**Spring Boot - Build a CRUD REST API with MongoDB Atlas(GUI)**

MongoDB Atlas is a fully managed cloud database that handles all the complexity of deploying, managing, and healing your deployments on the cloud service provider of your choice (AWS , Azure, and GCP). MongoDB Atlas is the best way to deploy, run, and scale MongoDB in the cloud.

In this tutorial you will understand how to connect your springboot project with mongodb atlas and also, we will perform one **CRUD** operation using mongodb atlas. Basically, mongodb atlas is database as service which offers you if you are using mongodb atlas trial version, then by default you will get 512 mb of storage. You will also get the feature called Monitoring and alerting. Then you can improve the performance of your query by using performance optimization tools. And all the data is in the form of end-to-end encryption. And the last feature is data visualization with atlas charts. When I say data visualization, you can visualize your data in the form of charts and graphs if you are using this. When I say scale up and scale down, it means simply you can consider the configuration to access this mongodb atlas is very simple. Directly go to the mongodb atlas user interface, then play with your configuration, do the changes, whatever you need as part of your requirement.

Okay, so I found this specific mongodb atlas product is really cool. Because it is giving option to you to choose the cloud infrastructure structure. What you want to use. Whether you want to use the aws Cloud or azure Cloud or Google Cloud, it is up to you.

**MongoDB Atlas Features:-**

* 512 MB of storage
* Comprehensive monitoring and alerting
* Performance optimization tools
* End-to-End-encryption
* Data visualization with Atlas Charts

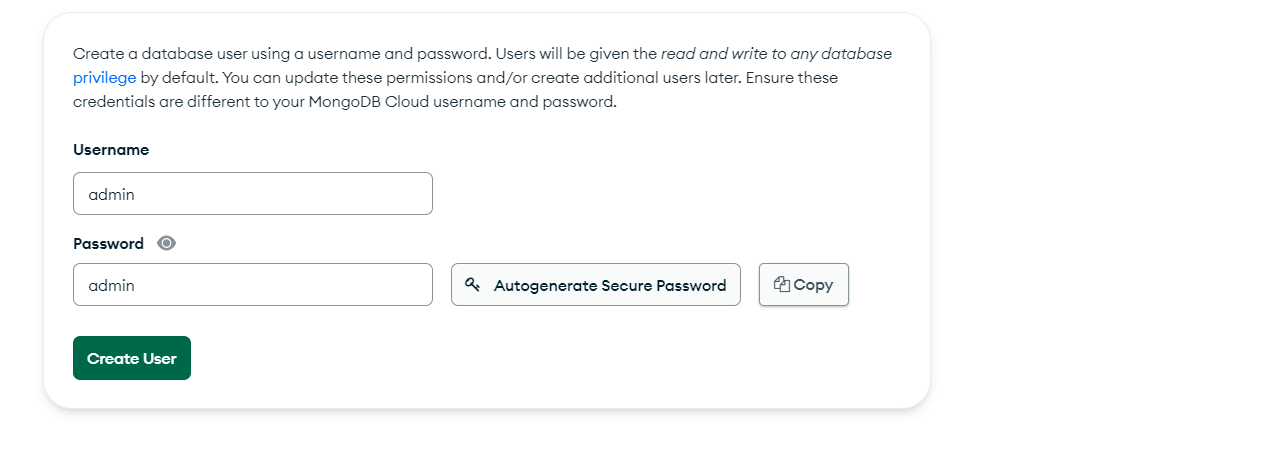
So, before we start developing the code, first you need to set up your mongodb atlas cluster.

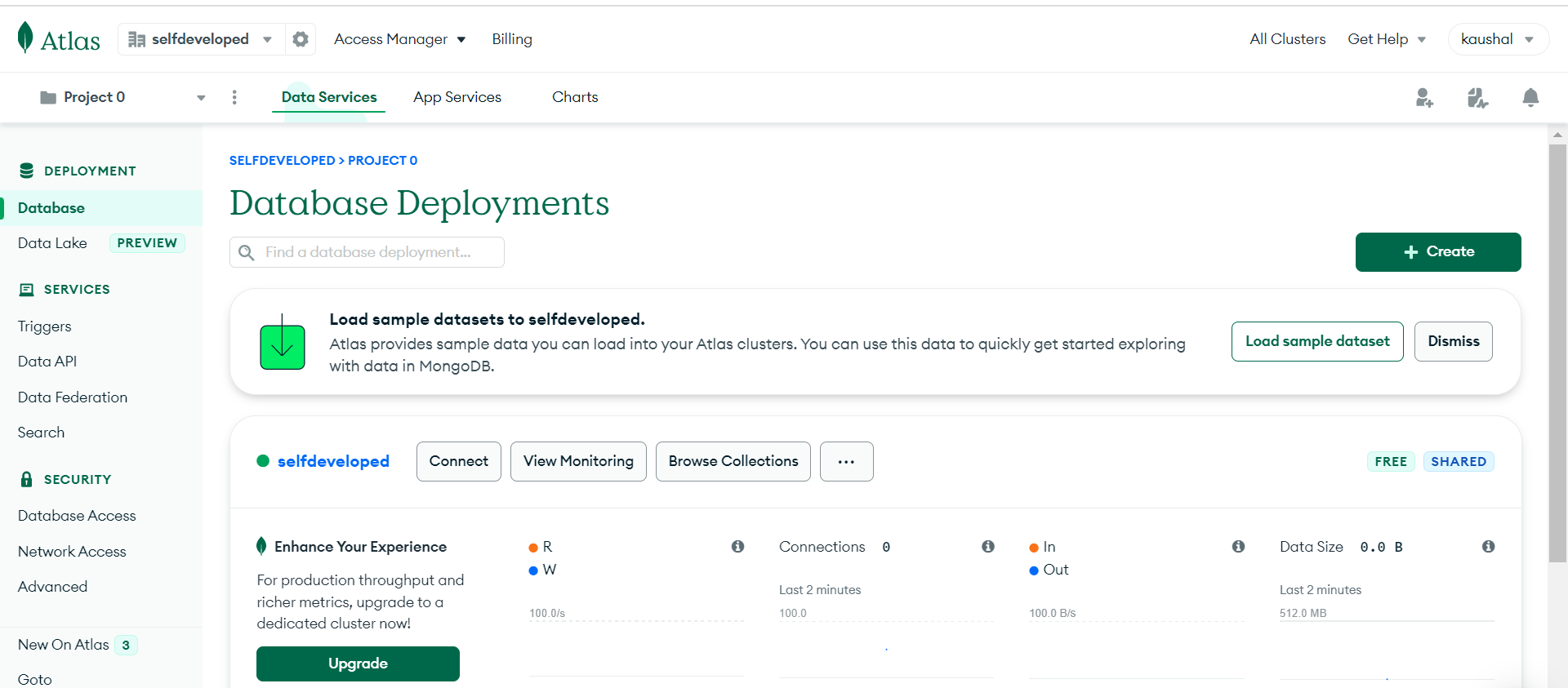
<https://www.mongodb.com/cloud/atlas/register>

