Final Year Project
Group 01

LATENCY-CONSTRAINED VERTICAL SCALING IN KUBERNETES

An Adaptive Online Learning Platform for Resource Optimization

In Collaboration With WSO2

Mid-Evaluation Presantation

OUR TEAM

Harith Abeysinghe

E/19/003

Udaya Jayarathna

E/19/155

Wishula Jayathunga

E/19/166

PROBLEM STATEMENT

Overprovisioned Kubernetes Cluster

CPU Usage:

Memory Usage:

Issues:

- High infrastructure costs
- Resource waste (60-70%)
- Static allocation
- Manual scaling decisions
- Poor cost efficiency
- Over-allocated pods

Wasted Resources

~$50k-100k/month unused capacity

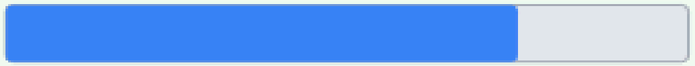
Traditional: 30-40% efficiency

OPTIMIZE

ML-Based Adaptive Tuning

CPU Usage:

Memory Usage:



Benefits:

- 60-80% cost reduction
- Real-time resource optimization
- Predictive scaling
- Automated decision making
- Dynamic workload adaptation
- Right-sized containers

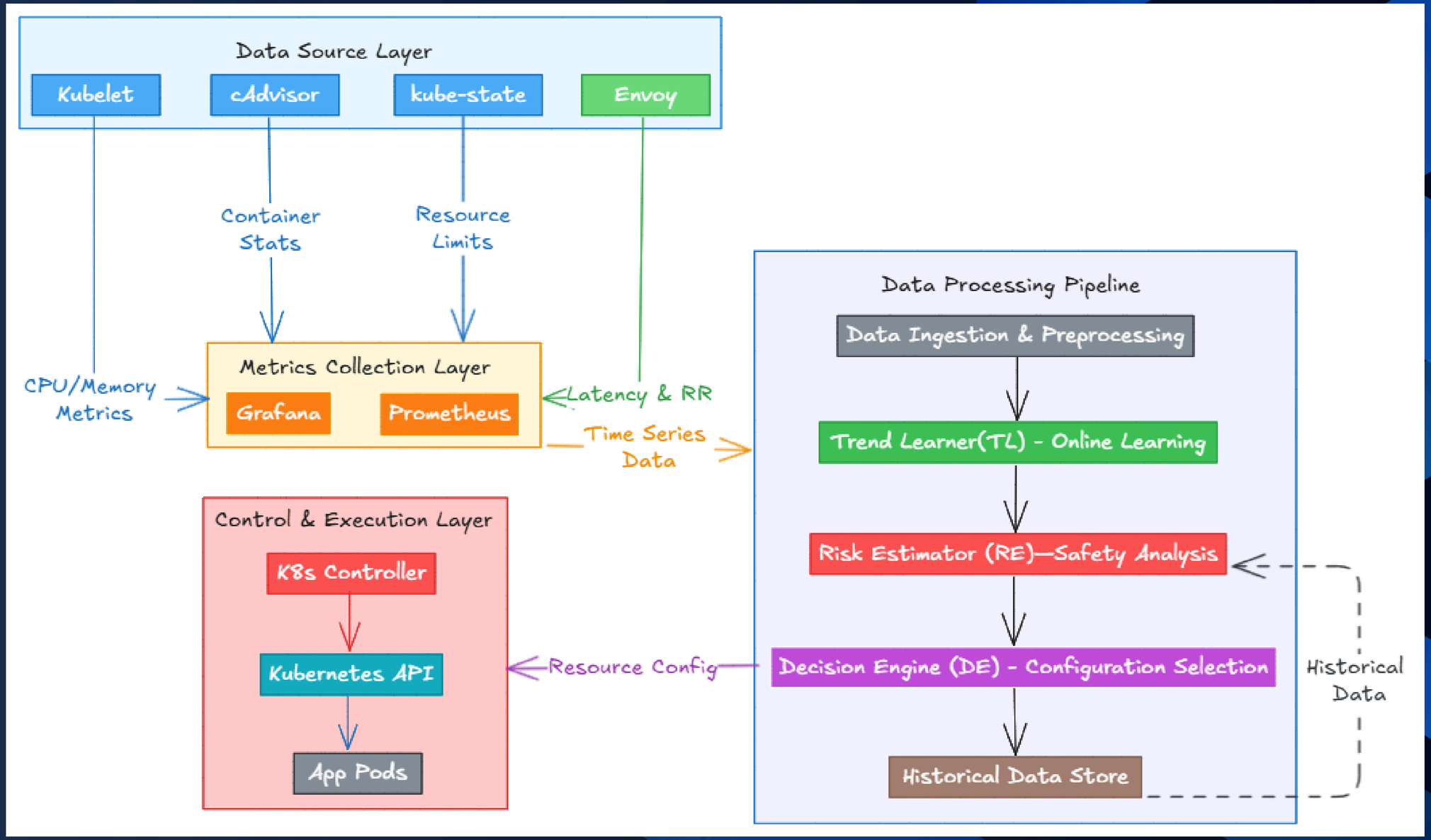
ML-Powered Intelligence

Pattern Recognition • Predictive Analytics
Anomaly Detection • Auto-scaling

ML-Adaptive: 75-85% efficiency

SYSTEM ARCHITECTURE

High Level Architecture



WHAT WE HAVE DONE?

- Experimental Analysis of Latency
- Data Collection
- Data Ingestion & Preprocessing
- Trend Learner Experimenting
- Kubernetes Controller Development

WHAT WE HAVE TO DO?

- Look into more literature to find the baselines of latency
- Experimental Analysis with varying Workloads
- Experimental Analysis with resource requests
- Risk Estimator Design
- Decision Engine Design

PROBLEMS ENCOUNTERED

Data collection Module Development

- **Lack of Native latency Metrics in Services**
  - Used envoy sidecare to capture latency
- **Internal Service-to-Service Latency was Invisible**
  - deployed sidecars inside the clusters
- **Metric Noise & Spikes**
  - Applied EMA
- **High Setup Complexity of Istio/Envoy**
  - Started with standalone Envoy sidecar
- **Latency Not Aligned with Resource Metrics**
  - Used Timestamp Alingment

PROBLEMS ENCOUNTERED (CTD.)

Trend Learner Module Development

- **Model Overfitting to Spikes**
  - Used EMA
- **Memory Usage Drifting Up Slowly**
  - Added Longer Term Trend Features
- **Lack of Ground Truth for optimal resource levels**
  - Used self-supervised Learning
- **Feature Selection & Engineering Complexity**
  - Simplified Model to use only core predictors
- **Slow Feedback Loop for Learning**
  - Implemented Cooldown Window & Delayed Labelling

PROBLEMS ENCOUNTERED (CTD.)

