K8s Nemesis

Data design

Product description

A cloud-native service designed to enhance application scalability and performance in Kubernetes (K8s) environments, specifically targeting machine learning (ML) applications with unique scaling and resource requirements.

Team: Sergey Lokhmatikov, Roman Kuzmenko, Andrey Tamplon

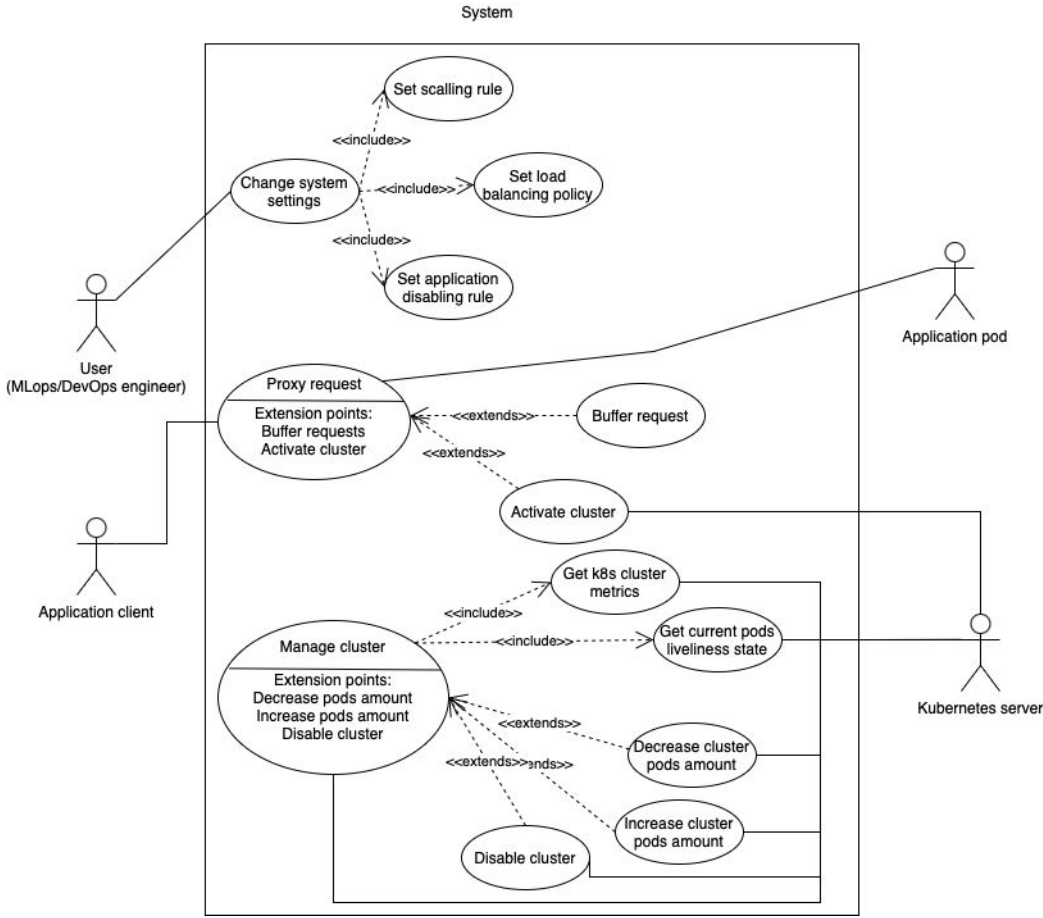
Repo: https://github.com/Lokhmat/k8s_nemesis/tree/main

