Laboratoire #1
Groupe: 02
Équipe: 02
Membres de l'équipe:

- **Alaaeddine Taif**
- **Adam Jelidi**
- **Martin Belser**

Participation

Nom de l'étudiant	Tâches réalisées	Facteur de participation
Adam Jelidi	**Ping/Echo, Diagrammes, Base de données, Dockerfile, Rapport de lab**	**0.33**
Martin Belser	**Ping/Echo, Diagrammes, Base de données, Dockerfile, Rapport de lab**	**0.33**
Alaaeddine Taif	**Ping/Echo, Diagrammes, Base de données, Dockerfile, Rapport de lab**	**0.33**

Introduction

Ce laboratoire a pour but de prendre en main un système de microservices s'exécutant dans Docker. Les services principaux (NodeController, STM, RouteTimeProvider et TripComparator) communiquent entre eux et interagissent avec des API externes (TomTom et STM réelle). Nous devions :

1. Configurer un fichier Dockerfile manquant pour RouteTimeProvider.
2. Ajouter une base de données (PostgreSQL) dans STM pour éviter l'in-memory (gourmand en RAM).
3. Mettre en place un mécanisme de Ping/Echo entre TripComparator et RouteTimeProvider pour détecter la destruction et la relance du conteneur.
4. Documenter l'architecture via des diagrammes (séquence, contextes), et répondre à des questions d'architecture et de communication.

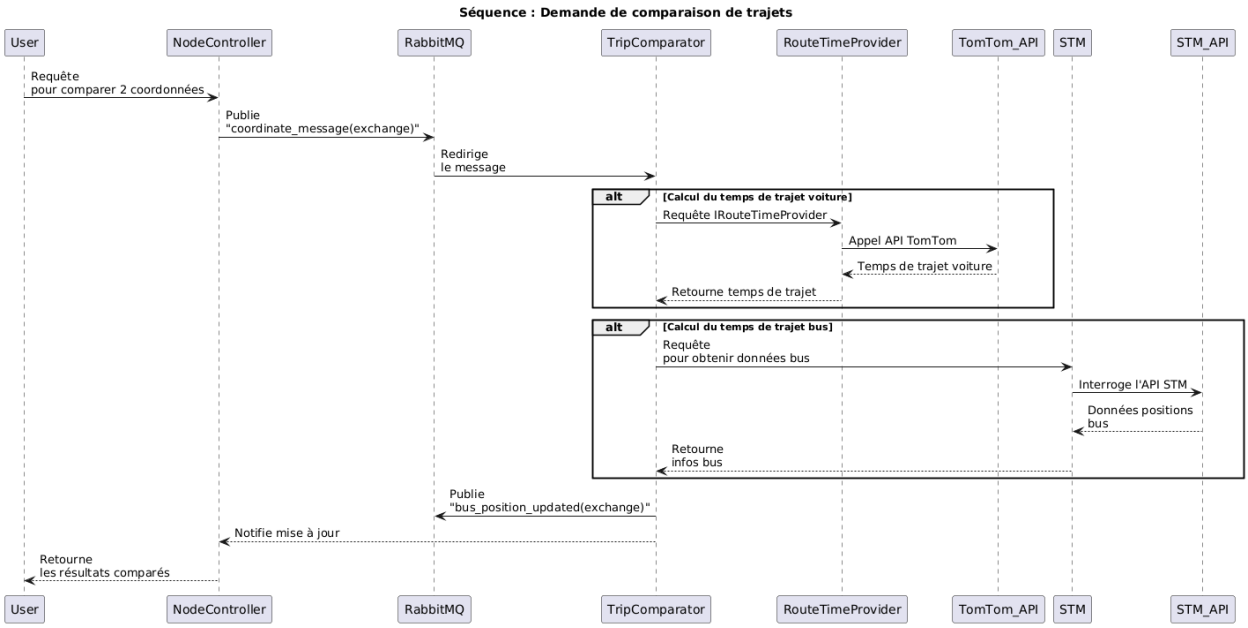
Le présent rapport décrit notre implémentation, les diagrammes illustrant le flot principal, ainsi que les réponses aux questions d'architecture liées aux attributs de qualité et aux mécanismes d'échange de requêtes.

