Appendix

ACIT5900

Master Thesis

in

Applied Computer and Information Technology (ACIT)

May 2023

Cloud-Based Services and Operations

Towards Green Container Management:

A novel approach for container resource allocation through statistical modeling

Håkon Borgersen Ay
Department of Computer Science
Faculty of Technology, Art and Design

Towards Green Container Management: A novel approach for container resource allocation through statistical modeling
Supervisor: Kyrre Begnum

https://www.oslomet.no

Printed: Oslo Metropolitan University

Contents

Figures

Code Listings

Appendix A

Survey Sent to Developers at Intility

Appendix B consists of all questions and answers from the survey sent out to developers at Intility. Below is the description of the survey which was sent out together with the survey

Hello, my name is Håkon Borgersen Ay. I work part-time at the developer infrastructure team and will be starting full-time as a trainee in August 2023. Right now I'm writing my master thesis with the working title: "Optimizing Resource Utilization in a Kubernetes Cluster: Investigating and Automating Resource Allocation using Workload Profiles".

I am conducting an anonymous survey to gather information about resource utilization and cloud waste associated with cloud resources. Your input will help me identify areas of improvement and potential cost savings.

The term Cloud is in this case used to describe Kubernetes clusters. Consider the resource usage vs. the allocated resources of clusters, nodes, and pods when answering. When the word resources are used I'm referring to the allocated CPU and RAM.

Please note that while some of the data collected through this survey may be used in my master thesis, certain information may be kept confidential due to policy constraints. I appreciate your participation.

