Load Balancer with Service Turn-Off

API Design

Project statement

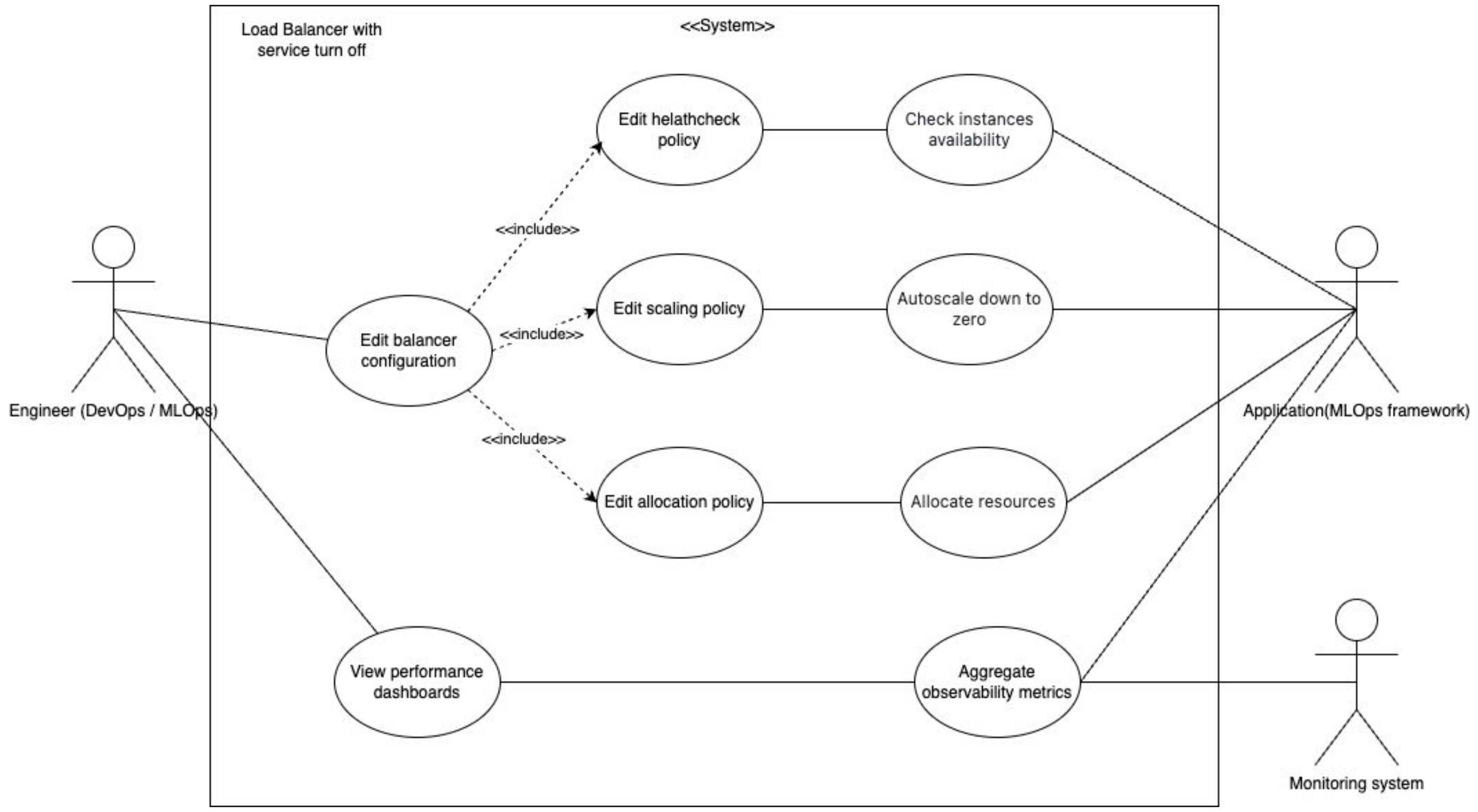
The project aims to enhance the functionality of a vanilla Kubernetes by enabling it to scale applications down to zero instances when they are not in use. This is relevant for ML applications that can be slow to start. The service will use event-driven automata to manage application scaling in response to real-time monitoring data and optimizing resource usage.