Tooling, Infrastructure & Stability Development

- **Prometheus Retention & Query Latency**
  - Used Recording Rules
- **Metric Resolution Too Slow**
  - Increased Scrape Interval
- **Missing Data Gaps During Deployment**
  - Added backfilling Logic & Skipping Updates

AI TOOLS

deepseek

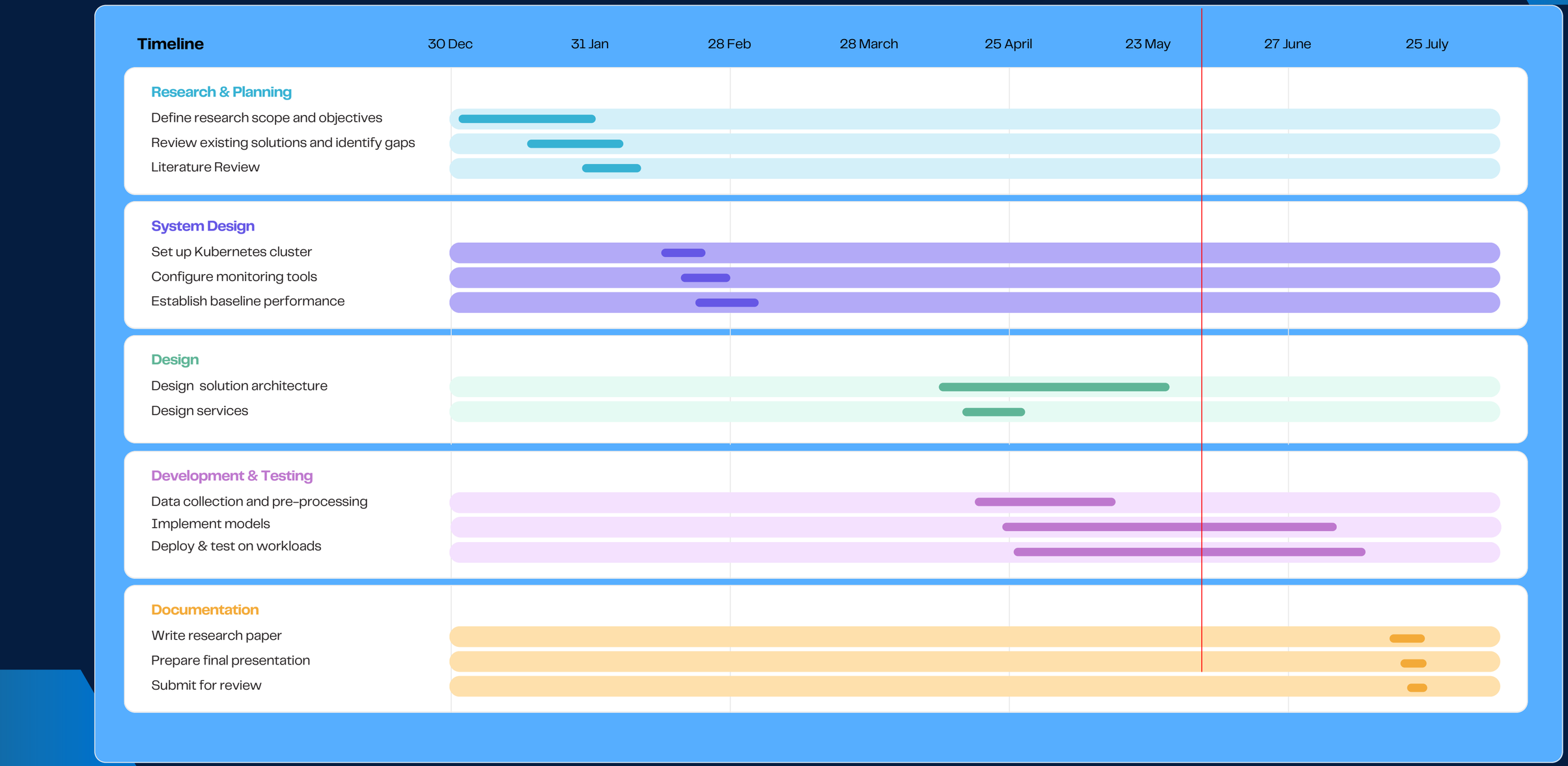
NotebookLM

ChatGPT

GitHub Copilot

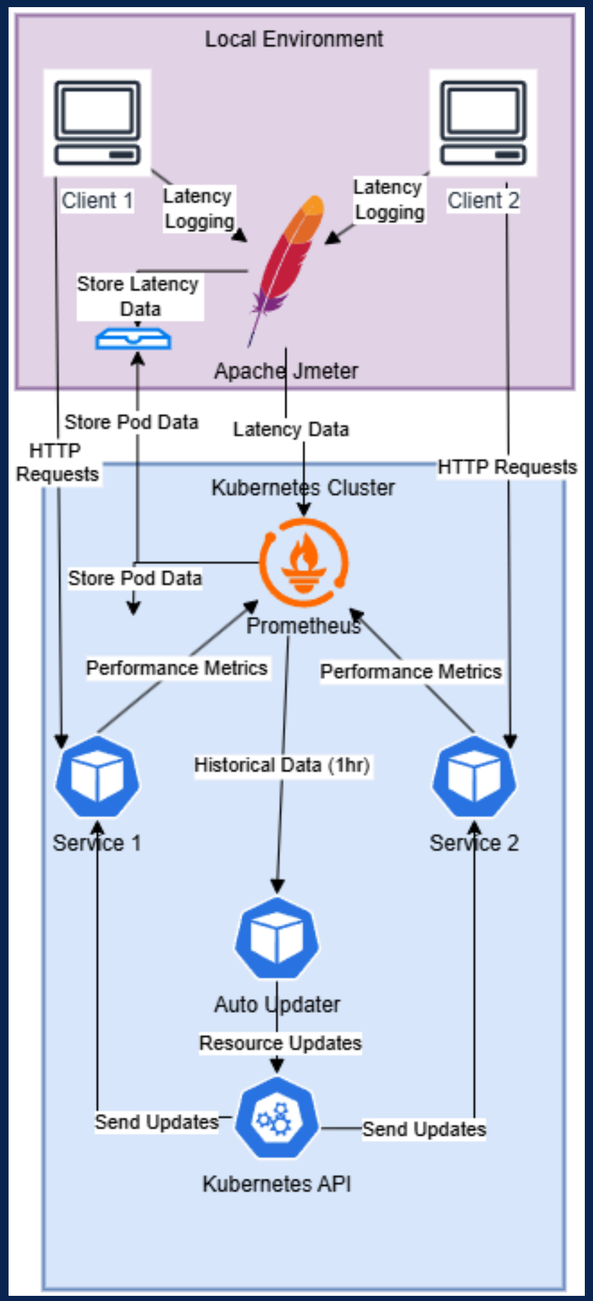
perplexity

TIMELINE


Timeline
30 Dec
31 Jan
28 Feb
28 March
25 April
23 May
27 June
25 July
Research & Planning
Define research scope and objectives
Review existing solutions and identify gaps
Literature Review
System Design
Set up Kubernetes cluster
Configure monitoring tools
Establish baseline performance
Design
Design solution architecture
Design services
Development & Testing
Data collection and pre-processing
Implement models
Deploy & test on workloads
Documentation
Write research paper
Prepare final presentation
Submit for review