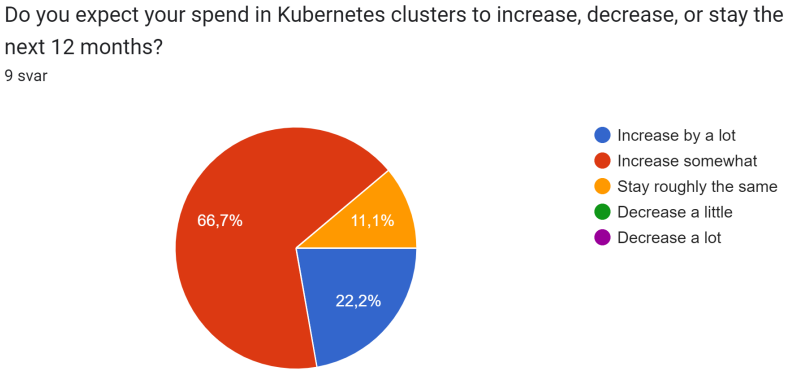

Figure A.1: Survey Sent to Developers at Intility: Question 1

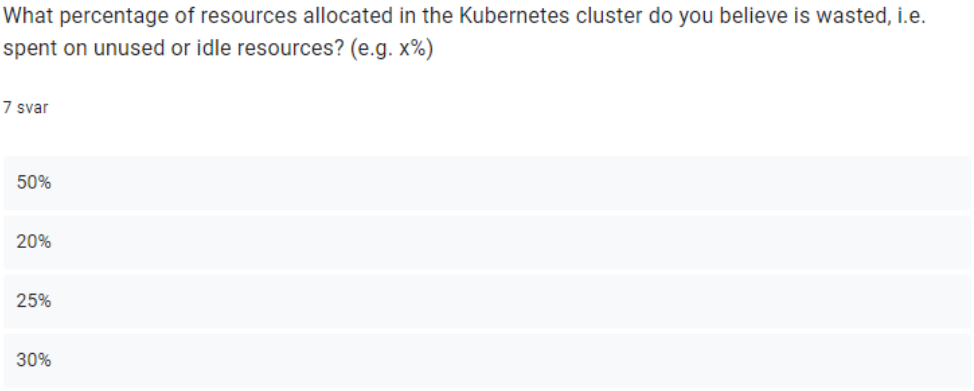

Figure A.2: Survey Sent to Developers at Intility: Question 2

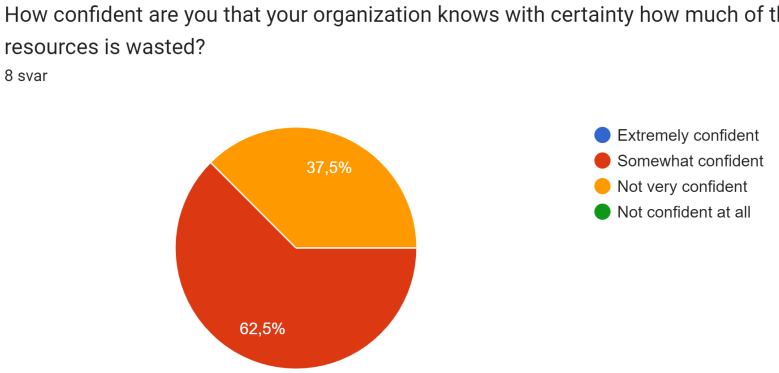

Figure A.3: Survey Sent to Developers at Intility: Question 3

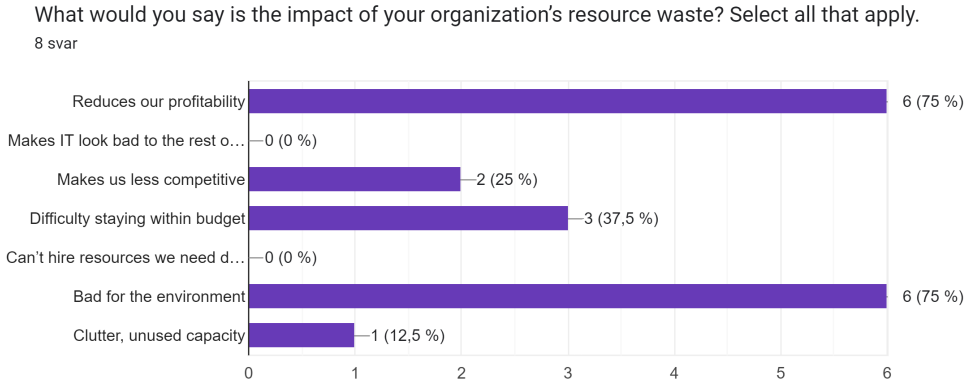

Figure A.4: Survey Sent to Developers at Intility: Question 4

Answers from Figure A.4

Reduces our profitability

Makes IT look bad to the rest of the organization

Makes us less competitive

Difficulty staying within budget

Can't hire resources we need due to budget constraints

Bad for the environment

Clutter, unused capacity

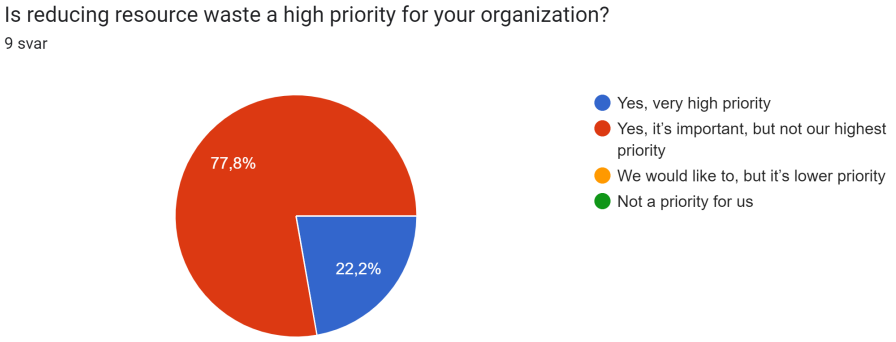

Figure A.5: Survey Sent to Developers at Intility: Question 5

Figure A.6: Survey Sent to Developers at Intility: Question 6

Answers from figure A.6:

Cloud complexity makes it hard to estimate how many resources are actually needed Over-provisioning to ensure applications perform well
Maintaining more environments than needed
Developers have no incentive to run apps efficiently
Not enough attention or management oversight of cloud spend
No visibility into actual resource utilization vs. what we're paying for
Controlling cloud spend is not a priority for us
Ensuring we are ready for a potential surge in demand
Applications are configured for high availability, causing us to run more resources than actually needed

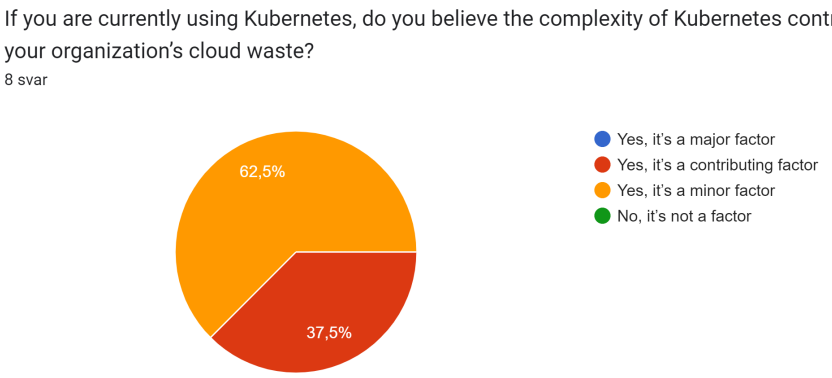

Figure A.7: Survey Sent to Developers at Intility: Question 7

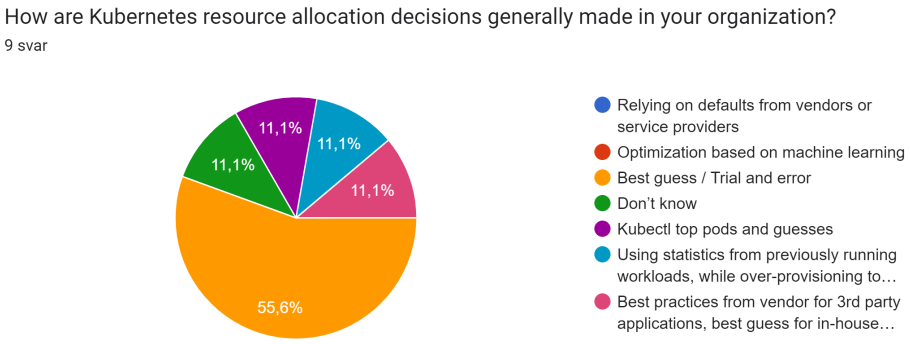

Figure A.8: Survey Sent to Developers at Intility: Question 8

Answers from Figure A.8

Relying on defaults from vendors or service providers

Optimization based on machine learning

Best guess / Trial and error

Don't know

Kubectl top pods and guesses

Using statistics from previously running workloads, while over-provisioning to ensure application performance

Best practices from vendor for 3rd party applications, best guess for in-house developed applications

