

News classificaTIon Design document

Capstone project



October 9, 2021

Group - 33

Srinivasa Chittaluru + Samara Simha Reddy

SWDD- Software Design Document

(Group 33 – KEY – Keep Educating Yourself - Team)

**(NEWS articles classification)**

#### Software Design Document

Name (s): Srinivasa Chittaluru

Samara Simha Reddy

Section: IITH – GL – Group 33

Workstation: Local

Date: October 9th 2021

**TABLE OF CONTENTS**

1.0 INTRODUCTION 3

1.1 Purpose 3

1.2 Objectives 3

1.3 Overview 3

1.4 Reference Material 4

1.5 Definitions and Acronyms 4

2.0 SYSTEM OVERVIEW 4

3.0 SYSTEM ARCHITECTURE 5

3.1 Architectural Design 5

3.2 Decomposition Description 6

3.3 Design Rationale 6

4.0 DATA DESIGN 7

4.1 Data Description 7

4.2 Data Dictionary 7

5.0 COMPONENT DESIGN 7

6.0 HUMAN INTERFACE DESIGN (TBD) 8

6.1 Overview of User Interface 8

6.2 Screen Images 8

6.3 Screen Objects and Actions 8

7.0 WEEKLY PROGRESS REPORT 8

Code repository is at: https://github.com/cavbtech/CapstoneProject2.git 8

8.0 MILESTONE REPORT (MSR) 9

9.0 SCREEN SHOTS 10

## INTRODUCTION

## Purpose

Currently, there are various NEWS sites provides various feeds. However, it is difficult to categorize the new feeds for any user and focus on those feeds which User is really interested in. This project is an approach to classify any statement or any news feeds or any news so that User can decide whether to read that or not. However, this is not full-blown project. This is a sample approach on how to classify or simulate the actual solution. The main objective of this project is to use various technologies and integrate them and predict any statement by applying machine learning models to classify

This document, shows how various technologies implemented and integrated to classify NEWS

## Objectives

* Design complete solution to demonstrate end-to-end pipeline development and manage a machine/ deep learning project
* Develop a understanding of all stages of a machine learning project lifecycle
* Demonstrate understanding of challenges encountered during the project development and provide ways to tackle them
* Showcase understanding of software engineering best practices while developing the project.

## Overview

This project is to classify News Articles into categories - With information overload today users are inundated with news articles of all topics, even the ones which may not be relevant to users. This system classifies incoming news articles and appropriately tag the corresponding category. It follows the following data pipeline which includes all the following stages of Machine Learning Project Life Cycle

* Data Ingestion
* Data Preparation
* Data Segregation and Model Training
* Model Deployment
* Model Prediction

## Reference Material

Please refer

* <https://olympus.greatlearning.in/courses/62111/files/3862120?module_item_id=1780005>
* Docker setup related <https://hub.docker.com/search?q=bde2020&type=image>

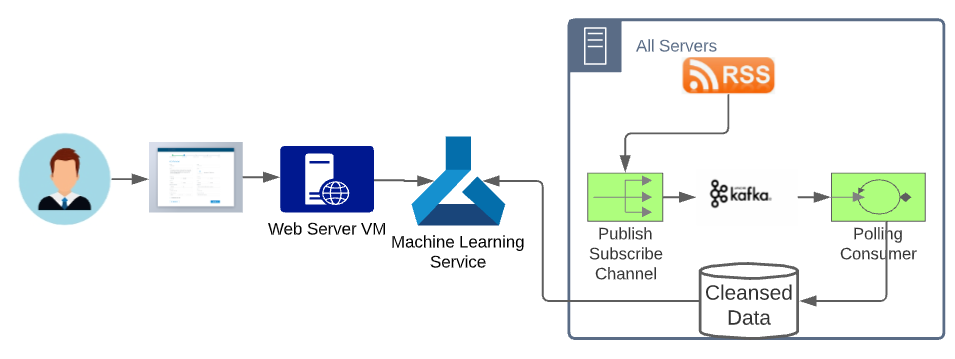
## Definitions and Acronyms

*NA*

## SYSTEM OVERVIEW

This section defines overall system view and dataflow. The major behavior of this system is, User can enter any text in the user interface for which the system should predict and classify the text.

The following diagram shows list of components in the System



The following dockers composed as part of the system. To get more information on each docker, refer 3.2 section

* Producer Docker:
* Kafka Broker Docker:
* Zookeeper Docker
* Consumer Docker:
* Machine learning Service:
* Web server docker

Technologies used

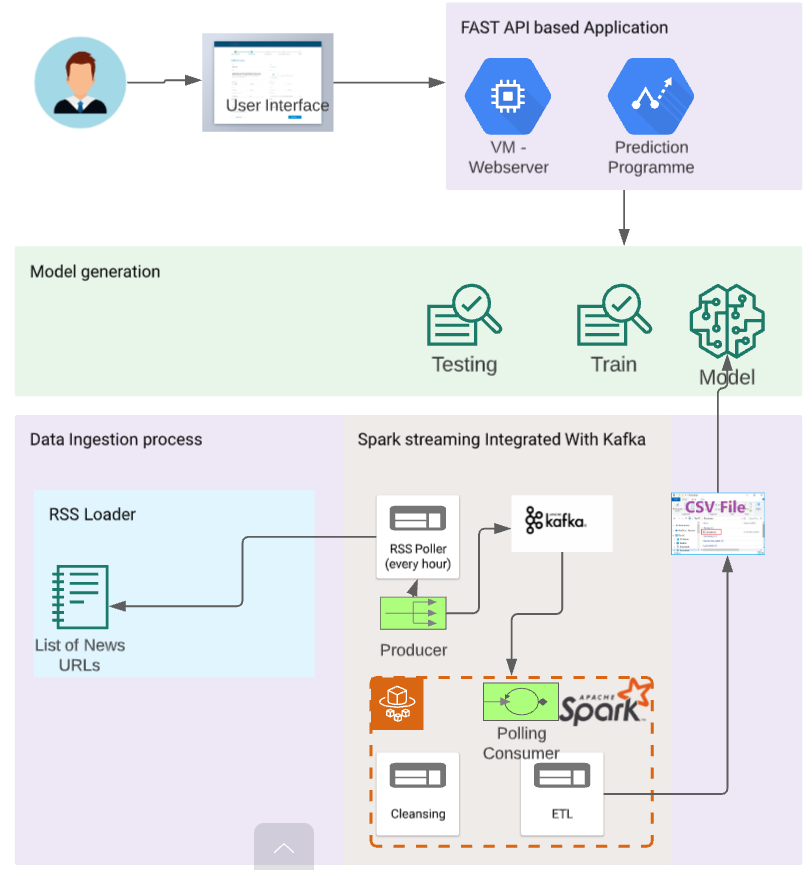
* Python
* Spark 3.1
* Spark streaming
* Python based Machine learning APIs

## SYSTEM ARCHITECTURE

## Architectural Design

This system is built based micro service architecture. There are various services developed and integrated based on the docker-composer architecture. This architecture is scalable and easily deployable in cloud by lift and shift paradigm.

The following architecture diagram gives an understanding of the system



## Decomposition Description

As described in the section-3 summary, there are various sub-system which run independently in micro service architecture.

* Producer Docker Sub-system: This docker reads RSS feeds with configured frequency and produces these feeds to Kafka Broker
* Kafka Broker Docker Sub-system: This docker is to collect producer messages and make it available to consumer to consume
* Zookeeper Docker Sub-system: Zoo keeper is need to track the cluster of Kafka brokers. However, as this system is not a full-blown system, it only maintains one broker
* Consumer Docker Sub-system: This docker consumes messages sent by producer and cleanses the data and stores them into a text file. Initially planned for DB to store these datasets but then the system is not support with too many dockers running at the same time. Hence, reduced DB docker by storing the data into text file
* Machine learning Service Sub-system: This is a micro service takes the cleansed data and prepares the model to train, test and predict.
* Web server docker Sub-system: This micro service plays a controller role where it takes the inputs from the User and sends the request to the machine learning service docker and obtains the predicted value and displays back to the User

## Design Rationale

Here are the major reasons to select this architecture and design to build this system

* This system Portable and deployable in any environment without any deployment challenges
* This system is Scalable. Can open any number of dockers for each sub system which would not only takes care of scalability but also availability of the system

## DATA DESIGN

## Data Description

This system takes RSS feeds from various news sites. Generally, these feeds come in specific categories. In General, RSS feeds follow the same JSON structure and hence there are various utilities to directly parse the RSS feeds. The following fields are obtained from RSS Feeds using the Feed parser python utility

* Title
* Date / Time
* Summary
* Source
* Category / Topic

These fields are then sent by producer docker to the Kafka broker based on the configure frequency. The consumer docker continually receives the same fields and then appends to a csv file

## Data Dictionary

This application uses only five fields as specified in the 4.1 section.

## COMPONENT DESIGN

In this section, we take a closer look at what each component does in a more systematic way. If you gave a functional description in section 3.2, provide a summary of your algorithm for each function listed in 3.2 in procedural description language (PDL) or pseudocode.

* Producer Docker : TBD
* Consumer Docker : TBD
* Machine learning docker : TBD
* Web service docker: TBD

## HUMAN INTERFACE DESIGN (To Be Developed)

## Overview of User Interface

Describe the functionality of the system from the User’s perspective. Explain how the User will be able to use your system to complete all the expected features and the feedback information displayed for the User.

## Screen Images

Display screenshots showing the interface from the User’s perspective. These can be hand­ drawn or you can use an automated drawing tool. Just make them as accurate as possible. (Graph paper works well.)

## Screen Objects and Actions

A discussion of screen objects and actions associated with those objects.

## WEEKLY PROGRESS REPORT

## Code repository is at: https://github.com/cavbtech/CapstoneProject2.git

|  |  |  |  |
| --- | --- | --- | --- |
| Week | Tasks | Status +Completed Date | Submission Date |
| Week-1 | * Architecture * Design * Environment setup * Data ingestion (or) raw data collection | Completed on 6th Oct 2021 | 10th Oct 2021 |
| Week-2 | * Setup the model-training-service project * Loading data into Spark RDD * Cleaning + preprocessing on Raw Data | Completed on 16th Oct 2021 | 18th Oct 2021 |
| Week-3 | * Prepared Training Dataset * Model Training and Deployment * On-Demand Training service * Serializing the retrained model * Pushing the model into model registry with hyperparameters |  |  |
| Week-4 | * Expose Model via Model prediction service * Dockerize all the projects * UI Integration |  |  |

## MILESTONE REPORT (MSR)



### Week-1 MSR

|  |  |
| --- | --- |
| Description | Details |
| Environment Details | * Docker Composer to start the following in net-pet network   + Kafka Broker   + Zoo Keeper   + Spark Master   + Spark Worker * Individual dockers with the same net-pet network. These will be integrated with Docker composer at the end of the project   + Producer Docker   + Composer Docker |
| Input | * RSS feeds   rss\_feed\_urls={"world":"http://rss.cnn.com/rss/edition\_world.rss",  "technology":"http://rss.cnn.com/rss/edition\_technology.rss",  "science&space":"http://rss.cnn.com/rss/edition\_space.rss",  "Entertainment":"http://rss.cnn.com/rss/edition\_entertainment.rss",  "sports":"http://rss.cnn.com/rss/edition\_sport.rss",  "travel":"http://rss.cnn.com/rss/edition\_travel.rss",  "cricket":"https://feeds.feedburner.com/ndtvsports-cricket",  "business":"https://feeds.feedburner.com/ndtvprofit-latest",  "auto":<https://feeds.feedburner.com/carandbike-latest>} |
| Input process algorithm | * Read the RSS feeds using Feed parser * Parse the feed and obtain Title, DateTime, Summary, Souce and Category fields * Keep all the above fields and instantiate a RSSFeed Python class and set the values * Prepare JSON object from the above object for each RSS Feed * Send the JSON object in binary json object to Kafka broker * Once the consumer docker receives the data from Kafka broker, the consumer program cleans the dataset and stores into csv file in a defined volume |
| Output | * Output of this algorithm is a CSV file with the following fields {title, datetime, summary, source, category} |
| Issues + Challenges | * Major challenges that we faced are integrated Dockers with each other, Spark, Scala and Kafka version compatibilities. After checking various options, installed latest versions Spark 3.1.1, Kafka with 2.12-2.3, Zoo-keeper 3.4.6 and Scala 2.12 which are compatible to each other * Infrastructure challenges. Initially planned to install MySQL or Mongo DB to store the Raw data but then because of 8GB RAM, we were not able to run all the dockers and make system scrawl. Hence, we avoided MySQL or Mongo DB |

