# **Use cases**

### **1. Request Routing**

* **Actors:** API consumers, DevOps Engineers
* **Preconditions:** API endpoint is defined, and K8s services are running.
* **Main Success Scenario:**
  1. API consumer sends a request to the gateway.
  2. The reverse proxy identifies the target service in the Kubernetes cluster.
  3. The proxy forwards the request to the appropriate K8s service.
  4. The service processes the request and sends a response back to the gateway.
  5. The gateway returns the response to the API consumer.
* **Postconditions:** The request is routed correctly, and a response is delivered.
* **Failure Conditions:** Target service not found, network issues.

#### Plantext UML:

@startuml

actor "API Consumer" as consumer

actor "DevOps Engineer" as devops

usecase "Send Request to Gateway" as uc1

usecase "Forward Request to K8s Service" as uc2

usecase "Process Request and Send Response" as uc3

usecase "Return Response to Consumer" as uc4

consumer --> uc1

uc1 --> uc2

uc2 --> uc3

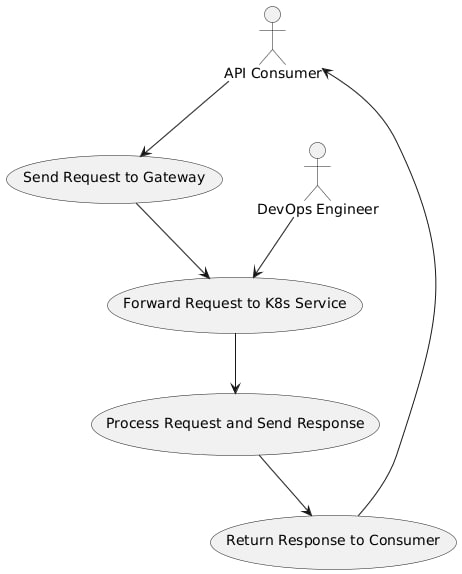
uc3 --> uc4

uc4 --> consumer

devops --> uc2

@enduml

#### Diagram:



### **2. Load Balancing**

* **Actors:** API consumers, DevOps Engineers
* **Preconditions:** Multiple instances of a service are running in the cluster.
* **Main Success Scenario:**
  1. API consumer sends a request.
  2. The gateway distributes the request across available service instances based on predefined weight (Round-Robin).
  3. The selected instance processes the request and sends the response.
  4. The response is delivered to the API consumer.
* **Postconditions:** Load is balanced across services, response is received.
* **Failure Conditions:** No healthy instances available, misconfiguration in load balancing rules.

#### Plantext UML:

@startuml

actor "API Consumer" as consumer

actor "DevOps Engineer" as devops

usecase "Send Request to Gateway" as uc1

usecase "Distribute Request Across Service Instances" as uc2

usecase "Process Request on Selected Instance" as uc3

usecase "Return Response to Consumer" as uc4

consumer --> uc1

uc1 --> uc2

uc2 --> uc3

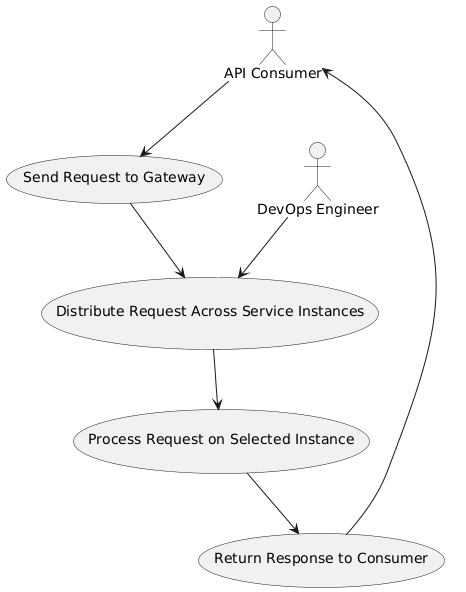
uc3 --> uc4

uc4 --> consumer

devops --> uc2

@enduml

#### Diagram:



### **3. Audit and Logging**

* **Actors:** Security Specialists, DevOps Engineers
* **Preconditions:** Gateway is configured to log requests/responses.
* **Main Success Scenario:**
  1. A request enters the system.
  2. The gateway logs relevant information (e.g., request details, response time, errors).
  3. Logs are pushed to an external system (e.g., ELK stack) for further analysis.
* **Postconditions:** Logs are stored securely for audit purposes.
* **Failure Conditions:** Logging system not accessible, log data corrupted.