Report: https://github.com/Lokhmat/k8s_nemesis/blob/main/task_11_slides.pdf

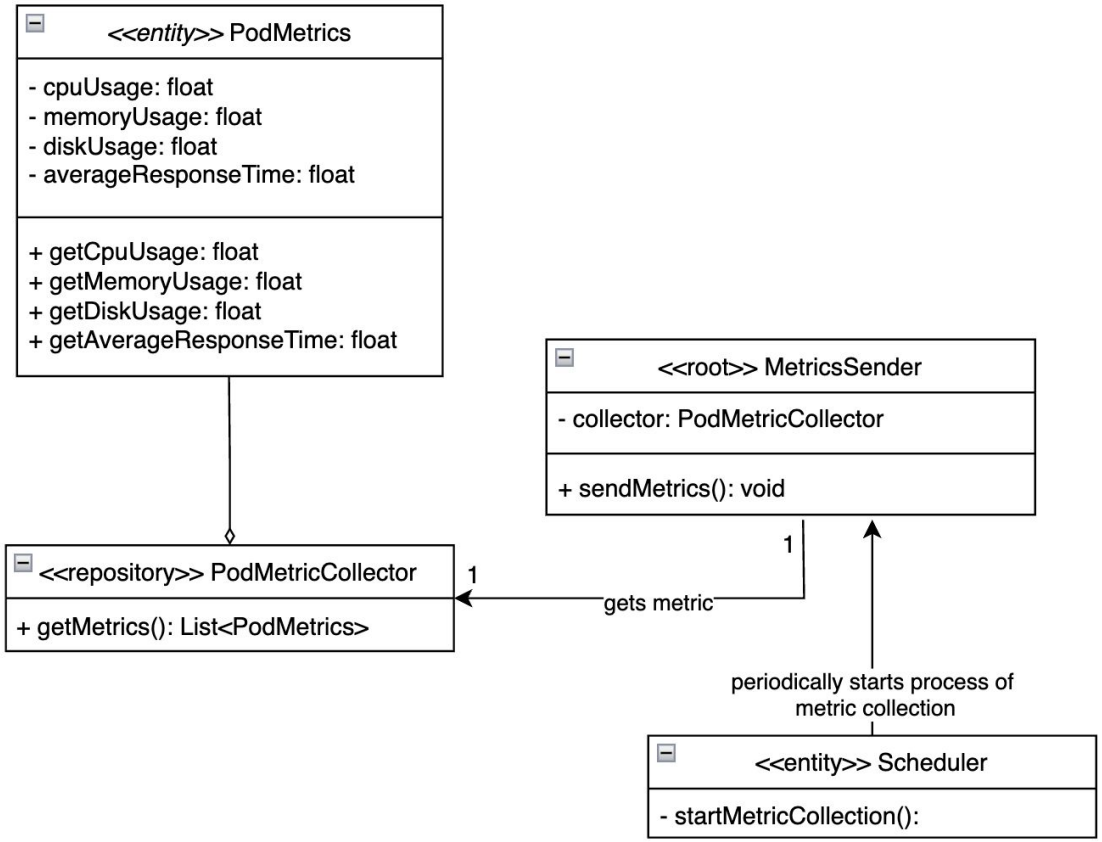
Use case diagram or event flow

Textual use case scenarios:

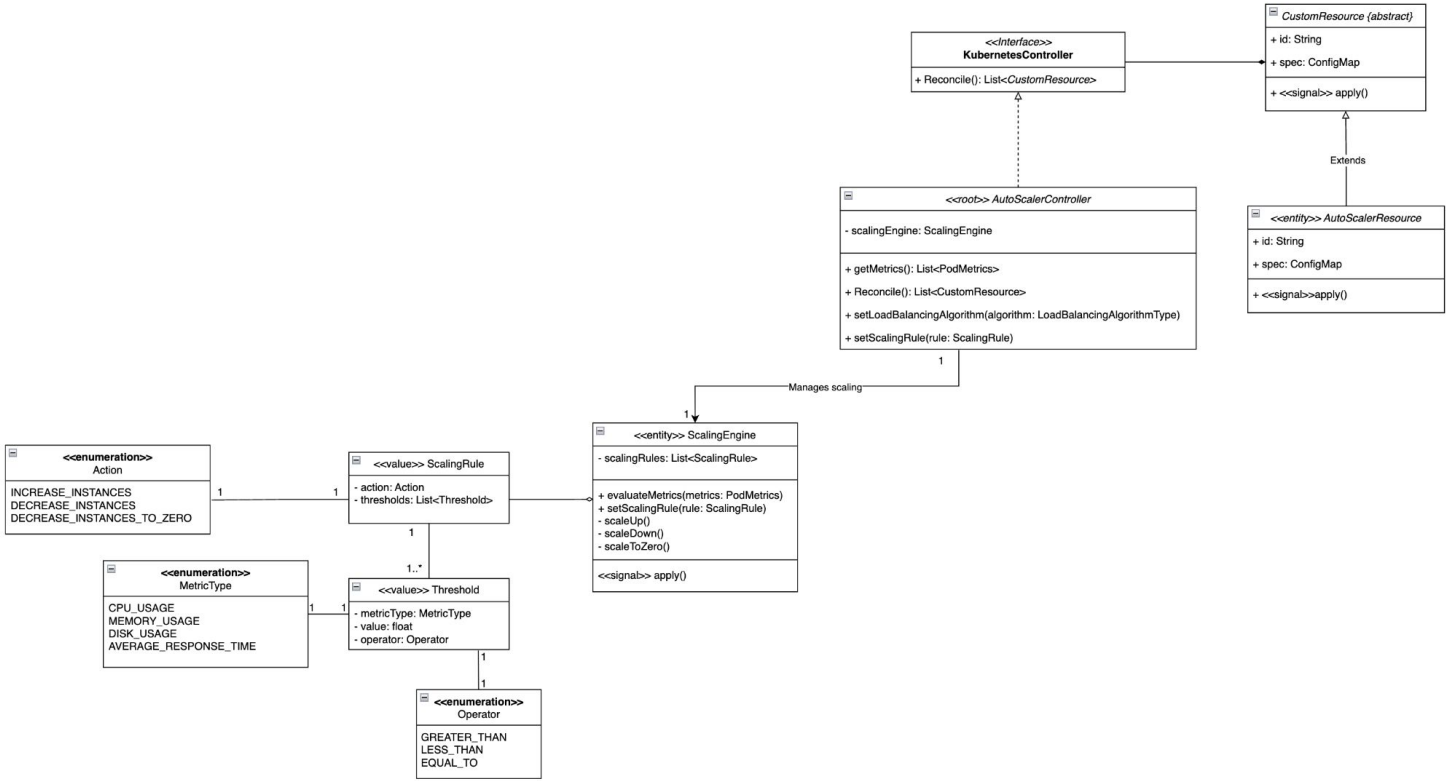
https://github.com/Lokhmat/k8s_nemesis/blob/main/final_task_materials/textual_use_cases.md