Team: Dmitry Kara, Daniil Mikulik, Ekaterina Karavayeva, Nikita Dumkin

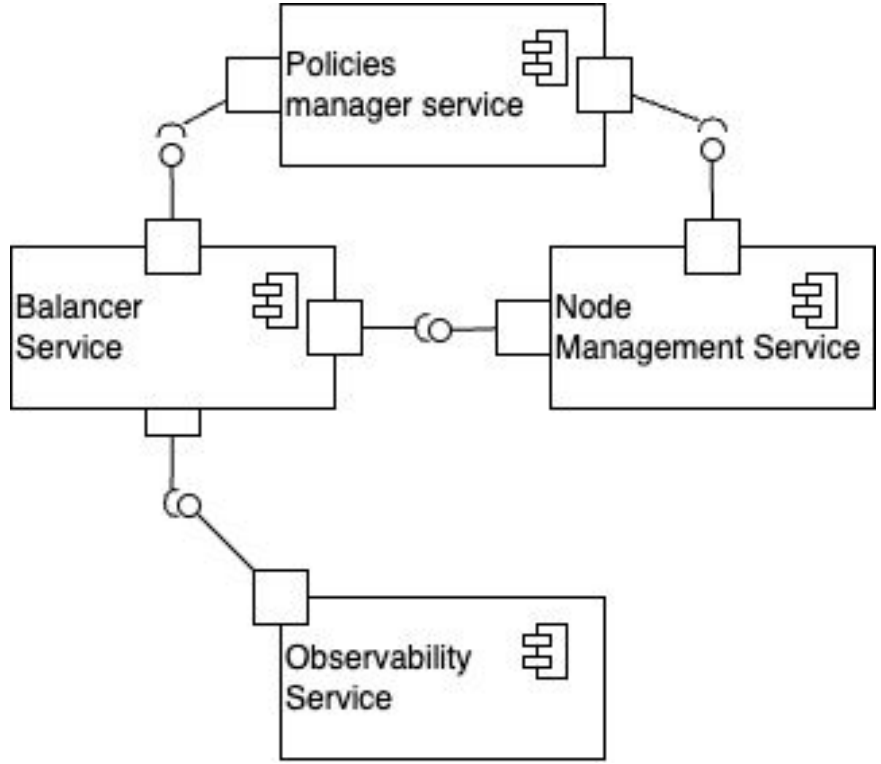
Project repo: https://github.com/dmitriykara/ads-tech-tornados-project

This report: https://docs.google.com/presentation/d/1oxI-nbPDCdRBPtet2sjLLUGJs-bWyLksfpWnbOg8wT0/edit?usp=sharing

Use case diagram


Load Balancer with service turn off
<<System>>
Edit helathcheck policy
Check instances availability
<<include>>
Edit scaling policy
Autoscale down to zero
<<include>>
Edit balancer configuration
<<include>>
Edit allocation policy
Allocate resources
View performance dashboards
Aggregate observability metrics
Engineer (DevOps / MLOps)
Application(MLOps framework)
Monitoring system

Service diagram



Open API

Using an Open API editor develop RESTful API for all microservices in the project, including data transfer objects. Store the API schema in your repository

(Note that the default example is excessive in detail)

