Università degli Studi di Salerno
DIEM, AY 2022-23
Course: SECURE CLOUD COMPUTING
Lecturers: Angelo Marcelli - Antonio Della Cioppa

Diabetes prediction of a patient

by *Group 06*

Enrico Maria Di Mauro	0622701706	e.dimauro5@studenti.unisa.it
Allegra Cuzzocrea	0622701707	a.cuzzocrea2@studenti.unisa.it

Summary

1. Introduction

The aim of the project is the exploration, training and serving of a machine learning model for Time Series Data forecasting by leveraging Kubeflow's main components.

The Time Series Data forecasting task consists in **predicting accurate blood glucose levels in Type 1 Diabetes patients**. Blood glucose level prediction is a challenging task for AI researchers, with the potential to improve the health and wellbeing of people with diabetes. Knowing in advance when blood glucose is approaching unsafe levels provides time to pro-actively avoid hypo- and hyperglycaemia and their concomitant complications. The drive to perfect an artificial pancreas has increased the interest in using machine learning approaches to improve prediction accuracy.

The problem to be solved is a time-series regression problem represented by the glucose levels in future instants of times, i.e., horizon $\boldsymbol{h}\Delta(\boldsymbol{t})$.

This regression is affected by considering available the glucose level $\boldsymbol{G}(\boldsymbol{t})$, the injected insulin $\boldsymbol{I}(\boldsymbol{t})$ and the carbohydrates intake $\boldsymbol{C}(\boldsymbol{t})$ in a time window of $\boldsymbol{k}\Delta(\boldsymbol{t})$ before the current instant $\boldsymbol{t}$.

The time series prediction task is transformed in the following **regression problem**:

$$\boxed{\hat{\boldsymbol{G}}(\boldsymbol{t}+\boldsymbol{h}\Delta(\boldsymbol{t})) = \boldsymbol{f}\left(\boldsymbol{G}(\boldsymbol{t}), \boldsymbol{G}(\boldsymbol{t}-\Delta(\boldsymbol{t})), \dots, \boldsymbol{G}(\boldsymbol{t}-\boldsymbol{k}\Delta(\boldsymbol{t})), \boldsymbol{I}(\boldsymbol{t}), \boldsymbol{I}(\boldsymbol{t}-\Delta(\boldsymbol{t})), \dots, \boldsymbol{I}(\boldsymbol{t}-\boldsymbol{k}\Delta(\boldsymbol{t})), \boldsymbol{C}(\boldsymbol{t}), \boldsymbol{C}(\boldsymbol{t}-\Delta(\boldsymbol{t})), \dots, \boldsymbol{C}(\boldsymbol{t}-\boldsymbol{k}\Delta(\boldsymbol{t}))\right)}$$

In this way having past values of glucose, insulin and carbohydrates it is possible to predict next glucose levels.

The **primary goals** of the project are:

- **Design a Kubeflow Pipeline** that creates a patient personalized ML prediction model using its recorded data
- **Design and deploy a Web App** that exploits the patient model and makes the glucose prediction to determine if there is the risk of a diabetes complication

2. Software

- **Kubeflow** and **Arrikto**: Kubeflow is an open-source platform for machine learning on Kubernetes, which simplifies the deployment of machine learning workflows. Kubeflow includes tools for model development (Kubeflow Notebooks), training and validation (Kubeflow Pipelines), monitoring and management (Kubeflow Dashboard) and automated machine learning (Katib). Arrikto offers a managed Kubeflow service called the **Arrikto Managed Kubeflow (AMK)**, which provides a fully managed and highly available Kubeflow environment. AMK makes it easy to use Kubeflow by abstracting away the complexity of deploying and managing Kubeflow on Kubernetes. Kubeflow and Arrikto can be used together to provide a powerful and flexible platform for building, training and deploying machine learning models. AMK makes it easy to deploy and manage Kubeflow on Kubernetes, while Kubeflow provides the tools and infrastructure needed to build, train and serve machine learning models at scale
- **Visual Studio Code**: it is a free, open-source code editor that supports multiple programming languages. It's a popular choice among developers due to its versatility and extensibility through the use of plugins and extensions
- **Streamlit**: it is an open-source Python library that allows developers to create interactive web applications for machine learning and data science quickly and easily with only a few lines of code
- **Docker**: it is a software platform that allows developers to package and deploy applications as portable containers. It provides a standardized way to build and distribute applications, making it easier to deploy and run them consistently across different computing environments. Docker containers are self-contained and include everything needed to run the application. This makes it easier to build and deploy applications in a consistent and reliable way and reduces the risk of compatibility issues between different computing environments
- **Kubernetes**: it is an open-source container orchestration platform that automates the deployment, scaling and management of containerized applications. It provides a way to manage and orchestrate containers across multiple hosts, making it easier to deploy and run applications in a scalable and fault-tolerant manner. Kubernetes uses a declarative configuration to define the desired state of the application and continuously monitors the actual state to ensure that it matches the desired state. It can automatically scale the application up or down based on resource utilization and can also handle failover and self-healing of the application in case of node failures or other issues. It supports a wide range of container runtimes, including Docker and provides a rich set of features such as load balancing, service discovery and rolling updates for container-based applications

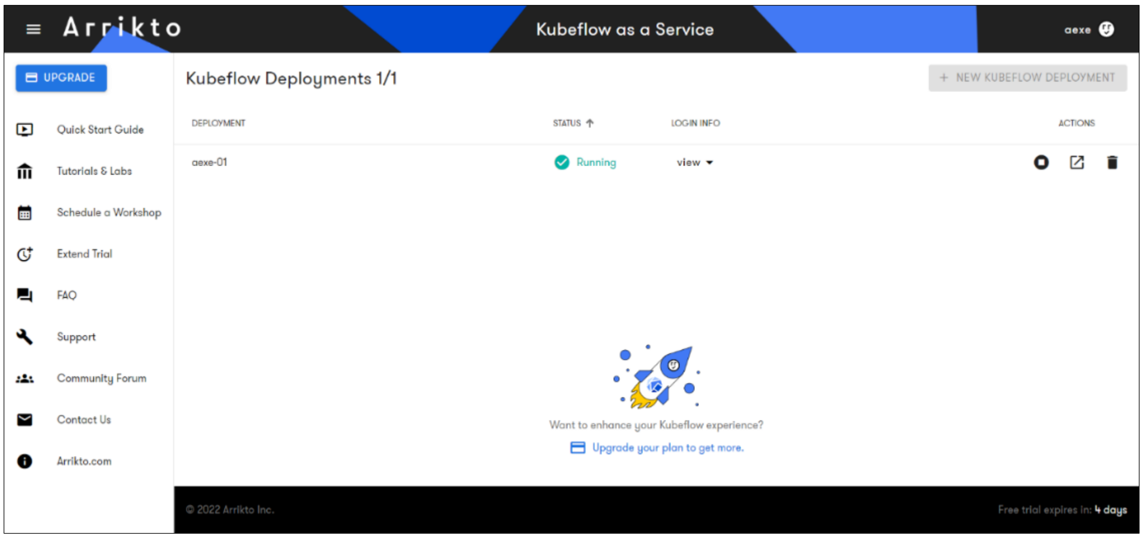
Note that everything was tested on both Operating Systems **Windows** and **Linux**. For the second one it was used **Windows Subsystem for Linux (WSL)** that is a Windows feature that permits to run Linux commands and applications natively on a Windows system without the need for a virtual machine or dual-boot setup

3. Pipeline

This section aims to present all steps that led to the generation of the Pipeline. In particular, the service **Arrikto Managed Kubeflow** was used to create the Pipeline and so to build, train and validate the patients personalized models.

3.1. Arrikto Deployment

The **registration** to Arrikto permitted to create a deployment. This took about 40 minutes.



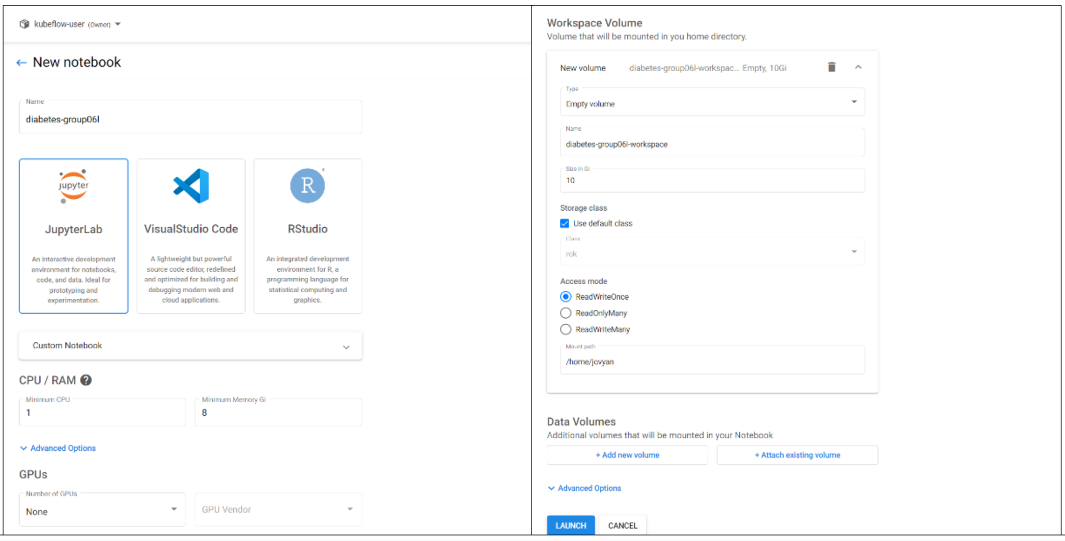
Kubeflow Deployments 1/1

DEPLOYMENT	STATUS	LOGIN INFO	ACTIONS
aexe-01	Running	view	

3.2. Kubeflow Notebook

After the deployment configuration a **Jupiter Notebook** was created with the following setup:

- **CPU**: 1
- **RAM**: 8 GB
- **Storage**: 10 GB


kubeflow-user (Owner)
JupyterLab
An interactive development environment for notebooks, code, and data. Ideal for prototyping and experimentation.
VisualStudio Code
A lightweight but powerful source code editor, redefined and optimized for building and debugging modern web and cloud applications.
RStudio
An integrated development environment for R, a programming language for statistical computing and graphics.
Advanced Options
+ Add new volume
+ Attach existing volume
Advanced Options
LAUNCH
CANCEL

New notebook

Name: diabetes-group06l

Custom Notebook

CPU / RAM

Minimum CPU: 1

Minimum Memory Gi: 8

GPUs

Number of GPUs: None

GPU Vendor

Workspace Volume

Volume that will be mounted in you home directory.

New volume diabetes-group06l-workspac... Empty, 10Gi

Type: Empty volume

Name: diabetes-group06l-workspace

Size in Gi: 10

Storage class

☑ Use default class

Class: rok

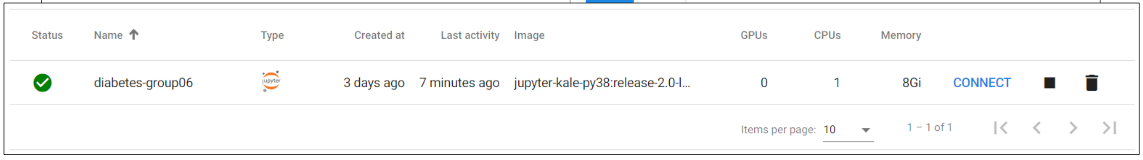
Access mode

[✓ ReadWriteOnce] ReadOnlyMany ReadWriteMany

Mount path: /home/jovyan

Data Volumes

Additional volumes that will be mounted in your Notebook

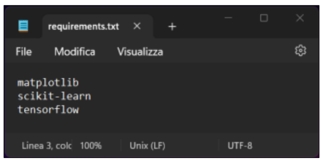


Status	Name	Type	Created at	Last activity	Image	GPUs	CPUs	Memory
✓	diabetes-group06	jupyter	3 days ago	7 minutes ago	jupyter-kale-py38:release-2.0-l...	0	1	8Gi

3.3. Kubeflow Pipeline

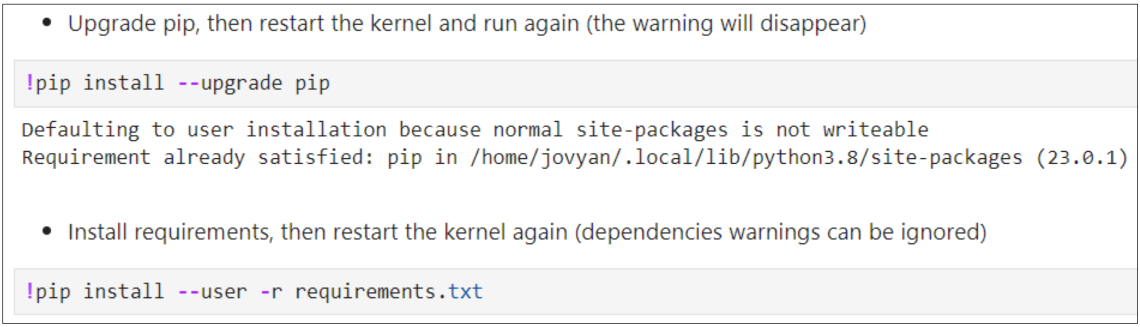
3.3.1. Prerequisites

The file **'requirements.txt'** is provided in order to install all necessary packages for the Pipeline. Before installing them, it is advisable to upgrade the pip package. It is possible to run the following commands:



```
matplotlib
scikit-learn
tensorflow
```

```
!pip install --upgrade pip

!pip install --user -r requirements.txt
```



- Upgrade pip, then restart the kernel and run again (the warning will disappear)

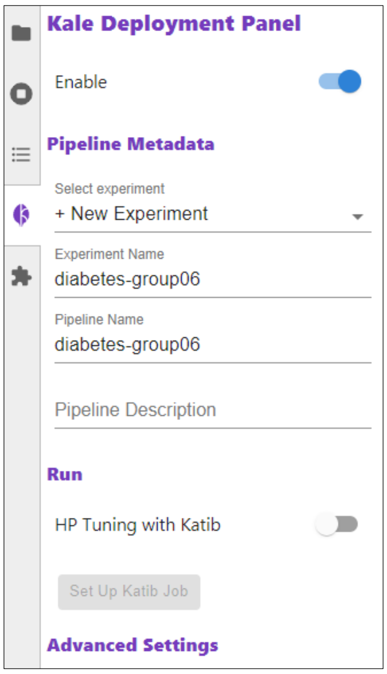
```
!pip install --upgrade pip
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pip in /home/jovyan/.local/lib/python3.8/site-packages (23.0.1)
```

- Install requirements, then restart the kernel again (dependencies warnings can be ignored)

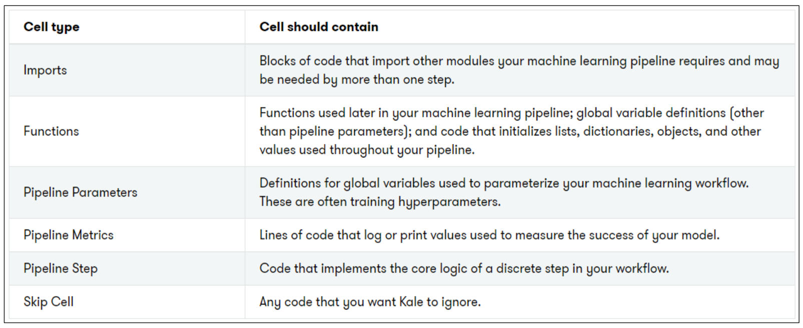
```
!pip install --user -r requirements.txt
```

Another prerequisite is enabling the **Kale Deployment Panel**. In fact, Kale is a tool that permits to create a Kubeflow Pipeline from a Jupyter Notebook. In particular, it is possible to annotate the cells of the Notebook selecting from 6 Kale cell types. The annotations are used by Kale to define the Pipeline, specifically to manage dependencies and marshal data correctly as inputs and outputs for each step.



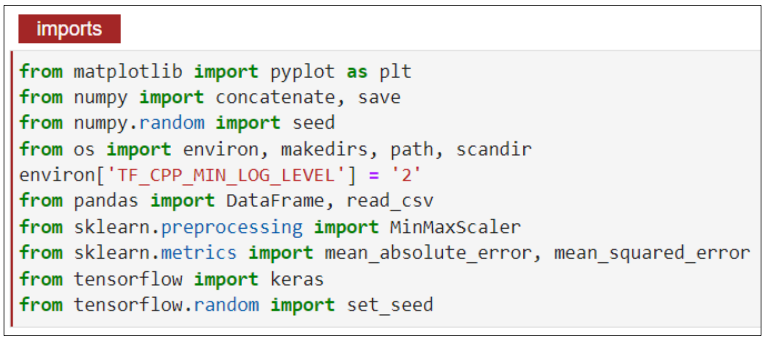
Enable: on
Select experiment: + New Experiment
Experiment Name: diabetes-group06
Pipeline Name: diabetes-group06
Pipeline Description
HP Tuning with Katib: off

Below it is shown the list and a brief summary of **cell types**.



