elements)
        return [r for r in results]
    else:
        return collection.find_one(elements)
```

А теперь давайте использовать эту функцию, чтобы найти некоторые документы:

```
result = find_document(series_collection, {'name': 'FRIENDS'})
print(result)
```

При выполнении нашей функции мы не предоставили параметр `multiple`, и в результате мы получили один документ:

```
{'_id': ObjectId('5e3031440597a8b07d2f4111'), 'name': 'FRIENDS', 'year': 1994}
```

Когда параметр `multiple` предоставлен, результатом является список всех документов в нашей коллекции, для которых установлен атрибут `name` как `FRIENDS`.

Обновление документов

Наша следующая функция `update_document()`, будет использоваться для обновления одного конкретного документа. Мы будем использовать `_id` документа и коллекцию, к которой он принадлежит, при его поиске:

```
def update_document(collection, query_elements, new_values):
    """ Function to update a single document in a collection.
    """
    collection.update_one(query_elements, {'$set': new_values})
```

Теперь давайте вставим документ:

```
new_show = {
    "name": "FRIENDS",
    "year": 1995
}
id_ = insert_document(series_collection, new_show)
```

Сделав это, давайте обновим документ, используя возвращенный `_id`:

```
update_document(series_collection, {'_id': id_}, {'name': 'F.R.I.E.N.D.S'})
```

И, наконец, давайте посмотрим его, чтобы убедиться, что новое значение было установлено, и напечатаем результат:

```
result = find_document(series_collection, {'_id': id_})
print(result)
```

Когда мы выполняем наш скрипт, мы увидим, что наш документ обновлен:

```
{'_id': ObjectId('5e30378e96729abc101e3997'), 'name': 'F.R.I.E.N.D.S', 'year': 1995}
```

Удаление документов

И, наконец, давайте напишем функцию для удаления документов:

```
# Imports and previous code truncated for brevity


def delete_document(collection, query):
    """ Function to delete a single document from a collection.
    """
    collection.delete_one(query)
```

Поскольку мы используем метод `delete_one`, только один документ может быть удален за вызов, даже если запрос соответствует нескольким документам.

Теперь давайте используем функцию для удаления записи:

```
delete_document(series_collection, {'_id': id_})
```

Если мы попытаемся получить тот же документ:

```
result = find_document(series_collection, {'_id': id_})
print(result)
```

Результатом будет:

```
None
```

Заключение

Мы выделили и использовали несколько методов `PyMongo` взаимодействия с нашим сервером MongoDB из скрипта Python. Тем не менее, мы не использовали все методы, доступные для нас через модуль.

Все доступные методы можно найти в официальной документации PyMongo и классифицировать в соответствии с подмодулями.

Мы написали простой скрипт, который выполняет элементарные функции CRUD для базы данных MongoDB. Хотя мы могли бы импортировать функции в более сложной кодовой базе или, например, в приложение Flask / Django, в этих средах есть библиотеки для достижения тех же результатов. Эти библиотеки делают его проще, удобнее и помогают нам более безопасно подключаться к MongoDB.