Machine learning project

Network Intrusion Detection system

Problem statement No.40 – Network Intrusion Detection

The Challenge:

Create a robust network intrusion detection system (NIDS) using machine learning. The system should be capable of analysing network traffic data to identify and classify various types of cyber-attacks (e.g., DoS, Probe, R2L, U2R) and distinguish them from normal network activity. The goal is to build a model that can effectively secure communication networks by providing an early warning of malicious activities.

About Dataset:

Background
The dataset to be audited was provided which consists of a wide variety of intrusions simulated in a military network environment. It created an environment to acquire raw TCP/IP dump data for a network by simulating a typical US Air Force LAN. The LAN was focused like a real environment and blasted with multiple attacks. A connection is a sequence of TCP packets starting and ending at some time duration between which data flows to and from a source IP address to a target IP address under some well-defined protocol. Also, each connection is labelled as either normal or as an attack with exactly one specific attack type. Each connection record consists of about 100 bytes.
For each TCP/IP connection, 41 quantitative and qualitative features are obtained from normal and attack data (3 qualitative and 38 quantitative features) .The class variable has two categories:
• Normal
• Anomalous

Kaggle dataset link - Used in this representation – this will generate a model that only give prediction of class(normal or anomaly)

Technology Used - IBM cloud lite services.

GitHub repo link

🚀 Upgraded NIDS Model – Attack Type Classification Rules

For a upgraded model ,traditional binary classification (Normal vs Anomaly) of kaggle dataset is updated to `DoS`, `Probe`, `R2L`, `U2R`, or `Normal` for more specific analysis on type of attack.

These categories are based on theoretical behavior patterns and implemented using rule-based logic in code using python:

```
import pandas as pd
import numpy as np
import random
from collections import defaultdict

# Define theoretical features for each attack type
```

```
attack_profiles = {
    "DoS": {
        "src_bytes": (0, 500),           # Low volume sent
        "dst_bytes": (1000, 10000),      # High volume received
        "count": (50, 100),              # High connection count
        "serror_rate": (0.8, 1.0),       # High error rate
        "same_srv_rate": (0.9, 1.0),
        "diff_srv_rate": (0.0, 0.2)
    },
    "Probe": {
        "src_bytes": (0, 200),
        "dst_bytes": (0, 500),
        "count": (20, 60),
        "serror_rate": (0.0, 0.2),
        "same_srv_rate": (0.2, 0.6),
        "diff_srv_rate": (0.4, 0.9)
    },
    "R2L": {
        "src_bytes": (0, 100),
        "dst_bytes": (0, 100),
        "count": (1, 5),
        "serror_rate": (0.0, 0.1),
        "num_failed_logins": (2, 5),
        "logged_in": (0, 0),
        "is_guest_login": (1, 1)
    },
    "U2R": {
        "src_bytes": (0, 50),
        "dst_bytes": (0, 50),
        "count": (1, 3),
        "serror_rate": (0.0, 0.1),
        "num_file_creations": (1, 5),
        "root_shell": (1, 1),
        "su_attempted": (1, 1)
    },
    "Normal": {
        "src_bytes": (100, 10000),
        "dst_bytes": (100, 10000),
        "count": (1, 10),
        "serror_rate": (0.0, 0.1),
        "same_srv_rate": (0.7, 1.0),
        "diff_srv_rate": (0.0, 0.3)
    }
}

# Common fields across all rows
common_fields = {
    "duration": (0, 60),
```

```
    "protocol_type": ["tcp", "udp", "icmp"],
    "service": ["http", "ftp", "smtp", "dns", "ssh"],
    "flag": ["SF", "REJ", "S0", "RSTR"],
}

# Map attack type to class (binary)
attack_class = {
    "DoS": "anomaly",
    "Probe": "anomaly",
    "R2L": "anomaly",
    "U2R": "anomaly",
    "Normal": "normal"
}

# Function to score how well a row matches a profile
def match_score(row, profile):
    score = 0
    for feature, (low, high) in profile.items():
        if feature in row:
            try:
                val = float(row[feature])
                if low <= val <= high:
                    score += 1
            except:
                continue
    return score

# Assign attack_type and class to each row
def assign_attack_type(row):
    best_attack = None
    best_score = -1
    for attack, profile in attack_profiles.items():
        score = match_score(row, profile)
        if score > best_score:
            best_score = score
            best_attack = attack
    return best_attack

# Generate a dataset
def generate_theoretical_attack_dataset(num_samples=1000):
    data = []
    attack_types = list(attack_profiles.keys())
    samples_per_attack = num_samples // len(attack_types)

    for attack in attack_types:
        for _ in range(samples_per_attack):
            row = {}
```

```
            # Add common fields
            row["duration"] = random.randint(*common_fields["duration"])
            row["protocol_type"] =
random.choice(common_fields["protocol_type"])
            row["service"] = random.choice(common_fields["service"])
            row["flag"] = random.choice(common_fields["flag"])

            # Add features for the current attack type
            for feature, (low, high) in attack_profiles[attack].items():
                row[feature] = round(random.uniform(low, high), 2)

            # Fill missing features with 0 if not defined for this attack type
            all_possible_features = set(f for d in attack_profiles.values()
for f in d)
            for feature in all_possible_features:
                if feature not in row:
                    row[feature] = 0

            # Add labels
            row["attack_type"] = attack
            row["class"] = attack_class[attack]

            data.append(row)

    df = pd.DataFrame(data)
    return df

# Generate and show the synthetic dataset
synthetic_df = generate_theoretical_attack_dataset(1000)


# Load your uploaded dataset
input_path = "Train_data.csv"
df = pd.read_csv(input_path)

df['attack_type'] = df.apply(assign_attack_type, axis=1)
df['class'] = df['attack_type'].map(attack_class)

# Save the dataset for export if needed
output_path = "Train_data_with_attack_types_enhanced.csv"
df.to_csv(output_path, index=False)

print(f"Saved labeled dataset to {output_path}")
```

🔍 Attack Type Breakdown

Attack Type	Full Form	Description
DoS	Denial of Service	Overwhelms network to disrupt service availability
Probe	Probing/Reconnaissance	Gathers network/system info through scanning
R2L	Remote to Local	External attacker tries to gain unauthorized access
U2R	User to Root	Authorized user tries to escalate to root privileges
Normal	—	Legitimate connection with no signs of intrusion

Rule-Based Classification Logic:

Below are the key behavioral indicators (features) and red flags used to classify each attack type:

1. DoS (Denial of Service)

- Large volume of traffic or repeated bursts
- Targets same service/host continuously
- Symptoms of SYN flood or similar volumetric attacks

🔑 Key Indicators:

- `count` ≥ 50
- `serror_rate` ≥ 0.8
- `diff_srv_rate` ≤ 0.2

2. Probe

- Scans across many services/host
- Low payload but high variety
- Attempting to map open ports/services

🔑 Key Indicators:

- count in [20, 60]
- diff_srv_rate ≥ 0.4
- serror_rate ≤ 0.2

3. R2L (Remote to Local)

- Suspicious or repeated login failures
- Attempts by external hosts to gain internal access

🔑 Key Indicators:

- num_failed_logins > 1
- logged_in = 0
- is_guest_login = 1

4. U2R (User to Root)

- Subtle privilege escalation attempts
- Presence of root access shells, su attempts, or file manipulations

🔑 Key Indicators:

- root_shell = 1
- su_attempted = 1
- num_file_creations > 0

✅ Normal Traffic

If none of the red flag rules above apply, the traffic is assumed to be Normal.

🧪 **Implementation Notes**

- All records are scored using a match_score() function.
- The highest-scoring attack type is selected as the predicted label.
- The label is stored in a new column: attack_type
- The original binary class is mapped as:

  anomaly → DoS, Probe, R2L, U2R

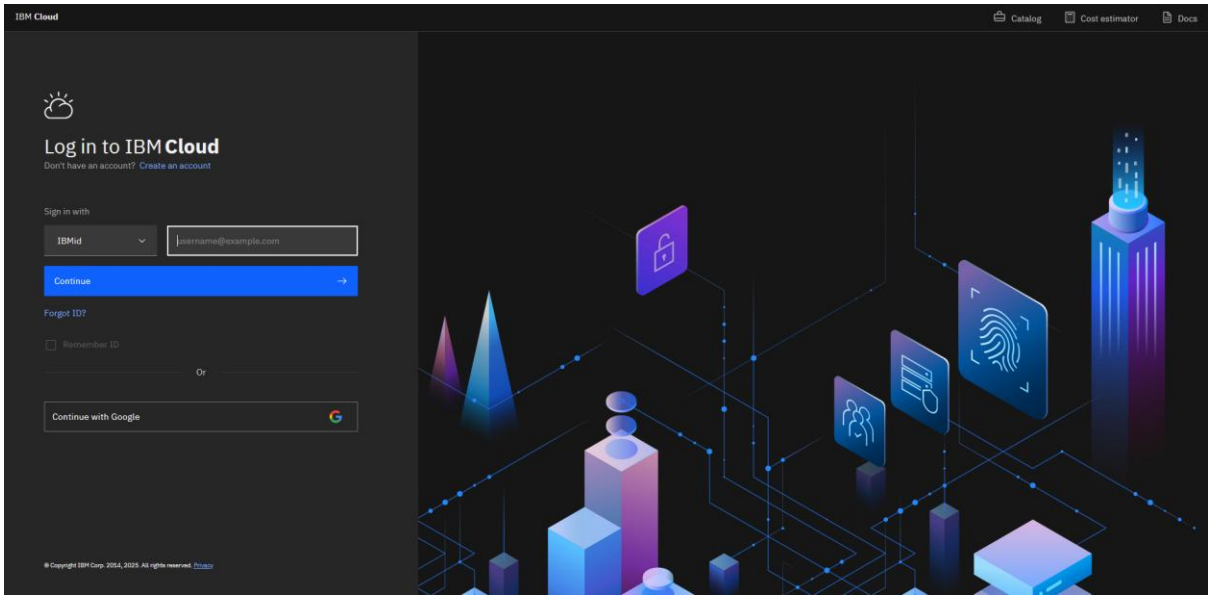
  normal → Normal

💡 Usage

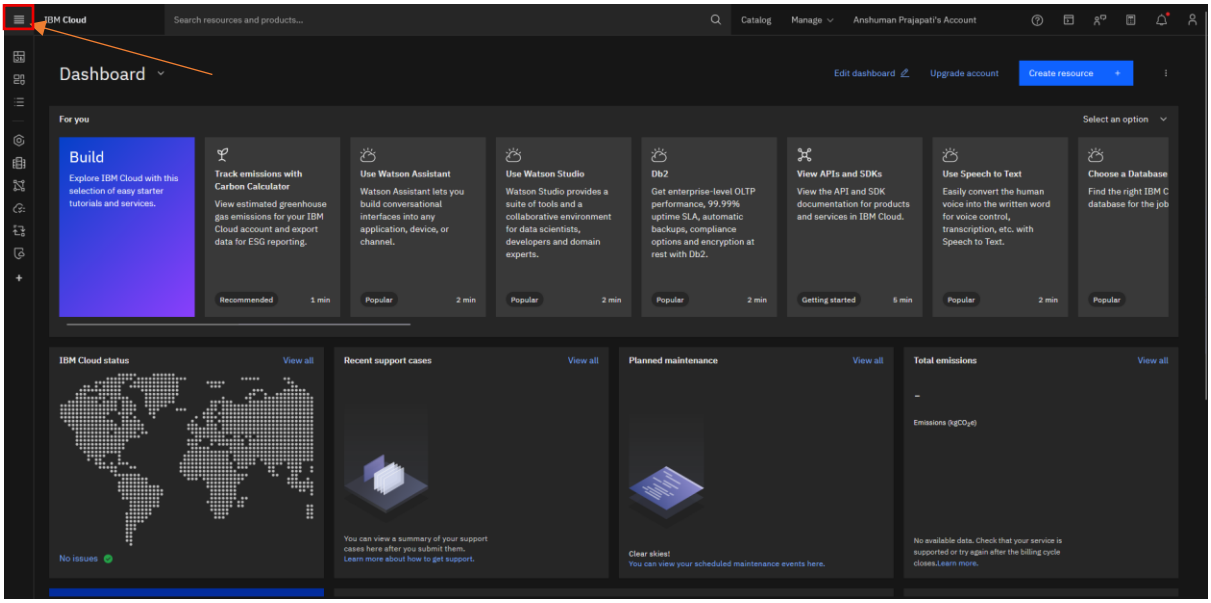
- Can be used as training data for supervised multi-class classification models (e.g., Random Forest, XGBoost).
- Compatible with IBM Watsonx.ai AutoAI for model generation and deployment.
- Provides interpretable security labels for SIEM or firewall logs.