### Week-2 MSR

|  |  |
| --- | --- |
| Description | Details |
| Environment Details | * Docker Composer to start the following in net-pet network   + Kafka Broker   + Zoo Keeper   + Spark Master   + Spark Worker * Individual dockers with the same net-pet network. These will be integrated with Docker composer at the end of the project   + Producer Docker   + Composer Docker   + Data Modeler (week-2)   + Webapp (week-2) |
| Input | * RSS feeds   rss\_feed\_urls={"world":"http://rss.cnn.com/rss/edition\_world.rss",  "technology":"http://rss.cnn.com/rss/edition\_technology.rss",  "science&space":"http://rss.cnn.com/rss/edition\_space.rss",  "Entertainment":"http://rss.cnn.com/rss/edition\_entertainment.rss",  "sports":"http://rss.cnn.com/rss/edition\_sport.rss",  "travel":"http://rss.cnn.com/rss/edition\_travel.rss",  "cricket":"https://feeds.feedburner.com/ndtvsports-cricket",  "business":"https://feeds.feedburner.com/ndtvprofit-latest",  "auto":<https://feeds.feedburner.com/carandbike-latest>}   * Kaggle dataset (week-2)   Apart from RSS feeds, we have taken a static news category dataset from Kaggle which holds around 150K records. This helps in predicting it better. Realtime data comes from RSS feeds |
| Input process algorithm | * Read the RSS feeds using Feed parser * Parse the feed and obtain Title, DateTime, Summary, Souce and Category fields * Keep all the above fields and instantiate a RSSFeed Python class and set the values * Prepare JSON object from the above object for each RSS Feed. While creating the JSON, it creates an id field which is sha-1 of all the fields. This id field is really handy to remove the duplicates * Send the JSON object in binary json object to Kafka broker * Once the consumer docker receives the data from Kafka broker, the consumer program cleans the dataset and stores into csv file in a defined volume   Week-2   * Once the dataset is saved into CSV file, the model program loads Kaggle static dataset and RSS data feeds csv for every 24 hours and cleanses the data with all incorrect charts, double quotes, single quotes * It then removes the duplicates based on the id field which is sha-1 of all the fields. * Final dataset is stored in ../datavol/cleansed/ * The model program applies MultinomialNB model on the dataset and gets a trained model. This model gets saved as binary file using pickle * The dictionary, which holds all the categories also get stored as a pickle tile * The predictor program, which is in the webapp application, loads theses pickle files every 24 hours and then predicts the category of any given string |
| Output | * Out of this algorithm is a CSV file with the following fields {id, title, datetime, summary, source, category} the final output is stored into ../datavol/cleansed/ * Model dataset in pickle binary form gets stored in (../datavol/trained/work) * Every 24 hours these files moved to (../datavol/trained/active) |
| Issues + Challenges | * Major challenges that we faced are integrated Dockers with each other, Spark, Scala and Kafka version compatibilities. After checking various options, installed latest versions Spark 3.1.1, Kafka with 2.12-2.3, Zoo-keeper 3.4.6 and Scala 2.12 which are compatible to each other * Infrastructure challenges. Initially planned to install MySQL or Mongo DB to store the Raw data but then because of 8GB RAM, we were not able to run all the dockers and make system scrawl. Hence, we avoided MySQL or Mongo DB * Organizing the Kaggle data * Integrating the dockers into docker-compose file. i.,e producer, composer, modeler and webapp modules are to be integrated into docker compose which is throwing few errors and this is yet to be fixed |

### Week-3 MSR

### <TBD>

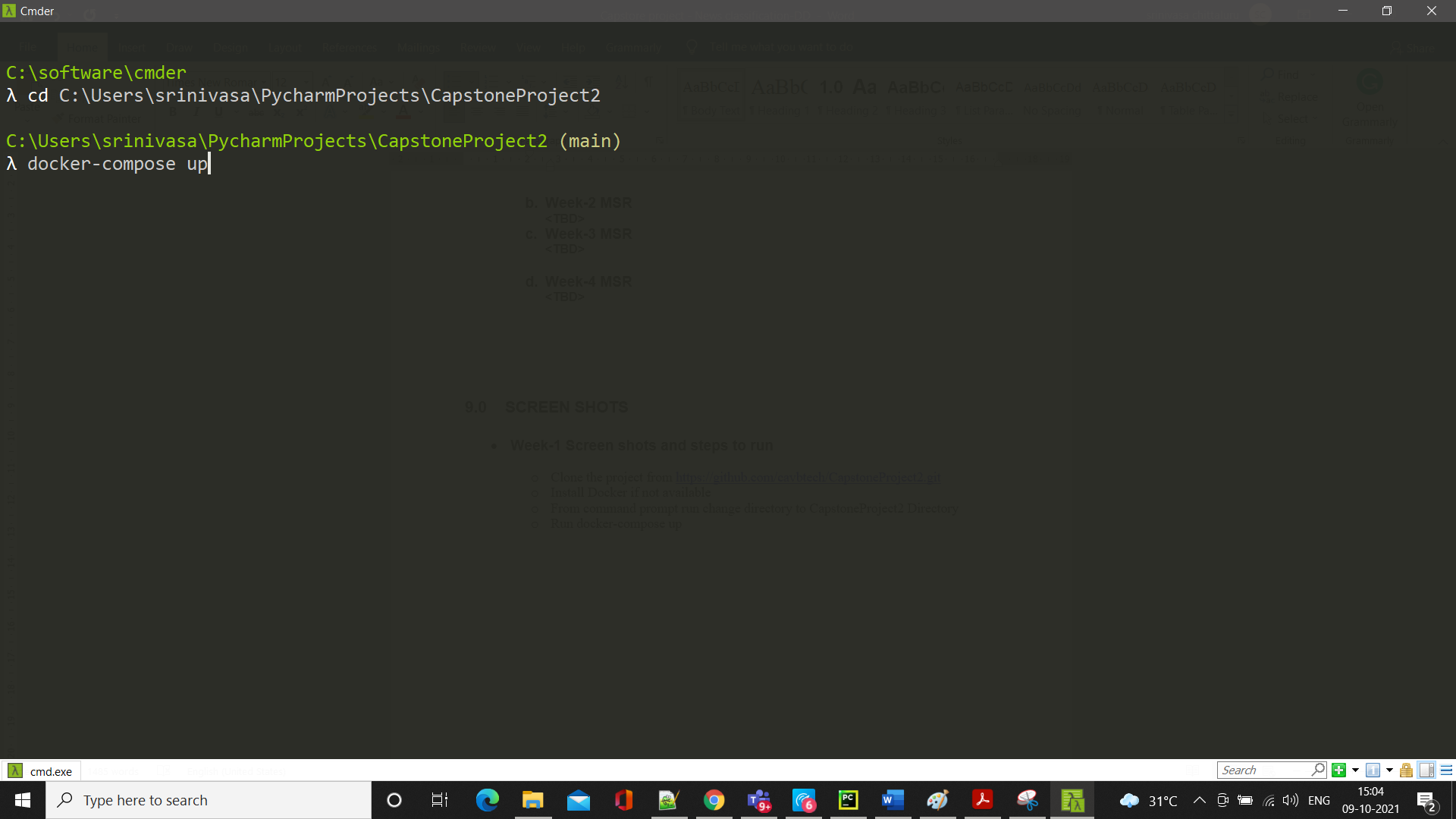
### Week-4 MSR

### <TBD>

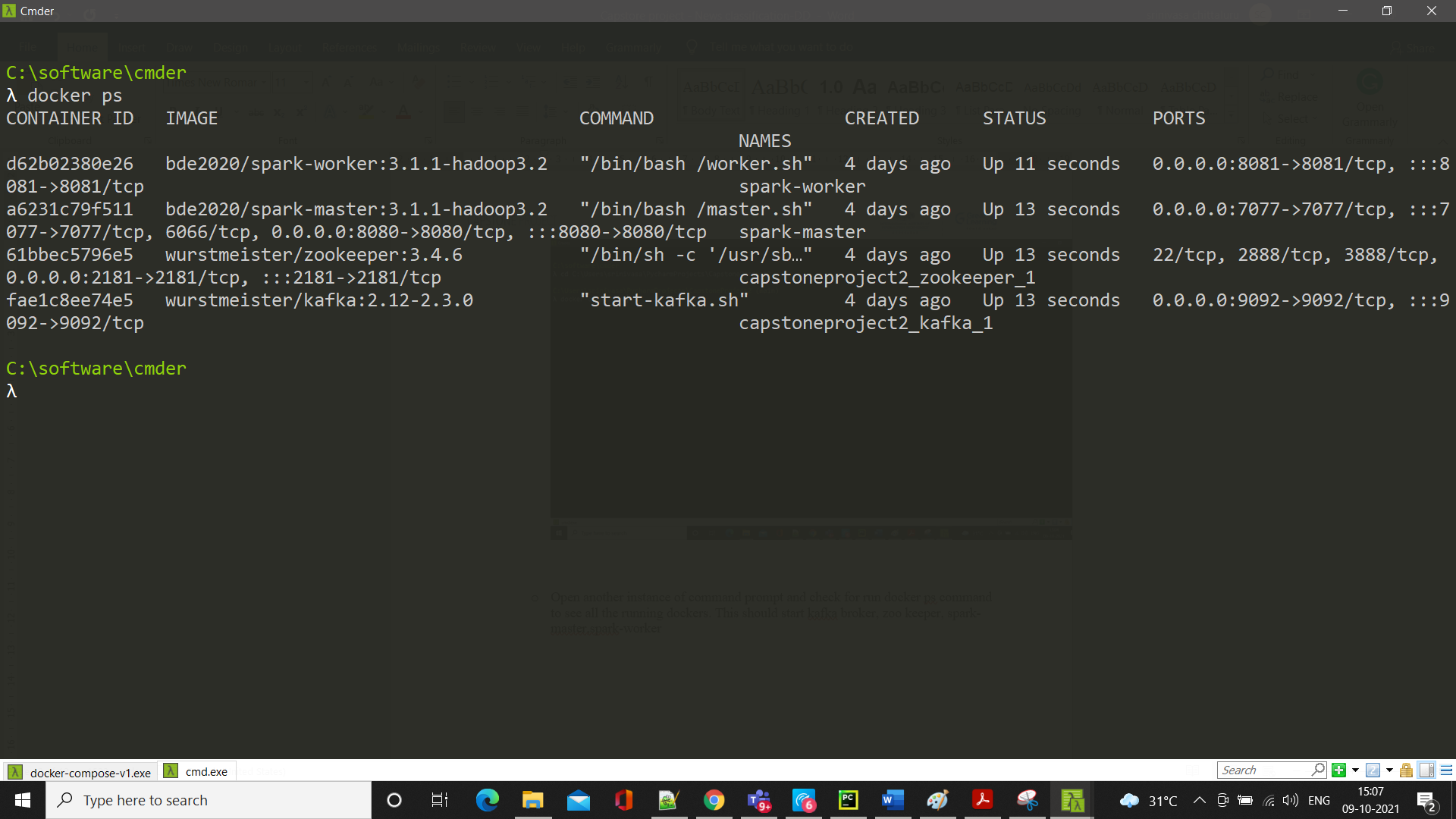
## SCREEN SHOTS

### Week-1 Screen shots and steps to run

* + Clone the project from <https://github.com/cavbtech/CapstoneProject2.git>
  + Install Docker if not available
  + From command prompt run change directory to CapstoneProject2 Directory
  + Run docker-compose up command. This should start kafka broker, zoo keeper, spark-master,spark-worker