DEMONSTRATION

EXPERIMENTAL SETUP



SERVER

Services

```
littleboy@littleboy-OptiPlex-7010:~$ kubectl get svc
NAME                                TYPE           CLUSTER-IP       EXTERNAL-IP      PORT(S)                      AGE
color-code-converter-service        LoadBalancer   10.107.89.149    192.168.49.104   3015:30088/TCP               22d
hash-gen-service                    LoadBalancer   10.110.94.165    192.168.49.102   3005:32567/TCP               26d
ip-geo-service                      NodePort       10.110.128.162   <none>           80:30080/TCP,8001:30081/TCP  21d
kubernetes                          ClusterIP      10.96.0.1        <none>           443/TCP                      35d
prometheus-alertmanager             ClusterIP      10.104.231.189   <none>           9093/TCP                     35d
prometheus-alertmanager-headless    ClusterIP      None             <none>           9093/TCP                     35d
prometheus-kube-state-metrics       ClusterIP      10.97.100.56     <none>           8080/TCP                     35d
prometheus-prometheus-node-exporter ClusterIP      10.98.204.132    <none>           9100/TCP                     35d
prometheus-prometheus-pushgateway   ClusterIP      10.103.132.32    <none>           9091/TCP                     35d
prometheus-server                   ClusterIP      10.105.165.79    <none>           80/TCP                       35d
rand-pw-gen-service                 LoadBalancer   10.96.80.47      192.168.49.103   3003:31913/TCP               25d
service-1-service                   LoadBalancer   10.110.34.197    192.168.49.101   3001:32360/TCP               34d
service-2-service                   LoadBalancer   10.111.88.122    192.168.49.100   3002:31662/TCP               29d
```

- An abstract layer that exposes a set of pods as a network service.
- Provides a stable IP and DNS name, even if the pods behind it change.
- Types of services:
  - ClusterIP (default) – internal-only access
  - NodePort – exposes service on node IPs at a static port
  - LoadBalancer – external access via cloud load balancers

SERVER

Pods

```
littleboy@littleboy-OptiPlex-7010:~$ kubectl get pods --field-selector=status.phase=Running
NAME                                          READY   STATUS    RESTARTS      AGE
auto-updater-c84f7dfc7-2ppdr                  1/1     Running   1 (25m ago)   29d
color-code-converter-deployment-6f77cc784f-vp8fl   1/1     Running   1 (25m ago)   15d
geoip-client-864b9ccdcf-kjlsz                 1/1     Running   1 (25m ago)   21d
geoip-client1-7c65f4b47b-7qjr5                1/1     Running   1             21d
hash-gen-deployment-65d778fdf8-g2m7c          1/1     Running   0             3m54s
ip-geo-service-589bd78c55-5jt59               1/1     Running   1 (25m ago)   21d
prometheus-alertmanager-0                     1/1     Running   2 (21m ago)   35d
prometheus-kube-state-metrics-c7cbfcc5d-zvnrk 1/1     Running   5 (19m ago)   35d
prometheus-prometheus-node-exporter-n9r4k     1/1     Running   1             35d
prometheus-prometheus-pushgateway-79c99bd4b8-wrbsp   1/1     Running   1             35d
prometheus-server-564448f5dc-b4kdh            2/2     Running   4 (19m ago)   15d
rand-pw-gen-deployment-868d98b687-vzfbc       1/1     Running   0             3m45s
service-1-deployment-6c8d55bdd8-nzq8h         1/1     Running   0             4m38s
service-2-deployment-789df4448b-z2zdz_        1/1     Running   0             4m37s
```

- The smallest deployable unit in Kubernetes.
- Shares storage, networking, and configuration as a single execution environment.
- Metrics are collected from,
  - Prime Verifier Service (Service 1 Deployment)
  - Echo Service (Service 2 Deployment)
  - Hash Generator Service (Hash Gen Deployment)
  - Random Password Generator Service (Rand PW Gen Deployment)

DATA SOURCES

- CSV logs: Timestamp, Service, CPU/Memory Requests, Limit, Usage, Latency

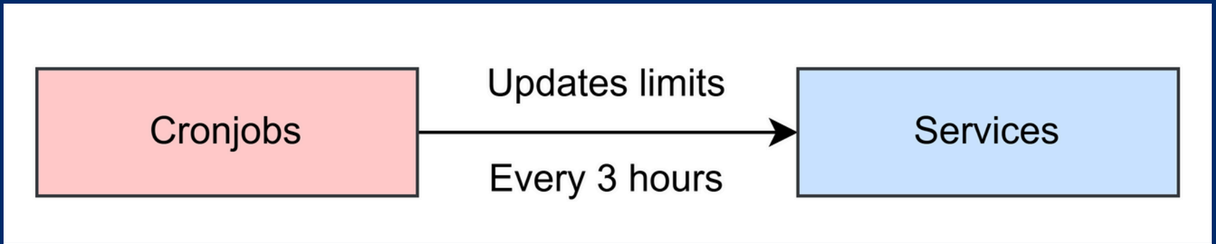
```
Timestamp,Service,CPU Request,Memory Request,CPU Limit,Memory Limit,Latency,CPU Usage,Memory Usage
2025-05-18T13:51:52.403000,hash-gen-deployment,0.1,134217728,0.12,209715200,0.017687878048750705,0.02247430787447146,161038336
2025-05-18T13:52:22.484000,hash-gen-deployment,0.1,134217728,0.12,209715200,0.018044501152214797,0.02279216977318943,161038336
2025-05-18T13:52:52.563000,hash-gen-deployment,0.1,134217728,0.12,209715200,0.018044502775255888,0.02347167807730682,161038336
2025-05-18T13:53:22.639000,hash-gen-deployment,0.1,134217728,0.12,209715200,0.018369237702859875,0.023668347092793265,161038336
2025-05-18T13:53:52.719000,hash-gen-deployment,0.1,134217728,0.12,209715200,0.018961007109451672,0.024251833032594285,161038336
```

- Metrics Sources: Prometheus, node-exporter, kube-state-metrics

```
prometheus-alertmanager                ClusterIP   10.104.231.189   <none>   9093/TCP   35d
prometheus-alertmanager-headless       ClusterIP   None             <none>   9093/TCP   35d
prometheus-kube-state-metrics          ClusterIP   10.97.100.56     <none>   8080/TCP   35d
prometheus-prometheus-node-exporter    ClusterIP   10.98.204.132    <none>   9100/TCP   35d
prometheus-prometheus-pushgateway      ClusterIP   10.103.132.32    <none>   9091/TCP   35d
prometheus-server                      ClusterIP   10.105.165.79    <none>   80/TCP     35d
```

