

BookComet Web Application

Project 2: Building and Deploying a Containerized Python Application with Flask and MongoDB on Azure Kubernetes Service (AKS)

Student Name: Atena Jafari Parsa

Course Name: Database Systems and Cloud Computing

Course Code: AIN3003 (1)



Logo retrieved from: [BrandCrowd](#)

- Table of Contents

- Introduction
- Abstract
- Azure Cosmos DB Set-up & Data Retrieval
- Connecting Visual Studio Code to Azure Cosmos DB
- Writing the Python Script & Necessary Files for Creating the Web-app
- The CRUD Operations
- Building the Docker Image and Pushing It to Azure Container Registry (ACR)
- Setting Up Azure Kubernetes Service

- Introduction

Books are one of the oldest means of sharing knowledge in human history. From ancient scrolls to kindles, there has always been a similar pattern: words and sentences, symbols, numbers, visualizations, etc. This shows the importance of finding an effective way to represent, share and store these blocks of knowledge, called books.

Considering the advances of Information Technology (IT), cloud-based online applications are a modern and easy way for the users to access books to buy/borrow them for reading.

- Abstract

- The aim of the project is to build and deploy a Containerized Python Application with Flask and MongoDB on Azure Kubernetes Service (AKS).
- For this purpose, first the data should be retrieved and stored in a proper format (through Azure CosmosDB Cloud shell (in Javascript)) which has been explained below in *Data Retrieval*.
- After the extension of MongoDB for Visual Studio Code was installed, and through the connection string, VSCode was connected to the Azure CosmosDB and the database was accessed and tested by some sample queries.
- Then a python script called app.py was written for the app routes to perform the CRUD operations. The necessary files including some html templates were also added.
- The image containing the python app was built. After logging into the azure account and the ACR, the image was tagged and pushed to the ACR.
- After that, a Kubernetes service and cluster is created on Azure. A single-image web app is created as the corresponding ACR and the pushed image is selected. The .yaml file is downloaded, and the app is kept.
- Under the Kubernetes resources in the cluster, services and ingresses is selected. After clicking on the app name, the External IP can be observed.

- Azure Cosmos DB Set-up & Data Retrieval

After creating an Azure CosmosDB resource on [Azure portal](#), the *Bookstore* database is created. It uses an Azure server in North Europe (Ireland) region.

The screenshot shows the Azure portal interface for the 'bookcomet' CosmosDB account. The left sidebar contains navigation links like Home, Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quick start, Notifications, Data Explorer, and Settings. The main content area is titled 'Overview' and includes sections for 'Essentials' (Status: Online, Resource group: AtenaJafariParsa_group, Subscription: Azure for Students, Backup policy: Periodic, Read Locations: North Europe, Write Locations: North Europe, URI: https://bookcomet.mongo.cosmos.azure.com:443/, Server Version: 4.2, Capacity mode: Serverless) and 'Collections' (a single collection named 'books' under the 'bookstore' database). Below these are 'Monitoring' charts for 'Number of requests per 5 minutes' and 'Request Charge' over a 24-hour period, with data points visible on the graphs.

The documents are then stored to *books* collection through the cloud shell provided by Azure Cosmos DB.

The input data initially contained 9 books in the IT category (each document containing the isbn, the book title, the publish year, the price, number of the pages, the category, the link to the cover image of the book, the publisher info. (name and location) with 9 distinct authors in total and 6 distinct publishers and their location. Later, more data was retrieved from <https://openlibrary.org/>. For the distinction of the added and initial data, the books added later are from categories different from 'IT', e.g. Fantasy, Sci Fi, etc.

For the ease of the process, first the books, then the corresponding publishers and authors documents are inserted to the collections through Atlas Mongo Shell in this example form:

After all the documents are inserted we check them in the CosmosDB interface or type some queries to confirm the correct insertion. What I used for books collection to look at the documents is: db.{books}.find().pretty() after switching to *bookstore* database.