DATA made using python and Kaggle dataset sets which can be used for upgraded model:

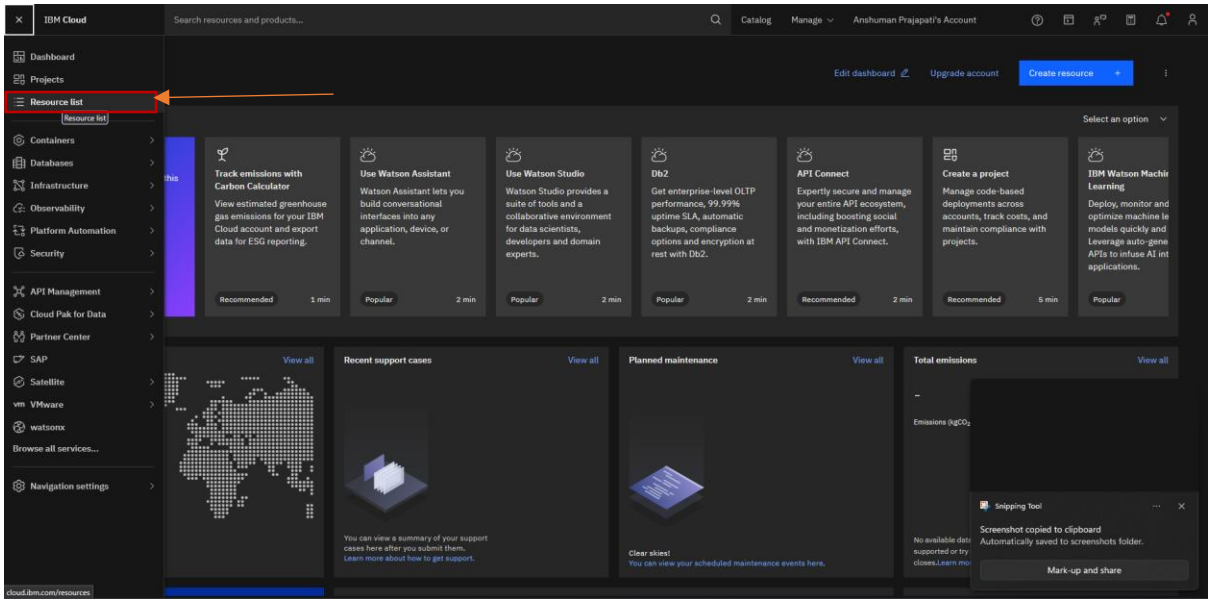
-New data set for training of model



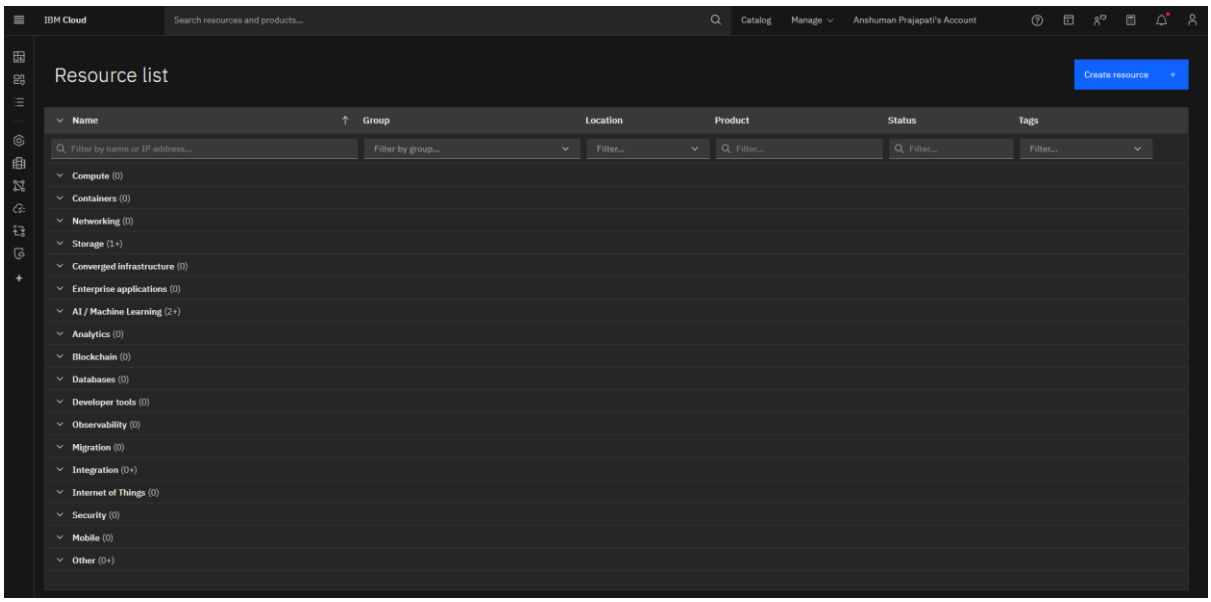
1. Login to [IBM cloud] with your credentials.



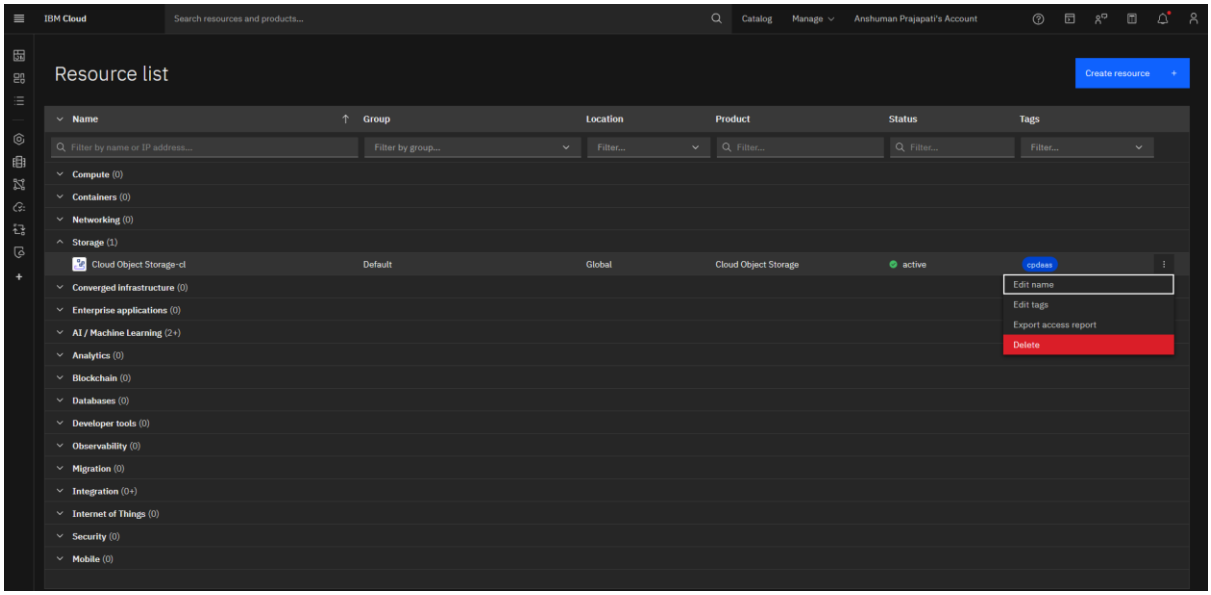
2. This is homepage of IBM cloud now press on top left corner of this page.

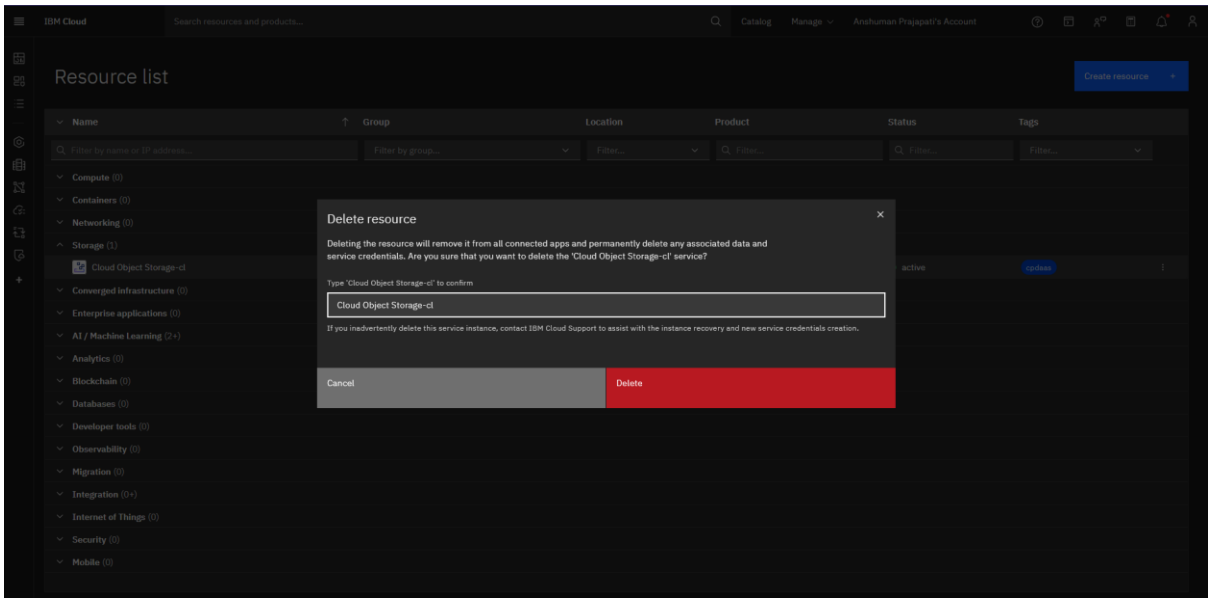


3. Press on Resource list

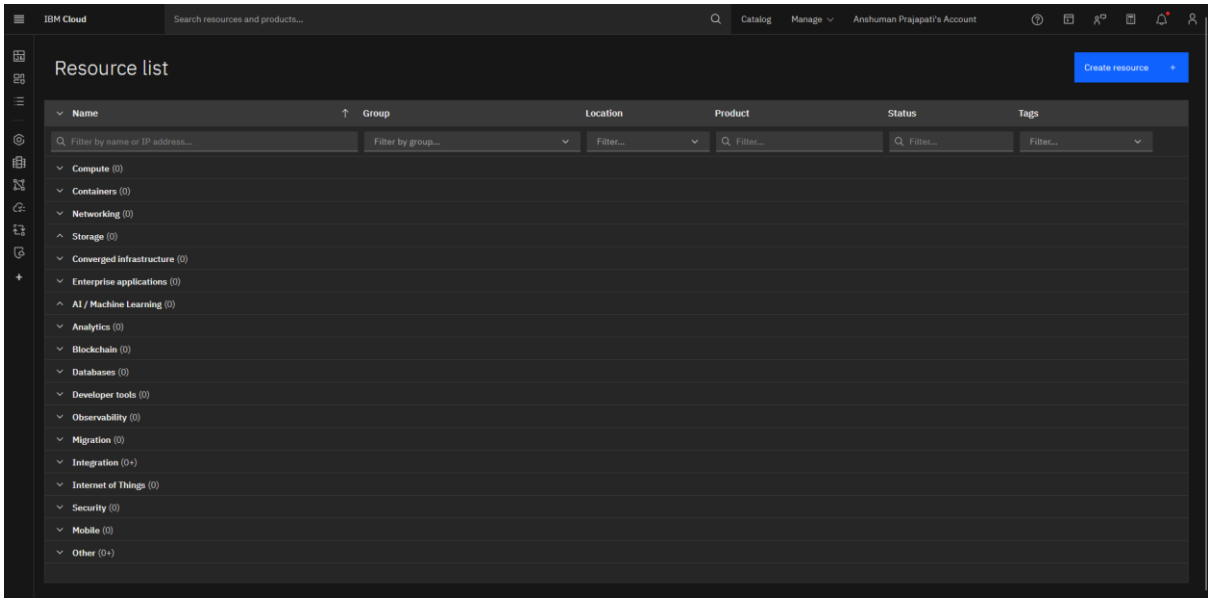


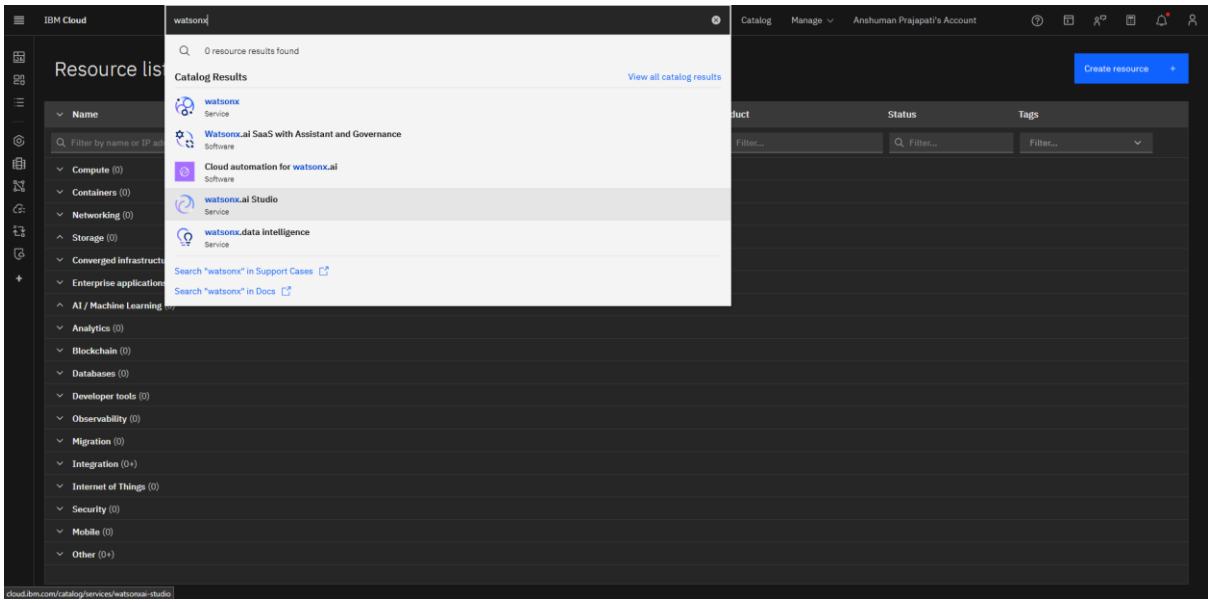
4. In this page if any resource is in use delete it and make it free.