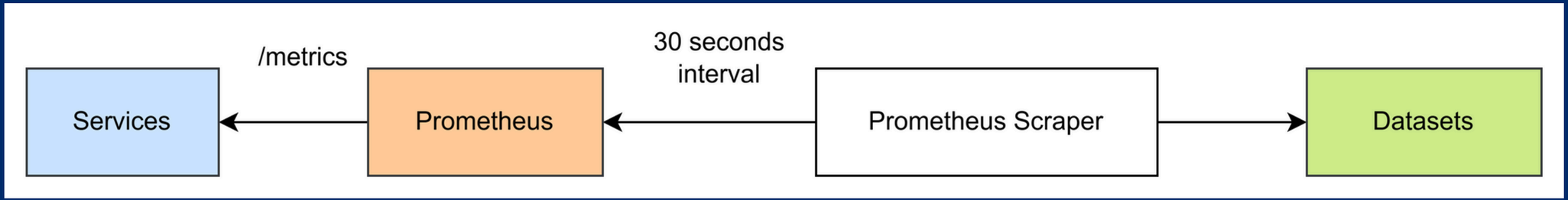
WORKFLOWS

Cronjobs



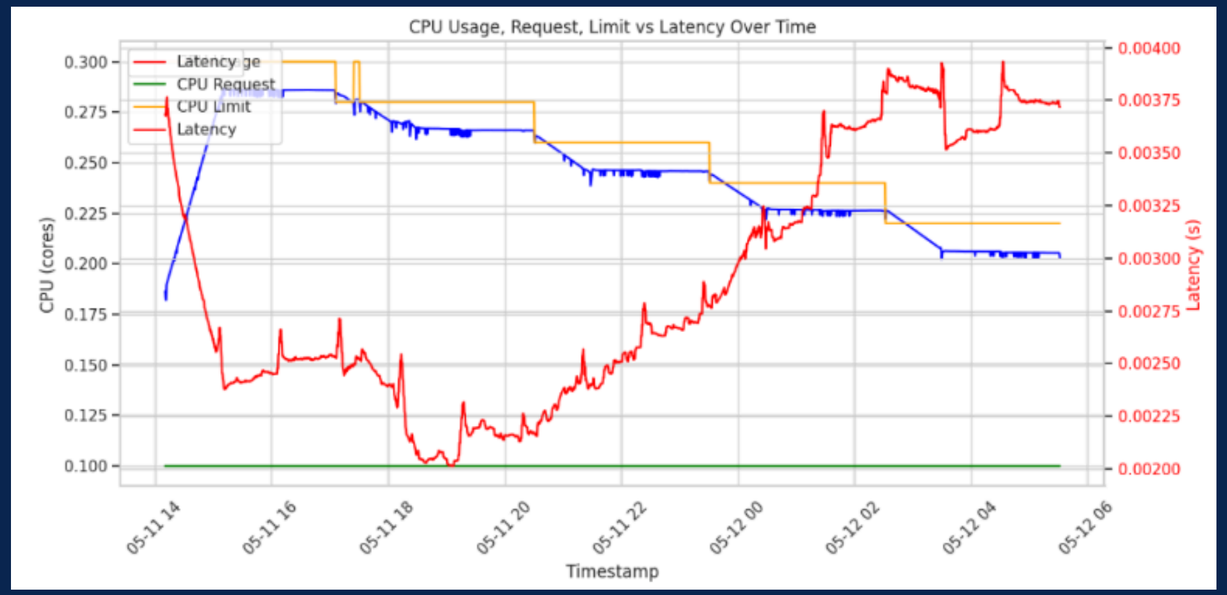
- Cronjobs are used to run scheduled tasks in Kubernetes.
- Each scheduled run creates a Job that executes to completion.

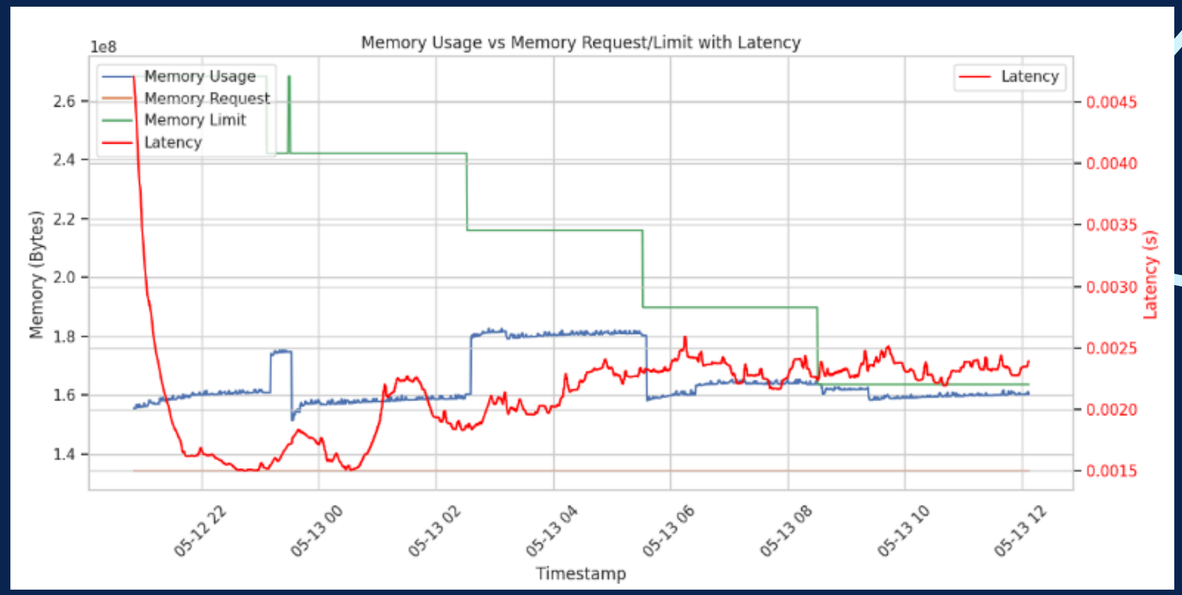
Data Gathering Process

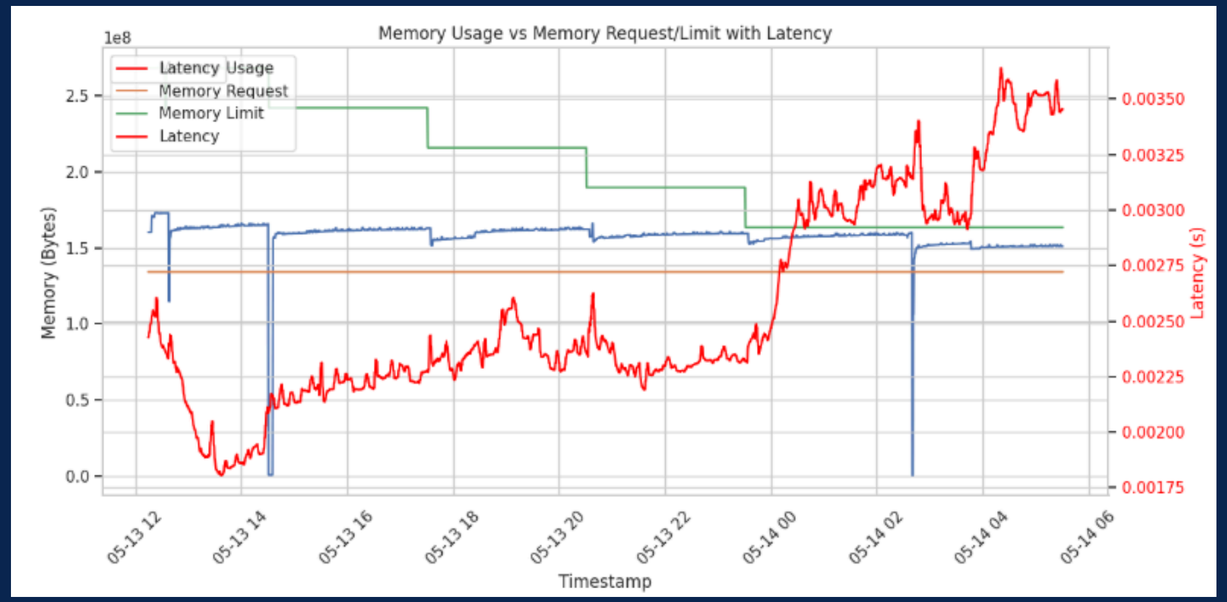


- Prometheus queries ***/metrics*** endpoints of services to monitor performance, enabling real-time observability and historical analysis.