do the sign Up and verify the account…

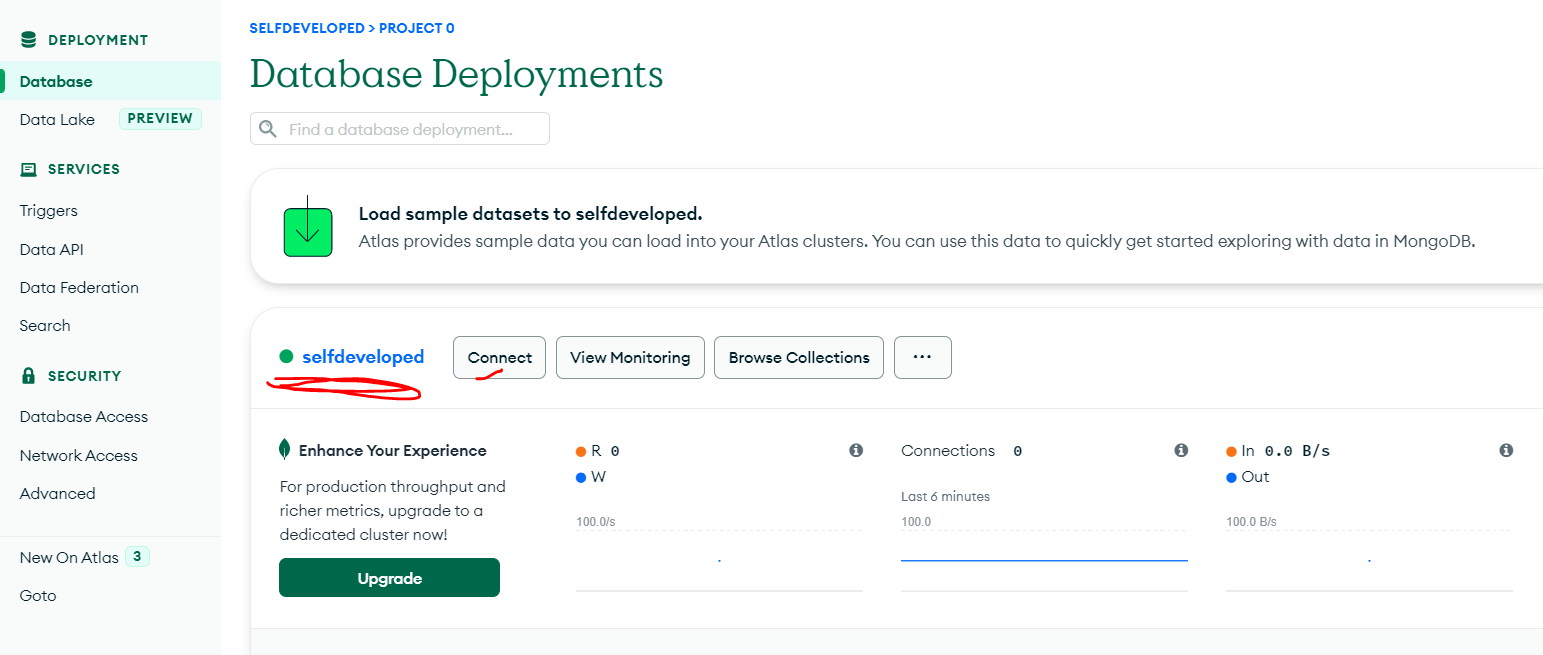
giving username and password as admin

I have sign Up with email: [**mongodb8707@gmail.com**](mailto:mongodb8707@gmail.com)password: **KrTum5h87@7**





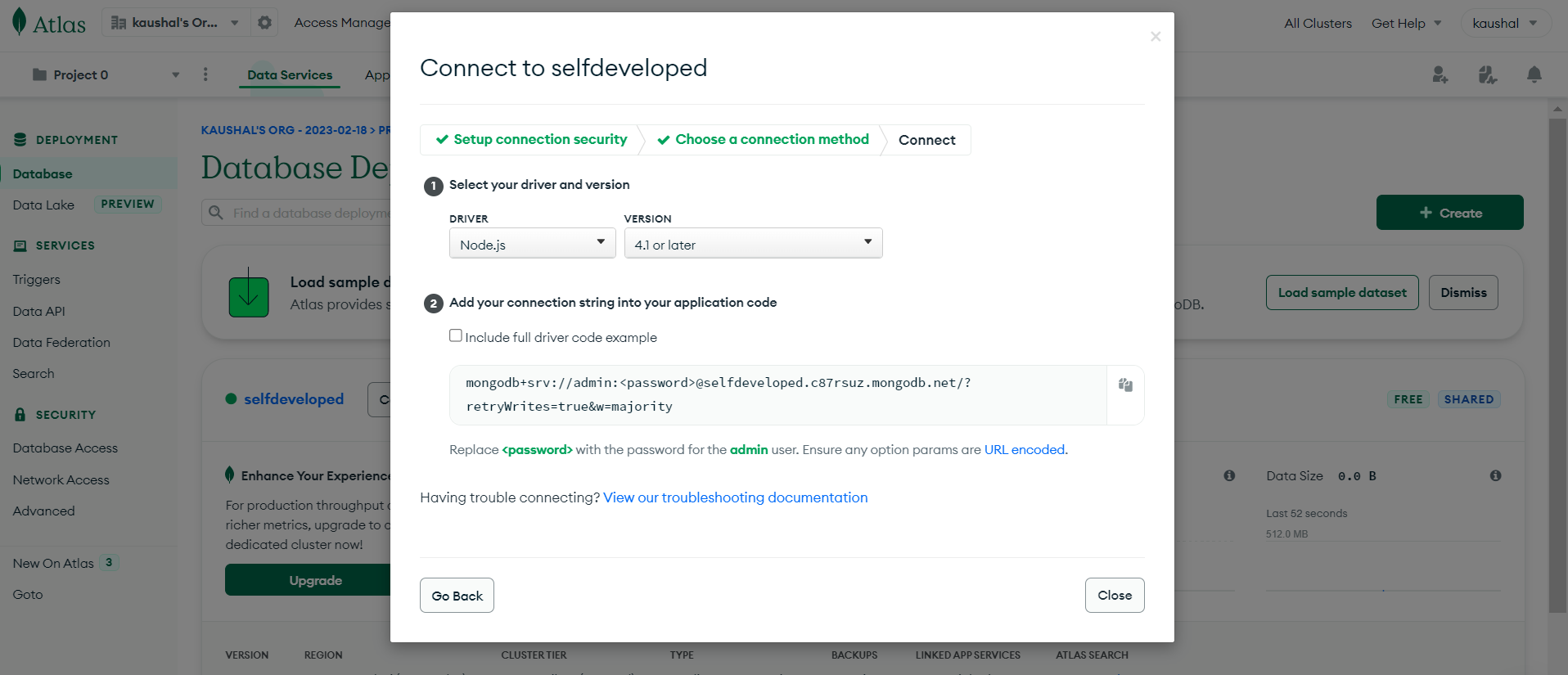
Here selfdeveloped is our collection name and then click to **connect**…



Now connect your application to your cluster using MongoDB’s native driver. When we click on connect, we will get the data source information. This is what your data source **URL** or **Uri**. Using this our spring boot application can connect the mongodb atlas which is hosted to the aws cloud infrastructure. Just copy this for future reference.

**DB URI –**

**mongodb+srv://admin:admin@selfdeveloped.c87rsuz.mongodb.net/?retryWrites=true&w=majority**



I wan to create the database and collection through the application so we will add as per the document.

So, we have done with our DB atlas server. Now let’s go to STS and create an application.

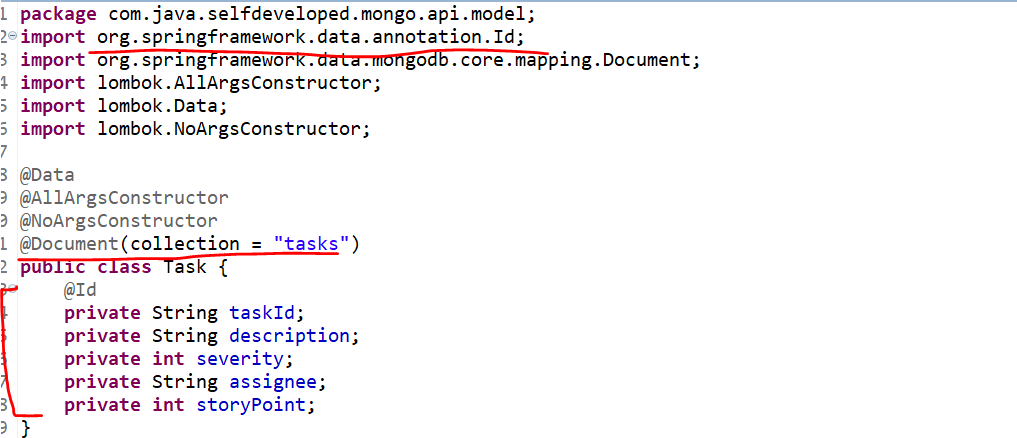
Let’s create an application…

**Application** – **springboot-mongo-atlas**

**Dependencies** – **Spring Web, Spring Data MongoDB, Lombok**

Let’s create a model with name **Task**. I am using Task as a Document since I am using MongoDB and it’s a NoSQL Database. I need to store these as a document.