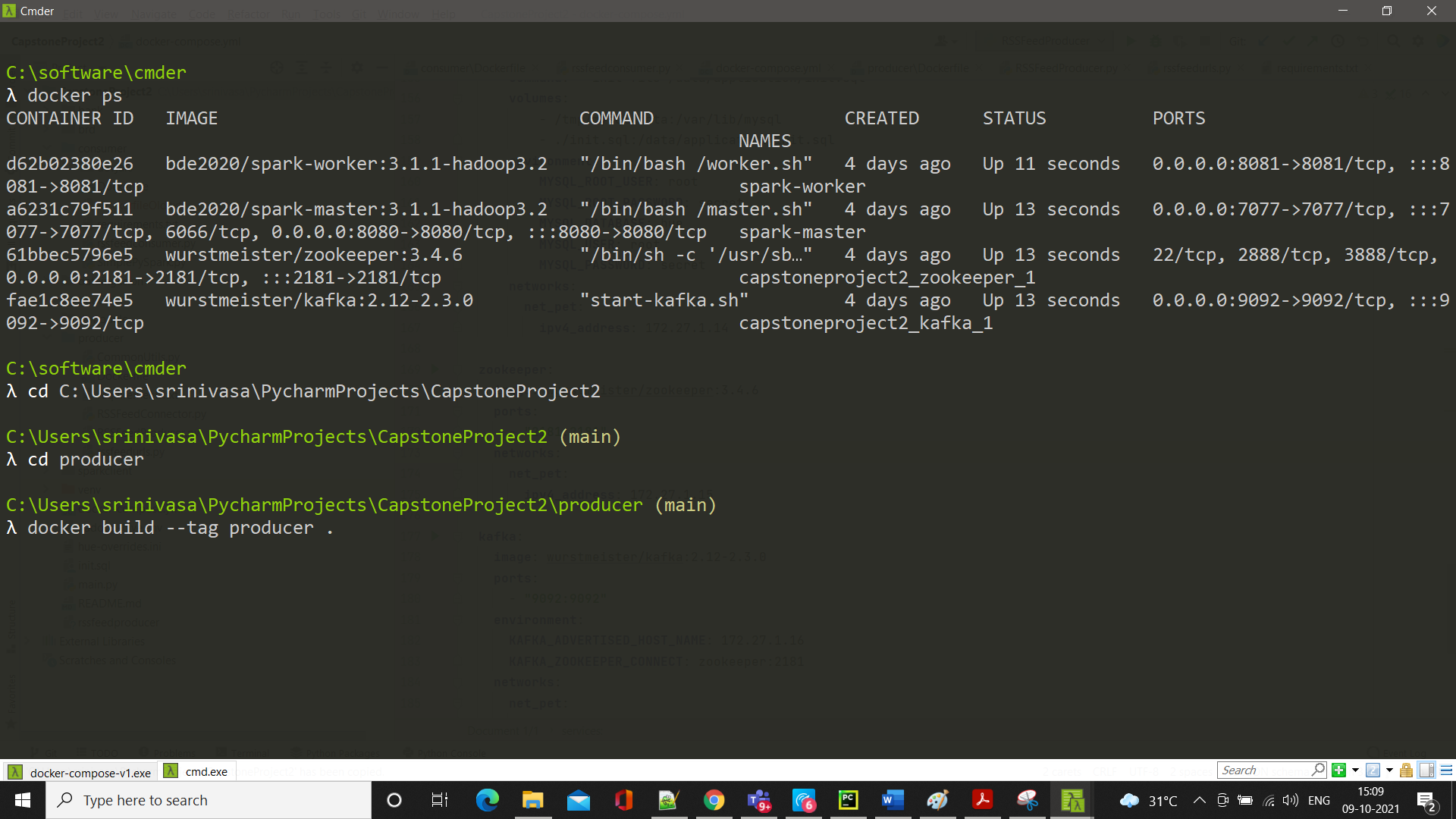


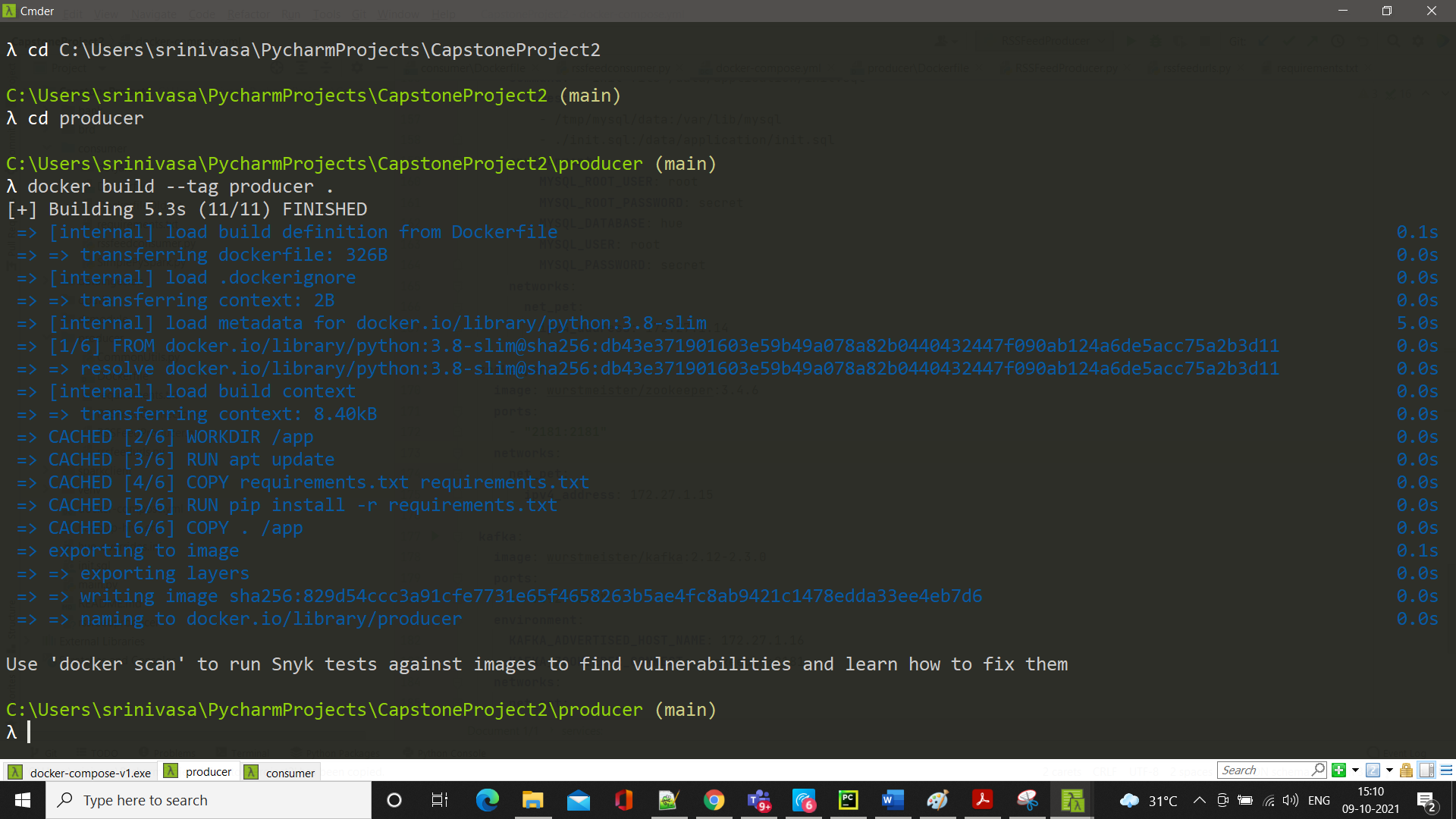
* + Open another instance of command prompt and check for run docker ps command to see all the running dockers. This should start kafka broker, zoo keeper, spark-master,spark-worker



* + Open another instance of command prompt and change directory to ..\ CapstoneProject2\producer and run the following command. This should start the producer image

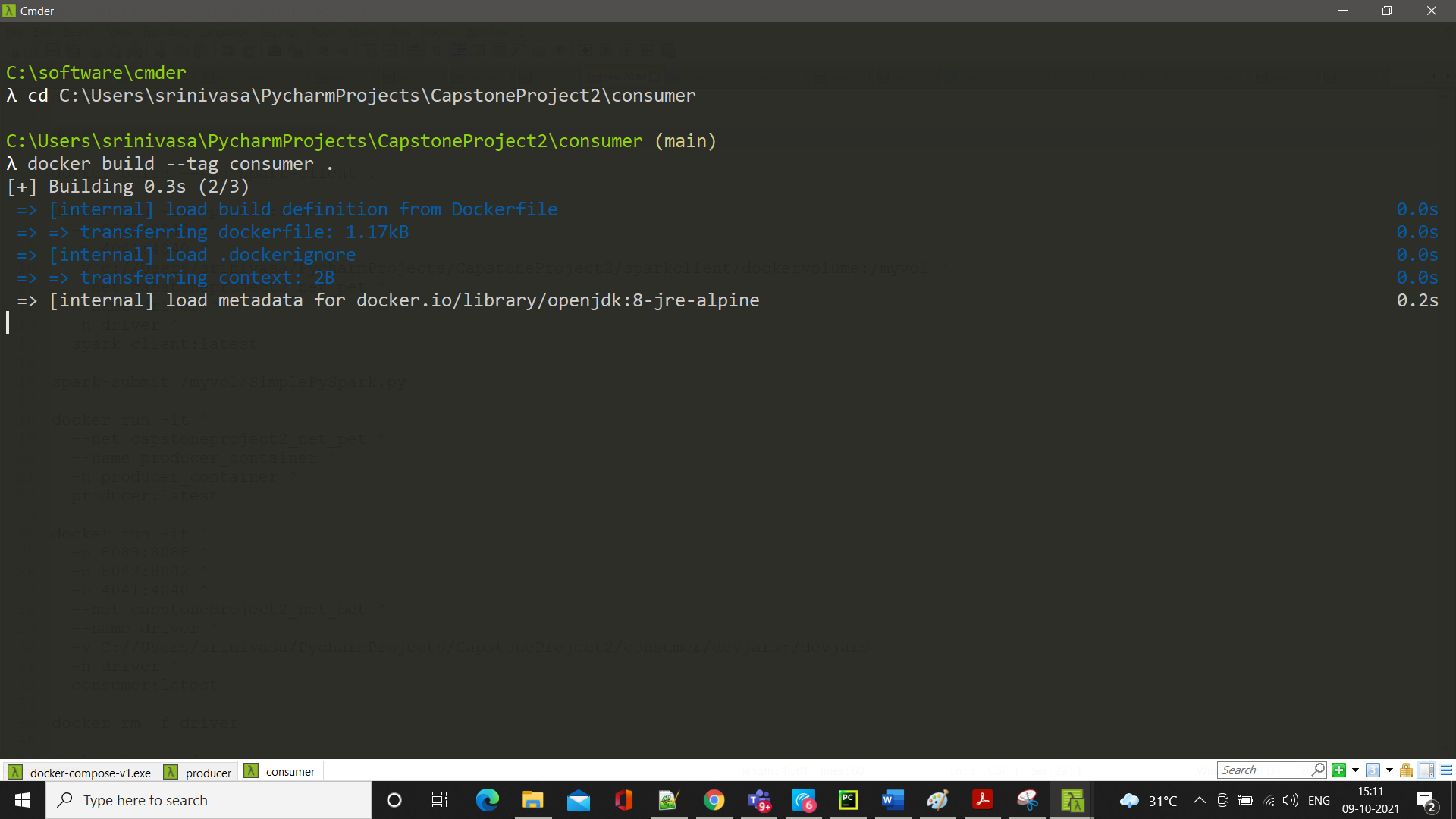
docker build --tag producer .

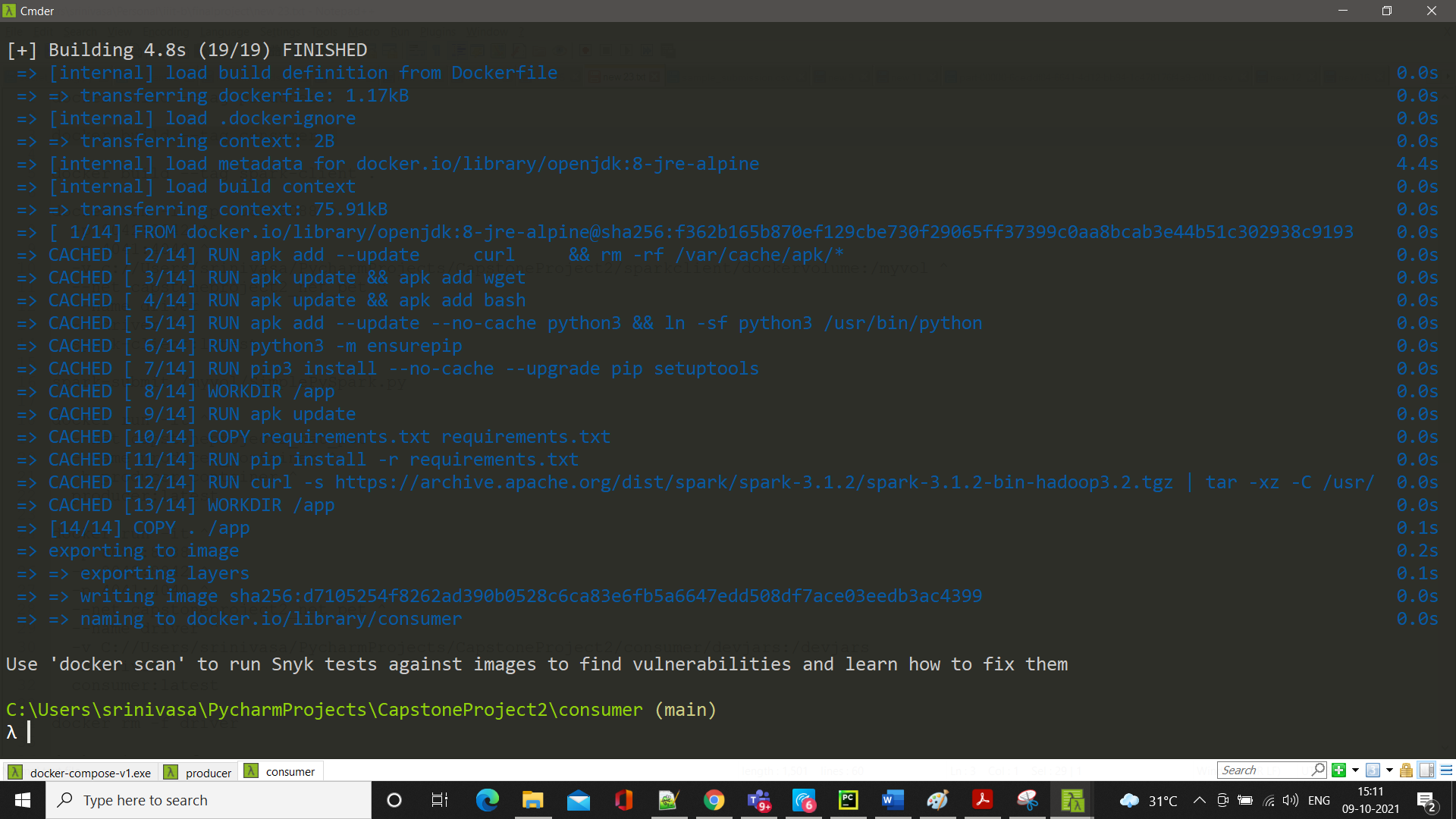




* + Open another instance of command prompt and change directory to ..\ CapstoneProject2\consumer and run the following command. This should start the consumer image

docker build --tag consumer .





