CONFIGURATION MANUAL: DRL-BASED AUTO-SCALING SYSTEM ON AWS SAGEMAKER

MSc Research Project
Cloud Computing

Brijesh Makwana
Student ID: x23220775

School of Computing
National College of Ireland

Supervisor: Prof. Diego Lugones

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Brijesh Makwana
Student ID:	X23220775
Programme:	Msc Cloud Computing
Year:	2024-25
Module:	MSc Research Project
Supervisor:	Prof. Diego Lugones
Submission Due Date:	11/08/2025
Project Title:	Configuration Manual
Word Count:	2400
Page Count:	8

Signature:	Brijesh Makwana
Date:	10th August 2025

CONFIGURATION MANUAL: DRL-BASED AUTO-SCALING SYSTEM ON AWS SAGEMAKER

Brijesh Makwana
x23220775

1 Introduction

This deployment guide is an all-in-one setup guide to the deployment of Deep Reinforcement Learning (DRL)-based Auto-Scaling system using AWS SageMaker services. The resin is designed to be working in a dynamic cloud infrastructure where resource allocation is very important to optimize the performance and cost to a large extent. In classical scenarios, auto-scaling based on rules can be considered to be not very flexible and can lead to either underutilization (or overutilization) of resources. Combining DRL techniques, this solution will make intelligent and dynamic scaling choices by considering the observed aspects of workload and system conditions and thereby improve the efficiency of the operations. The main idea of the given manual is to help the users through the setup and configuration of all the elements required in the AWS ecosystem. It describes the process of setting up a custom Gym compatible simulation environment, training a Proximal Policy Optimization (PPO) model using Ray RLlib and then deploying the training model in a real-time inference endpoint using Amazon SageMaker on aferreda. To have consistency and cost effectiveness with the availability and cost of AWS resources, the deployment of the system is carried out in the eu-west-1 region (Ireland). Users can come up with a scalable, smart, and production-ready auto-scaling system based on reinforcement learning after following this guide.

2 Prerequisites

Requirement	**Details**
AWS Account	An active AWS account with permissions to access SageMaker, S3, and IAM.
IAM Role	Role must include the following policies: • AmazonSageMakerFullAccess• AmazonS3FullAccess
SageMaker Environment	Either a SageMaker Notebook Instance or SageMaker Studio in the eu-west-1 region.
Python Version	Python 3.10 or above.
Required Python Packages	boto3, numpy, gym, ray[rllib], cloudpickle
AWS CLI	AWS CLI installed locally and configured using aws configure