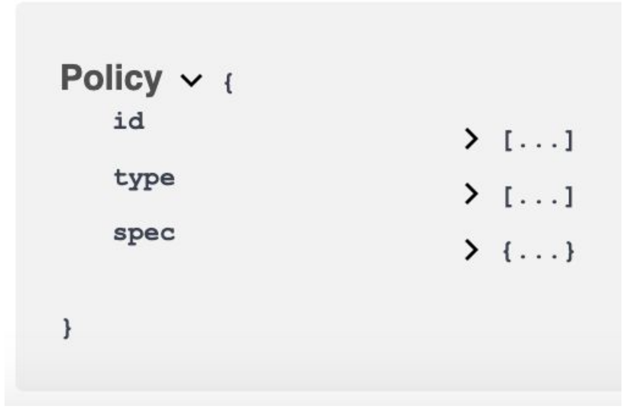
https:///editor.swagger.io

Verify step-by-step that your API supports scenarios or event flow

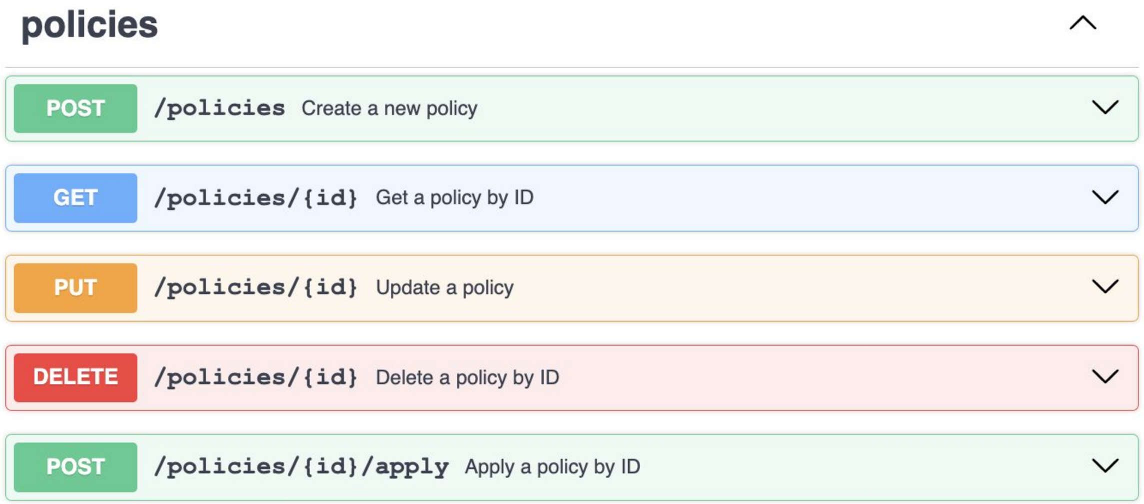
For each of the microservices show a scenario fragment/flow fragment where its API is invoked

<At least one microservice per team member>

API usage Policies Service



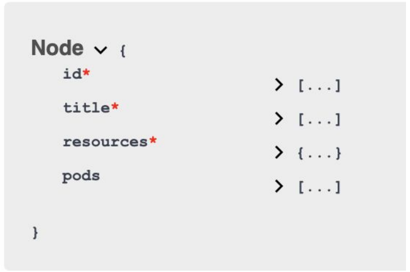
```
Policy ⌄ {
    id          > [...]
    type        > [...]
    spec        > {...}
}
```



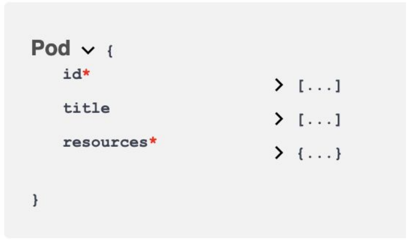
policies

POST	`/policies`	Create a new policy
GET	`/policies/{id}`	Get a policy by ID
PUT	`/policies/{id}`	Update a policy
DELETE	`/policies/{id}`	Delete a policy by ID
POST	`/policies/{id}/apply`	Apply a policy by ID

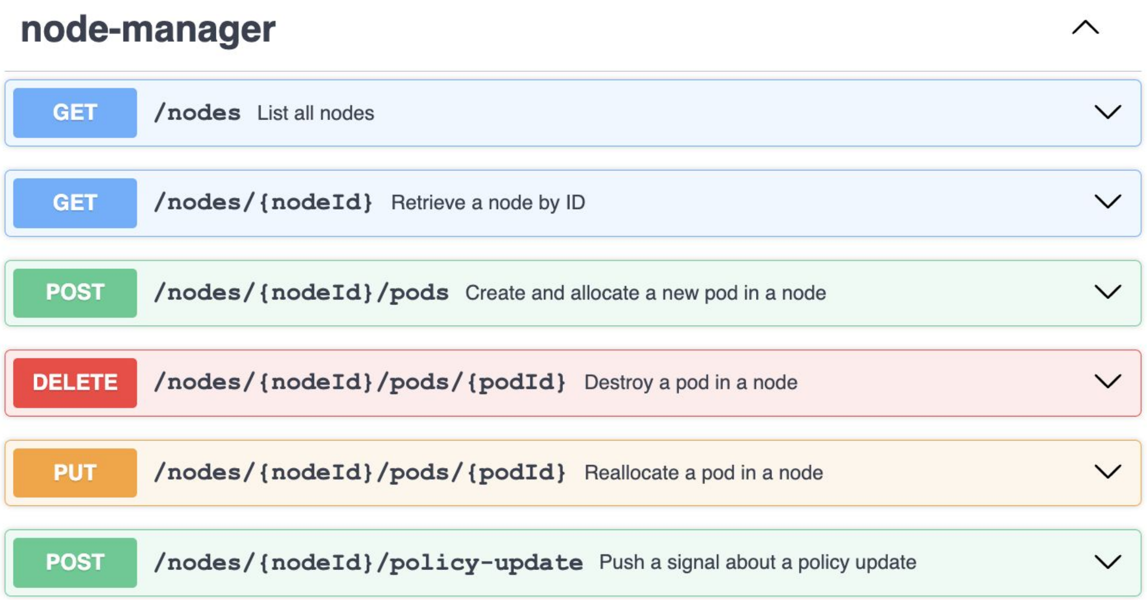
API usage Node Manager Service



```
Node ⌄ {
    id*                              > [...]
    title*                           > [...]
    resources*                       > {...}
    pods                             > [...]
}
```



