Project Name:

Dynamic scaling of Containers and its parameters based on LSTM RNN based Forecasting

About the Project (Max. 100 words):

The project aims to develop a small scaling engine to effectively scale quantity and parameters of a container/pod inside a kubernetes cluster based on demand forecasting methods using LSTM RNN learning method.

Abstract (Max. 250 words):

Kubernetes, the container orchestrator for cloud-deployed applications, offers automatic scaling for the application provider in order to meet the ever-changing intensity of processing demand. This auto-scaling feature can be customized with a parameter set, but those management parameters are static while incoming Web request dynamics often change. The need is to be proactive about the scaling parameters rather than being reactive to the loads. In case of a cluster containing many containers deployed across multiple servers and CDN’s, it’s wise to dynamically scale these containers to the required memory and compute power based on the trends and seasons of the services needed when it's required to make the overall infrastructure cost effective and efficient. The problem is how to forecast the incoming load dynamically and scale the cluster effectively. The proposed solution consists of a Kubernetes scaling engine (model) that makes the auto-scaling decisions apt for handling the actual variability of incoming requests. In this engine LSTM RNN based deep learning forecast method is used to forecast the demand that may be required by the service.

Expected Outcome (in bullet points):

1. A LSTM RNN Model that can accurately forecast the demand that the application may need
2. Exposure to a Container cluster architecture which is configured to handle dynamic scaling
3. A front end application to effectively view the containers and their images and their configurations and an option to simulate various demands and view the autoscaling feature.

Prior Work (include references):

Paper[1] describes a way to competitively forecast the demand of the service based on multiple machine learning algorithms. This implementation uses various machine learning forecast methods where they compete with each other via a short-term evaluation loop in order to always give the lead to the method that best suits the actual request dynamics. With just a few, but fundamentally different, and competing forecast methods, the auto-scaler engine, implemented

in Kubernetes, resulted in significantly fewer lost requests with just slightly more provisioned resources compared to the default baseline.
Paper[2] In this paper, a comprehensive literature review of existing machine learning-based container orchestration for behavior modeling and prediction of multi-dimensional performance metrics is discussed. Insights to improve the quality of resource provisioning decisions in response to the changing workloads under complex environments is discussed.
Paper[3] This paper presents DeepScaler, a deep learning-based holistic autoscaling approach for microservices that focus on coping with service dependencies to optimize service-level agreements (SLA) assurance and cost efficiency of containers.

[1] C. Meng, S. Song, H. Tong, M. Pan and Y. Yu, "DeepScaler: Holistic Autoscaling for Microservices Based on Spatiotemporal GNN with Adaptive Graph Learning," in 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), Luxembourg, Luxembourg, 2023 pp. 53-65. doi: 10.1109/ASE56229.2023.00038
[2] Zhong, Zhiheng, and Xu, Minxian, et al. "Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions". ACM COMPUTING SURVEYS, vol.54,no.10S, 2022, pp. 35-. doi:10.1145/3510415
[3] L. Toka, G. Dobreff, B. Fodor and B. Sonkoly, "Machine Learning-Based Scaling Management for Kubernetes Edge Clusters," in IEEE Transactions on Network and Service Management, vol. 18, no. 1, pp. 958-972, March 2021, doi: 10.1109/TNSM.2021.3052837.

<u>Problem formulation (in bullet points):</u>

1. How to forecast the demand requirements of services based on previous demands and trends proactively.
2. How to effectively use these forecast demands and manage the parameters of multiple containers and orchestrate their demands

<u>Proposed Solution (Numbered, in sequence):</u>

1. Use of LSTM RNN based forecasting methods to forecast demand of the memory, compute power or images of the containers.
2. Configuring these forecasting engines inside a cluster environment to efficiently scale the cluster as per real time requirements.

<u>Deliverables (Numbered):</u>

1. **Preprocessing:** Preprocess the data (here e-commerce traffic) to identify type of requests, apply appropriate time formatting, what kind and amount of containers and nodes will be used (can be extracted based on the kind of requests that are being sent) and store in MySQL. Due to the size of the dataset of 3.5GB directly storing using pandas in a limited ram of 8GB can lead to resource bottleneck. Instead my approach is to use SQL to store and retrieve data part by part and convert to pandas dataframe and process it.

2. **Model Development and Training:** Training the model based on the dataset and predicting the features based on demand
3. **Setting up cluster environment:** Setting up minikube (local kubernetes cluster orchestration environment) and setting up some simple backend servers inside containers to respond to the requests. Two simple http servers need to be set up using node.js and express.js to replicate the client and server model.
4. **Integrating the Trained Model inside the cluster to scale the containers dynamically**
5. **Deployment on GitHub**

Software/Hardware/ Dataset Requirements:

Software Stack: Keras, Scikit Learn, Kubernetes Orchestration Engine (Docker, Minikube), python, jupyterlab, MySQL(for storing the large dataset), Dask

Languages: Javascript, Python, SQL

Dataset: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/3QBYB5

Network Simulation: I want to focus on the implementation of the Forecasting method using LSTM Neural Network with implementation on containers and hence want to keep the simulation part as simple as possible. For this instead of using a network simulator like ns3, I will run two basic http servers and pass http headers from one to another replicating the client and server model. Based on these headers, the model will read and forecast the demand of containers (nodes and compute engines)

Timeline:

Objectives	**Status**	**Date of Status Change**	**Notes**
Preprocessing	Not started		Deadline: 25 Nov 2023
Model Development and Training	Not started		Deadline: 30 Nov 2023
Setting up cluster environment	Not started		Deadline: 7 Dec 2023
Integration	Not started		Deadline: 14 Dec 2023
Deployment	Not started		Deadline: 15 Dec 2023