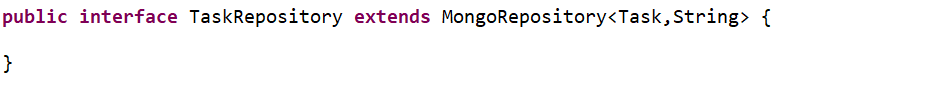
So first I need to use @**Document,** and then I need to give a collection name. If we won’t give collection name by default it will take collection name as your class name.



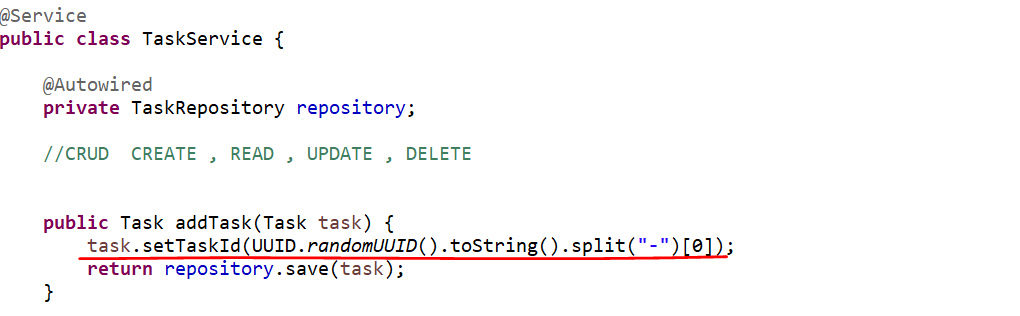
Now I need to annotate **taskId** as a primary key so let’s use and make sure this Id should be imported from **org.springframework.data.annotation**.

Let’s create a repository…

While creating our repository we need to extend this interface with **MongoRepository** instead of **JPARepository**.



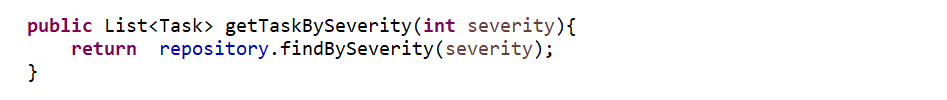
Now let’s create a Service class.



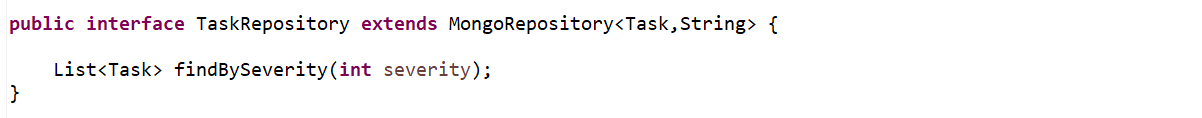
Before save that since if you observe into the Task Document, I have created taskId as a primary key. But in MongoDB Id can’t be auto generated directly. So as of now let’s use this taskId as an UUID.



Now if we want to fetch the data other than Id then we need to create that method with prefix **findBy+ fieldname** .we can get or retrieve all data as a list based on a specific field then we need to create that method in a Repository.

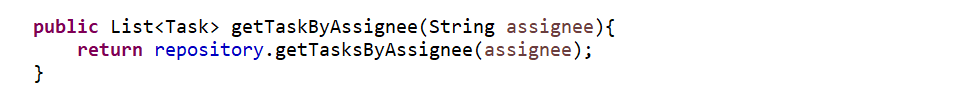


----------------------------------------------------------------------------------------------------

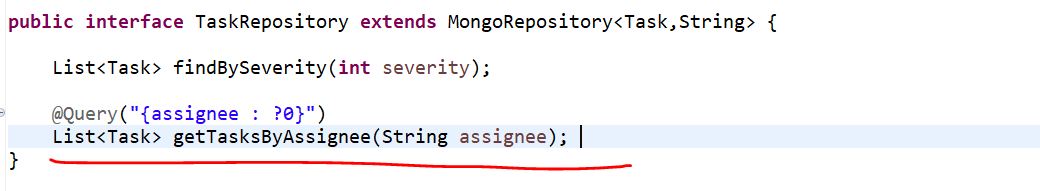


We can also write a Query to fetch the details based on some particular field and we create a method in our Repository.

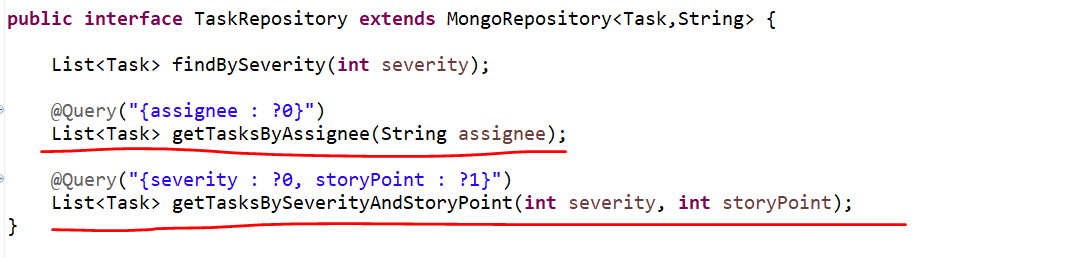
I just want to fetch all the tasks based on the particular field using @**Query**.

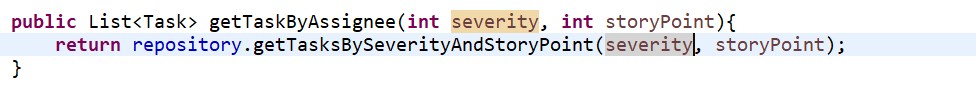


--------------------------------------------------------------------------------------------------------



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





Now we have created our service class…

**@Service**

**public class TaskService {**

**@Autowired**

**private TaskRepository repository;**

**//CRUD CREATE , READ , UPDATE , DELETE**

**public Task addTask(Task task) {**

**task.setTaskId(UUID.randomUUID().toString().split("-")[0]);**

**return repository.save(task);**

**}**

**public List<Task> findAllTasks() {**

**return repository.findAll();**

**}**

**public Task getTaskByTaskId(String taskId){**

**return repository.findById(taskId).get();**

**}**

**public List<Task> getTaskBySeverity(int severity){**

**return repository.findBySeverity(severity);**

**}**

**public List<Task> getTaskByAssignee(String assignee){**

**return repository.getTasksByAssignee(assignee);**

**}**

**public Task updateTask(Task taskRequest){**

**//get the existing document from DB**

**// populate new value from request to existing object/entity/document**

**Task existingTask = repository.findById(taskRequest.getTaskId()).get();**

**existingTask.setDescription(taskRequest.getDescription());**

**existingTask.setSeverity(taskRequest.getSeverity());**

**existingTask.setAssignee(taskRequest.getAssignee());**

**existingTask.setStoryPoint(taskRequest.getStoryPoint());**

**return repository.save(existingTask);**

**}**

**public String deleteTask(String taskId){**

**repository.deleteById(taskId);**

**return taskId+" task deleted from dashboard ";**

**}**

**public List<Task> getTaskByAssignee(int severity, int storyPoint){**

**return repository.getTasksBySeverityAndStoryPoint(severity, storyPoint);**

**}**

**}**

Let’s create a controller class…

@RestController

@RequestMapping("/tasks")

**public** **class** TaskController {

@Autowired

**private** TaskService service;

@PostMapping

@ResponseStatus(value = HttpStatus.***CREATED***)

**public** Task createTask(@RequestBody Task task){

**return** service.addTask(task);

}

@GetMapping

**public** List<Task> getTasks() {

**return** service.findAllTasks();

}

@GetMapping("/{taskId}")

**public** Task getTask(@PathVariable String taskId){

**return** service.getTaskByTaskId(taskId);

}

@GetMapping("/severity/{severity}")

**public** List<Task> findTaskUsingSeverity(@PathVariable **int** severity){

**return** service.getTaskBySeverity(severity);

}

@GetMapping("/assignee/{assignee}")