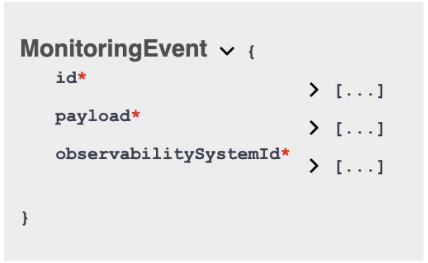
```
Pod ⌄ {
    id*                              > [...]
    title                            > [...]
    resources*                       > {...}
}
```



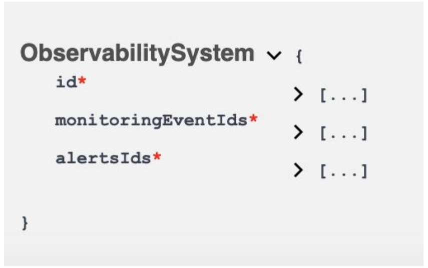
node-manager

GET	/nodes	List all nodes
GET	/nodes/{nodeId}	Retrieve a node by ID
POST	/nodes/{nodeId}/pods	Create and allocate a new pod in a node
DELETE	/nodes/{nodeId}/pods/{podId}	Destroy a pod in a node
PUT	/nodes/{nodeId}/pods/{podId}	Reallocate a pod in a node
POST	/nodes/{nodeId}/policy-update	Push a signal about a policy update

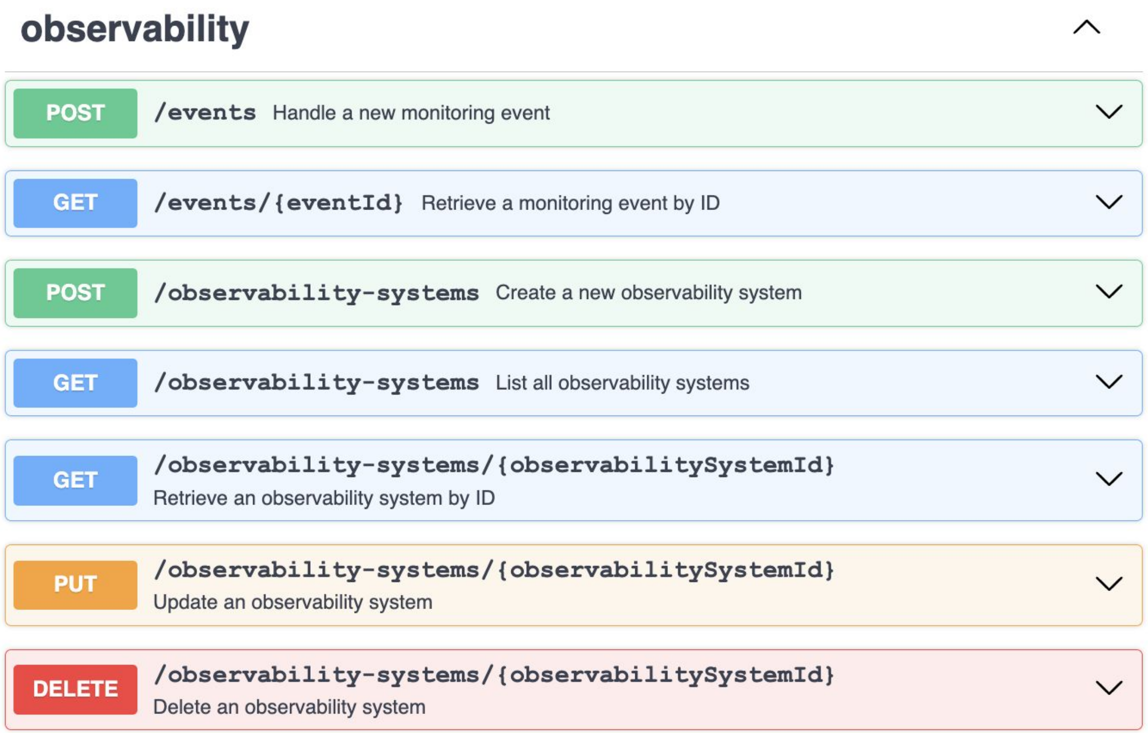
API usage Monitoring Service



```
MonitoringEvent ⌄ {
   id*                         > [...]
   payload*                    > [...]
   observabilitySystemId*      > [...]
}
```