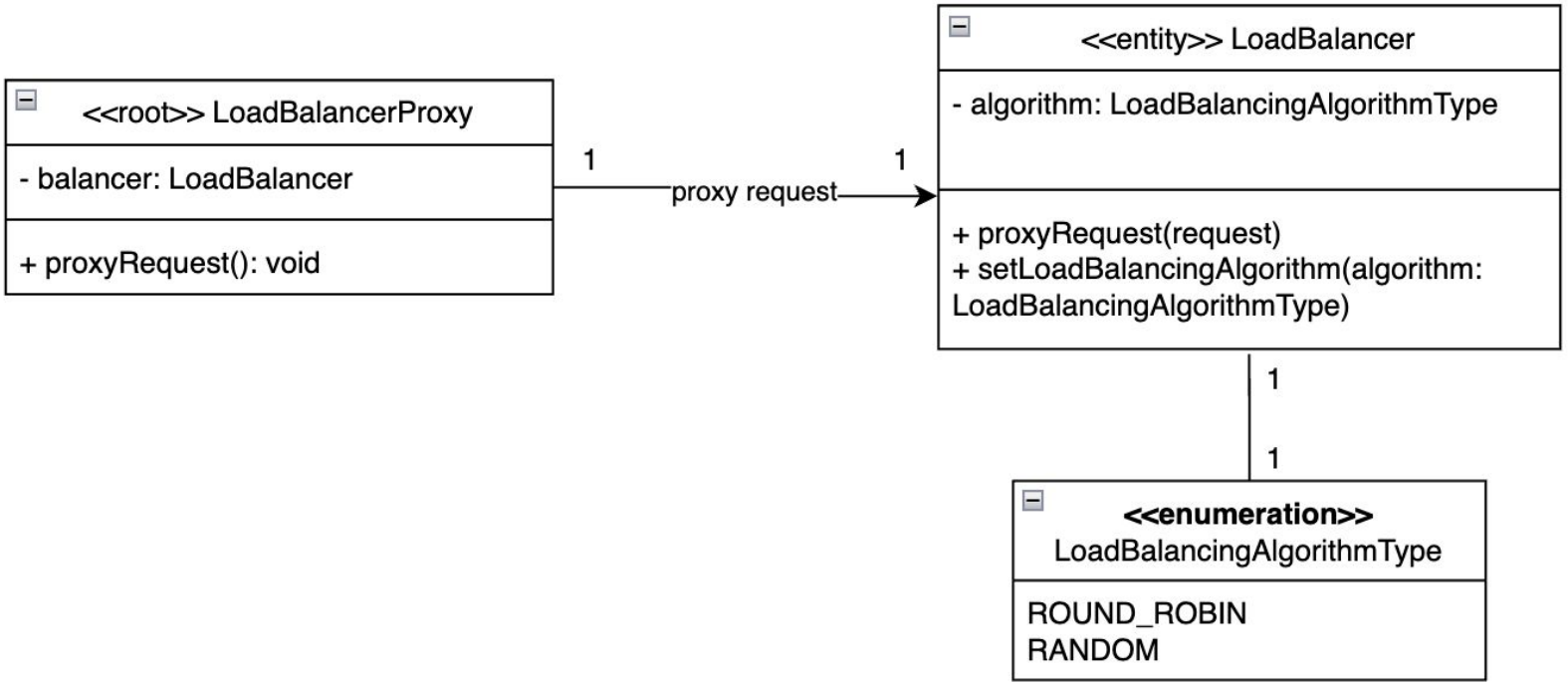
Class diagram - SideCar



Class diagram - AutoScalerController



Class diagram - LoadBalancer



API summary Autoscaler

GET	`/apis/autoscaler-group.autoscaler/v1alpha1/namespaces/{namespace}/customautoscalers`	Получить список всех CustomAutoscaler ресурсов в указанном пространстве имен
POST	`/apis/autoscaler-group.autoscaler/v1alpha1/namespaces/{namespace}/customautoscalers`	Создать новый CustomAutoscaler ресурс
GET	`/apis/autoscaler-group.autoscaler/v1alpha1/namespaces/{namespace}/customautoscalers/{name}`	Получить конкретный CustomAutoscaler ресурс
PUT	`/apis/autoscaler-group.autoscaler/v1alpha1/namespaces/{namespace}/customautoscalers/{name}`	Обновить существующий CustomAutoscaler ресурс
PATCH	`/apis/autoscaler-group.autoscaler/v1alpha1/namespaces/{namespace}/customautoscalers/{name}`	Частично обновить существующий CustomAutoscaler ресурс
POST	`/metrics`	Отправить метрики от сайдкара в автоскейлер

API summary LoadBalancer

GET	`/loadbalancer/settings`	Get current load balancing settings
PUT	`/loadbalancer/settings`	Update load balancing settings
GET	`/proxy/logs`	Get logs of client requests

Physical schema: AutoScaller

```go
type ScalingRule struct {
    ID         string      `json:"id"`
    Action     Action      `json:"action"`
    Thresholds []Threshold `json:"thresholds"`
}

type Threshold struct {
    MetricType MetricType `json:"metricType"`
    Value      float64    `json:"value"`
    Operator   Operator   `json:"operator"`
}

type Podmetric struct {
    ID                  string    `json:"id"`
    CPUUsage            float64   `json:"cpuUsage"`
    MemoryUsage         float64   `json:"memoryUsage"`
