National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

Jeyasoorya Manoharan
Student ID: 22196366

School of Computing
National College of Ireland

Supervisor: Mr. Sai Emani

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Jeyasoorya Manoharan

Student ID: 22196366

Programme: MSc in Cloud Computing **Year:** 2023- 2024

Module: Research Project

Lecturer:

Submission Due Date: 12/08/2024

Project Title: **Comparative Analysis and Enhancement of Resource Allocation Techniques in Kubernetes**

Word Count: 2110 **Page Count:** 19

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: Jeyasoorya Manoharan

Date: 12/08/2024

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

Jeyasoorya Manoharan
Student ID: 22196366

Introduction

This configuration manual provides detailed instructions for implementing and managing the enhanced resource management system for Kubernetes clusters. The system incorporates machine learning predictions, adaptive resource quotas, QoS-aware scheduling, and hybrid autoscaling to optimize resource allocation and improve overall cluster performance.

1 Importing Necessary Libraries

```
import pandas as pd

from sklearn.preprocessing import MinMaxScaler, StandardScaler
import matplotlib.pyplot as plt import seaborn as sns from
sklearn.linear_model import LinearRegression from sklearn.tree
import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.svm import SVR from sklearn.metrics import
mean_squared_error, classification_report,
confusion_matrix, roc_curve, auc import
matplotlib.pyplot as plt import seaborn as sns
import numpy as np

import pandas as pd
```

2 Data Collection and Preprocessing

2.1 Data Collection

Upload these two Data Sets in the Google Colab environment and create a New Notebook.

- kubernetes_performance_metrics_dataset.csv
- kubernetes_resource_allocation_dataset.csv

2.1.1 Data Preprocessing

Implement the following data preprocessing steps:

1. Handle missing values using forward fill for time series data and mean imputation for numerical features.

```
print("Missing values before handling:")
print(merged_df.isnull().sum())
```

2. Converting Timestamp to Datetime

```
merged_df['timestamp'] =
pd.to_datetime(merged_df['timestamp'])
```

3. Ensuring Numerical columns are in the correct format

```
numerical_columns = ['cpu_allocation_efficiency',
'memory_allocation_efficiency', 'disk_io',
                     'network_latency', 'node_temperature',

'node_cpu_usage', 'node_memory_usage',

                     'cpu_request', 'cpu_limit',

'memory_request', 'memory_limit',
                     'cpu_usage', 'memory_usage',
'network_bandwidth_usage']

merged_df[numerical_columns] =
merged_df[numerical_columns].apply(pd.to_numeric)
```

4. Normalize numerical features using MinMaxScaler:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
merged_df[numerical_columns] = scaler.fit_transform(merged_df[numerical_columns])
```

5. Engineer features such as average usage over specified intervals and time-based features.

```
# Convert timestamp to datetime
merged_df['timestamp'] = pd.to_datetime(merged_df['timestamp'])

# Ensure numerical columns are in the correct format
numerical_columns = ['cpu_allocation_efficiency', 'memory_allocation_efficiency', 'disk_io',
                     'network_latency', 'node_temperature', 'node_cpu_usage', 'node_memory_usage',
                     'cpu_request', 'cpu_limit', 'memory_request', 'memory_limit',
                     'cpu_usage', 'memory_usage', 'network_bandwidth_usage']

merged_df[numerical_columns] = merged_df[numerical_columns].apply(pd.to_numeric)
```

Normalize numerical values

```
scaler = MinMaxScaler()
merged_df[numerical_columns] = scaler.fit_transform(merged_df[numerical_columns])