- Connecting Visual Studio Code to Azure Cosmos DB

After having installed the MongoDB extension, the Bookstore database is connected through the connection string (cosmos uri) that can be found as shown below:

The beginning lines in app.py, import the libraries (e.g. flask, jsonify) and connect Visual Studio Code to the database in Azure Cosmos DB:

```
from flask import Flask, render_template, request, redirect, url_for, jsonify
from pymongo import MongoClient
app = Flask(__name__)
# Replace these values with your Cosmos DB connection string and database/collection
details
cosmos_uri = "*****"
client = MongoClient(cosmos_uri)
db = client["bookstore"]
books_collection = db["books"]
```

- Writing the Python Script & Necessary Files for Creating the Web-app i.e. .dockerignore, app.py, requirements.txt, gunicorn.conf.py, html templates

- app.py:

Includes the app routes, the python script for performing CRUD operations on the database. When the app.py is running, the local website can be accessed. The web app can be accessed locally on <http://127.0.0.1:5001/>.

- The html Templates

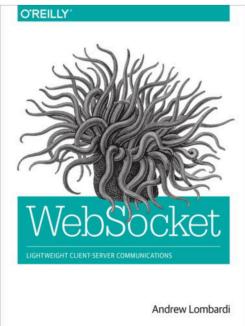
The .html files were added for providing a simpler and more clear interface for the user.

- The main html template is `index.html` which is responsible for the first page that appears on the website and shows all books. The corresponding app route and function in app.py for it is

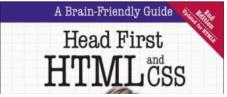
- `@app.route('/')`
- `def index() #which fetches all books and displays them`

All Books

• [WebSocket](#)
 ISBN: 978-1449369279
 Year: 2015
 Price: \$25.99
 Page Count: 144
 Category: IT



Cover Photo:
 Author: Andrew Lombardi
 Publisher: O'Reilly Media (USA)
[Update](#) [Delete](#)
 • [Head First HTML And CSS](#)
 ISBN: 0596159900
 Year: 2012
 Price: \$26.78
 Page Count: 768
 Category: IT



● The CRUD Operations

Each app route in app.py redirects to one of the html templates:

Create

- `@app.route('/add_book', methods=['GET', 'POST'])`
- `def add_book()`
- `add_book.html`

Let's add a new book to the database, we'll get the field values from

<https://openlibrary.org/works/OL16325201W/Insurgent>.

1. Scroll down to the end of the book displays and click on Add a new book



Cover Photo:

Author: Ana Huang

Publisher: Little, Brown Book Group Limited (London, UK)

[Update](#)

[Delete](#)

• [Nutrition in clinical dentistry](#)

ISBN: 0721624235

Year: 1989

Price: \$8.53

Page Count: 465

Category: Science



Cover Photo:

Author: Abraham J. Nizel

Publisher: Saunders (Philadelphia, USA)

[Update](#)

[Delete](#)

[Add a New Book](#)



2. Insert the field values in the corresponding boxes for every field, then press [Add Book](#).

Add a New Book

ISBN:

Title:

Year:

Price:

Page:

Category:

Cover Photo URL:

Publisher Location:

Publisher Name:

Author Name:

[Add Book](#)

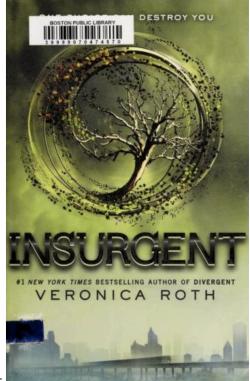


[Back to All Books](#)

3. The book has been added to the end of the books and is now displayed on the main page.

• [Insurgent](#)

ISBN: 9780062024046
 Year: 2012
 Price: \$5.07
 Page Count: 525
 Category: Sci Fi