#### Plantext UML:

@startuml

actor "DevOps Engineer" as devops

actor "Security Specialist" as security

usecase "Log Request and Response" as uc1

usecase "Push Logs to External System" as uc2

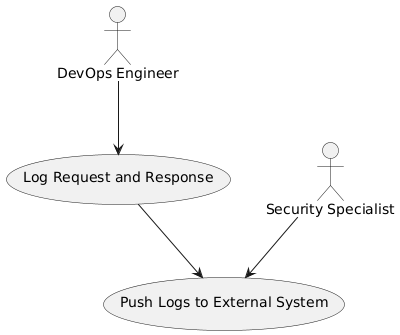
devops --> uc1

security --> uc2

uc1 --> uc2

@enduml

#### Diagram:



### **4. SSO (Single Sign-On) and Authorization**

* **Actors:** API consumers, Security Specialists
* **Preconditions:** SSO system is integrated with the gateway, user credentials are stored.
* **Main Success Scenario:**
  1. API consumer attempts to access a protected resource.
  2. The gateway checks the authorization token (JWT/OAuth).
  3. If valid, the request is processed and forwarded.
  4. If invalid, the user is redirected to the SSO login page.
  5. Upon successful login, the session is stored for future requests.
* **Postconditions:** User is authenticated and authorized, session maintained.
* **Failure Conditions:** Token expired, unauthorized access.

#### Plantext UML:

@startuml

actor "API Consumer" as consumer

actor "Security Specialist" as security

usecase "Check Authorization Token" as uc1

usecase "Redirect to SSO Login" as uc2

usecase "Forward Authorized Request" as uc3

consumer --> uc1

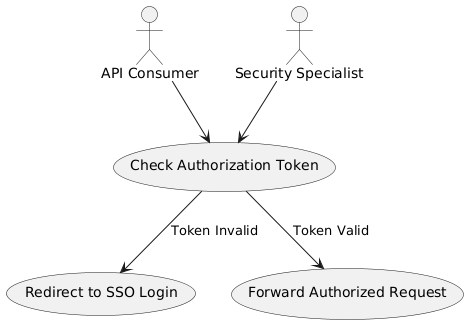
uc1 --> uc2 : Token Invalid

uc1 --> uc3 : Token Valid

security --> uc1

@enduml

#### Diagram:



### **5. Request Validation**

* **Actors:** API consumers, DevOps Engineers
* **Preconditions:** OpenAPI schema is generated and integrated.
* **Main Success Scenario:**
  1. API consumer sends a request.
  2. The gateway validates the request against the OpenAPI schema.
  3. If valid, the request is forwarded to the service.
  4. If invalid, the gateway returns an error response.
* **Postconditions:** Only valid requests reach the services.
* **Failure Conditions:** Request does not conform to the schema.