* + Go to the terminal where you would have opened the producer and the following command. In case of cmder “^” used for multi-line command. However, it other terminals it could be “\” for multi-line command. Please note the net, it is same the network where kafka, spark are in

docker run -it ^

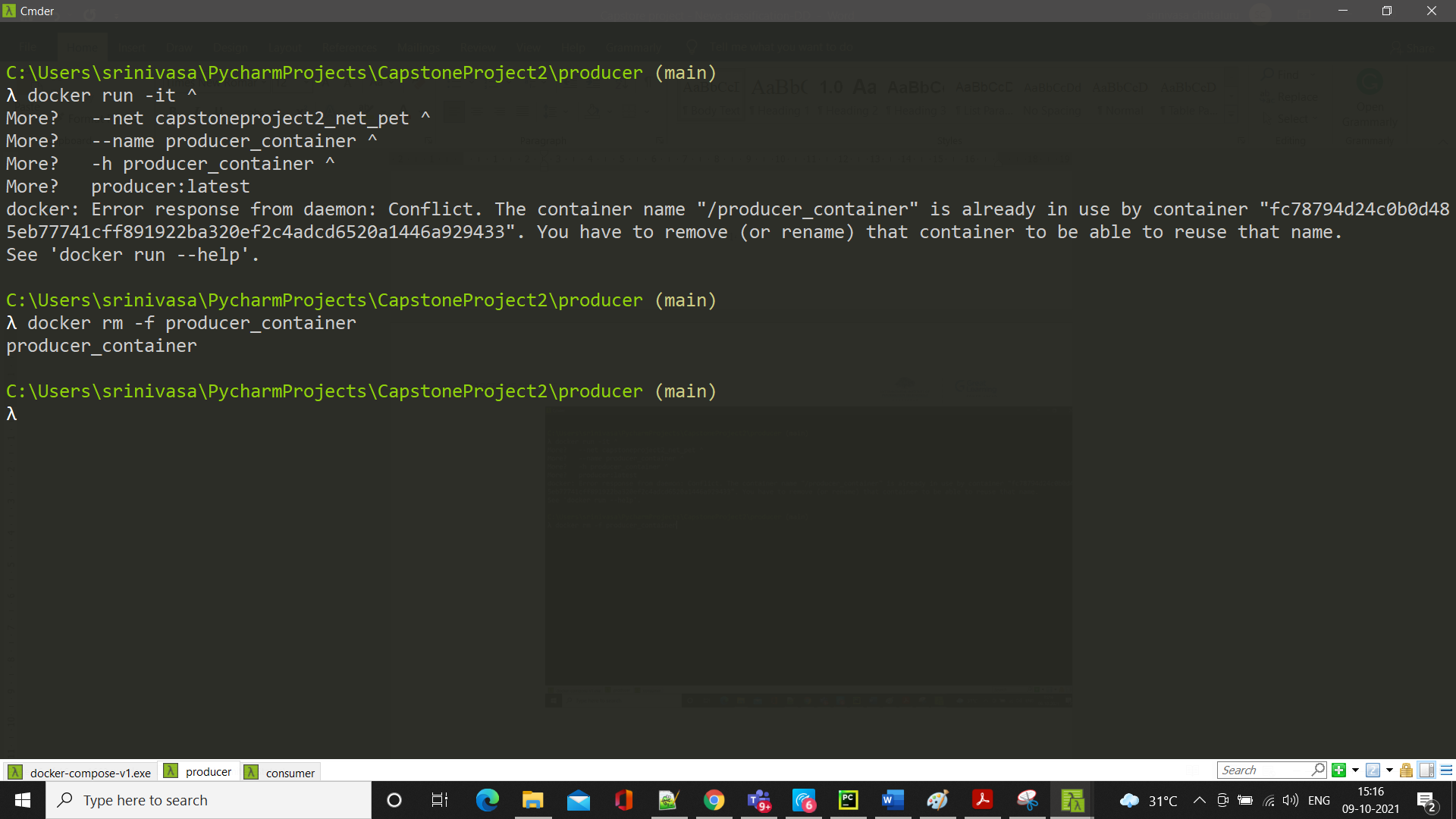
--net capstoneproject2\_net\_pet ^

--name producer\_container ^

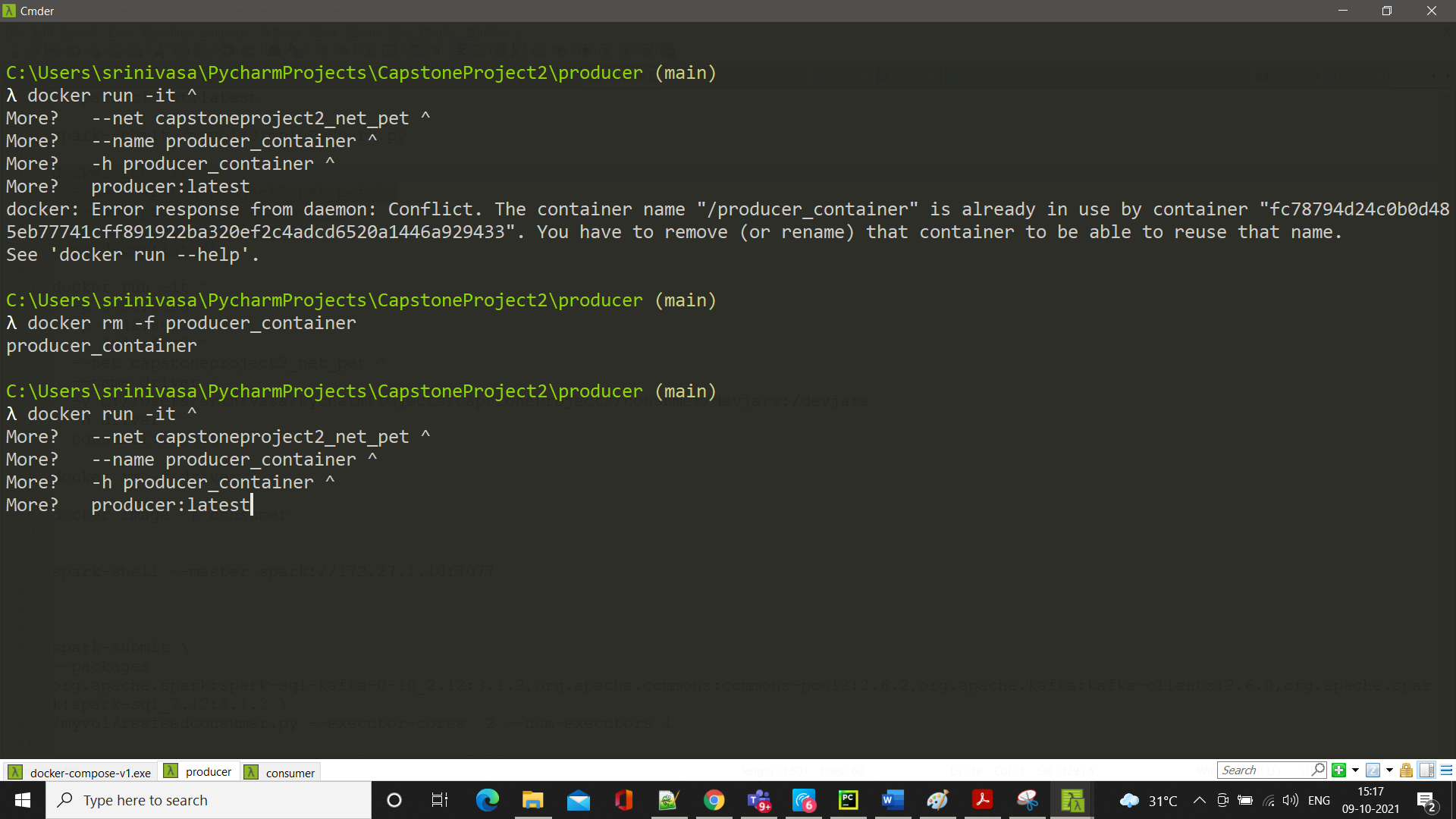
-h producer\_container ^

producer:latest

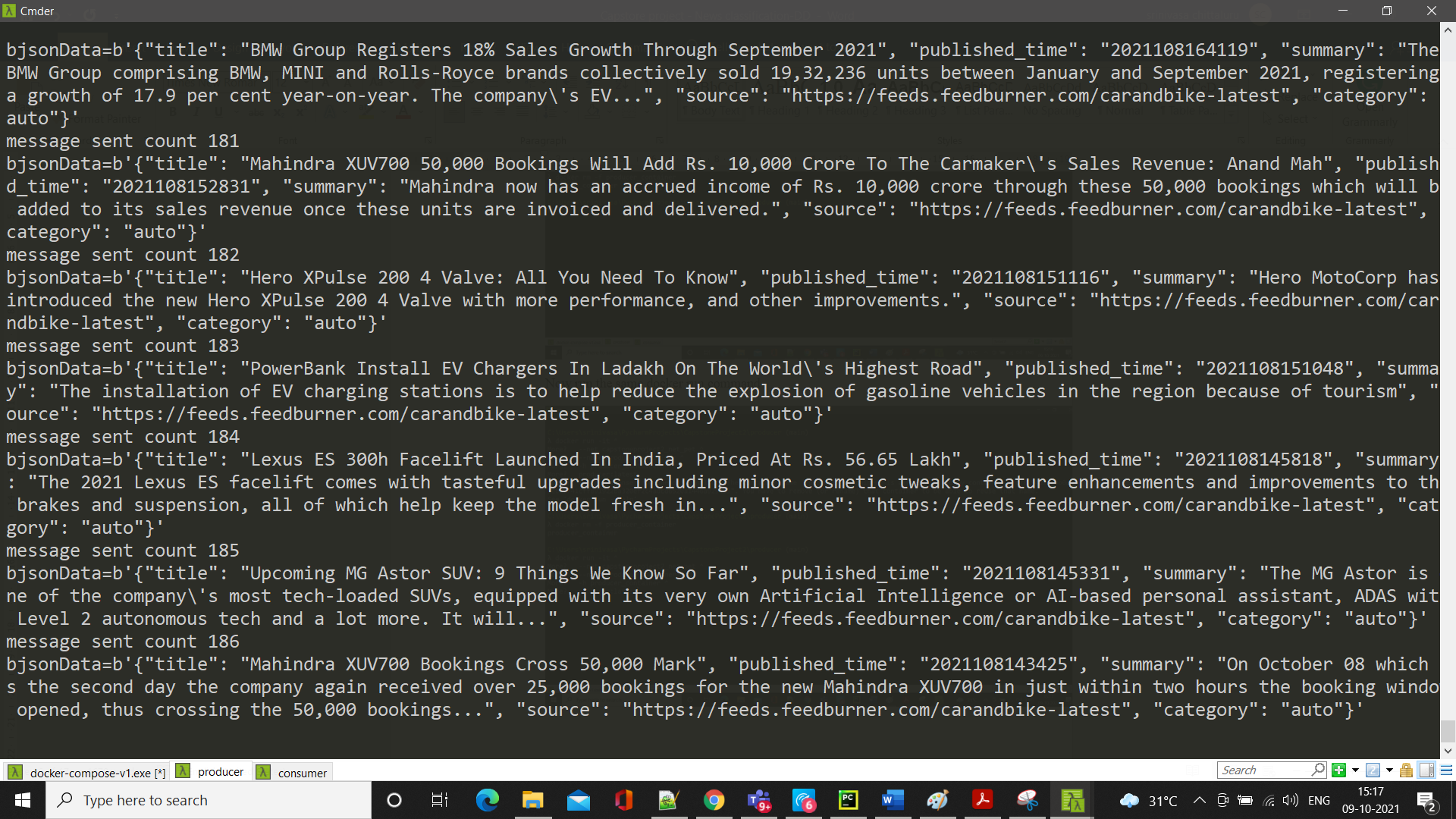
if the name with producer\_container already exist then drop it by docker rm -f producer\_container