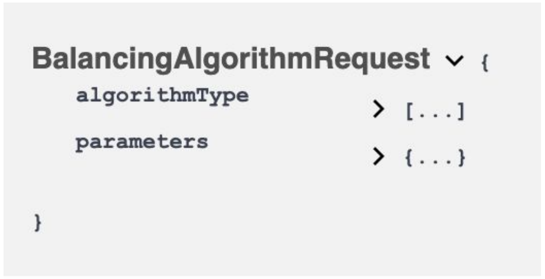
```
ObservabilitySystem ⌄ {
   id*                         > [...]
   monitoringEventIds*         > [...]
   alertsIds*                  > [...]
}
```



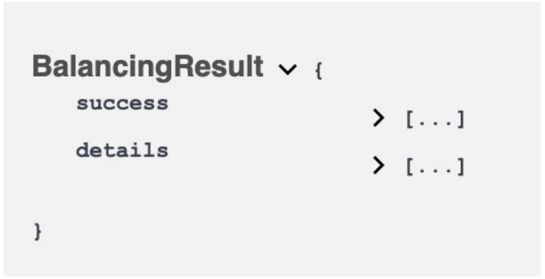
observability

POST	`/events`	Handle a new monitoring event
GET	`/events/{eventId}`	Retrieve a monitoring event by ID
POST	`/observability-systems`	Create a new observability system
GET	`/observability-systems`	List all observability systems
GET	`/observability-systems/{observabilitySystemId}`	Retrieve an observability system by ID
PUT	`/observability-systems/{observabilitySystemId}`	Update an observability system
DELETE	`/observability-systems/{observabilitySystemId}`	Delete an observability system

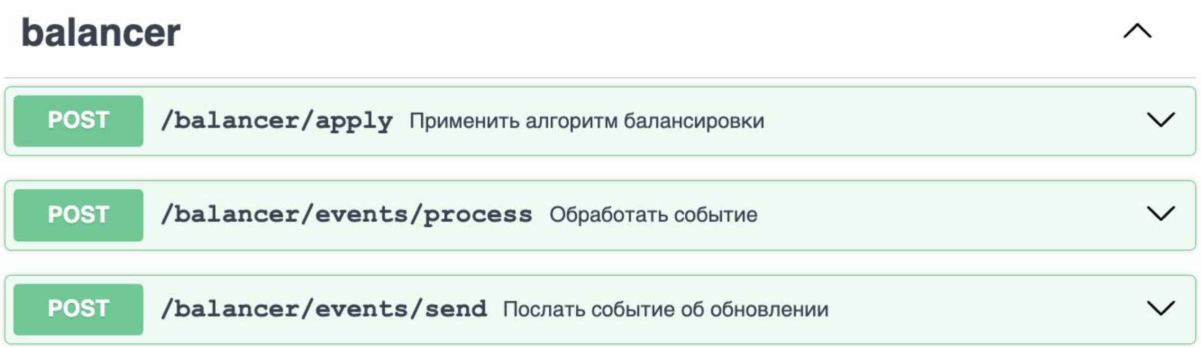
API usage Balancer Service



```
BalancingAlgorithmRequest ⌄ {
    algorithmType        > [...]
    parameters           > {...}
}
```



```
BalancingResult ⌄ {
    success              > [...]
    details              > [...]
}
```



balancer

POST /balancer/apply Применить алгоритм балансировки

POST /balancer/events/process Обработать событие

POST /balancer/events/send Послать событие об обновлении

Solution stack (prepare)

Find an example implementation of a microservices application in the programming language chosen.
Specify one value for each option below

Implementation

- OpenAPI
- Python, FastAPI
- RestAPI/JSON

Asynchronous interactions (optional)

- Celery

Testing tools

- Pytest

Operations

- Python3
- GitHUB CI/CD
- Docker Compose
- Prometheus

Some references

https://github.com/mfornos/awesome-microservices

https://awesomeopensource.com/projects/microservices-architecture

https://www.redhat.com/en/blog/comparing-openapi-grpc

https://cloud.google.com/apis/design/resources