Cell type	Cell should contain
Imports	Blocks of code that import other modules your machine learning pipeline requires and may be needed by more than one step.
Functions	Functions used later in your machine learning pipeline; global variable definitions (other than pipeline parameters); and code that initializes lists, dictionaries, objects, and other values used throughout your pipeline.
Pipeline Parameters	Definitions for global variables used to parameterize your machine learning workflow. These are often training hyperparameters.
Pipeline Metrics	Lines of code that log or print values used to measure the success of your model.
Pipeline Step	Code that implements the core logic of a discrete step in your workflow.
Skip Cell	Any code that you want Kale to ignore.

The **imports** in the Notebook are the following ones:

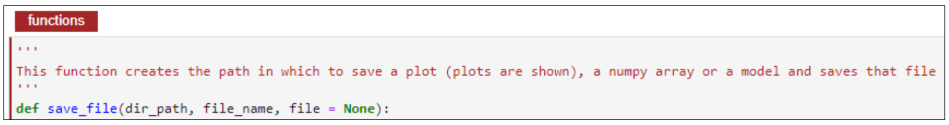


```
from matplotlib import pyplot as plt
from numpy import concatenate, save
from numpy.random import seed
from os import environ, makedirs, path, scandir
environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
from pandas import DataFrame, read_csv
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
from tensorflow import keras
from tensorflow.random import set_seed
```

3.3.2. Components

In this section all Pipeline components are reported and the role of each of them is explained.

3.3.2.1. `save_file`



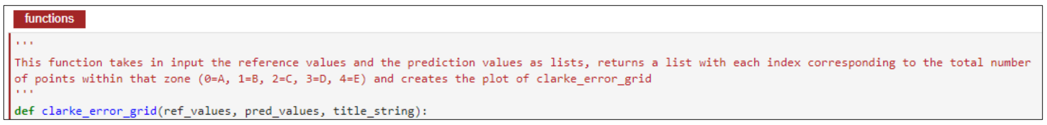
```
'''
This function creates the path in which to save a plot (plots are shown), a numpy array or a model and saves that file
'''
def save_file(dir_path, file_name, file = None):
```

This function is useful because it permits to save all kinds of file used. In particular, it saves plots, numpy arrays and models.

It takes in input:

- `dir_path`: path where to save the file. If it does not exist, the function creates it
- `file_name`: name of the file to save
- `file`: optional parameter. If it is not specified, the function saves a plot otherwise it saves the file using the appropriate method based on the extension

3.3.2.2. `clarke_error_grid`

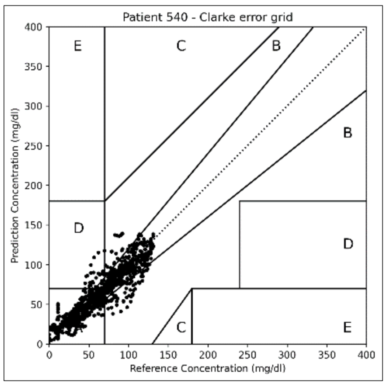


```
'''
This function takes in input the reference values and the prediction values as lists, returns a list with each index corresponding to the total number
of points within that zone (0=A, 1=B, 2=C, 3=D, 4=E) and creates the plot of clarke_error_grid
'''
def clarke_error_grid(ref_values, pred_values, title_string):
```

This function generates an image representing the **Clarke Error Grid**.

It takes in input:

- `ref_values`: list containing actual values
- `pred_values`: list containing predicted values
- `title_string`: string containing the title of the image



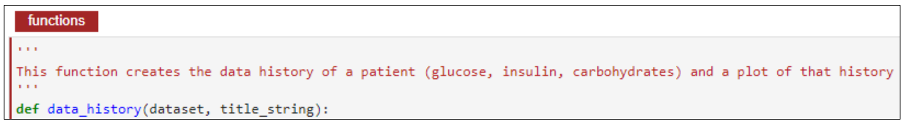
The Clark Error Grid is a useful tool for evaluating the accuracy of glucose monitoring systems and for identifying any issues or errors in monitoring. However, it does not provide information on the degree of glucose control over time or the patient's ability to manage their own glucose.

The Clark Error Grid divides the glucose graph into 5 zones, representing the combination of glucose values measured by the monitoring system and those measured in the laboratory. The zones represent the degree of discrepancy between the 2 measurements and the associated risk of clinical error.

The 5 zones are:

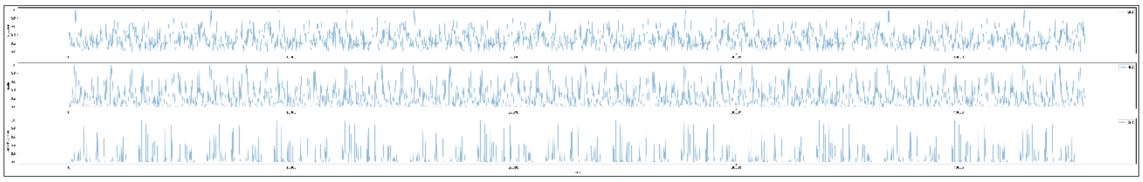
- **Zone A**: prediction values within 20% of the actual values
- **Zone B**: predictions outside 20% of the actual values but not leading to inappropriate treatments
- **Zone C**: predicted values leading to unnecessary treatments
- **Zone D**: prediction points indicating a potential dangerous failure to detect hypoglycaemia or hyperglycaemia
- **Zone E**: prediction values that would confuse the treatment of hypoglycaemia with hyperglycaemia and vice versa

3.3.2.3. `data_history`



```
'''
This function creates the data history of a patient (glucose, insulin, carbohydrates) and a plot of that history
'''
def data_history(dataset, title_string):
```

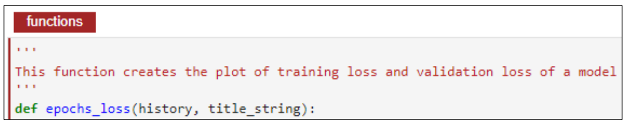
This function creates the data history of a patient based on the input dataset. The obtained data history is represented by a plot which has time on the x-axis and glucose, insulin and carbohydrates on the y-axis.



The function takes in input:

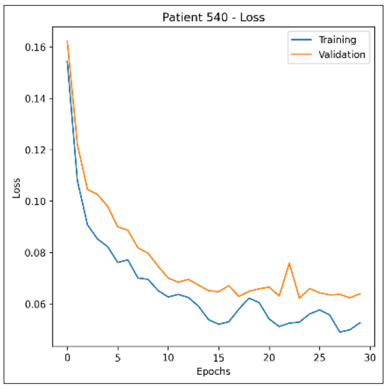
- `dataset`: patient's data
- `title_string`: string containing the title of the plot

3.3.2.4. `epochs_loss`



```
'''
This function creates the plot of training loss and validation loss of a model
'''
def epochs_loss(history, title_string):
```

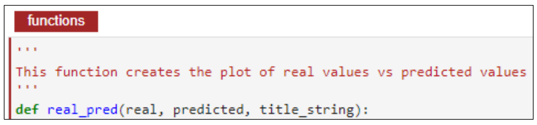
This function creates a plot which compares the training loss and the validation loss of a model. In particular, the plot has the number of epochs on the x-axis and the training and validation loss on the y-axis.



The function takes in input:

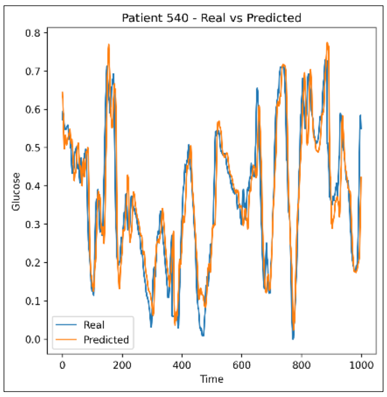
- `history`: output of fit method
- `title_string`: string containing the title of the plot

3.3.2.5. `real_pred`



```
'''
This function creates the plot of real values vs predicted values
'''
def real_pred(real, predicted, title_string):
```

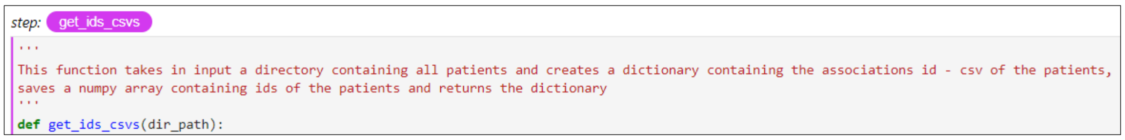
This function creates a plot which compares real values and predicted values. In particular, the plot has time on the x-axis and glucose on the y-axis.



The function takes in input:

- `real`: list containing real values
- `predicted`: list containing predicted values
- `title_string`: string containing the title of the plot

3.3.2.6. `get_ids_csvs`



step: get_ids_csvs

```
'''
This function takes in input a directory containing all patients and creates a dictionary containing the associations id - csv of the patients,
saves a numpy array containing ids of the patients and returns the dictionary
'''
def get_ids_csvs(dir_path):
```

This function is useful to extract the data of interest from the entire dataset. In particular, it creates a dictionary consisting of the associations ids-csvs of the patients and saves a NumPy array containing ids of the patients in '.npy' format.

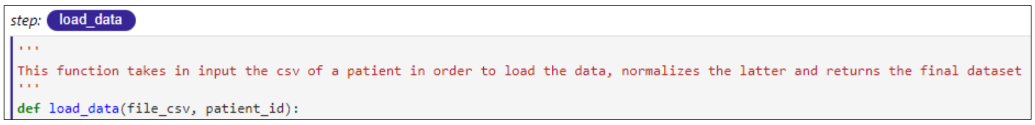
It takes in input:

- `dir_path`: path of the directory containing all patients

It returns:

- `ids_csvs`: dictionary consisting of associations ids-csvs of the patients

3.3.2.7. `load_data`



step: load_data

```
'''
This function takes in input the csv of a patient in order to load the data, normalizes the latter and returns the final dataset
'''
def load_data(file_csv, patient_id):
```

This function loads the data of a patient, normalizes them and provides the final dataset.

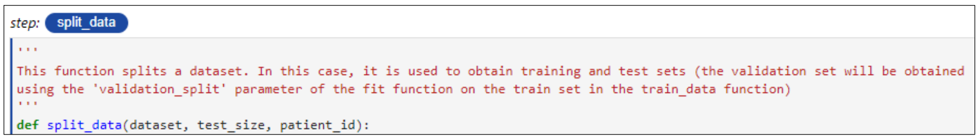
It takes in input:

- `file_csv`: csv of a patient
- `patient_id`: id of the patient

it returns:

- `final_dataset`: final dataset of the patient

3.3.2.8. `split_data`



step: split_data

```
'''
This function splits a dataset. In this case, it is used to obtain training and test sets (the validation set will be obtained
using the 'validation_split' parameter of the fit function on the train set in the train_data function)
'''
def split_data(dataset, test_size, patient_id):
```

This function splits a dataset in Training set and Test set. The Validation set will be obtained using the 'validation_split' parameter of the fit function on the Training set used in the train_data function. Furthermore, the function saves the data of the obtained Test set and the associated labels as NumPy arrays in '.npy' format because these files will be useful in Web App phase.

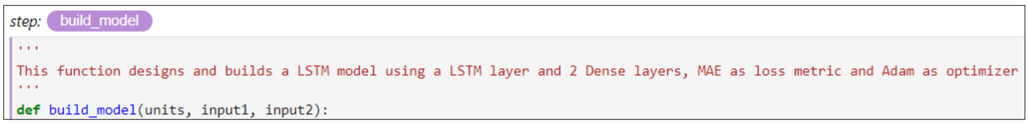
The function takes in input:

- `dataset`: data of the patient
- `test_size`: desired size of the Test set
- `patient_id`: id of the patient

It returns:

- `train_X`: data of the obtained Training set
- `train_y`: labels of the Training set
- `test_X`: data of the obtained Test set
- `test_y`: labels of the testing set

3.3.2.9. `build_model`



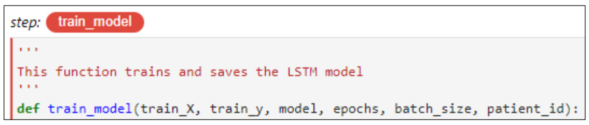
```
'''
This function designs and builds a LSTM model using a LSTM layer and 2 Dense layers, MAE as loss metric and Adam as optimizer
'''
def build_model(units, input1, input2):
```

This function designs and builds a **LSTM model**. The created model consists of a LSTM layer and 2 Dense layers. Therefore, Mean Absolute Error (MAE) was used as loss metric and Adam as optimizer.

The function takes in input:

- `units`: neurons number of the LSTM layer
- `input1`: timesteps number
- `input2`: features number

3.3.2.10. `train_model`



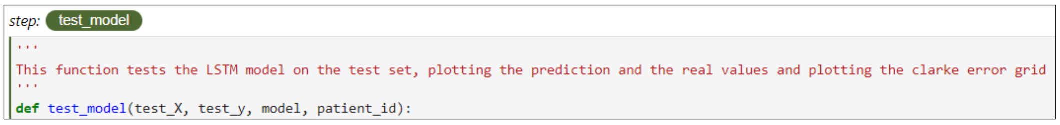
```
'''
This function trains and saves the LSTM model
'''
def train_model(train_X, train_y, model, epochs, batch_size, patient_id):
```

This function trains the LSTM model and saves it.

It takes in input:

- `train_X`: data of the obtained Training set
- `train_y`: labels of the Training set
- `model`: trained model to us for the training
- `epochs`: number of epochs for the training
- `batch_size`: batch size for the training
- `patient_id`: id of the patient

3.3.2.11. `test_model`



```
'''
This function tests the LSTM model on the test set, plotting the prediction and the real values and plotting the clarke error grid
'''
def test_model(test_X, test_y, model, patient_id):
```

This function tests the LSTM model on the Test set, plotting the comparison between the real values and the predicted ones. Furthermore, it plots the Clarke Error Grid.

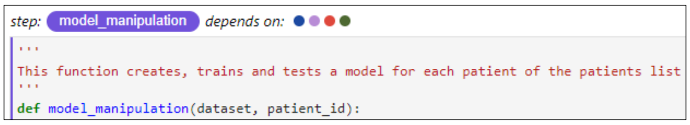
The function takes in input:

- `test_X`: data of the Test set
- `test_y`: labes of the Test set
- `model`: model used for the testing
- `patient_id`: id of the patient

It returns:

- `inv_y`: actual values
- `inv_yhat`: predicted values

3.3.2.12. model_manipulation


step: model_manipulation depends on: ● ● ● ●

```
'''
This function creates, trains and tests a model for each patient of the patients list
'''
def model_manipulation(dataset, patient_id):
```

This function creates, trins and tests a model for each patient of the patients list.

It takes in input:

- `dataset`: dataset of the patient of interest
- `patient_id`: id of the patient

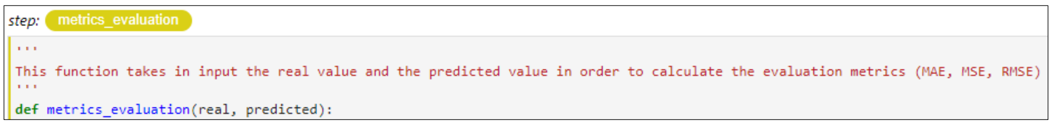
It returns:

- `real`: actual values
- `predicted`: predicted values

This component has the following dependencies:

- `split_data`
- `build_model`
- `train_model`
- `test_model`

3.3.2.13. metrics_evaluation


step: metrics_evaluation

```
'''
This function takes in input the real value and the predicted value in order to calculate the evaluation metrics (MAE, MSE, RMSE)
'''
def metrics_evaluation(real, predicted):
```

This function calculates the evaluation metrics MAE, MSE and RMSE. In particular:

- **MAE**: Mean Absolute Error $\boxed{MAE = \sum_{i=1}^{n} \frac{|\hat{y}_i - y_i|}{n}}$
- **MSE**: Mean Squared Error $\boxed{MSE = \sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$
- **RMSE**: Root Mean Squared Error $\boxed{RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}}$

Note that this function permits to calculate 3 types of evaluation metrics but in the next steps **only MAE will be used**.