Diagramme de séquence de haut niveau

Explication générale du fonctionnement du système

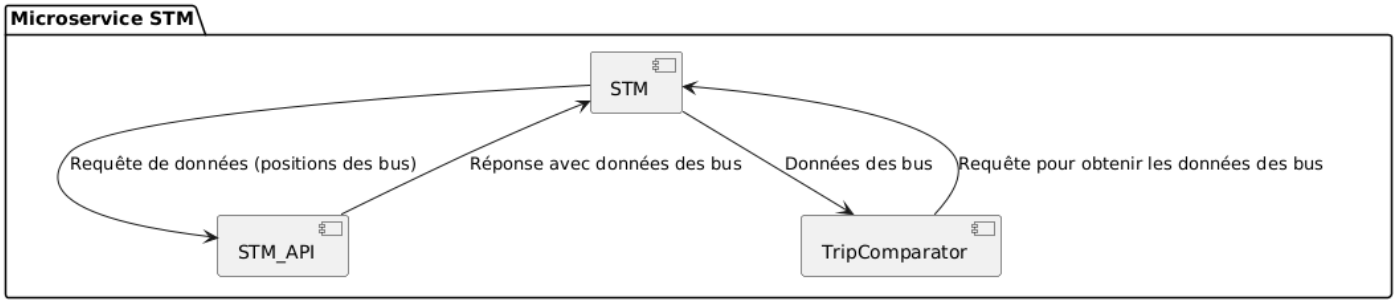
- **NodeController :**
  - Orchestration Docker, redéploiement automatique des conteneurs (répliques), gestion du chaos engineering.
  - Fournit une API de service discovery et d'équilibrage de charge (RabbitMQ).
  - Relaye aussi les requêtes depuis le Dashboard (coordonnées) vers TripComparator.
- **STM :**
  - Adapte et consomme l'API réelle de la STM pour récupérer les positions de bus en temps réel.
  - Charge également des données statiques GTFS (horaires, arrêts) – en base de données.
  - Retourne au TripComparator les informations sur les bus à suivre.
- **RouteTimeProvider :**
  - Reçoit les requêtes de TripComparator pour calculer le temps de trajet en voiture.
  - Interroge l'API TomTom en fournissant deux coordonnées de départ/arrivée.
  - Retourne le temps de trajet (en secondes).
- **TripComparator :**
  - Reçoit la requête de comparaison (voiture vs. bus).
  - Appelle RouteTimeProvider (voiture) et STM (bus) pour obtenir les données nécessaires.
  - Rassemble les résultats et renvoie la comparaison vers le NodeController (puis au Dashboard).

