« Rendez-vous autour de VMS »

Intégrer les systèmes OpenVMS dans les architectures « WebServices / MicroServices »

Zoom, Mardi 15 novembre 2022

VMS generations

Bienvenue au « Rendez-vous autour de VMS »

Intégrer les systèmes OpenVMS dans les architectures « WebServices / MicroServices »

- Nouveau format autour d'un atelier interactif, de témoignages et d'expériences clients
  - Accueil – introduction — Miroslaw Szczeblewski, Benoît Maillard
  - Présentation des outils et de la méthode — Jean-François Piéronne, Rémi Jolin
  - Témoignages de mise en œuvre — Rémi Jolin, Christian Levrey
  - Atelier interactif : webservices VMS-Linux — Jean-François Piéronne
  - Questions/réponses
- Remerciements à ProVMS, VSI
- La prochaine session sera orientée « produits » (actualité fournisseurs VSI, Oracle,...)

VMSgenerations

Lab OpenVMS
WebServices / MicroServices

Rendez-vous autour de VMS
15 novembre 2022

VMSgenerations

Problématique

- Applications historiques sous OpenVMS :
  - Utilisent des données locales
  - Interface utilisateur mode VT
  - Traitements locaux
- Aujourd'hui les systèmes d'informations sont :
  - Multi-plateformes
  - Applications distribuées sur des systèmes hétérogènes
  - Interface multi-modes
    - Navigateurs
    - Smartphones
    - IOT (Internet des objets)...

Problématique (suite)

- La tradition : sous OpenVMS le développement est :
  - Interface utilisateur en mode VT
  - Parfois gestion de code avec CMS/MMS/LSE (DECset)
- Aujourd'hui :
  - Nouveaux langages (Python, JavaScript, TypeScript, Go, Rust,…)
  - Utilisation d'outils type DevOps (forges Github, Gitlab/Heptapod,…)
  - Éditeurs de sources évolués (VisualStudio Code par exemple)

- *[Le thème DevOps/forge n'est pas couvert dans cette présentation, et pourra faire l'objet d'un atelier spécifique ultérieur]*

Le sujet du jour

- **Intégrer les systèmes OpenVMS dans les architectures « WebServices / MicroServices»**
  - Par « *WebServices / MicroServices* », comprendre :
    - Multi plateformes
    - Utilisé parfois par un humain au travers d'un navigateur
    - Plus souvent c'est une application qui parle à une autre
    - Lorsqu'un service Web est utilisé, un client envoie une requête à un serveur et déclenche ainsi une action auprès de ce serveur. Le serveur renvoie ensuite une réponse au client.

Service web

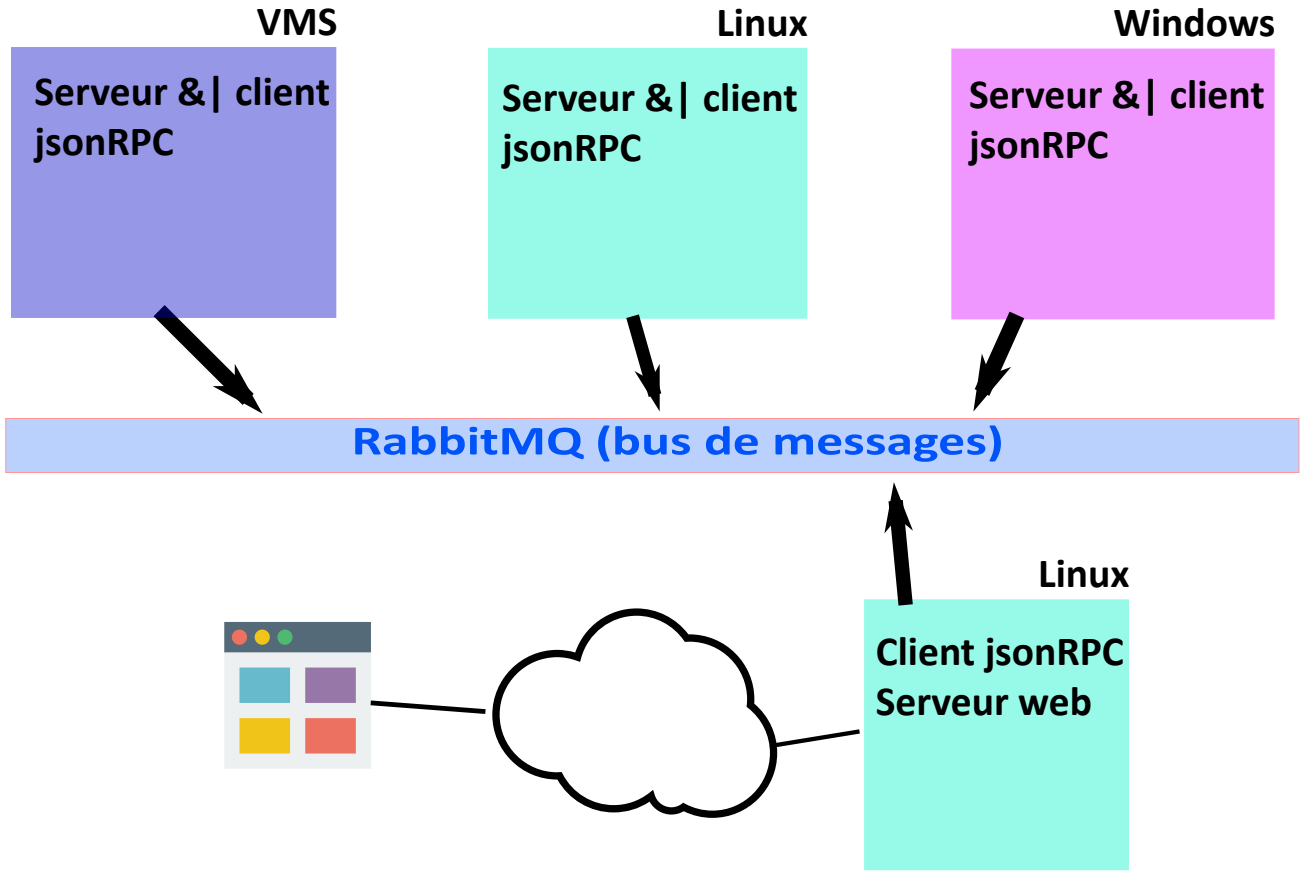
- Un service Web met à disposition un service via un réseau. Il constitue ainsi une interface permettant à deux machines (ou applications) de communiquer. Pour y parvenir, la technologie doit disposer de deux propriétés essentielles :
  - être multi plateforme : il n'est pas nécessaire que le client et le serveur aient la même configuration pour pouvoir communiquer. Le service Web leur permet de se retrouver à un même niveau.
  - être partagée : dans la plupart des cas, un service Web est à disposition de plus d'un client. Différents clients accèdent à ce service via Internet.
- Lorsqu'un service Web est utilisé, un client envoie une requête à un serveur et déclenche ainsi une action auprès de ce serveur. Le serveur renvoie ensuite une réponse au client.