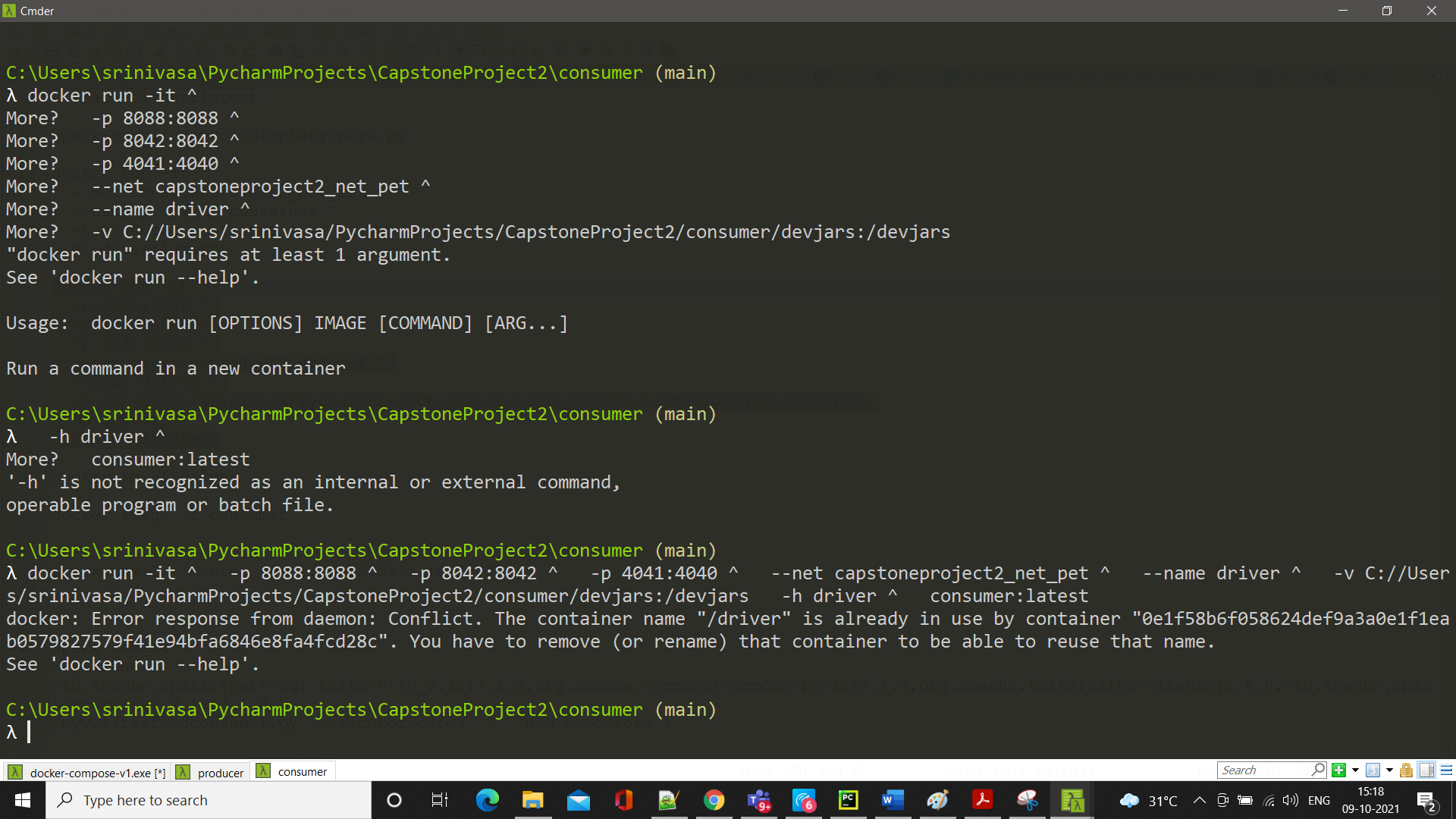
Now run the same docker run command



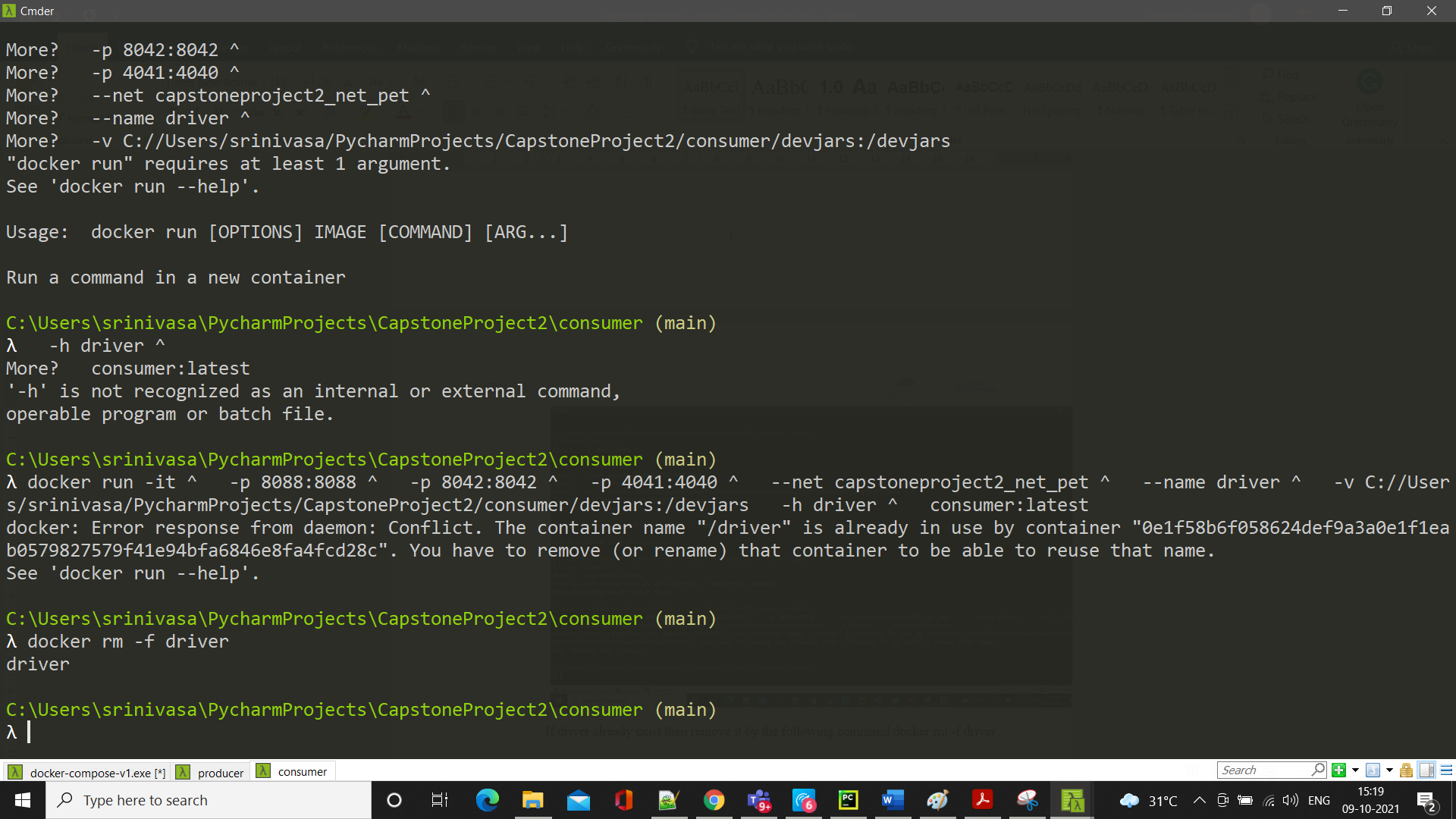
This would send the messages to broker



* + Go to the terminal where you would have opened the consumer and run the following command. In case of cmder “^” used for multi-line command. However, it other terminals it could be “\” for multi-line command. Please note the net, it is same the network where kafka, spark are in



If driver already exist then remove it by the following command docker rm -f driver



Now run the following command. Note the volume directory. You can use the volume directory accordingly. The directory in your local system must be created before starting the consumer. In docker, it creates the volume or mounts the volume automatically

docker run -it ^

-p 8088:8088 ^

-p 8042:8042 ^

-p 4041:4040 ^

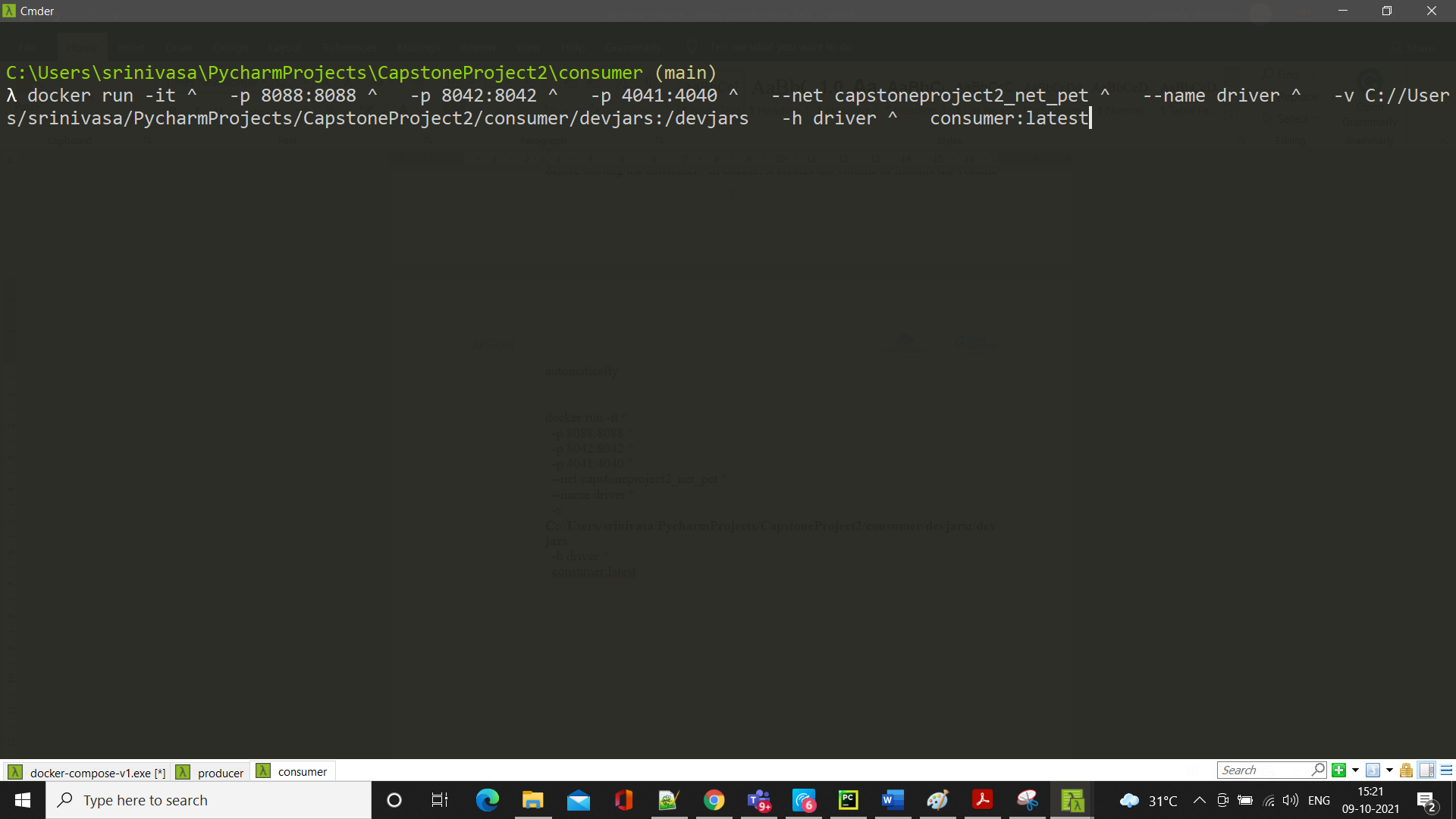
--net capstoneproject2\_net\_pet ^

--name driver ^

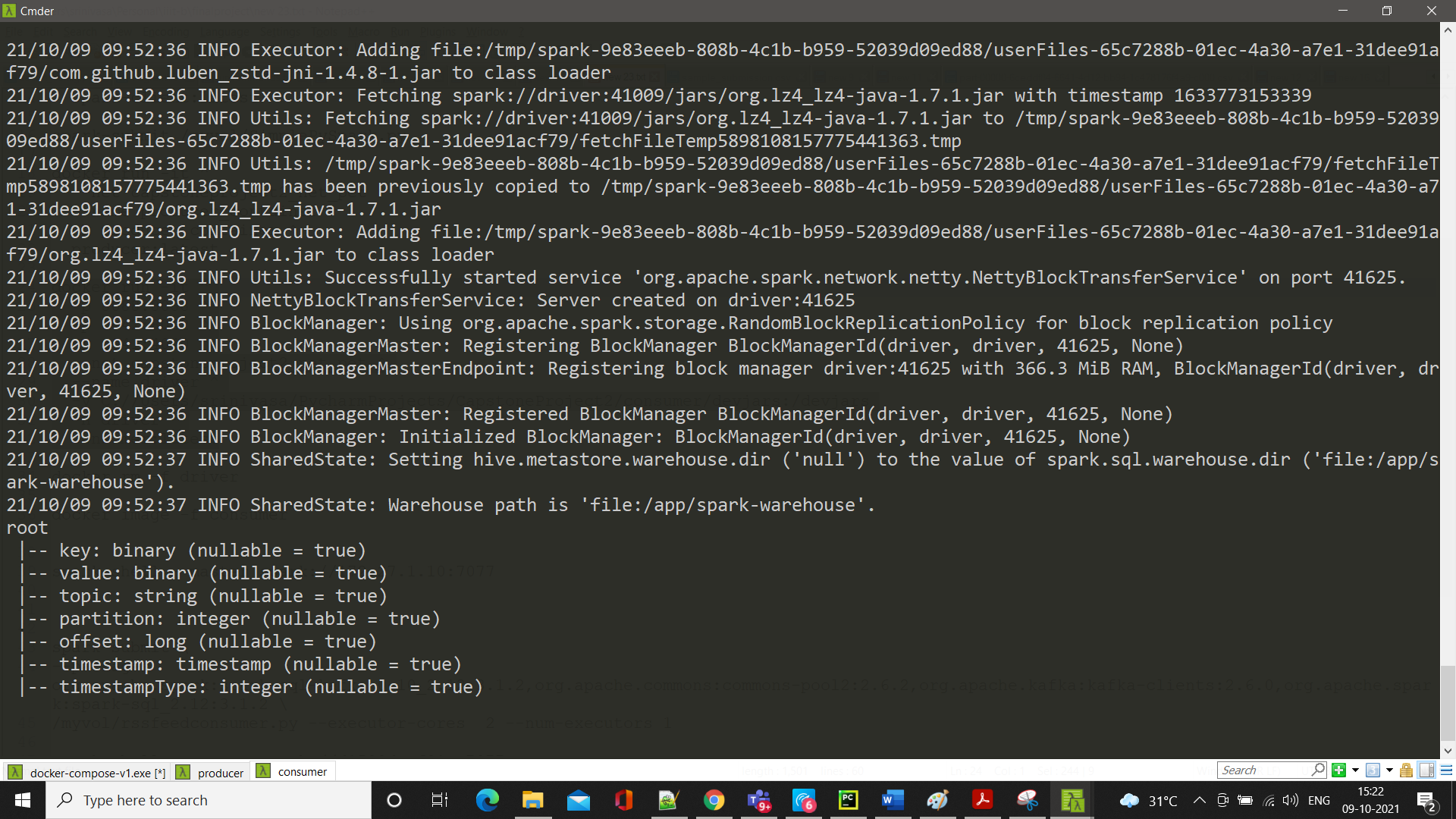
-v **C://Users/srinivasa/PycharmProjects/CapstoneProject2/consumer/devjars:/devjars**