PRIME NUMBER SERVICE

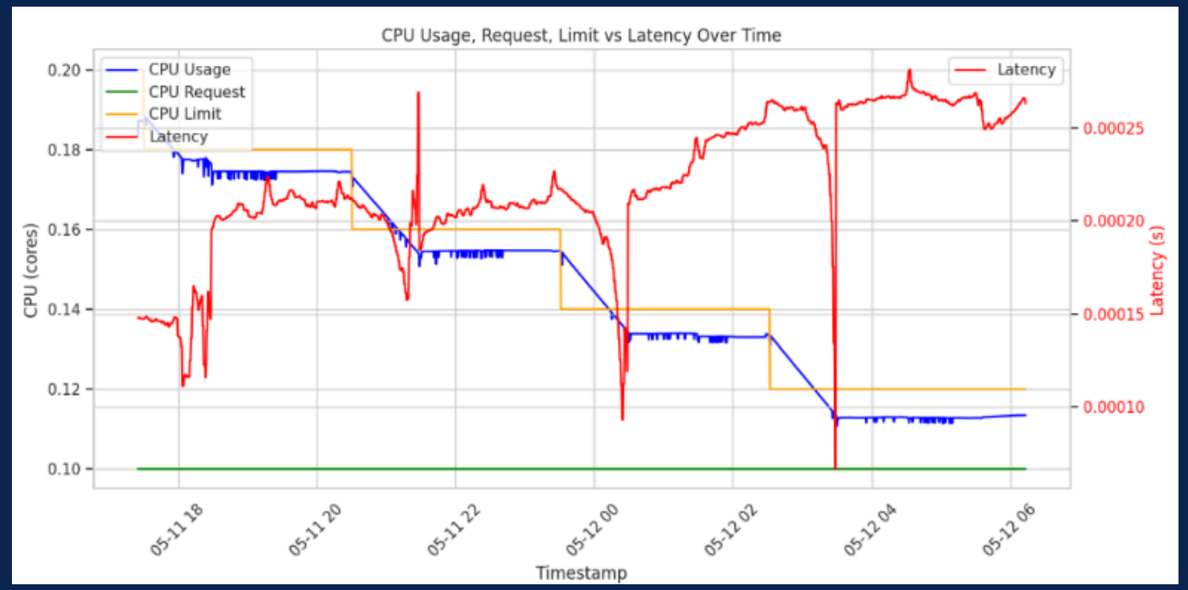


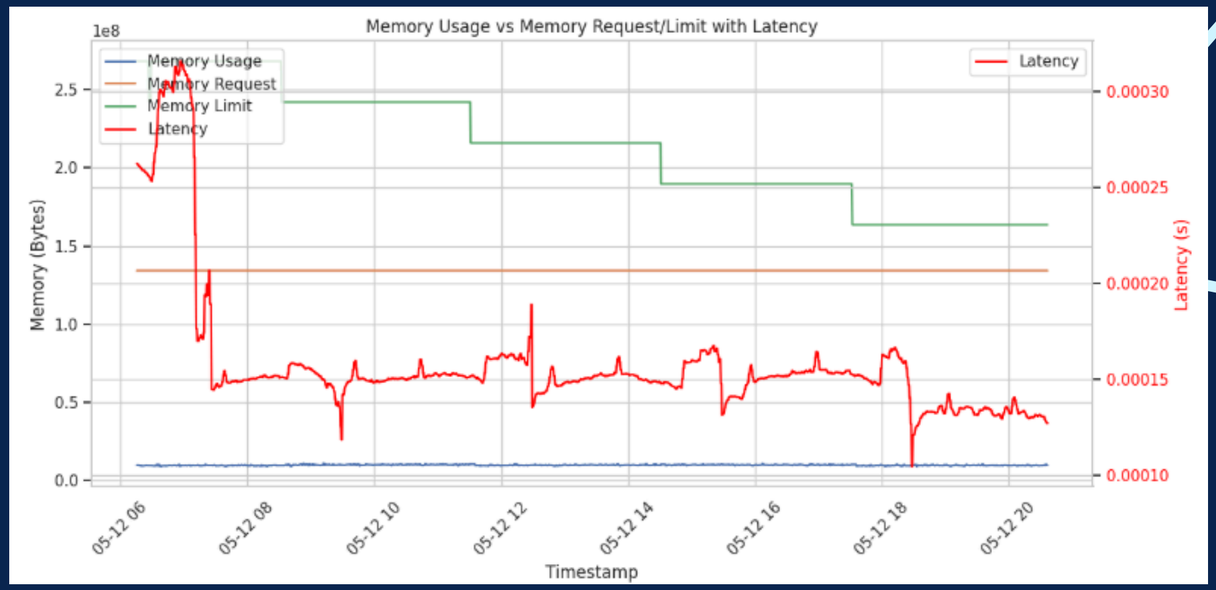


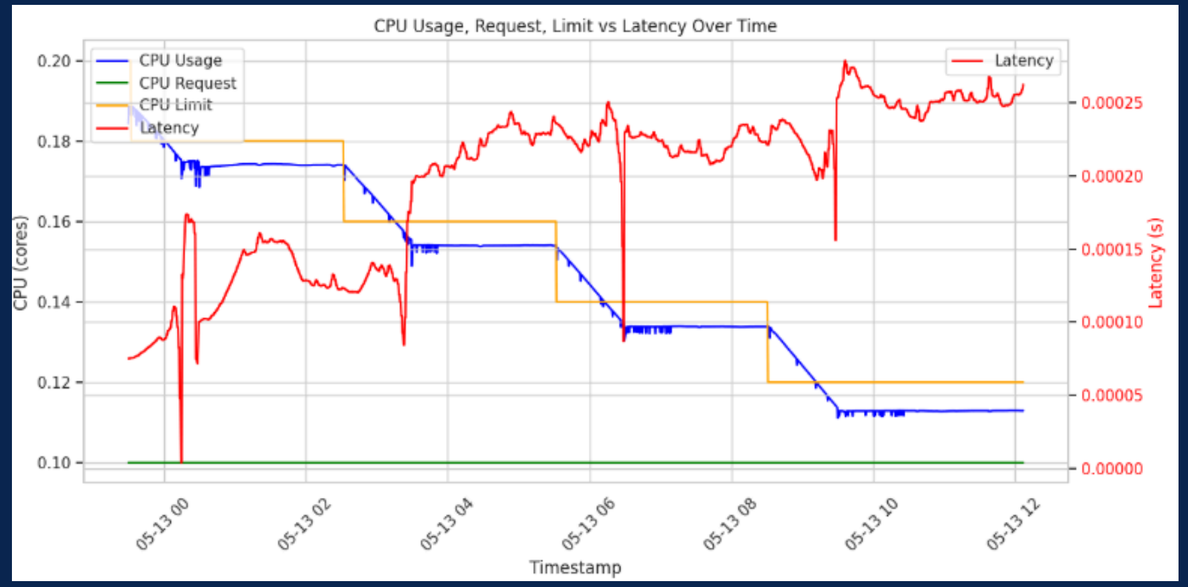


- CPU -bound
- Performance Depends on CPU Availability
- Memory alone doesn’t affect strongly
- Nonlinear Latency Jumps