Les outils

- **RabbitMQ**
  - Message broker : routeur de messages
- **FastAPI**
  - Framework HTTP
- **JSON-RPC** pour le formalisme des messages échangés

- VS Code, Heptapod pour la gestion des développements

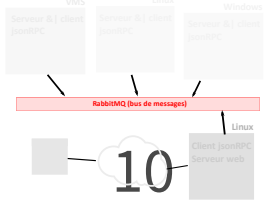
Schéma général



RabbitMQ

« *RabbitMQ is the most widely deployed open source message broker.* »

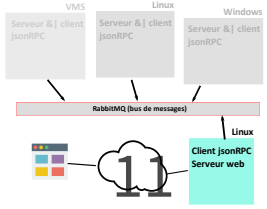
- Redondance
- Driver multi plateformes/multi langages
- Chargé de distribuer les messages dans des « queues » consommées par des serveurs



Serveur HTTP

C’est lui qui fait le lien entre le monde HTTP (web services) et les serveurs JsonRPC via RabbitMQ.

- Basé sur FastAPI (framework populaire HTTP en Python)
- Pour chaque web service, vérifie les paramètres d’entrée, transmet la requête au serveur JsonRPC via RabbitMQ et renvoie la réponse à l’appelant.



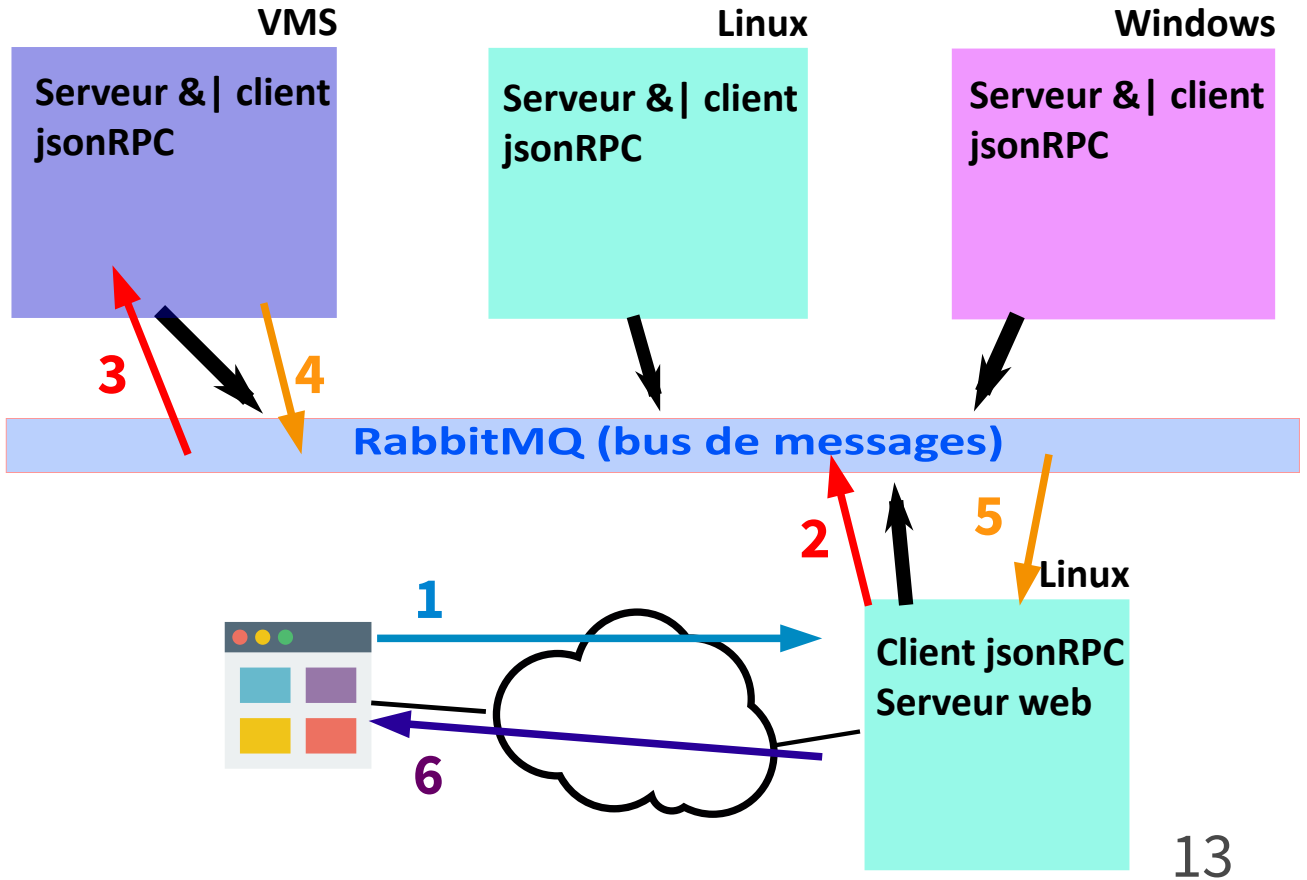
JSON-RPC

- Normalisation des messages échangés
- Support : JSON
- Exemples :
  - *--> {"jsonrpc": "2.0", "method": "subtract", "params": {"moins": 23, "valeur": 42}, "id": 3}*
  - *<-- {"jsonrpc": "2.0", "result": 19, "id": 3}*

  - *--> {"jsonrpc": "2.0", "method": "foobar", "id": "1"}*
  - *<-- {"jsonrpc": "2.0", "error": {"code": -32601, "message": "Method not found"}, "id": "1"}*

Flux de données « web service »

- Requête HTTP
- Requête JsonRPC
- Réponse JsonRPC
- Réponse HTTP



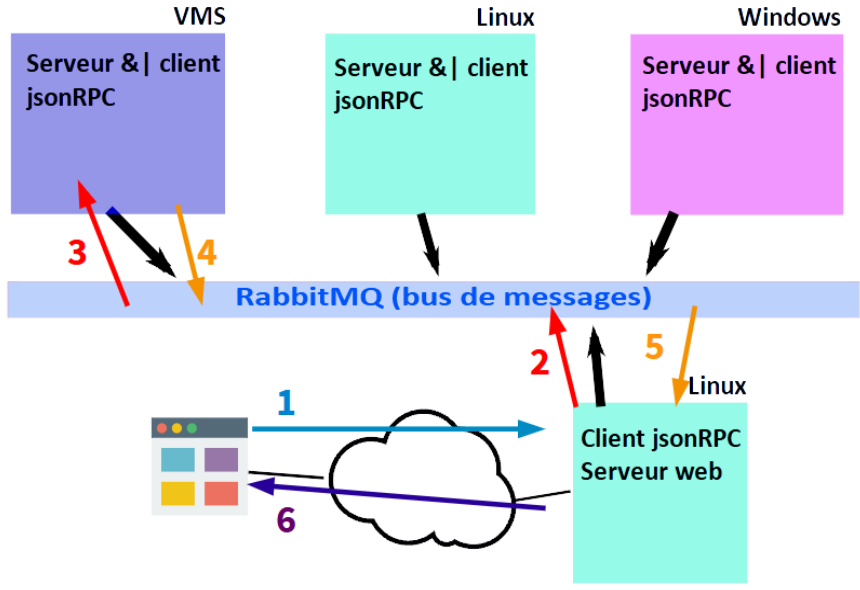
Flux de données « web service »

1 l'application envoie sa requête au serveur HTTP (GET, POST, PUT,...)

2 le serveur HTTP transforme la demande en JsonRPC et envoie le message à RabbitMQ
3 RabbitMQ délivre le message dans la file d'attente du serveur