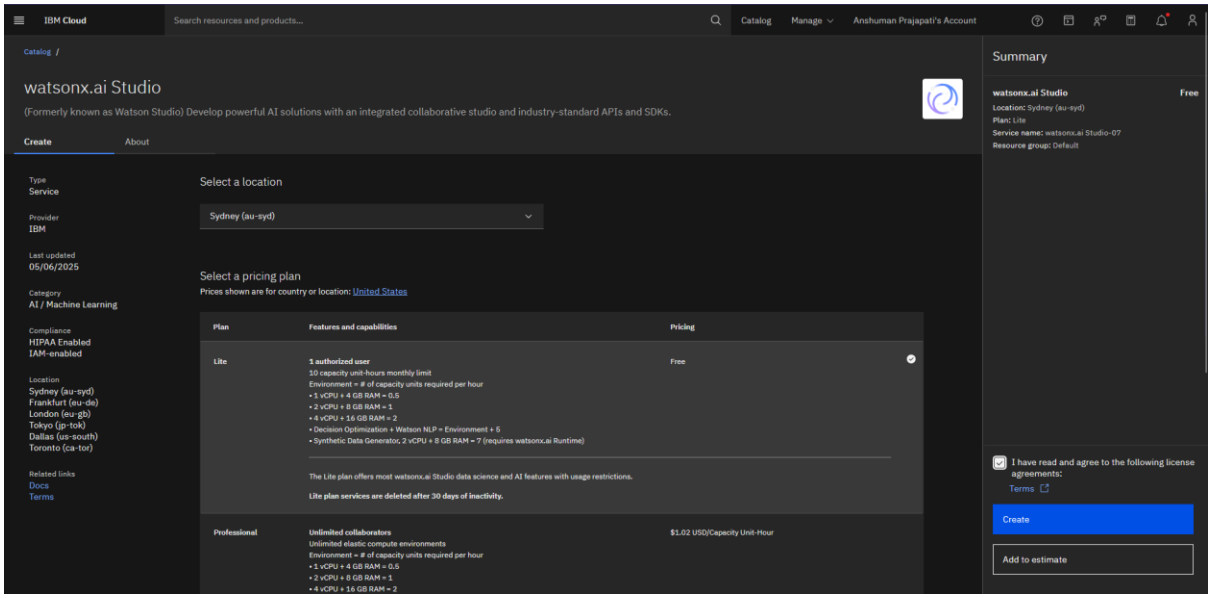


5. Repeat this for all the occupied resources and make sure no resources are in use.

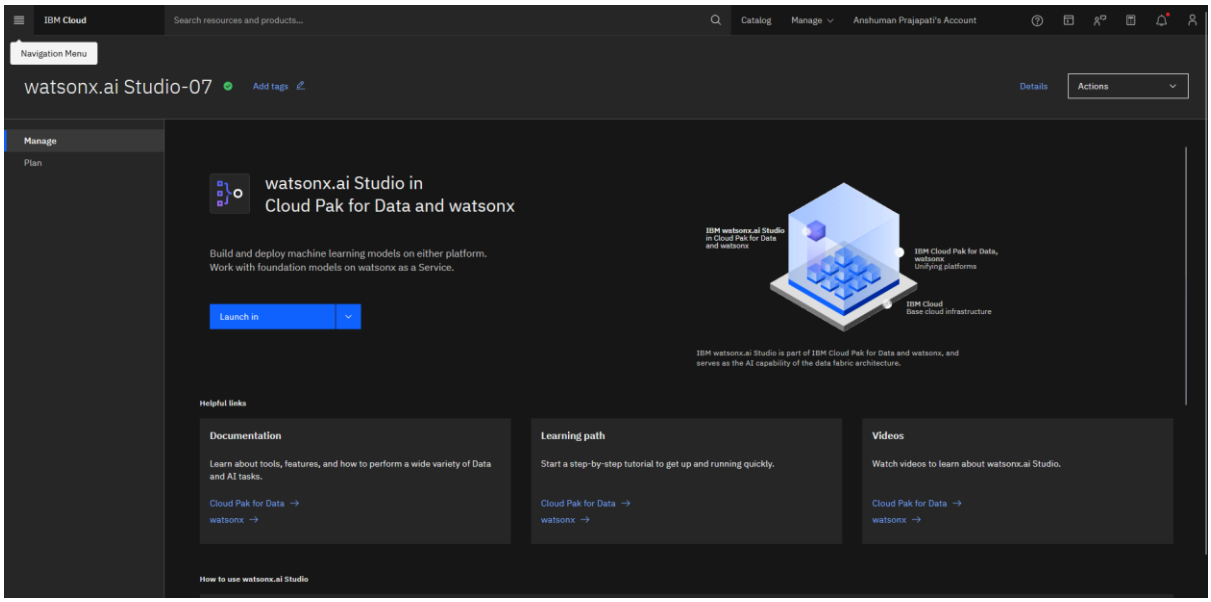




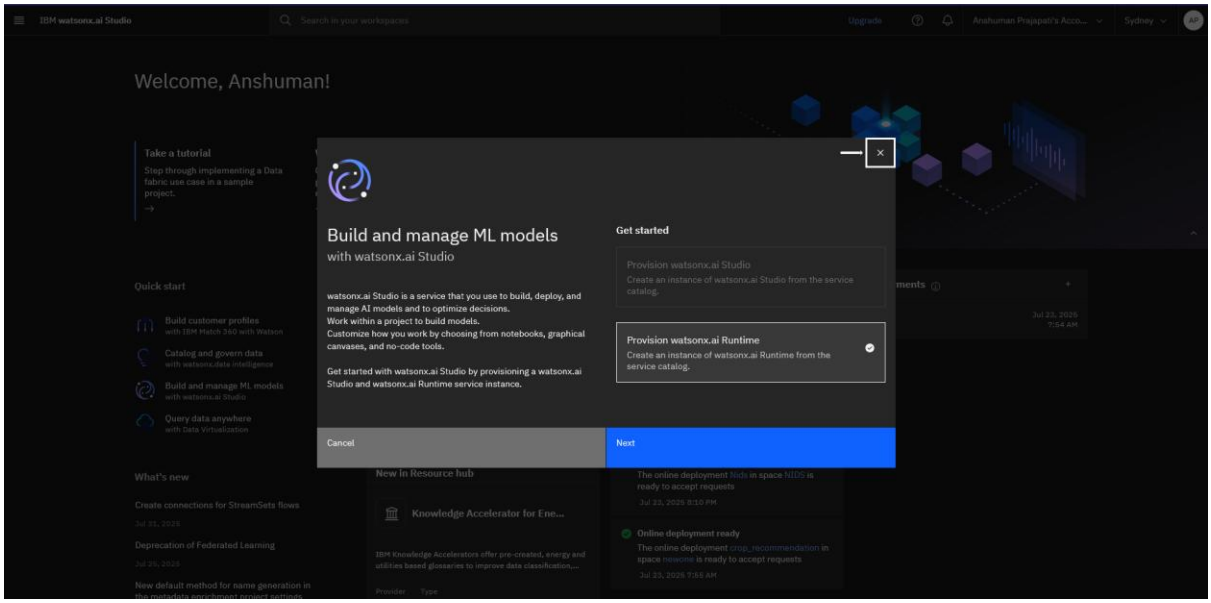
6. Now click search for watsonx.ai Studio in search bar and press enter.



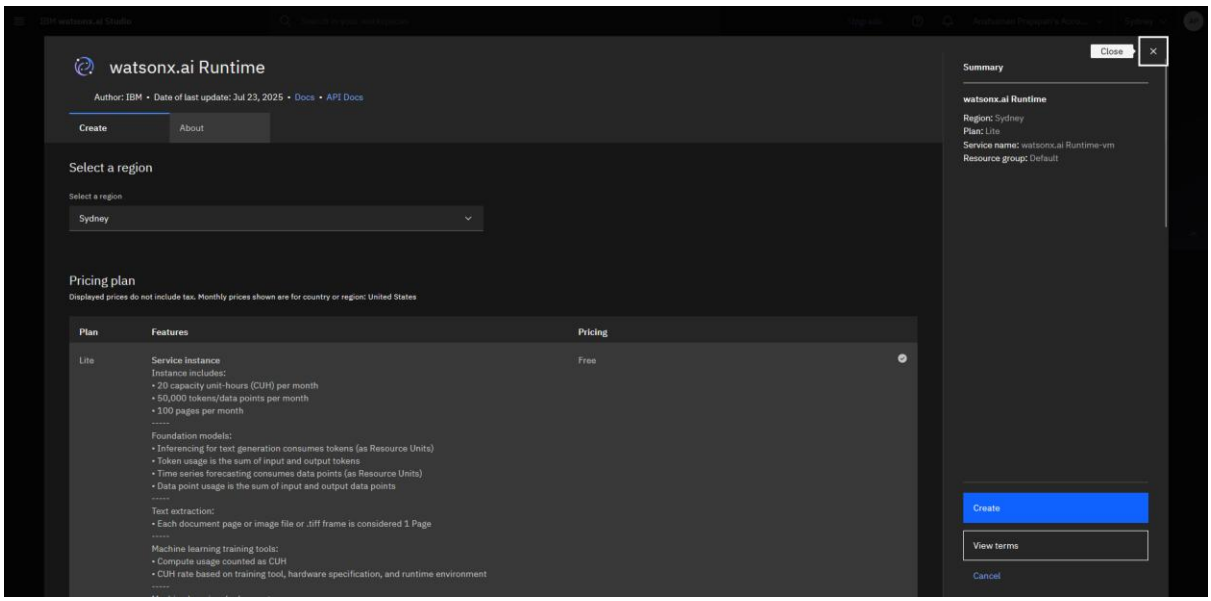
7. Now create with lite plan and and make sure to check on license agreements.



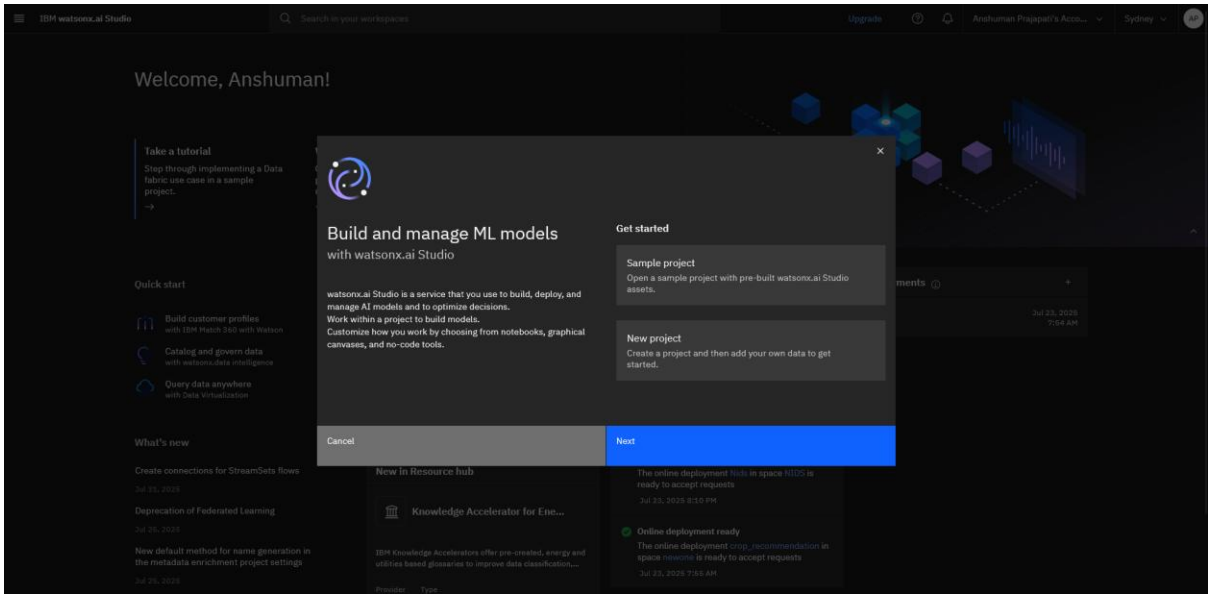
8. Now click on launch in.



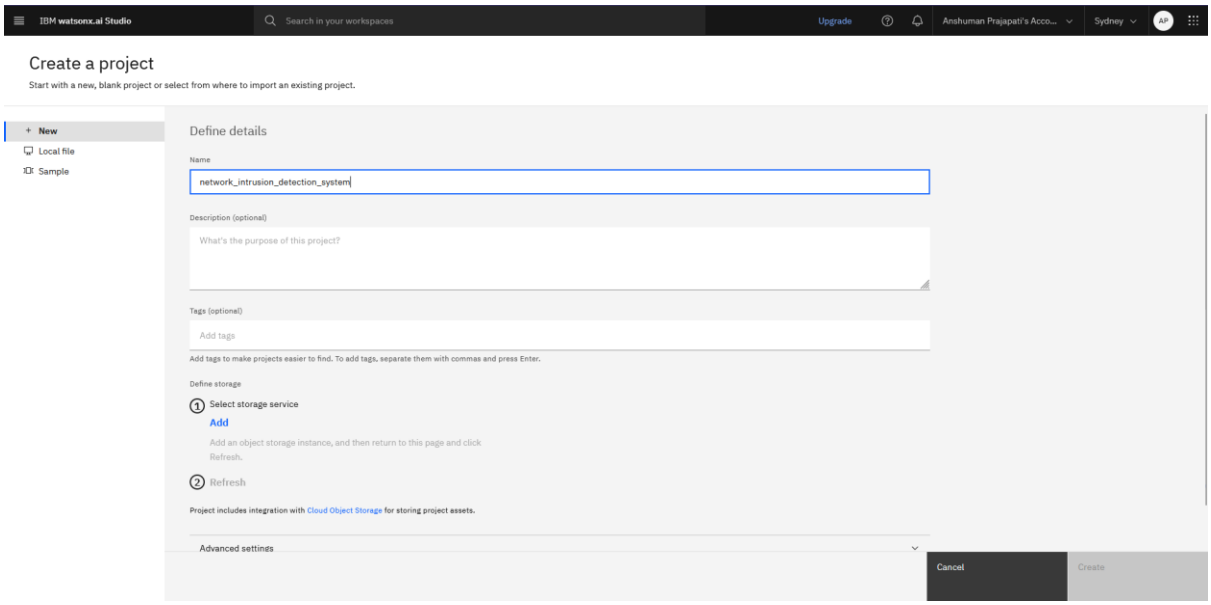
9. Now select provision watsonx.ai Runtime and press next.