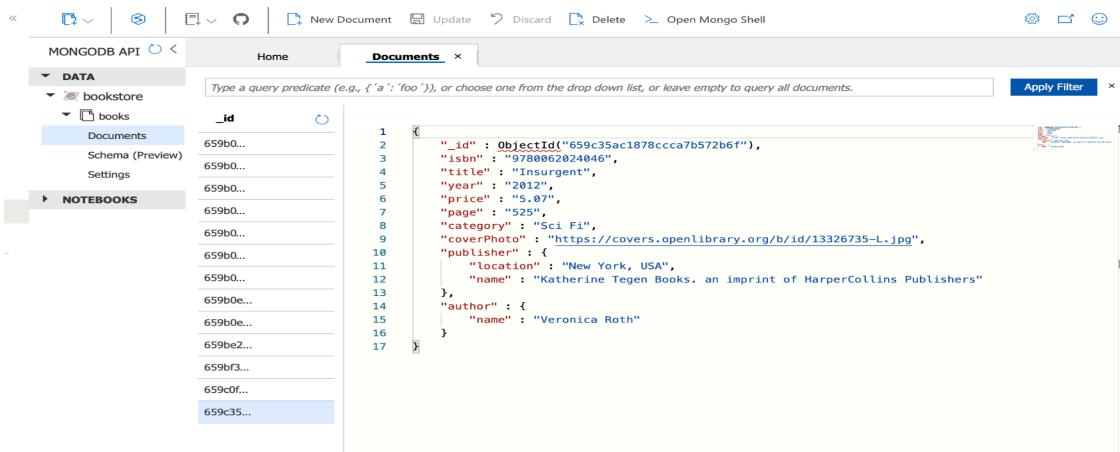


Cover Photo:
 Author: Veronica Roth
 Publisher: Katherine Tegen Books, an imprint of HarperCollins Publishers (New York, USA)

[Update](#) [Delete](#)

[Add a New Book](#)

It has also been added as a document to the books collection in bookstore database in Azure Cosmos DB.



The screenshot shows the MongoDB API interface with the 'Documents' tab selected. On the left, the 'bookstore' database is expanded, showing the 'books' collection. A specific document is selected, showing its details in the main pane. The document structure is as follows:

```

1  {
2      "_id" : ObjectId("659c35ac1878ccca7b572b6f"),
3      "isbn" : "9780062024046",
4      "title" : "Insurgent",
5      "year" : "2012",
6      "price" : "5.07",
7      "page" : "525",
8      "category" : "Sci Fi",
9      "coverPhoto" : "https://covers.openlibrary.org/b/id/13326735-L.jpg",
10     "publisher" : {
11         "location" : "New York, USA",
12         "name" : "Katherine Tegen Books, an imprint of HarperCollins Publishers"
13     },
14     "author" : {
15         "name" : "Veronica Roth"
16     }
17 }

```

Read

- `@app.route('/book/<isbn>')`
- `def book_details(isbn):`
- `# Fetches and displays details of a specific book`
- `book_details.html`

1. For the Read operation, simply click on the title of the book in the main page.

All Books

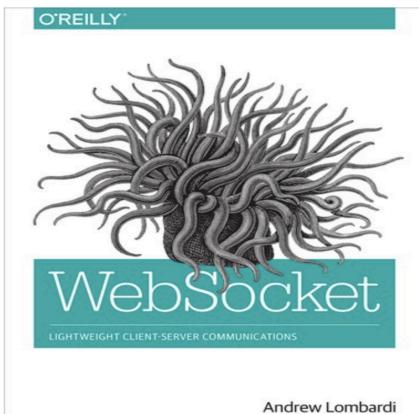
- [WebSocket](#)
- ISBN: 978-1449369279
Year: 2015
Price: \$23.99
Page Count: 144
Category: IT

O'REILLY

2. View all the available information and details on the book, including the cover image:

Book Details

Title: WebSocket
ISBN: 978-1449369279
Publish_Year: 2015
Price: 23.99
Number_of_Pages: 144
Publisher_name: O'Reilly Media
Publish_location: USA
Author: Andrew Lombardi



Update

- `@app.route('/update_book/<isbn>', methods=['GET', 'POST'])`
- `def update_book(isbn)`
- `update_book.html`