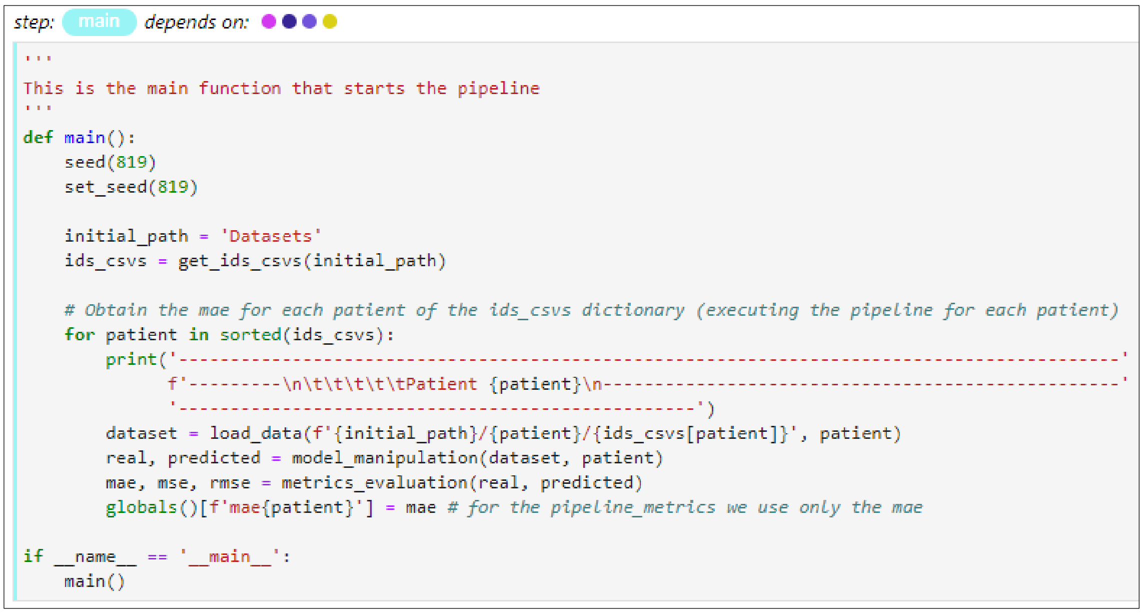
It takes in input:

- `real`: actual values
- `predicted`: predicted values

It returns:

- `mae`: calculated Mean Squared Error
- `mse`: calculated Mean Absolute Error
- `rmse`: calculated Root Meand Squared Error

3.3.2.14. `main`



```python
'''
This is the main function that starts the pipeline
'''
def main():
    seed(819)
    set_seed(819)

    initial_path = 'Datasets'
    ids_csvs = get_ids_csvs(initial_path)

    # Obtain the mae for each patient of the ids_csvs dictionary (executing the pipeline for each patient)
    for patient in sorted(ids_csvs):
        print('--------------------------------------------------------------------------------------------'
              f'---------\n\t\t\t\t\tPatient {patient}\n---------------------------------------------------'
              '--------------------------------------------------')
        dataset = load_data(f'{initial_path}/{patient}/{ids_csvs[patient]}', patient)
        real, predicted = model_manipulation(dataset, patient)
        mae, mse, rmse = metrics_evaluation(real, predicted)
        globals()[f'mae{patient}'] = mae # for the pipeline_metrics we use only the mae

if __name__ == '__main__':
    main()
```

This function is the general function that calls the other ones to generate a trained model for each patient of the patients list and to calculate the evaluation metrics.

This component has the following dependencies:

- `get_ids_csvs`
- `load_data`
- `model_manipulation`
- `metrics_evaluation`

3.3.3. Graph

The Pipeline is a **Directed Acyclic Graph (DAG)**. In particular, each node of the graph is a component and a Docker container.

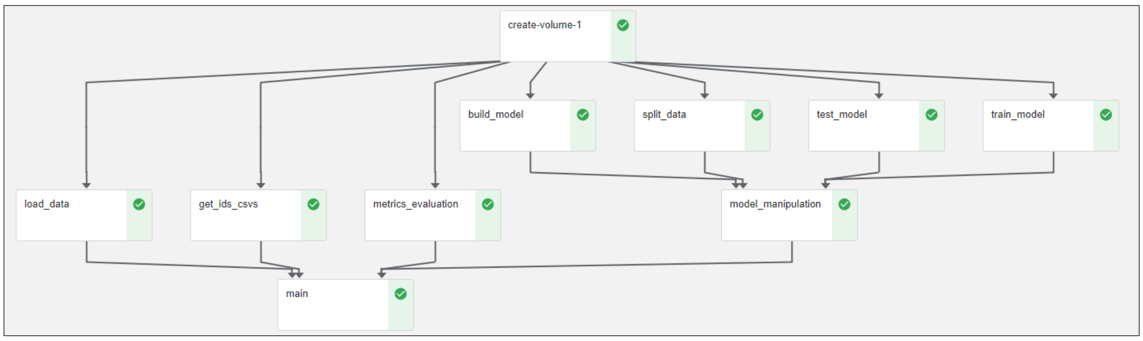
The Pipeline blocks in Kubeflow are the functional components that make up the Machine Learning workflow. Each block performs a specific operation.

The Pipeline blocks are executed as Docker containers on Kubernetes. This means that each Pipeline block is packaged in a Docker image that contains the application, dependency libraries, configuration data, and so on.

The fact that Pipeline blocks are executed as Docker containers on Kubernetes offers numerous advantages. Firstly, it permits the isolation of the execution environment of each block, ensuring that the dependencies and libraries used by one block do not interfere with those of other blocks. Secondly, it enables the entire Pipeline process to be reproducible, as each block is executed in a uniform and predictable environment.

In conclusion, the use of Docker containers on Kubernetes makes Kubeflow Pipeline blocks isolated, reproducible, and scalable, facilitating the management of the entire Machine Learning workflow.

Below it is shown the obtained Pipeline.



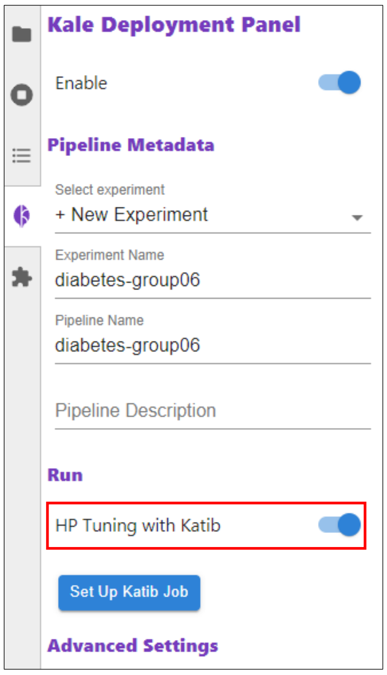
3.3.4. Katib

Katib is a Kubeflow component which allows for the **Automated Hyperparameters Tuning**. In a previous section Kale was introduced as a tool to define a Pipeline and it was said that it is necessary to enable the Kale Deployment Panel.

To enable Katib it is necessary to enable the **HP Tuning with Katib** too.

Furthermore, the following cells must be added in the Jupyter Notebook:

- **Pipeline-parameters**: it contains the editable parameters. This cell is located after the Imports cell within the Notebook
- **Pipeline-metrics**: it contains the metrics which have to be optimized by varying the parameters contained in the Pipeline-parameters cell

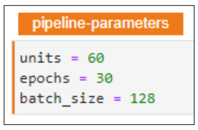


Select experiment
+ New Experiment

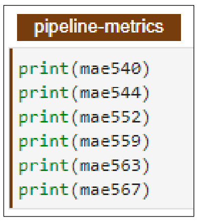
Experiment Name
diabetes-group06

Pipeline Name
diabetes-group06

Pipeline Description



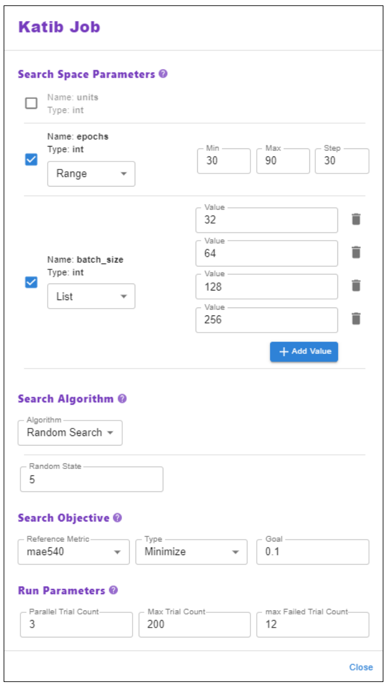
```
units = 60
epochs = 30
batch_size = 128
```



```
print(mae540)
print(mae544)
print(mae552)
print(mae559)
print(mae563)
print(mae567)
```

To configure Katib it is possible to click on **Set Up Katib Job**.

Below it is explained the utility of each section in Katib Job configuration.



Katib Job

Search Space Parameters

☐ Name: units
Type: int

☑ Name: **epochs**
Type: **int**
Range
Min 30 | Max 90 | Step 30

☑ Name: **batch_size**
Type: **int**
List
Value 32
Value 64
Value 128
Value 256

Search Algorithm

Algorithm: Random Search

Random State: 5

Search Objective

Reference Metric: mae540 | Type: Minimize | Goal: 0.1

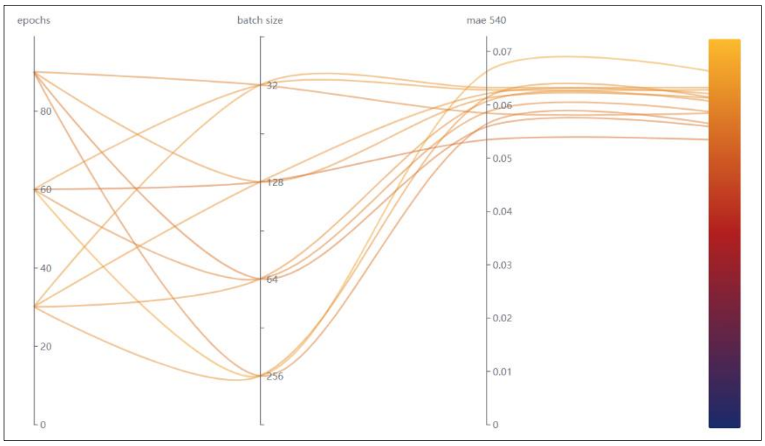
Run Parameters

Parallel Trial Count: 3 | Max Trial Count: 200 | max Failed Trial Count: 12

- **Search Space Parameters**: it permits to specify the parameters which can be edited during the execution. It is possible to choose specific values or a range. In the figure it is possible to see that the number of epochs vary from 30 to 90 with a step of 30 and the values for the batch size are 32, 64, 128, 256. Furthermore, because of the excessive quantity of time required units was excluded
- **Search Algorithm**: it permits to specify the algorithm to optimize the metrics. As it is possible to see in the figure, Random Search was selected
- **Search Objective**: it permits to specify a metric to optimize and the optimization way (minimization or maximization). Only one metric at a time can be optimized. As it is possible to see in the figure, MAE for a patient at a time was selected
- **Run Parameters**: it permits to specify the number of parallel trials, the maximum number of trials and the maximum number of failed trials. Exceeding the second or the third one the execution ends. As it is possible to see in the figure, the number of parallel trials was set to 3, the maximum number of trials was set to 200 and the maximum number of failed trials was set to 12

Finally, to run the Pipeline using the Automated Hyperparameters Tuning it is possible to click on **Compile and Run Katib Job**.

Below it is shown the final result.



As it is possible to see in the figure, the best values configuration for epochs and batch size is:

- **Epochs**: 60
- **Batch size**: 128

This configuration permits to obtain a MAE for patient with id 540 of approximately equal to 0.05.

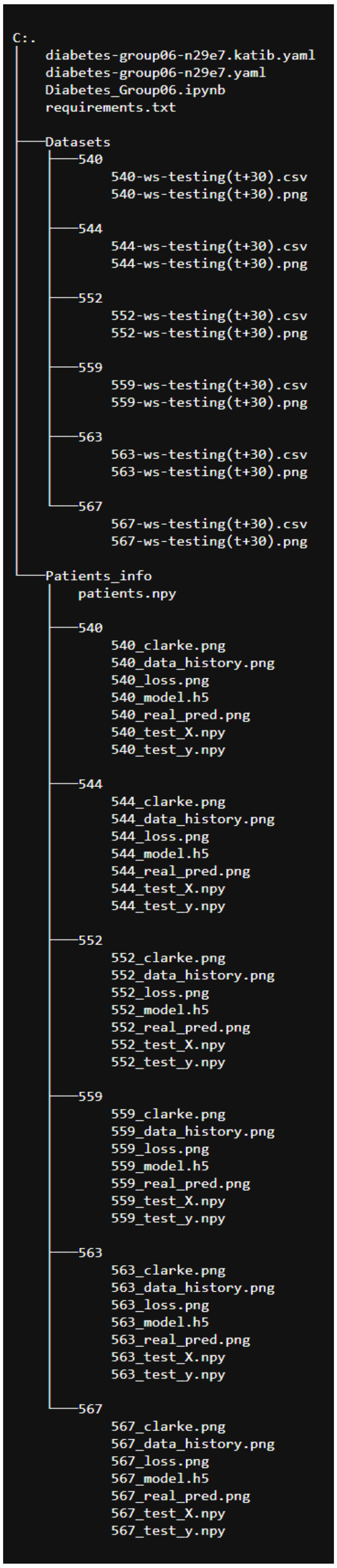
Doing the same thing for each patient, it was obtained that MAE assumes similar values for all patients.

3.4. Further information

For the correct functioning of the Pipeline and so to create a model for a patient, it is necessary to have the corresponding dataset. In particular, in datasets directory there has to be a subdirectory named with the patient id containing a csv file named ‘patient_id-ws-testing(t+30).csv’.

Note that each created file is saved in ‘Patients_info/patient_id’ in order to simplify the future loadings.

The **‘pipeline’ directory** of the project is characterized by the structure represented alongside.



```
C:.
│   diabetes-group06-n29e7.katib.yaml
│   diabetes-group06-n29e7.yaml
│   Diabetes_Group06.ipynb
│   requirements.txt
│
├───Datasets
│   ├───540
│   │       540-ws-testing(t+30).csv
│   │       540-ws-testing(t+30).png
│   │
│   ├───544
│   │       544-ws-testing(t+30).csv
│   │       544-ws-testing(t+30).png
│   │
│   ├───552
│   │       552-ws-testing(t+30).csv
│   │       552-ws-testing(t+30).png
│   │
│   ├───559
│   │       559-ws-testing(t+30).csv
│   │       559-ws-testing(t+30).png
│   │
│   ├───563
│   │       563-ws-testing(t+30).csv
│   │       563-ws-testing(t+30).png
│   │
│   └───567
│           567-ws-testing(t+30).csv
│           567-ws-testing(t+30).png
│
└───Patients_info
    │   patients.npy
    │
    ├───540
    │       540_clarke.png
    │       540_data_history.png
    │       540_loss.png
    │       540_model.h5
    │       540_real_pred.png
    │       540_test_X.npy
    │       540_test_y.npy
    │
    ├───544
    │       544_clarke.png
    │       544_data_history.png
    │       544_loss.png
    │       544_model.h5
    │       544_real_pred.png
    │       544_test_X.npy
    │       544_test_y.npy
    │
    ├───552
    │       552_clarke.png
    │       552_data_history.png
    │       552_loss.png
    │       552_model.h5
    │       552_real_pred.png
    │       552_test_X.npy
    │       552_test_y.npy
    │
    ├───559
    │       559_clarke.png
    │       559_data_history.png
    │       559_loss.png
    │       559_model.h5
    │       559_real_pred.png
    │       559_test_X.npy
    │       559_test_y.npy
    │
    ├───563
    │       563_clarke.png
    │       563_data_history.png
    │       563_loss.png
    │       563_model.h5
    │       563_real_pred.png
    │       563_test_X.npy
    │       563_test_y.npy
    │
    └───567
            567_clarke.png
            567_data_history.png
            567_loss.png
            567_model.h5
            567_real_pred.png
            567_test_X.npy
            567_test_y.npy
```

4. Web App

This section aims to present all steps that led to the realization of the Web App able to perform the prevision.

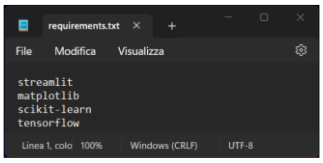
4.1. Creation

This section describes the process of creation and testing of the Web App.

In particular, the open-source Python library **Streamlit** was used.

4.1.1. Streamlit

After writing the code, to launch the App it is necessary to place in the webapp directory, install the requirements present in the provided **'requirements.txt'** file by running the following command:

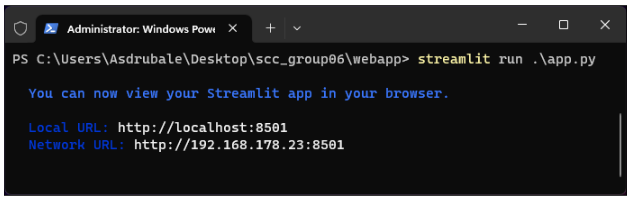


```
streamlit
matplotlib
scikit-learn
tensorflow
```

```
pip install -r requirements.txt
```

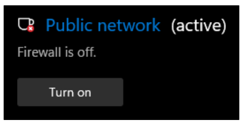
At this point it is possible to run the **'app.py'** file by using the following command:

```
streamlit run app.py
```



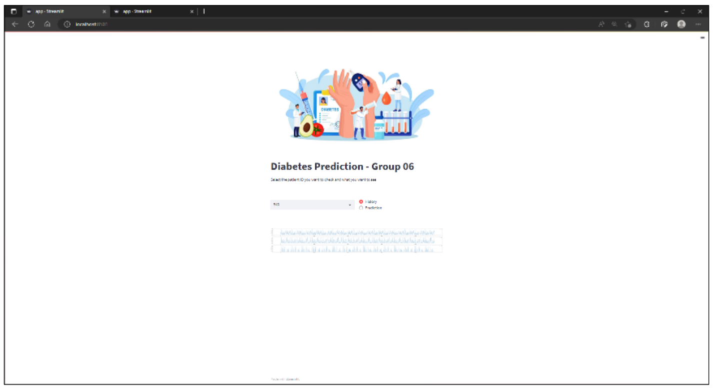
```
PS C:\Users\Asdrubale\Desktop\scc_group06\webapp> streamlit run .\app.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.178.23:8501
```

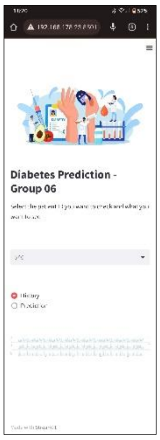
After running this command any device present in the same network can access the app using the second URL which represents the IP Address of the machine which the command is run on. To do this, if the command above is run on Windows it is necessary to **deactivate the Firewall**.