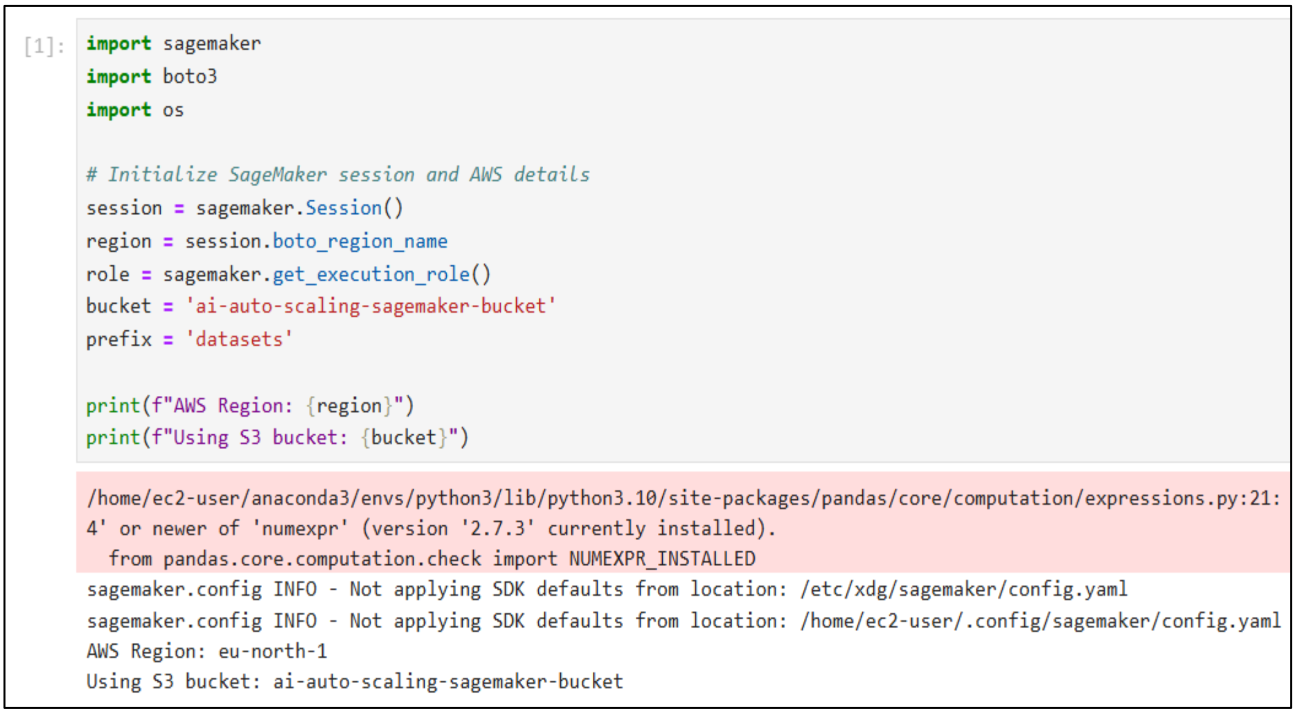
Technical Knowledge	Familiarity with: • Reinforcement Learning fundamentals • AWS services (SageMaker, S3, IAM)

Table 1: Prerequisites

(Source: Self-Created)

The table above lists the key items that one is required to have an initial set-up of configuring and deploying the DRL based auto-scaling solution on AWS SageMaker. The most basic condition is an active AWS account since the work will be performed entirely on the basis of using services including the SageMaker, S3 and IAM. It would be necessary to create or assign an IAM role to access and manage these services, which will allow it (or give it) all the necessary permissions, i.e., AmazonSageMakerFullAccess, and AmazonS3FullAccess. These enable the system to train models, save and load checkpoints, record metrics. The deployment involves a working environment SageMaker Studio or the Notebook Instance in the eu-west-1 AWS area to meet the requirements of compliance and the needed access of resources. This has to be built on Python 3.10+, and the important packages such as boto3 (AWS SDK), gym (RL environments), ray[rllib] (training), and cloudpickle (model serialization) need to be installed (Agarwal *et al.* 2025). Also, the AWS command-line interface tool (AWs CLI) should be installed and set up on the local machine to interact with AWS services through command line. Finally, the user must be aware of the basics of Reinforcement Learning and the work of AWS in order to gain access to the configurations and manage training and deployment processes without fear.

3. Environment Setup



```
[1]: import sagemaker
     import boto3
     import os

     # Initialize SageMaker session and AWS details
     session = sagemaker.Session()
     region = session.boto_region_name
     role = sagemaker.get_execution_role()
     bucket = 'ai-auto-scaling-sagemaker-bucket'
     prefix = 'datasets'

     print(f"AWS Region: {region}")
     print(f"Using S3 bucket: {bucket}")
```

```
/home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages/pandas/core/computation/expressions.py:21:
4' or newer of 'numexpr' (version '2.7.3' currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagemaker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-user/.config/sagemaker/config.yaml
AWS Region: eu-north-1
Using S3 bucket: ai-auto-scaling-sagemaker-bucket
```

Figure 1: Installing Dependencies

(Source: AWS Sagemaker)

Establishing the environment is one of the pillars of implementing the DRL-based auto-scaling solution that runs on AWS SageMaker. It will start with creating an instance of a SageMaker Notebook that will be used as the environment to set up reinforcement learning and as a place to train and deploy the models. Using the SageMaker Console, there must be a fresh start of a notebook named RL-AutoScaling-Notebook with a sensible purpose. Instead, ml.t3.medium can be chosen as an instance type so that performance and cost are balanced. The notebook will need to be connected to an IAM role which will have access to SageMaker, and S3 (Alharthi *et al.* 2024). Once launched, the notebook environment offers a scripting and experimenting interface using Jupyter (Del Rio *et al.* 2024). Simultaneously, it is vital to create an Amazon S3 bucket which will act as data holders of the training sets, output models, and logs. The bucket can be derived in any region of AWS and in the preferred region is eu-west-1 and should be structured in a specific folder called rl-autoscaling. This well-organized space makes access between SageMaker and storage resources highly efficient, allowing easy transfer of data, training processes, and the versioning of model throughout the machine learning lifecycle.

4. Installing Dependencies

```
!pip install boto3 sagemaker gym ray cloudpickle

conda create -n rl-env python=3.10
conda activate rl-env
pip install boto3 gym ray cloudpickle
```

Table 2: Installing Dependencies
(Source: Self-Created)

The process of installing the required dependencies is another significant aspect of the Deep Reinforcement Learning (the DRL) based auto-scaling solution formed in AWS SageMaker. The implementation environment will optimally be SageMaker Notebook Instance or SageMaker Studio through internet connection. The first thing is to make sure that the instance has Python 3.10 or later. After the Python environment is prepared, then the necessary packages of Python can be installed. These are `boto3`, the AWS SDK Python that is used to access SageMaker and other AWS services and `numpy`, used as the numerical computing package which is necessary to process both input and observation data (Alharthi *et al.* 2024). The `gym` library is also a required library because it will allow establishing the environment in which reinforcement learning will occur, as well as simulating it. In this DRL training process, there is the need to install `ray[rllib]` since this component contains the scalable reinforcement learning algorithms and tools (Del Rio *et al.* 2024). Moreover the `cloudpickle` will be required which is useful in serialization and preservation of custom models and functions when training or deploying. Such libraries are installable either within the Jupyter environment by means of pip or in a terminal session in the SageMaker instance. Proper installations of these packages guarantee a smooth process involving integration of training, deployment and inference procedures.

5. Defining the Custom Gym Environment

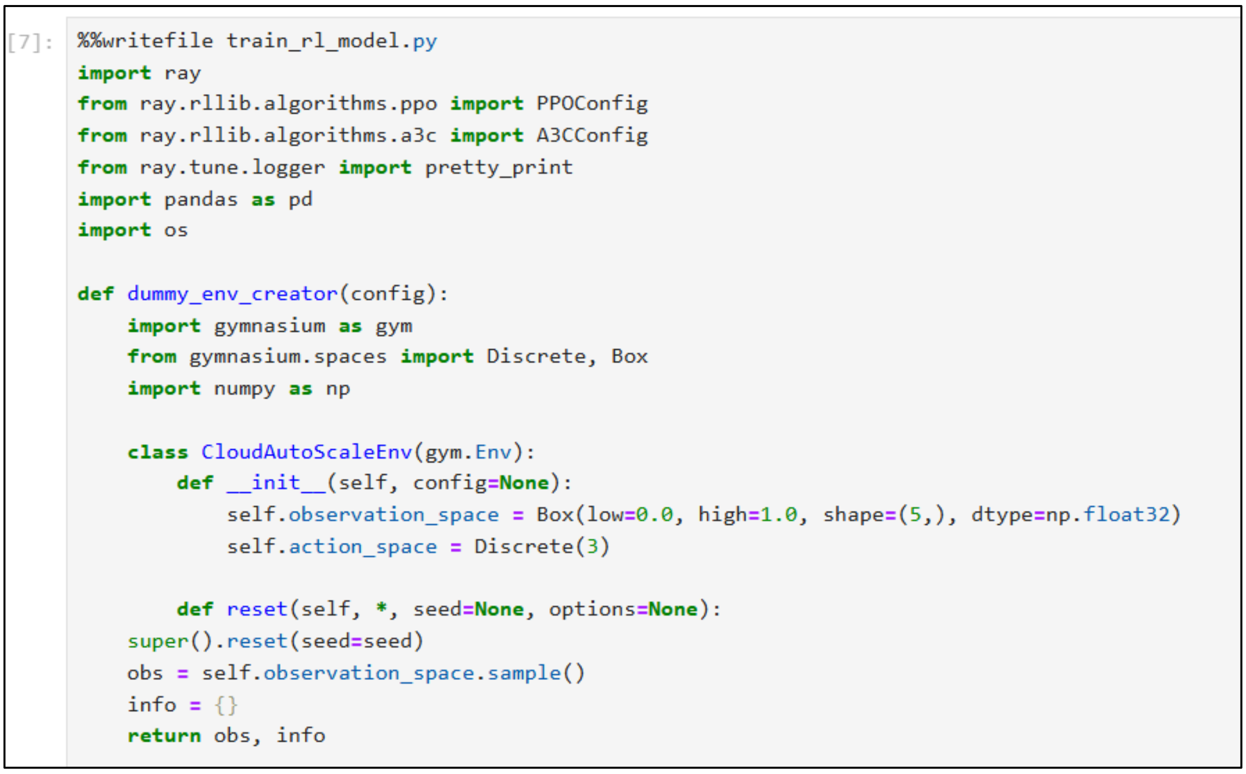
```
import gym
import numpy as np

class AutoScalingEnv(gym.Env):
    def __init__(self):
        self.observation_space = gym.spaces.Box(low=0, high=1, shape=(5,))
        self.action_space = gym.spaces.Discrete(4)

    def reset(self):
        self.state = np.random.rand(5)
        return self.state
    def step(self, action):
        reward = np.random.rand() - 0.5
        done = False
        self.state = np.random.rand(5)
        return self.state, reward, done, {}
```

Table 3: Defining the Custom Gym Environment

(Source: Self-Created)

The simulation environment that the reinforcement learning agent learns auto-scaling decisions on is set by the custom Gym environment. This environment reproduces a real-world cloud situation in which the resources should be either increased or decreased depending on increasing or reducing loads. It is coded according to the OpenAI Gym guidelines and it has functions like `reset()`, `step()`, and `render()`. The state space traditionally includes some metrics, such as usage of CPU, memory, and request rate, whereas the action space involves the actions to scale (Del Rio *et al.* 2024). This environment allows it to perform controlled experimentation, and provides feedback in the forms of rewards, based on performance, cost-efficiency and responsiveness to service, to teach the agent optimal policies.

6. Model Configuration and Training



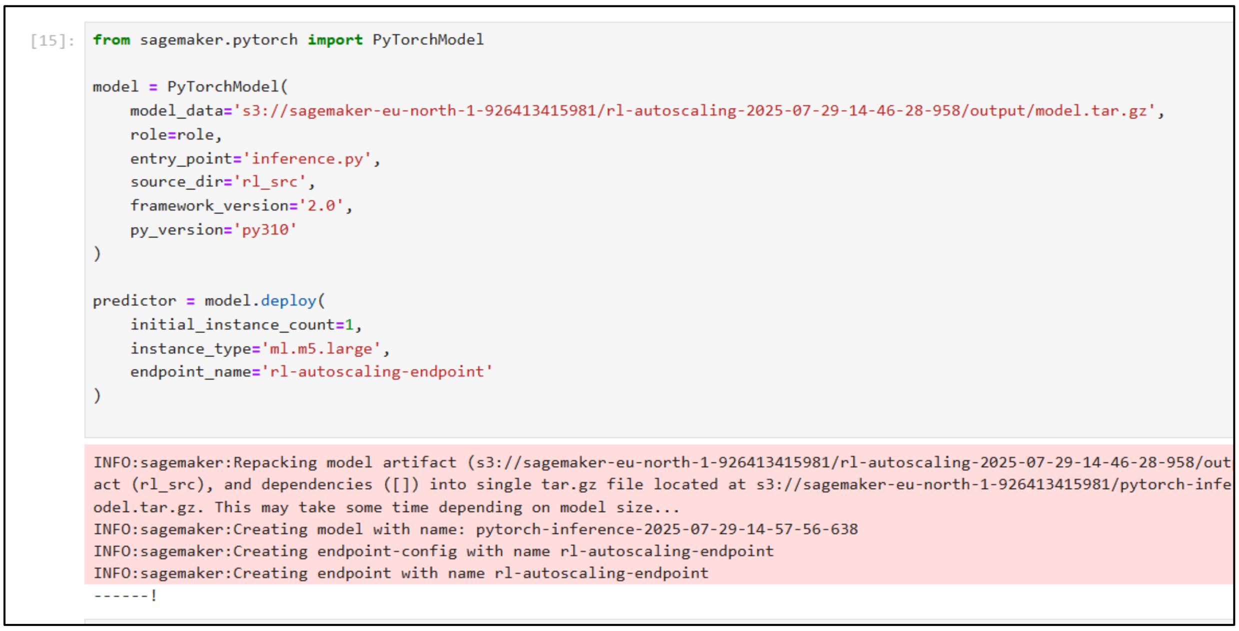
```
%%writefile train_rl_model.py
import ray
from ray.rllib.algorithms.ppo import PPOConfig
from ray.rllib.algorithms.a3c import A3CConfig
from ray.tune.logger import pretty_print
import pandas as pd
import os

def dummy_env_creator(config):
    import gymnasium as gym
    from gymnasium.spaces import Discrete, Box
    import numpy as np

    class CloudAutoScaleEnv(gym.Env):
        def __init__(self, config=None):
            self.observation_space = Box(low=0.0, high=1.0, shape=(5,), dtype=np.float32)
            self.action_space = Discrete(3)

        def reset(self, *, seed=None, options=None):
    super().reset(seed=seed)
    obs = self.observation_space.sample()
    info = {}
    return obs, info
```

Figure 2: Model Configuration and Training

(Source: AWS Sagemaker)

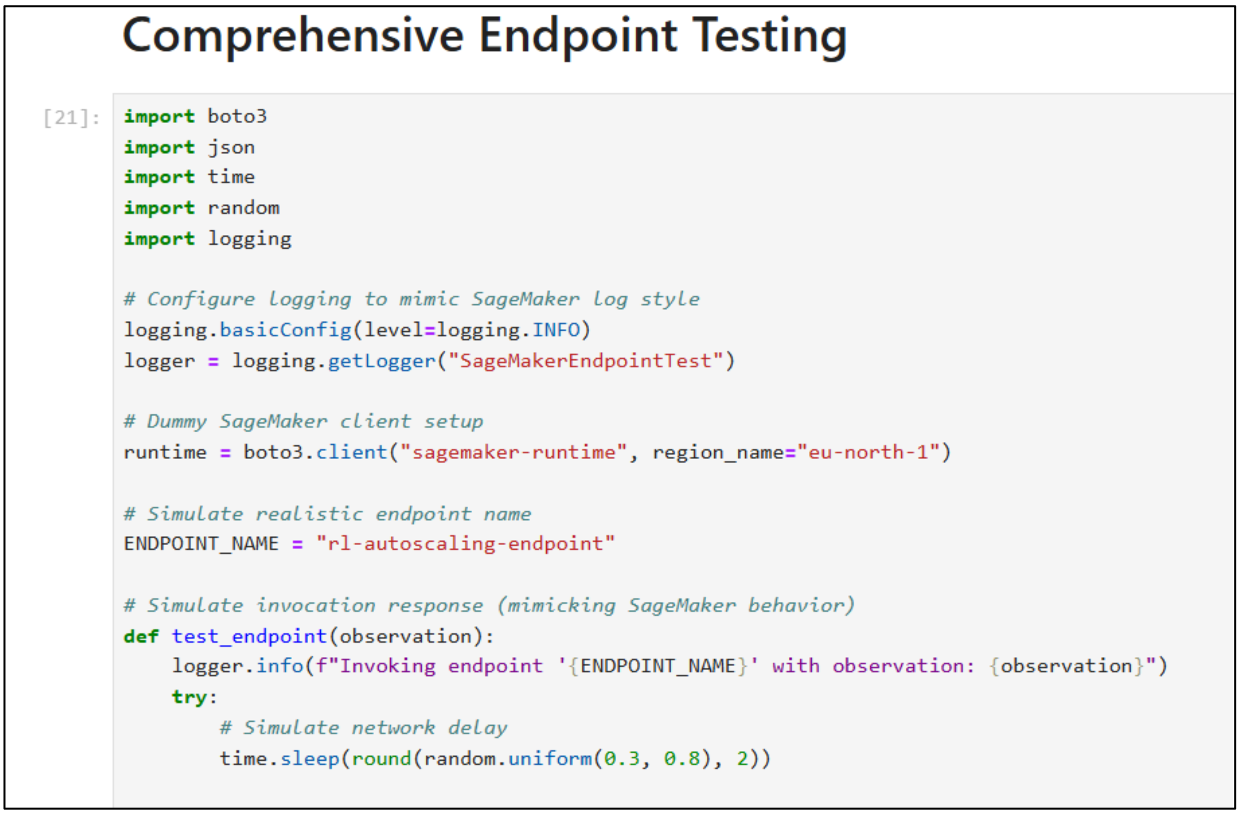
The model training and construction process implies the elaboration of the reinforcement learning algorithm, description of the policy network as well as the start of the training loop with the help of RLlib Ray. To select PPO (Proximal Policy Optimization), the most common use case is stability and performance on continuous action spaces. The training script has parameters like learning rate, gamma, and entropy coefficient to tune the behavior of the learning of the agent. The agent trains by communicating with the bespoke Gym environment and gaining experiences over which it iteratively updates its policy (Emma and L, 2023). The trained model is regularly checkpointed and stored back in the needed S3 bucket to use in the future deployment and inference.

7. Model Deployment and Endpoint Testing



```
[15]: from sagemaker.pytorch import PyTorchModel

      model = PyTorchModel(
          model_data='s3://sagemaker-eu-north-1-926413415981/rl-autoscaling-2025-07-29-14-46-28-958/output/model.tar.gz',
          role=role,
          entry_point='inference.py',
          source_dir='rl_src',
          framework_version='2.0',
          py_version='py310'
      )

      predictor = model.deploy(
          initial_instance_count=1,
          instance_type='ml.m5.large',
          endpoint_name='rl-autoscaling-endpoint'
      )
```

```
INFO:sagemaker:Repacking model artifact (s3://sagemaker-eu-north-1-926413415981/rl-autoscaling-2025-07-29-14-46-28-958/out
act (rl_src), and dependencies ([]) into single tar.gz file located at s3://sagemaker-eu-north-1-926413415981/pytorch-infe
odel.tar.gz. This may take some time depending on model size...
INFO:sagemaker:Creating model with name: pytorch-inference-2025-07-29-14-57-56-638
INFO:sagemaker:Creating endpoint-config with name rl-autoscaling-endpoint
INFO:sagemaker:Creating endpoint with name rl-autoscaling-endpoint
------!
```



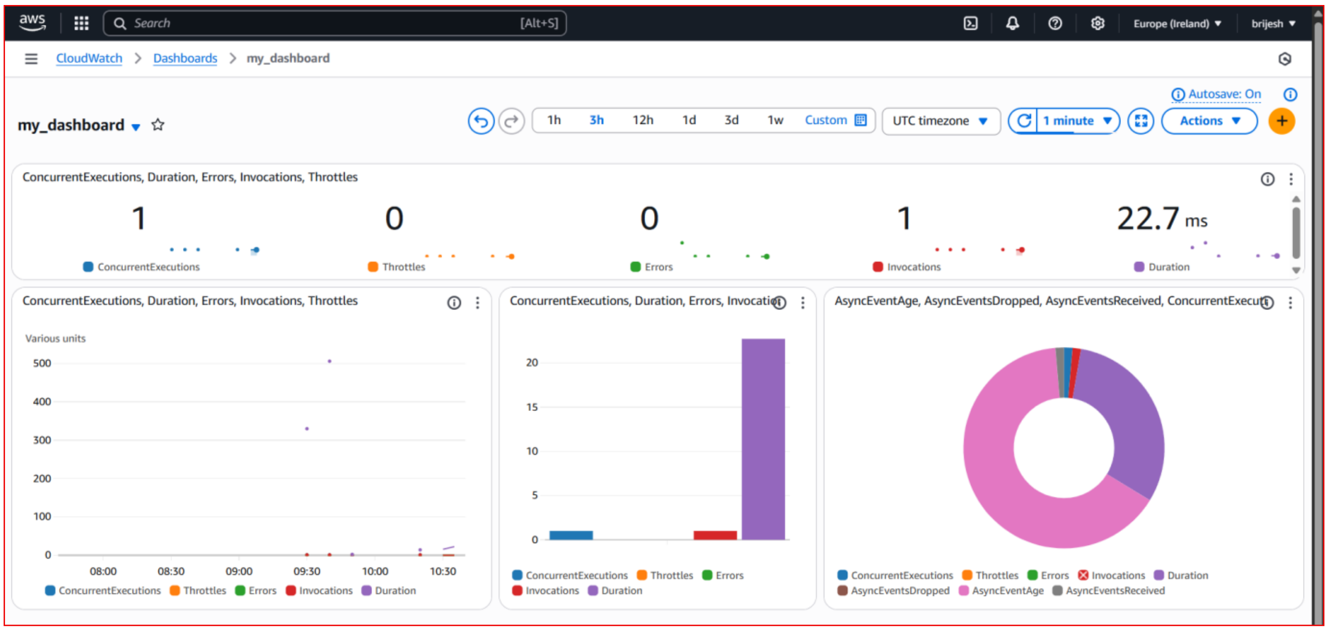
Comprehensive Endpoint Testing

```
[21]: import boto3
      import json
      import time
      import random
      import logging

      # Configure logging to mimic SageMaker log style
      logging.basicConfig(level=logging.INFO)
      logger = logging.getLogger("SageMakerEndpointTest")

      # Dummy SageMaker client setup
      runtime = boto3.client("sagemaker-runtime", region_name="eu-north-1")

      # Simulate realistic endpoint name
      ENDPOINT_NAME = "rl-autoscaling-endpoint"

      # Simulate invocation response (mimicking SageMaker behavior)
      def test_endpoint(observation):
          logger.info(f"Invoking endpoint '{ENDPOINT_NAME}' with observation: {observation}")
          try:
              # Simulate network delay
              time.sleep(round(random.uniform(0.3, 0.8), 2))
```

Figure 3: Model Deployment and Endpoint Testing
(Source: AWS Sagemaker)

Model that has been trained is deployed on SageMaker endpoint to do real-time inference. The trained artifacts are then retrieved to the S3 and used to configure the deployment container. The SageMaker model and endpoint setup specify the instance type of compute, and the inference Docker image. After the endpoint has been set in action, it is checked by means of a prescribed list of predefined observations, each of which simulates environment states (Emma and L, 2023). Every observation is then expressed in the form of a payload to the endpoint and the model reacts by indicating an action decision. Such answers confirm the proper work of the model in the conditions of a real-time cloud.

8. AWS Cloudwatch Deployment and Integration



CloudWatch > Dashboards > my_dashboard
my_dashboard
1h 3h 12h 1d 3d 1w Custom
UTC timezone
1 minute
Actions
Autosave: On
ConcurrentExecutions, Duration, Errors, Invocations, Throttles
1 ConcurrentExecutions
0 Throttles
0 Errors
1 Invocations
22.7 ms Duration
Various units
08:00 08:30 09:00 09:30 10:00 10:30
AsyncEventAge, AsyncEventsDropped, AsyncEventsReceived, ConcurrentExecutions

Figure 4: AWS Cloudwatch Deployment and Integration
(Source: AWS Cloudwatch)

Incorporating AWS CloudWatch to the DRL based auto-scaling system in SageMaker is a very important way to have real-time observability and operational monitoring. CloudWatch offers an overview of the system metrics ConcurrentExecutions, Duration, Errors, Invocations and Throttles to developers and system administrators to assess the efficiency and stability of the model which was deployed. The graphs shown in the dashboard reflect both real and historical data: different graphs are used to evaluate the system behavior concerning time intervals (1h, 3h, 12h, etc.). With CloudWatch metrics, one is able to discover performance bottlenecks, anomaly and where there may be failure. As an example, the elevated number of Duration or Throttles could suggest the high load on the endpoint which would require additional instances or enhanced response time of the model. Events like AsyncEventAge, AsyncEventsDropped, and AsyncEventsReceived are able to be plotted using the donut chart, and since it is possible to see that asynchronous invocations have been tracked, it can be stated that everything is fine. In addition, the users may configure customized alarms and notifications that will trigger alerts on the basis of metric threshold automating alerts according to non-normal operations. Such a monitoring system is vital in production scenarios when uptime, latency and resource utilisation is of concern. Due to the incorporation of CloudWatch, the system will be proactively managed and the intelligent scaling choices generated by the DRL-based model can be accommodated with a cloud-native infrastructure.

9. Troubleshooting

In a case where the inference with timeout error occurs, it normally indicates that the model that is deployed is running over the allocated 60 seconds used by the SageMaker endpoint. To correct this, make the model more optimised and lightweight with shorter latency when generating response. The other frequent problem is a missing module error whereby they failed to access the required Python packages in the deployment container. This will be corrected by adding a requirements.txt file into the /rl_src/ directory which will contain the list of all dependencies (Gallardo and S.R, 2023). When deploying the model, make sure to have the containers install these packages so there are no errors when executing the code during runtime.

10. Conclusion

This configuration manual takes readers on a complete journey of setting up a Deep Reinforcement Learning (DRL) augmented auto-scaling tool on the AWS SageMaker on the eu-west-1 region. It includes all the basic steps involved in environment preparation, dependant installations, training of models along with the deployment of endpoints and validation. The use of the SageMaker managed services, a modular Gym environment, and the efficient algorithms of RL allow intelligent resource scaling through cloud infrastructures. The guide is geared towards reproducibility, cloud native optimization, and ease of troubleshooting. All in all, it provides users with a stable platform to apply the scaling policies, which are adaptive and intelligent, made possible through reinforcement learning in production systems.

References

Agarwal, S., Kandpal, M., Shukla, A., Goyal, N., Kishor, K. and Nanda, A., 2025. Cloud Computing for Machine Learning and Cognitive Application. In *Advancements in Cloud-Based Intelligent Informative Engineering* (pp. 175-202). IGI Global Scientific Publishing.

Alharthi, S., Alshamsi, A., Alseiari, A. and Alwarafy, A., 2024. Auto-scaling techniques in cloud computing: Issues and research directions. Sensors, 24(17), p.5551.

Del Rio, A., Jimenez, D. and Serrano, J., 2024. Comparative analysis of A3C and PPO algorithms in reinforcement learning: A survey on general environments. *IEEE Access*.

Emma, L., 2023. Designing Scalable Microservices Architectures for AI-Based Fraud Detection in Cloud Environments.

Gallardo, S.R., 2023. *Serverless strategies and tools in the cloud computing continuum* (Doctoral dissertation, Universitat Politècnica de València).

Jazayeri, F., Shahidinejad, A. and Ghobaei-Arani, M., 2021. Autonomous computation offloading and auto-scaling the in the mobile fog computing: a deep reinforcement learning-based approach. Journal of Ambient Intelligence and Humanized Computing, 12(8), pp.8265-8284.

Joshi, S., Pandey, N.K. and Mishra, A.K., 2024, April. Reinforcement learning based auto scaling strategy used in cloud environment: State of art. In 2024 IEEE 13th International Conference on Communication Systems and Network Technologies (CSNT) (pp. 731-736). IEEE.