    DiskUsage           float64   `json:"diskUsage"`
    AverageResponseTime float64   `json:"averageResponseTime"`
    ReceivedAt          time.Time `json:"receivedAt"`
}

type Operator string
type MetricType string
type Action string

const (
    OperatorGreaterThan Operator = "GREATER_THAN"
    OperatorLessThan    Operator = "LESS_THAN"
    OperatorEqualTo     Operator = "EQUAL_TO"

    MetricTypeCPUUsage            MetricType = "CPU_USAGE"
    MetricTypeMemoryUsage         MetricType = "MEMORY_USAGE"
    MetricTypeDiskUsage           MetricType = "DISK_USAGE"
    MetricTypeAverageResponseTime MetricType = "AVERAGE_RESPONSE_TIME"

    ActionIncreaseInstances       Action = "INCREASE_INSTANCES"
    ActionDecreaseInstances       Action = "DECREASE_INSTANCES"
    ActionDecreaseInstancesToZero Action = "DECREASE_INSTANCES_TO_ZERO"
)
```

Physical schema: AutoScaller

```go
func saveScalingRule(rdb *redis.Client, scalingRule ScalingRule) error {
    key := fmt.Sprintf("scaling_rule:%s", scalingRule.ID)
    data, err := json.Marshal(scalingRule)
    if err != nil {
        return err
    }
    ctx := context.Background()
    err = rdb.Set(ctx, key, data, 0).Err()
    if err != nil {
        return err
    }

    return nil
}

func getScalingRule(rdb *redis.Client, id string) (*ScalingRule, error) {
    key := fmt.Sprintf("scaling_rule:%s", id)
    ctx := context.Background()
    data, err := rdb.Get(ctx, key).Result()
    if err != nil {
        return nil, err
    }

    var scalingRule ScalingRule
    err = json.Unmarshal([]byte(data), &scalingRule)
    if err != nil {
        return nil, err
    }

    return &scalingRule, nil
}
```