4 Le serveur envoie sa réponse à RabbitMQ (JsonRPC)
5 RabbitMQ fait suivre cette réponse au serveur HTTP

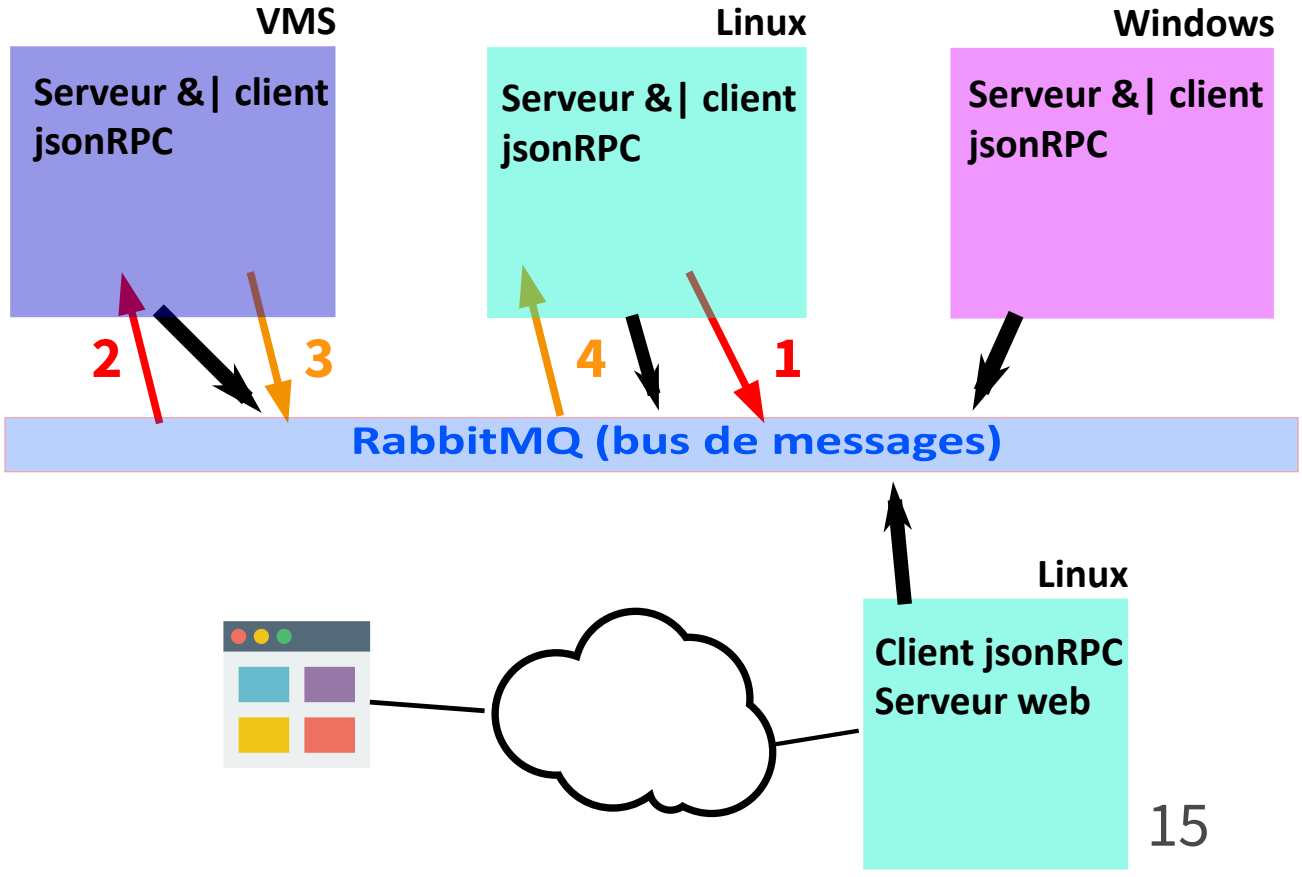
6 Le serveur HTTP fait suivre la réponse à l'application (Json)



Flux de données « JsonRPC »

Requête JsonRPC

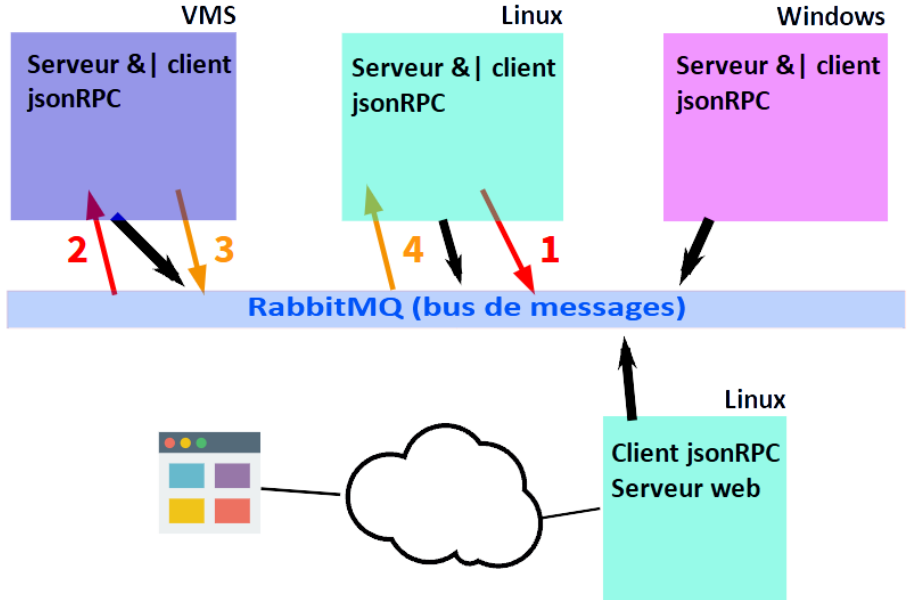
Réponse JsonRPC



Flux de données « JsonRPC »