ECHO SERVICE

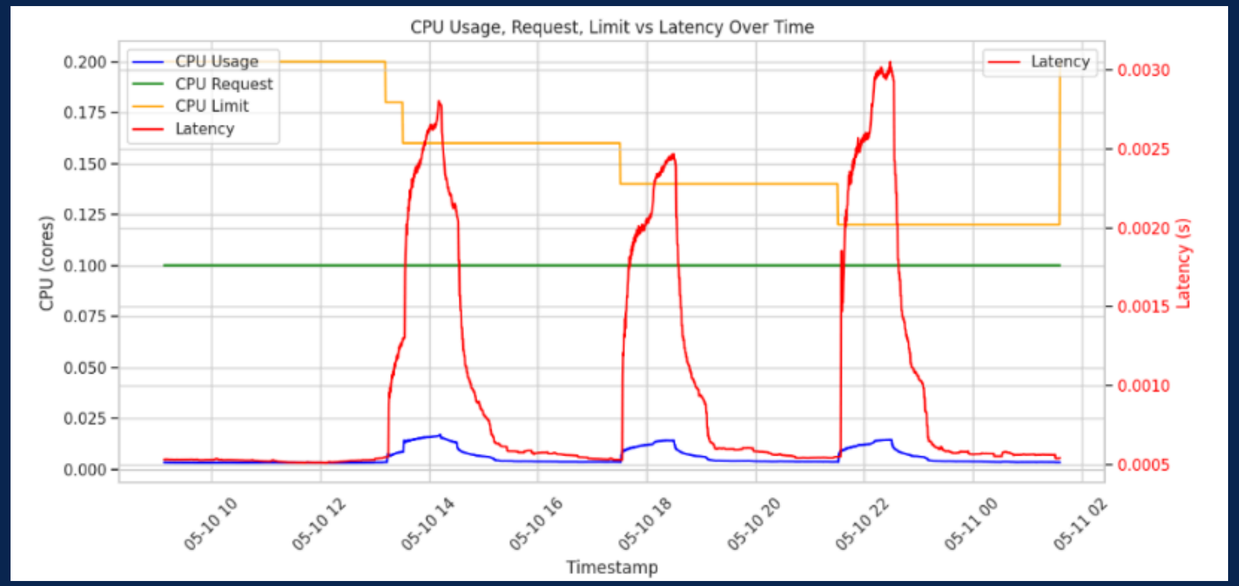


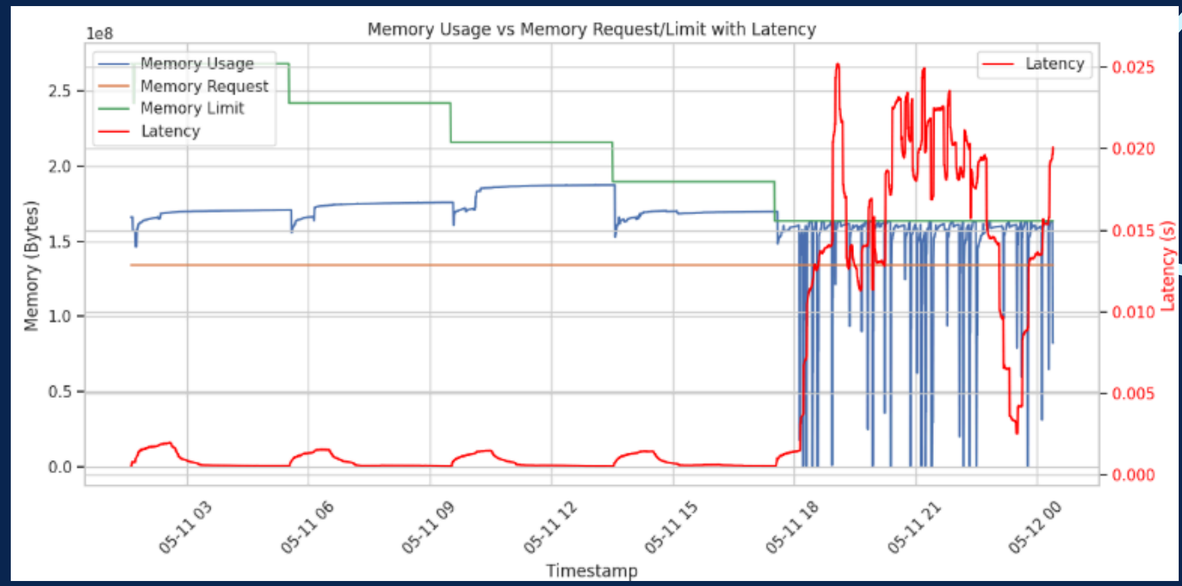


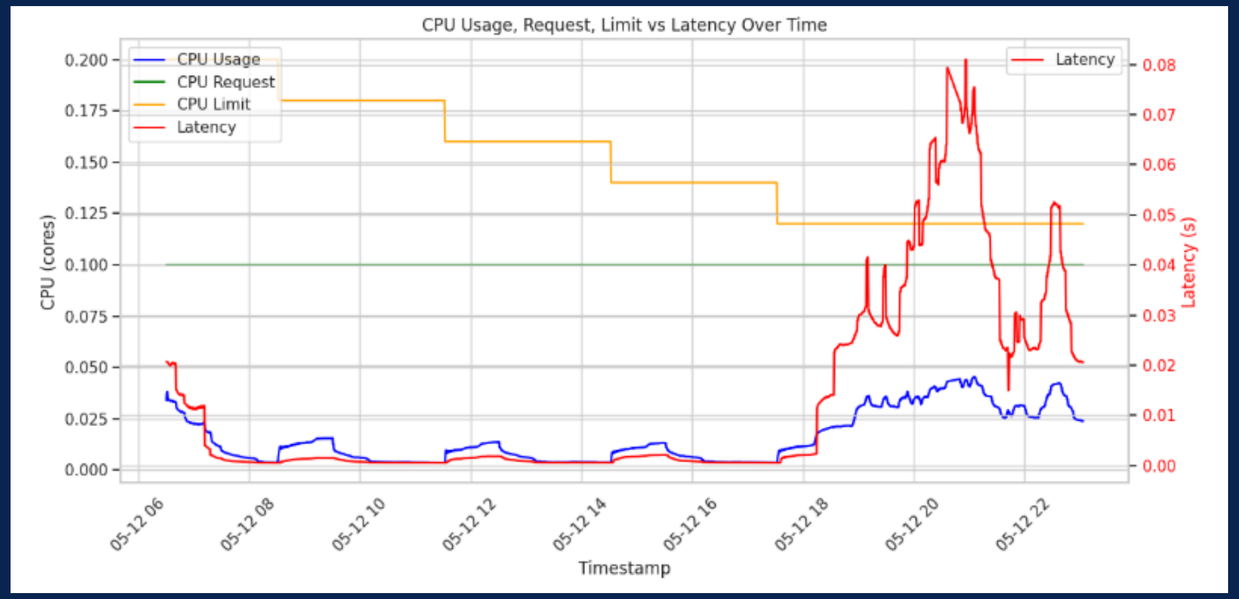


- Minimalistic & Stateless
- Miminum Resource Impact
- Ideal Resilience

HASH GENERATOR SERVICE

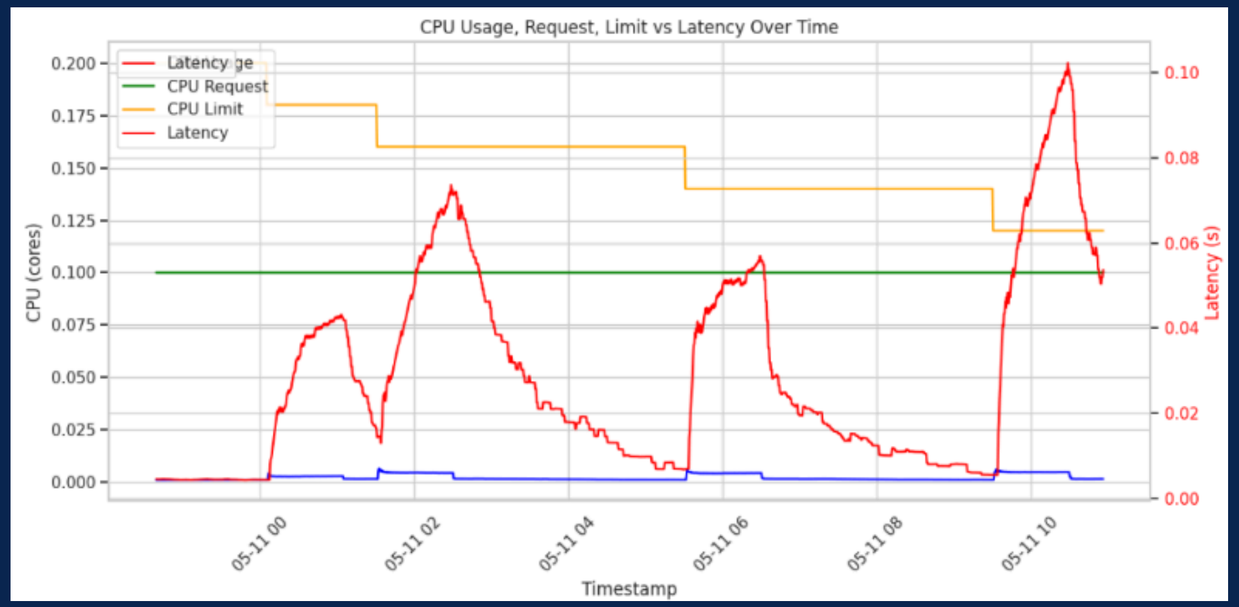


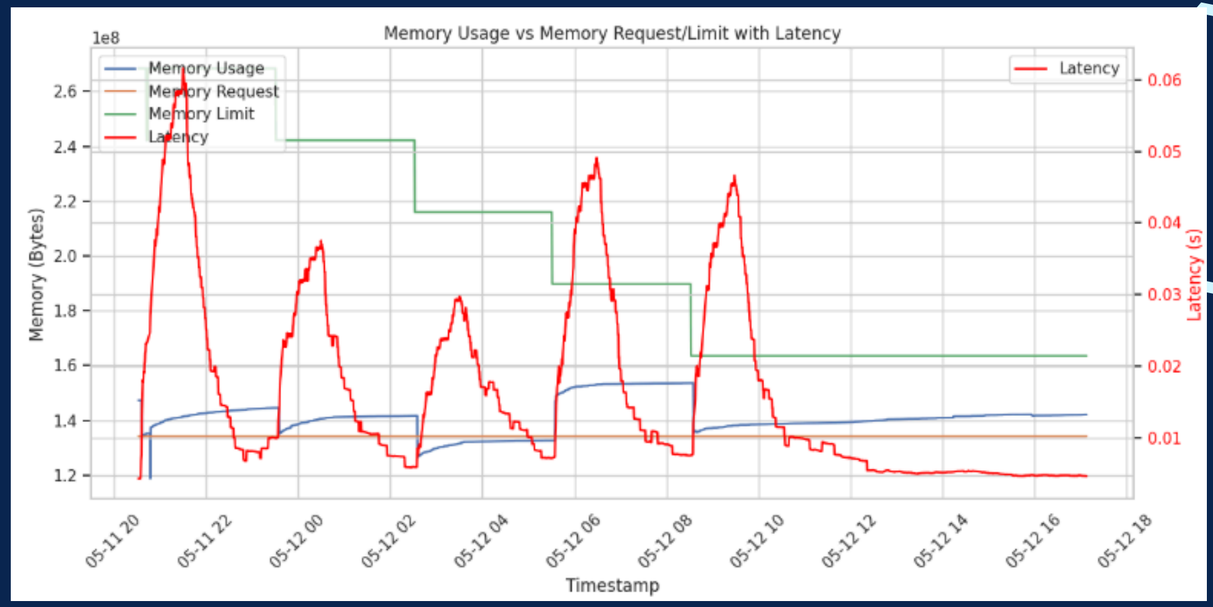