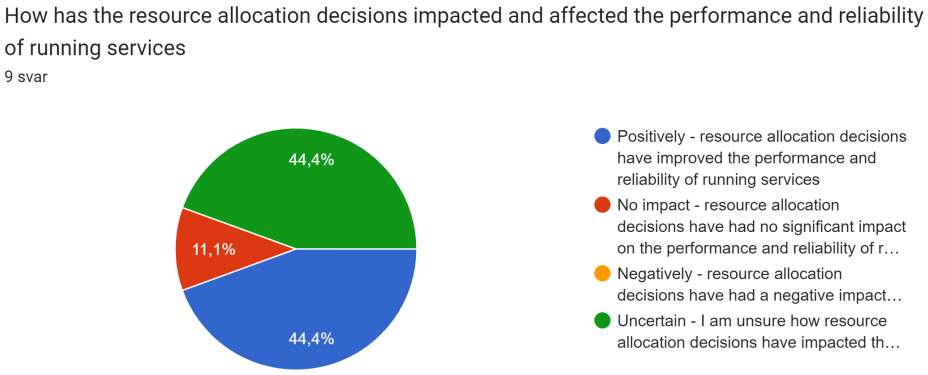

Figure A.9: Survey Sent to Developers at Intility: Question 9

Figure A.10: Survey Sent to Developers at Intility: Question 10

Appendix B

Python Code

B.1 Pre-processing Code

B.1.1 Reading and pre-processing .csv files

Code listing B.1: Reading and appending .csv files

```
# Reading and appending .csv files (ram)

container=1
df1 = pd.read_csv(f'B{container}.csv',
                  sep=',',
                 low_memory=False)

orig_name= df1.columns[1]
df1 = df1.rename(columns={orig_name: 'Containers: Memory usage'})

# Find the index of the last non-null value
last_non_null_index = df1['Containers: Memory usage'].last_valid_index()

# Remove rows with missing values after the last non-null value
df1 = df1.loc[:last_non_null_index]

# Counting missing values and find the percentage
#df1.info()
#print(df1['Containers: Memory usage'].value_counts())
#print(df1[' bytes - collector-xcfsq collector | collector'].value_counts())

# Standardized date format
df1['Date'] = pd.to_datetime(df1['Date'])

```

```
# converting n/a's to mean value
df1['Containers: Memory usage'] = df1['Containers: Memory usage'].apply(
    convert_memory_to_float)
df1['Containers: Memory usage'] = df1['Containers: Memory usage'].fillna(df1['
    Containers: Memory usage'].mean())

fig, ax = plt.subplots(figsize=(12, 6))
ax.set_xlabel('Date')
ax.set_ylabel('Memory Usage (MB)')
ax.set_title('Memory Usage Over Time container01')
ax.grid()

# Customize date ticks and format
ax.xaxis.set_major_locator(mdates.AutoDateLocator())
ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.setp(ax.get_xticklabels(), rotation=30, ha='right')

#plt.savefig(f'memory_usage_c{container}.png', dpi=300, bbox_inches='tight')
plt.show()
```

B.1.2 Converting memory to float

Code listing B.2: Converting memory to float

```
def convert_memory_to_float(value):
    # Check if the value is already numeric
    if isinstance(value, (int, float)):
        return float(value)

    # If value is a string, check the unit
    if isinstance(value, str):
        unit = None
        number = value.strip()
        if 'B' in value.upper() and 'MB' not in value.upper() and 'KB' not in value
            .upper():
            unit = 'B'
            number = value[:-1].strip()
        elif 'KB' in value.upper():
            unit = 'KB'
            number = value[:-2].strip()
        elif 'MB' in value.upper():
            unit = 'MB'
            number = value[:-2].strip()
        elif 'GB' in value.upper():
```

```
            unit = 'GB'
            number = value[:-2].strip()

        # Convert the numeric part to a float
        value = float(number)

        # Convert the value to MB based on the unit
        if unit == 'B':
            value = value / (1024 * 1024)
        elif unit == 'KB':
            value = value / 1024
        elif unit == 'GB':
            value = value * 1024

    return value
```

B.1.3 Calculating skewness

Code listing B.3: Calculating skewness in container

```
column_name = "Container: Memory usage"

skewness = df1['Containers: Memory usage'].skew()

print(df1['Containers: Memory usage'].median())
# Create a histogram with 10 bins and a density plot
fig, ax = plt.subplots()
ax.hist(df1['Containers: Memory usage'], bins=20, density=True, alpha=0.5, color="
    blue")
ax.set_xlabel(column_name)
ax.set_ylabel("Density")

ax.set_xlim(100, max(df1['Containers: Memory usage']))

# Add a vertical line at the median, mean, mode, and skewness value
ax.axvline(df1['Containers: Memory usage'].median(), color="red", linestyle="--",
    label="Median")
ax.axvline(df1['Containers: Memory usage'].mean(), color="orange", linestyle="--",
    label="Mean")
ax.axvline(df1['Containers: Memory usage'].mode()[0], color="green", linestyle="--"
    , label="Mode")
ax.axvline(skewness, color="purple", linestyle="--", label="Skewness: {:.2f}".
    format(skewness))