1 le client JsonRPC (ici Linux) envoie le message à RabbitMQ
2 RabbitMQ délivre le message dans la file d'attente du serveur de traitement (ici OpenVMS)

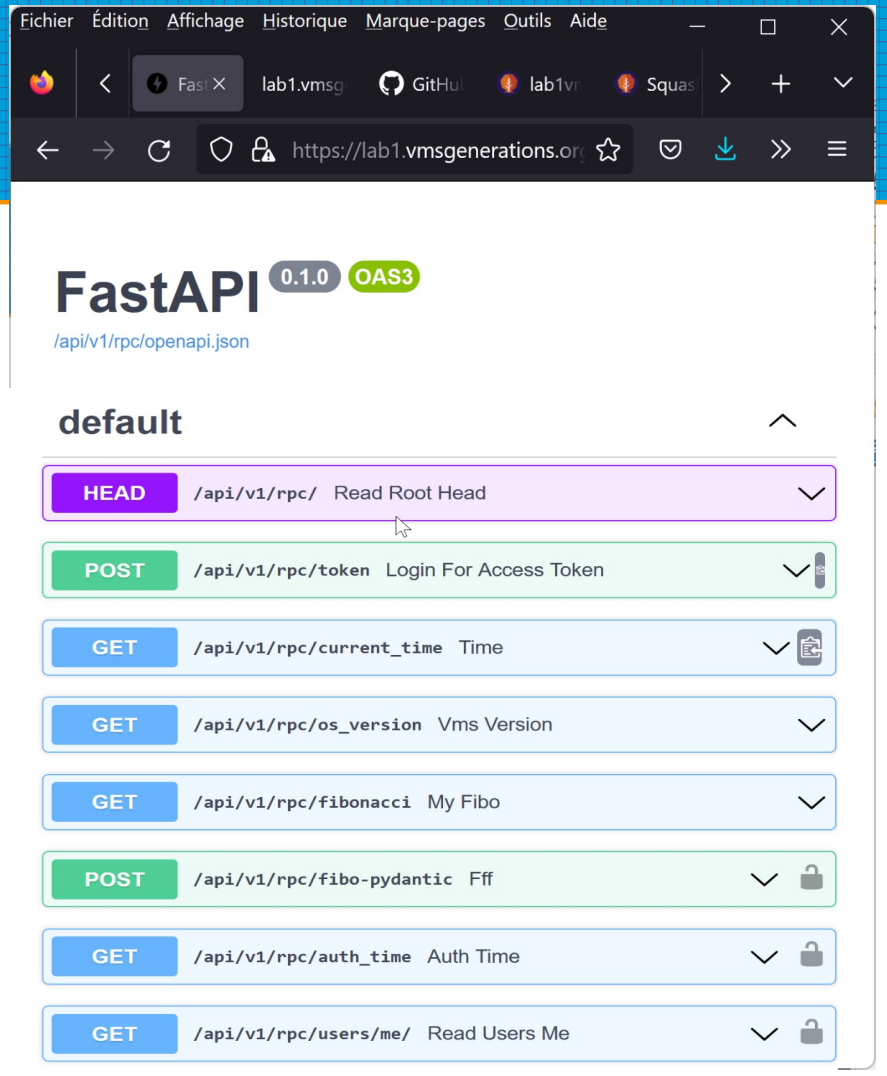
3 Le serveur de traitement envoie sa réponse à RabbitMQ (JsonRPC)
4 RabbitMQ fait suivre cette réponse au client JsonRPC



Serveur HTTP

Exemple de points d'entrée :

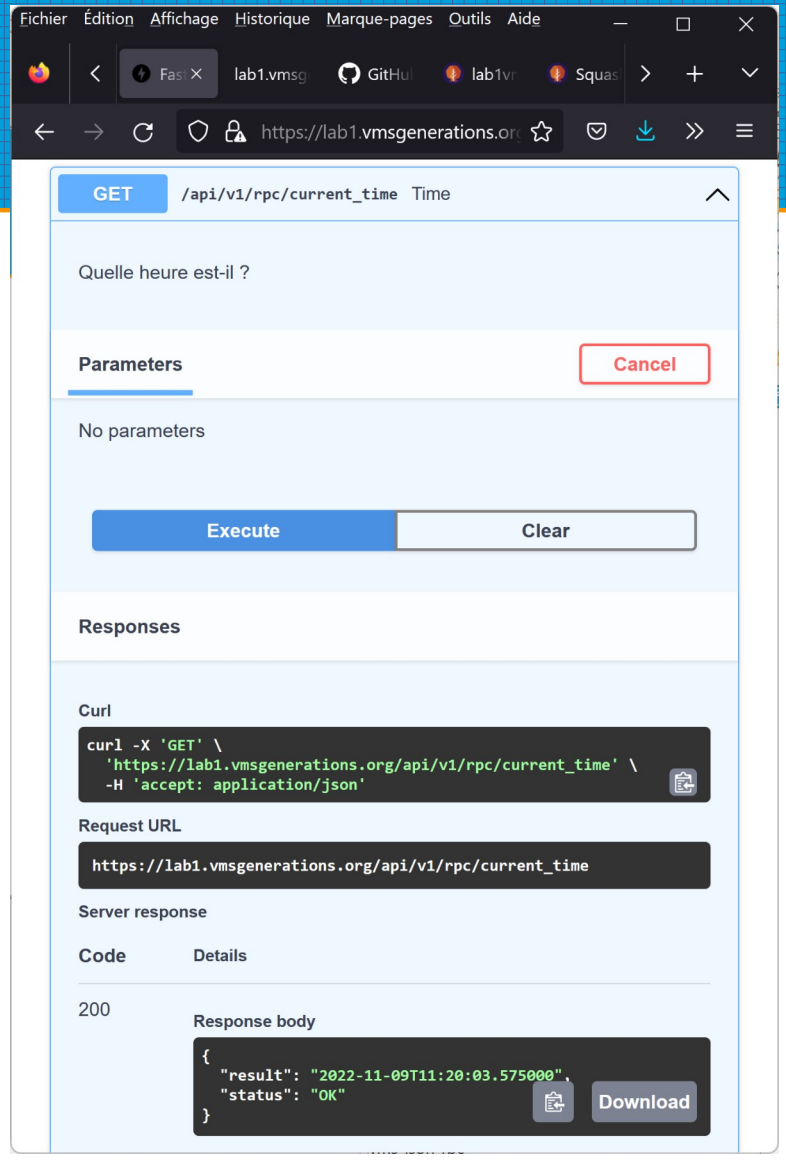
- Vocabulaire simple :
  - GET, POST, PUT, HEAD…
- Points d'entrée pour notre démo :
  - Calcul de fibonacci
  - Récupération de la date/heure du serveur
  - Version de l'OS du serveur exécutant la requête
- Authentifiés ou non
- FastAPI génère la documentation et des pages de tests



Serveur HTTP

Exemple de point d'entrée :

```
@router.get('/current_time')
@rpc_call.rpc_route('current_time')
def time():
    """Quelle heure est-il ?"""
    pass
```



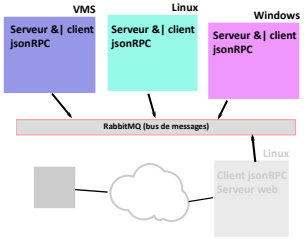
Serveur JsonRPC

C’est lui qui traite les données, exécute le code métier et renvoie un résultat

- Multi plateforme
- Multi langage

Exemple de code Python s’exécutant sur le serveur applicatif (OpenVMS ou autre)

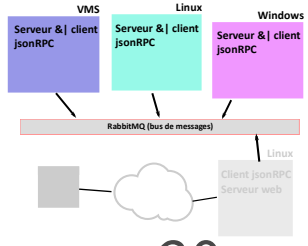
```
@method
def current_time():
    return datetime.datetime.now()
```



Serveur JsonRPC

Exemple en « C »

```
/*
  send the message
 */
status = rpc_send(exchange, &props, root, &response);
return_on_error(status, "rpc_send", -4);
if (status) {
   char *restxt = cJSON_Print(response);
   printf("\nError: %d (%s) %s\n", status,
            amqp_constant_name(status), restxt);
   cJSON_free(restxt);
} else {
   cJSON *res = cJSON_GetObjectItem(response, "result");
   if (res == NULL) {
            char *restxt = cJSON_Print(response);
            printf("\nNo result: %s\n", restxt);
            cJSON_free(restxt);
   } else if (cJSON_IsNumber(res)) {
            dres = cJSON_GetNumberValue(res);
            printf("fibonacci for %d result: %d\n", i, (int) dres);
   } else {
            char *restxt = cJSON_Print(response);
            printf("\nError returned: %s\n", restxt);
            cJSON_free(restxt);
   }
}
cJSON_Delete(response);
}
```