**public** List<Task> getTaskByAssignee(@PathVariable String assignee){

**return** service.getTaskByAssignee(assignee);

}

@GetMapping("/severity\_storypoint/{severity}/{storypoint}")

**public** List<Task> getTaskByAssignee(@PathVariable **int** severity,

@PathVariable **int** storypoint){

**return** service.getTaskBySeverityAndStoryPoint(severity,storypoint);

}

@PutMapping

**public** Task modifyTask(@RequestBody Task task){

**return** service.updateTask(task);

}

@DeleteMapping("/{taskId}")

**public** String deleteTask(@PathVariable String taskId){

**return** service.deleteTask(taskId);

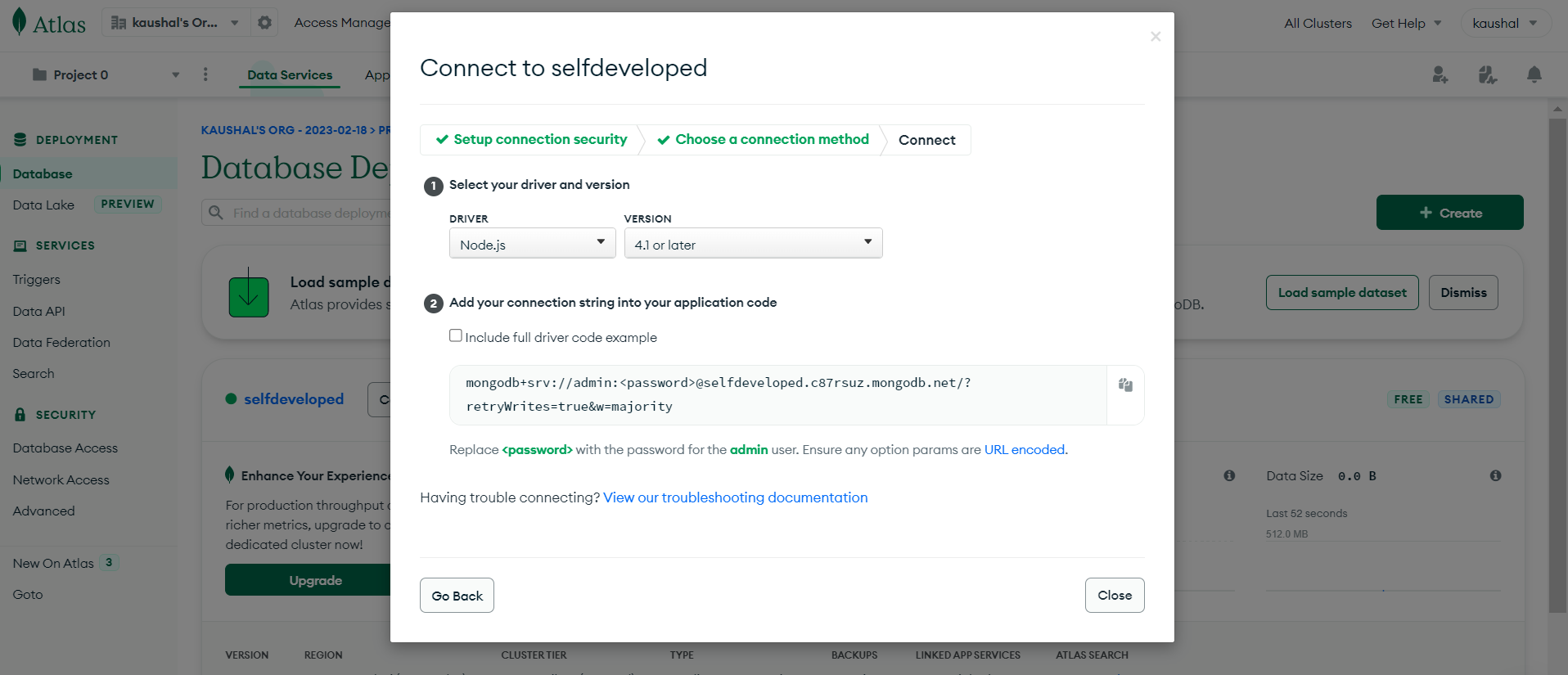
}

}

Now if you remember we have created an account in MongoDB Atlas which is there in Cloud. So now go to MongoDB Atlas cloud and copy the data source URL and paste it in your **application**.**yml** file.