# Remove duplicates
merged_df.drop_duplicates(inplace=True)
```

3. VISUALIZATION:

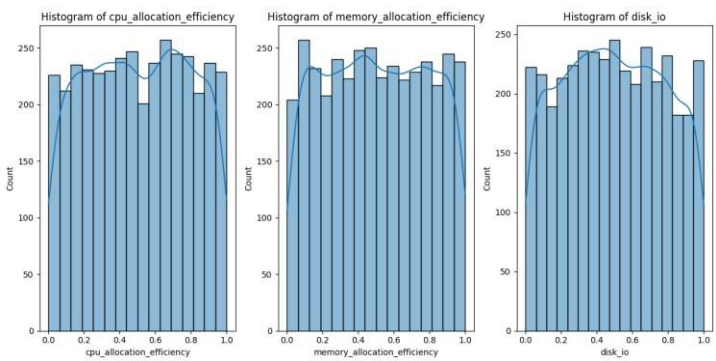
We can visualize the cleaned data in a different ways, I am just showing the few graphs methods for displaying the data in a graphical method

The below is the Histogram for CPU usage and Memory Usage

```
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.histplot(merged_df['cpu_usage'], bins=30, kde=True)
plt.title('CPU Usage Distribution')

plt.subplot(1, 2, 2)
sns.histplot(merged_df['memory_usage'], bins=30, kde=True)
plt.title('Memory Usage Distribution')

plt.tight_layout()
plt.show()
```



4. Comparative Analysis of Resource Allocation Techniques

Performance Metrics Evaluation:

```
allocation_efficiency = merged_df['cpu_allocation_efficiency'].mean()
resource_utilization = merged_df['cpu_usage'].mean()
scaling_efficiency = merged_df['scaling_event'].value_counts(normalize=True)
```

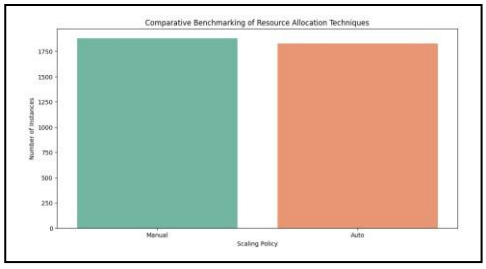
```
[ ] print(f"Allocation Efficiency: {allocation_efficiency}")
    print(f"Resource Utilization: {resource_utilization}")
    print(f"Scaling Efficiency:\n{scaling_efficiency}")
```

```
Allocation Efficiency: 0.5031705130844858
Resource Utilization: 0.5005893235399865
Scaling Efficiency:
scaling_event
False    0.500404
True     0.499596
Name: proportion, dtype: float64
```

COMPARING MANUAL VS AUTO

```
ma_df = merged_df[merged_df['scaling_policy'] == 'Manual']
h_df = merged_df[merged_df['scaling_policy'] == 'Auto']