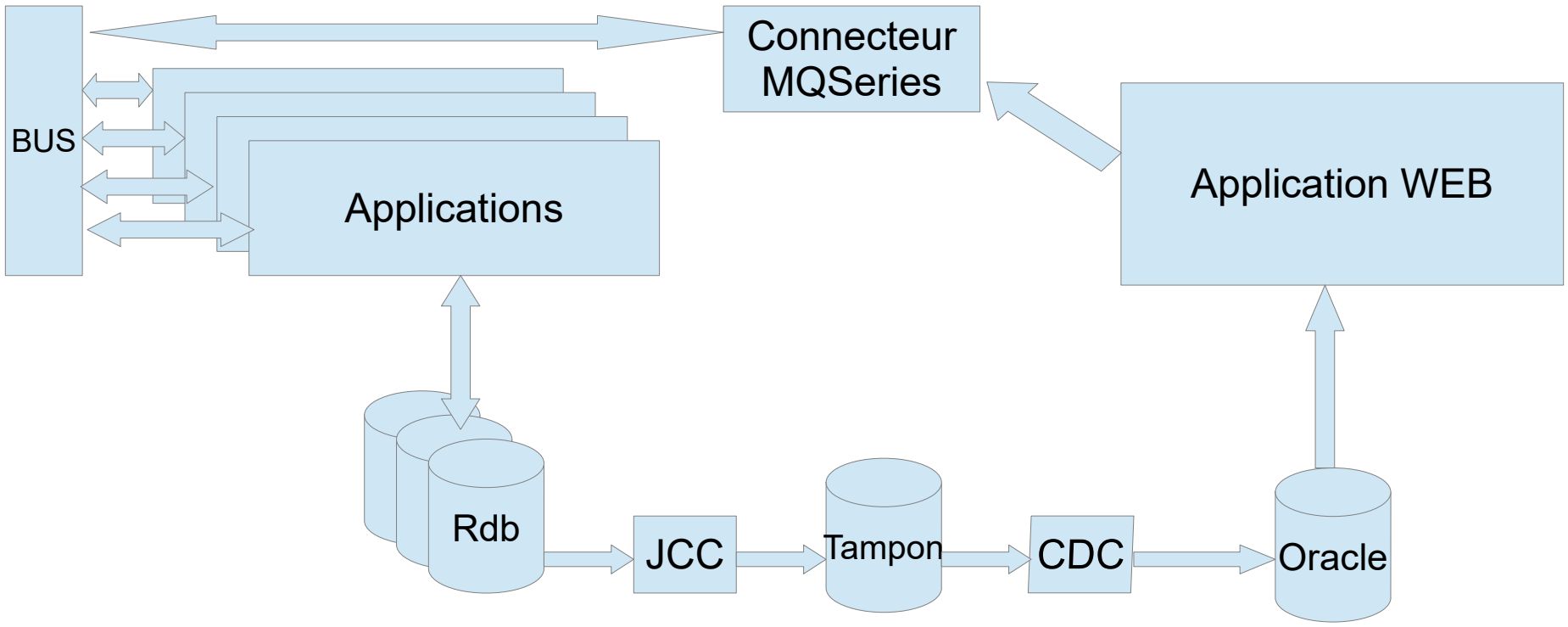
Expérience utilisateur

- Collection de plus de 90 applications
  - Principalement écrites en COBOL
  - Base de données Rdb
  - Interface en mode caractères (FMS, DECForms)
- Faciliter l'accès aux informations
  - Regrouper les informations de plusieurs applications sur une seule page
  - Agréger les informations de plusieurs applications
- Respecter la contrainte :
  - Pas de modifications des informations en base de données par des programmes hors OpenVMS

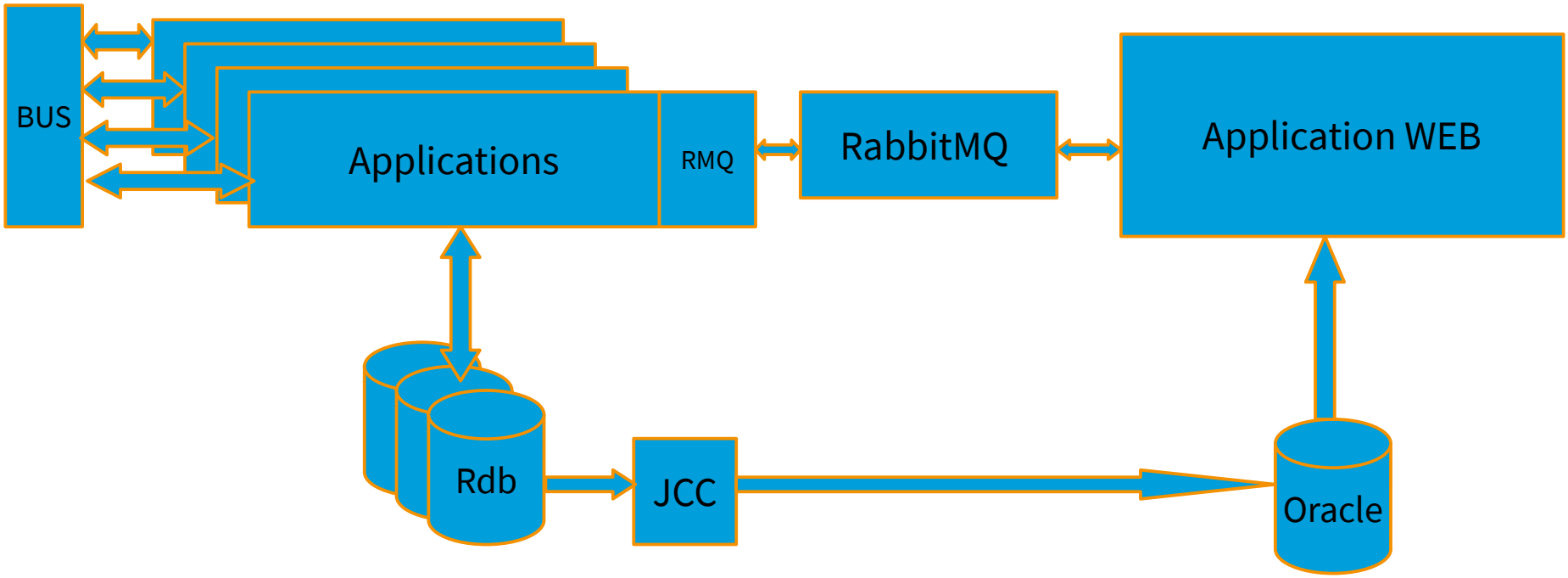
Expérience utilisateur

- Réplication en temps réel des bases de données Rdb dans une base ORACLE tampon avec JCC, une base Rdb vers un schéma de la base tampon
- Agrégation des données de la base tampon avec l'outil CDC vers la base ORACLE accessible par l'interface WEB
- Mise à jour de données depuis l'interface WEB par envoi de messages via MQSeries vers un programme spécifique sur le serveur OpenVMS, celui-ci effectuant la mise à jour de la base de données Rdb, l'utilisateur n'est pas informé immédiatement du succès ou de l'échec de sa demande