Diagramme de séquence

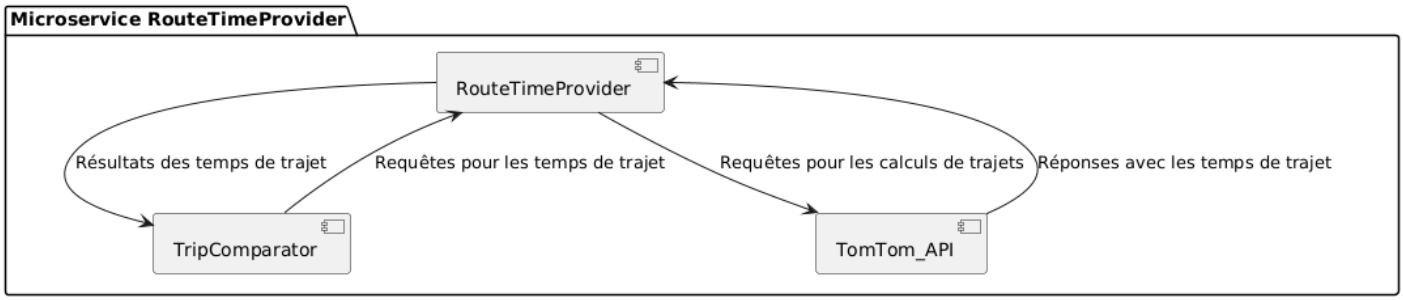


Diagrammes de contexte de haut niveau

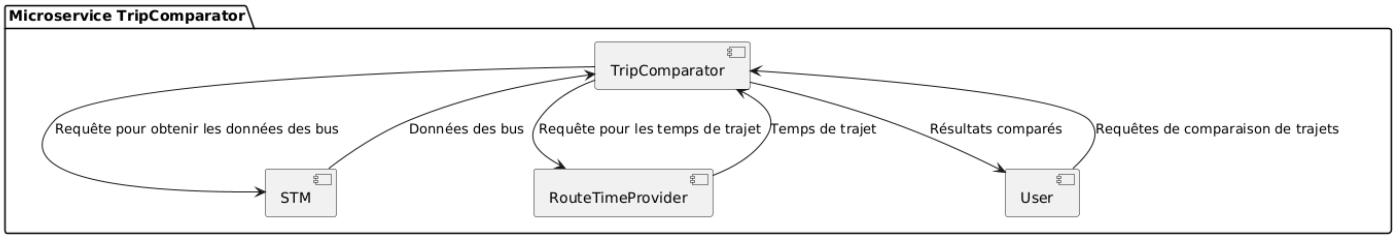
Microservice STM



Microservice RouteTimeProvider



Microservice TripComparator



Questions générales

1. **L'ajout de la base de données est une amélioration au système. Quel attribut de qualité est amélioré par l'ajout d'une base de données ? Expliquez votre réponse.**
   Attribut amélioré : **disponibilité** (**fiabilité, persistence**).
   L'ajout d'une base de données permet de conserver les données critiques entre les redémarrages ou les interruptions des services. Cela améliore la **fiabilité** du système en garantissant que les données ne dépendent pas uniquement de la mémoire vive, réduisant ainsi les pertes potentielles en cas de panne.

   - Avant, STM chargeait toutes ses données en mémoire ; si le conteneur tombait, on perdait tout.

   - Avec une base de données, les données **persistent** même si le microservice est redémarré. On réduit aussi le risque de crash par manque de RAM.

   - Conformément aux attributs de qualité vus en classe (p.ex. *Availability*, *Reliability*, *Performance*), avoir une BD résiliente prévient les **indisponibilités** liées à la surcharge mémoire ou la perte de données.

2. **Lorsqu'une requête de comparaison du temps de trajet faite sur le Dashboard est envoyée au NodeController, différentes requêtes sont échangées sur le réseau. Dans la minute suivant l'envoi de la requête initiale, combien de requêtes sont échangées entre les microservices **NodeController**, **TripComparator**, **RouteTimeProvider** et **STM** et avec les API externes de la STM et de TomTom ? Comptez chaque requête échangée. Ainsi, si une requête est envoyée 10 fois, elle comptera comme 10 requêtes. Décrivez également rapidement les différentes requêtes échangées.**

   Conteneur: STM

- **Type :** Application.Queries.GetEarliestBus.GetEarliestBusHandler[0]
    - **Nombre**: 270
    - **Description** : Ces requêtes concernent la recherche du prochain bus disponible selon des paramètres tels que la localisation de départ, la destination et l'heure. STM utilise probablement un service interne pour interroger sa base de données ou des API externes pour planifier les trajets.

  Conteneur: TripComparator
- **Type:** Controllers.Controllers.TripComparatorMqController[0]
    - **Nombre:** 1
    - **Description** : Cette requête est déclenchée pour comparer les temps de trajet entre une paire de coordonnées de départ et d'arrivée en utilisant deux modes de transport : **bus** et **voiture**.