1. Scroll down to the book you would like to update, then press Update under the *Publisher:*

- **Head First HTML And CSS**

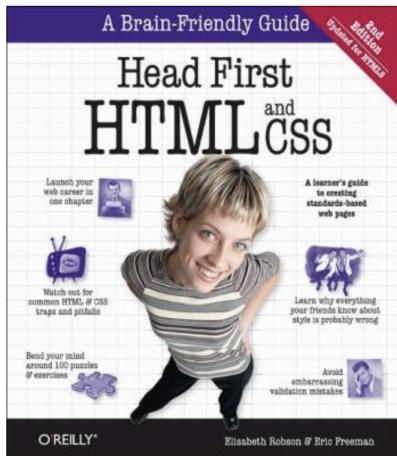
ISBN: 0596159900

Year: 2012

Price: \$26.78

Page Count: 768

Category: IT



Cover Photo:

Author: Elisabeth Robson

Publisher: O'Reilly Media (USA)

[Update](#)

[Delete](#)

2. Update the value, and press Update again. The updates would be applied to the website and the database in Azure Cosmos DB simultaneously.

Please note that after a book is published in real life, fields like ISBN, book title, publish year and author's name cannot be changed. However, values like price, can keep getting updated as time goes on. This is why the Price field was used for the update operation.

Update Book Price

Invalid input. Please enter a valid numeric value for Price.

Price: 26.78

[Update Price](#)

[Back to Book Details](#)

Delete

- @app.route('/delete_book/<isbn>', methods=['GET', 'POST'])
- def delete_book(isbn)
- delete_book.html

1. Proceed to the book that has to be deleted and press the Delete button next to Update. After the Delete button is pressed, the book would be removed immediately both from the website and from the database.

- [Nutrition in clinical dentistry](#)

ISBN: 0721624235

Year: 1989

Price: \$8.53

Page Count: 465

Category: Science



Cover Photo:

Author: Abraham J. Nizel

Publisher: Saunders (Philadelphia, USA)

[Update](#)

[Delete](#) 

- Building the Docker Image and Pushing It to Azure Container Registry (ACR)

After confirming that the web app can be accessed locally on <http://127.0.0.1:5001/> and works as desired, the next step is to prepare it for deployment on Azure Kubernetes Service (AKS). In order to do that, the python application should be containerized in a Docker image and pushed to ACR.

After deciding the image name, this command should be run in the terminal in the same directory as the project for the Docker image to be built:

[docker build -t mybookstore](#) .

After logging in to the azure account and the ACR (by using the Access Keys), and tagging ([docker tag mybookstore myregistry2024p.azurecr.io/mybookstore](#)) the Docker image should be pushed to the container: [docker push myregistry2024p.azurecr.io/mybookstore](#)

- Setting Up Azure Kubernetes Service

After logging in to [Azure portal](#), a Kubernetes Service resource named BookComet_Cluster is created. Then, a single image web app is created and the container (MyRegistry2024P) and the image is selected. An automatic .yaml file (YAML_5.text) is then created for the deployment. Keep is selected for the app.

The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with 'Microsoft Azure' and a search bar. Below it, the breadcrumb navigation shows 'Home > BookComet_Cluster > Create a starter application >'. The main title is 'Create a single-image application ...'. A section titled '2. Deploying resources' contains a note: 'Select a deployment name to see more details, including the individual pods that were created as part of the deployment.' Below this is a table showing resource details:

Resource	Type	Status
default-1704851165812	Namespace	Success
bookcomet	Deployment	Success
bookstore-service	Service	Success
bookcomet-pods	Pod	1/1 pods ready

A section titled '3. Next steps' follows, with a note: 'Here are some actions you can take once your application is deployed.' It lists several options:

- [View the application](#): View the deployed application by going to the external IP address associated with the frontend service. (View application)
- [Learn how to create and deploy custom container images](#): Set up a GitOps pipeline to automatically deploy your applications from source control to your cluster. (Configure GitOps)
- [Give feedback](#): Help improve this page

At the bottom are 'Previous' and 'Close' buttons.

- How to access the deployed app

BookComet_Cluster is selected, then, under kubernetes resources, the option services and ingresses is selected. The service (bookstore-service) is selected and eventually, the External IP. This is how the deployed app could be accessed.

<http://bookcomet.northeurope.cloudapp.azure.com/>