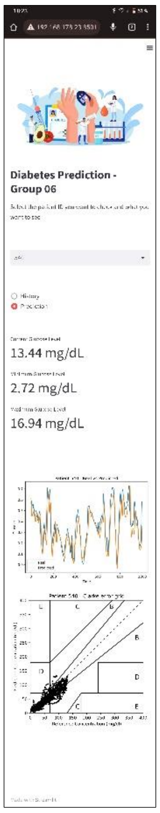


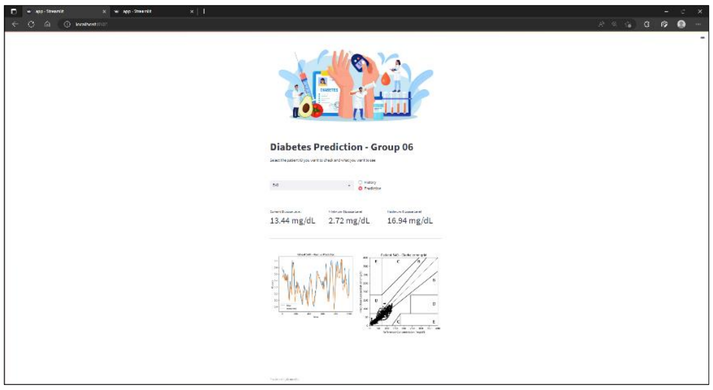
Public network (active)
Firewall is off.

The Web App visualizations on a computer and on a smartphone are the following ones:

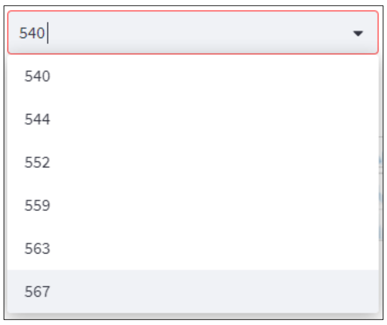


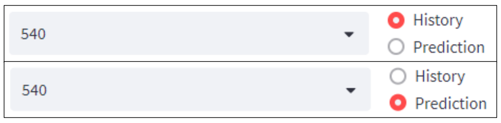






Thanks to the selection box it is possible to select a patient. Selectable patients are those of which it is known a dataset. The list of patients is obtained by loading the **'patients.npy'** file previously created by the Pipeline.

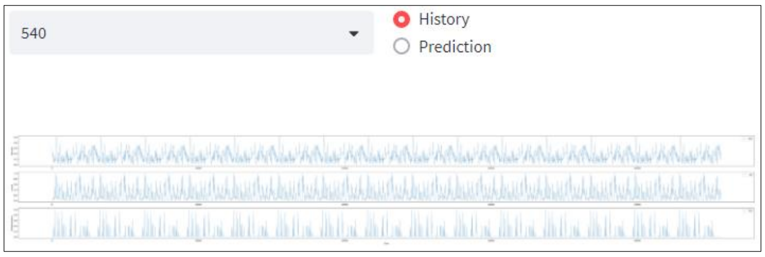




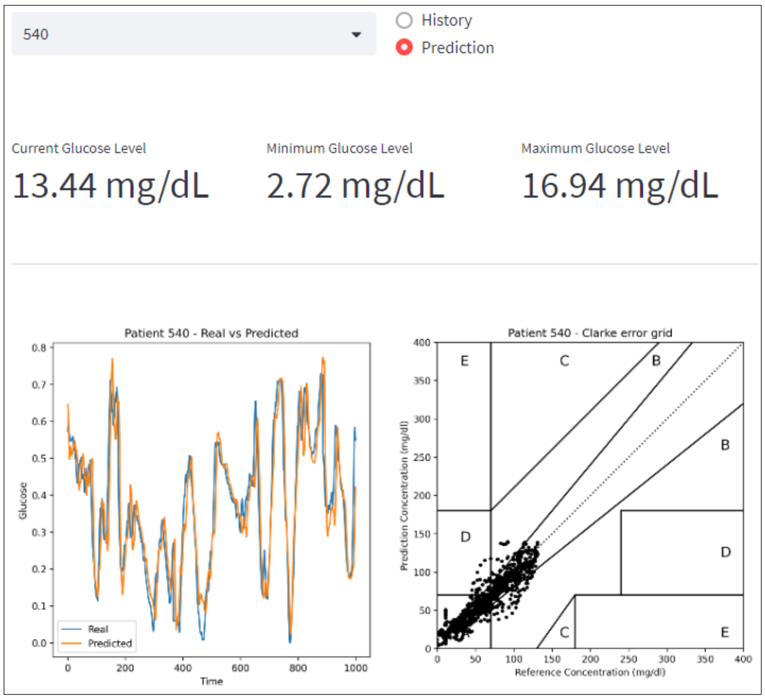
Once selected a patient it is possible to specify what to show thanks to a radio button.

In particular, the 2 possibilities are:

- **History**: the patient data trend over time is shown. Data are represented by glucose, insulin and carbohydrates values.
  The image containing the plot is obtained by loading the **'data_history.png'** image previously created by the Pipeline



- **Prediction**: current, minimum and maximum glucose predicted values are calculated and shown after reading the patient csv.
  In addition to these, there are 2 plots which represent the comparison between real and predicted values and the Clark Error Grid.
  To do this the appropriate patient **'test_X.npy'**, **'test_y.npy'** and model previously created by the Pipeline are loaded. Then the **'test_model.h5'** function makes the prediction and saves the plots.


540
History [✓ Prediction]
Patient 540 - Real vs Predicted
Glucose
Time
Real
Predicted
Patient 540 - Clarke error grid
Prediction Concentration (mg/dl)
Reference Concentration (mg/dl)
A
B
C
D
E

Current Glucose Level
13.44 mg/dL

Minimum Glucose Level
2.72 mg/dL

Maximum Glucose Level
16.94 mg/dL

4.2. Deployment

This section describes the Web App deployment process.

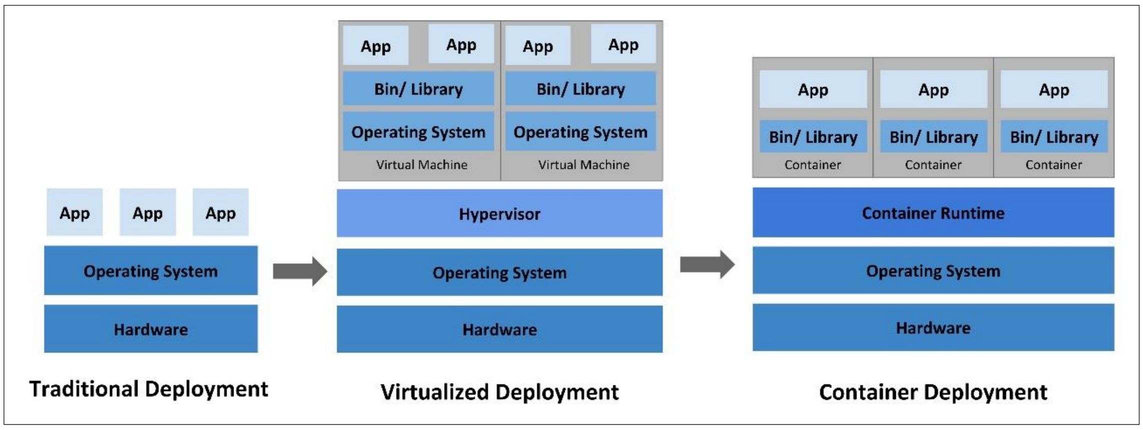
Deployment is the process of transferring a web application from the development server to the production server, so that it can be accessed by its end users on the Internet. In other words, deployment is the process of 'putting into production' the web application, so that it can be run on a server and made available to its users. Deployment is a critical part of the web application lifecycle, as an incorrect deployment can cause security, performance, reliability, and availability issues for the application.

In particular, **Docker** and **Kubernetes** were used.

4.2.1. Docker

Docker is a containerization technology that allows developers to package an application, configurations and all its dependencies into a single container. Using Docker simplifies the Web App deployment process. In particular, it offers several advantages:

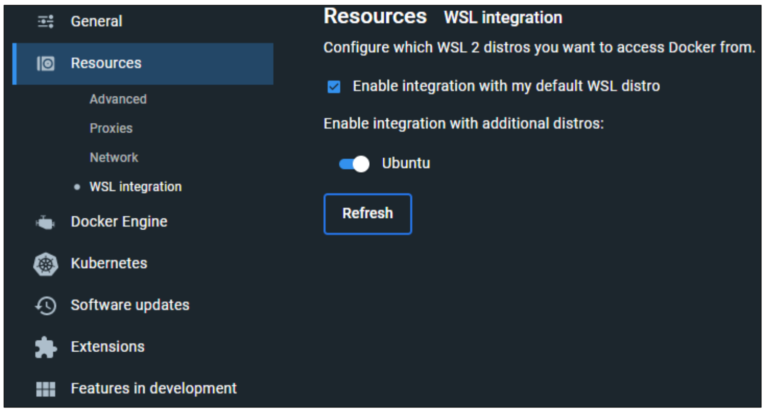
- **Portability**: Docker containers can be run on any system that supports Docker, regardless of the underlying operating system. This means that you can develop and test your app on one machine, and then deploy it on another without worrying about compatibility issues
- **Consistency**: Docker ensures that your app is deployed with the same configuration and dependencies every time, which reduces the risk of issues caused by differences in environments
- **Scalability**: Docker makes it easy to scale your app horizontally by spinning up additional containers as needed. This allows you to handle spikes in traffic or scale your app to meet changing demands
- **Isolation**: Docker containers provide a level of isolation that helps prevent conflicts between applications or dependencies. This means that you can run multiple applications on the same server without worrying about them interfering with each other
- **Versioning**: Docker allows you to easily manage and deploy different versions of your application, making it easy to roll back to a previous version if needed
- **Efficiency**: Docker containers are lightweight and consume minimal resources, which allows you to run more applications on the same server than you could with traditional virtual machines



A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Docker Desktop was installed.

Since everything was tested both on Windows and WSL, it was necessary for the second one to **enable WSL integration**.



Resources WSL integration

Configure which WSL 2 distros you want to access Docker from.

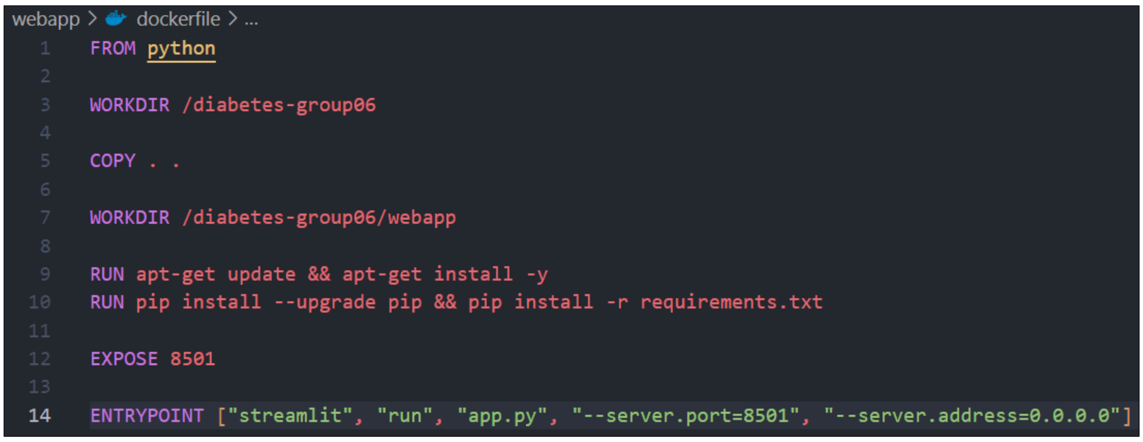
☑ Enable integration with my default WSL distro

Enable integration with additional distros:

Ubuntu

To create a Docker image it was necessary to create a **dockerfile** in the Web App folder.

A dockerfile is a text file that contains a series of instructions that Docker uses to build a Docker image. The dockerfile provides a recipe for building a container image that includes all the necessary dependencies, configuration, and code needed to run a specific application



webapp > dockerfile > ...

```
FROM python

WORKDIR /diabetes-group06

COPY . .

WORKDIR /diabetes-group06/webapp

RUN apt-get update && apt-get install -y
RUN pip install --upgrade pip && pip install -r requirements.txt

EXPOSE 8501

ENTRYPOINT ["streamlit", "run", "app.py", "--server.port=8501", "--server.address=0.0.0.0"]
```

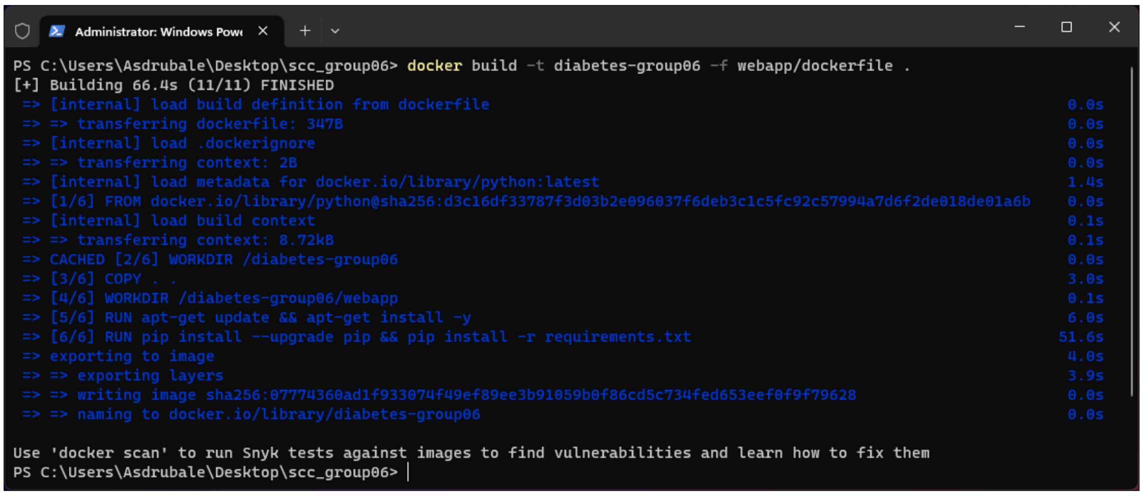
Below it is described the functionality of each part of the dockerfile:

- **FROM**: it specifies the base image to use for this Docker image. In this case, the base image is Python, which means that Python is already installed in the Docker image
- **WORKDIR**: it sets the working directory for the subsequent instructions. In this case the first one sets it to '/diabetes-group06', while the second one sets the working directory to a subdirectory ('/webapp') of that set before
- **COPY**: it copies all the files and directories from the current directory (the directory where the Docker build command is run) to the working directory in the Docker image ('/diabetes-group06')
- **RUN**: it runs a Linux command. In this case the firs one updates the package manager and installs any necessary packages, while the second one upgrades pip (the package installer for Python) and then installs the dependencies specified in the 'requirements.txt' file

- **EXPOSE**: it exposes ports informing Docker that the container will listen on the specified network port(s) at runtime. In this case it exposes the port 8501 that is the Streamlit default port
- **ENTRYPOINT**: it specifies the command to run when the Docker container is started. In this case, it runs the 'streamlit' command to start the application with the 'app.py' file as the main entry point. It also specifies the port to use (8501) and the server address to use (0.0.0.0).
  Note that **'--server-address=0.0.0.0'** is used to allow the server inside the Docker container to listen on all available network interfaces, making it accessible from outside the container. Thanks to it, in this case, when the docker image is run the available URLs will become the same provided by Streamlit

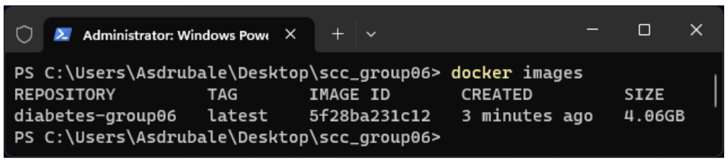
To create the **Web App Docker image**, it is necessary to place in the root directory ('scc_group06') and run the following command:

```
docker build -t diabetes-group06 -f webapp/dockerfile .
```

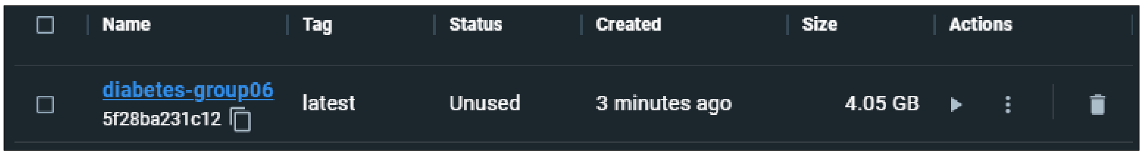


```
PS C:\Users\Asdrubale\Desktop\scc_group06> docker build -t diabetes-group06 -f webapp/dockerfile .
[+] Building 66.4s (11/11) FINISHED
 => [internal] load build definition from dockerfile                                                   0.0s
 => => transferring dockerfile: 347B                                                                   0.0s
 => [internal] load .dockerignore                                                                      0.0s
 => => transferring context: 2B                                                                        0.0s
 => [internal] load metadata for docker.io/library/python:latest                                       1.4s
 => [1/6] FROM docker.io/library/python@sha256:d3c16df33787f3d03b2e096037f6deb3c1c5fc92c57994a7d6f2de018de01a6b  0.0s
 => [internal] load build context                                                                      0.1s
 => => transferring context: 8.72kB                                                                    0.1s
 => CACHED [2/6] WORKDIR /diabetes-group06                                                             0.0s
 => [3/6] COPY . .                                                                                     3.0s
 => [4/6] WORKDIR /diabetes-group06/webapp                                                             0.1s
 => [5/6] RUN apt-get update && apt-get install -y                                                     6.0s
 => [6/6] RUN pip install --upgrade pip && pip install -r requirements.txt                            51.6s
 => exporting to image                                                                                 4.0s
 => => exporting layers                                                                                3.9s
 => => writing image sha256:07774360ad1f933074f49ef89ee3b91059b0f86cd5c734fed653eef0f9f79628           0.0s
 => => naming to docker.io/library/diabetes-group06                                                    0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
PS C:\Users\Asdrubale\Desktop\scc_group06> |
```

Then it is possible to see the built image whether running the `docker images` command or going to the Docker Desktop application in Images section.



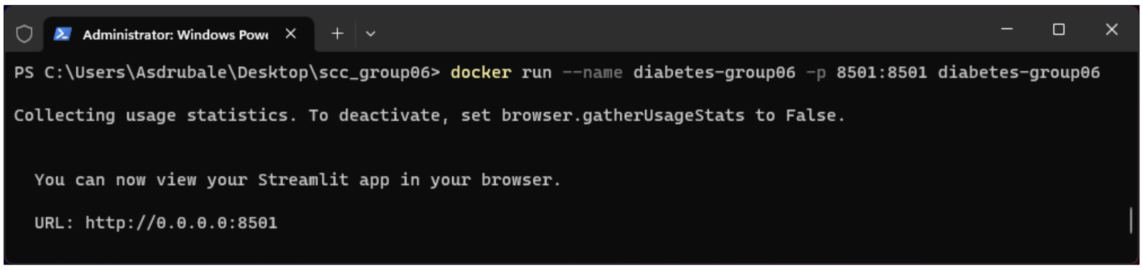
```
PS C:\Users\Asdrubale\Desktop\scc_group06> docker images
REPOSITORY         TAG       IMAGE ID       CREATED         SIZE
diabetes-group06   latest    5f28ba231c12   3 minutes ago   4.06GB
PS C:\Users\Asdrubale\Desktop\scc_group06>
```



	Name	Tag	Status	Created	Size	Actions
☐	diabetes-group06 5f28ba231c12	latest	Unused	3 minutes ago	4.05 GB	

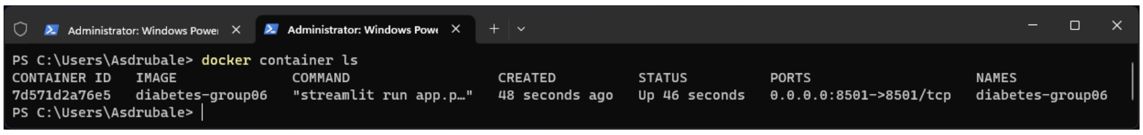
To run the Web App Docker image in a **Docker container** it is possible to use the following command:

```
docker run --name diabetes-group06 -p 8501:8501 diabetes-group06
```

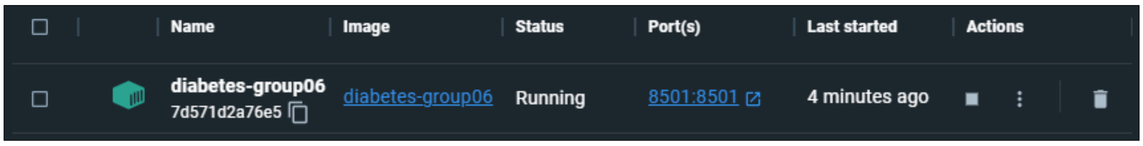


```
PS C:\Users\Asdrubale\Desktop\scc_group06> docker run --name diabetes-group06 -p 8501:8501 diabetes-group06

Collecting usage statistics. To deactivate, set browser.gatherUsageStats to False.


  You can now view your Streamlit app in your browser.

  URL: http://0.0.0.0:8501
```

Then it is possible to see the run container whether running the `docker container ls` command or going to the Docker Desktop application in Containers section.



```
PS C:\Users\Asdrubale> docker container ls
CONTAINER ID   IMAGE              COMMAND                  CREATED          STATUS          PORTS                    NAMES
7d571d2a76e5   diabetes-group06   "streamlit run app.p…"   48 seconds ago   Up 46 seconds   0.0.0.0:8501->8501/tcp   diabetes-group06
PS C:\Users\Asdrubale> |
```

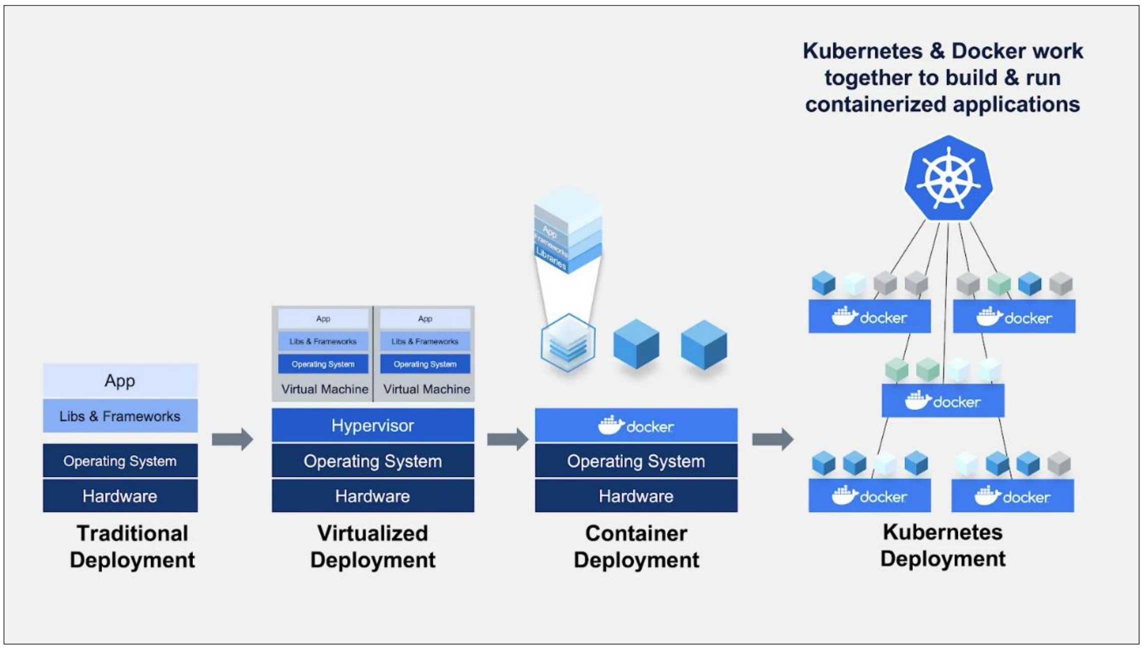


Name	Image	Status	Port(s)	Last started	Actions
diabetes-group06 7d571d2a76e5	diabetes-group06	Running	8501:8501	4 minutes ago	

4.2.2. Kubernetes

Kubernetes is a popular open-source container orchestration platform that provides several advantages for deploying web applications. Here are some of the benefits of using Kubernetes to deploy a web app:

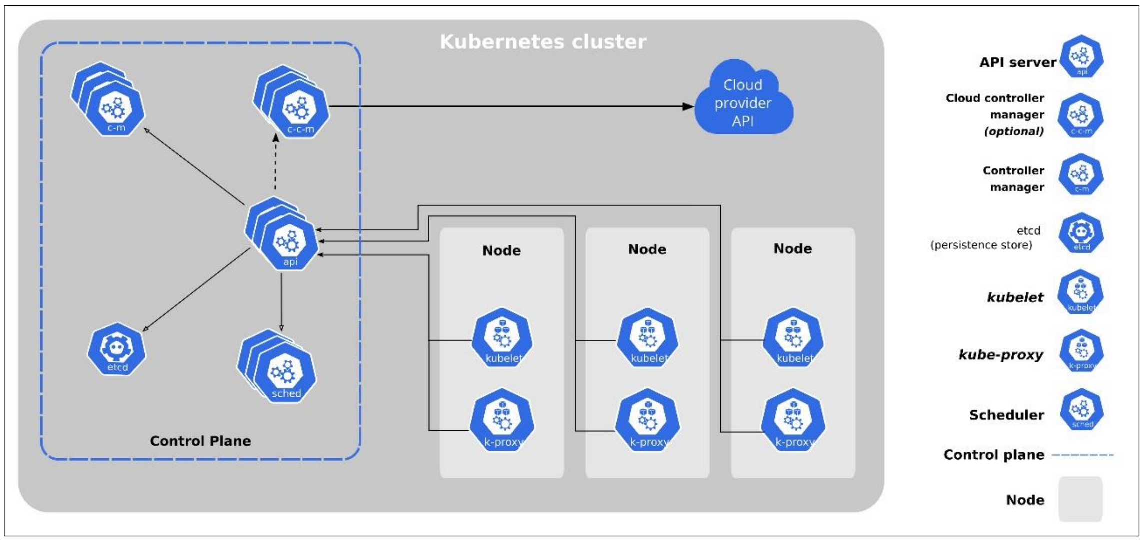
- **Scalability**: Kubernetes enables you to easily scale your web application by adding or removing containers as required. It can automatically adjust the number of containers based on traffic or resource utilization, ensuring that your application can handle high traffic loads
- **High availability**: Kubernetes provides automatic failover and rescheduling of containers in case of failures, ensuring that your web application is always available to users
- **Disaster recovery**: Kubernetes provides a process of restoring a system or application after a catastrophic event
- **Simplified deployment**: Kubernetes provides a declarative approach to deploying applications, making it easier to manage and deploy updates to your web application. It also supports rolling updates, allowing you to update your application without downtime
- **Portability**: Kubernetes provides a standardized way of deploying and managing containers, making it easy to move your web application across different environments, such as from development to production
- **Extensibility**: Kubernetes is highly extensible, with a large ecosystem of plugins and tools that can be used to extend its functionality



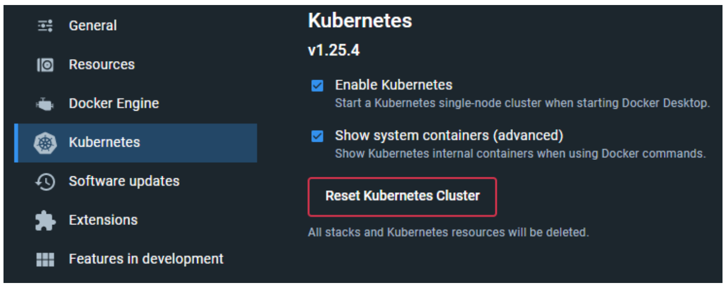
The **architecture** of Kubernetes can be broken down into several components:

- **Control Plane**: The control plane is the brain of Kubernetes, responsible for managing the entire system. It consists of several components that work together to orchestrate the deployment and management of containerized applications. The main components of the control plane are:
  - **API Server**: it is the central component of the control plane, which exposes the Kubernetes API for managing the system
  - **etcd**: it is a distributed key-value store used by Kubernetes to store all the configuration and state information of the system
  - **Scheduler**: it is responsible for scheduling containers to run on specific nodes based on their resource requirements and availability
  - **Controller Manager**: it is responsible for managing the lifecycle of Kubernetes objects, such as Pods, Services, and Replica Sets
- **Nodes**: Nodes are the worker machines in Kubernetes, responsible for running containerized applications. Each node runs a container runtime, such as Docker, to manage containers. The main components of a node are:
  - **kubelet**: it is an agent that runs on each node and communicates with the control plane to manage containers on the node
  - **kube-proxy**: it is responsible for managing network connectivity between containers on the same node and between nodes
- **Objects**: Kubernetes objects are the building blocks of the system, representing the desired state of a particular resource, such as a Pod, Service, or Replication Controller. Some of the key objects in Kubernetes are:
  - **Pod**: it is the smallest deployable unit in Kubernetes, representing one or more containers that run together on a single node
  - **Service**: it provides a stable IP address and DNS name for a set of Pods, enabling other components to discover and communicate with them
  - **Replica Set**: it ensures that a specified number of replicas of a Pod are running at all times

Note that while nodes are the physical machines where containers run, objects are the logical abstractions that represent different components of your application. Nodes are managed by the Kubernetes system, whereas objects are created and managed by users or applications. In other words, nodes are the infrastructure that supports objects, and objects are the applications that run on nodes.



To have a Kubernetes cluster running on a local machine it is necessary to **enable Kubernetes** on Docker Desktop.



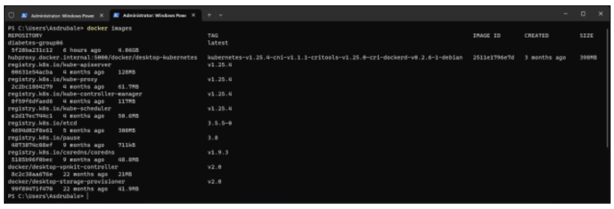
Kubernetes
v1.25.4

[✓] Enable Kubernetes
Start a Kubernetes single-node cluster when starting Docker Desktop.

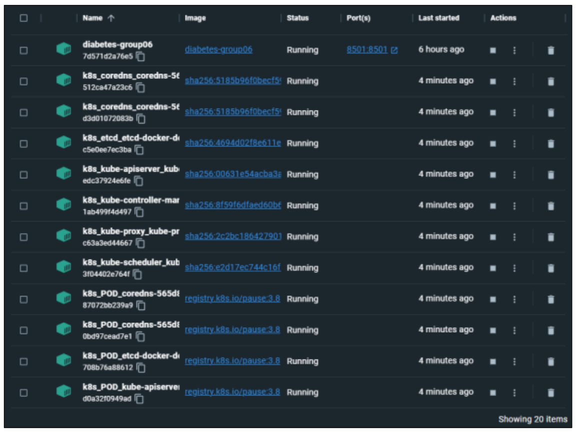
[✓] Show system containers (advanced)
Show Kubernetes internal containers when using Docker commands.

All stacks and Kubernetes resources will be deleted.

At this point all necessary Kubernetes images and containers are visible through the ways already seen in the previous section.