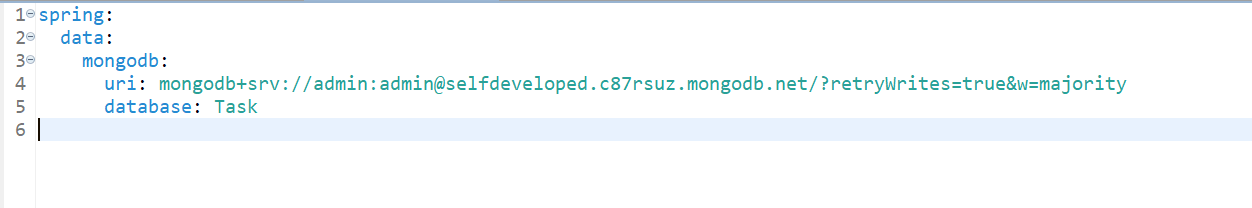
**DB URI –**

**mongodb+srv://admin:admin@selfdeveloped.c87rsuz.mongodb.net/?retryWrites=true&w=majority**

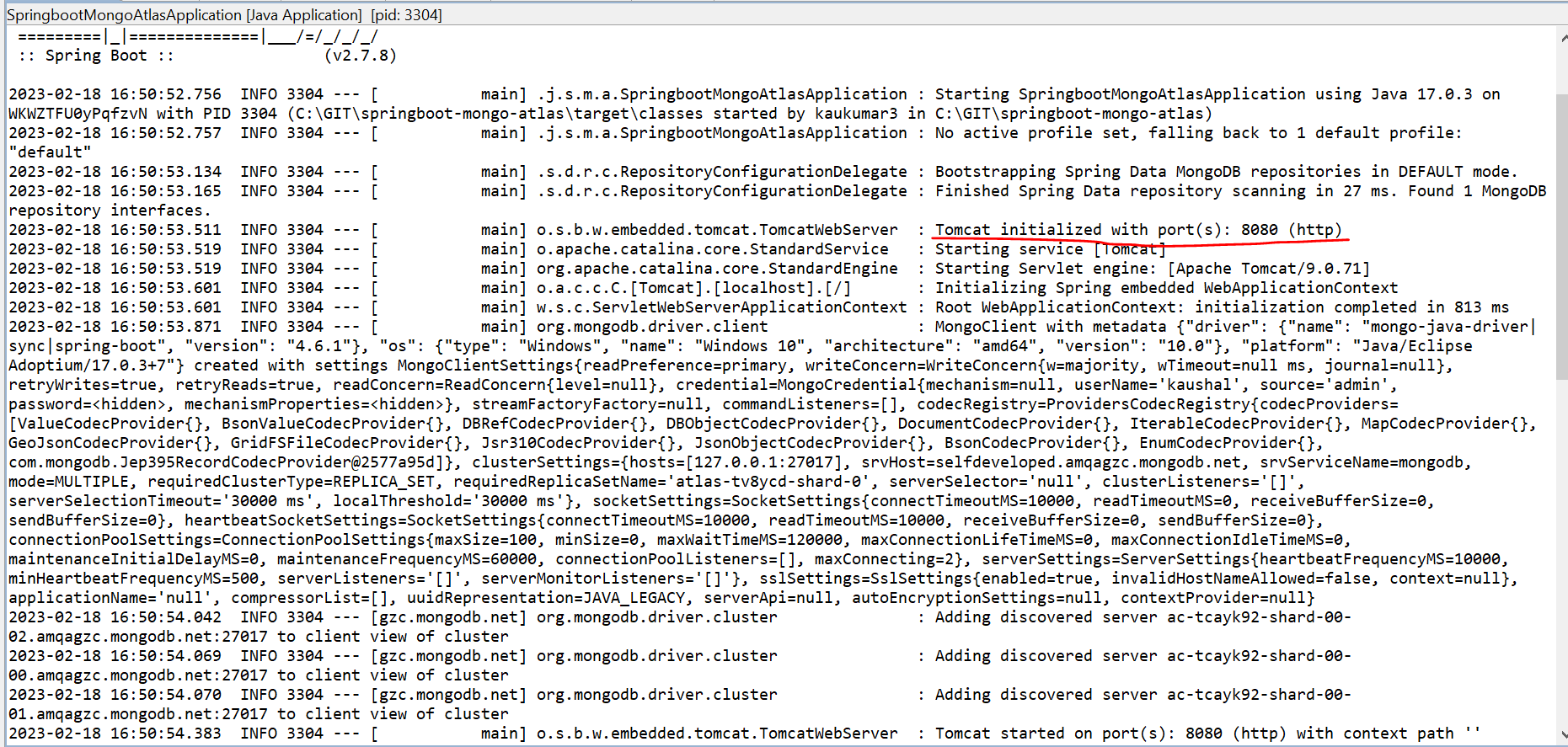


**mongodb+srv://admin:admin@selfdeveloped.c87rsuz.mongodb.net/?retryWrites=true&w=majority**

admin is my password , admin is my username and selfdeveloped is my cluster name. so, this is the mongodb atlas Uri which is hosted in aws cloud.



Let’s Run our application -



Now our service is up. Now let’s call the Rest API for create task…so we need to pass Task json object as a Payload from postman. As taskId we are generating using UUID so let’s pass rest of the fields.