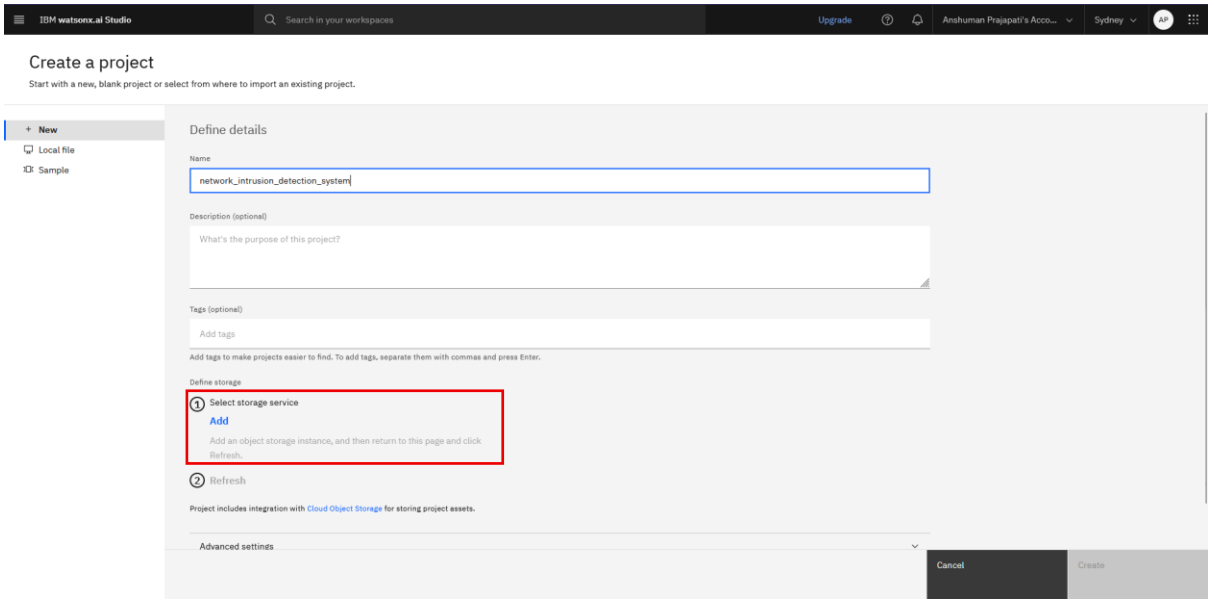
10. Press create.



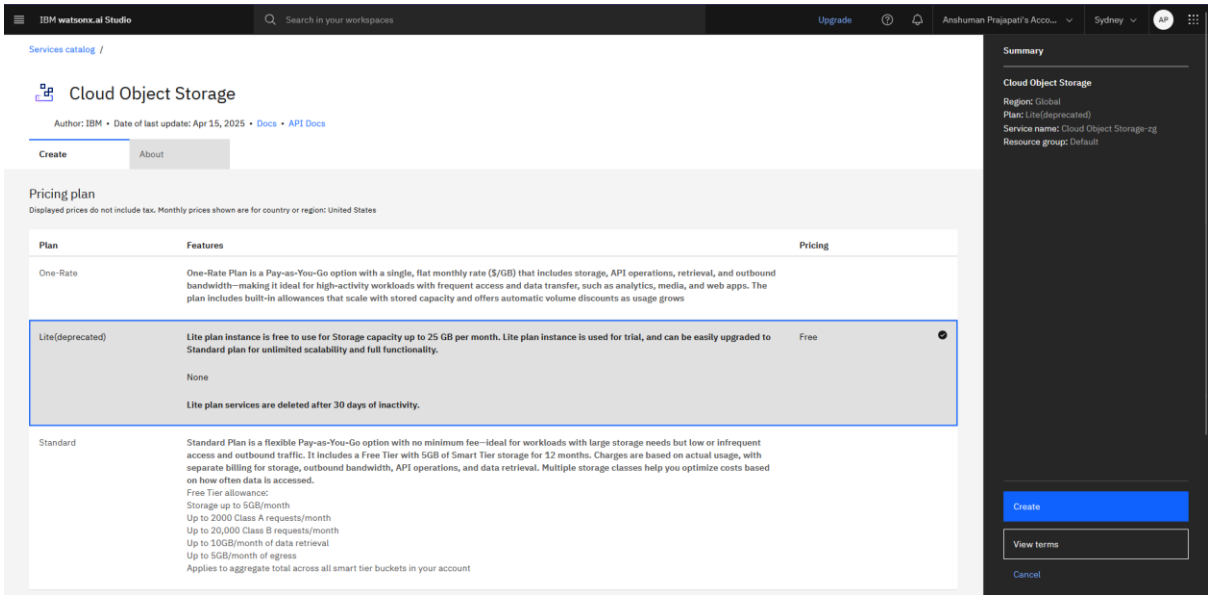
11. Select new project and press next.



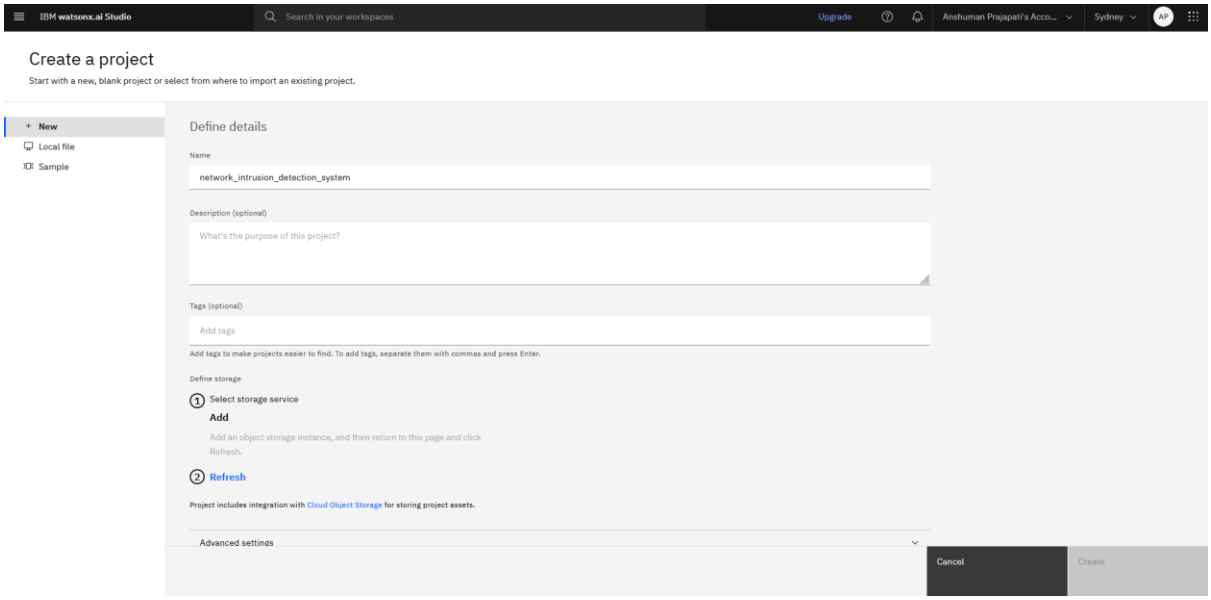
12. Give the project a name.



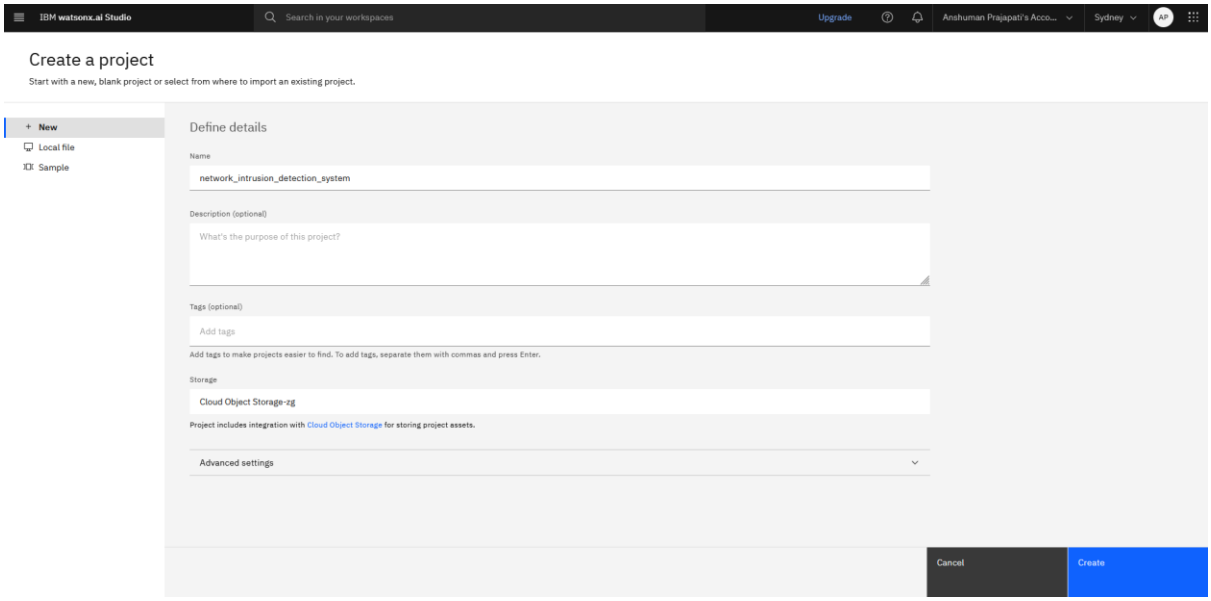
13. Now add storage to it.



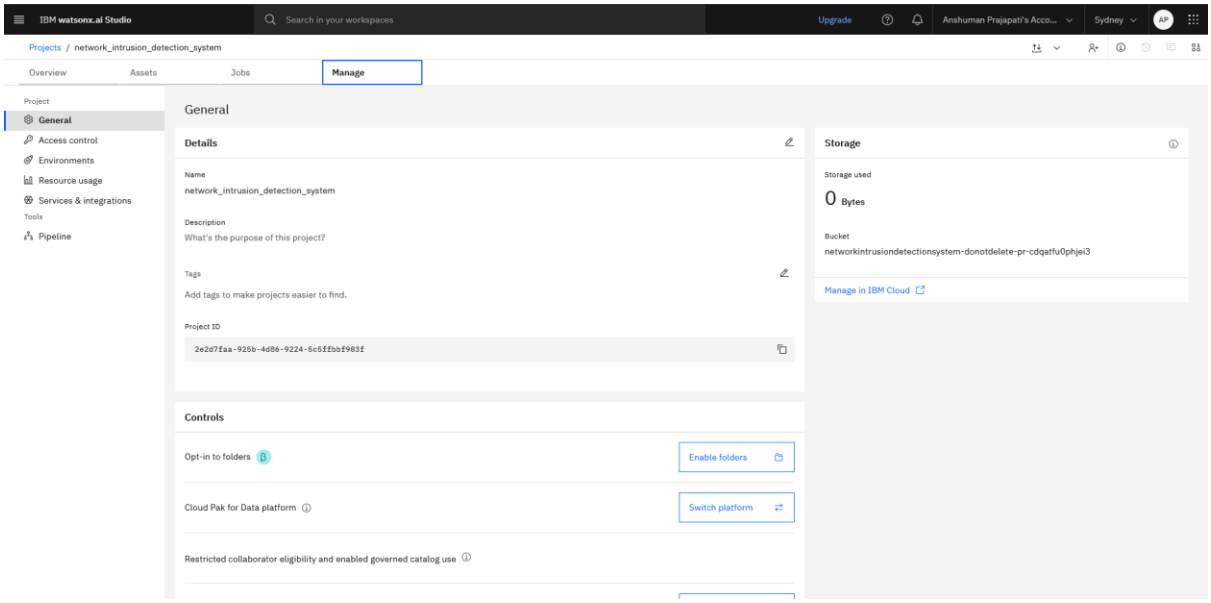
14. Create a object storge in lite plan.



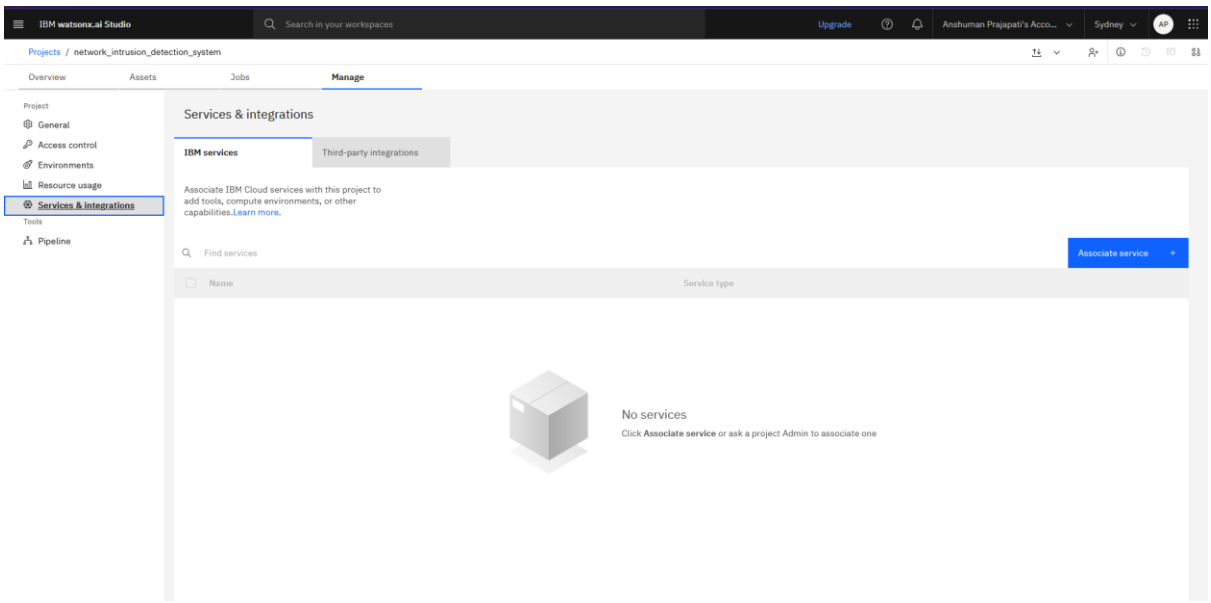
15. Press refresh to show up the storage.



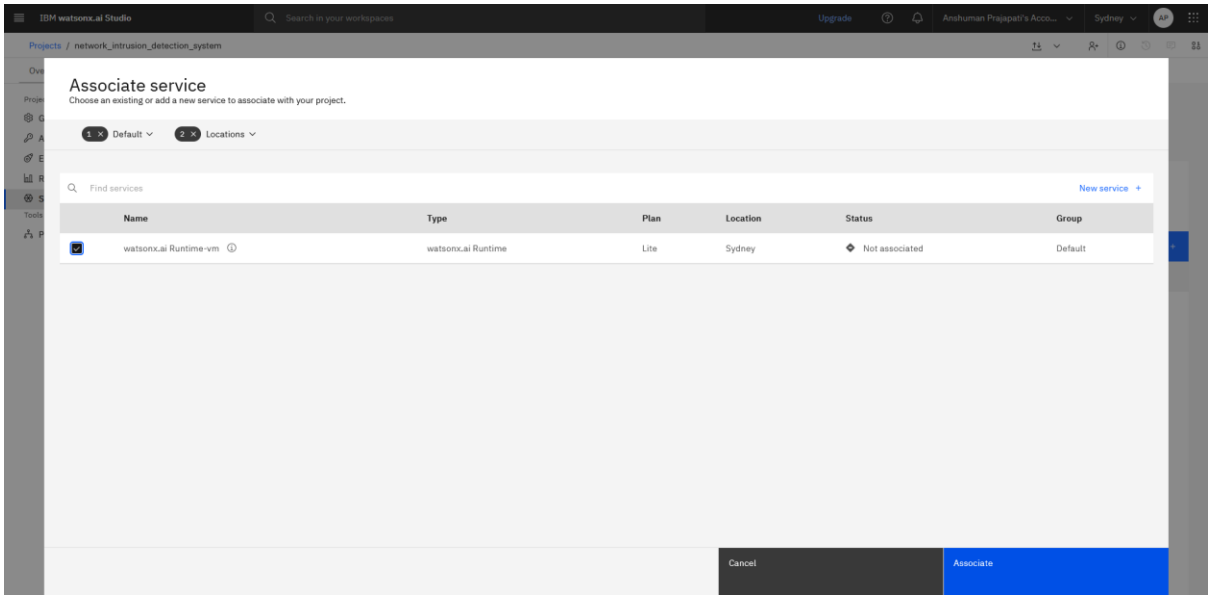
16. Press create.



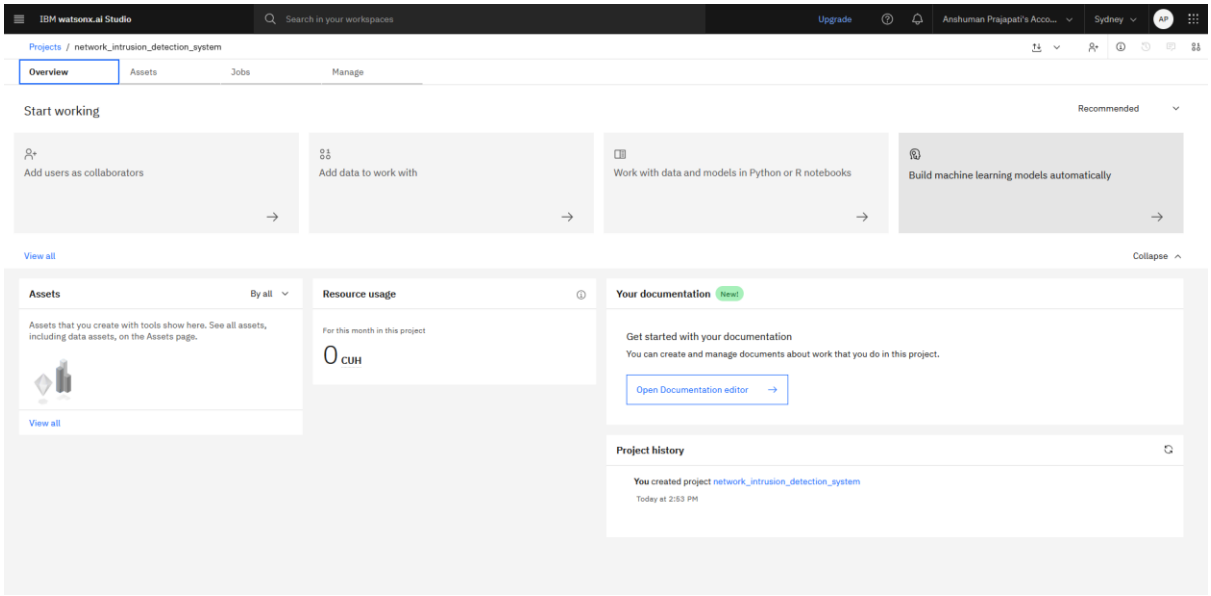
17. Now on manage tab go to services and integration.



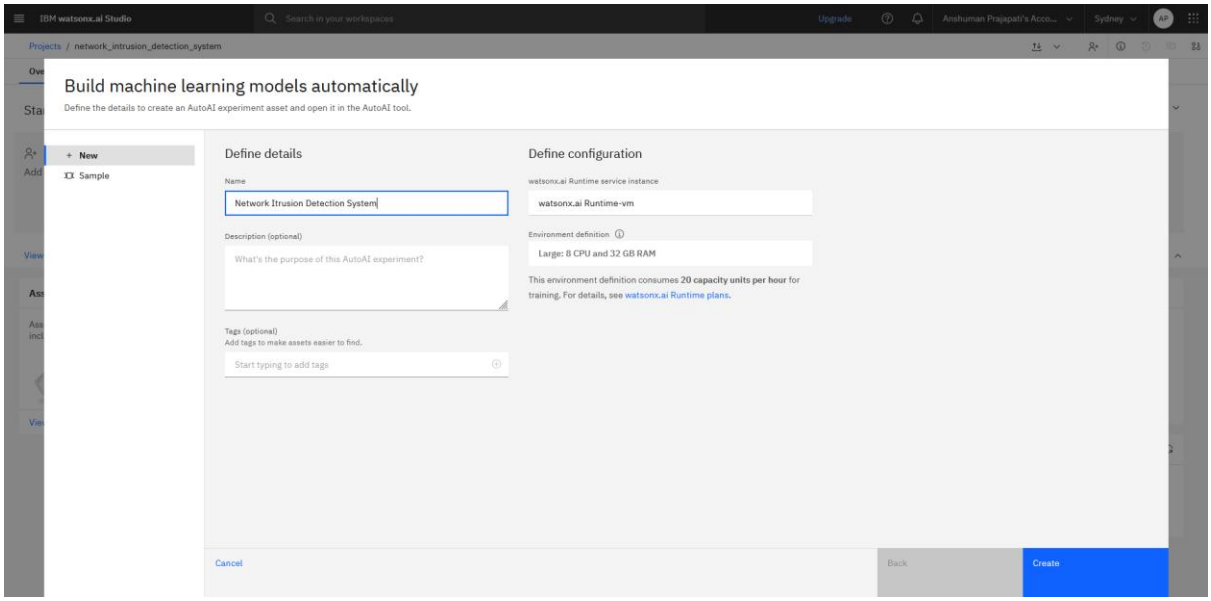
18. Now click on associate service.



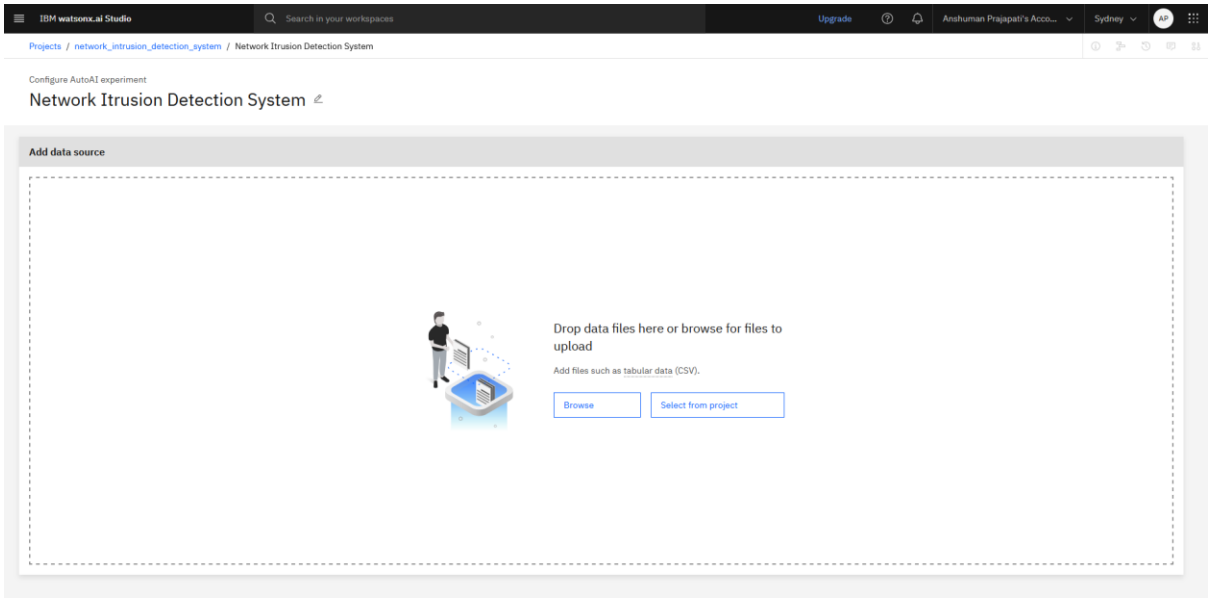
19. Check on watsonx.ai Runtime service and associate it.



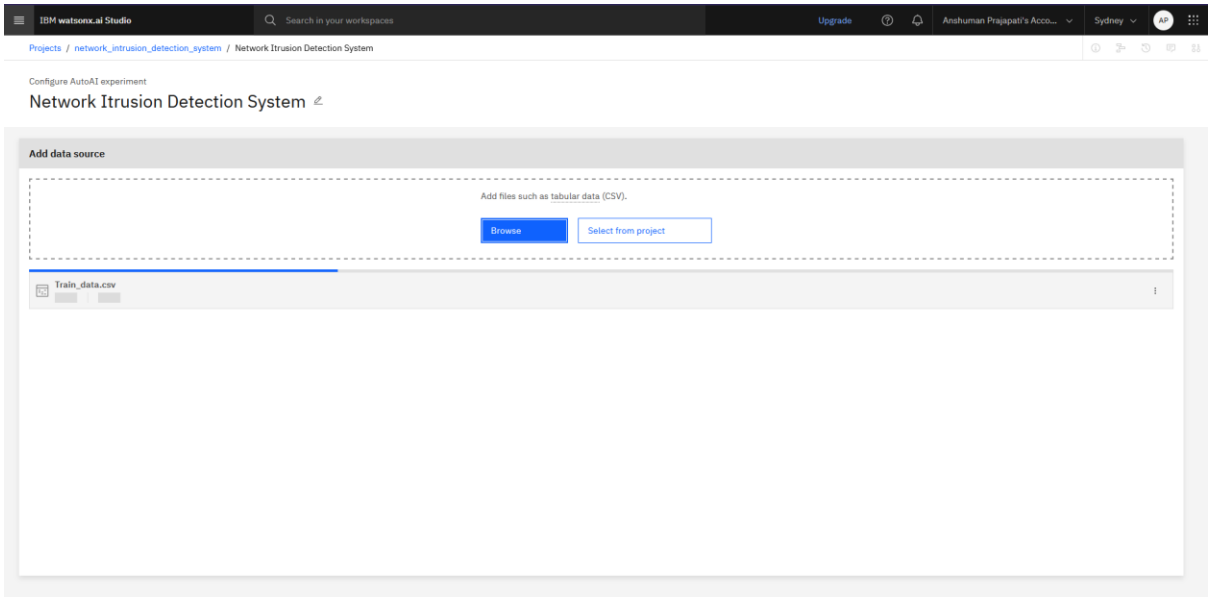
20. Now comeback to overview tab and select ***Build machine learning models automatically***



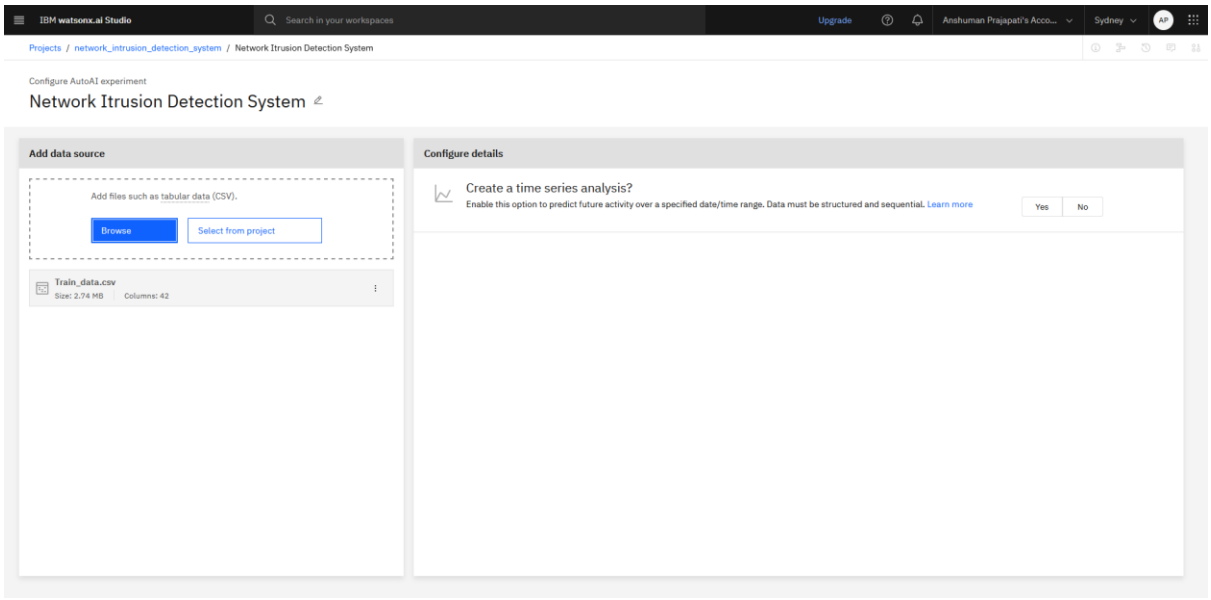
21. Give the experiment a name and click on create.