Expérience utilisateur



Expérience utilisateur



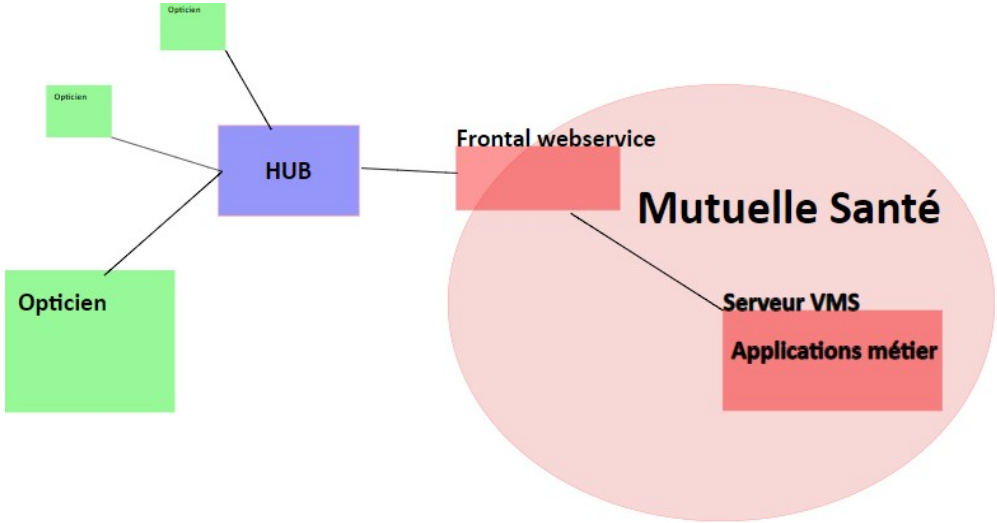
Expérience utilisateur

- Simplification du processus de réplication
  - Agrégation des données par JCC (deux bases sources vers une base cible) donc suppression de CDC et de la base tampon
- Les mises à jour de données depuis l'interface WEB par envoi de messages JSON via RabbitMQ vers un programme récepteur en mode RPC, la réponse informe immédiatement du résultat de sa demande
- Le programme récepteur, en fonction du type de demande appelle une routine chargée de décoder le JSON et d'appeler la routine chargée d'effectuer la requête, il s'agit de la même routine utilisée par les programmes interactifs en mode VT

Témoignage

- Une mutuelle qui valide les devis d'opticiens depuis ses applications sur VMS (applications Basic + L4G ApTools) en répondant immédiatement, via Python et Soap.

- Auparavant il fallait pour les opticiens échanger par téléphone sur chaque dossier patient.



Avantages / Inconvénients

Avantages :

- Répond à la problématique d'intégration
- Garde la logique métier existante sous OpenVMS
- Capitalisation sur un existant éprouvé
- Sécurité, pas d'accès direct sur VMS
- Peu (ou pas) d'ajustement au code existant
- Ouverture multi langages
- Architecture moderne et motivante pour les jeunes développeurs
- Évolutivité / modularité (multi machines, disponibilité, migration,…)
- Agilité des développements

Inconvénients :

- Gestion des interactions clavier / écran à adapter
- Ré-urbanisation des applications monolithiques
- Nouveaux outils et méthodes (middleware)
- Sensibilisation nécessaire des développeurs historiques

Démonstration

Video disponible sur

https://www.vmsgenerations.fr/rdv-15-nov-2022/

Questions ?

Documents disponibles sur

https://www.vmsgenerations.fr/rdv-15-nov-2022/

Outils nécessaires pour la démo

- OpenVMS
- Linux
- RabbitMQ
- FastAPI
- Visual Studio Code

Plus d’infos sur les outils utilisés

- RabbitMQ :
  - https://fr.wikipedia.org/wiki/RabbitMQ
  - https://www.rabbitmq.com/
- FastAPI :
  - https://en.wikipedia.org/wiki/FastAPI
  - https://fastapi.tiangolo.com/
- JSON-RPC
- https://en.wikipedia.org/wiki/JSON-RPC
  - https://www.jsonrpc.org/
- Visual Studio Code :
  - https://code.visualstudio.com/

Les intervenants

- Miroslaw Szczeblewski — miroslaw.s@resyst.fr
- Benoît Maillard — benoitmaillard75@gmail.com
- Jean-François Piéronne — jfp@sysgroup.fr
- Rémi Jolin — remi.jolin@sysgroup.fr
- Christian Levrey — christian.levrey@cclc.fr

VMS generations

http://www.vmsgenerations.fr

VMSgenerations