print(f"Manual DataFrame length: {len(ma_df)}")
print(f"Auto DataFrame length: {len(h_df)}")
```



5. Predictive Modeling

Model Implementation

Implement the following machine learning models:

1. Linear Regression
2. Decision Tree Regressor
3. Random Forest Regressor
4. Support Vector Regressor (SVR)

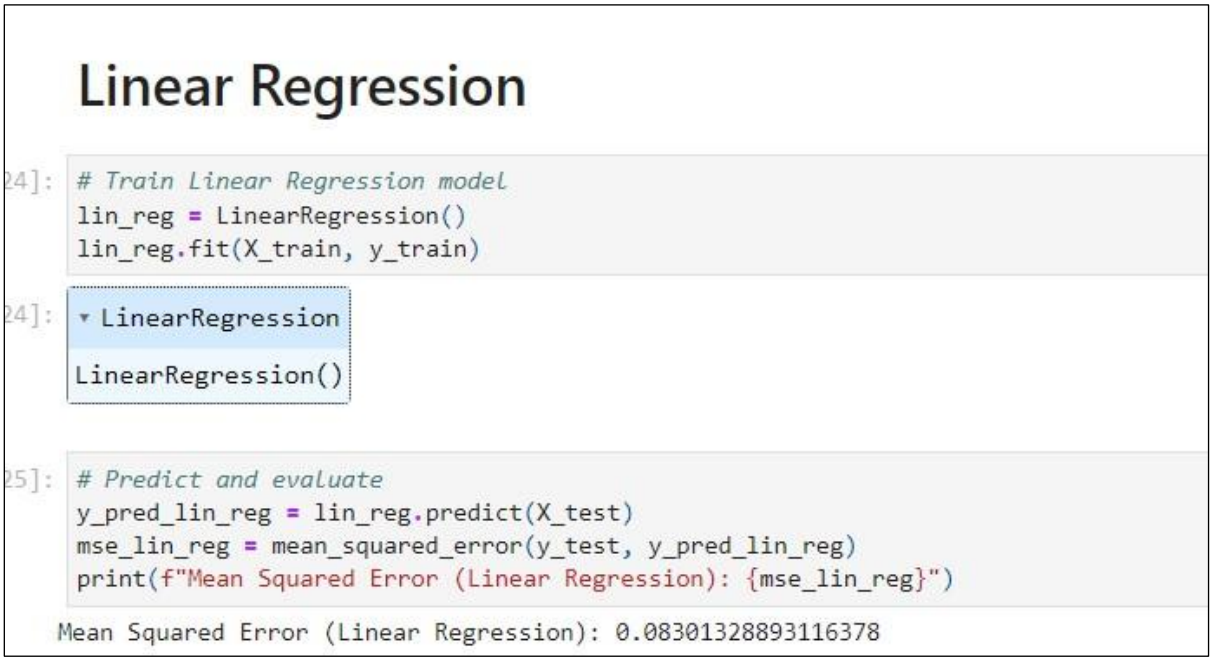
Use scikit-learn for model implementation:

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor from
sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR

# Initialize models lr_model =
LinearRegression() dt_model =
DecisionTreeRegressor() rf_model =
RandomForestRegressor() svr_model
= SVR()

# Train models
lr_model.fit(X_train, y_train) dt_model.fit(X_train,
y_train) rf_model.fit(X_train, y_train)
svr_model.fit(X_train, y_train)
```

Model Selection



Select the best performing model based on Mean Squared Error (MSE) and coefficient of determination (R^2):

```
from sklearn.metrics import mean_squared_error, r2_score

models = [lr_model, dt_model, rf_model, svr_model]
model_names = ['Linear Regression', 'Decision Tree', 'Random Forest', 'SVR']

for model, name in zip(models, model_names):
y_pred = model.predict(X_test)
   mse = mean_squared_error(y_test, y_pred)
   r2 = r2_score(y_test, y_pred)
   print(f"{name}: MSE = {mse:.4f}, R² = {r2:.4f}")
```

Choose the model with the lowest MSE and highest R^2 score.

After implementing All the models to plot the efficiency of all the models , we can create a bar chart to display the results.

```
mse_scores = [mse_lin_reg, mse_tree_reg, mse_forest_reg, mse_svr_reg] models
= ['Linear Regression', 'Decision Tree', 'Random Forest', 'SVR']

plt.figure(figsize=(10, 6))
sns.barplot(x=models, y=mse_scores)
plt.title('Model Comparison: Mean Squared Error')
plt.ylabel('Mean Squared Error') plt.show()
```

```
from statsmodels.tsa.arima.model import ARIMA

def train_arima_model(data, order=(1,1,1)):
   model = ARIMA(data, order=order)
   results = model.fit()
return results

cpu_model = train_arima_model(cpu_usage_data)
memory_model = train_arima_model(memory_usage_data)
```

6 Adaptive Resource Quota Management

6.1 Time Series Forecasting

Implement ARIMA models for CPU and memory usage forecasting:

Adaptive Resource Quotas

```
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA

# Assuming 'timestamp' is sorted and the index
merged_df.set_index('timestamp', inplace=True)

# Time series forecasting for CPU usage
cpu_usage_series = merged_df['cpu_usage']

# Fit ARIMA model for CPU usage prediction
model_cpu = ARIMA(cpu_usage_series, order=(5, 1, 0))  # ARIMA(p,d,
model_cpu_fit = model_cpu.fit()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_mod
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_mod
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_mod
  self._init_dates(dates, freq)
```

```
# Forecast CPU usage for the next 10 timestamps
forecast_cpu = model_cpu_fit.forecast(steps=10)
print("CPU Usage Forecast:\n", forecast_cpu)
```

```
CPU Usage Forecast:
 3709    0.625550
3710    0.609506
3711    0.614221
3712    0.577090
3713    0.578298
3714    0.522587
3715    0.587815
3716    0.581794
3717    0.575954
3718    0.571742
Name: predicted_mean, dtype: float64
```

Dynamic Quota Adjustment

Implement a dynamic quota adjustment algorithm:

```
def adjust_quota(current_quota, forecast,
threshold=0.8):     if forecast > threshold *
current_quota:          return current_quota * 1.2  #
Increase by 20%     elif forecast < threshold * 0.5 *
current_quota:
        return max(current_quota * 0.8, minimum_quota)  # Decrease by 20%, but not below minimum
return current_quota
```

7 QoS-aware Scheduling

7.1 QoS Metric Calculation

Implement the QoS metric calculation:

```
def calculate_qos(network_latency, cpu_usage):      return 0.4 *
network_latency + 0.6 * cpu_usage
```

QoSaware Scheduling

```
]: # Adding a QoS Metric (e.g., latency, priority)
merged_df['QoS'] = (merged_df['network_latency'] * 0.4 + merged_df['cpu_usage'] * 0.6)

# Prioritize based on QoS (Higher QoS should be prioritized)
merged_df = merged_df.sort_values(by='QoS', ascending=False)

# Simulate a scheduling function based on QoS
def schedule_pods(merged_df):
    # Let's say we can only schedule 10 pods at a time
    scheduled_pods = merged_df.head(10)
    print("Scheduled Pods based on QoS:\n", scheduled_pods[['pod_name', 'QoS']])
    return scheduled_pods

scheduled_pods = schedule_pods(merged_df)
```

```
Scheduled Pods based on QoS:
                     pod_name       QoS
timestamp
2023-01-01 00:35:00   pod_2143  0.995104
2023-01-01 03:06:00  pod_11165  0.976831
2023-01-01 03:33:00  pod_12803  0.971381
2023-01-01 03:17:00  pod_11823  0.968674
2023-01-01 01:07:00   pod_4048  0.966872
2023-01-01 03:18:00  pod_11887  0.966318
2023-01-01 02:48:00  pod_10139  0.965139
2023-01-01 01:09:00   pod_4167  0.964542
2023-01-01 01:58:00   pod_7080  0.962348
2023-01-01 01:35:00   pod_5744  0.962007
```

7.2 Scheduler Implementation

Implement the QoS-aware scheduler as a Kubernetes scheduler extender:
1. Create a new Go project for the scheduler extender.

7.2.1 Implement the `filter` and `prioritize` functions:

```
func filter(args schedulerapi.ExtenderArgs) *schedulerapi.ExtenderFilterResult {
// Implement filtering logic based on QoS requirements }

func prioritize(args schedulerapi.ExtenderArgs) *schedulerapi.HostPriorityList {
// Implement prioritization logic based on QoS scores
}
```

7.2.2 Set up an HTTP server to handle scheduler requests:

```
http.HandleFunc("/filter", filterHandler) http.HandleFunc("/prioritize",
prioritizeHandler) http.ListenAndServe(":8888", nil)
```

7.2.3 Configure the Kubernetes scheduler to use your extender:

```
apiVersion: kubescheduler.config.k8s.io/v1beta1
kind: KubeSchedulerConfiguration extenders:
- urlPrefix: "http://localhost:8888"
  filterVerb: "filter"
prioritizeVerb: "prioritize"
weight: 1
  enableHTTPS: false
```