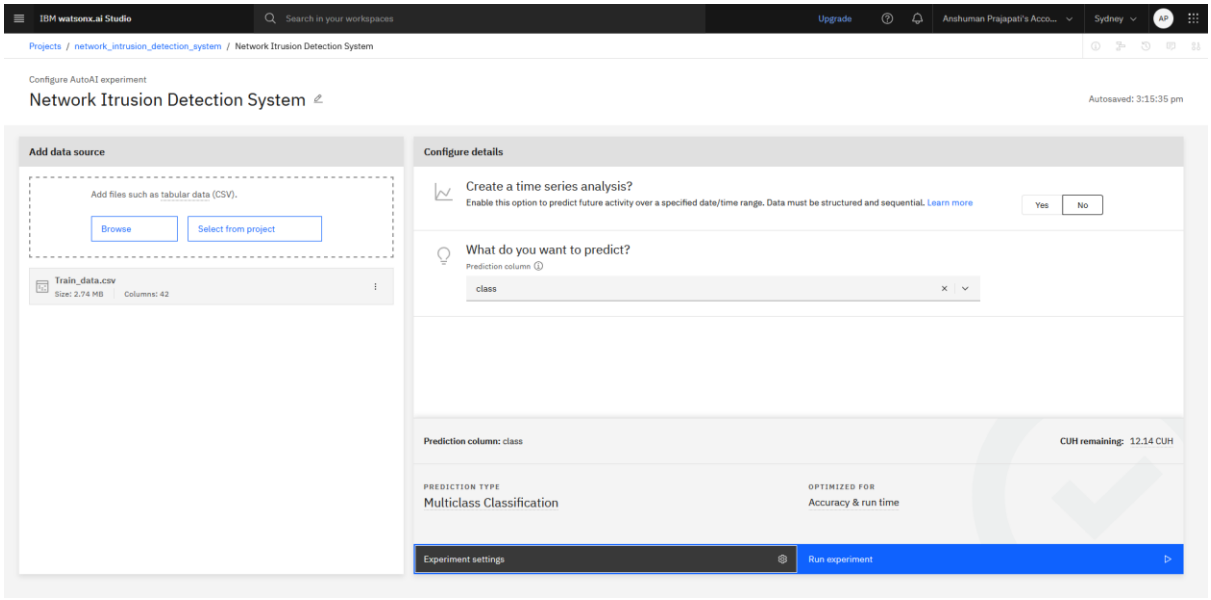
22. Now browse the file to upload (i.e. the .csv file to train the model).



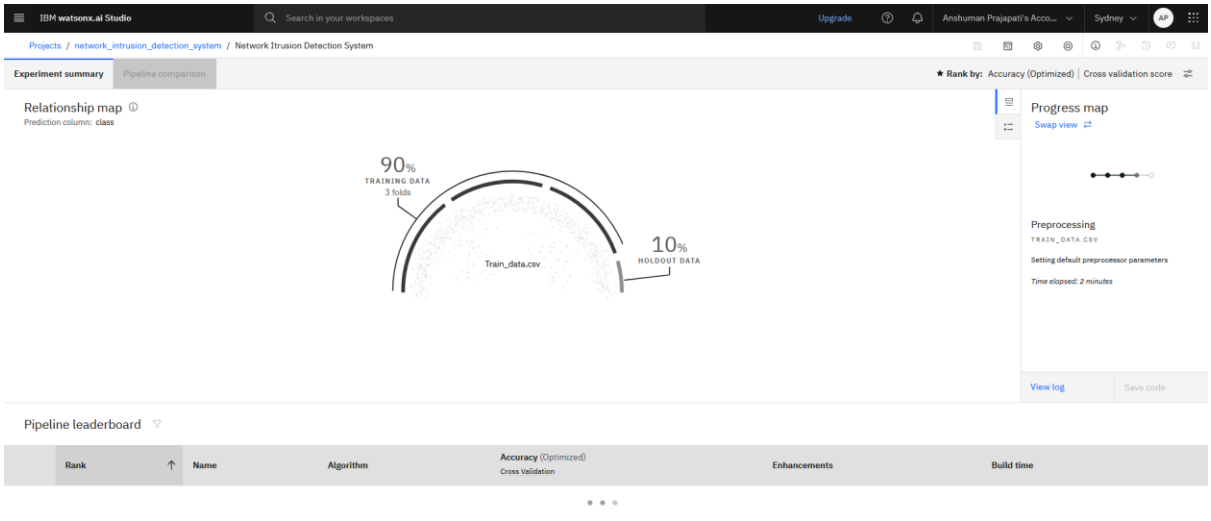
23. Wait for the file to upload.



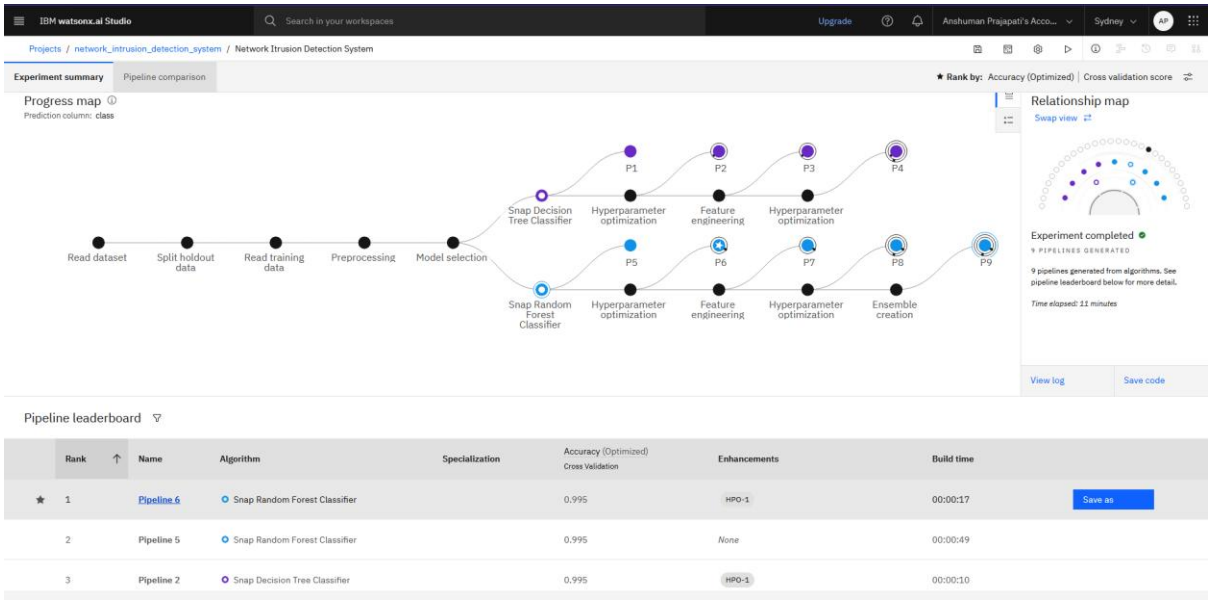
24. Now after the file gets completely uploaded it will ask for time series analysis select ***no***.



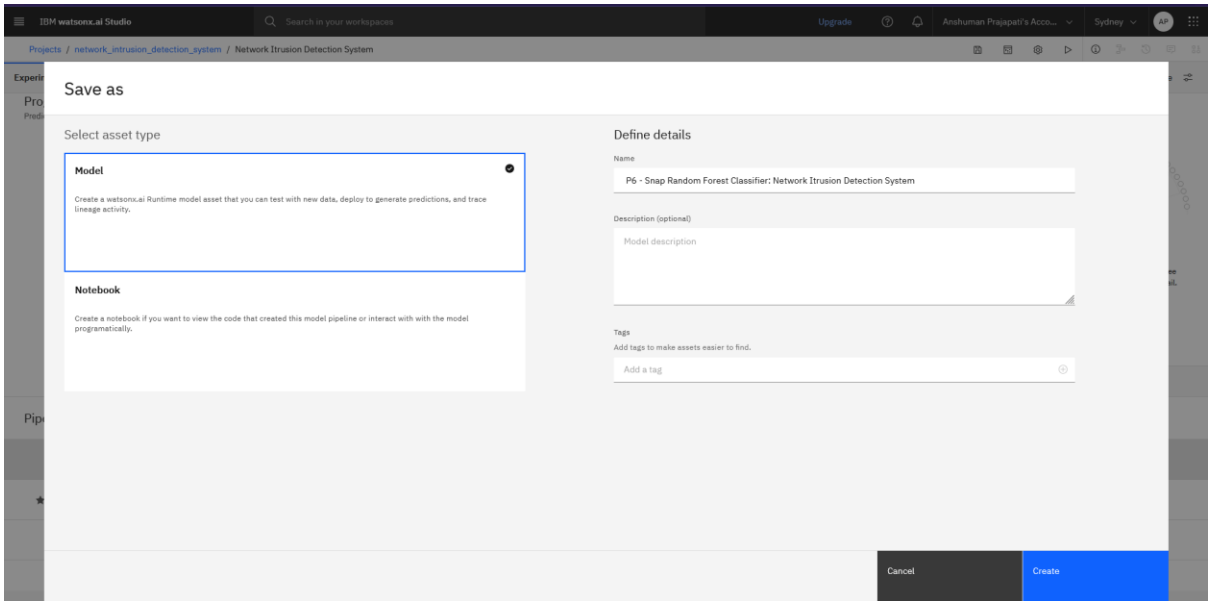
25. Select the prediction type (in this case its class) and make sure the prediction type is multiclass classification(as prediction column contains multiple distinct categories) and rum experiment.



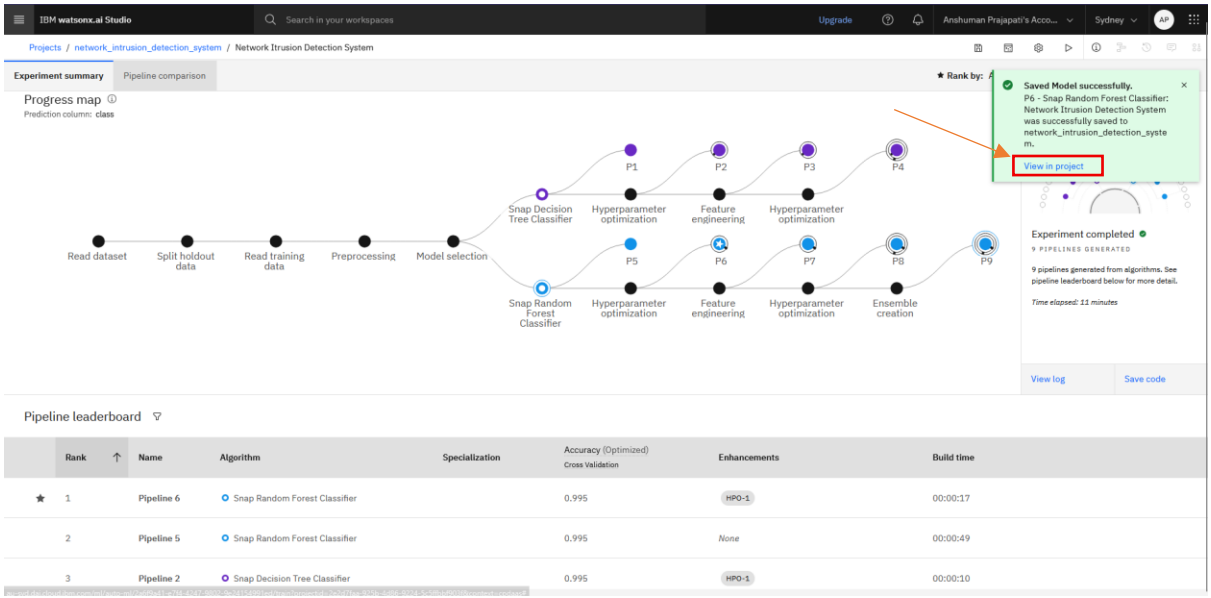
26. It will take some time to perform the experiment.



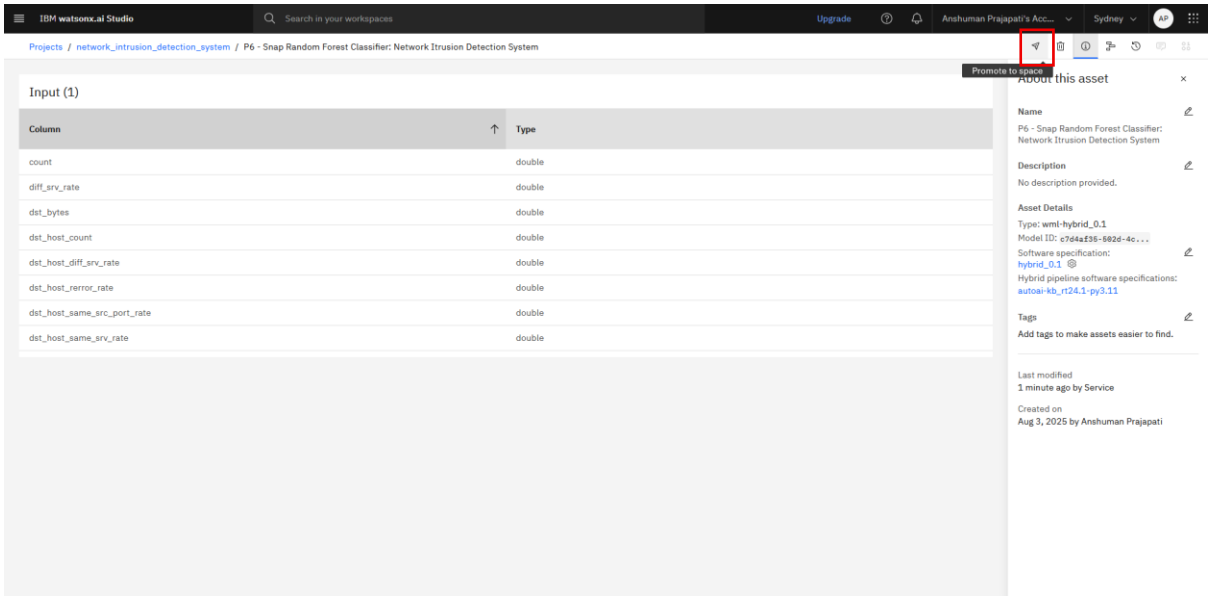
27. Now after the experiment is completed we can see the best performing algorithm is Pipeline 6 with enhancement of hyperparameter optimization now click on save as.