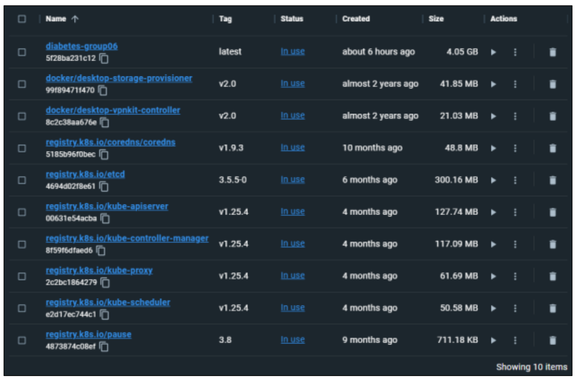


```
PS C:\Users\Asdrubale> docker images
REPOSITORY                                                 TAG                                                                    IMAGE ID       CREATED        SIZE
diabetes-group06                                           latest
 5f28ba231c12   6 hours ago    4.06GB
hubproxy.docker.internal:5000/docker/desktop-kubernetes    kubernetes-v1.25.4-cni-v1.1.1-critools-v1.25.0-cri-dockerd-v0.2.6-1-debian   2511e1796e7d   3 months ago   398MB
registry.k8s.io/kube-apiserver                             v1.25.4
 00631e54acba   4 months ago    128MB
registry.k8s.io/kube-proxy                                 v1.25.4
 2c2bc1864279   4 months ago    61.7MB
registry.k8s.io/kube-controller-manager                    v1.25.4
 8f59f6dfaed6   4 months ago    117MB
registry.k8s.io/kube-scheduler                             v1.25.4
 e2d17ec744c1   4 months ago    50.6MB
registry.k8s.io/etcd                                       3.5.5-0
 4694d02f8e61   5 months ago    300MB
registry.k8s.io/pause                                      3.8
 4873874c08ef   9 months ago    711kB
registry.k8s.io/coredns/coredns                            v1.9.3
 5185b96f0bec   9 months ago    48.8MB
docker/desktop-vpnkit-controller                           v2.0
 8c2c38aa676e   22 months ago   21MB
docker/desktop-storage-provisioner                         v2.0
 99f89471f470   22 months ago   41.9MB
PS C:\Users\Asdrubale>
```



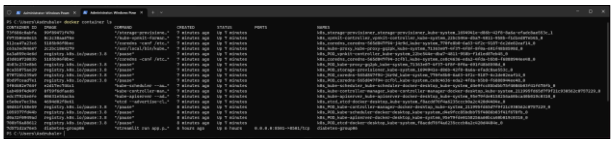
Name	Image	Status	Port(s)	Last started	Actions
diabetes-group06 7d571d2a76e5	diabetes-group06	Running	8501:8501	6 hours ago	
k8s_coredns_coredns-56 512ca47a23c6	sha256:5185b96f0becf5	Running		4 minutes ago	
k8s_coredns_coredns-56 d3d01072083b	sha256:5185b96f0becf5	Running		4 minutes ago	
k8s_etcd_etcd-docker-d c5e0ee7ec3ba	sha256:4694d02f8e611e	Running		4 minutes ago	
k8s_kube-apiserver_kub edc37924e6fe	sha256:00631e54acba3a	Running		4 minutes ago	
k8s_kube-controller-mar 1ab499f4d497	sha256:8f59f6dfaed60bf	Running		4 minutes ago	
k8s_kube-proxy_kube-pr c63a3ed44667	sha256:2c2bc186427901	Running		4 minutes ago	
k8s_kube-scheduler_kub 3f04402e764f	sha256:e2d17ec744c16f	Running		4 minutes ago	
k8s_POD_coredns-565d8 87072bb239a9	registry.k8s.io/pause:3.8	Running		4 minutes ago	
k8s_POD_coredns-565d8 0bd97cead7e1	registry.k8s.io/pause:3.8	Running		4 minutes ago	
k8s_POD_etcd-docker-d 708b76a88612	registry.k8s.io/pause:3.8	Running		4 minutes ago	
k8s_POD_kube-apiserver d0a32f0949ad	registry.k8s.io/pause:3.8	Running		4 minutes ago	

Showing 20 items



Name	Tag	Status	Created	Size	Actions
diabetes-group06 5f28ba231c12	latest	In use	about 6 hours ago	4.05 GB	
docker/desktop-storage-provisioner 99f89471f470	v2.0	In use	almost 2 years ago	41.85 MB	
docker/desktop-vpnkit-controller 8c2c38aa676e	v2.0	In use	almost 2 years ago	21.03 MB	
registry.k8s.io/coredns/coredns 5185b96f0bec	v1.9.3	In use	10 months ago	48.8 MB	
registry.k8s.io/etcd 4694d02f8e61	3.5.5-0	In use	6 months ago	300.16 MB	
registry.k8s.io/kube-apiserver 00631e54acba	v1.25.4	In use	4 months ago	127.74 MB	
registry.k8s.io/kube-controller-manager 8f59f6dfaed6	v1.25.4	In use	4 months ago	117.09 MB	
registry.k8s.io/kube-proxy 2c2bc1864279	v1.25.4	In use	4 months ago	61.69 MB	
registry.k8s.io/kube-scheduler e2d17ec744c1	v1.25.4	In use	4 months ago	50.58 MB	
registry.k8s.io/pause 4873874c08ef	3.8	In use	9 months ago	711.18 KB	

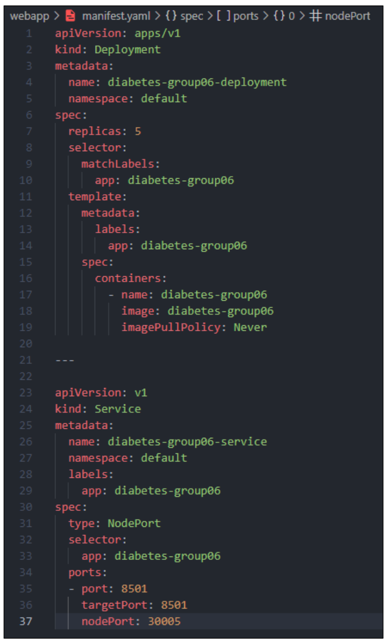
Showing 10 items



```
PS C:\Users\Asdrubale> docker container ls
CONTAINER ID   IMAGE   COMMAND   CREATED   STATUS   PORTS   NAMES
[illegible]
PS C:\Users\Asdrubale>
```

In Kubernetes, a **manifest file** is a YAML file that permits to configure the desired state of a Kubernetes object, such as a Pod, deployment, service, or replication controller. The manifest file specifies the configuration of the object, including its metadata, specifications, and optionally, any other relevant information. Manifest files are used by Kubernetes to deploy and manage applications in a declarative manner.

Alongside it is described the functionality of each part of the manifest file.


webapp > manifest.yaml > {} spec > [ ] ports > {} 0 > # nodePort

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: diabetes-group06-deployment
  namespace: default
spec:
  replicas: 5
  selector:
    matchLabels:
      app: diabetes-group06
  template:
    metadata:
      labels:
        app: diabetes-group06
    spec:
      containers:
        - name: diabetes-group06
          image: diabetes-group06
          imagePullPolicy: Never

---

apiVersion: v1
kind: Service
metadata:
  name: diabetes-group06-service
  namespace: default
  labels:
    app: diabetes-group06
spec:
  type: NodePort
  selector:
    app: diabetes-group06
  ports:
  - port: 8501
    targetPort: 8501
    nodePort: 30005
```

This file defines a Kubernetes deployment and service for a containerized application named 'diabetes-group06'.

The **deployment section** specifies the name of the deployment as 'diabetes-group06-deployment' and that it will run 5 replicas of the 'diabetes-group06' container image, using a selector to match the labels with the 'app: diabetes-group06'.

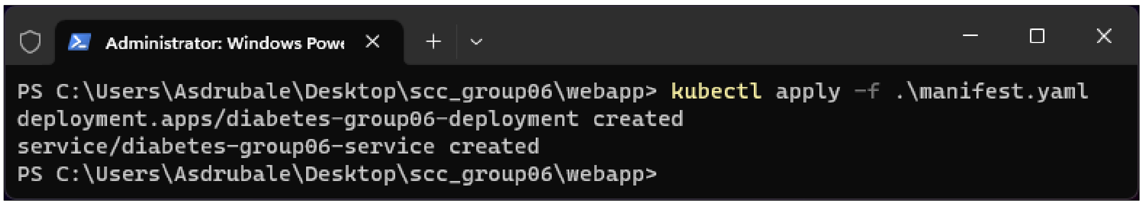
The **template subsection** specifies the configuration for the Pod that will be created for the deployment. It has a single container, also named 'diabetes-group06', with an imagePullPolicy set to 'Never', which means that the container image should be available on the node before the Pod starts running.

The **service section** creates a Kubernetes service that exposes the deployment to the external world. It specifies the name of the service as 'diabetes-group06-service' and sets the type to 'NodePort'. The selector is set to match the labels with the 'app: diabetes-group06', which means that it will route traffic to the Pods with the matching label. The ports section maps the container port 8501 to the target port 8501 and sets the nodePort to 30005.

Note that it was used 'NodePort' as type of the service because it is the easiest way to expose a service to external traffic, but it's not as scalable or secure as the other types of service exposure (Load Balancer or ClusterIP with an Ingress controller).

To deploy the manifest file it is necessary to place in the webapp directory and run the following command:

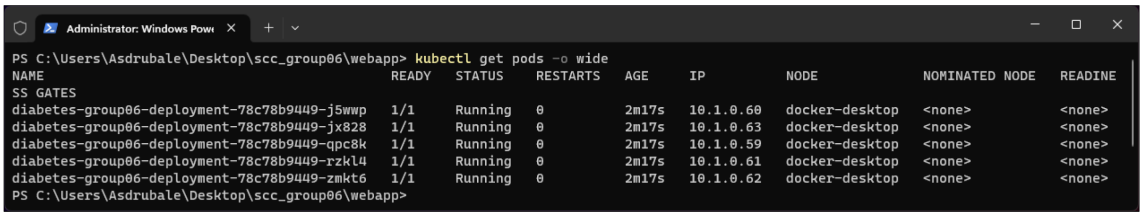
```
kubectl apply -f manifest.yaml
```



```
PS C:\Users\Asdrubale\Desktop\scc_group06\webapp> kubectl apply -f .\manifest.yaml
deployment.apps/diabetes-group06-deployment created
service/diabetes-group06-service created
PS C:\Users\Asdrubale\Desktop\scc_group06\webapp>
```

To check the created Pods, its IP addresses and the node on which they are running it is possible to run the following command:

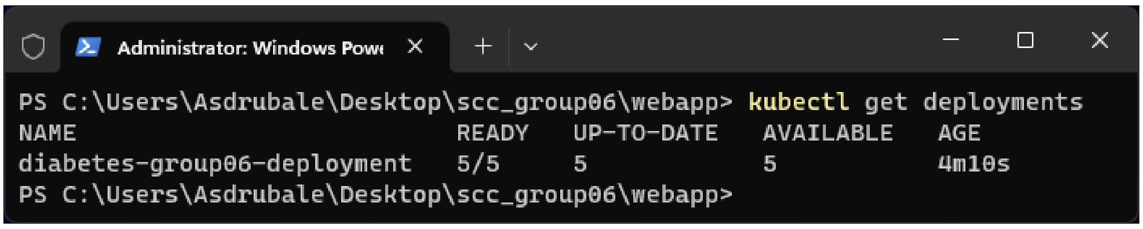
```
kubectl get Pods -o wide
```



```
PS C:\Users\Asdrubale\Desktop\scc_group06\webapp> kubectl get pods -o wide
NAME                                           READY   STATUS    RESTARTS   AGE     IP          NODE             NOMINATED NODE   READINE
SS GATES
diabetes-group06-deployment-78c78b9449-j5wwp   1/1     Running   0          2m17s   10.1.0.60   docker-desktop   <none>           <none>
diabetes-group06-deployment-78c78b9449-jx828   1/1     Running   0          2m17s   10.1.0.63   docker-desktop   <none>           <none>
diabetes-group06-deployment-78c78b9449-qpc8k   1/1     Running   0          2m17s   10.1.0.59   docker-desktop   <none>           <none>
diabetes-group06-deployment-78c78b9449-rzkl4   1/1     Running   0          2m17s   10.1.0.61   docker-desktop   <none>           <none>
diabetes-group06-deployment-78c78b9449-zmkt6   1/1     Running   0          2m17s   10.1.0.62   docker-desktop   <none>           <none>
PS C:\Users\Asdrubale\Desktop\scc_group06\webapp>
```

To check the running deployments it is possible to run the following command:

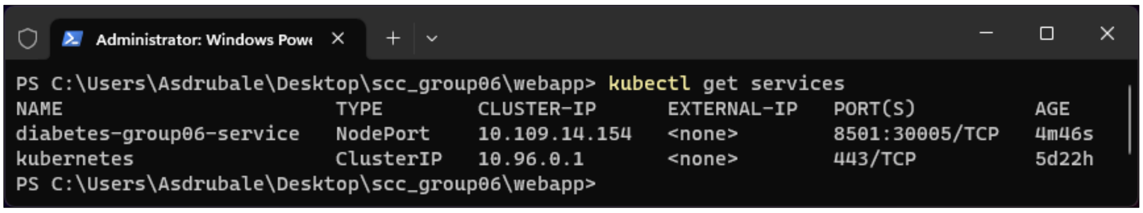
```
kubectl get deployments
```



```
PS C:\Users\Asdrubale\Desktop\scc_group06\webapp> kubectl get deployments
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
diabetes-group06-deployment    5/5     5            5           4m10s
PS C:\Users\Asdrubale\Desktop\scc_group06\webapp>
```

To check the running services it is possible to run the following command:

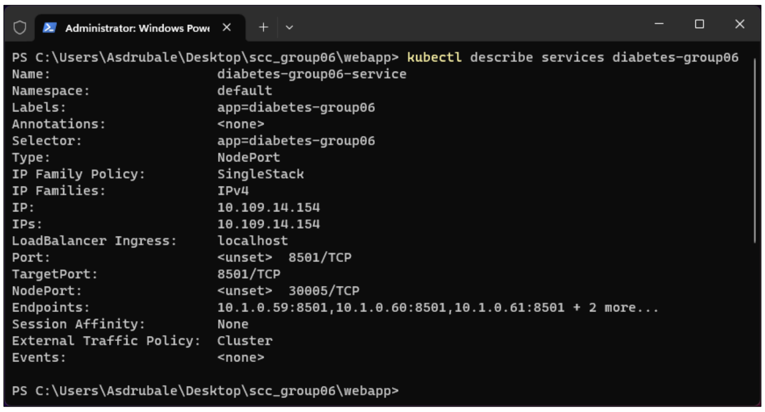
```
kubectl get services
```



```
PS C:\Users\Asdrubale\Desktop\scc_group06\webapp> kubectl get services
NAME                       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
diabetes-group06-service   NodePort    10.109.14.154   <none>        8501:30005/TCP   4m46s
kubernetes                 ClusterIP   10.96.0.1       <none>        443/TCP          5d22h
PS C:\Users\Asdrubale\Desktop\scc_group06\webapp>
```

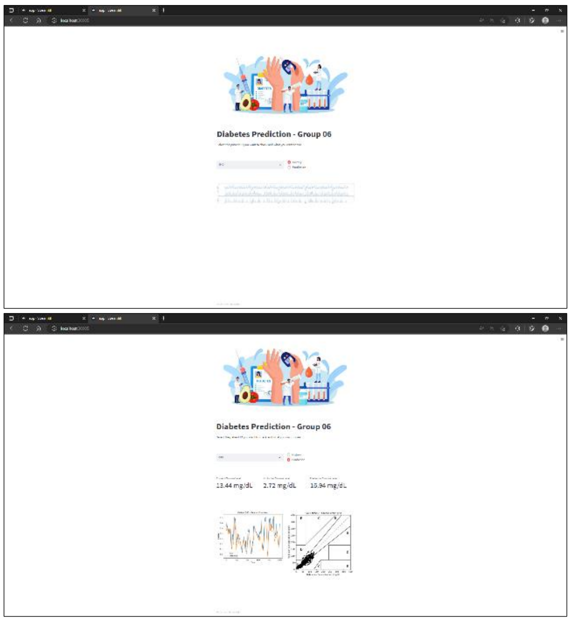
To get more information about a service it is possible to run the following command:

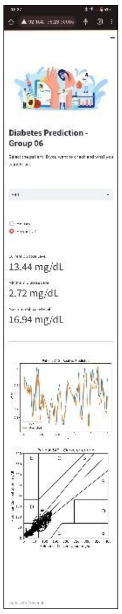
```
kubectl describe services diabetes-group06-service
```



```
PS C:\Users\Asdrubale\Desktop\scc_group06\webapp> kubectl describe services diabetes-group06
Name:                     diabetes-group06-service
Namespace:                default
Labels:                   app=diabetes-group06
Annotations:              <none>
Selector:                 app=diabetes-group06
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.109.14.154
IPs:                      10.109.14.154
LoadBalancer Ingress:     localhost
Port:                     <unset>  8501/TCP
TargetPort:               8501/TCP
NodePort:                 <unset>  30005/TCP
Endpoints:                10.1.0.59:8501,10.1.0.60:8501,10.1.0.61:8501 + 2 more...
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>

PS C:\Users\Asdrubale\Desktop\scc_group06\webapp>
```

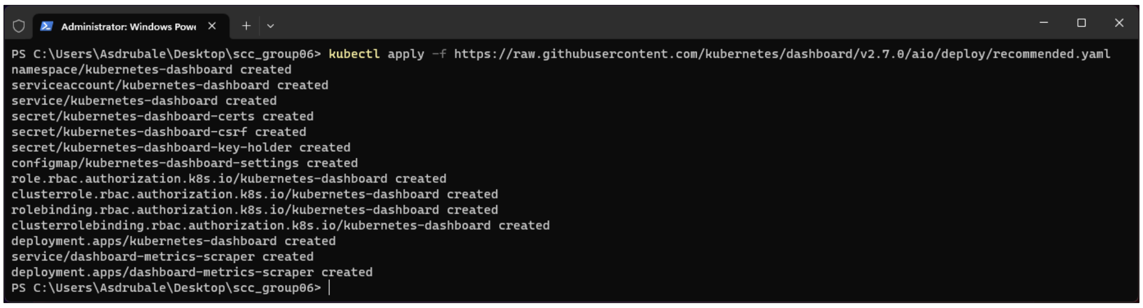
To visualize the local deployed Web App it is possible to use the same URLs provided by Streamlit and also used by Docker with the difference that nodePort has to be used and it is 30005.


Diabetes Prediction - Group 06
Diabetes Prediction - Group 06
13.44 mg/dL
2.72 mg/dL
16.94 mg/dL



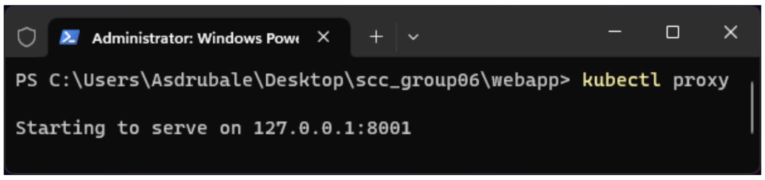
To visualize the **Kubernetes Web Dashboard** it is necessary to run the following commands:

- `kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/recommended.yaml`