#### Plantext UML:

@startuml

actor "API Consumer" as consumer

actor "DevOps Engineer" as devops

usecase "Send Request to Gateway" as uc1

usecase "Validate Request Against Schema" as uc2

usecase "Forward Valid Request" as uc3

usecase "Return Error for Invalid Request" as uc4

consumer --> uc1

uc1 --> uc2

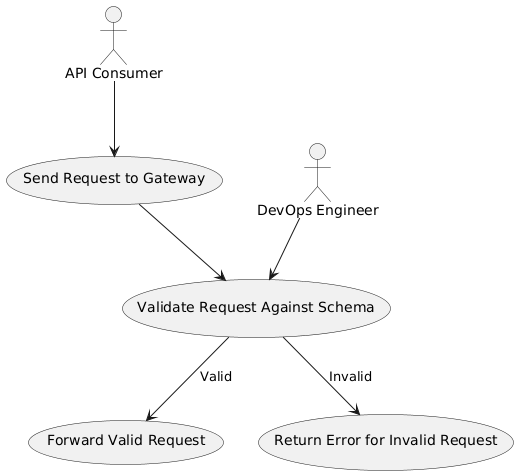
uc2 --> uc3 : Valid

uc2 --> uc4 : Invalid

devops --> uc2

@enduml

#### Diagram:



### **6. Response Caching**

* **Actors:** API consumers, DevOps Engineers
* **Preconditions:** Caching is enabled for certain responses.
* **Main Success Scenario:**
  1. API consumer sends a request.
  2. The gateway checks if the response is cached.
  3. If cached, the gateway serves the cached response.
  4. If not cached, the request is processed, and the response is cached for future requests.
* **Postconditions:** Response time is reduced for cached requests.
* **Failure Conditions:** Cache miss, outdated cached data.

#### Plantext UML:

@startuml

actor "API Consumer" as consumer

actor "DevOps Engineer" as devops

usecase "Send Request to Gateway" as uc1

usecase "Check if Response is Cached" as uc2

usecase "Serve Cached Response" as uc3

usecase "Process Request and Cache Response" as uc4

consumer --> uc1

uc1 --> uc2

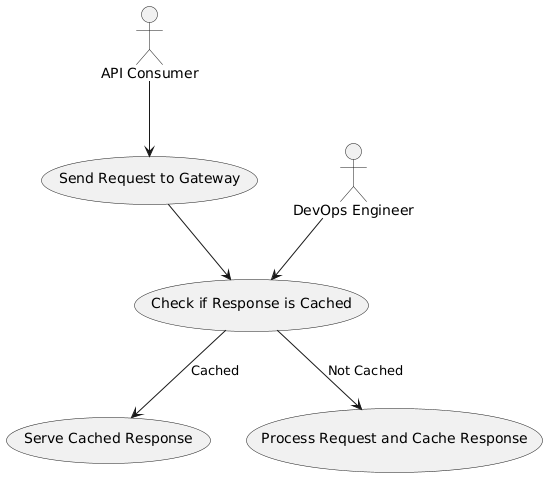
uc2 --> uc3 : Cached

uc2 --> uc4 : Not Cached

devops --> uc2

@enduml

#### Diagram:



### **7. Modular Deployment of Models**

* **Actors:** ML Engineers, DevOps Engineers
* **Preconditions:** ML models are containerized and available in the repository.
* **Main Success Scenario:**
  1. ML engineer uploads a model to the repository.
  2. The gateway orchestrates the deployment of the model into the K8s cluster.
  3. The model is containerized and available for requests.
* **Postconditions:** The model is deployed and accessible.
* **Failure Conditions:** Model not properly containerized, deployment failure.

#### Plantext UML:

@startuml

actor "ML Engineer" as mlengineer

actor "DevOps Engineer" as devops

usecase "Upload Model to Repository" as uc1

usecase "Deploy Model into K8s Cluster" as uc2

usecase "Make Model Available for Requests" as uc3

mlengineer --> uc1

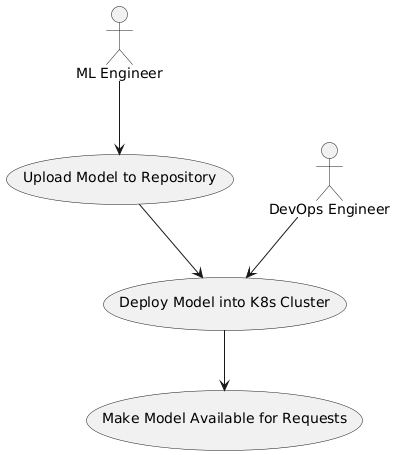
uc1 --> uc2

uc2 --> uc3

devops --> uc2

@enduml

#### Diagram:



### **8. Containerization**

* **Actors:** ML Engineers, DevOps Engineers
* **Preconditions:** Model code is ready for deployment.
* **Main Success Scenario:**
  1. ML engineer pushes the model to the repository.
  2. The gateway triggers the containerization process using Docker.
  3. The model is packaged into a Docker container and deployed.
* **Postconditions:** Model is successfully containerized and running in the K8s cluster.
* **Failure Conditions:** Containerization fails due to code errors or missing dependencies.