**POST** http://localhost:8080/tasks

**{**

**"description" : "Implement Security",**

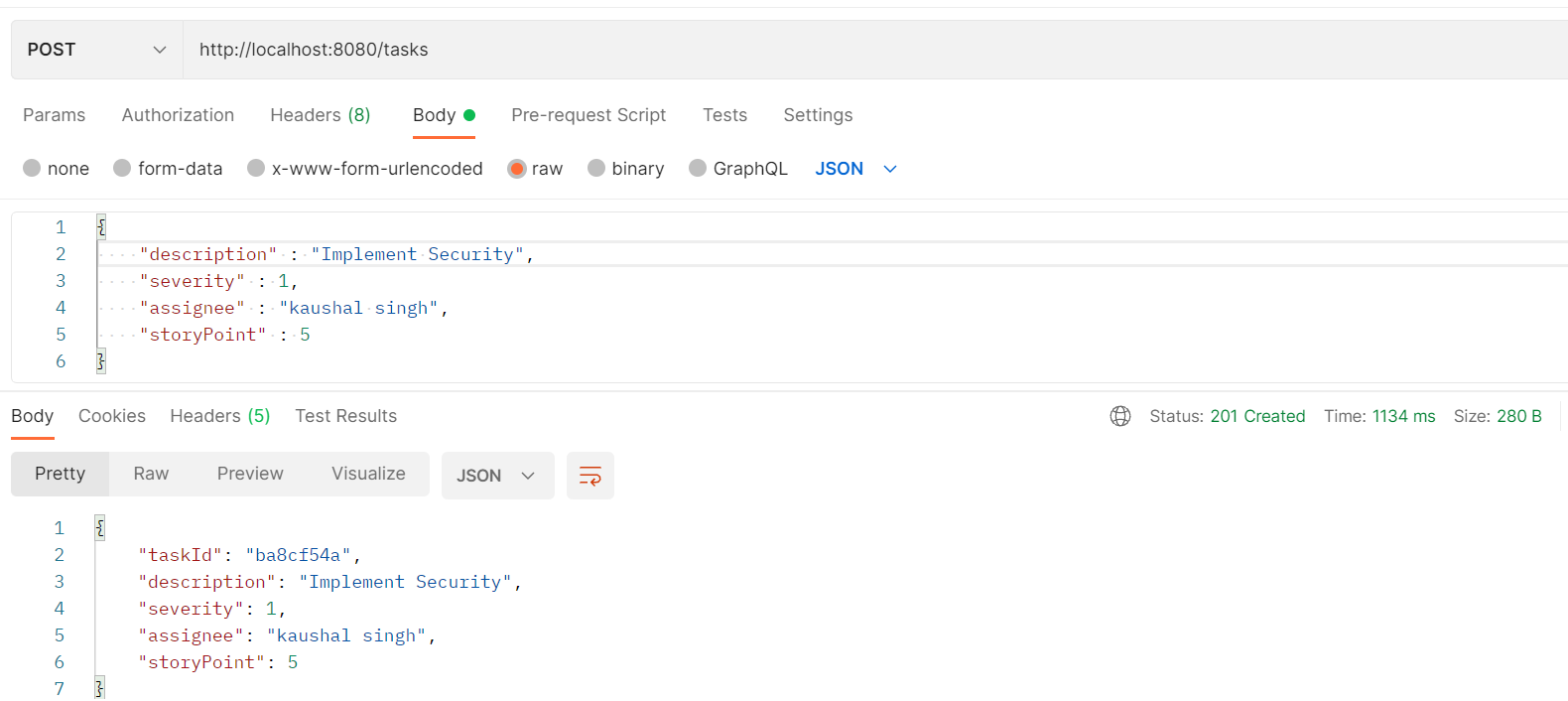
**"severity" : 1,**

**"assignee" : "kaushal singh",**

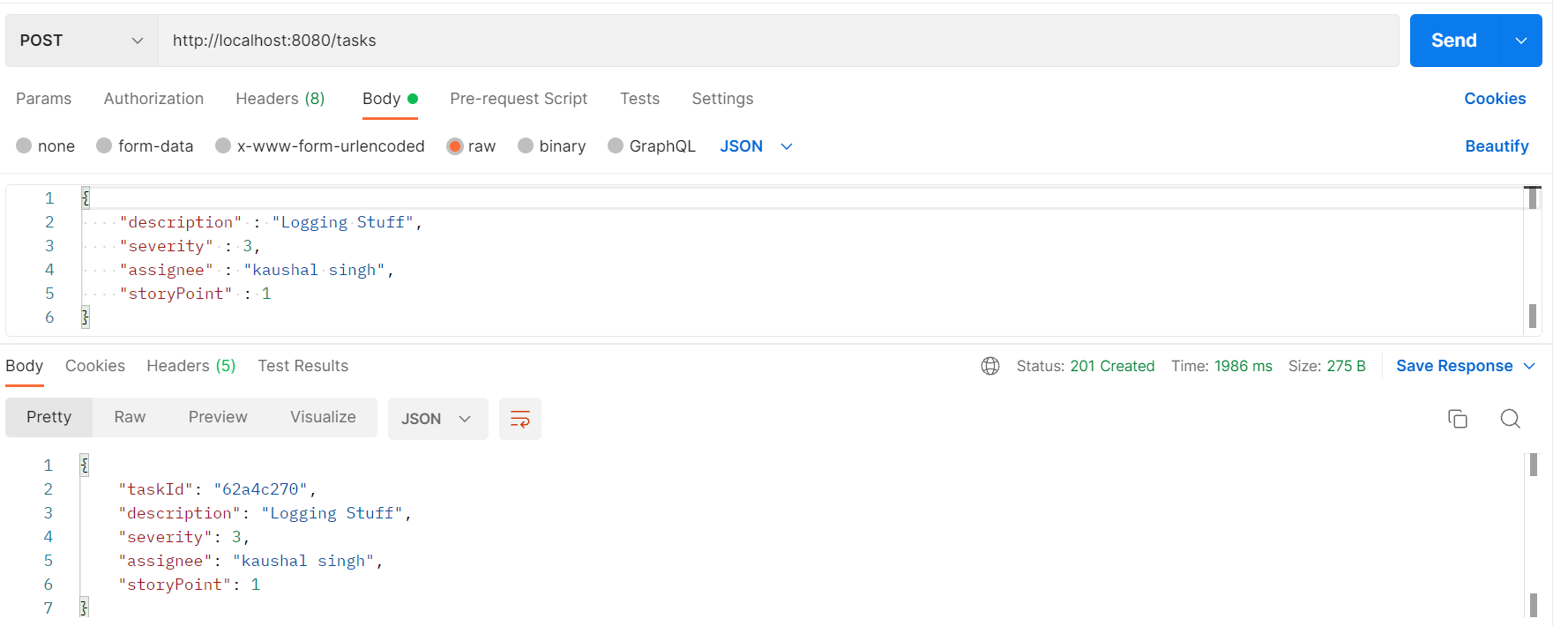
**"storyPoint" : 5**

**}**

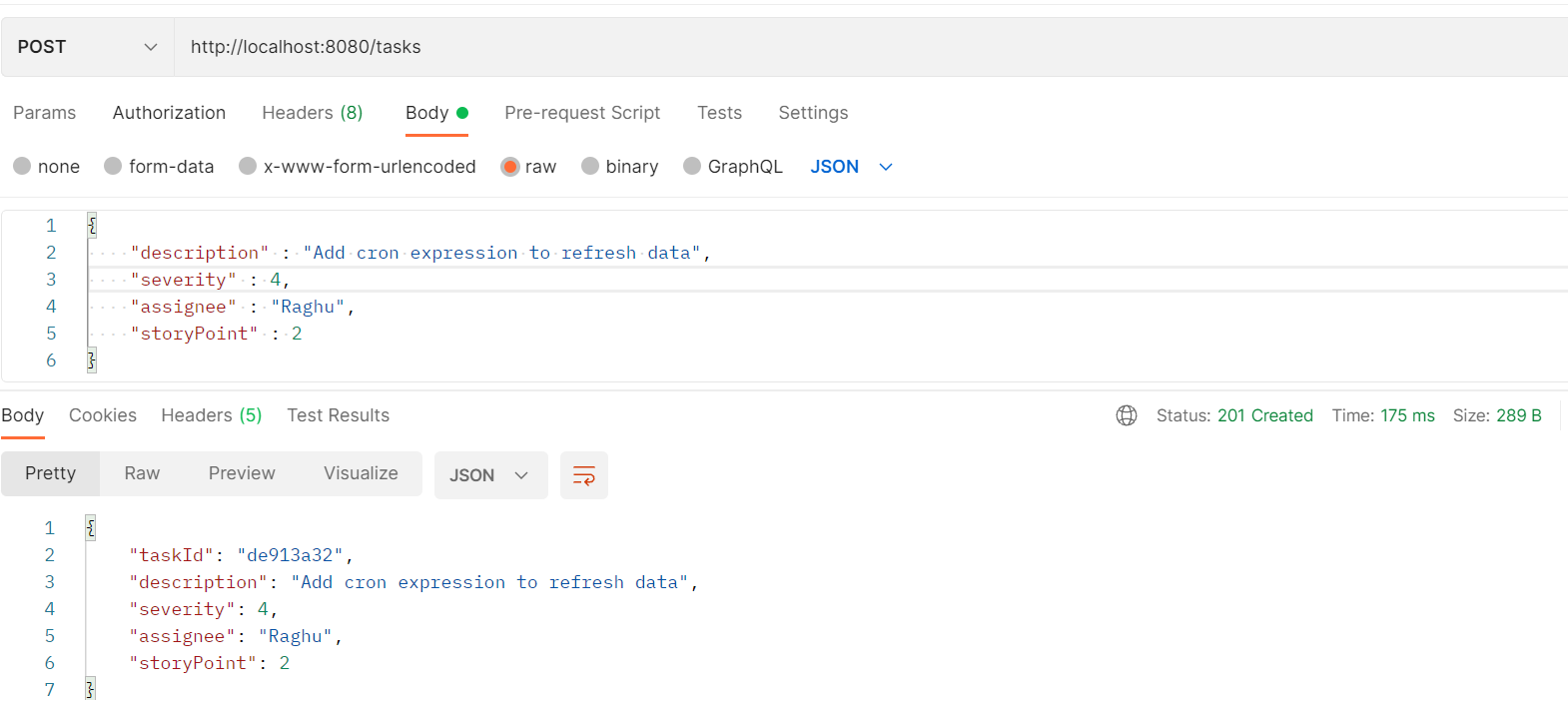
**-------------------------------------------------------------------------------------**



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

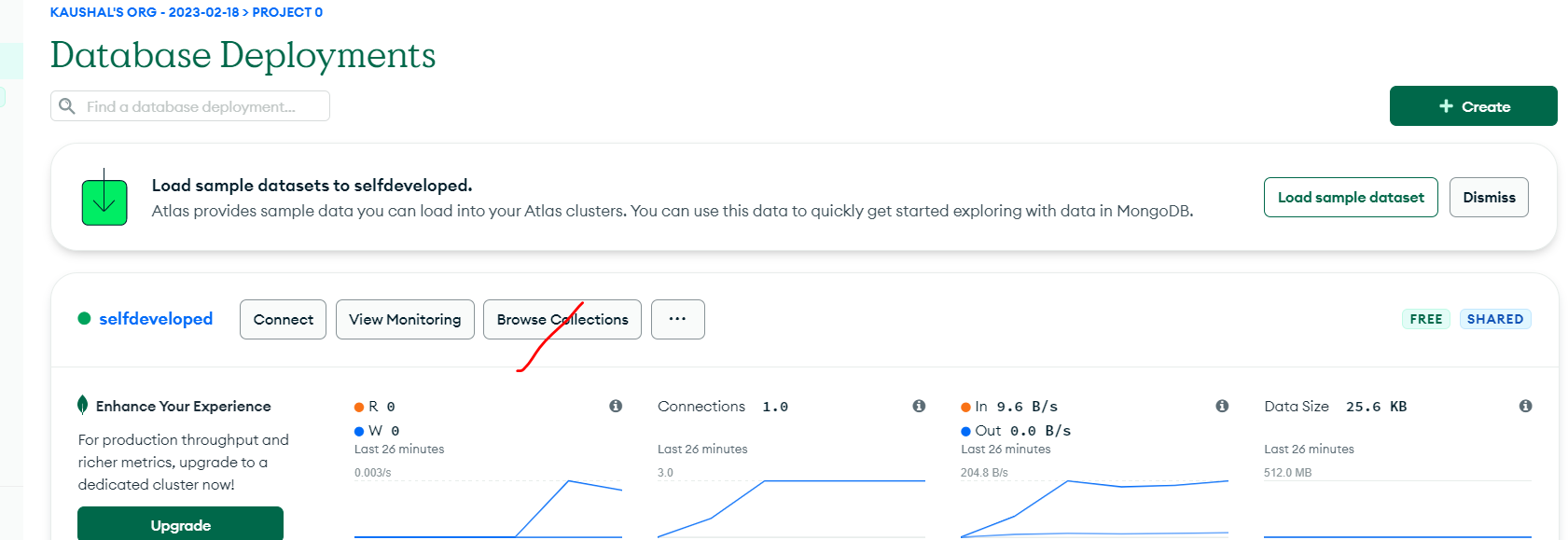


----------------------------------------------------------------------------------------------------------------------------------------