```
PS C:\Users\Asdrubale\Desktop\scc_group06> kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/aio/deploy/recommended.yaml
namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf created
secret/kubernetes-dashboard-key-holder created
configmap/kubernetes-dashboard-settings created
role.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
deployment.apps/kubernetes-dashboard created
service/dashboard-metrics-scraper created
deployment.apps/dashboard-metrics-scraper created
PS C:\Users\Asdrubale\Desktop\scc_group06>
```

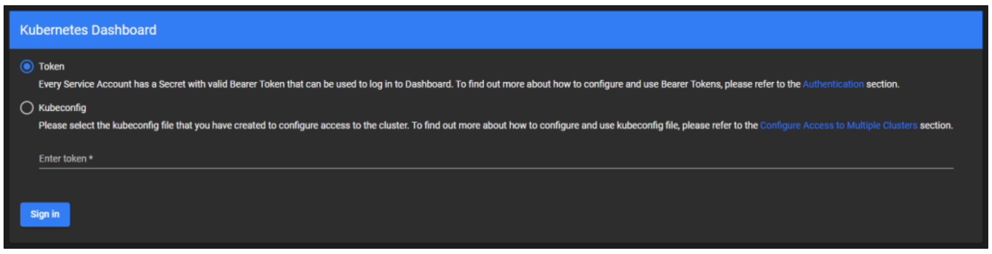
- `kubectl proxy`



```
PS C:\Users\Asdrubale\Desktop\scc_group06\webapp> kubectl proxy
Starting to serve on 127.0.0.1:8001
```

and connect to the following browser link:

http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/#/login



Kubernetes Dashboard

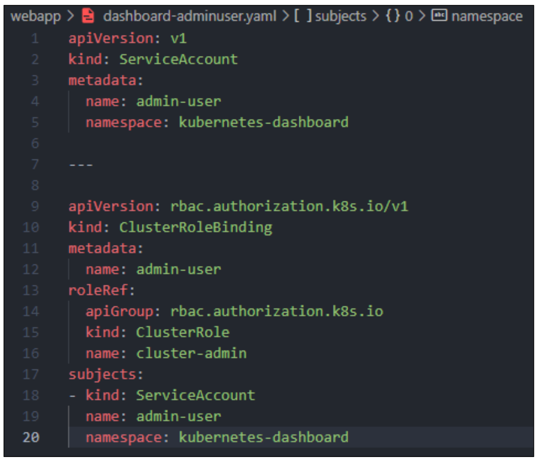
(●) Token
Every Service Account has a Secret with valid Bearer Token that can be used to log in to Dashboard. To find out more about how to configure and use Bearer Tokens, please refer to the Authentication section.

( ) Kubeconfig
Please select the kubeconfig file that you have created to configure access to the cluster. To find out more about how to configure and use kubeconfig file, please refer to the Configure Access to Multiple Clusters section.

Enter token *

Sign in

At this point an enter token it is requested and to obtain it an admin user must be created. To do this it is necessary to create a '.yaml' file (usually named 'dashboard-adminuser').



webapp > dashboard-adminuser.yaml > [ ] subjects > { } 0 > namespace

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard

---

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kubernetes-dashboard
```

Alongside it is described the functionality of each part of the file.

This Kubernetes file defines a **ServiceAccount** named admin-user in the kubernetes-dashboard namespace and a **ClusterRoleBinding** named admin-user that binds the cluster-admin role to the admin-user service account.

The ServiceAccount resource defines an identity for a Pod or set of Pods, allowing them to authenticate and interact with the Kubernetes API server. The metadata section specifies the name of the service account (admin-user) and the namespace where it is created (kubernetes-dashboard).

The ClusterRoleBinding resource binds a role to a subject, which can be a user, group, or service account. In this case, the roleRef section specifies that the cluster-admin role from the rbac.authorization.k8s.io API group should be used. The subjects section specifies that the role should be bound to the admin-user service account in the kubernetes-dashboard namespace.

Together, these resources create a service account that has the permissions of a cluster admin. This can be useful for managing the Kubernetes cluster using the Kubernetes dashboard, or for other administrative tasks that require elevated privileges.

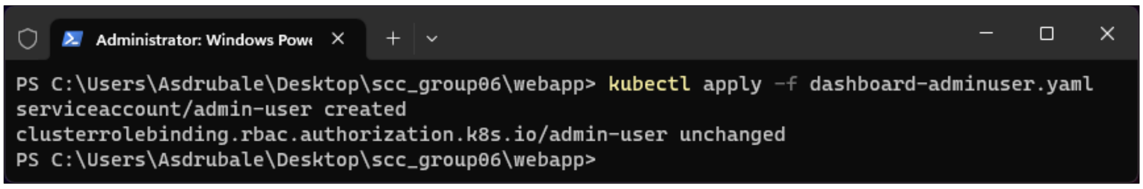
RBAC stands for Role-Based Access Control. It is a method of controlling access to resources in a computer system based on the roles assigned to users or subjects.

In Kubernetes, RBAC is a powerful feature that allows you to define granular permissions for different users and applications in your cluster. It allows you to control access to resources at a fine-grained level, defining who can perform specific actions on specific resources.

RBAC can help you to create a more secure and manageable cluster by controlling access to resources and operations based on user roles and permissions. It can also help you to comply with regulations and industry standards by ensuring that only authorized users have access to sensitive data and resources.

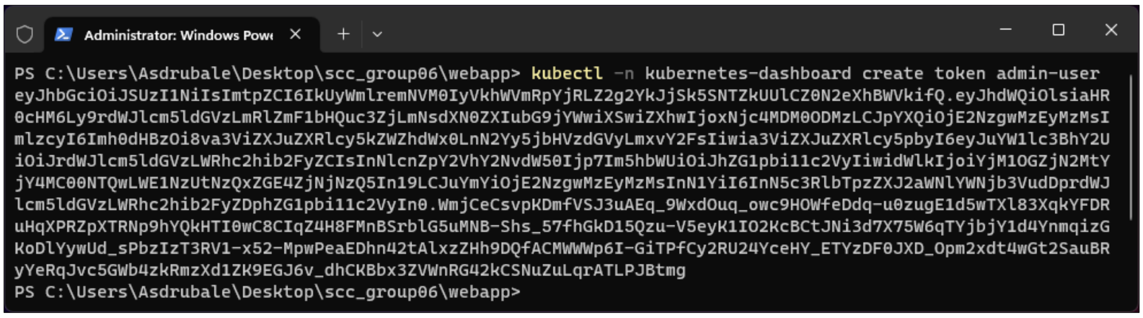
At this point the last 2 things left to do are:

- Creation of the admin user using the '.yaml' file just described with the following command:
  `kubectl apply -f dashboard-adminuser.yaml`



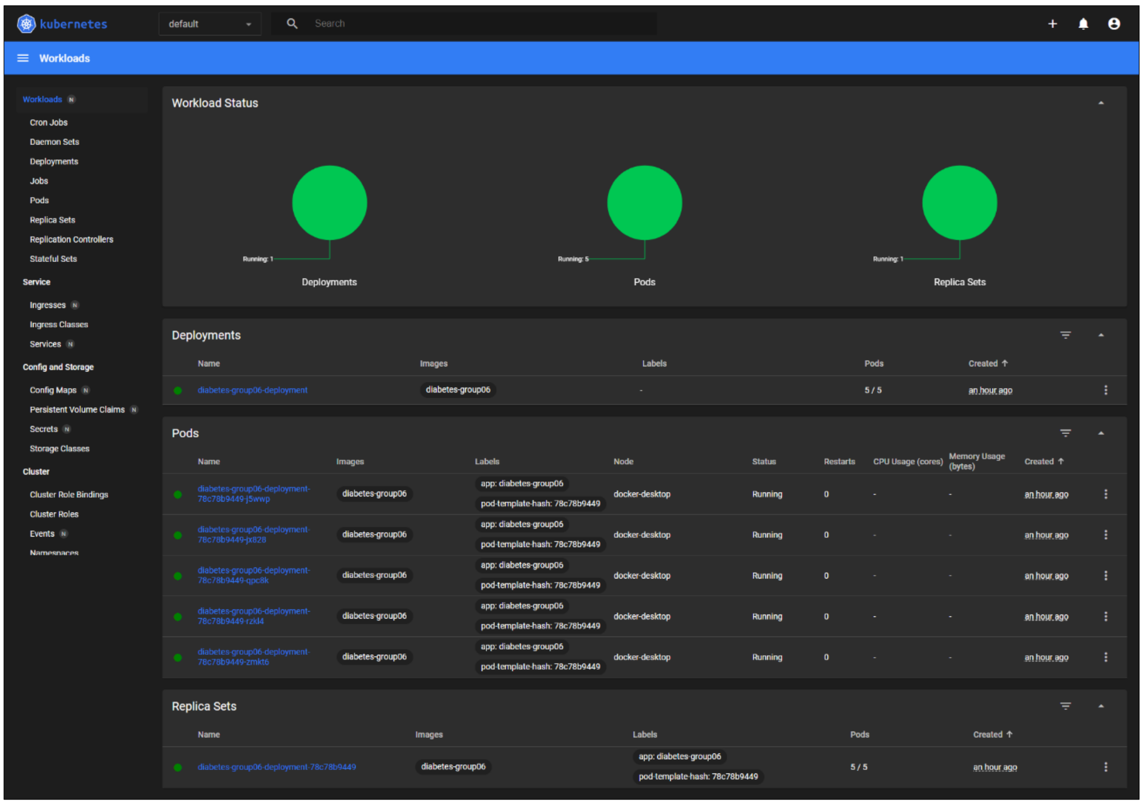
```
PS C:\Users\Asdrubale\Desktop\scc_group06\webapp> kubectl apply -f dashboard-adminuser.yaml
serviceaccount/admin-user created
clusterrolebinding.rbac.authorization.k8s.io/admin-user unchanged
PS C:\Users\Asdrubale\Desktop\scc_group06\webapp>
```

- Creation of an enter token with the following command:
  `kubectl -n kubernetes-dashboard create token admin-user`



```
PS C:\Users\Asdrubale\Desktop\scc_group06\webapp> kubectl -n kubernetes-dashboard create token admin-user
eyJhbGciOiJSUzI1NiIsImtpZCI6IkUyWmlremNVM0IyVkhWVmRpYjRLZ2g2YkJjSk5SNTZkUUlCZ0N2eXhBWVkifQ.eyJhdWQiOlsiaHR
0cHM6Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiXSwiZXhwIjoxNjc4MDM0ODMzLCJpYXQiOjE2NzgwMzEyMzMsI
mlzcyI6Imh0dHBzOi8va3ViZXJuZXRlcy5kZWZhdWx0LnN2Yy5jbHVzdGVyLmxvY2FsIiwia3ViZXJuZXRlcy5pbyI6eyJuYW1lc3BhY2U
iOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIsInNlcnZpY2VhY2NvdW50Ijp7Im5hbWUiOiJhZG1pbi11c2VyIiwidWlkIjoiYjM1OGZjN2MtY
jY4MC00NTQwLWE1NzUtNzQxZGE4ZjNjNzQ5In19LCJuYmYiOjE2NzgwMzEyMzMsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDprdWJ
lcm5ldGVzLWRhc2hib2FyZDphZG1pbi11c2VyIn0.WmjCeCsvpKDmfVSJ3uAEq_9WxdOuq_owc9HOWfeDdq-u0zugE1d5wTXl83XqkYFDR
uHqXPRZpXTRNp9hYQkHTI0wC8CIqZ4H8FMnBSrblG5uMNB-Shs_57fhGkD15Qzu-V5eyK1IO2KcBCtJNi3d7X75W6qTYjbjY1d4YnmqizG
KoDlYywUd_sPbzIzT3RV1-x52-MpwPeaEDhn42tAlxzZHh9DQfACMWWWp6I-GiTPfCy2RU24YceHY_ETYzDF0JXD_Opm2xdt4wGt2SauBR
yYeRqJvc5GWb4zkRmzXd1ZK9EGJ6v_dhCKBbx3ZVWnRG42kCSNuZuLqrATLPJBtmg
PS C:\Users\Asdrubale\Desktop\scc_group06\webapp>
```

Once copied and pasted the token in the browser page seen before it is possible to access the Kubernetes dashboard.



Deployments

	Name	Images	Labels	Pods	Created ↑
●	diabetes-group06-deployment	diabetes-group06	-	5 / 5	an hour ago

Pods

	Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created ↑
●	diabetes-group06-deployment-78c78b9449-j5wwp	diabetes-group06	app: diabetes-group06 pod-template-hash: 78c78b9449	docker-desktop	Running	0	-	-	an hour ago
●	diabetes-group06-deployment-78c78b9449-jx828	diabetes-group06	app: diabetes-group06 pod-template-hash: 78c78b9449	docker-desktop	Running	0	-	-	an hour ago
●	diabetes-group06-deployment-78c78b9449-qpc8k	diabetes-group06	app: diabetes-group06 pod-template-hash: 78c78b9449	docker-desktop	Running	0	-	-	an hour ago
●	diabetes-group06-deployment-78c78b9449-rzkl4	diabetes-group06	app: diabetes-group06 pod-template-hash: 78c78b9449	docker-desktop	Running	0	-	-	an hour ago
●	diabetes-group06-deployment-78c78b9449-zmkt6	diabetes-group06	app: diabetes-group06 pod-template-hash: 78c78b9449	docker-desktop	Running	0	-	-	an hour ago

Replica Sets

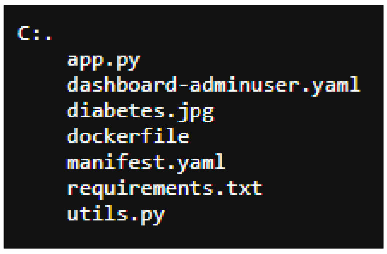
	Name	Images	Labels	Pods	Created ↑
●	diabetes-group06-deployment-78c78b9449	diabetes-group06	app: diabetes-group06 pod-template-hash: 78c78b9449	5 / 5	an hour ago

According with everything was said about Kubernetes advantages, when you edit the **'manifest.yaml'** file to increase the number of replicas for a Kubernetes deployment, Kubernetes will automatically handle the scaling up of the deployment. Kubernetes will create new Pods and ensure that the specified number of replicas are running. Similarly, if you update the image of your application in the manifest file, Kubernetes will automatically create new Pods with the updated image and terminate the old Pods running the previous one.

Kubernetes uses the desired state configuration model, which means that the manifest file defines the desired state of the deployment or service and ensures that the current state matches the desired state by making the necessary changes in real time. This makes it easy to scale up or update your application without having to manually create or delete Pods.

4.3. Further information

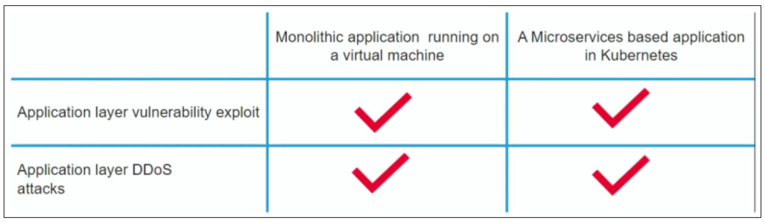
For the correct functioning of the Web App and its deployment it was created the **'webapp' directory** of the project characterized by the structure represented below.



```
C:.
    app.py
    dashboard-adminuser.yaml
    diabetes.jpg
    dockerfile
    manifest.yaml
    requirements.txt
    utils.py
```

5. Security

The security of cloud technology, such as Kubernetes containers, is a **major challenge**. **NGINX**, an open-source web server, has evolved from its initial success as a server to become a reverse proxy, HTTP cache, and Load Balancer. In terms of security, there is no difference between a monolithic application running on a virtual machine and a microservices-based application running on Kubernetes, as both are vulnerable to any type of attack, including DDoS attacks.

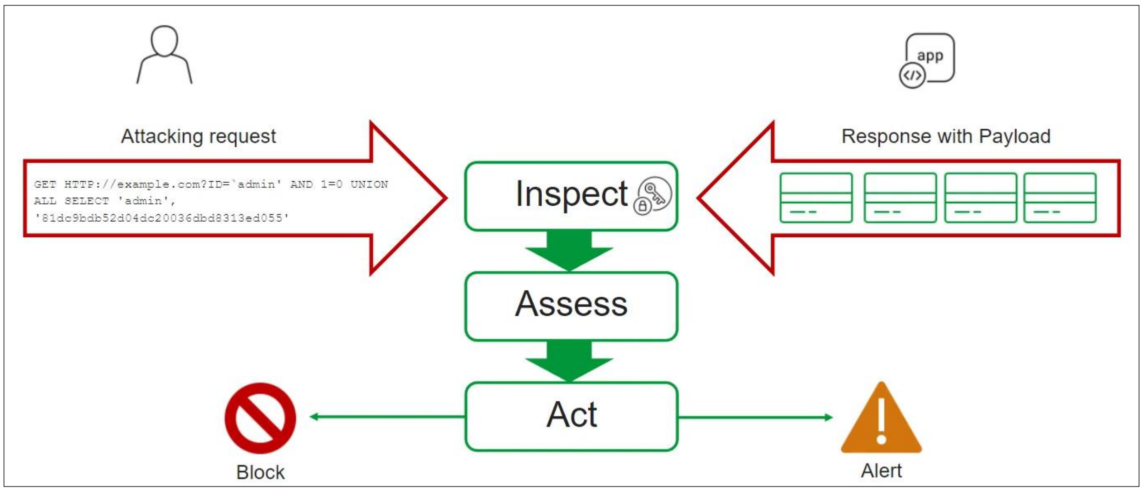


	Monolithic application running on a virtual machine	A Microservices based application in Kubernetes
Application layer vulnerability exploit	✓	✓
Application layer DDoS attacks	✓	✓

To protect applications from these attacks, it is necessary to consider vulnerabilities in the application layer. Suppose there is a GET request attack (e.g. random SQL injection) that is about to compromise data (e.g. credit card numbers).