```

```
# Add a legend and title
ax.legend()
ax.set_title("Distribution of memory usage")

# Display the plot
#plt.savefig(f'memory_usage_distribution_c{container}.png', dpi=300, bbox_inches='
    tight')
plt.show()
```

B.1.4 Script for showing daily memory usage during a 2 week period

Code listing B.4: Script showing daily memory usage for a 2 week period

```

  def plot_24_hour_segments(df, column):
    days = df.index.normalize().unique()

    plt.figure(figsize=(12, 6))

    for day in days:
        day_data = df.loc[day:day + pd.Timedelta('1D') - pd.Timedelta('5m'), column
            ]
        plt.plot(day_data.index.time, day_data.values, label=day.date())


df_days=df1
df_days = df_days[['Date','Containers: Memory usage']]
df_days['Date'] = pd.to_datetime(df_days['Date'])
df_days.set_index('Date', inplace=True)



def plot_24_hour_segments(df, column, window_size=12):
    first_timestamp = df.index[0]
    days = df.index.normalize().unique()

    # Calculate end date
    last_full_day = df.index[-1].normalize() - pd.Timedelta('1D')
    end_date = last_full_day + pd.Timedelta(hours=first_timestamp.hour, minutes=
        first_timestamp.minute)

    # Apply rolling average
    df_smooth = df.rolling(window=window_size, center=True).mean()

```

```
    plt.figure(figsize=(12, 6))

    for day in days:
        start_time = day + pd.Timedelta(hours=first_timestamp.hour, minutes=
            first_timestamp.minute)
        end_time = start_time + pd.Timedelta('1D') - pd.Timedelta('5m')

        if end_time <= end_date:
            day_data = df_smooth.loc[start_time:end_time, column]

            if not day_data.empty:
                times = [t.hour * 60 + t.minute for t in day_data.index.time]
                plt.plot(times, day_data.values, label=day.date())

    plt.xlabel('Time of Day')
    plt.ylabel('Memory Usage in MiB')
    plt.legend()
    plt.title('Daily Memory Usage Patterns (Smoothed)')

    one_day_minutes = 24 * 60
    plt.xlim(0, one_day_minutes)

    # Set x-axis ticks and labels
    x_ticks = np.arange(0, one_day_minutes + 1, 2 * 60)  # Every 4th hour in
        minutes
    x_labels = [f'{t // 60:02d}:00' for t in x_ticks]
    plt.xticks(x_ticks, x_labels, rotation=45)

    plt.savefig('daily_memory_usage_patterns.png', dpi=300, bbox_inches='tight')
    plt.show()


plot_24_hour_segments(df_days, 'Containers: Memory usage')
```