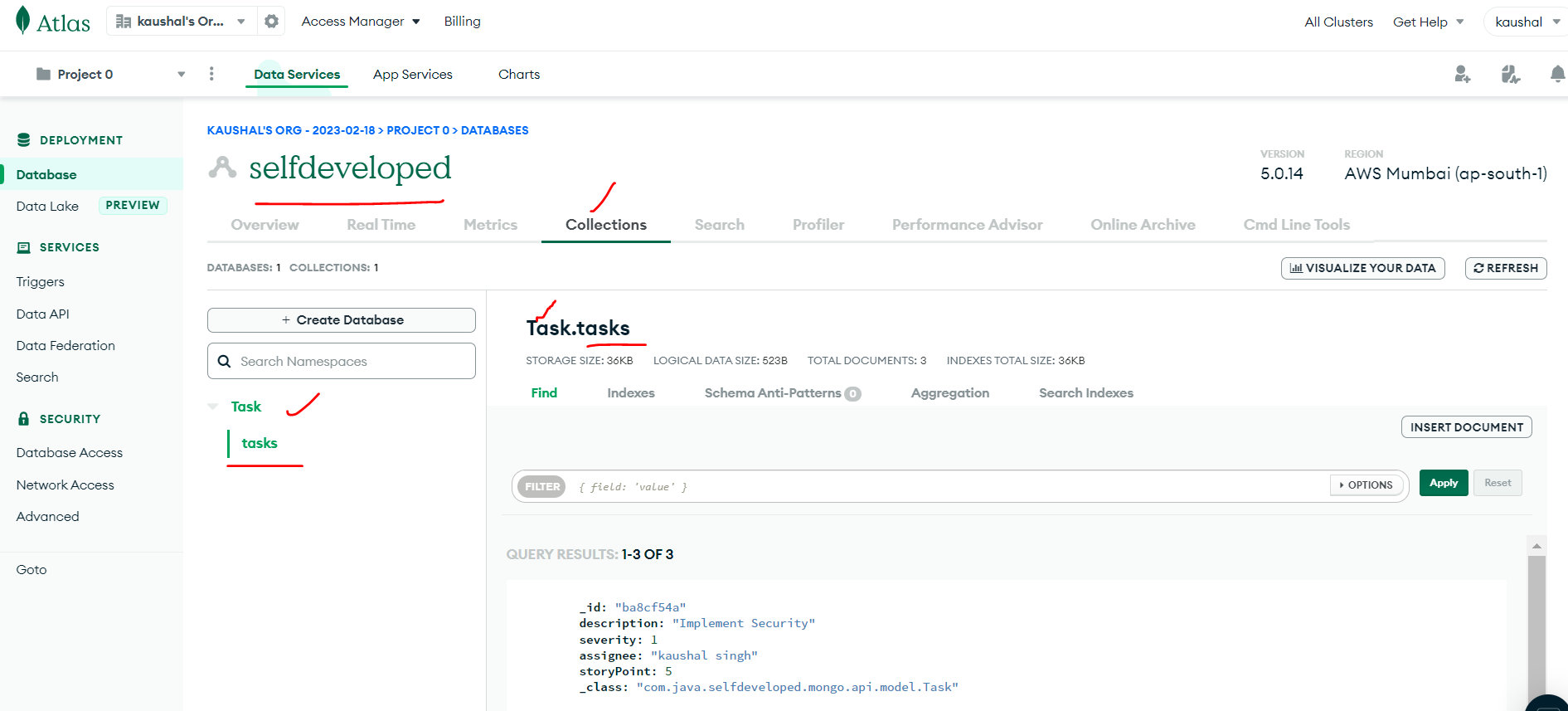


Now if I will go and verify in my database, so there is a option called **Browse Collections**

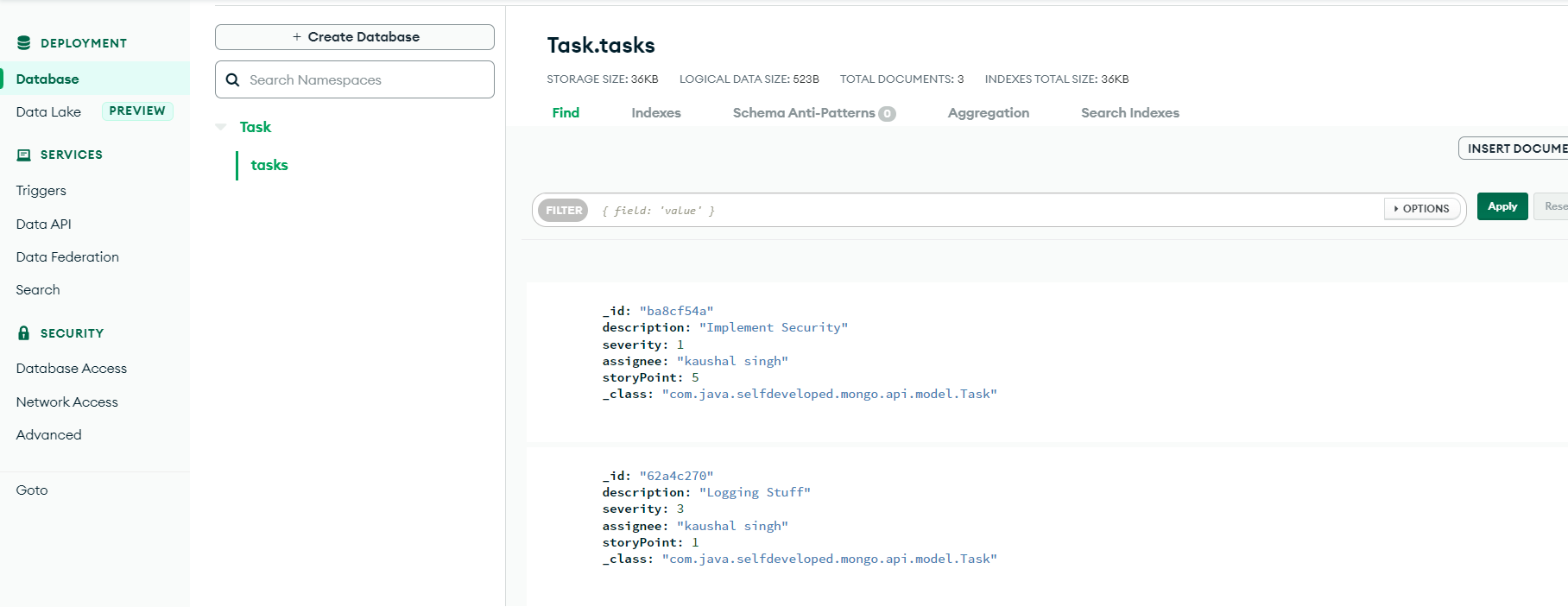
<https://cloud.mongodb.com/v2/63f0bfd4638e1d2da6249fce#/clusters>



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



So, **Task** is my database name and **tasks** is my collection created and we have added the 3 entries, 3 object or 3 **document**.



Now let’s check all the tasks from DB. So, the endpoint is same \**tasks** with Http **GET** method **–**

**GET** [**http://localhost:8080/tasks**](http://localhost:8080/tasks)

[

    {

        "taskId": "ba8cf54a",

        "description": "Implement Security",

        "severity": 1,

        "assignee": "kaushal singh",

        "storyPoint": 5

    },

    {

        "taskId": "62a4c270",

        "description": "Logging Stuff",

        "severity": 3,

        "assignee": "kaushal singh",

        "storyPoint": 1

    },

    {

        "taskId": "de913a32",

        "description": "Add cron expression to refresh data",

        "severity": 4,

        "assignee": "Raghu",

        "storyPoint": 2

    }

]

**Next** is get the Task by using the **taskId**

**GET** <http://localhost:8080/tasks/ba8cf54a>

{

    "taskId": "ba8cf54a",

    "description": "Implement Security",

    "severity": 1,

    "assignee": "kaushal singh",

    "storyPoint": 5

}

**Now** next is get the Task by using the **severity**

**GET** <http://localhost:8080/tasks/severity/3>

[

    {

        "taskId": "62a4c270",

        "description": "Logging Stuff",

        "severity": 3,

        "assignee": "kaushal singh",

        "storyPoint": 1

    }

]

**Now** next is get the List of Tasks by using the **Assignee**

**GET** http://localhost:8080/tasks/assignee/kaushal singh

[

    {

        "taskId": "ba8cf54a",

        "description": "Implement Security",

        "severity": 1,

        "assignee": "kaushal singh",

        "storyPoint": 5

    },

    {

        "taskId": "62a4c270",

        "description": "Logging Stuff",

        "severity": 3,

        "assignee": "kaushal singh",

        "storyPoint": 1

    }

]

Now next is get the List of Tasks by using the severity and story point

**GET** [**http://localhost:8080/tasks/severity\_storypoint/1/5**](http://localhost:8080/tasks/severity_storypoint/1/5)

[

    {

        "taskId": "ba8cf54a",

        "description": "Implement Security",

        "severity": 1,

        "assignee": "kaushal singh",

        "storyPoint": 5

    }

]

**PUT** <http://localhost:8080/tasks>

Make sure you need to give the valid Id and that Id should be non-editable bcz we are updating data based on our taskId.