  Conteneur: NodeController
- **Type: Infrastructure.ApiClients.IngressClient[0]**
    - **Nombre**: 11
    - **Description** : Ces requêtes correspondent à des appels aux API externes pour obtenir des données en temps réel, incluant :
      - **TomTom** : Récupération des temps de trajet, conditions de trafic ou itinéraires alternatifs.
      - **STM** : Données sur les horaires et trajets des transports en commun.

      Ces requêtes sont essentielles pour fournir les informations nécessaires aux comparaisons de temps de trajet effectuées par d'autres services.

- **Type: Controllers.Events.ExperimentMqController[0]**
    - **Nombre**: 25
    - **Description** : Ces requêtes sont des messages échangés via un système de communication asynchrone (ex. RabbitMQ), incluant :
        - La coordination entre le **NodeController** et d'autres microservices.
        - Le déclenchement des actions dans les microservices dépendants comme **TripComparator** ou **RouteTimeProvider**.

  Conteneur: RouteTimeProvider

- **Type: RouteTimeProvider.Controllers.RouteTimeController[0]**
  - **Nombre**: 1
  - **Description** : Cette requête représente un appel pour récupérer le temps de trajet en voiture entre deux coordonnées spécifiques.

Analyse globale des échanges
Nombre total de requêtes :

- STM: 270
- TripComparator: 1
- NodeController: 36 (11 + 25)
- RouteTimeProvider: 1
- **Total**: 308 requêtes.

2.1 **Dans le code source du projet, nous pouvons ajuster le taux de lancement des requêtes envoyées par le microservice TripComparator vers le microservice STM. Expliquez comment nous pouvons nous y prendre pour ce faire en donnant un ou des extraits de code.**

Dans le code, on peut identifier un Timer qui déclenche la mise à jour. Le fichier HealthCheckService.cs a un Timer défini ainsi :

_timer = new Timer(ExecuteHealthCheck, null, TimeSpan.Zero, TimeSpan.FromSeconds(1));

Pour augmenter ou diminuer la fréquence, on modifie le deuxième paramètre de TimeSpan. Par exemple :

// Appeler ExecuteHealthCheck toutes les 5 secondes au lieu de 1 seconde
_timer = new Timer(ExecuteHealthCheck, null, TimeSpan.Zero, TimeSpan.FromSeconds(5));

3. **Proposez une tactique permettant d'améliorer la disponibilité de l'application lors d'une attaque des conteneurs de computation (**TripComparator**, **RouteTimeProvider**, et **STM**) lors du 2e laboratoire.**

Une tactique serait:

- Activer la réplication (via Docker Compose et NodeController) de chaque service critique.
- Activer un load balancer (Round Robin) ou un circuit breaker pour diriger le trafic vers les réplicas en bonne santé.
- Stocker les données critiques (STM) en base de données et s'assurer d'avoir des volumes persistants (pour éviter perte de données quand le conteneur meurt).
- Ainsi, si l'un des conteneurs de computation est tué, le NodeController en relancera un autre, et le système restera disponible.

Conclusion du laboratoire

Ce laboratoire nous a permis de :

- **Découvrir** la communication entre microservices Dockerisés (avec orchestration NodeController).
- **Mettre en place** une base de données pour STM, réduisant l'empreinte mémoire et améliorant la persistance des données.
- **Implémenter** un mécanisme de *Ping/Echo* pour surveiller la disponibilité de RouteTimeProvider et s'adapter dynamiquement en cas de redémarrage.
- **Documenter** l'architecture via des diagrammes de séquence et de contexte, et **analyser** les attributs de qualité liés à la disponibilité, la persistance et la résilience.

Finalement, cette expérience illustre l'importance d'une conception résiliente dans un environnement **microservices** : la **disponibilité**, la **scalabilité** et la **tolérance aux pannes** sont cruciales pour la stabilité d'un système distribué.