B.2 Linear programming

Code listing B.5: The Linear Programming Model

```

# Linear programming modeling
import pulp

safety_margin_percentage = 10  # Additional memory allocated as a percentage of the
    max usage
```

```
safety_margin = max(df1['Containers: Memory usage']) * (safety_margin_percentage / 100)

# Calculate the typical memory usage (e.g., the mean or median)
typical_memory_usage = np.mean(df1['Containers: Memory usage']) * 1.05

# Defining the problem, and specifying that the objective is to minimize a function
prob = pulp.LpProblem("ContainerResourceAllocation", pulp.LpMinimize)

# Variables
x_request = pulp.LpVariable("x_request", lowBound=typical_memory_usage)
x_limit = pulp.LpVariable("x_limit", lowBound=max(df1['Containers: Memory usage']) + safety_margin)

# Objective function
prob += x_request

# Constraints
prob += x_request <= x_limit

# Solve the problem
status = prob.solve()

# Print the results
#print("Status:", pulp.LpStatus[status])
req_value = pulp.value(x_request)
lim_value = pulp.value(x_limit)
print("Request limit for Container 1:", pulp.value(x_request))
print("Memory limit for Container 1:", pulp.value(x_limit))

fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(df1['Date'], df1['Containers: Memory usage'])
ax.set_xlabel('Date')
ax.set_ylabel('Memory Usage (MB)')
ax.set_title(f'Memory Usage Over Time container0{container}')
ax.grid()

# Customizing date ticks and format
ax.xaxis.set_major_locator(mdates.AutoDateLocator())
ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.setp(ax.get_xticklabels(), rotation=30, ha='right')


ax.axhline(y=req_value, color='r', linestyle='--', label='Request')
ax.axhline(y=lim_value, color='b', linestyle='--', label='Limit')
```

```
49
50 ax.legend()
51
52 # print plot
53 plt.savefig(f'memory_usage_linear_programming_c{container}', dpi=300, bbox_inches='
       tight')
54 # Show the plot
55 plt.show()
```

B.2.1 Calculating The MREI

Code listing B.6: Calculating the MREI

```
1  def calculate_kpi(df, request_value):
2      """
3      Calculate the percentage of container usage memory datapoints under the set
           request.
4
5      :param df: DataFrame with container memory usage data
6      :param request_value: Request limit for the container
7      :return: KPI as a percentage
8      """
9      under_request = df[df['y'] < request_value]
10     percentage = (len(under_request) / len(df)) * 100
11     return percentage
```

B.3 ARIMA

Determining Differencing

Code listing B.7: Determining differencing for the ARIMA model

```
1
2  adf_p_value_0 = sm.tsa.stattools.adfuller(df1['Containers: Memory usage'])[1]
3  adf_p_value_1 = sm.tsa.stattools.adfuller(df1['Containers: Memory usage'].diff().
       dropna())[1]
4  adf_p_value_2 = sm.tsa.stattools.adfuller(df1['Containers: Memory usage'].diff().
       diff().dropna())[1]
5
6  # Create a single figure with the desired layout
7  fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(18, 8))
8
9  # Without Differencing
```

```
axes[0, 0].set_title(f'Without Differencing (p={adf_p_value_0:.2e})')
axes[0, 0].plot(df1['Containers: Memory usage'])
sm.graphics.tsa.plot_acf(df1['Containers: Memory usage'].dropna(), ax=axes[1, 0])

# 1st Order Differencing
axes[0, 1].set_title(f'1st Order Differencing (p={adf_p_value_1:.2e})')
axes[0, 1].plot(df1['Containers: Memory usage'].diff())
sm.graphics.tsa.plot_acf(df1['Containers: Memory usage'].diff().dropna(), ax=axes
    [1, 1])

# 2nd Order Differencing
axes[0, 2].set_title(f'2nd Order Differencing (p={adf_p_value_2:.2e})')
axes[0, 2].plot(df1['Containers: Memory usage'].diff().diff())
sm.graphics.tsa.plot_acf(df1['Containers: Memory usage'].diff().diff().dropna(), ax
    =axes[1, 2])

plt.savefig(f'memory_usage_plots_with_adf_c{container}.png', dpi=300, bbox_inches='
    tight')

# Show the plots
plt.show()
```

B.3.1 ACF and PACF

Code listing B.8: Calculating ACF and PACF for ARIMA

```
stationary_data = df1['Containers: Memory usage'].diff().dropna()
# Create ACF and PACF plots
fig, axes = plt.subplots(2, 1, figsize=(12, 8))

sm.graphics.tsa.plot_acf(stationary_data, lags=40, ax=axes[0])
sm.graphics.tsa.plot_pacf(stationary_data, lags=40, ax=axes[1])

plt.savefig(f'memory_usage_plots_PACFnACF_c{container}.png', dpi=300, bbox_inches='
    tight')
plt.show()
```