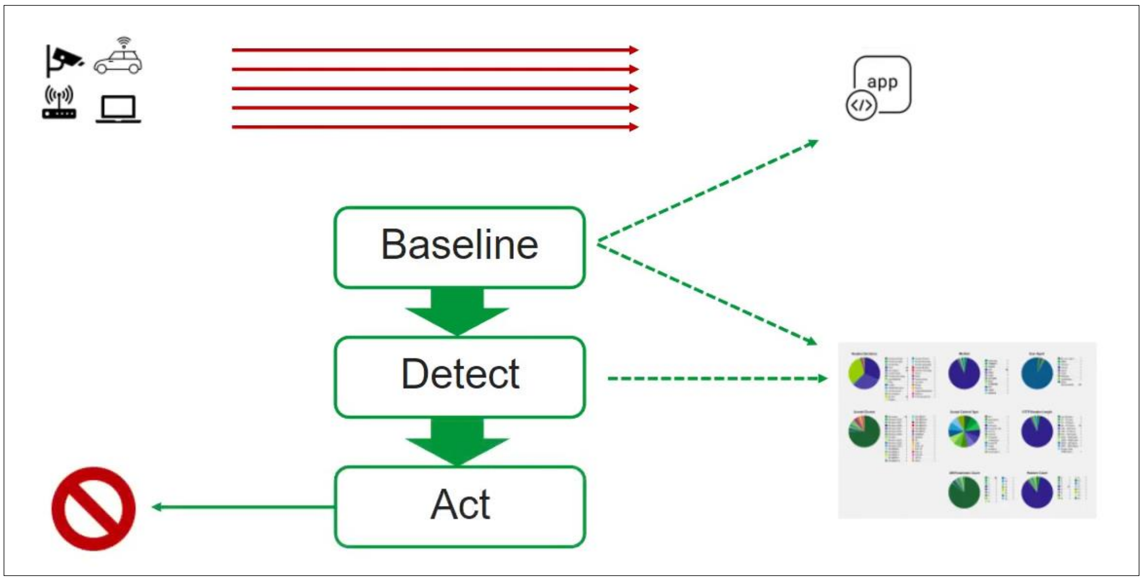
To **prevent and mitigate** the attack it is needed something in the middle between client and server that can decipher/inspect the code at the application level, rather than the packet level. Once the code has been inspected, it is necessary to evaluate it to determine whether it is malicious or not, and this should also be possible with the response. An attack does not pose a threat until the request receives a response or crashes the running processes.

After evaluating the request and/or response, it is necessary to block, alert, or do nothing. In the case of blocks, the connection should not be closed because it would render the application unreachable, and if it was a false positive, we would have prevented correct functioning. A response of ‘this request has been blocked’ should be sent so that the sender can then contact us and request information to allow us to investigate the incident.



```
GET HTTP://example.com?ID=`admin' AND 1=0 UNION
ALL SELECT 'admin',
'81dc9bdb52d04dc20036dbd8313ed055'
```

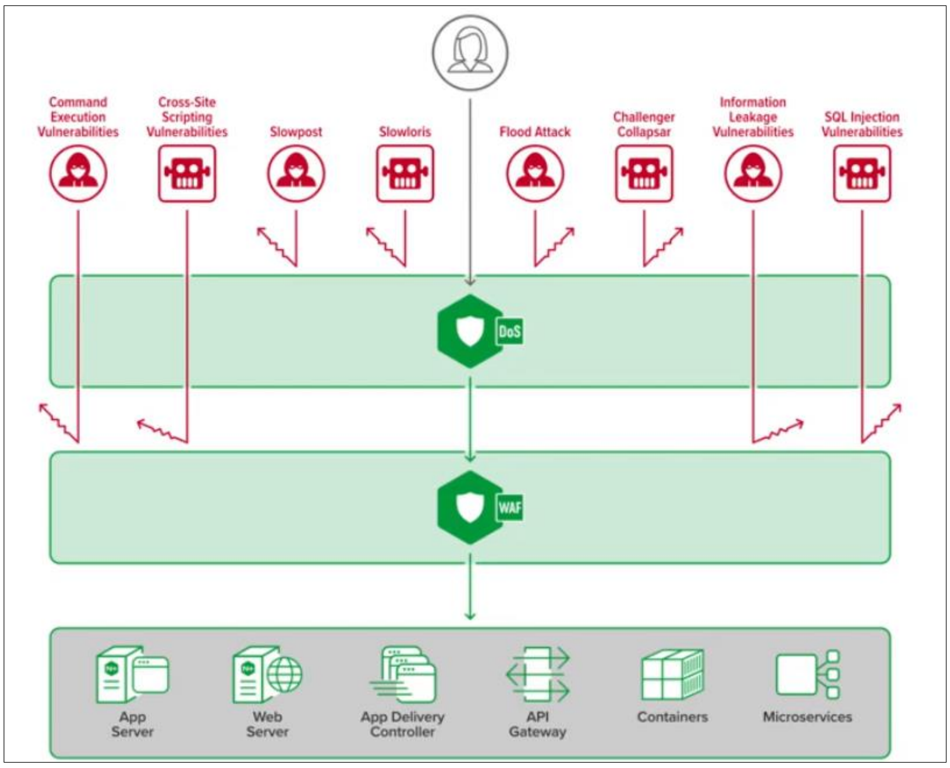
Mitigating DDoS attacks at the application level requires working slightly differently. The first thing to do is to build a baseline, that is how the application should behave because this is the best way to detect DDoS attacks as they modify correct functioning by slowing down the connection. The baseline can detect that there are problems and that the server has become slower in responding. Once an attack is detected, appropriate action can be taken.



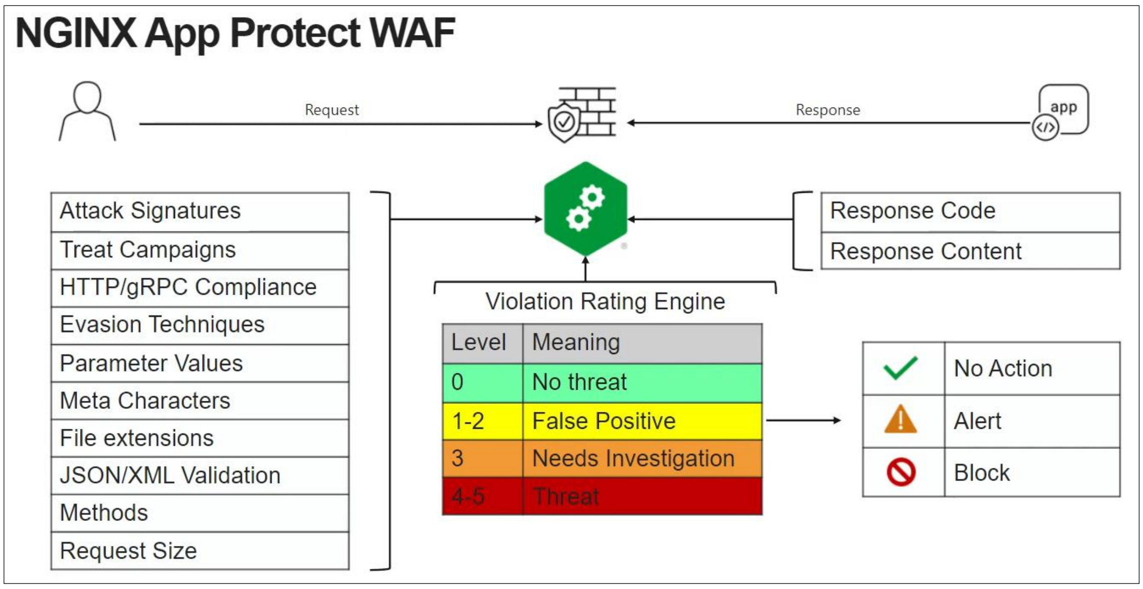
To solve these types of problems, 2 add-on modules are presented:

- **NGINX App Protect WAF**
- **NGINX App Protect DoS**

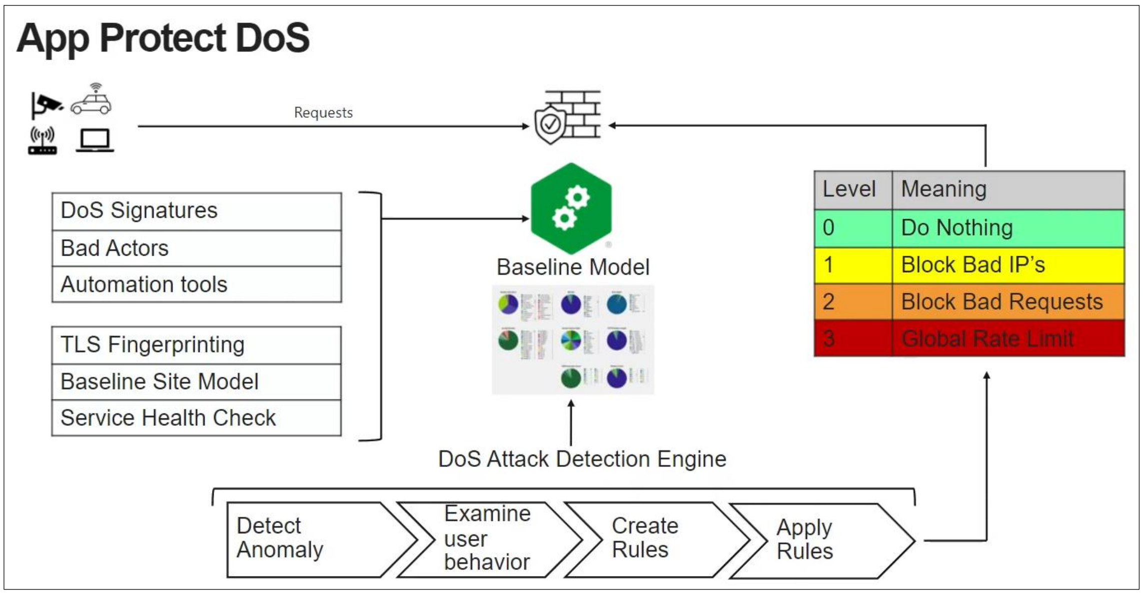
These modules can be used in Kubernetes to enhance security.


Command Execution Vulnerabilities
Cross-Site Scripting Vulnerabilities
Slowpost
Slowloris
Flood Attack
Challenger Collapsar
Information Leakage Vulnerabilities
SQL Injection Vulnerabilities
DoS
WAF
App Server
Web Server
App Delivery Controller
API Gateway
Containers
Microservices

NGINX App Protect WAF is responsible for **monitoring requests and responses**. Monitoring is done by specifying the items shown in the image below. For each request and response, a level of risk is assigned through the **Violation Rating Engine**. Based on the level, anyone using this add-on can configure actions to decide whether to block, alert, or do nothing. There may be false positives, but tests suggest the probabilities are low.



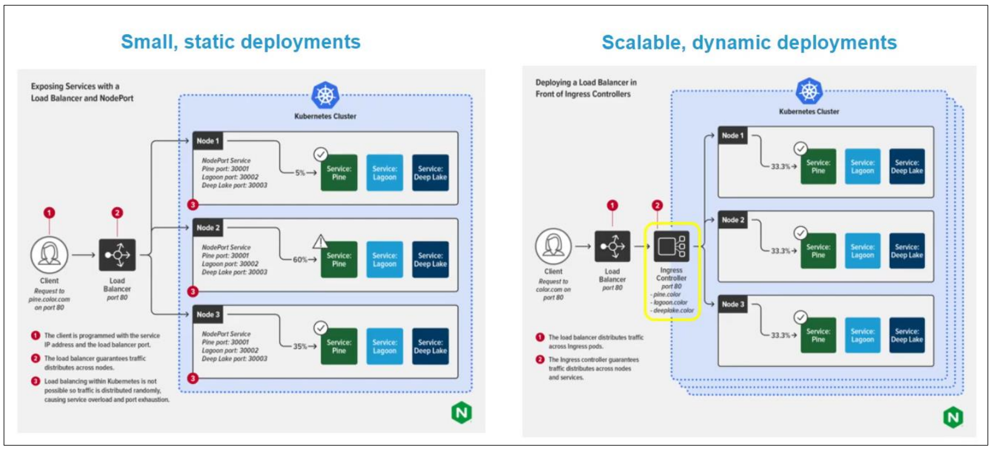
On the other hand, NGINX App Protect DoS is responsible for **monitoring DoS attacks**. As previously mentioned, the first step in protecting against DoS attacks is to **create a baseline**. The add-on acts on this baseline to detect and notify any changes/anomalies based on what is shown in the image below. Once an anomaly is detected, it must examine user behavior to understand the cause. Specifically, it must check incoming traffic to understand the pattern, where it comes from, what type of request it is, and then dynamically create a rule to detect bad traffic. Rules should not be created through hand-written code, but rather dynamically created and applied. The application of such rules should occur according to the rules chosen based on the needs.



There are various ways to expose services, or applications, with Kubernetes.

The most popular method is to use a Load Balancer and NodePort, but this method suffers from disadvantages when dealing with large deployments because load balancing with many IP addresses can take a long time. Additionally, this method does not guarantee some interesting features such as TLS encryption and rate limiting.

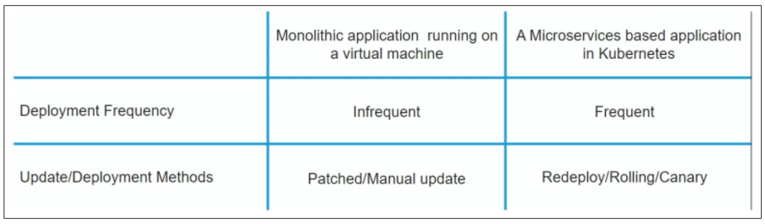
Another method is to use a Load Balancer and the Ingress controller. The controller can be synchronized with Kubernetes or with NGINX, and in this way, we can expose any application of any size and have the additional features that we did not have before.



1 The client is programmed with the service IP address and the load balancer port.
2 The load balancer guarantees traffic distributes across nodes.
3 Load balancing within Kubernetes is not possible so traffic is distributed randomly, causing service overload and port exhaustion.

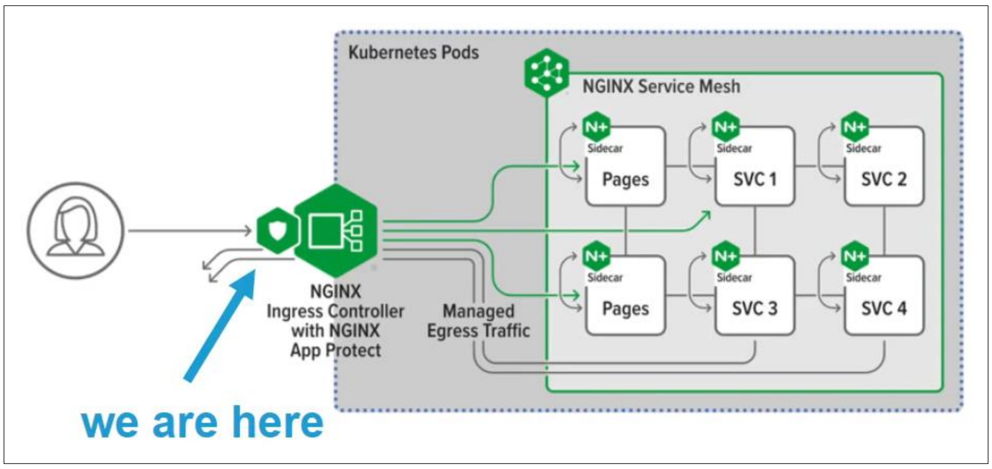
1 The load balancer distributes traffic across Ingress pods.
2 The Ingress controller guarantees traffic distributes across nodes and services.

Application deployment in a monolithic architecture does not happen very frequently, while in microservices, it happens more frequently. When updating or deploying a monolithic application, it is done manually or with a patch, while in Kubernetes or a microservices-centered architecture, it is done with a redeploy, rolling, or canary.



	Monolithic application running on a virtual machine	A Microservices based application in Kubernetes
Deployment Frequency	Infrequent	Frequent
Update/Deployment Methods	Patched/Manual update	Redeploy/Rolling/Canary

The advantages of having a Nginx Ingress controller with Nginx App Protect are that it reduces complexity in the environment, making it easier to manage. Nginx Ingress Controller has other advantages over Kubernetes Ingress controller, such as the possibility to have customized Nginx resources and be easily natively deployed on the cloud. Furthermore, as already mentioned, it introduces policy objects, TLS encryption, etc.


Kubernetes Pods
NGINX Service Mesh
N+
Sidecar
Pages
SVC 1
SVC 2
SVC 3
SVC 4
NGINX Ingress Controller with NGINX App Protect
Managed Egress Traffic
we are here