Physical schema: AutoScaller

```go
func getPodmetricsDuration(rdb *redis.Client, id string, duration time.Duration) ([]Podmetric, error) {
    key := fmt.Sprintf("podmetric:%s", id)

    endTime := time.Now().Unix()
    startTime := endTime - int64(duration.Seconds())

    zRangeBy := &redis.ZRangeBy{
        Min: fmt.Sprintf("%d", startTime),
        Max: fmt.Sprintf("%d", endTime),
    }

    ctx := context.Background()
    results, err := rdb.ZRangeByScoreWithScores(ctx, key, zRangeBy).Result()
    if err != nil {
        return nil, err
    }

    var podmetrics []Podmetric
    for _, result := range results {
        var podmetric Podmetric
        err = json.Unmarshal([]byte(result.Member.(string)), &podmetric)
        if err != nil {
            return nil, err
        }
        podmetrics = append(podmetrics, podmetric)
    }

    return podmetrics, nil
}
```

Physical schema: LoadBalancer

```
CREATE TYPE loadbalancer_policy AS ENUM (
    'round_robin',
    'least_connections',
    'random'
);

CREATE TABLE loadbalancer_settings (
    id SERIAL PRIMARY KEY,
    environment VARCHAR(50) NOT NULL,
    policy loadbalancer_policy NOT NULL,
    max_connections INT NOT NULL,
    timeout_seconds INT NOT NULL,
    updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_loadbalancer_settings_env ON loadbalancer_settings(environment);
```

Physical schema: LoadBalancer

```sql
CREATE TABLE proxy_logs (
    id BIGSERIAL PRIMARY KEY,
    timestamp TIMESTAMP NOT NULL,
    client_ip INET NOT NULL,
    target_url TEXT NOT NULL,
    http_method VARCHAR(10) NOT NULL,
    response_status INT NOT NULL,
    latency_ms INT,
);

CREATE INDEX idx_proxy_logs_timestamp ON proxy_logs(timestamp);
CREATE INDEX idx_proxy_logs_target_url ON proxy_logs(target_url);
```

Physical schema: SideCar

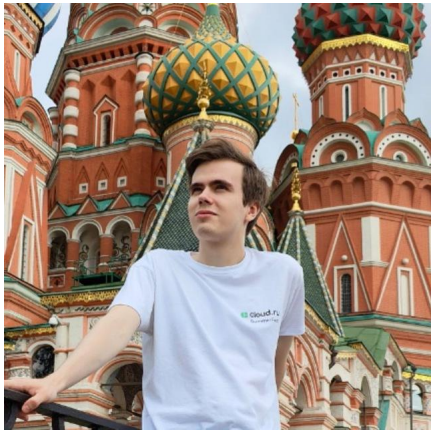
```go
func savePodmetric(rdb *redis.Client, podmetric Podmetric) error {
    key := fmt.Sprintf("podmetric:%s", podmetric.ID)
    data, err := json.Marshal(podmetric)
    if err != nil {
        return err
    }

    ctx := context.Background()
    score := float64(podmetric.ReceivedAt.Unix())
    err = rdb.ZAdd(ctx, key, &redis.Z{
        Score:  score,
        Member: data,
    }).Err()
    if err != nil {
        return err
    }

    return nil
}

func getPodmetrics(rdb *redis.Client, id string, start, end time.Time) ([]Podmetric, error) {
    key := fmt.Sprintf("podmetric:%s", id)
    zRangeBy := &redis.ZRangeBy{
        Min: fmt.Sprintf("%d", start.Unix()),
        Max: fmt.Sprintf("%d", end.Unix()),
    }

    ctx := context.Background()
    results, err := rdb.ZRangeByScoreWithScores(ctx, key, zRangeBy).Result()
    if err != nil {
        return nil, err
    }

    var podmetrics []Podmetric
    for _, result := range results {
        var podmetric Podmetric
        err = json.Unmarshal([]byte(result.Member.(string)), &podmetric)
        if err != nil {
            return nil, err
        }
        podmetrics = append(podmetrics, podmetric)
    }

    return podmetrics, nil
}
```

```go
type Podmetric struct {
    ID                  string    `json:"id"`
    CPUUsage            float64   `json:"cpuUsage"`
    MemoryUsage         float64   `json:"memoryUsage"`
    DiskUsage           float64   `json:"diskUsage"`
    AverageResponseTime float64   `json:"averageResponseTime"`
    ReceivedAt          time.Time `json:"receivedAt"`
}
```

Teamwork

Sergey Lokhmatikov
@Lohmat_Sergey
Autoscaler

Roman Kuzmenko
@definitely_not_rk
LoadBalancer



Andrey Tamplon
@andreytamplon
Sidecar