B.3.2 Fitting the ARIMA model

Code listing B.9: Fitting the Arima model

```

memory_usage = df1['Containers: Memory usage']
```

```
3
4  train_size = int(len(memory_usage) * 0.80)  # 80% of the data for training
5  train_data = memory_usage[:train_size]
6  test_data = memory_usage[train_size:]
7
8  p,d,q = 1,2,1
9
10 model = ARIMA(train_data, order=(p, d, q))
11 results = model.fit()
12
13 predictions = results.predict(start=train_size, end=len(memory_usage) - 1, dynamic=
       True)
14
15 # Plot the training data, test data, and predictions
16 fig, ax = plt.subplots(figsize=(12, 6))
17 ax.set_xlabel('Date')
18 ax.set_ylabel('Memory Usage (MB)')
19 ax.set_title(f'Memory Usage Over Time with ARIMA Predictions for c{container}')
20 ax.grid()
21
22 # Customizing date ticks and format
23 ax.xaxis.set_major_locator(mdates.AutoDateLocator())
24 ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
25 plt.setp(ax.get_xticklabels(), rotation=30, ha='right')
26
27 ax.plot(df1['Date'][:train_size], train_data, label='Training Data')
28 ax.plot(df1['Date'][train_size:], test_data, label='Test Data')
29 ax.plot(df1['Date'][train_size:], predictions, label='Predictions')
30
31 ax.legend()
32
33 # Save and show the plot
34 plt.savefig(f'arima_memory_usage_c{container}.png', dpi=300, bbox_inches='tight')
35 plt.show()
```

B.3.3 Test Data Underprediction Rate (TDUR)

Code listing B.10: Calculating TDUR

```
1
2  def calculate_kpi_predictions(df, predictions):
3      """
4      Calculate the number of container usage memory datapoints under the predicted
           values.
```

```
5
6       :param df: DataFrame with container memory usage data
7       :param predictions: Predicted values from the ARIMA model
8       :return: Number of datapoints under the predicted line
9       """
10      test_data = df[-len(predictions):]  # Get the test data corresponding to the
            predictions
11      under_prediction = test_data[test_data['Containers: Memory usage'] <
            predictions]
12      percentage = (len(under_prediction) / len(test_data)) * 100
13      return percentage
14
15  # Assuming the 'predictions' variable contains the ARIMA model predictions
16  kpi = calculate_kpi_predictions(df1, predictions)
17  print(kpi)
```

B.3.4 MAE, MAPE and RMSE

Code listing B.11: Calculating MAE, MAPE, and RMSE

```
1
2   def mean_absolute_percentage_error(y_true, y_pred):
3       y_true, y_pred = np.array(y_true), np.array(y_pred)
4       return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
5
6   mae = np.mean(np.abs(predictions - test_data))
7   mape = mean_absolute_percentage_error(test_data, predictions)
8   rmse = np.sqrt(np.mean((test_data - predictions) ** 2))
```

B.4 Facebook's Prophet Model

Code listing B.12: Facebook's Prophet Model

```
1   model = Prophet()
2
3   df = pd.DataFrame({'ds': df1['Date'], 'y': df1['Containers: Memory usage']})
4   df['ds'] = pd.to_datetime(df['ds'])
5
6   # Split the data into train and test sets
7   train_size = int(len(df) * 0.8)  # 80% of the data for training, 20% for testing
8   train_df = df[:train_size]
9   test_df = df[train_size:]
```

```

# Fit the model on the training data
model.fit(train_df)

# Make predictions on the test data
future = model.make_future_dataframe(periods=len(test_df), freq='5min',
    include_history=False)
forecast = model.predict(future)

# Plot the forecast

fig = model.plot(forecast)
ax = fig.gca()
ax.set_title(f'Prophet Forecast for c{container}')
ax.set_xlabel('Date')
ax.set_ylabel('Memory Usage (MB)')

# Plot the test data
ax.scatter(test_df['ds'], test_df['y'], color='purple', label='Test Data',s=10)

# Add a legend
ax.legend(['Observed', 'Forecast', 'Uncertainty Interval', 'Test Data'])
plt.savefig(f'prophetC{container}.png', dpi=300, bbox_inches='tight')

plt.show()
```

