National College of Ireland

Configuration Manual

MSc Research Project
Unsupervised learning for Anomaly and Misconfigurations detection of cloud storage services

Gift Maseno
Student ID: x22242601

School of Computing
National College of Ireland

Supervisor: Raza Mustafa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Gift Neema Maseno
Student ID:	x22242601
Programme	MSCCYBE_JANO21 **Year:** 2025
Module:	Msc Thesis
Lecturer:	Raza Mustafa
Submission Due Date:	11 Dec 2025
Project Title:	Configuration Manual
Word Count:	1067 **Page Count:** 10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: 

Date: 9 Dec 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Gift Maseno
Student ID: x22242601

1. Introduction

This piece details the manual for establishing and training an autoencoder model to showcase the proof of concept on how unsupervised machine learning can be used for anomaly detection, primarily investigating insecure S3 buckets. It bridges the gap between the thesis and artifacts by offering an overview of the prerequisite steps

2. Hardware requirements

System type: MacBook Pro 2020
Processor: Apple M1
Memory RAM: 8GB
Storage: 500GB SSD

3. Software requirements

Firstly, Google Colab environment was utilized for the development of this project. Google Colab is an open source hosted Jupyter Notebook environment providing free access to the Python ecosystem computing resources well-suited for machine learning, data science and education.
To begin with, dependencies were installed in the environment using the pip command as illustrated below.

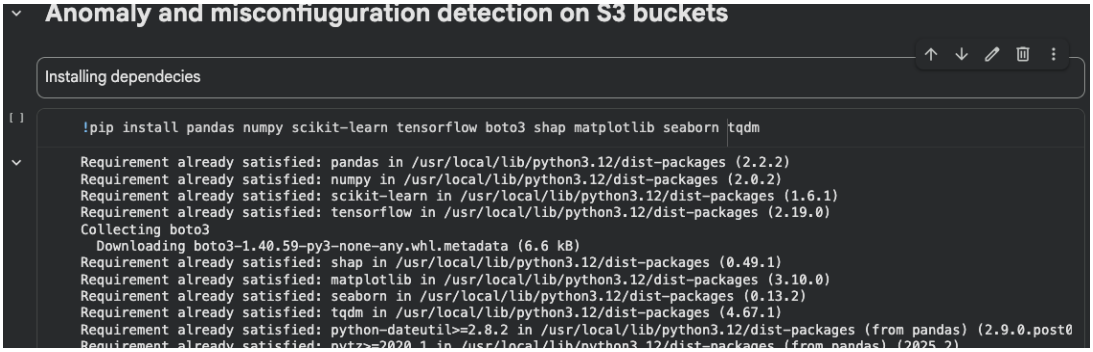


Figure 1: Installing dependencies on Google Colab

Various libraries necessary for the development of the model were then installed. These included:

- **Keras** – an open-source library for deep neural networks
- **TensorFlow-** an open-source machine learning framework for training of neural networks.
- **Pandas**- an open-source library for data manipulation and analysis.
- **Numpy**- an opensource library for mathematical and scientific functions.

- **Matplotlib**- a library for generating static, animated or interactive visualizations.

```
# Importing Libraries
import os, tarfile, gzip, json, random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, roc_auc_score, roc_curve
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.optimizers import Adam
from pathlib import Path
from typing import Any, Dict, Iterable, List, Optional
from tqdm import tqdm
```

Figure 2: Importing of library packages

4. Dataset selection

4.1. Data extraction

This study utilized the flAWS[1] dataset which was sourced from Summit Route. The dataset contained over 1.9 million event entries of anonymized AWS CloudTrail logs. This dataset was chosen as it is current and contained useful S3 bucket features and configuration patterns relevant for this study. The dataset was in a *.tar* archive and this was extracted prior to data manipulation as shown below.

Extracting .tar and finding JSON CloudTrail files

```
#Extracting the tar archive
tar_path = "/content/flaws_cloudtrail_logs.tar"
extract_dir = "/content/extracted_cloudtrail"
os.makedirs(extract_dir, exist_ok=True)

with tarfile.open(tar_path, 'r') as tar:
    tar.extractall(path=extract_dir)

print("Extracted sample files:")
for root, dirs, files in os.walk(extract_dir):
    for i, f in enumerate(files[:10]):
        print("📄", os.path.join(root, f))
    break
```

```
/tmp/ipython-input-168407528.py:7: DeprecationWarning: Python 3.14 will
  tar.extractall(path=extract_dir)
Extracted sample files:
```

Figure 3: Extracting CloudTrail files

[1] Piper, S. (2020) 'Public dataset of Cloudtrail logs from flaws.cloud'. SummitRoute. https://summitroute.com/blog/2020/10/09/public_dataset_of_cloudtrail_logs_from_flaws_cloud/.

```python
# Extracting individual cloudtail events
def _read_text(path: Path) -> str:
    if path.suffix == ".gz" or path.name.endswith(".json.gz"):
        with gzip.open(path, "rb") as f:
            return f.read().decode("utf-8", errors="replace")
    return path.read_text(encoding="utf-8", errors="replace")

def load_cloudtrail_events_from_file(path: Path) -> List[Dict[str, Any]]:
    text = _read_text(path).strip()
    if not text:
        return []
    try:
        obj = json.loads(text)
        if isinstance(obj, dict) and "Records" in obj:
            return [r for r in obj["Records"] if isinstance(r, dict)]
        elif isinstance(obj, dict):
            return [obj]
        elif isinstance(obj, list):
            return [r for r in obj if isinstance(r, dict)]
    except json.JSONDecodeError:
        pass
    # fallback: NDJSON
    events = []
    for line in text.splitlines():
        try:
            j = json.loads(line)
            if isinstance(j, dict):
                events.append(j)
        except json.JSONDecodeError:
            continue
    return events

def iter_all_events(data_dir: Path) -> Iterable[Dict[str, Any]]:
    for path in data_dir.rglob("*"):
        if path.suffix in {".json", ".gz"} or path.name.endswith(".json.gz"):
            for ev in load_cloudtrail_events_from_file(path):
                yield ev

# Helper function for safely extracting dictionary or string values form complex/nested data structures
def _ensure_dict(x: Any) -> Dict[str, Any]:
    if isinstance(x, dict): return x
    if isinstance(x, str):
        try:
            j = json.loads(x)
            return j if isinstance(j, dict) else {}
        except: return {}
    return {}

def _first_str(x: Any) -> Optional[str]:
    if isinstance(x, str): return x
    if isinstance(x, list):
        for v in x:
            if isinstance(v, str): return v
    return None
```

Figure 4: Extracting individual CloudTrail events

4.2. Building bucket-level configuration

Since the dataset was too big, filtering was done to select a sample of the most pertinent features. The rationale for selecting these features was influenced by the report by (TrendMicro, 2021) which highlights the leading causes of misconfigurations in the AWS architecture. The table below highlights the features that were selected for model training.

Feature	**Description**
Publicly_accessible	Represents publicly accessible S3 buckets.
Policy_public	Denotes buckets allowing public access via “Principal”: “*”
Logging_enabled	Checks whether logging is enabled.
Public_access_block_enabled	Checks if AWS Public Access Block is enabled
Acl_insecure	Identifies misconfigured ACLs granting public read/write permissions.
Encryption_enabled	Checks whether server-side encryption (SSE) is enabled.
Versioning_enabled	Checks whether versioning is enabled to prevent overwriting of data.
Cors_unrestricted	Identifies unrestricted configurations that could allow requests regardless of the origin.

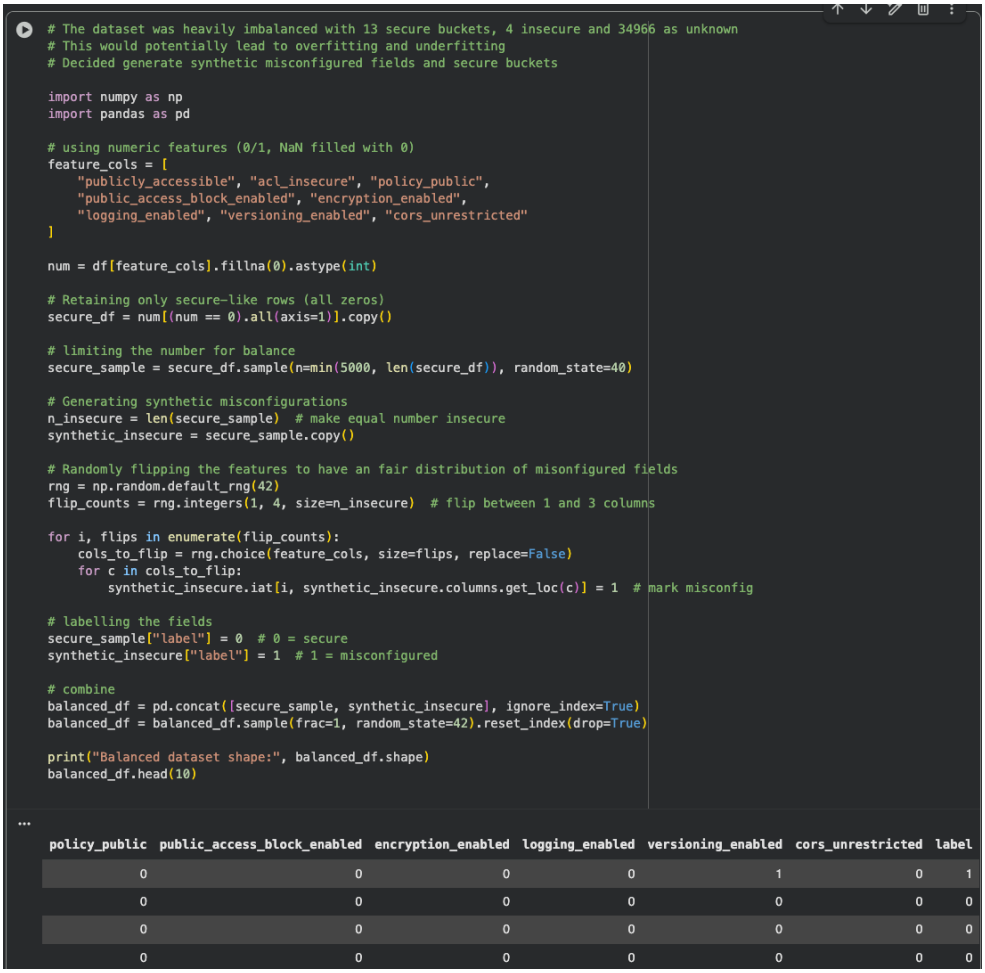
Table 1: S3 related record features used for model training

```python
# Establishing S3 feature rules to be detected
# ----------------------------------------
# Checking for insecure ACL configurations like piblic read/write
def flag_acl_insecure(ev):
    if ev.get("eventName") not in ("PutBucketAcl","PutObjectAcl"): return None
    rp=_ensure_dict(ev.get("requestParameters"))
    canned=rp.get("ACL") or rp.get("x-amz-acl") or rp.get("CannedAcl")
    if isinstance(canned,str) and canned.lower() in {"public-read","public-read-write"}:
        return True
    acp=rp.get("AccessControlPolicy") or rp.get("acl")
    if isinstance(acp,dict):
        aclist=acp.get("AccessControlList") if isinstance(acp.get("AccessControlList"),dict) else None
        grants=(aclist or {}).get("Grant")
        if isinstance(grants,list):
            for g in grants:
                grantee=(g or {}).get("Grantee") or (g or {}).get("grantee")
                if isinstance(grantee,dict):
                    uri=grantee.get("URI") or grantee.get("uri") or ""
                    if "AllUsers" in uri or "AuthenticatedUsers" in uri:
                        return True
    return False

# Checking for public buckets
def flag_policy_public(ev):
    if ev.get("eventName")!="PutBucketPolicy": return None
    rp=_ensure_dict(ev.get("requestParameters"))
    pol=rp.get("policy") or rp.get("Policy")
    if isinstance(pol,str):
        try: pol=json.loads(pol)
        except: return None
    if not isinstance(pol,dict): return None
    stmts=pol.get("Statement")
    stmts=stmts if isinstance(stmts,list) else [stmts]
    for s in stmts:
        if not isinstance(s,dict): continue
        p=s.get("Principal")
        if p=="*" or (isinstance(p,dict) and "*" in p.values()): return True
    return False

# Checking whether public access is blocked or not
def flag_public_access_block_enabled(ev):
    if ev.get("eventName")!="PutPublicAccessBlock": return None
    rp=_ensure_dict(ev.get("requestParameters"))
    cfg=rp.get("PublicAccessBlockConfiguration") or rp.get("publicAccessBlockConfiguration")
    if not isinstance(cfg,dict): return None
    keys=("BlockPublicAcls","IgnorePublicAcls","BlockPublicPolicy","RestrictPublicBuckets")
    return all(bool(cfg.get(k)) for k in keys)

# Checking whether encryption exists
def flag_encryption(ev):
    if ev.get("eventName")=="PutBucketEncryption": return True
    if ev.get("eventName")=="DeleteBucketEncryption": return False
    return None

# Checking if logging is enabled
def flag_logging(ev):
    name=ev.get("eventName")
    rp=_ensure_dict(ev.get("requestParameters"))
```

Figure 5: Building bucket-level configuration states

4.3. Synthetic data generation

The resulting dataset was highly imbalanced showing only 13 secure buckets and 4 as insecure while 34966 buckets had unknown states. In order to have a more balanced dataset, rule-based data transformation was done to address the scarcity of misconfigured fields. The configuration attributes were encoded to binary form and randomly flipped to either 0 or 1, with 0 representing secure attributes and 1 for insecure. A random state of 40 with seed sample of 5000 were utilized for reproducibility.



```python
# The dataset was heavily imbalanced with 13 secure buckets, 4 insecure and 34966 as unknown
# This would potentially lead to overfitting and underfitting
# Decided generate synthetic misconfigured fields and secure buckets

import numpy as np
import pandas as pd

# using numeric features (0/1, NaN filled with 0)
feature_cols = [
    "publicly_accessible", "acl_insecure", "policy_public",
    "public_access_block_enabled", "encryption_enabled",
    "logging_enabled", "versioning_enabled", "cors_unrestricted"
]

num = df[feature_cols].fillna(0).astype(int)

# Retaining only secure-like rows (all zeros)
secure_df = num[(num == 0).all(axis=1)].copy()

# limiting the number for balance
secure_sample = secure_df.sample(n=min(5000, len(secure_df)), random_state=40)

# Generating synthetic misconfigurations
n_insecure = len(secure_sample)  # make equal number insecure
synthetic_insecure = secure_sample.copy()

# Randomly flipping the features to have an fair distribution of misonfigured fields
rng = np.random.default_rng(42)
flip_counts = rng.integers(1, 4, size=n_insecure)  # flip between 1 and 3 columns

for i, flips in enumerate(flip_counts):
    cols_to_flip = rng.choice(feature_cols, size=flips, replace=False)
    for c in cols_to_flip:
        synthetic_insecure.iat[i, synthetic_insecure.columns.get_loc(c)] = 1  # mark misconfig

# labelling the fields
secure_sample["label"] = 0  # 0 = secure
synthetic_insecure["label"] = 1  # 1 = misconfigured

# combine
balanced_df = pd.concat([secure_sample, synthetic_insecure], ignore_index=True)
balanced_df = balanced_df.sample(frac=1, random_state=42).reset_index(drop=True)

print("Balanced dataset shape:", balanced_df.shape)
balanced_df.head(10)
```

policy_public	public_access_block_enabled	encryption_enabled	logging_enabled	versioning_enabled	cors_unrestricted	label
0	0	0	0	1	0	1
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Figure 6: Synthetic data generation

From a visualization standpoint, the co-occurrence heatmap highlights the frequency of certain misconfigurations and how they relate with other common S3 bucket misconfigurations. Certain misconfigurations frequently occur together, for instance, publicly accessible buckets would often lack encryption and logging capabilities which points to poor systemic configuration or weak policy enforcement.

These patterns provide valuable insights with regard to prioritizing security interventions, as addressing one misconfiguration (e.g., enforcing encryption) could potentially reduce the likelihood of linked vulnerabilities. Therefore, the analysis emphasizes the need for integrated monitoring methods that can identify and remediate multiple misconfigurations concurrently instead of treating misconfigurations as isolated instances.

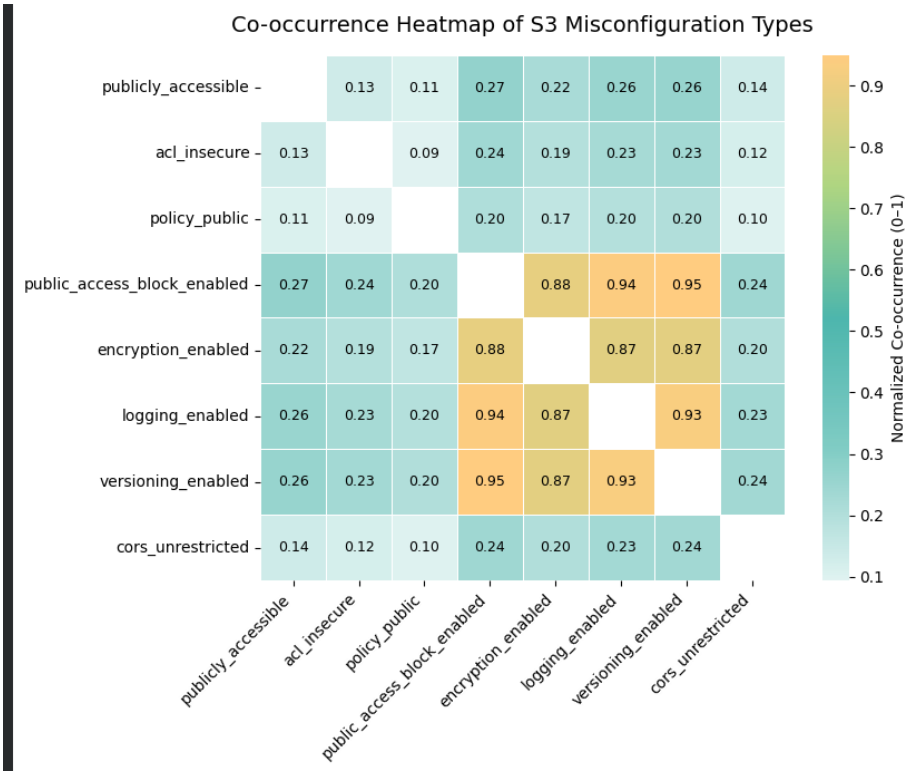


Figure 7: Co-occurrence heatmap of S3 Misconfigurations

4.4. Model development

The dataset was then split into a train and test set using a 70:30 ratio with 70% of the samples being used for training the model while 30% were withheld for testing. This ratio was utilized as it allows the model sufficient data to learn from without overfitting. The *StandardScaler* was used for neural convergence so that features with higher numeric range do not dominate the model's learning process.

```
# Split & Scale
X_train_full, X_test, y_train_full, y_test = train_test_split(
    X_all, y_all, test_size=0.3, random_state=42, stratify=y_all
)

# Train Autoencoder on normals only
X_train_norm = X_train_full[y_train_full == 0]

# Fit scaler on the FULL set (gives variance), then transform
scaler = StandardScaler()
X_all_s   = scaler.fit_transform(X_all)
X_train_s = scaler.transform(X_train_norm)
X_test_s  = scaler.transform(X_test)

print(f"Train normals: {X_train_s.shape}, Test mixed: {X_test_s.shape}")
```

```
Train normals: (4391, 8), Test mixed: (3382, 8)
```

Figure 8: Train-test-split

The autoencoder was then defined consisting of eight neurons, two dense layers, Gaussian noise, a decoder and output layer for linear activation for the reconstruction of standardized outputs.

```
Model: "functional"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 8)	0
gaussian_noise (GaussianNoise)	(None, 8)	0
dense (Dense)	(None, 8)	72
dense_1 (Dense)	(None, 4)	36
dense_2 (Dense)	(None, 8)	40
dense_3 (Dense)	(None, 8)	72

```
Total params: 220 (880.00 B)
Trainable params: 220 (880.00 B)
Non-trainable params: 0 (0.00 B)
```

Figure 9: Definition of the Autoencoder

Model evaluation

```python
# Confusion matrix heatmap
plt.figure(figsize=(5,4))
sns.heatmap(cm,
            annot=True, fmt='d', cmap='Greens',
            xticklabels=['Predicted Secure', 'Predicted Misconfig'],
            yticklabels=['Actual Secure', 'Actual Misconfig'])
plt.title('Confusion Matrix (Autoencoder Anomaly Detection)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.tight_layout()
plt.show()
```

```
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Chosen threshold (target FPR ≤ 1.0%): 0.001003

Confusion matrix (raw counts):
[[1878    4]
 [ 347 1153]]

Classification Report:
              precision    recall  f1-score   support

           0      0.844     0.998     0.915      1882
           1      0.997     0.769     0.868      1500

    accuracy                          0.896      3382
   macro avg      0.920     0.883     0.891      3382
weighted avg      0.912     0.896     0.894      3382
```

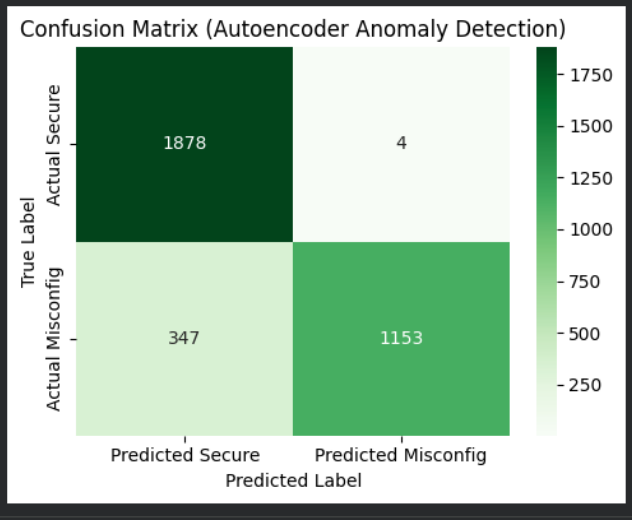


Figure 10: Confusion Matrix

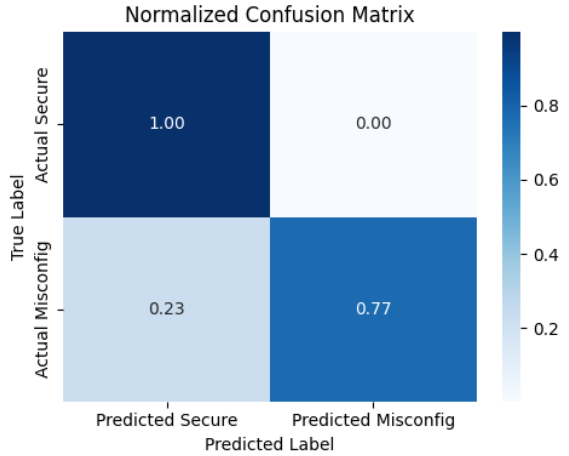


Figure 11: Normalized confusion matrix- based on the whole dataset

References

TrendMicro (2021) "The most common cloud misconfigurations that could lead to security breaches," *Trend Micro*, 25 October. https://www.trendmicro.com/vinfo/ie/security/news/virtualization-and-cloud/the-most-common-cloud-misconfigurations-that-could-lead-to-security-breaches.