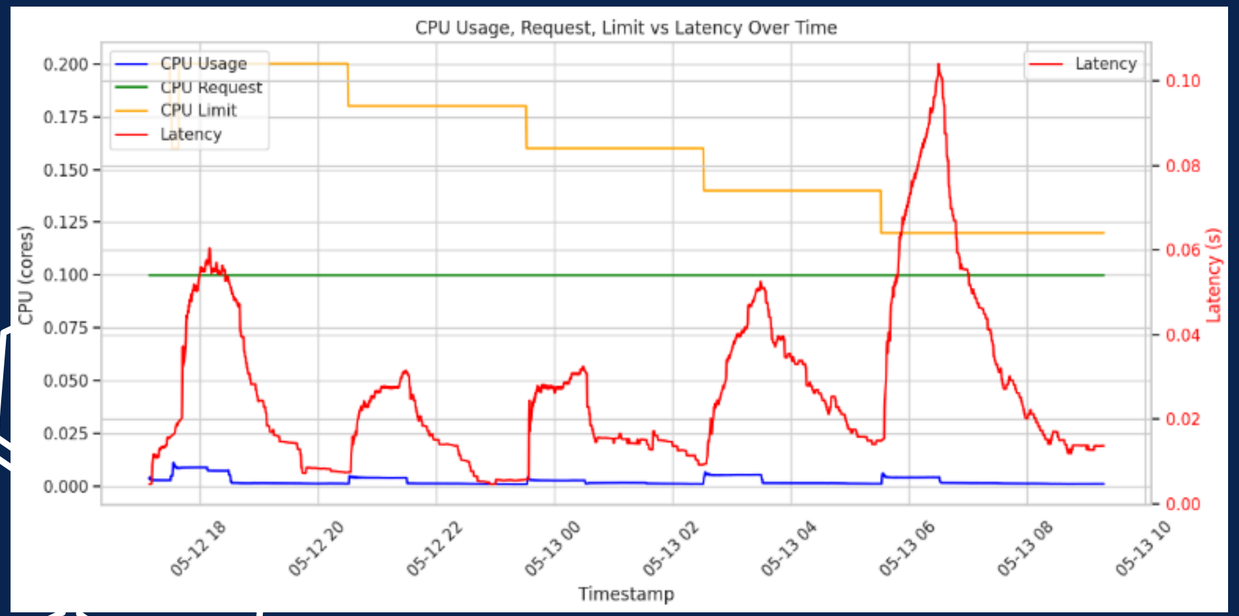


- CPU -bound
- Immediate Performance Degradation
- Minimum Memory Impact
- Sharp Latency Increment at Dual Pressure

RANDOM PASSWORD GENERATOR SERVICE







- Uses Random Number Generation
- Regular, Predictable Spikes
- Bursty Latency with Memory Reduction
- Unexpected Dips in Latency