**Request**:        {

        "taskId": "ba8cf54a",

        "description": "Adding Kafka Configuration",

        "severity": 9,

        "assignee": "Pavni",

        "storyPoint": 10

    }

**Response**:

{

    "taskId": "ba8cf54a",

    "description": "Adding Kafka Configuration",

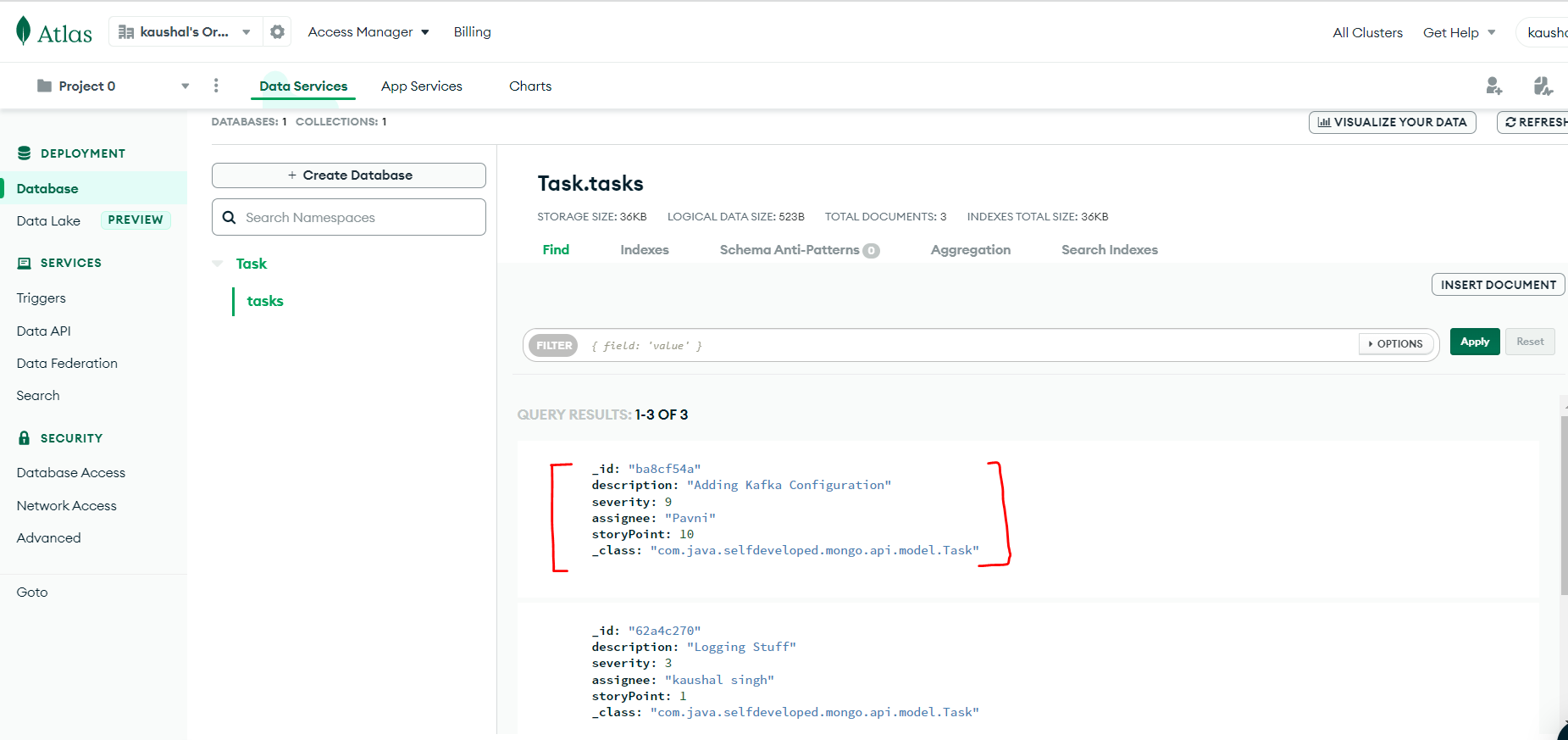
    "severity": 9,

    "assignee": "Pavni",

    "storyPoint": 10

}

**We can see our collection got updated…**

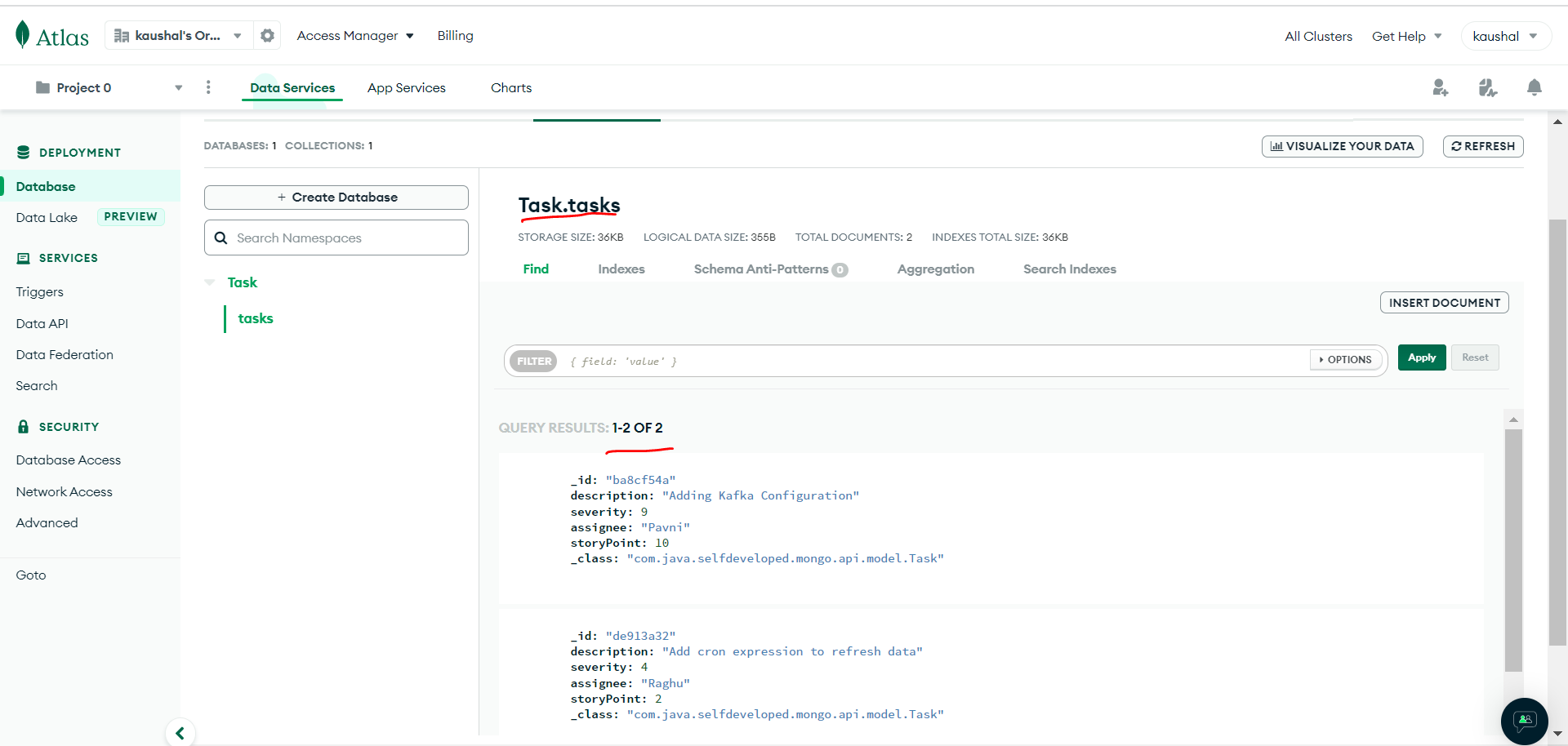


Now next is we are going to delete a task by Id – **62a4270**

**DELETE** [**http://localhost:8080/tasks/62a4c270**](http://localhost:8080/tasks/62a4c270)

Response-

62a4c270 task deleted from dashboard



Now we can see one task got deleted.

So, MongoDB Atlas we have used for MongoDB GUI