B.5 P+LP

Code listing B.13: Finalized P+LP model, included plotting

```
model = Prophet(changepoint_prior_scale=0.01)

df = pd.DataFrame({'ds': df1['Date'], 'y': df1['Containers: Memory usage']})
df['ds'] = pd.to_datetime(df['ds'])

orig_req=15
orig_lim=15

# Split the data into train and test sets
train_size = int(len(df) * 0.8)  # 80% of the data for training, 20% for testing
train_df = df[:train_size]
test_df = df[train_size:]
```

```
13
14 # Fit the model on the training data
15 model.fit(train_df)
16
17 # Make predictions on the test data
18 future = model.make_future_dataframe(periods=len(test_df), freq='60min',
       include_history=False)
19 forecast = model.predict(future)
20
21 # Plot the forecast
22 fig = model.plot(forecast)
23 ax = fig.gca()
24 ax.set_title(f'P+LP Forecast for c{container}')
25 ax.set_xlabel('Date')
26 ax.set_ylabel('Memory Usage (MB)')
27
28 # Plot the test data
29 ax.scatter(test_df['ds'], test_df['y'], color='purple', label='Test Data', s=10)
30
31 # Calculate the new request and limit lines
32 new_request = np.mean(forecast['yhat']) * 1.05  # Adding 10% safety margin
33
34 max_observed = max(train_df['y'])
35 max_predicted = max(forecast['yhat_upper'])
36 higher_value = max(max_observed, max_predicted)
37 new_limit = higher_value * 1.15  # 10% over the higher value
38
39 # Old request and limit lines calculation
40 safety_margin_percentage = 15  # Additional memory allocated as a percentage of the
       max usage
41 safety_margin = max(train_df['y']) * (safety_margin_percentage / 100)
42 typical_memory_usage = np.mean(train_df['y']) * 1.05
43
44 # Define the problem
45 prob = pulp.LpProblem("ContainerResourceAllocation", pulp.LpMinimize)
46
47 # Variables
48 x_request = pulp.LpVariable("x_request", lowBound=typical_memory_usage)
49 x_limit = pulp.LpVariable("x_limit", lowBound=max(train_df['y']) + safety_margin)
50
51 # Objective function
52 prob += x_request
53
54 # Constraints
55 prob += x_request <= x_limit
```

```

# Solve the problem
status = prob.solve()

# Get the old request and limit values
req_value = pulp.value(x_request)
lim_value = pulp.value(x_limit)
```

B.6 Performance Score

Code listing B.14: Calculating Performance Score

```
def calculate_score(mrei, resource_request_difference, ideal_target=95, wiggle_room
    =4, mrei_weight=0.7, diff_weight=0.3):
    mrei_difference = abs(mrei - ideal_target)
    score_factor = abs(1 - (mrei_difference / ideal_target))

    if resource_request_difference > 0:
        resource_request_difference = 0

    if mrei == ideal_target:
        performance_score = 100
    elif mrei_difference <= wiggle_room:
        mrei_component = score_factor * mrei_weight * 100
        diff_component = abs(resource_request_difference) * diff_weight
        performance_score = mrei_component + diff_component
    else:
        mrei_component = score_factor **2 * mrei_weight * 100
        print(mrei_component)
        diff_component = abs(resource_request_difference) * diff_weight

        performance_score = mrei_component + diff_component

    # Clip the performance score to be between 0 and 100
    performance_score = max(0, min(100, performance_score))

    return performance_score

# Example usage
mrei = 100
resource_request_difference = +80
score = calculate_score(mrei, resource_request_difference)

```

```
print(f"Performance Score: {score}")
```