-h driver ^

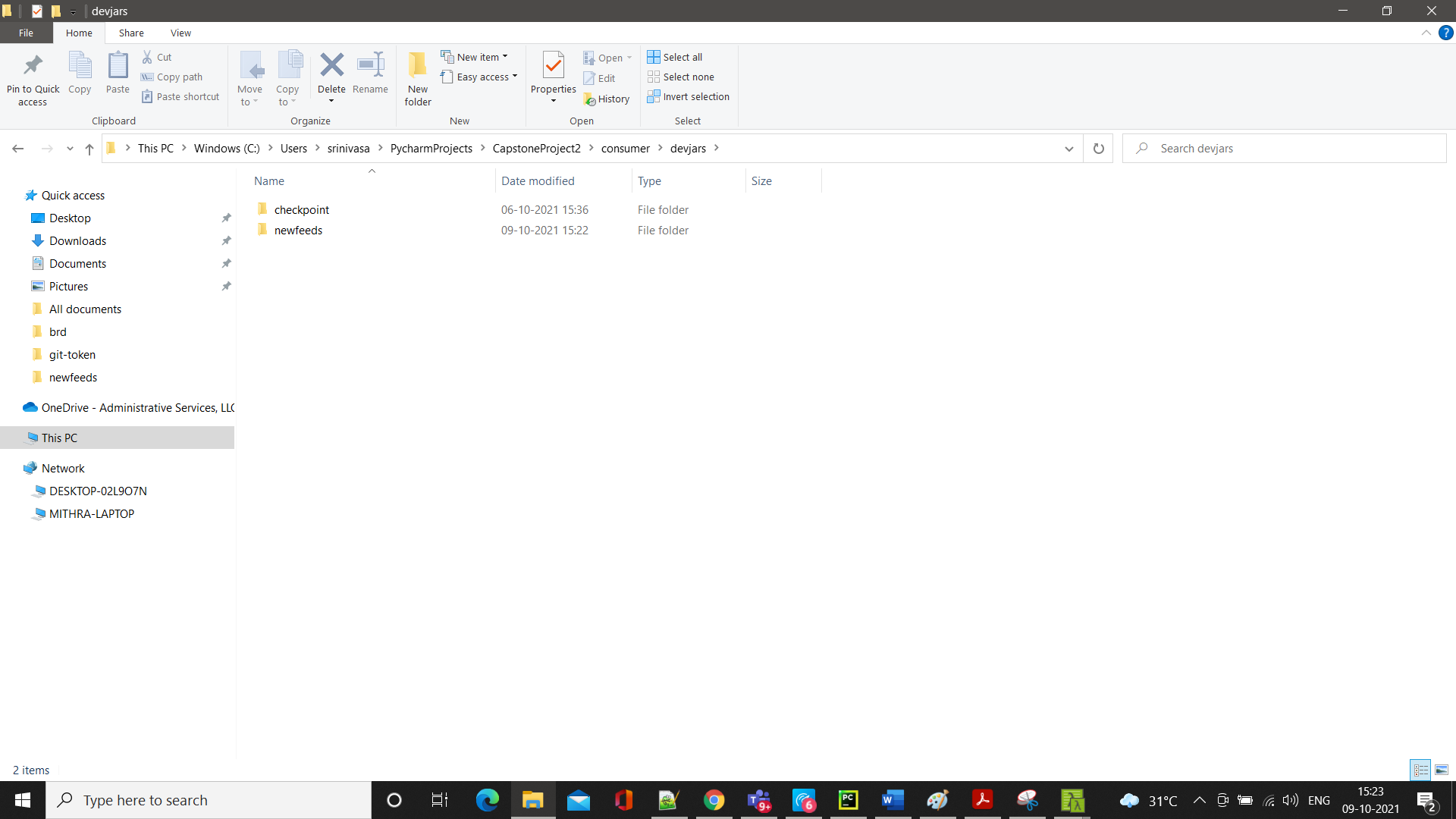
consumer:latest



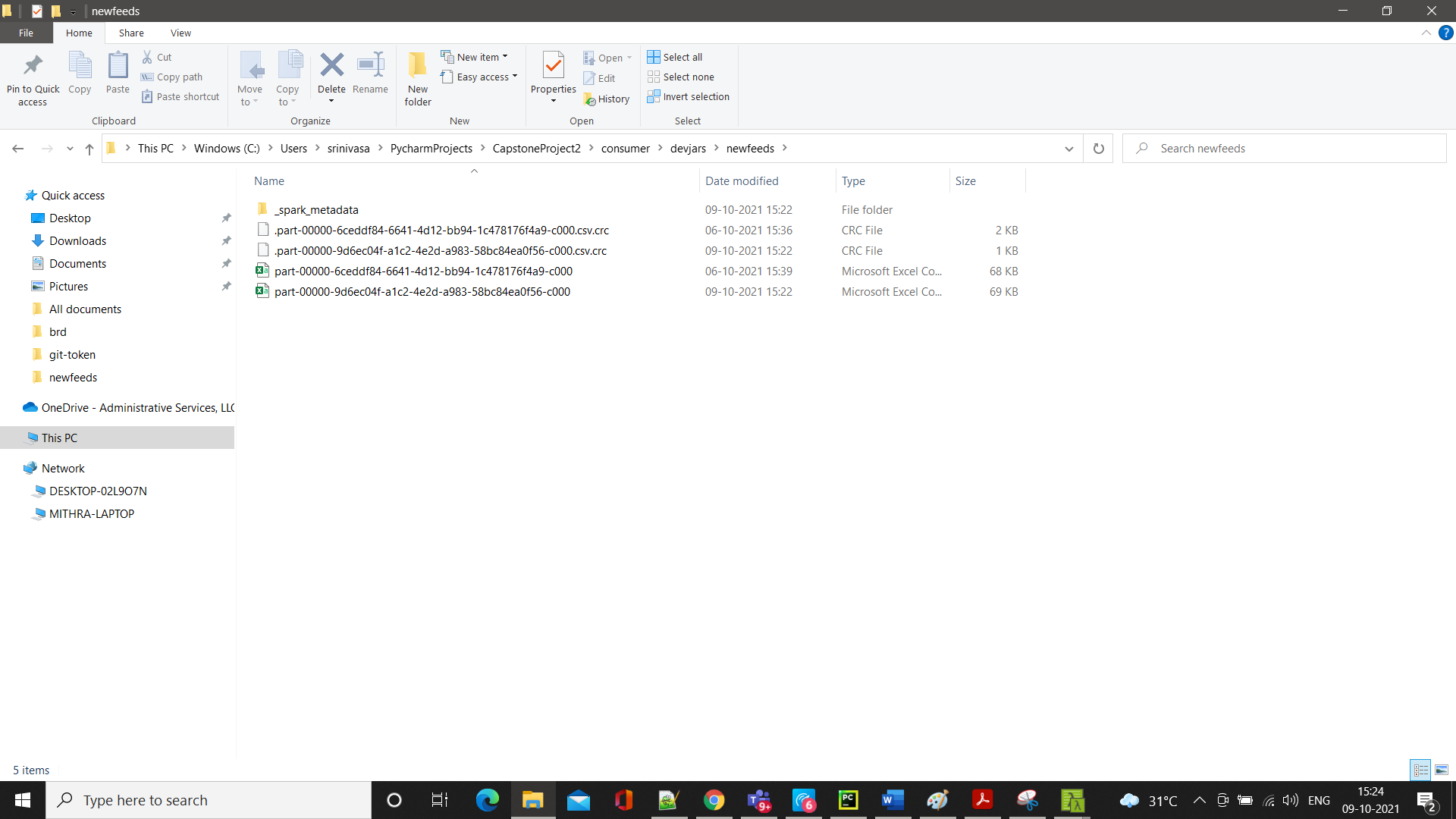
Consumer should be running



Now check the location



Get into news feeds directory where you should see the CSV files

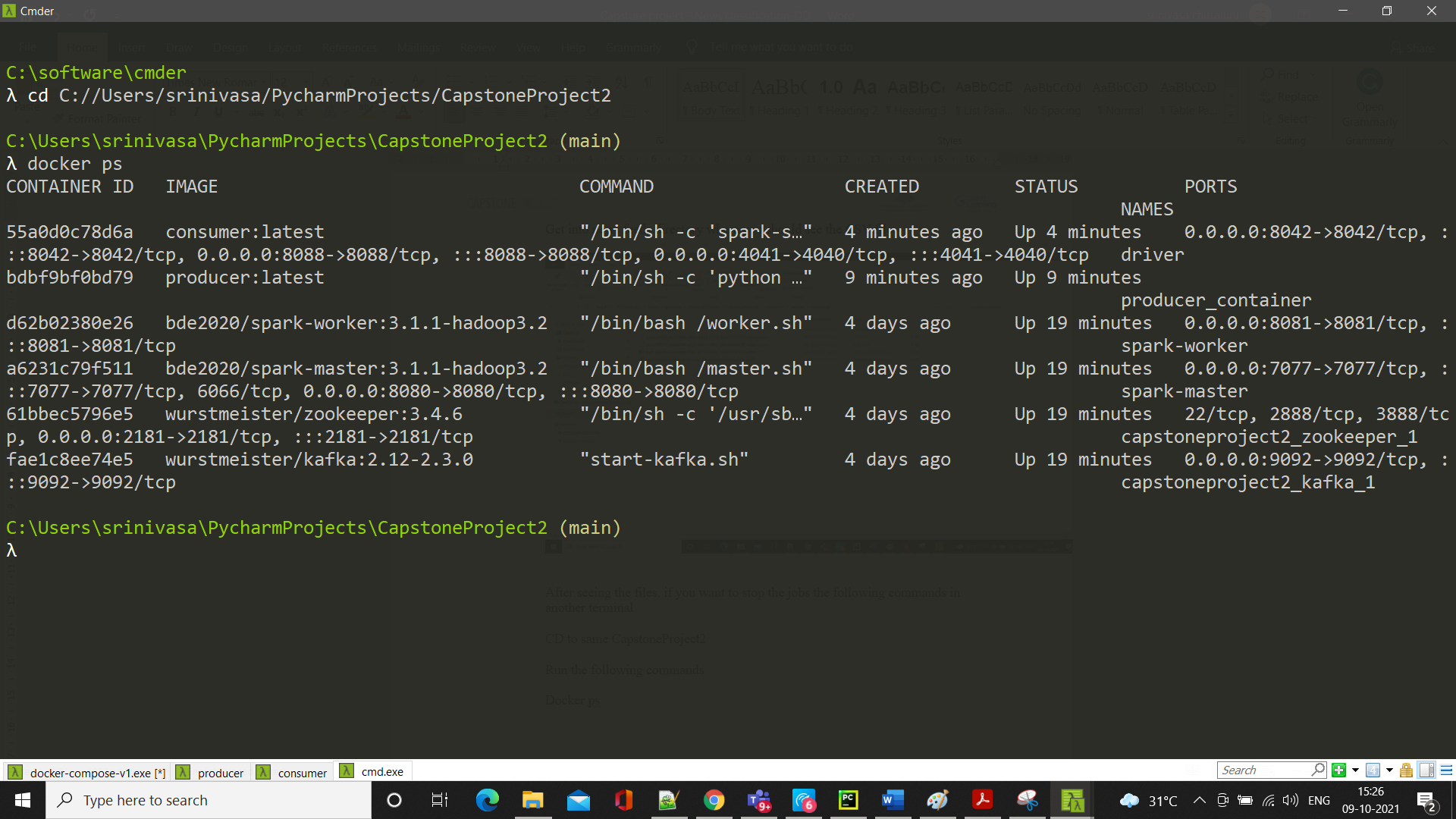


After seeing the files, if you want to stop the jobs the following commands in another terminal

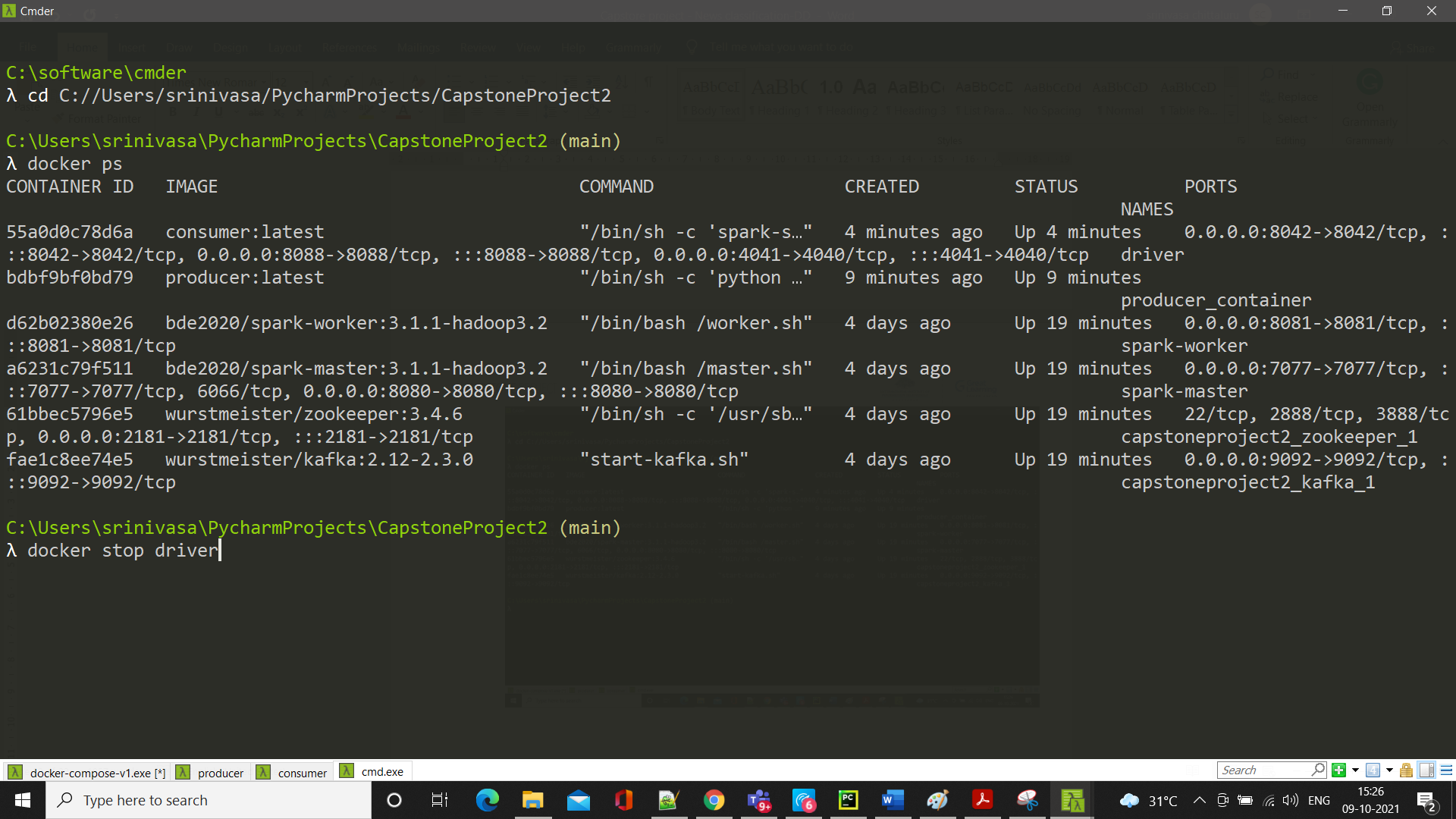
CD to same CapstoneProject2

Run the following commands

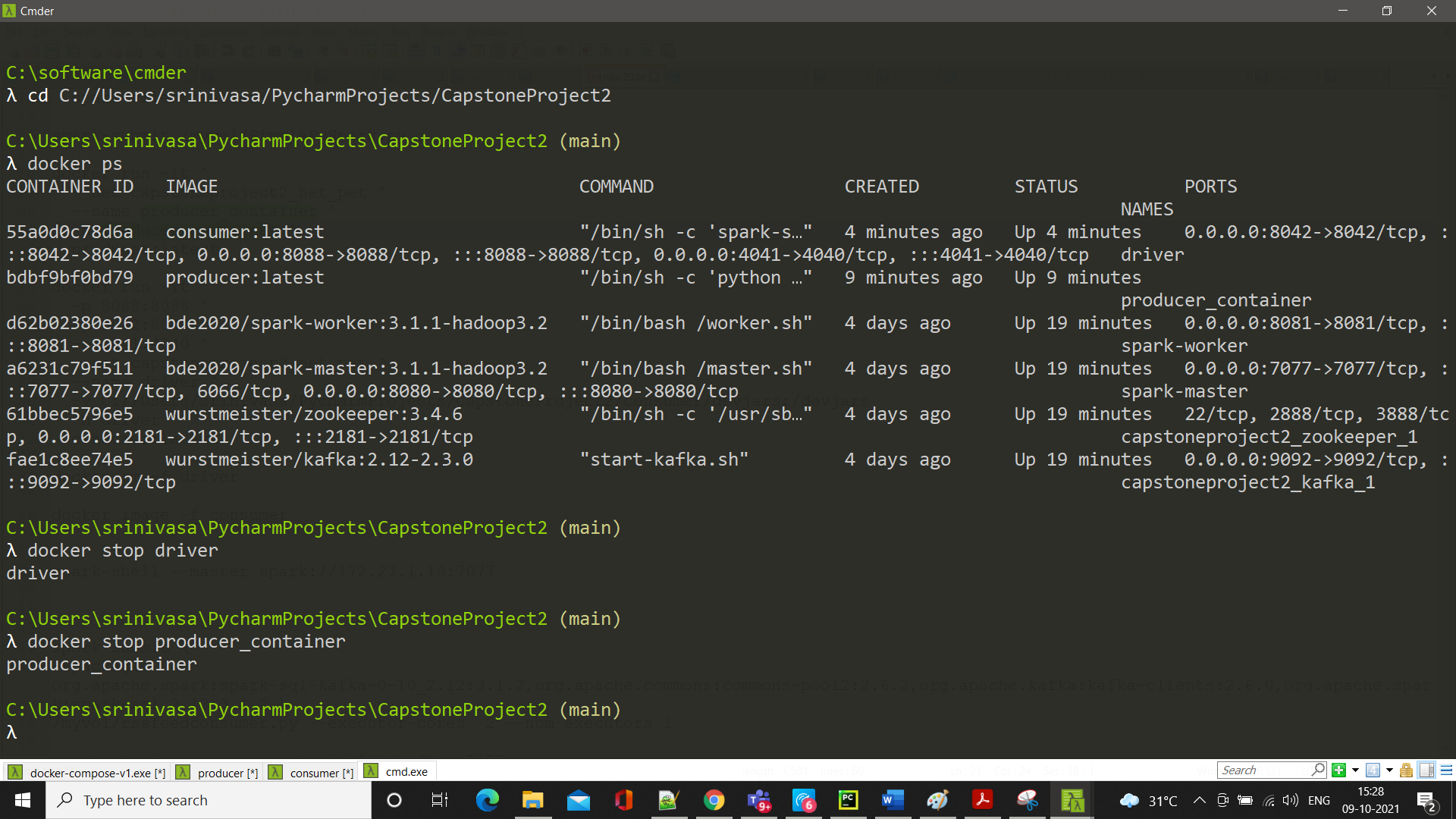
Docker ps



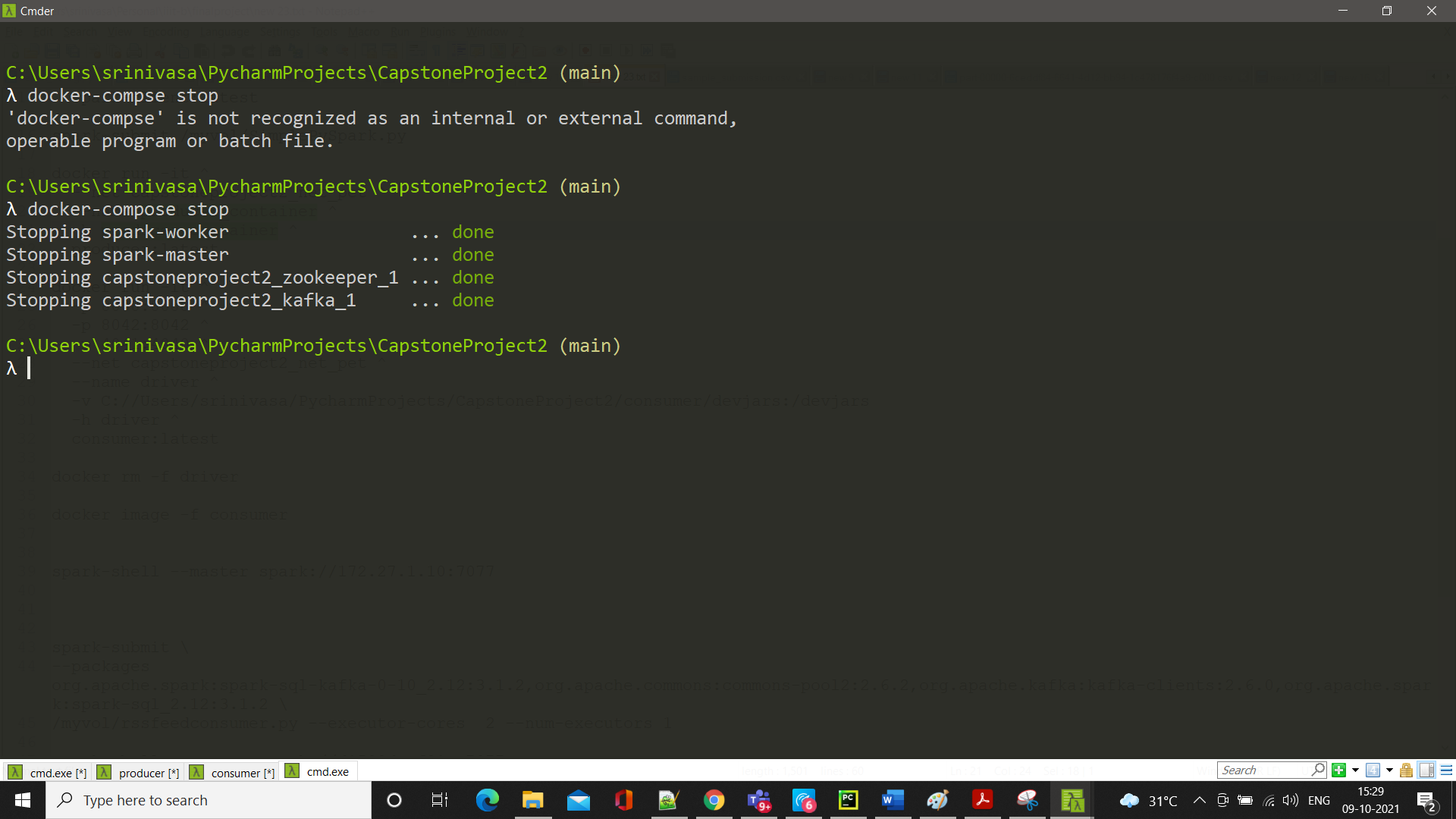
docker stop driver



docker stop producer\_container

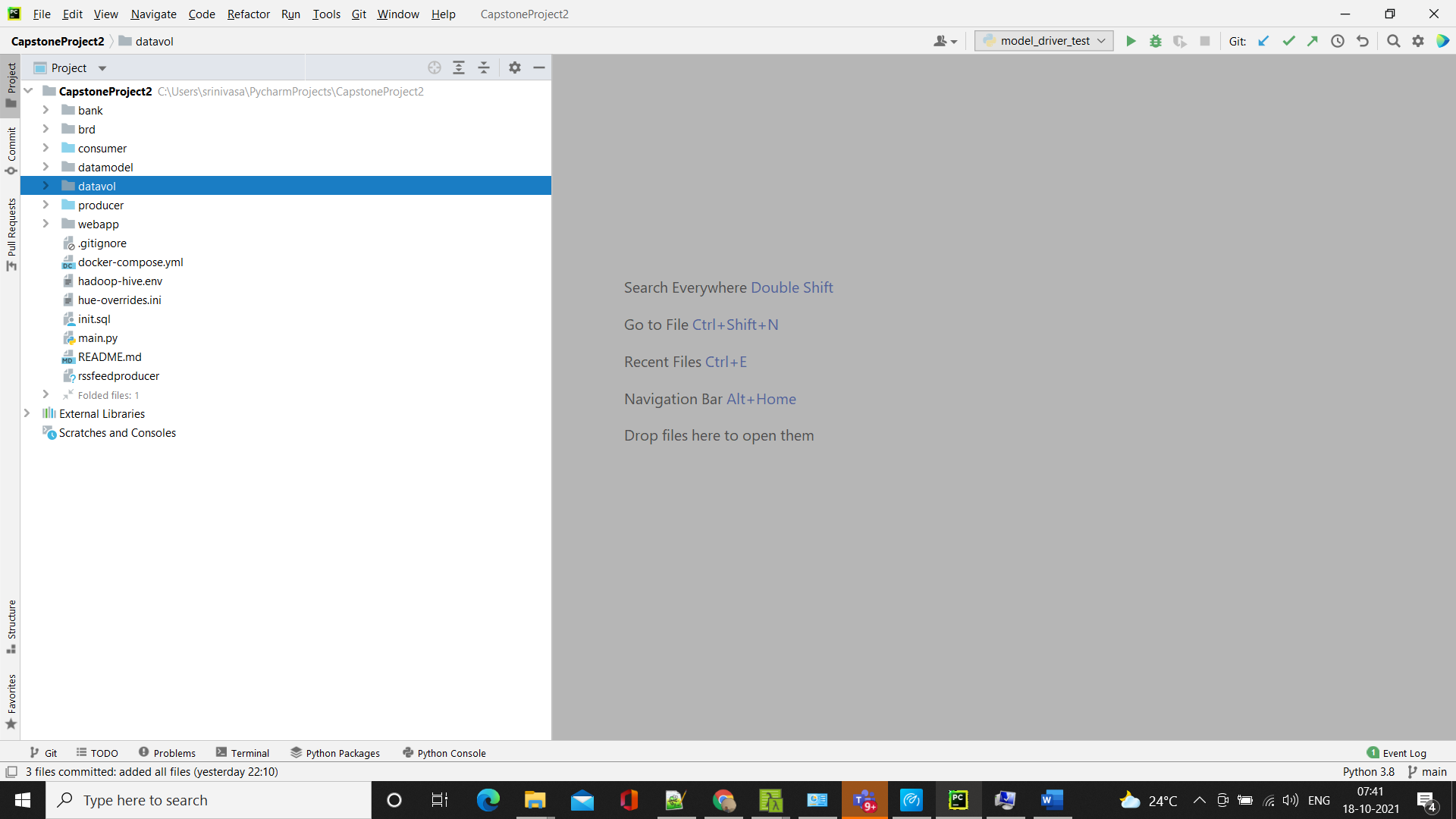


Finally run the following command docker-compose stop



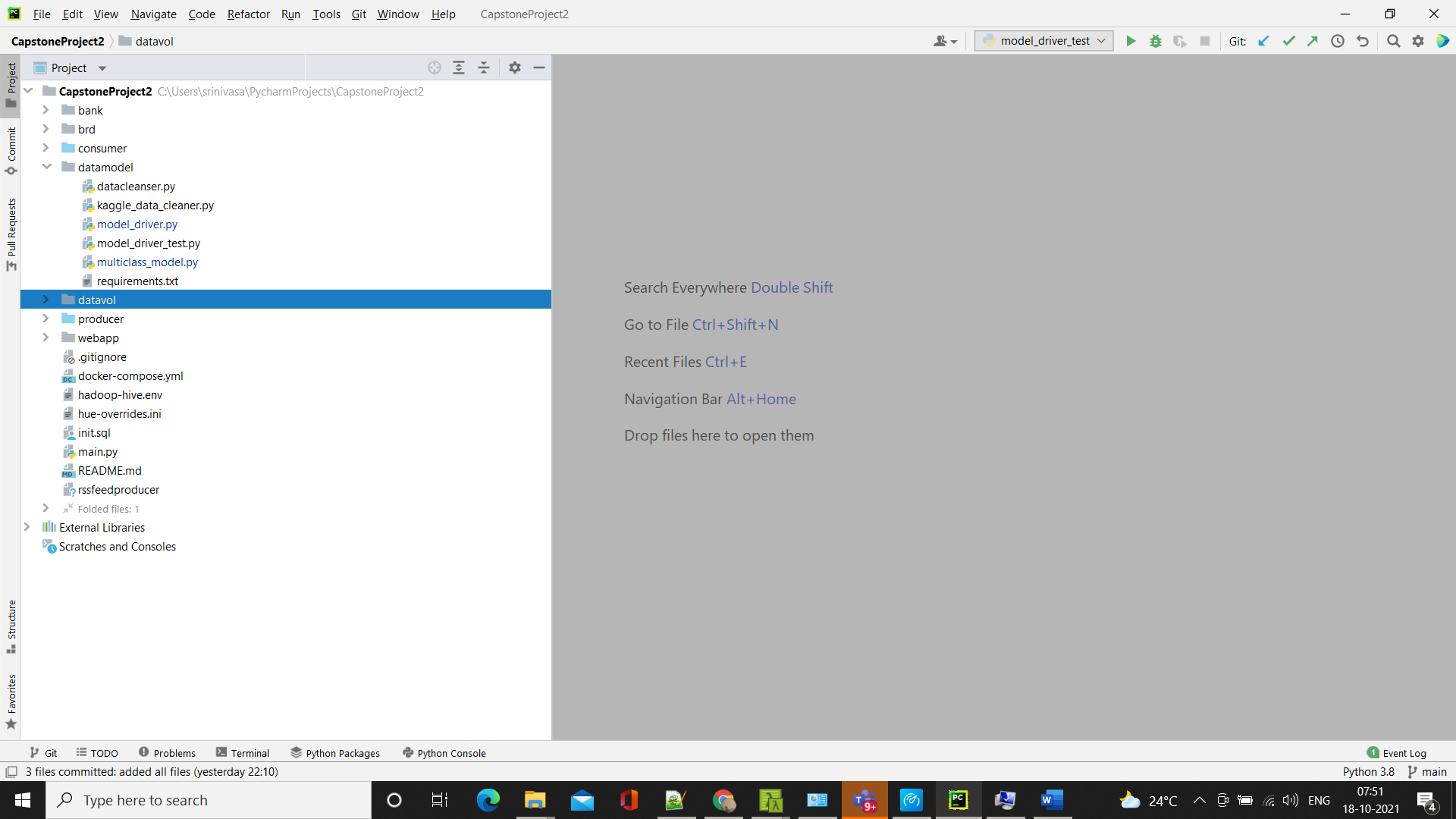
### Week-2 Screen shots and steps to run

After week-1 processing



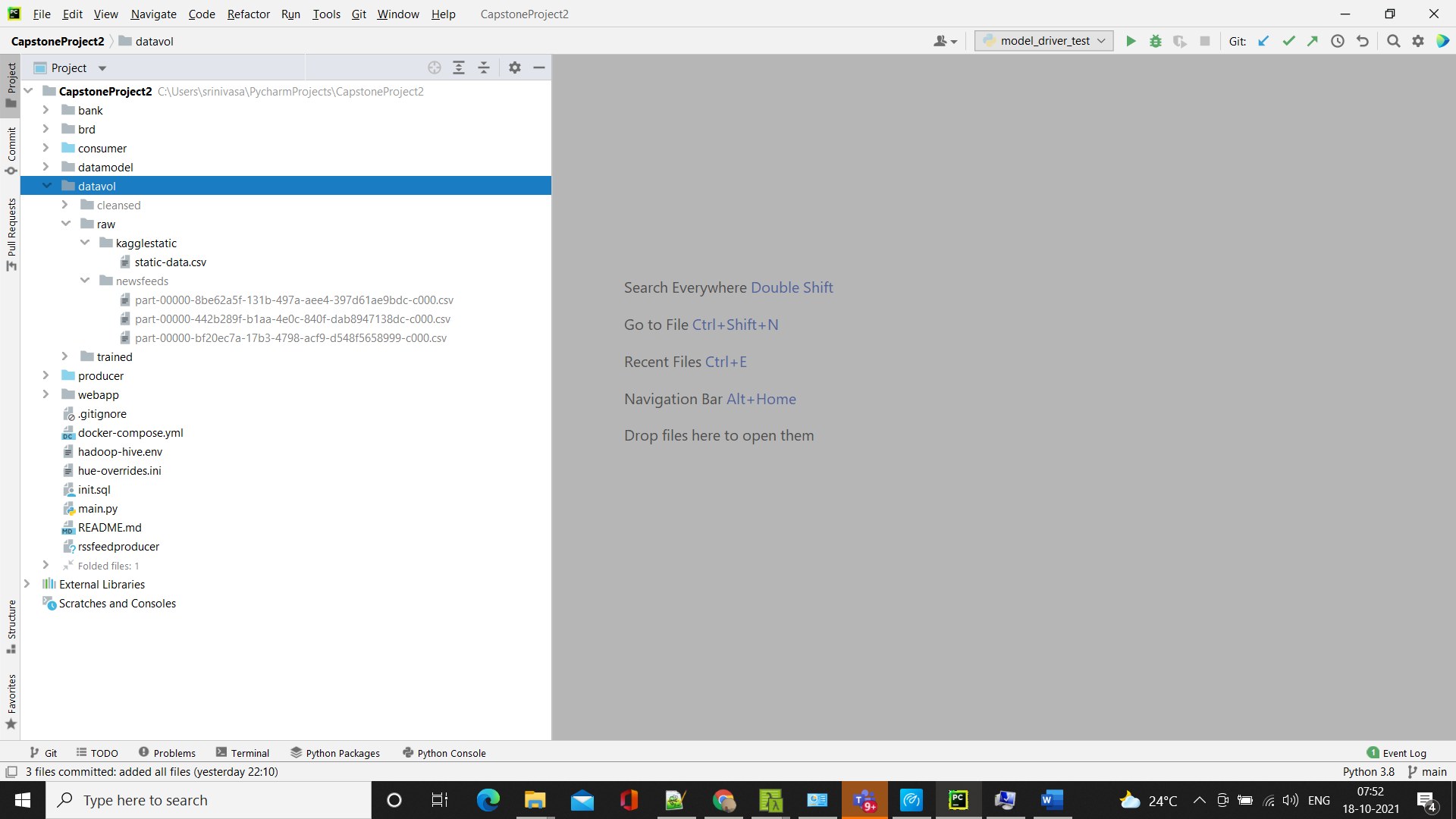
Project structure has majorly Producer, Consumer, Datamodeler, Webapp and Data Volume directories which are needed for these programs. Apart from these 5 directories, you can see docker-composer and directories brd, bank. BRD-Business requirement document which is this document and bank directory is testing purpose



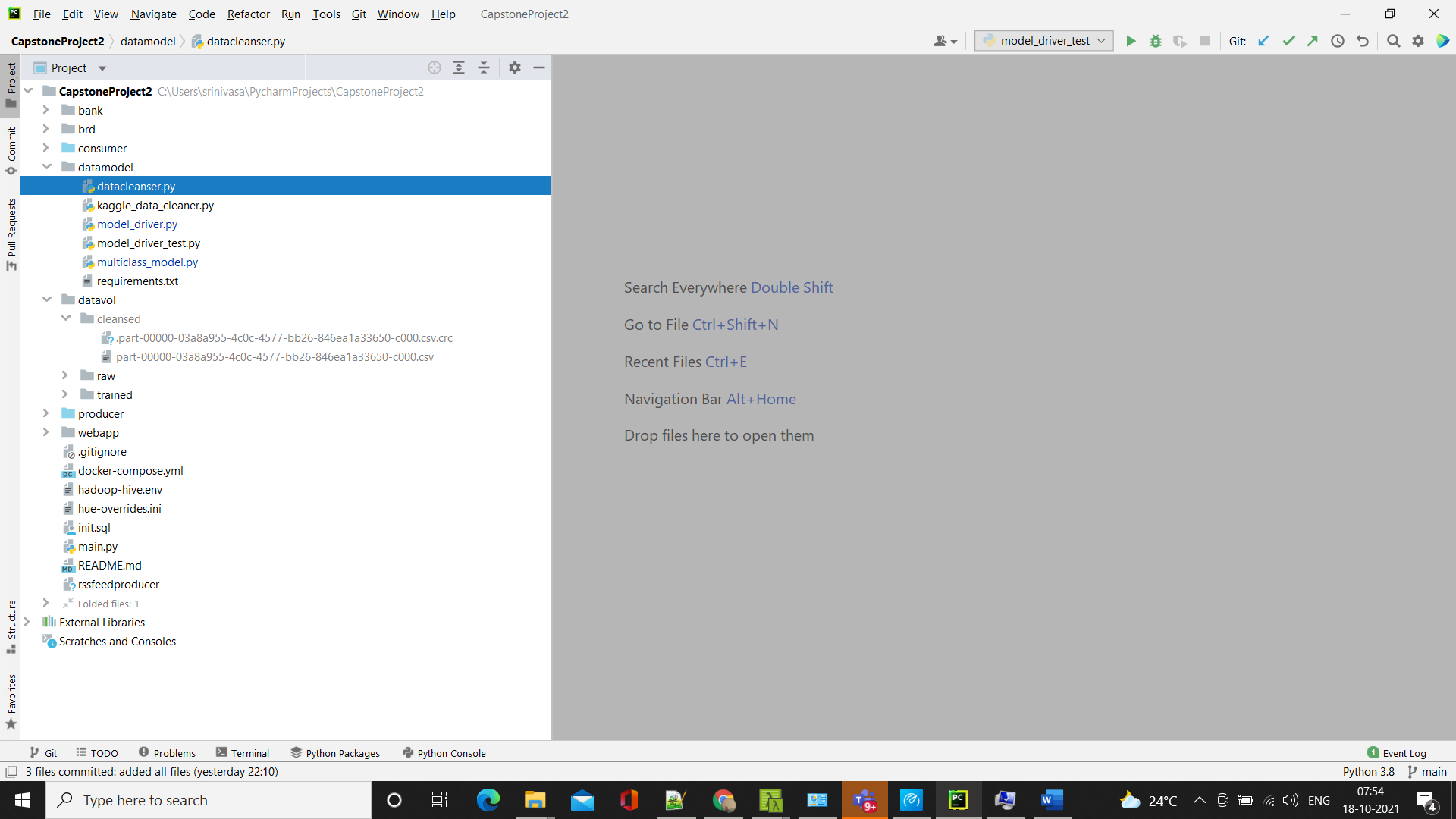


Refer model\_driver program, which calls both datacleanser followed by multicast\_model

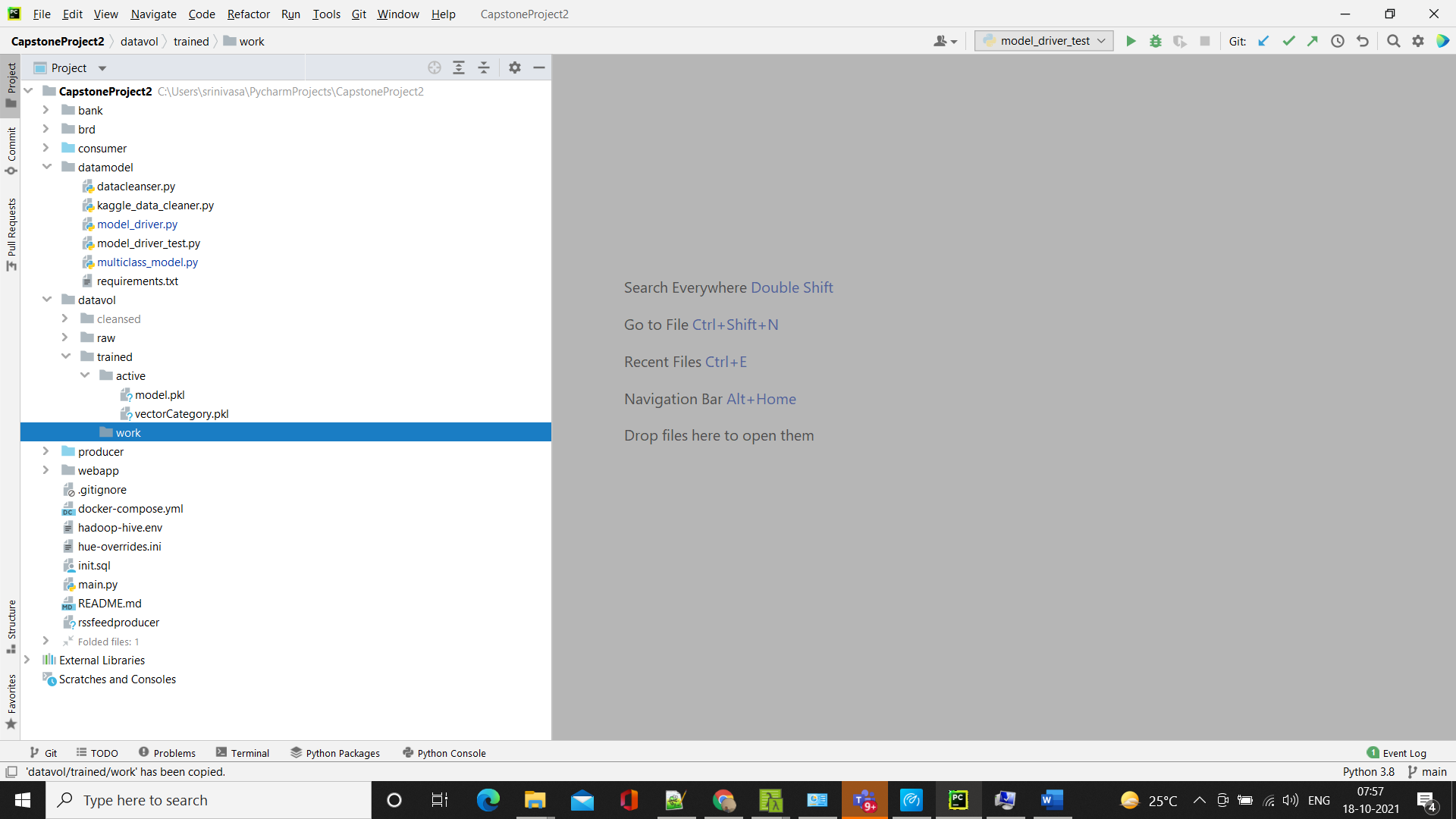
Before running these programs check for the datasets kagglestatic and newsfeeds. Ideally, news feeds dataset come when you run the producer and consumer dockers. The output of the producer and consumer gets stored into newsfeeds



The output of data cleanser gets stored into ../datavol/cleansed



The output of data cleanser ../datavol/cleansed is an input for multicast\_modeler. The output of multicast model gets stored into ../datavol/trained/work and then after 24 hours it gets moved into ../datavol/trained/active



Refer model.pkl and vectorCategory.pkl which are input for predictor.py. As of now, it is tested locally and will be deployed into docker for webapp application