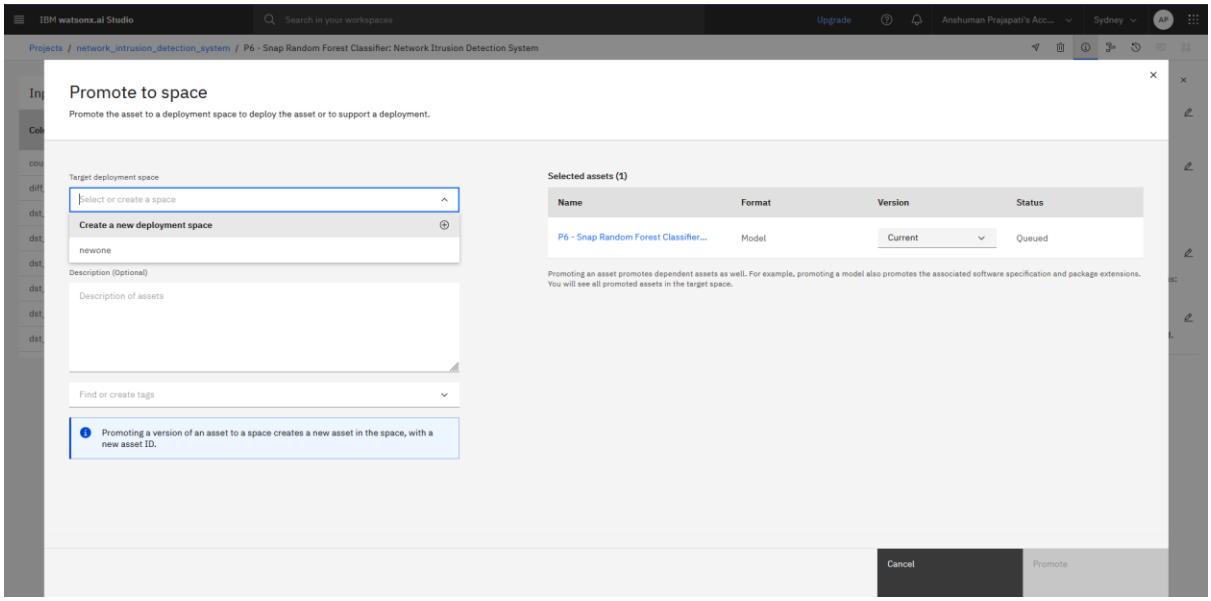
28. Press on create.



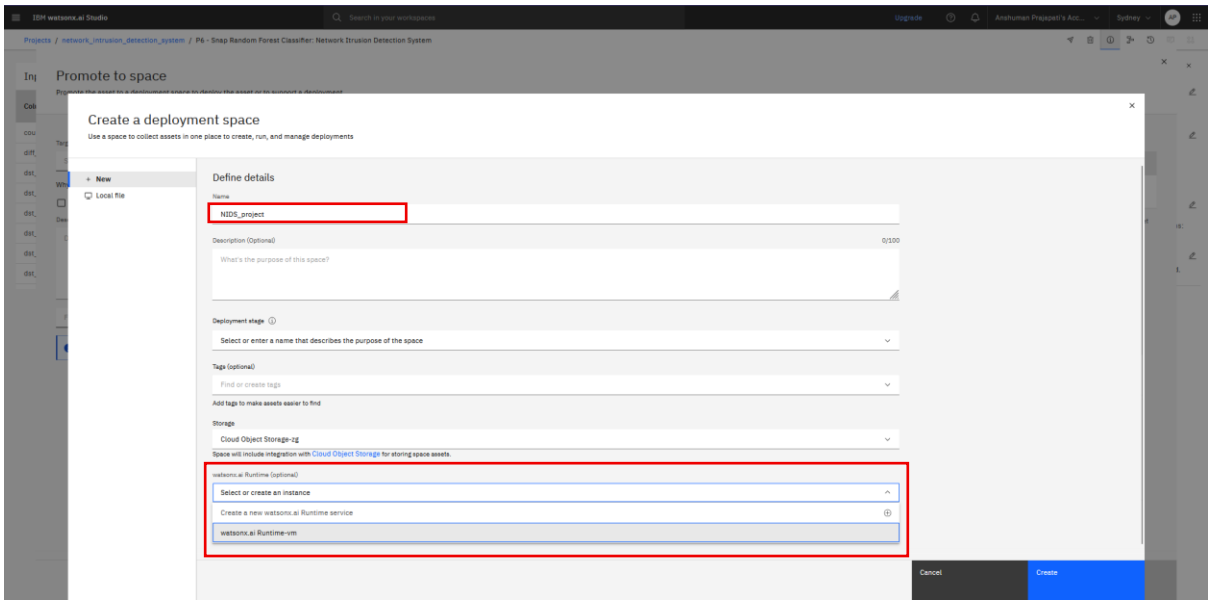
29. After the model is saved click on view in project.



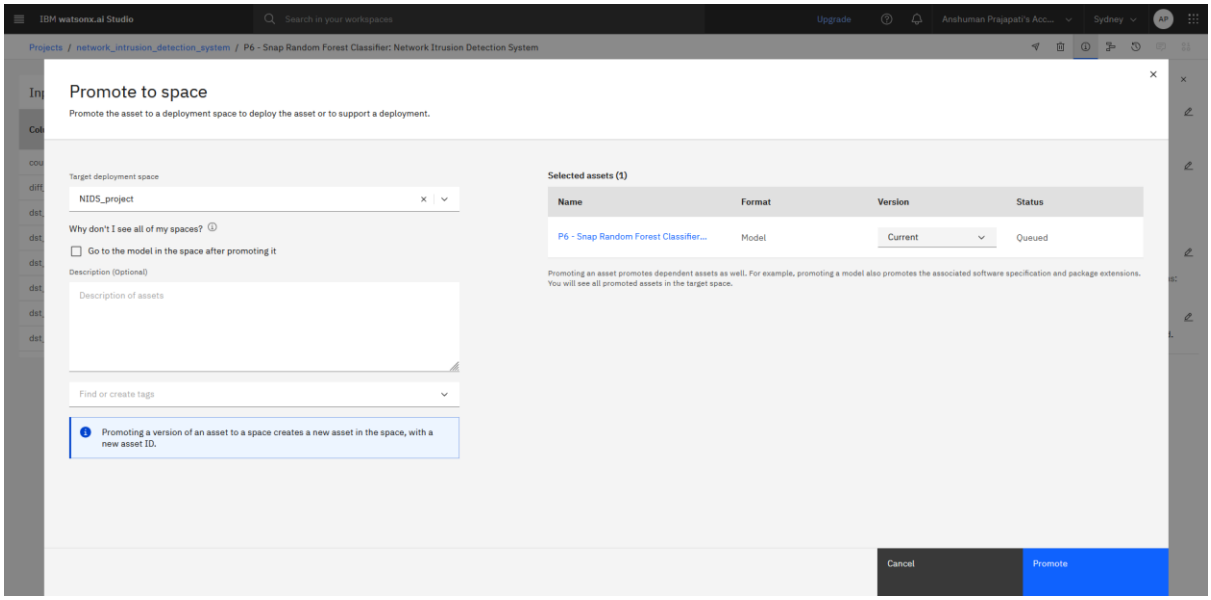
30. Now on the top towards right press on promote to space.



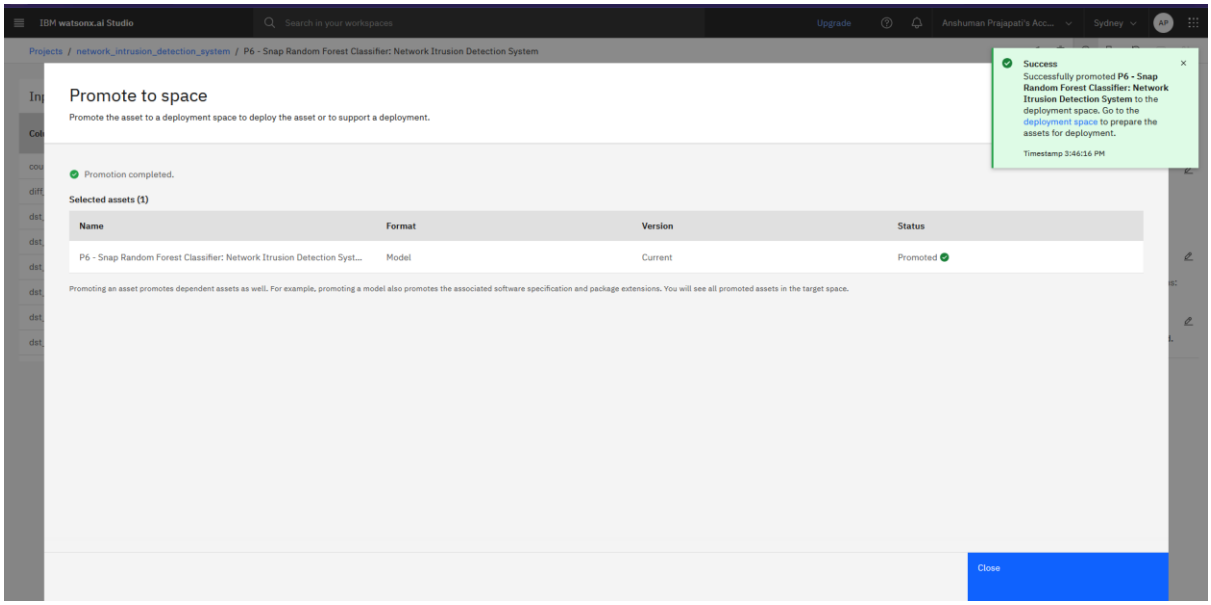
31. Select create new deployment .



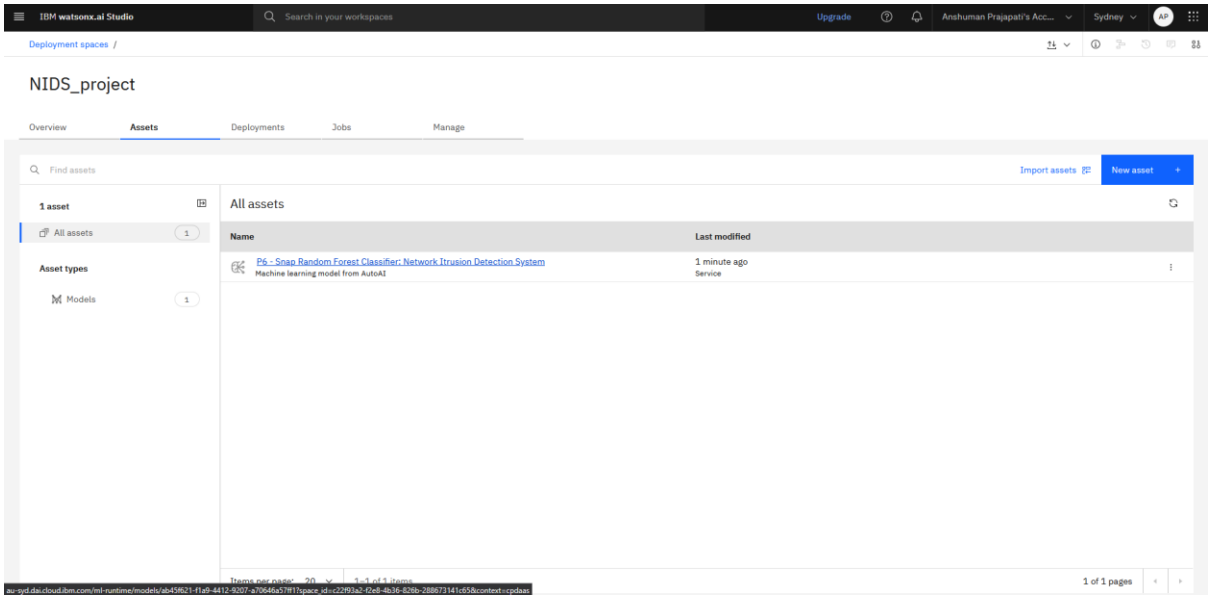
32. Name the deployment space and make sure to select instance of watsonx.ai Runtime and press create.



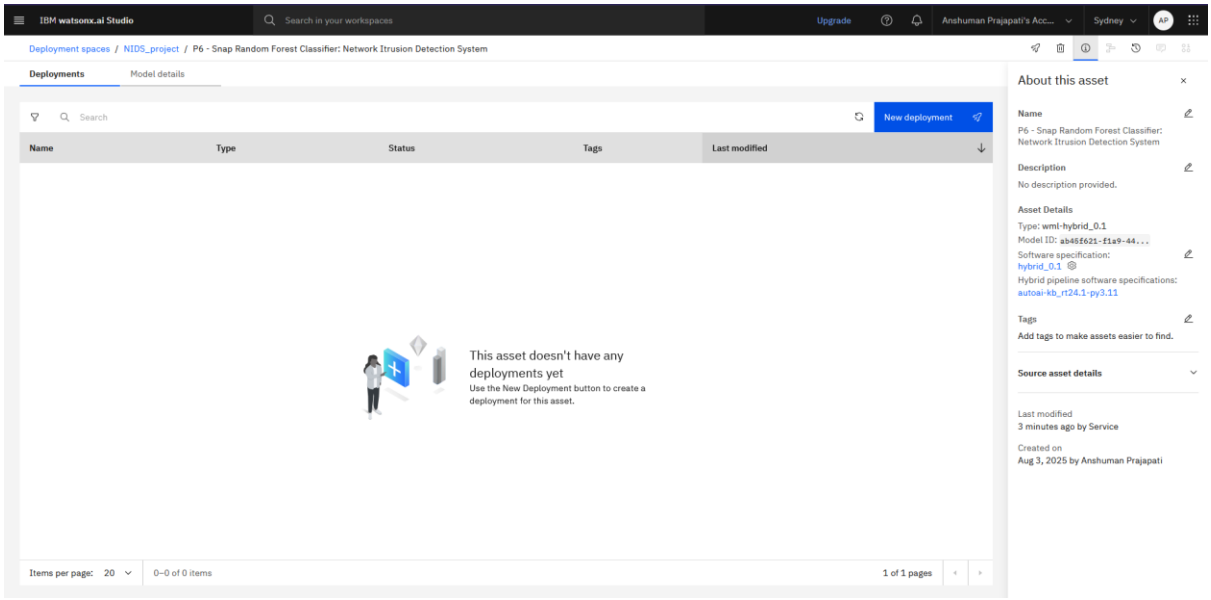
33. Now press on promote.