TREND LEARNER WITH EMA & RIVER

- Trend Learner Train Structure

```python
def train(self, X, y_cpu, y_mem):
    for xi, y_cpu_i, y_mem_i in zip(X, y_cpu, y_mem):
        xi = xi.reshape(1, -1)
        xi_ema = self._apply_ema(xi)

        if self.backend.startswith("river"):
            x_dict = {f"f{i}": xi_ema[0][i] for i in range(xi_ema.shape[1])}
            self.cpu_model.learn_one(x_dict, y_cpu_i)
            self.mem_model.learn_one(x_dict, y_mem_i)
            self.is_fitted = True
        else:
            if not self.is_fitted:
                self.cpu_model.fit(xi_ema, [y_cpu_i])
                self.mem_model.fit(xi_ema, [y_mem_i])
                self.is_fitted = True
            else:
                self.cpu_model.partial_fit(xi_ema, [y_cpu_i])
```

- Adapts Depending on th Selected Backend
- Helps to Easy Switch

- EMA-enhanced regression & Execution

```python
# EMA-enhanced regression
self.ema_feature = self.alpha * X + (1 - self.alpha) * self.ema_feature

# River integration
x_dict = {f"f{i}": xi_ema[0][i] for i in range(xi_ema.shape[1])}
self.cpu_model.learn_one(x_dict, y_cpu_i)
```

- Used EMA
- Made Predictions More Stable

MATHEMATICAL EQUATIONS

- EMA Equation

$$\mathrm{EMA}_t = \alpha x_t + (1-\alpha)\mathrm{EMA}_{t-1}$$

- Ignores Short Spikes

- MAPE

$$\mathrm{MAPE} = \frac{1}{n}\sum \left| \frac{y_i - \hat{y}_i}{y_i} \right| \times 100$$

- Shows Error in Percentage Form

- Resource Usage

$$\text{Safe CPU Range} = [\hat{y} \cdot (1-m), \hat{y} \cdot (1+m)]$$

- Predicts Safe Range

TREND LEARNER RESULTS

- Fast API Endpoint Output

```
C:\Users\Wishula>curl -X POST "http://localhost:8000/predict" -H "Content-Type: application/json" -d "{\"CPU_Usage\": 0.65, \"Memory_Usage\": 231523895, \"R
equestRate\": 350.39, \"CPU_Limit\": 0.8, \"Memory_Limit\": 321678954, \"Latency\": 0.004, \"RequestRate_Delta\": 2.5, \"Hour_sin\": 0.5, \"Hour_cos\": 0.86
6}"
{"forecast":{"CPU_Usage_Forecast":0.65,"Memory_Usage_Forecast":231485792.0,"CPU_Delta":0.0,"Memory_Delta":-38103.0},"safe_range":{"cpu_range_m":[0.585,0.715
],"mem_range_mib":[208337213.0,254634371.0]},"status":"success"}
```

- Steamlit Evaluation Dashboard

MODELS

- Trained multiple regression models, including
  - Linear Regression
  - Random Forest
  - LightBGM
  - XGBoost
- Started with default configurations to establish a baseline performance
- Explored test size variations without shuffling to evaluate model behavior while maintaining temporal order.
- Introduced new variables through feature engineering to capture deeper patterns.
  - Rolling Mean
  - CPU Spikes
  - Memory Spikes
  - Latency Trend
- Evaluated each model using both train and test scores (RMSE, R^2) to detect overfitting or underfitting.

MODELS

XGBoost Model

```python
for name, df in configs.items():
    for test_size in test_sizes:
        print(f"Training XGBoost for {name} - CPU Usage with test size {test_size}")
        model_cpu = train_xgboost_model(df, "CPU Usage", test_size)

        print(f"Training XGBoost for {name} - Memory Usage with test size {test_size}")
        model_mem = train_xgboost_model(df, "Memory Usage", test_size)
        print()
```

✓ 10.0s

```
Training XGBoost for Service 1 - CPU Usage with test size 0.3
CPU Usage - Train RMSE: 0.0065, R²: 0.9505
CPU Usage - Test  RMSE: 0.0244, R²: 0.1778
Training XGBoost for Service 1 - Memory Usage with test size 0.3
Memory Usage - Train RMSE: 5.0943, R²: 0.7758
Memory Usage - Test  RMSE: 10.2621, R²: -1.4573

Training XGBoost for Service 1 - CPU Usage with test size 0.2
CPU Usage - Train RMSE: 0.0063, R²: 0.9482
CPU Usage - Test  RMSE: 0.0263, R²: -1.0619
Training XGBoost for Service 1 - Memory Usage with test size 0.2
Memory Usage - Train RMSE: 5.2501, R²: 0.7332
Memory Usage - Test  RMSE: 9.6428, R²: -0.6867
```

New Features

```python
def add_rolling_features(df, window=3):
    df = df.copy()
    df['Timestamp'] = pd.to_datetime(df['Timestamp'], format='mixed')

    df = df.sort_values(['Service', 'Timestamp'])  # Service-wise time sorting
    df.set_index('Timestamp', inplace=True)

    # Rolling averages per service
    for col in ['CPU Usage', 'Memory Usage', 'Latency']:
        df[f'{col}_RollingMean'] = df.groupby('Service')[col].transform(lambda x: x.rolling(window, min_periods=1).mean())
        df[f'{col}_RollingSTD'] = df.groupby('Service')[col].transform(lambda x: x.rolling(window, min_periods=1).std())

    # Spike detection
    df["CPU_Spike"] = df["CPU Usage"] - df["CPU Usage_RollingMean"]
    df["Memory_Spike"] = df["Memory Usage"] - df["Memory Usage_RollingMean"]

    # Latency trend direction
    df["Latency_Trend"] = df.groupby("Service")["Latency"].transform(lambda x:
                    x.diff().fillna(0).apply(lambda y: 1 if y > 0 else (-1 if y < 0 else 0)))

    df.reset_index(inplace=True)  # Reset index to include Timestamp again
    df.dropna(inplace=True)  # Optional: drop rows with NaNs from rolling
    return df
```

MODELS

CPU Predictions

Service	Model	Best R2 Score
Prime Verifier	Linear Regression	0.849
	Random Forest Regressor	0.9998
	GRU	0.9978
	LightGBM	0.8472
	XGBoost	0.988
Echo Service	Linear Regression	0.9526
	Random Forest Regressor	0.9999
	GRU	0.9992
	LightGBM	0.9972
	XGBoost	0.9968
Hash Generator	Linear Regression	0.77
	Random Forest Regressor	0.9998
	GRU	0.9989
	LightGBM	0.9956
	XGBoost	0.9851
Random Password Generator	Linear Regression	0.3363
	Random Forest Regressor	0.998
	GRU	0.9738
	LightGBM	0.9729
	XGBoost	0.9864

MODELS

Memory Predictions

Service	Model	Best R2 Score
Prime Verifier	Linear Regression	0.1759
	Random Forest Regressor	0.8453
	GRU	0.4053
	LightGBM	0.4122
	XGBoost	0.2914
Echo Service	Linear Regression	0.0315
	Random Forest Regressor	0.9991
	GRU	0.4883
	LightGBM	0.4466
	XGBoost	0.9534
Hash Generator	Linear Regression	0.202
	Random Forest Regressor	0.9975
	GRU	0.5939
	LightGBM	0.5649
	XGBoost	0.969
Random Password Generator	Linear Regression	0.0278
	Random Forest Regressor	0.9942
	GRU	0.9924
	LightGBM	0.8094
	XGBoost	0.9596

Q&A

THANK YOU

GITHUB REPOSITORY

PROJECT WEB PAGE