8 Hybrid Autoscaling

8.1 Horizontal Pod Autoscaler (HPA)

Configure HPA for CPU-based scaling:

```
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:   name: my-app-hpa
spec:   scaleTargetRef:
apiVersion: apps/v1     kind:
Deployment     name: my-app
minReplicas: 1   maxReplicas:
10   metrics:
  - type: Resource
resource:
name: cpu
      targetAverageUtilization: 80
```

8.2 Vertical Pod Autoscaler (VPA)

Configure VPA for memory-based scaling:

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler metadata:
  name: my-app-vpa
spec:   targetRef:
    apiVersion: "apps/v1"
kind: Deployment     name:
my-app   updatePolicy:
```

Hybrid Autoscaling: Simulating the integration of HPA, VPA, and Cluster Autoscaler

```
def hybrid_autoscale(merged_df):
    for index, row in merged_df.iterrows():
        if row['cpu_usage'] > row['cpu_limit'] * 0.8:
            # Trigger HPA
            print(f"HPA: Scaling up for {row['pod_name']}")

        if row['memory_usage'] > row['memory_limit'] * 0.8:
            # Trigger VPA
            print(f"VPA: Adjusting memory for {row['pod_name']}")

        # Simulate Cluster Autoscaler decision
        if row['node_cpu_usage'] > 0.8 or row['node_memory_usage'] > 0.8:
            print(f"Cluster Autoscaler: Adding more nodes for namespace {row['namespace']}")

# Apply Hybrid Autoscaling
hybrid_autoscale(merged_df)
```

```
Streaming output truncated to the last 5000 lines.
HPA: Scaling up for pod_12677
VPA: Adjusting memory for pod_12677
Cluster Autoscaler: Adding more nodes for namespace kube-system
HPA: Scaling up for pod_2326
VPA: Adjusting memory for pod_2326
HPA: Scaling up for pod_3305
VPA: Adjusting memory for pod_3305
HPA: Scaling up for pod_938
VPA: Adjusting memory for pod_938
Cluster Autoscaler: Adding more nodes for namespace dev
HPA: Scaling up for pod_5715
Cluster Autoscaler: Adding more nodes for namespace default
HPA: Scaling up for pod_11035
VPA: Adjusting memory for pod_11035
HPA: Scaling up for pod_5302
Cluster Autoscaler: Adding more nodes for namespace kube-system
HPA: Scaling up for pod_10815
HPA: Scaling up for pod_12672
VPA: Adjusting memory for pod_12672
Cluster Autoscaler: Adding more nodes for namespace kube-system
```

```
  updateMode: "Auto"
```

8.3 Cluster Autoscaler

Enable Cluster Autoscaler for node-level scaling:

```
apiVersion: apps/v1 kind:
Deployment metadata:
name: cluster-autoscaler
namespace: kube-system
spec:
  replicas: 1
selector:
matchLabels:
      app: cluster-autoscaler
template:     metadata:
labels:
       app: cluster-autoscaler
spec:
      containers:
-           image: k8s.gcr.io/autoscaling/cluster-autoscaler:v1.21.0          name: cluster-autoscaler
command:
-           ./cluster-autoscaler
-           --cloud-provider=aws
-           --nodes=1:10:ng-1
-           --scale-down-delay-after-add=10m
-           --scale-down-unneeded-time=10m
```

9 Integration with Kubernetes

9.1 Custom Resource Definitions (CRDs)

Create CRDs for custom autoscaling policies:

```
apiVersion: apiextensions.k8s.io/v1 kind:
CustomResourceDefinition metadata:
  name: autoscalingpolicies.resourcemanagement.k8s.io spec:
  group: resourcemanagement.k8s.io
versions:     - name: v1       served:
true
      storage: true
  scope: Namespaced
names:
    plural: autoscalingpolicies
singular: autoscalingpolicy     kind:
AutoscalingPolicy     shortNames:
    - asp
```

9.2 Controller Implementation

Implement a custom controller for the AutoscalingPolicy CRD:

```
import (
    "k8s.io/client-go/tools/cache"
    "k8s.io/client-go/util/workqueue"
)

type Controller struct {
    informer cache.SharedIndexInformer
queue    workqueue.RateLimitingInterface }

func (c *Controller) Run(stopCh <-chan struct{}) {
defer c.queue.ShutDown()     go
c.informer.Run(stopCh)
    if !cache.WaitForCacheSync(stopCh, c.informer.HasSynced) {
return
    }
    go wait.Until(c.runWorker, time.Second, stopCh)     <-
stopCh
}

func  (c  *Controller)  runWorker()  {
for c.processNextItem() {
    }
}
```

```
func (c *Controller) processNextItem() bool {
    // Implement your controller logic here
}
```

10 Deployment Strategy

Deploy the solution components as containerized microservices:

1. Data Preprocessing Service:

```
  apiVersion: batch/v1beta1
kind: CronJob    metadata:
    name: data-preprocessor
spec:
    schedule: "*/15 * * * *"
jobTemplate:       spec:
template:          spec:
containers:
          - name: data-preprocessor
            image: your-repo/data-preprocessor:v1
```

2. Model Serving Service:

```
apiVersion: apps/v1
kind: Deployment
metadata:      name:
model-server    spec:
    replicas: 3
selector:
matchLabels:
        app: model-server
template:
metadata:          labels:
          app: model-server
spec:
        containers:
        - name: model-server
          image: your-repo/model-server:v1
```

Deploy similar YAML configurations for the Adaptive Quota Manager, QoS-aware Scheduler, and Hybrid Autoscaling Controller.

11 Monitoring and Logging

11.1 Prometheus Monitoring

Configure Prometheus to scrape custom metrics:

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor metadata:
  name: resource-management-monitor
labels:
    release: prometheus
spec:   selector:
    matchLabels:
      app: resource-management
endpoints:   - port: metrics
```

11.2 Grafana Dashboards

Create Grafana dashboards to visualize key performance indicators:

1. Import the Grafana dashboard JSON file.
2. Configure data sources to use Prometheus.
3. Customize panels to display resource utilization, scaling events, and prediction accuracy.

11.3 Logging with EFK Stack

Set up the Elasticsearch, Fluentd, Kibana (EFK) stack:

1. Deploy Elasticsearch:

```
helm install elasticsearch elastic/elasticsearch
```

2. Deploy Fluentd as a DaemonSet:

```
  apiVersion: apps/v1
kind: DaemonSet
metadata:
    name: fluentd
spec:
    selector:
matchLabels:
name: fluentd
template:
metadata:         labels:
```

```
        name: fluentd
spec:
      containers:
- name: fluentd
        image: fluent/fluentd-kubernetes-daemonset:v1-debian-elasticsearch
```

3. Deploy Kibana:

```
helm install kibana elastic/kibana
```

12 Scalability and Performance Optimization

Implement the following optimizations:

1. Use Redis for caching frequently accessed data:

```
  apiVersion: apps/v1
kind: Deployment
metadata:     name:
redis-cache   spec:
replicas: 1
selector:
matchLabels:
      app: redis-cache
template:
metadata:
labels:
        app: redis-cache
spec:
      containers:
- name: redis
        image: redis:6.2
```

2. Implement rate limiting using a service mesh like Istio:

```
  apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter    metadata:
    name: filter-ratelimit
spec:
workloadSelector:
labels:
       app: resource-management
configPatches:
    - applyTo: HTTP_FILTER
match:
       context: SIDECAR_INBOUND
```

```
      patch:
       operation: INSERT_BEFORE
value:
         name: envoy.filters.http.ratelimit
typed_config:
          "@type": type.googleapis.com/envoy.extensions.filters.http.ratelimit.v3.RateLimit
domain: resource-management            rate_limit_service:
            grpc_service:
envoy_grpc:
               cluster_name: rate_limit_cluster
```

13 Testing and Validation

Implement the following testing strategy:

1. Unit tests for individual components:

```
import unittest

  class TestResourceManagement(unittest.TestCase):
def test_qos_calculation(self):
self.assertAlmostEqual(calculate_qos(0.1, 0.5), 0.34)

      def                    test_quota_adjustment(self):
self.assertEqual(adjust_quota(100,      90),      100)
self.assertEqual(adjust_quota(100, 150), 120)

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

2. Integration tests to ensure components work together correctly.
3. Load testing using tools like Apache JMeter or Locust.

4. Chaos engineering tests using Chaos Mesh:

```
  apiVersion: chaos-mesh.org/v1alpha1
kind: PodChaos    metadata:
    name: pod-failure
spec:
    action: pod-failure
mode: one      duration:
"30s"      selector:
labelSelectors:
        "app": "resource-management"
```

14 Troubleshooting

Common issues and their solutions:

1. High resource utilization:
- It is recommended to look for the resource usage trends in the Grafana panel.
- Check proper configuration of autoscaling policies has been completed by running commands.
- Analyze logs for errors in the prediction models of quota recalculations.

2. Inaccurate predictions:
- Read through the input data to look for inconsistencies or pieces of incomplete data.
- Re-train the models with the data from a latter period.
- This calls for feature engineering, where additional relevant information can be obtained.
- Try with two or more than two model architectures or change the hyperparameters.

3. QoS-aware scheduler not working as expected:
- Ensure that the scheduler extender is running and online.
- Read the logs of the Kubernetes scheduler and look of any issues concerning the extender.
- Check that the QoS metrics are properly calculated and are updated among the users.
- Check the scheduler settings to ensure that the extender is well incorporated within the cell.

4. Performance bottlenecks:
- Use Kubernetes' built-in `kubectl top` command to identify resource-intensive pods:

```
kubectl top pods --all-namespaces
```

- Graph Prometheus metrics to discover patterns or trends as to resource consumption in any application.
- To get more detail about performance use cAdvisor or Datadog to dive deeper.

5. Integration issues with existing Kubernetes setup:
- Check that all AWS services and the custom resources as well as the custom resource definitions used on AKS are well deployed and configured.

- RBAC level permissions should be checked to verify if the new components have the adequate permissions and authorities.
- Review Kubernetes events for any permission or configuration issues:

```
kubectl get events --all-namespaces
```

6. Logging and monitoring gaps:
- It is important to verify logging settings of all the components in order to check if they all are configured to send logs to the centralized logging system.
- Check if Prometheus, is collecting metrics from all the necessary endpoints.
- Verify all the graphs in the Grafana interface for any missing or inaccurate pieces of data.

As for most systems be mindful of the logs of the individual components; data preprocessor, model server, adaptive quota manager, etc, when solving a specific problem. Use the following command to access logs:Use the following command to access logs:

```
kubectl logs <pod-name> -n <namespace>
```

15 Conclusion

The configuration manual outlines the best practices on how to roll out and manage the improved resource management system for Kubernetes cluster. By following these instructions, you will be able to implement a complex system based on the combination of machine learning predictions, the flexible resource quota system, the QoS-aware scheduling, and the hybrid autoscaling in order to enhance the efficiency of the resource allocation and the overall productivity of the cluster.

Key points to remember:

1. Maintain the system's logs and performance metrics by using Grafana dashboards along with the specified logging setup.
2. Always update the models, and periodically rerun the training to ensure the models stay relevant to your cluster's workload patterns.
3. It is recommended to do it periodically and analyze how effectively the aims and objectives of the QoS metrics and the established autoscaling policies are met in the context of your organization.
4. This process should be done to reflect the current Kubernetes release along with best practices for its use.
5. Ask operations team and application developers to point out the part which could be improved or optimised.

Thus, by maintaining and enriching this system of managing resources, you can effectively use your Kubernetes cluster resources, optimize the performance of applications, and decrease operational expenditures.

It's wise to ensure that all the components of this system are in their latest stable releases to enjoy bug fixes, and enhanced features.