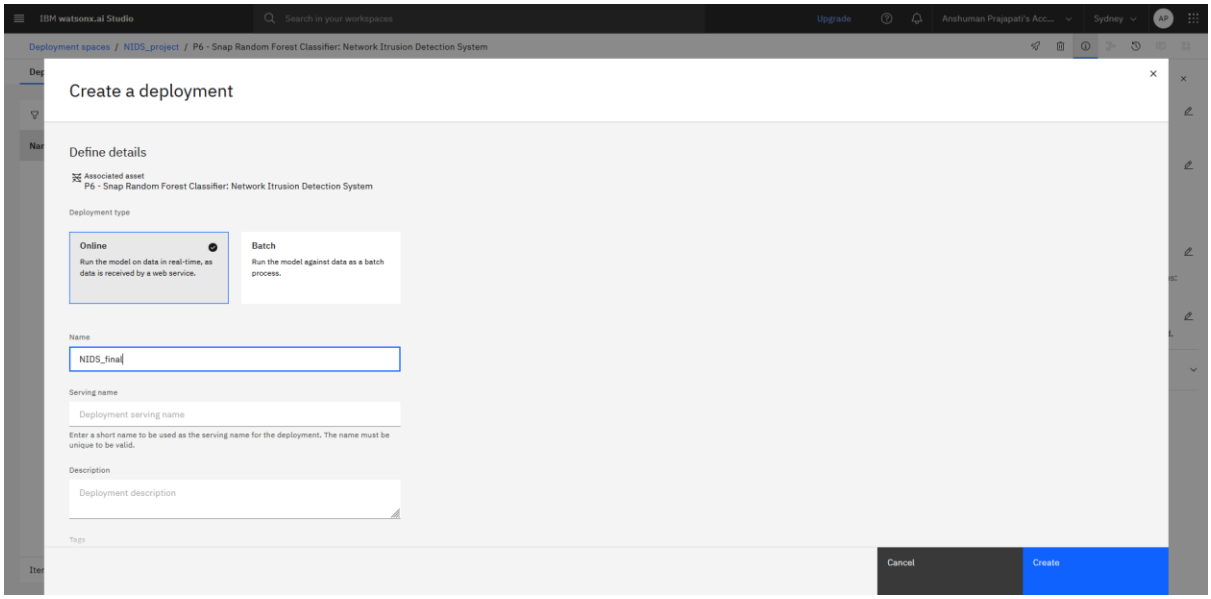
34. Now after it got promoted to space press on deployment space ,the pop up on top right corner.



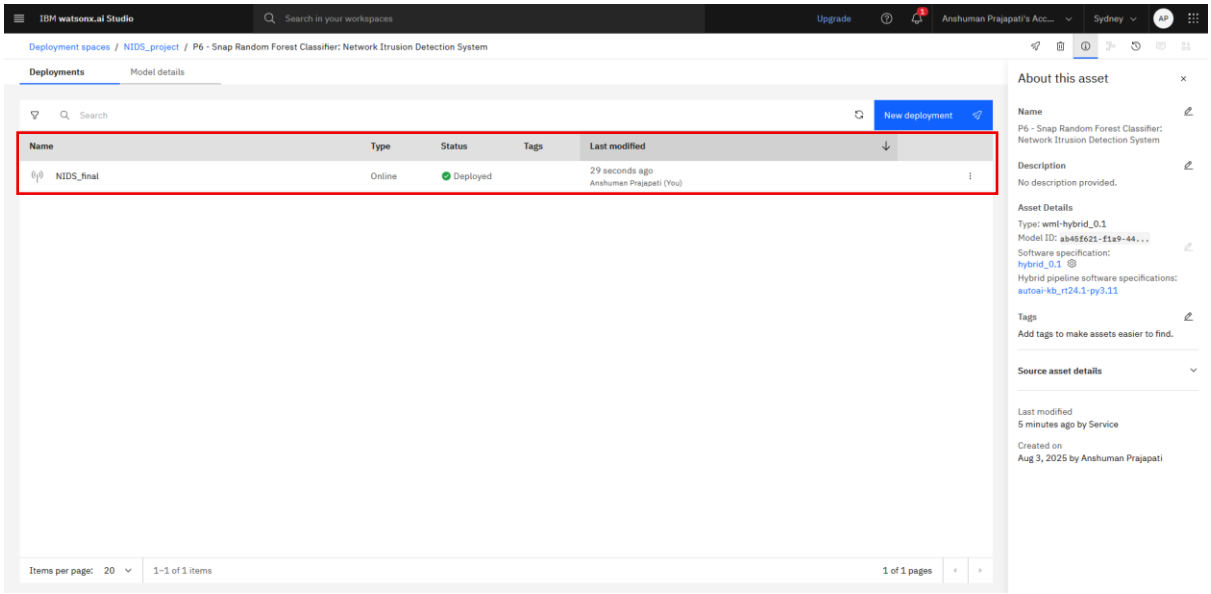
35. Now click on the asset.



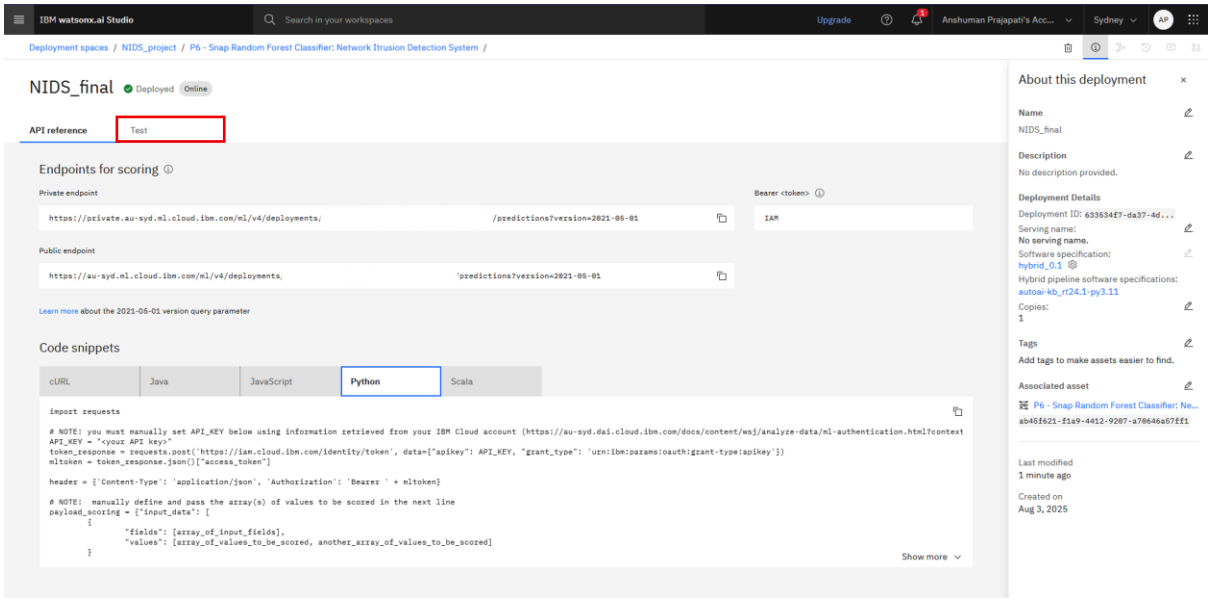
36. Now click on new ***New Deployment***



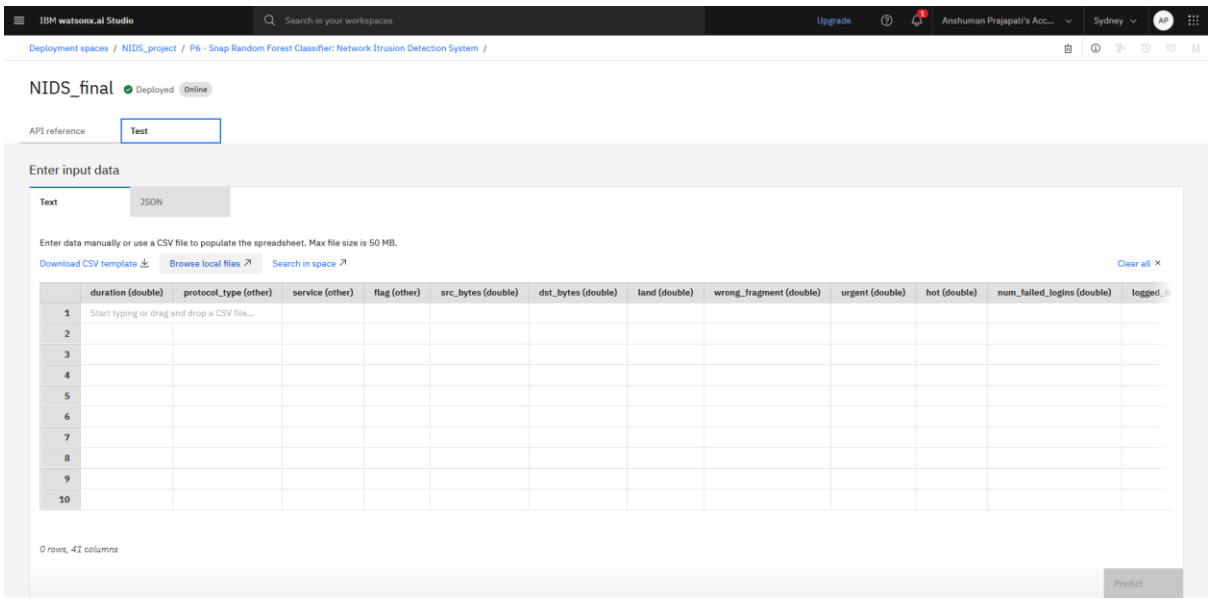
37. Now select online and give this final deployment a name and press create.



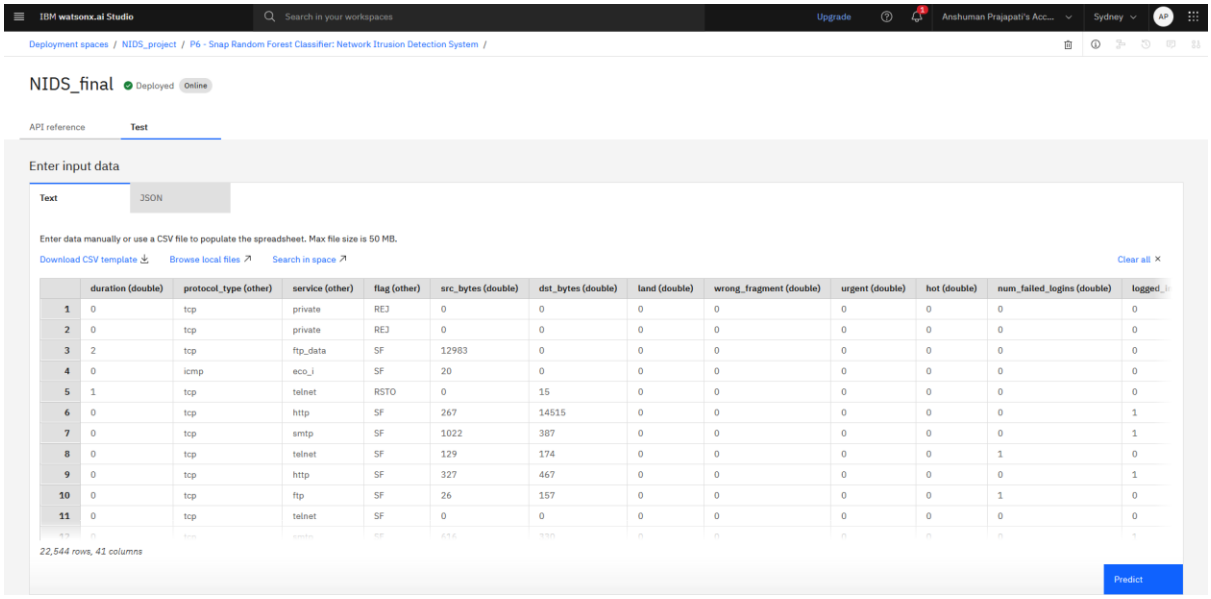
38. After some time the deployment will be completed, then press on the deployment name.



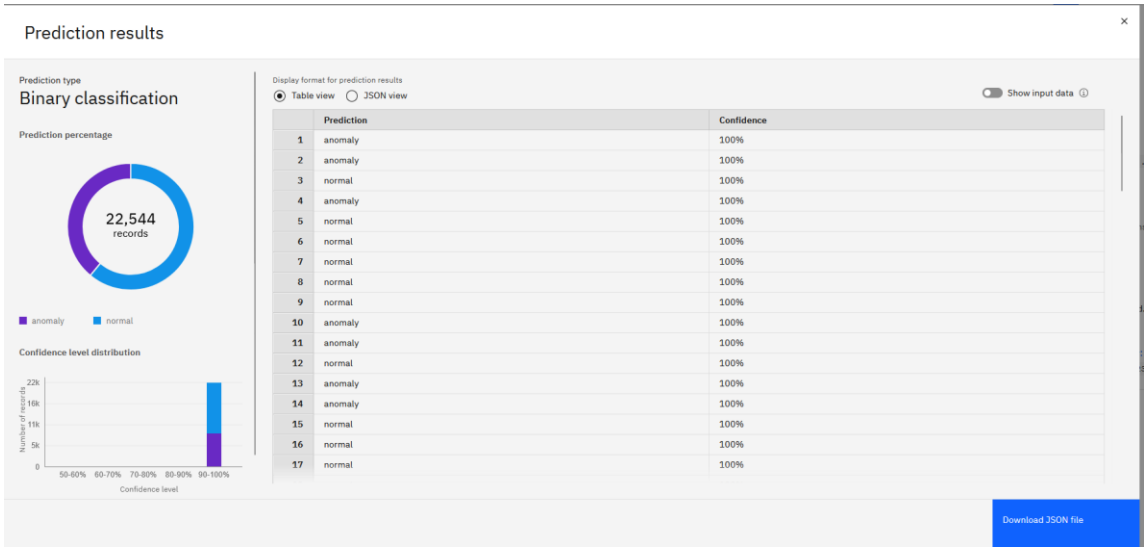
39. This will give the API reference page where API's are present to implement this model with other projects. Now press on test to test the model.



40. We have many options to give input for test ,I choose local file as I have one.



41. After the file is loaded press predict. It will take some time to generate the result.



42. This will generate a final result with up to 100% accuracy.

For upgrade of different types of attack detection just the data(.csv file) on which model is trained is to be changed for new data(Train_data_with_attack_types_enhanced.csv file) provided above and also you can refer this GitHub repo.