#### Plantext UML:

@startuml

actor "ML Engineer" as mlengineer

actor "DevOps Engineer" as devops

usecase "Push Model to Repository" as uc1

usecase "Containerize Model with Docker" as uc2

usecase "Deploy Container to K8s Cluster" as uc3

mlengineer --> uc1

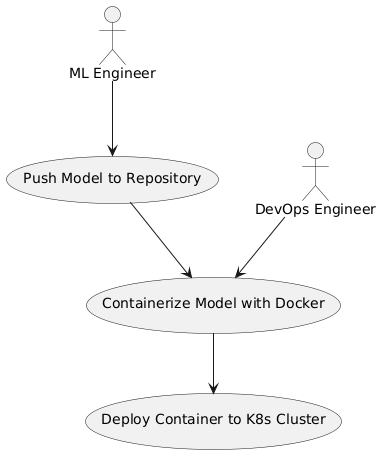
uc1 --> uc2

uc2 --> uc3

devops --> uc2

@enduml

#### Diagram:



### **9. Service Deployment**

* **Actors:** ML Engineers, DevOps Engineers
* **Preconditions:** Model version is ready, CRD is configured.
* **Main Success Scenario:**
  1. ML engineer selects a model version from the repository.
  2. The gateway automates the deployment into the cluster.
  3. Kubernetes services, deployments, and pods are created automatically.
* **Postconditions:** The model is deployed and available through an endpoint.
* **Failure Conditions:** Deployment fails due to configuration issues or resource limits.

#### Plantext UML:

@startuml

actor "ML Engineer" as mlengineer

actor "DevOps Engineer" as devops

usecase "Select Model Version for Deployment" as uc1

usecase "Automate Deployment to K8s Cluster" as uc2

usecase "Create Services and Pods" as uc3

mlengineer --> uc1

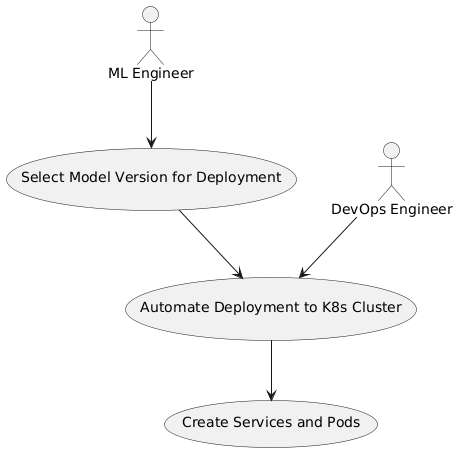
uc1 --> uc2

uc2 --> uc3

devops --> uc2

@enduml

#### Diagram:



### **10. Model Auto-Documentation**

* **Actors:** API consumers, ML Engineers
* **Preconditions:** OpenAPI schema generation is configured.
* **Main Success Scenario:**
  1. ML engineer deploys the model via the gateway.
  2. The gateway reads the model’s source code or CRD.
  3. An OpenAPI specification is auto-generated for the model.
  4. The API documentation is available for API consumers.
* **Postconditions:** API consumers can view documentation to interact with the model.
* **Failure Conditions:** Schema generation fails, documentation not accessible.

#### Plantext UML:

@startuml

actor "API Consumer" as consumer

actor "ML Engineer" as mlengineer

usecase "Deploy Model via Gateway" as uc1

usecase "Generate OpenAPI Specification" as uc2

usecase "Provide API Documentation" as uc3

mlengineer --> uc1

uc1 --> uc2

uc2 --> uc3

consumer --> uc3

@enduml

Diagram:

