

Developpement d'APIs au MAA : Comment ça marche ?

👍 Je souhaite développer une API

d'échange, de partage de référentiel, ... sans interface
(uniquement la partie Back end)

=> l'offre de service APYS est disponible

👍 Besoin d'information ?

=> [Je peux, ici, retrouver la documentation sur cette l'offre de service APYS.](#)

👍 **Pour prendre un bon départ** , obtenir des conseils d'architectures, de faisabilités, les bonnes pratiques....

=> Je fais une **demande d'appui PRO+**

👍 **Je fais valider mon choix en CAAT/ COTEC**

Et pour les développements orientés CLOUD ?

Les défis liés au développement Cloud => Cloud design patterns de Microsoft

- en fr => <https://docs.microsoft.com/fr-fr/azure/architecture/patterns/>

- en en => <https://docs.microsoft.com/bs-latn-ba/azure/architecture/patterns/>

Attention :Le traduction en Français est assez moyenne! 😊

- [Présentation ODS APYS](#)
- [Bonnes pratiques API](#)
- [Cas d'usages des API au MAA](#)
- [APYS, mais qu'est-ce-que c'est ?](#)
- [Architecture d'APYS - Modules de base et connecteurs](#)
- [Articles et supports de présentation sur les API](#)
- [Dernière communication sur la montée de version et support de l'offre APYS](#)
- [Gestion des API : Espace collaboratif](#)

Présentation ODS APYS

Nom	APYS
Version	v3.5.1
État	PUBLIÉE
Date de disponibilité	2 janv. 2024
Domaine	Socle d'application
MOA de l'offre	SNUM/BMQO
MOE de l'offre	<i>SNUM/XX</i>
Points de contact	Nora SAIFDINE
Espace Maaffluence	https://orion.agriculture/confluence/x/upEw

Besoins couverts par l'offre de service

APYS est une boîte à outils de développement d'API au MASA. Cette boîte à outils est contextualisée avec les connecteurs du MASA (bibliothèques de connexion aux services EAP, TYK OFS...).

Prerequis

Présentation des prérequis et contraintes

Exemple:

- Utilisation de JAVA8
- Utilisation de la Forge vXX

Mise en oeuvre

L'offre de Service fournit :

Architecture

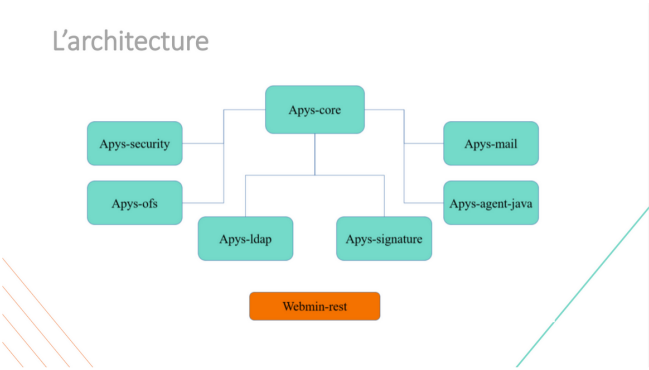
- Un Archetype Maven permettant d'initialiser une API
- Une documentation technique

L'architecture d'APYS est basé sur les briques du framework opensource JAVA Spring et Springboot.

Les versions disponibles sont:

Version	Description
APYS 3.5.1	<ul style="list-style-type: none">• Vulnérabilité: Version de Spring Framework upgradée en 5.3.33 pour corriger la CVE-2024-22259.• Ajout de la compatibilité avec la brique de signature V3.
APYS 3.4.0	<ul style="list-style-type: none">• Compatibilité VM RedHat8 : remplacement de la librairie javax. mail par l'implementation jakarta mail-api.

Briques Apys : vision globale



Pour lancer une procédure de migration sur APYS, suivre la procédure suivante: <https://reports.forge.agriculture.o2/apys-doc/release/3.3.1/apys-doc/migrate.html>

Pour démarrer un projet APYS, il faut utiliser l'archetype apys-archetype qui permet de générer un squelette de projet basé sur la dernière version APYS. Pour ce faire, utiliser la commande suivante:

```
mvn archetype:generate -DarchetypeGroupId=fr.gouv.agriculture.pro.archetypes -DarchetypeArtifact
```

Pour démarrer un projet Webmin-Rest, il faut utiliser l'archetype webmin-rest-server-archetype qui permet de générer un squelette de projet basé sur la nouvelle version de Webmin-rest.

Documentation

- Revoir la structure / organisation de la documentation APYS [Nora SAIFDINE](#)

Outillage

N/A

- [Présentation ODS APYS](#)
- [Bonnes pratiques API](#)
- [Cas d'usages des API au MAA](#)
- [APYS, mais qu'est-ce que c'est ?](#)
- [Architecture d'APYS - Modules de base et connecteurs](#)
- [Articles et supports de présentation sur les API](#)
- [Dernière communication sur la montée de version et support de l'offre APYS](#)
- [Gestion des API : Espace collaboratif](#)

Bonnes pratiques API

[APSMIB-636](#) - Consolider les bonnes pratiques sur le développement d'API

TERMINÉ

- [Guide de conception des API](#)
 - [Choix d'architecture](#)
 - [Format de réponse](#)
 - [Exemple de requête/réponse au format JSON](#)
 - [Exemple de requête/réponse au format XML](#)
 - [Conception de l'API](#)
 - [Mode « ressources » VS mode « services »](#)
 - [Mode ressources](#)
 - [Mode services](#)
 - [Principes généraux de design D'API](#)
 - [KISS - « Keep it simple, stupid »](#)
 - [API Stateless](#)
 - [Documentation](#)
 - [Granularité](#)
 - [Niveau d'ouverture des APIs](#)
 - [Construction de l'URI](#)
 - [Nommage des endpoints](#)
 - [Utiliser le pluriel](#)
 - [Casse cohérente](#)
 - [Versionning](#)
 - [Réponses partielles](#)
 - [Usage des verbes HTTP](#)
 - [Gestion des erreurs](#)
 - [Performances](#)
 - [Pagination](#)
 - [Filtres](#)
 - [Tris](#)
 - [Low Latency & Asynchronisme](#)
 - [Gestion des transactions](#)
- [Références](#)
- [Point à discuter](#)

Guide de conception des API

Ce document décrit les **principes d'architecture des APIs** retenus et les règles à suivre pour la conception d'API du Système d'Information Opérationnel du MAA.

Choix d'architecture

Il s'agit de s'accorder sur une normalisation de la conception des API.

✔ Une API doit être développée selon une architecture REST.

❗ **REST** (Representational State Transfer) ou RESTful est un style d'architecture permettant de construire des applications (Web, Intranet, Web Service). Il s'agit d'un ensemble de conventions et de bonnes pratiques à respecter et non d'une technologie à part entière. L'architecture REST utilise les spécifications originelles du protocole HTTP, plutôt que de réinventer une surcouche (comme le font SOAP ou XML-RPC par exemple).

Format de réponse

Pour répondre au mieux au consommateur d'API, il est nécessaire de définir un format de réponse unique.

✔ L'API doit répondre au format JSON à minima.

Exemple de requête/réponse au format JSON

Requête

```
CURL -X POST \  
-H "Accept: application/json" \  
https://api.national.agri/v1/users/007
```

Réponse

```
{  
  "id": "007",  
  "first_name": "James",  
  "name": "Bond",  
  "address": { "street": "Horsen Ferry Road" }  
}
```

L'API pourra renvoyer plusieurs autres formats si besoin (csv, xml...) en plus du format JSON.

Exemple de requête/réponse au format XML

Requête

```
CURL -X POST \  
-H "Accept: application/xml" \  
https://api.national.agri/v1/users/007
```

Réponse

```
<?xml version="1.0" ?>  
<id>007</id>  
<first_name>James</first_name>
```

```
<name>Bond</name>
<address>
  <street>Horsen Ferry Road</street>
</address>
```

Il est possible de préciser plusieurs format dans "Accept", séparé par ";"

```
GET https://api.national.agri/v1/users/007
Accept: application/xml; application/json    XML préféré à JSON
< 200 OK
< [XML]
```

```
GET https://api.national.agri/v1/users/007
Accept: text/plain; application/json    Text non pris en charge par l'api
< 200 OK
< [JSON]
```

Dans les cas où il n'est pas possible de fournir le format demandé, une erreur http 406 est retournée (cf. Gestion des erreurs)



Les noms dans le corps des messages sont en « [snake_case](#) ».

Conception de l'API

Mode « ressources » VS mode « services »

Il existe deux modes de conception des API REST : le mode « ressources » et le mode « services ».

Mode ressources

Le mode « ressources » permet une exposition simple et intuitive de la donnée, implique un découplage entre le fonctionnement de l'API et l'application qui l'utilise (notamment avec le modèle physique de données de l'application).

C'est ce mode qui est recommandé dans le cadre de l'APIsation. C'est une vision au service des consommateurs.

Exemples

Lecture

```
GET clients/007
```

Création

```
POST clients
```

Mise à jour

```
PUT clients/007
```

Suppression

```
DELETE users/007
```

Mode services

Le mode « **services** » permet la réutilisation de traitements complexes et implique une forte dépendance entre le consommateur (application, organisme ou personne) qui utilise l'API et le fournisseur : l'application doit connaître le fonctionnement de l'API pour l'utiliser. Ce mode n'est pas autorisé mais peut être utilisé de manière exceptionnelle : une dérogation explicite doit être demandée au groupe des experts échanges et API.

✓ La conception de l'API doit se faire en mode « ressources ».

"Non-Resources" scénarios

D'après la théorie RESTful, il est préférable d'utiliser le mode « ressources ». Or, en pratique, cela ne sera pas toujours possible, comme, par exemple, dans le cas de calculs ou des services métiers parfois complexes inhérents à un SI.

Dans ces cas là, l'opération doit être représentée par un verbe au lieu d'un nom.

Exemple de "Non-ressources" API:

Google Translate API

GET <https://www.googleapis.com/language/translate/v2.key=INSERT-YOUR-KEY&target=fr&q=Hello%20World>

Google Calendar API

POST <https://www.googleapis.com/language/calendar/v3/calendars/calendarId/clear>

Twitter Authentication

GET https://api.twitter.com/oauth/authenticate?oauth_token=TOKEN

Principes généraux de design D'API

KISS - « Keep it simple, stupid »

Faire des APIs simples techniquement, notamment pas d'API générique. Ne pas développer des « usines à gaz » génériques (typiquement pour le CRUD), préférer faire autant d'endpoints que de ressources.

API Stateless

✓ Les API REST doivent être stateless.

Stateless signifie que les APIs ne doivent conserver aucun contexte entre deux appels successifs.

Chaque appel est autoportant et son effet ne dépend que de l'état des ressources au moment de l'appel.

Documentation

Il est important, au moment de la conception de l'API, d'avoir l'objectif de créer une API intuitive, simple et attractive au service des consommateurs.

Il est donc nécessaire de mettre en œuvre les principes suivant :

- ✓ Une API doit être documentée et la documentation doit être compatible avec le catalogue du gestionnaire d'API : Swagger (basé sur la spécification [Open API](#))

Il est conseillé d'illustrer systématiquement les appels d'API via des exemples cURL dans la documentation de l'API, qui peuvent être utilisés en copier-coller, pour lever toute ambiguïté concernant les modalités d'appel et améliorer le TTFAC (Time To First API Call)

Exemple de commande cURL

```
CURL -X POST \  
-H "Accept: application/json" \  
-d '{"state":"running"}' \  
https://api.national.agri/v1/clients/007/orders
```

i Time To First API Call (TTFAC)

C'est le temps qu'il faut à un développeur pour faire un premier appel à l'api en production, enrôlement compris.

Granularité

En général, 1 API par application (par bloc applicatif du POS).

Chaque URL doit exposer un seul type de ressource (et éventuellement ses sous-ressources).

Si la théorie "une ressource = une URL" pousse à découper l'ensemble des ressources, il semble important de garder une limite raisonnable dans ce découpage.

Quelques points à retenir (cf blog octo).

- Grouper les ressources qui seront accédées à la suite de manière quasi systématique.
- Ne pas grouper les collections pouvant avoir de nombreux composants.
- Se limiter à deux niveaux d'imbrication (exemple : récupérer les informations d'une personne qui contiennent une adresse courante qui elle-même contient un pays → `/v1/users/addresses/countries`)

Niveau d'ouverture des APIs

All service interfaces, without exception, must be designed from the ground up to be externalizable.

Jeff Bezos

<https://api-university.com/blog/the-api-mandate>

Une bonne pratique serait de concevoir dès le départ une API comme si elle devait être ouverte au grand public (même pour les APIs internes).

Cela permet d'améliorer la qualité des développements et le passage au niveau d'ouverture supérieure se fait sans effort.

Construction de l'URI

Afin de faciliter la consommation des API de manière transverse (inter-domaines fonctionnels), il est important de normaliser les URIs et de respecter les règles de constructions décrites ci-dessous :

<PROTOCOLE> : // <QSI> <TYPE> <ENVIRONNEMENT> . national . agri / <API> / <VERSION> / <RESSOURCE> / <ID>

avec:

- PROTOCOLE: https
- QSI: QSI du ministère (Optionnel, suivi s'un tiret « - »)
- TYPE: « doc.api » s'il s'agit de la documentation, sinon « api »
- ENVIRONNEMENT: « -pprd » pour la préprod, ne pas renseigner pour la production
- API: API (Contexte) à laquelle la ressource est reliée
- VERSION: la version majeure de l'API
- RESSOURCE: Ressource recherchée
- ID: Identifiant de l'instance de la ressource recherchée

Exemple:

```
api.national.agri (api en production)
api-pprd.national.agri (api en environnement de pre-production)
doc.api.national.agri (documentation d'une API sur l'environnement de production)
doc.api-pprd.national.agri (documentation d'une API sur l'environnement de pre-production)
```

Exemple avec les QSI

```
dal.api.national.agri
dal-pprd.api.national.agri
doc.dal.api.national.agri
doc.dal-pprd.api.national.agri
```

Nommage des endpoints

✓ Ne pas utiliser de verbe mais des noms dans la définition de l'URI.

CONFORME	NON CONFORME
GET /clients	GET /getallclients GET /allclients

Utiliser le pluriel

Variation des noms de ressources entre singulier et pluriel complexifie l'utilisation des API. Pour cela, il est préconisé d'utiliser le pluriel pour gérer les deux types de ressources (collection et instance).

✓ Utiliser le pluriel pour la récupération d'une instance ainsi que pour lire les collections.

CONFORME	NON CONFORME
----------	--------------


Récupérer la liste des clients GET /clients Lire le client 007 GET /clients/007	Récupérer la liste des clients GET /client Lire le client 007 GET /client/007
--	--

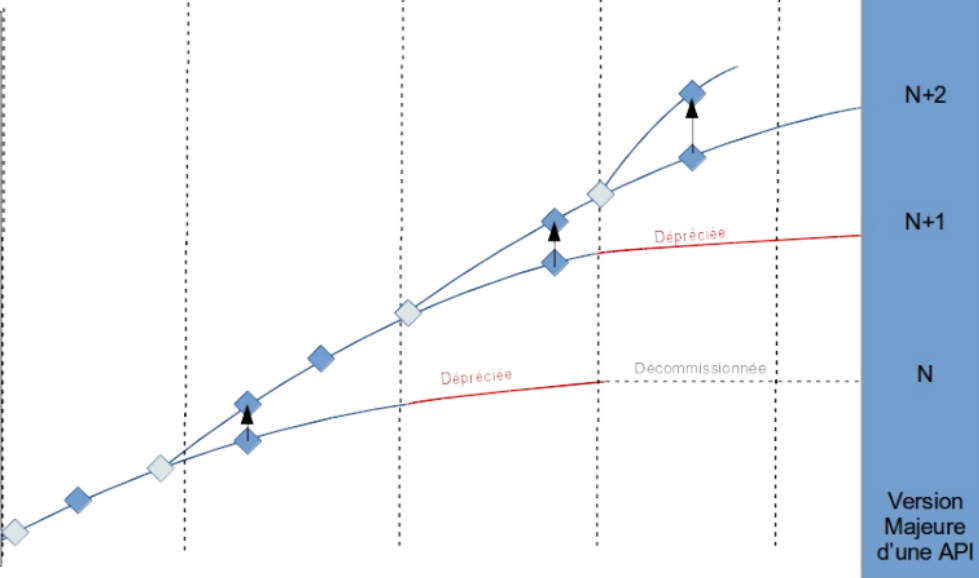
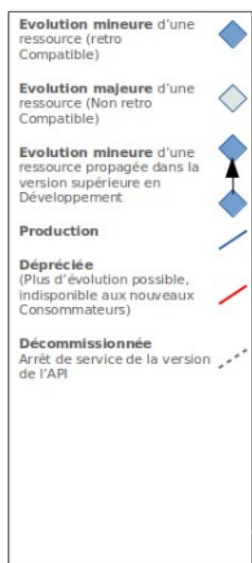
Casse cohérente

Utiliser le « [Snake case](#) » pour les URI et les body

CONFORME	NON CONFORME
GET /orders?id_client=007 POST /orders {"id_client":"007"}	GET /orders?idClient=007 POST /orders {"idClient":"007"}

Versionning

- 
- Une version d’une API (indépendamment de la version de l’application qui porte l’API) est écrite avec le format suivant : vX.Y.Z :
 - Version majeure (X, ex : 2) : concerne les mises à jour non rétro compatibles qui impactent les appels des consommateurs (changement de structure des objets échangés, suppression de champ ou d’éléments aux objets, suppression ou modification de services/opérations dans l’API...).
 - Version mineure (Y, ex : 8) : concerne les mises à jour rétro compatibles qui n’impactent pas les appels consommateurs existants (ajout d’option d’appel, de champs aux objets, de sous éléments aux objets, ajout d’opérations...).
 - Version patch (Z, ex : 5) : concerne la résolution de bugs en production (rétro compatibles)
 - La rétro compatibilité est l’ensemble des changements restants compatibles avec la version d’API précédente.



Le versionning de l'API doit se faire via l'URI. Seule la version **majeure** est exposée dans l'URI.
La version majeure figure dans l'URL au plus haut niveau du path de l'URI, préfixée par un « v ».

CONFORME	NON CONFORME
GET https://api.national.agri/api_name/ v2 /clients/007	https://api.national.agri/api_name/ v2.0 /clients/007
	https://api.national.agri/api_name/ 2 /clients/007
	https://api.national.agri/api_name/clients/007? version=2

Il est préconisé de supporter, au plus, deux versions en parallèle.

Réponses partielles

Dans certains cas (notamment nomades), le client d'une API peut ne pas avoir besoin de toute la grappe métier. Dans ce cas, l'API ne retournera que les informations demandées pour la ressource.

Nous appliquerons la norme jsonapi: <https://jsonapi.org/format/#fetching-sparse-fieldsets>



L'utilisation du paramètre **include** permet de spécifier les champs à récupérer.

Exemples

Récupérer le nom et le prénom du client 007

GET /clients/007?include=first_name,last_name

Dans le cas où l'on souhaite récupérer

Récupérer le nom, le prénom ainsi que la rue et la ville de l'adresse du client 007

```
GET /clients/007?include=first_name,last_name&fields[address]=street,city
```

Usage des verbes HTTP

Il est nécessaire de cadrer l'usage des verbes HTTP sur les ressources (via URI) pour conserver l'API intuitive.

Toutes les actions sur une ressource doivent être décrites avec un verbe et il est essentiel d'utiliser les mêmes méthodes pour la même action quelle que soit la ressource.

Verbe HTTP	Action associée: Collection (par exemple /orders)	Action associée: Instance (par exemple: /orders/{id de l'instance})	Autorisé
GET	Lister une collection de la ressource	Consulter une instance de la ressource	✓
POST	Créer une nouvelle instance de cette ressource		✓
PUT		Mettre à jour une instance de cette ressource (sans création)	✓
PATCH		Mettre à jour partiellement cette instance de la ressource	✓
DELETE		Supprimer cette instance de ressource	✓
TRACE	Echo la requête reçue pour qu'un client puisse voir quels changements ou ajouts (s'il y en a eu) ont été faits par des serveurs intermédiaires.		✗
HEAD		Vérifier l'existence de cette instance de ressource (sans la récupérer) et éventuellement consulter ses metadata	✗
OPTIONS	Déclarer les verbes disponibles pour une ressource donnée, et donc les actions autorisée sur cette dernière		✗

Gestion des erreurs

Utiliser les erreurs standard HTTP et s'appuyer sur les codes retour HTTP.

	HTTP Status	Description
20X: SUCCESS	200 OK	Code succès classique, fonctionnant dans les principaux cas. Spécialement utilisé lors d'une première requête
	201 Created	Indique qu'une nouvelle ressource a été créée. C'est la réponse typique aux requêtes PUT et POST, en incluant
	202 Accepted	Indique que la requête a été acceptée pour être traitée ultérieurement. C'est la réponse typique à un appel asy
	204 No Content	Indique que la requête a été traitée avec succès, mais qu'il n'y a pas de réponse à retourner. Souvent retourné
	206 Partial Content	La réponse est incomplète. Typiquement retourné par les réponses paginées.

40X: CLIENT ERROR	400 Bad Request	<p>Généralement utilisé pour les erreurs d'appels, si aucun autre statut ne correspond. On peut distinguer deux types :</p> <ul style="list-style-type: none"> Request behaviour error, exemple: <pre>GET /users?payed=1 < 400 Bad Request < {"error": "invalid_request", "error_description": "There is no 'payed' property"}</pre> <ul style="list-style-type: none"> Application condition error, exemple: <pre>POST /users {"name": "John Doe"} < 400 Bad Request < {"error": "invalid_user", "error_description": "A user must have an email address"}</pre>
	401 Unauthorized	<p>Je ne vous connais pas, dites moi qui vous êtes et je vérifierai vos habilitations.</p> <pre>GET /users/42/orders < 401 Unauthorized < {"error": "no_credentials", "error_description": "This resource is under protection"}</pre>
	403 Forbidden	<p>Vous êtes correctement authentifié, mais vous n'êtes pas suffisamment habilité.</p> <pre>GET /users/42/orders < 403 Forbidden < {"error": "not_allowed", "error_description": "You're not allowed to perform this action"}</pre>
	404 Not Found	<p>La ressource que vous demandez n'existe pas.</p> <pre>GET /users/999999/ < 404 Not Found < {"error": "not_found", "error_description": "The user with the id '999999' does not exist"}</pre>
	405 Method not allowed	<p>Soit l'appel d'une méthode n'a pas de sens sur cette ressource, soit l'utilisateur n'est pas habilité à réaliser cette action.</p> <pre>POST /users/8000 < 405 Method Not Allowed < {"error": "method_does_not_make_sense", "error_description": "How would you create a user?"}</pre>
	406 Not Acceptable	<p>Rien ne match au Header Accept-* de la requête. Par exemple, vous demandez une ressource XML or la ressource n'est que JSON.</p> <pre>GET /usersAccept: text/xmlAccept-Language: fr-fr < 406 Not Acceptable < Content-Type: application/json < {"error": "not_acceptable", "available_languages": ["us-en", "de", "kr-ko"]}</pre>
50X: SERVER ERROR	500	<p>L'appel de la ressource est valide, mais un problème d'exécution est rencontré. Le client ne peut pas réellement savoir ce qui s'est passé.</p> <pre>GET /users < 500 Internal server error < Content-Type: application/json < {"error": "server_error", "error_description": "Oops! Something went wrong.."} </pre>

Performances

Pagination

Il est nécessaire de prévoir dès le début de l'API la pagination des ressources.

Nous appliquerons la norme jsonapi pour la pagination: <https://jsonapi.org/format/#fetching-pagination>



Les paramètres **page[number]** et **page[size]** seront utilisés pour la pagination.

Paginer les ressources avec des valeurs par défaut lorsque celles-ci ne sont pas spécifiées par l'appelant.

Exemple

Filtre

```
GET /articles?page[number]=3&page[size]=1
```

Filtres

Nous appliquerons la norme jsonapi pour filtrer les résultats: <https://jsonapi.org/format/#fetching-filtering>



Pour filtrer les résultats, le client d'une API précise en paramètres HTTP un paramètre **filter** pour chaque critère de filtre, contenant: le nom de l'attribut entre crochets, l'opérateur entre crochets, le caractère « = », suivi des valeurs de filtre, chacune séparée par une virgule.

Exemple

Filtre

```
GET /users?filter[birthday.year][gte]=1983&filter[gender][eq]=1
```

Tris

Nous appliquerons la norme jsonapi pour trier les résultats: <https://jsonapi.org/format/#fetching-sorting>



- Le paramètre **sort** doit être utilisé. Il contient le noms des attributs, séparés par une virgule, sur lesquels effectuer le tri
- Par défaut le tri est ascendant (ou croissant). Afin de l'obtenir de façon descendant, il suffit de préfixer le champs par le caractère « - » (U+002D HYPHEN-MINUS, “-“)

Exemple

Récupérer la liste des utilisateurs triée alphabétiquement sur le nom

```
GET /users?sort=name
```

Récupérer la liste des utilisateurs (ORDER BY name DESC)

```
GET /users?sort=-name
```

Récupérer la liste des utilisateurs (ORDER BY name DESC, first_name ASC, salary DESC)


```
GET /users?sort=-name,first_name,-salary
```

Récupérer la liste des utilisateurs (ORDER BY address.city ASC)

```
GET /users?sort=address.city
```

Low Latency & Asynchronisme

En lecture, les temps de réponses doivent être en dessous de la seconde. Au delà, il sera préférable de passer en mode asynchrone.

 Un temps acceptable de réponse, pour une meilleure expérience utilisateur, correspond en moyenne à 200ms.

Gestion des transactions

Dans le cadre d'une API REST, il n'y a pas de contexte transactionnel entre 2 appels API:


- HTTP est un protocole requête/réponse,
- REST est une architecture stateless

Exemple du blog d'Octo avec la gestion de comptes:

Si, en tant que client, je souhaite faire un virement entre deux comptes, le fait de réaliser deux requêtes PATCH successives sur `/accounts/{1}` et `/accounts/{2}` sera hors contexte transactionnel. Si une exception se produit entre les deux actions, alors votre argent est potentiellement perdu dans le [cyberespace](#)...

pour résoudre cette problématique, il existe deux solutions :

1. utiliser une compensation fonctionnelle. Coté client, appeler une action de rollback (par exemple un appel à DELETE `/accounts/{1}` si PATCH `/accounts/{2}` échoue). Ou bien annuler la transaction coté serveur via un batch par exemple, et avertir l'utilisateur du comportement exceptionnel de manière asynchrone, tout rétablissant la cohérence du/des système(s)
2. modéliser cette suite d'actions interdépendantes par une ressource unique (un message) qui fera alors l'objet d'un seul appel. Si on reprend l'exemple précédent, alors il devient évident que ce que l'on cherche à faire est d'enregistrer une transaction bancaire. La ressource POST `/transaction`, avec un corps de requête adapté sera alors plus évident. La garantie de l'atomicité et éventuellement de la réversibilité de l'action relève de l'implémentation de l'API et non de son utilisation correcte par les clients.

 La gestion des transactions doit être étudiée au cas par cas, mais la solution d'une transaction par une ressource unique est préconisée.

Références

Guide des bonnes pratiques du MEN	MENJ - APIisation - Règles - v1.7.pdf
Blog de la société Octo	https://blog.octo.com
DAL - Principes d'architecture des APIs	DAL-CT_71_DA_Principes_d'architecture_des_APIs_V1.odt
Spécification JSONAPI 1.0	https://jsonapi.org

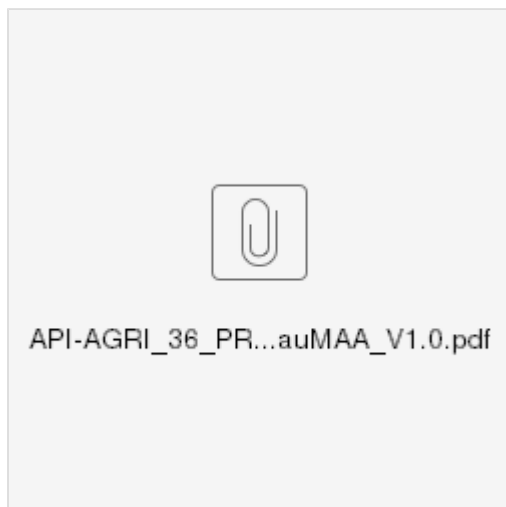
Point à discuter

1. Gestion des dépendances (cf document du DAL)
 - a. [PGDep] Le client d'une API doit être livré par le fournisseur de l'API. L'application qui utilise cette API recense cette dépendance d'infrastructure via une dépendance Maven pour pouvoir suivre les versions des APIs utilisées, et quelles versions de quels clients utilisent telle version de telle API.
2. API (uri, body, documentation) en anglais ou en français ?
3. Exposer des UUID au lieu des ids pour identifier plus proprement les ressources, et se protéger contre des attaques.
4. Gestion du temps (utiliser le format Timestamp UTC)
5. Devons nous préconiser HATEOAS (Hypermedia as the Engine of Application State) ?
6. Quota, throttling à documenter dans l'API ?
7. Low Latency & Asynchronisme
8. Gestion des transactions (en modification) étalées sur plusieurs APIs

Cas d'usages des API au MAA

EN COURS

DOCUMENT de TRAVAIL



Orientations prévues, à discuter, au cas par cas, en COTEC

Les cas d'usage des API

Exposition de données consommées en temps réel, extraction de données par les clients (type échange de données), mise à jour de données, données référentielles, données publiques, données métier, proposer un service, un calcul, API destinée à être appelée par d'autres API, micro services, des API techniques ...

Les API sont vouées essentiellement à être utilisées par un client (utilisateur : B2C ou application cliente : B2B) pour offrir un service ou des données. Il existe néanmoins plusieurs distinctions dans l'usage (la consommation) d'une API et également dans la construction de cette API et par rapport au type de service qu'elle pourrait rendre. A partir de là, il s'agit de catégoriser les différentes API et plus particulièrement les différents usages. Ceci permettra par la suite de définir des pattern (d'architecture) pour les différents cas d'usages.

NB : une API doit apporter de la valeur (à son utilisateur/client)

Réflexions sur les catégorisation possibles des cas d'usages API

Pas de typologie d'API mais on peut les catégoriser par domaine métier ou par type.

Pour établir une catégorisation et dresser la liste des cas d'usage on repose sur plusieurs aspects :

- l'aspect fonctionnel : API de détection, API d'enrichissement, API de perception, API d'action
- l'aspect technique : adaptation technique pur (interface legacy), service technique,
- l'aspect sécurité : privée, publiques, sensible, etc
- l'aspect état/mode de communication : synchrone, asynchrone, workflow, publish & subscribe
- l'aspect utilisation : volumétrie, quota, facturation

Autre découpage possible, suivant :

- Organisation (par exemple, développeur mobile ou équipe principale)
- Appartenance (frontières sectorielles)
- Sécurité (zone délimitarisée ou réseau interne)
- Qualité de service (système mainframe ou distribué)
- Écosystème (public ou partenaire)

De plus, les applications/le rôle d'une API peut avoir différente portée : API donnée, API pour l'infra/le cloud (son rôle est surtout l'automatisation), API pour un produit (habilitations, configuration, ...), API pour un composant technique (authentification ? mailing ?)

Liste (non exhaustive) de cas d'usage d'API (pour essayer de catégoriser par la suite et définir les pattern) :

- API et mise en réseau (networking) : intégration avec le réseau pour trace et information, automatisation du provisionnement réseau (configuration), visibilité réseau avec "on premise network service"
- API et service transverse : mailing, authentification LDAP, édition de document, signature de document
- API de données référentielles : BDNU
- API de transformation/consolidation de données : données géographique
- API d'adaptation (passe-plat) : interface legacy (export de certaines données en base sous forme fichier pour flux partenaire)
- Besoins de mise en place d'échanges de données (ce qui est actuellement géré par Secoia en point à point)
- ...

Exemple dans l'industrie : <https://developer.ibm.com/apiconnect/2015/11/01/api-use-cases-for-every-industry/>

(mais là c'est vraiment une approche par domaine métier et purement fonctionnelle)

Par rapport à ce qui a été évoqué, les cas d'usages dans le contexte du MAA seront axés autour des impacts sur l'architecture. Les cas d'usages sont donc les suivants :

Tableau de synthèse

Usage	Catégorie	Commentaire
API de structure intra-applicatif	API Métier - Architecture	<p>Cas d'usage surtout pour le découpage front-end / back-end au sein de la même application. Données à usage privée (en général mono-applicatif). Peut-être également une première étape avant l'exposition de données de l'application (en réutilisant la même API, mais plus de sécurisation à prévoir).</p> <p>Par exemple : Cepp, Intrants, Sati v2</p>
API d'un produit	API Métier /Technique ? - Administration ?	<p>Ce cas d'usage couvre les utilisations d'API de produits (progiciels ou applications) qui offre des API pour configurer les produits à distance (hors IHM). Ces API en général sont très sécurisées (parfois réservé uniquement aux administrateurs) et à usage privée uniquement. Peut être utilisé également pour de l'automatisation (scripting) de configuration du produit.</p> <p>Par exemple : Tyk, API Cloud (networking : provisionning d'infrastructure => openstack)</p>
API de données référentielles	API Métier - Référentiel	<p>Utilisé pour la centralisation de données de références (dans le cas de données référentielles). L'API peut être à vocation interne MAA ou ouverte à tous. La sécurisation est directement lié à la sensibilité des données.</p> <p>Par exemple : , Miquado (Sirius API) pour le référentiel des usagers du MAA</p>
API de données métiers	API Métier - Référentiel	<p>Utilisé pour partager des données métiers d'un domaine métier particulier. L'API peut être à vocation interne MAA ou ouverte à tous. La sécurisation est directement lié à la sensibilité des données.</p> <p>Par exemple : Arpent (données métiers "examens"), export vers daa.gouv.fr ou OpenDataSoft des données Ensagri</p>
API façade / d'adaptation	API Métier - Architecture ?	<p>Ce cas d'usage porte uniquement sur la vocation d'exposer une interface (en général legacy) existante qui ne pourrait pas être exposée tel quel. Dans ce cas de figure, l'API exposée est un "passe-plat" qui n'altère pas les données d'origines.</p> <p>Par exemple : certains flux de SECOIA qui sont au format fichier ou données SQL, transformation d'une interface exposée par un ESB.</p> <p>NB : actuellement c'est le cas pour tous nos webservices SOAP qui ne peuvent pas être exposés tel quel dans Tyk (soit on les réécrit, soit on utilise une API façade/d'adaptation).</p>

API de services métiers	API Métier - Traitement ?	<p>Cas d'usage pour consolider des données brutes (exposées elles-même en API ou non) en général en y apportant les traitements métiers adaptés pour des clients qui n'ont besoin que de certaines données résultats. Ce cas est généralement utilisé sur des données brutes complexes (en terme de connaissance métier) qui en l'état ne serait pas exploitable pour la plupart des clients.</p> <p>Par exemple : données géographiques consolidation sous forme de rapport synthétique (proche du décisionnel néanmoins), IFT</p> <p>A noter qu'il peut y avoir une hiérarchie d'API (une API appelant un ou plusieurs autres API).</p>
API de services techniques (transverse)	API Technique - Traitement ?	<p>Ces API servent en général à centraliser l'utilisation d'une plateforme transverse dédiée à rendre un service unique à un ensemble d'applications clientes. Ces API sont rarement publiques car elles sont en frontal d'une plateforme transverse stratégique (en terme de disponibilité et/ou de sécurité).</p> <p>Par exemple : Plateforme de Signature, Distancier</p>

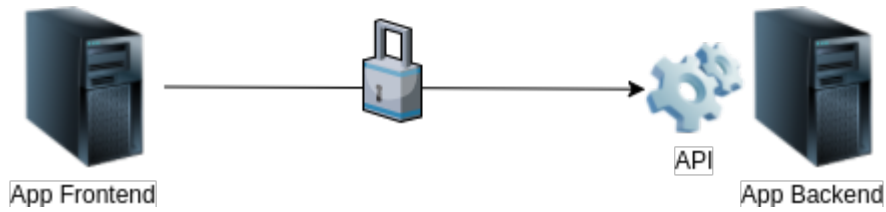
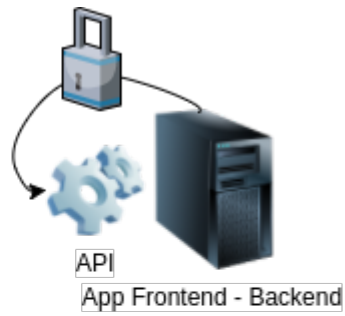
Caractéristiques des API

Pour chaque cas d'usage cités précédemment on a une liste d'attributs qui déterminent les caractéristiques majeures de l'API :

- Portée : privées, partenaires, publiques
- Niveau de sécurité (DICP) : authentification, habilitations, délégation, chiffrement, ...
- Gouvernance / contrat de service : exposition sans API manager, documentation, facturation, quota, versionning, ...
- Technologie : REST, SOAP, ...
- mode de communication : synchrone / asynchrone, utilisation d'un workflow, événementiel (publish & subscribe) a une influence sur le cas d'usage d'une API dans certains cas, surtout vis à vis de l'architecture.
 - ou pourrait parler même du mode d'intégration = quels composants tiers sont directement liés à ce que permet de faire l'API ?

Réflexions sur les pattern d'architecture

Cas 1 : API de structure intra-applicatif



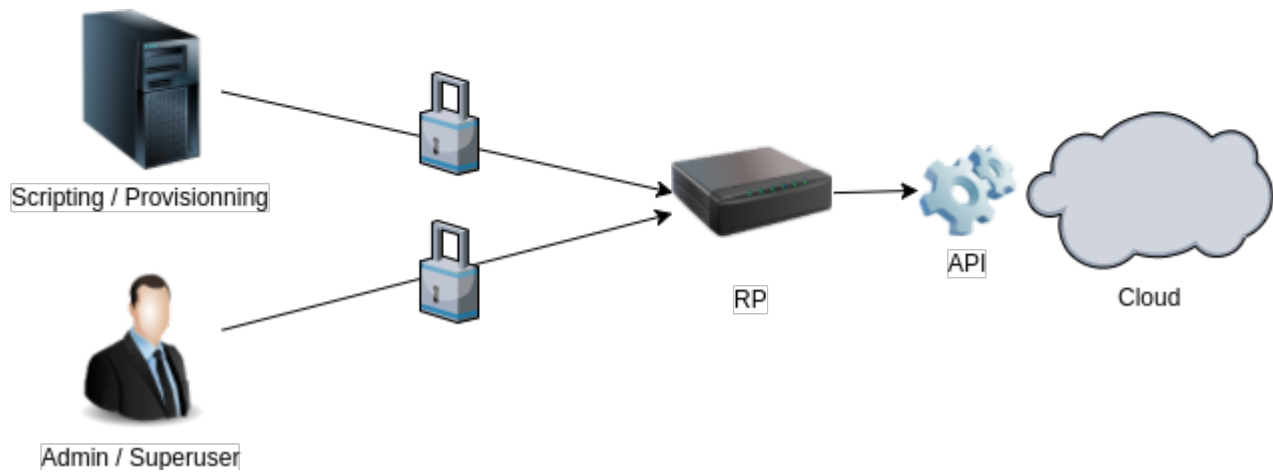
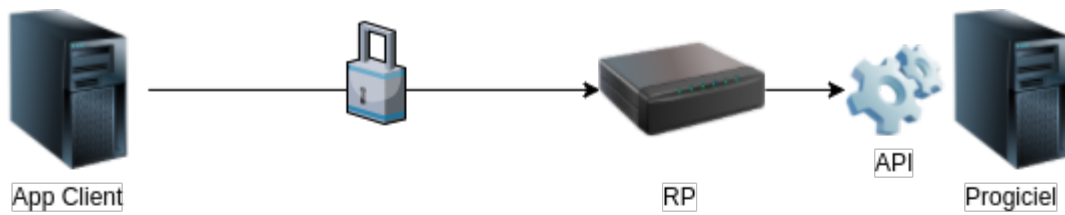
Sécurité : SSL, API Key, HMAC Signature

API Manager : Non

Infrastructure réseau : pas besoin de RP

Spécificités : le découpage structurelle peut être sur le même hostname (premier schéma) ou sur deux différents (second schéma)

Cas 2 : API d'un produit



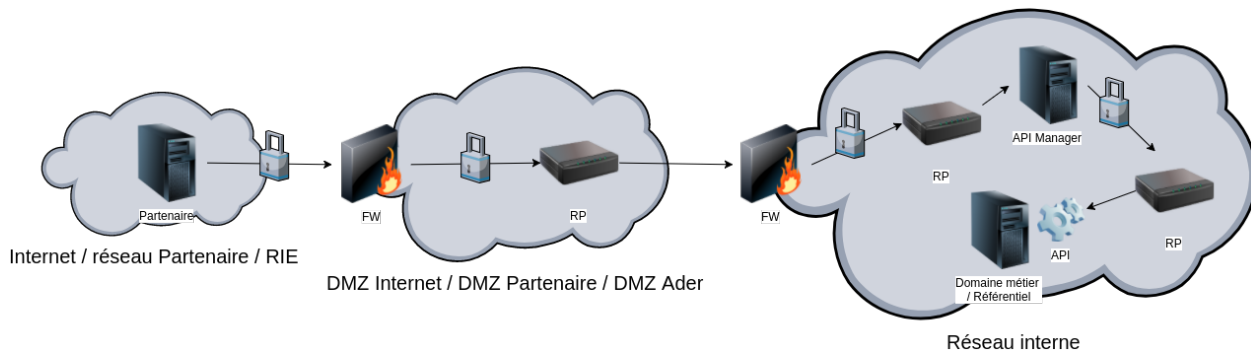
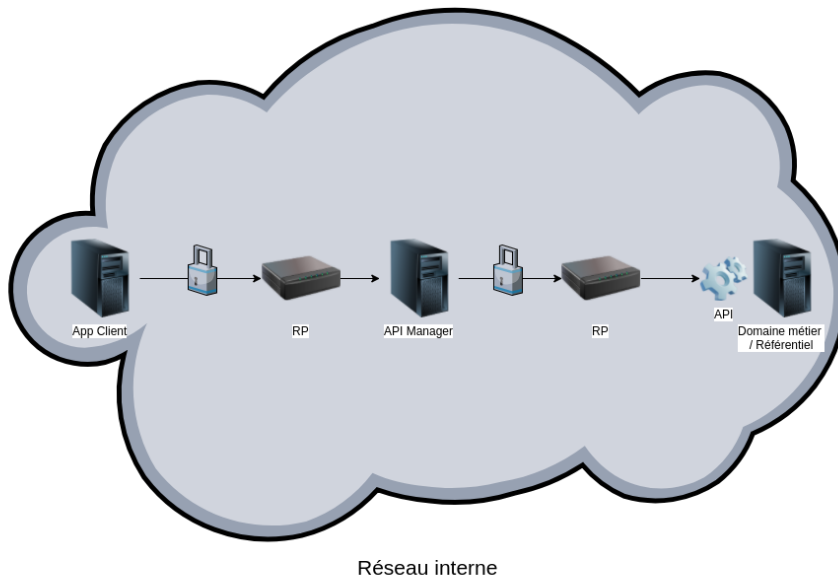
Sécurité : SSL, API Key, Clé privée (suivant le produit)

API Manager : Non

Infrastructure réseau : utilisation d'un RP

Spécificités : API essentiellement d'administration ou de rôle "superuser", utilisé soit par un applicatif, soit par un utilisateur identifié.
La portée est essentiellement privée

Cas 3 & 4 : API de données référentielles / métiers



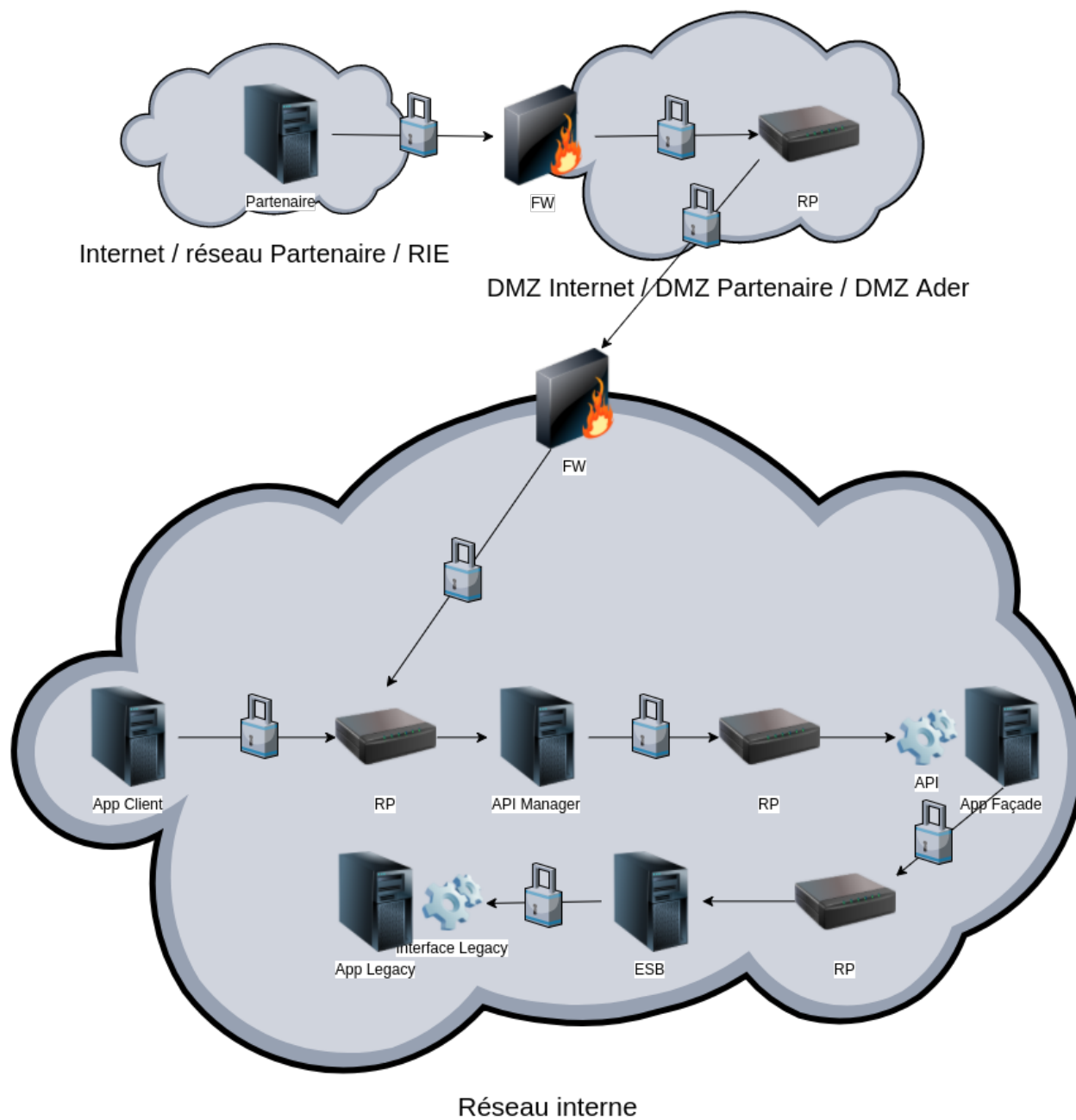
Sécurité : SSL, API Key, HMAC Signature

API Manager : Oui

Infrastructure réseau : utilisation d'un RP, DMZ, Firewall

Spécificités : la portée peut être partenaire (réseau interne ou RIE) ou publique (internet)

Cas 5 : API façade / d'adaptation



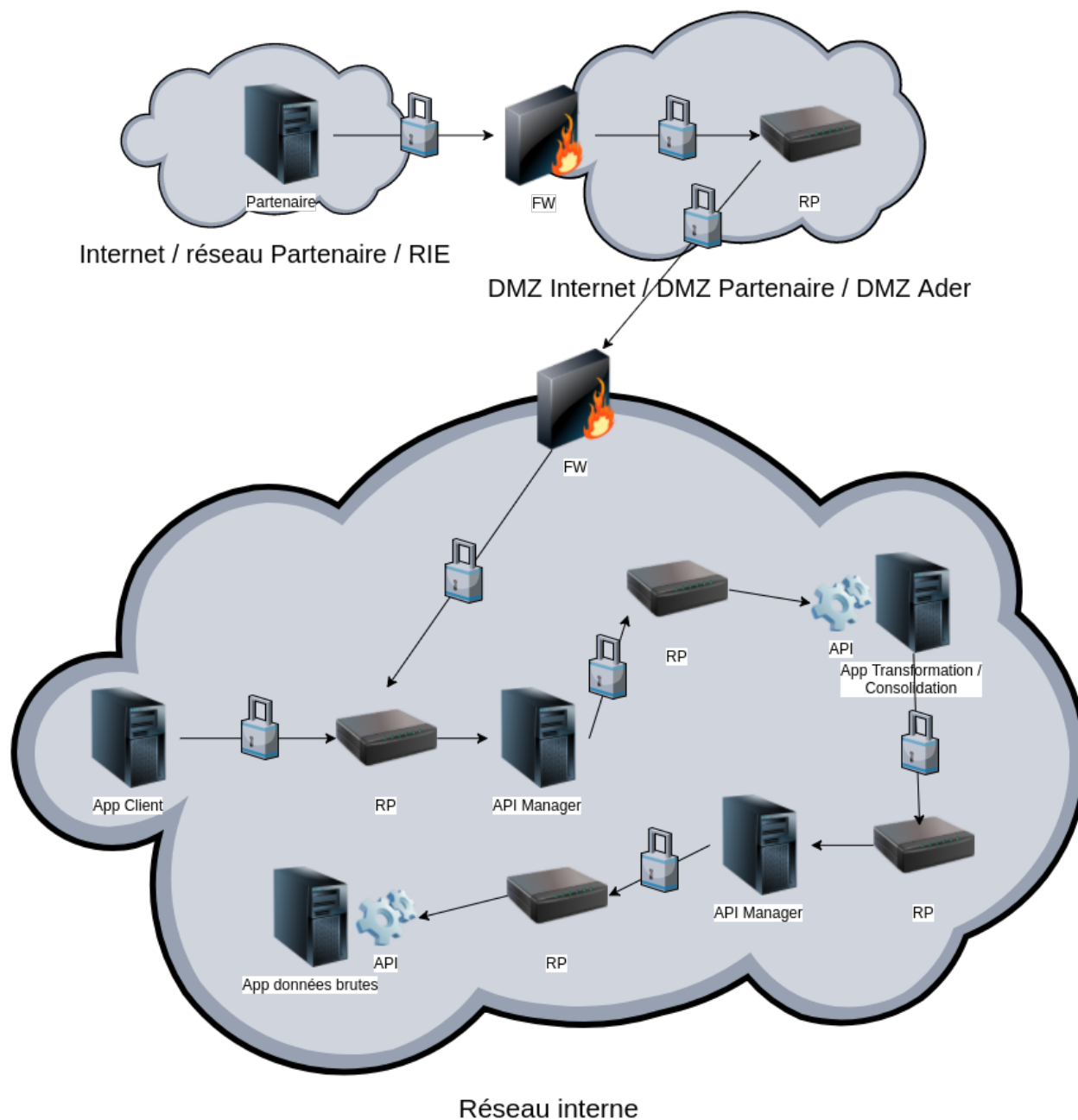
Sécurité : SSL, API Key, HMAC Signature

API Manager :Oui

Infrastructure réseau : utilisation d'un RP, DMZ, Firewall

Spécificités : API "passe-plat" (façade), la portée peut être partenaire ou publique

Cas 6 : API de services métier



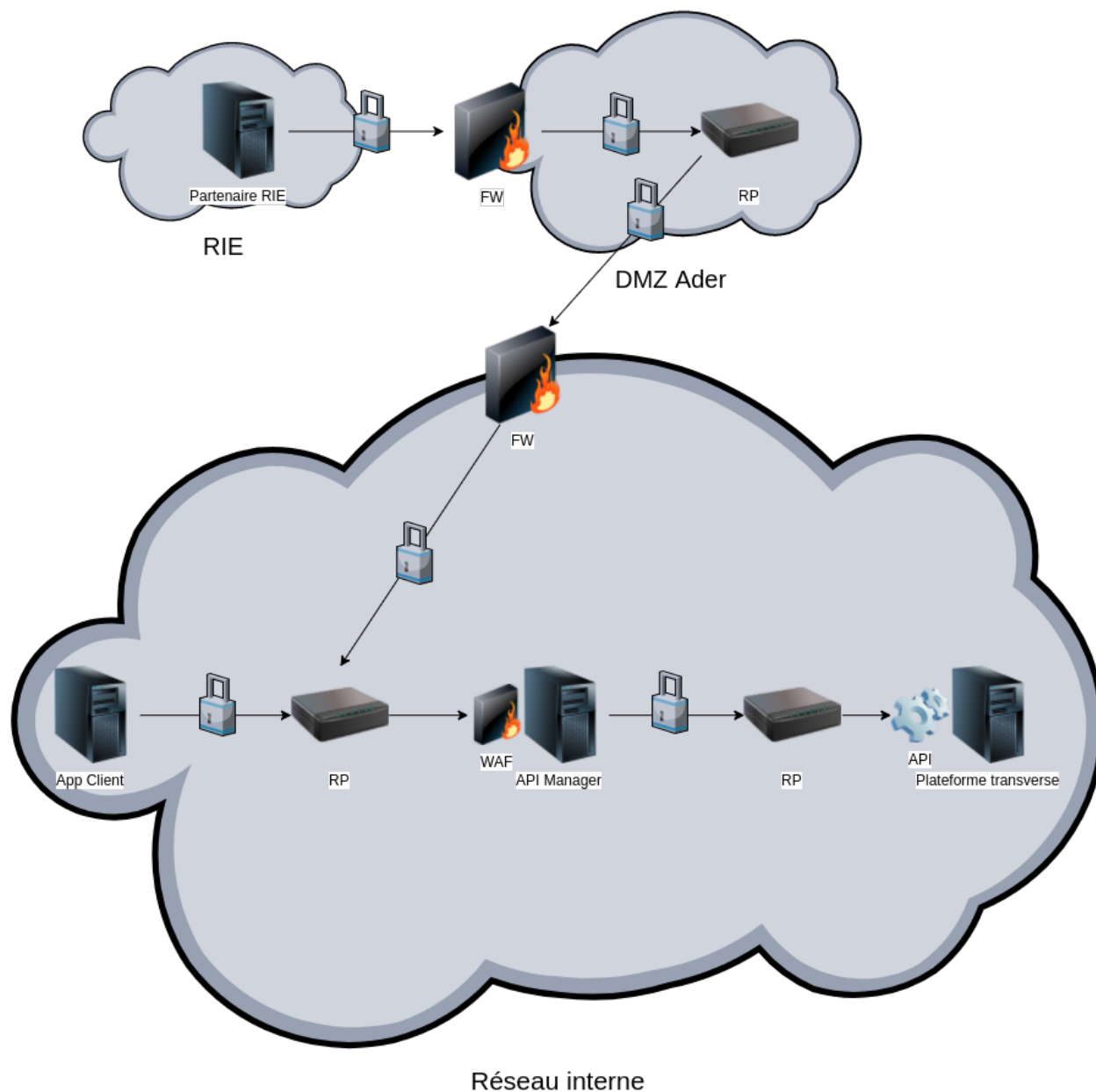
Sécurité : SSL, API Key, Clé privée (suivant le produit)

API Manager : Oui

Infrastructure réseau : utilisation d'un RP, DMZ, Firewall

Spécificités : la portée peut être partenaire ou publique, masque la complexité des données brutes

Cas 7 : API technique / Service transverse



Sécurité : SSL, HMAC Signature, Certificat

API Manager : Oui

Infrastructure réseau : utilisation d'un RP, WAF (Firewall applicatif) ?

Spécificités : la plateforme en backend est généralement stratégique/sensible, sécurisation forte et portée uniquement partenaire

APYS, mais qu'est-ce que c'est ?



APYS est une nouvelle "boîte à outils"

- à l'origine, développée pour les besoins de France Agrimer
- basée sur les briques du framework opensource JAVA Spring et Springboot,
- conçue pour faciliter le développement d'API au MAA
- contextualisée avec les connecteurs du MAA (bibliothèques de connexion aux services EAP, TYK OFS...)

Pourquoi choisir le framework Spring ?

- il est opensource
- il est basé sur une architecture en microservice
- il est compatible cloud
- il a un cadre de développement souple
- il est reconnu de la communauté des développeurs
- il est connu par la majorité des prestataires de services
- il permet de développer des applications JAVA modernes
- et il est toujours maintenu et mis à jour régulièrement.

La trajectoire

Les orientations prises dans la réalisation de cette boîte à outil permettent de suivre la **tendance technologique** en terme de développements et de s'inscrire dans la **stratégie d'évolution du MAA** à savoir :

- l'APIsation du MAA (amplifiée par la sortie de Secoia)
- la mise en place du Cloud
- "dé-corneriser" les applications du MAA
- externalisation des développements
- des versions régulières, pour intégrer à minima une version Spring par an



APYS c'est plusieurs versions par an, dont les plus importantes

- **APYS 3.5.1**
 - **Vulnérabilité:** Version de Spring Framework upgradée en 5.3.33 pour corriger la CVE-2024-22259.
 - Ajout de la compatibilité avec la brique de signature V3.
- **APYS 3.4.0**

Pour aller plus loin, avec un peu de lecture...

Documentation APYS :

- [L'architecture - Les modules et les connecteurs](#)
- [Manuel d'utilisation](#)

Documentation Webmin-rest :

- [Manuel d'utilisation Webmin-Rest 1.0](#)



Et concrètement !

- Pour lancer une [procédure de migration sur APYS, c'est ici](#).
- Pour démarrer un projet APYS, vous avez [un archetype apys-archetype qui permet de générer un squelette de projet basé sur la dernière version APYS, par là](#).
- Ou bien, pour démarrer un nouveau projet Webmin-Rest, vous avez [un archetype webmin-rest-server-archetype en version 1.0 qui permet de générer un squelette de projet basé sur la nouvelle version de Webmin-rest](#)

```
mvn archetype:generate -DarchetypeGroup
```

- Compatibilité VM RedHat8 : remplacement de la librairie javax.mail par l'implementation jakarta mail-api.

- **APYS 3.3.1**

- Ajout de l'authentification via EAF

- **APYS 3.2.1**

- Montée de version Spring Boot en version 2.4.5

- **APYS 3.1**

- Version Authentification LDAP

- **APYS 3.0**

- Version Cloud-Ready

- **APYS 2.0**

- Version de fin d'accompagnement des projets "pilotes" à l'utilisation de APYS et MELYS

- **APYS 1.0**

- Première version avec le pilote Endor 1.0

Stratégie

Pour suivre les évolutions de Spring, nous réalisons une montée de version par an pour upgrader la version du framework.

Et si ce n'est pas suffisant !

Vous pouvez consulter les [notes de version](#).

Et sinon, il y a les offres de service au sein du centre de production

Pour le déploiement des API

=> deux offres de service au catalogue du BSIP (TOMCAT ou WILDFLY)

=> par le [guichet web \(IWS\) du BSIP](#).

En intégration/contrôle, les machines Tomcat 8, Wildfly16 sont accessibles via AdminBastion et le déploiement se fait via Rundeck: [livraison-et-déploiement-war-vers-tomcat](#)



Maintenance des versions

La maintenance des versions APYS est assurée sur la plus récente version intermédiaire des deux dernières versions majeures.

Version évolutive	Date de mise à disposition	Date de fin d'assistance (+5 an)	Date de fin de maintenance	Version corrective actuelle	Date de mise à disposition
1.0	30 nov. 2018	30 nov. 2023	2 oct. 2020	1.0.10	1 avr. 2020
2.0	10 juil. 2020	10 juil. 2025	Sous maintenance		
3.0	2 oct. 2020	2 oct. 2025	6 avr. 2021	Aucune	
3.1	6 avr. 2021	6 avr. 2026	28 mai 2021	Aucune	
3.2	28 mai 2021	28 mai 2026	26 juil. 2021		
3.3.1	26 juil. 2021	26 juil. 2026	Sous maintenance		
3.4.0	4 juil. 2023				
3.5.0	22 avr. 2024				
3.5.1	22 avr. 2024				

PS : Spring assure le support sur les 2 dernières versions sortie, et propose une version majeure par an.

Aide, Assistance et Appui

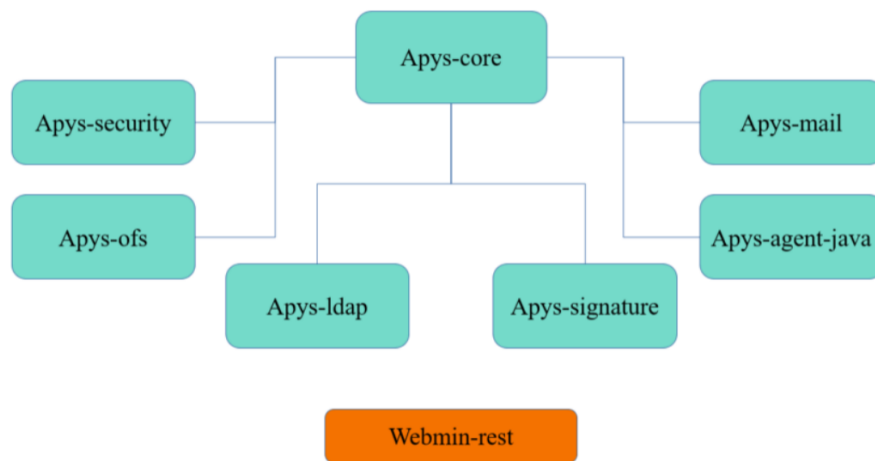
Les procédures de demandes d'assistance ou d'évolution sont les mêmes que pour celles d'Orion

=> ticket IWS

Architecture d'APYS - Modules de base et connecteurs

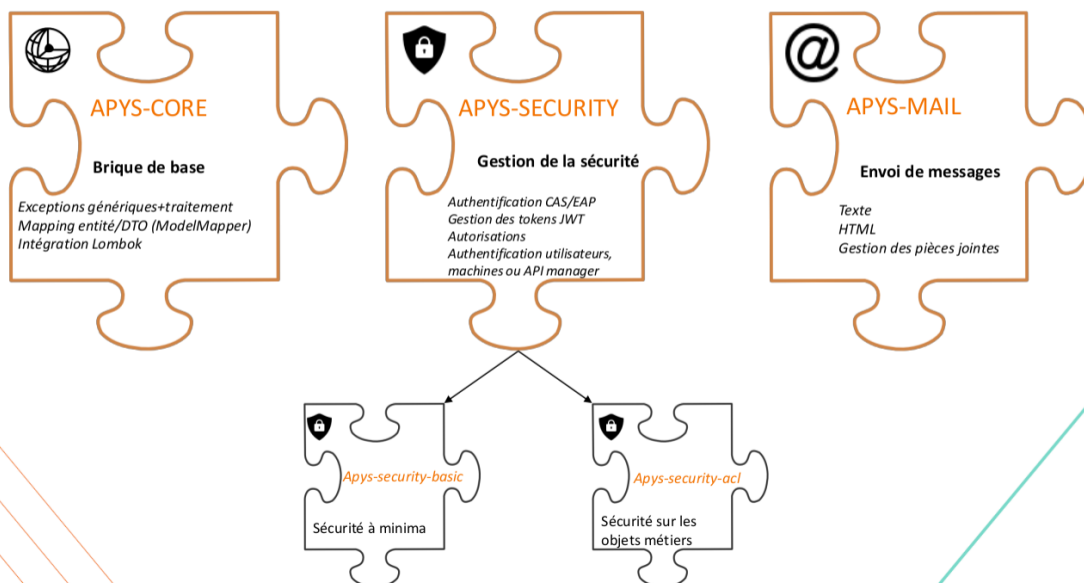
Briques Apys : vision globale

L'architecture



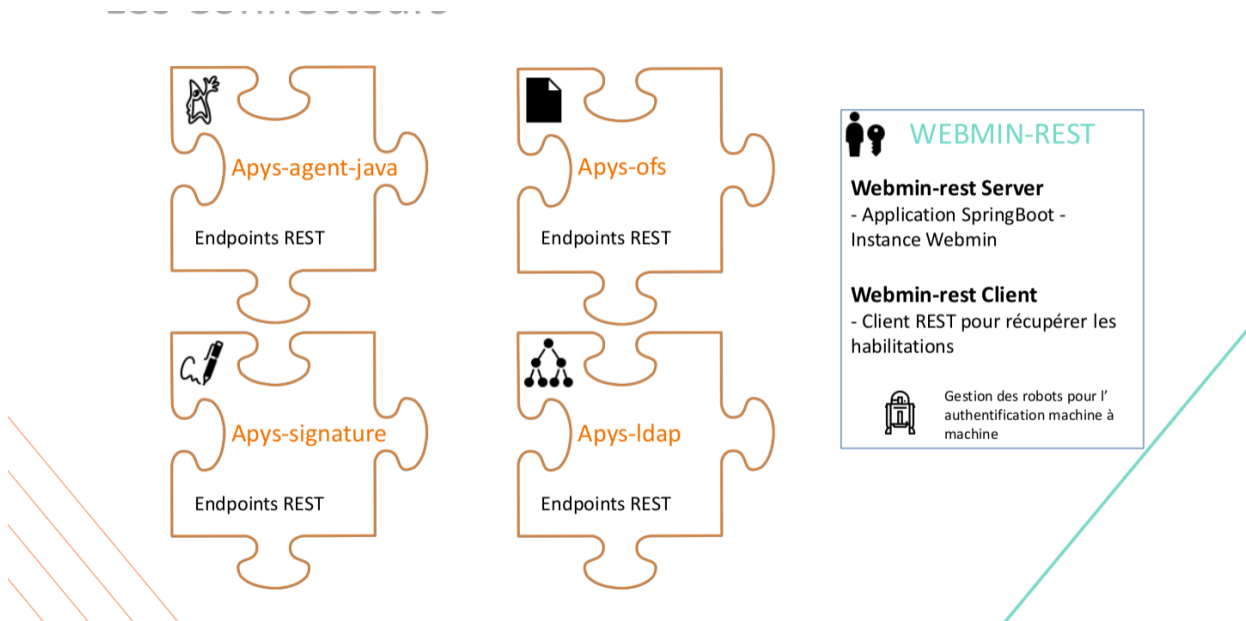
Modules Apys : socle

Les modules de base



Connecteurs Apys : utilisation étendue aux outils du MAA

Les Connecteurs



Articles et supports de présentation sur les API

- [Supports fait par le ministère Agri.](#)
- [Enjeux](#)
- [Conception API](#)
- [Sécurité](#)
- [Open API / Open Data](#)

Supports faits par le ministère

- **Rapport de préconisations WaveStone (Juin 2018) : document de référence pour l'élaboration de notre cadre technique**
 - [Plan d'action proposé associé](#) au rapport
- **Démarche API-AGRI :**
 - [Support de présentation agents SDSI](#) (Septembre 2018)
 - [Support de présentation réseau AMOA](#) (Septembre 2018)
 - [Feuille de route](#) - (Juillet 2018)
 - [Point stratégie](#) - (Novembre 2017)
 - [Validation trajectoire TYK](#) - (Juin 2017)
 - [Réunion de lancement du projet Tykado](#) - 27 avr. 2018
- **Présentations projets**
 - [IFT à l'AppliDay 2018](#)
 - [Consommer la Base adresse national \(BAN\)](#) - Caflab 15 mars 2018
 - [Miquado : Api Sirius](#) - Caflab 14 déc. 2017
 - [Présentation de l'API-MANAGER](#) - Caflab 12 déc. 2017
- **Retour d'expérience faits hors ministère**
 - [CRIP : GROUPE DE TRAVAIL « APIisation du SI »](#) (Avril 2019)
 - ITES Deauville CRIP (mars 2019)
 - [CRIP SudOuest Météo-France](#) (Janvier 2019)

- APIdays (Décembre 2018 - anglais)
 - [Support de présentation](#)
 - [Texte du discours](#)
 - [Vidéo de la présentation](#) (disponible également sur [youtube](#))
- [Usage de TYK au MAA présenté à la DINSIC](#) (Décembre 2018)
- [Article sur le site du ministère au sujet de l'APIsation](#) (Décembre 2018)

Comprendre les enjeux stratégiques

- [Bref article Octo qui synthétise le sujet](#)
- [Découplage, découplage, découplage](#)
- [OCTO Talks - Stratégie d'architecture API](#) Article à lire impérativement pour comprendre le sujet. API dans son ensemble.

Notes de lecture

- API technique vs API as a Product => Il y aura là une distinction à faire.
- Qu'est qu'on met exactement derrière l'idée 'API' ?
 - fonctionnellement
 - techniquement

Problématique / enjeu

- ATAWAD : « Any time, anywhere, any device »
- Ouverture du SI : pour développer + de possibilités métiers et accélérer l'innovation

En répondant à ces deux enjeux, l'API est le vecteur qui offre la capacité à industrialiser sa présence dans le digital.

Définitions:

API : interface normalisée par laquelle un logiciel offre des services à un autre logiciel

- **Web API** est + précis si on ouvre cette interface via http pour donner accès à un SI

API c'est avant tout : exposition des [ressources](#) et le système de persistance sous-jacent

API = un ensemble de ressources exposées

Les applications finales reposent sur la composition de plusieurs ressources provenant éventuellement de systèmes disjoints : nous parlons de [mashup \(ou application composite\)](#).

Le pattern rejoint ainsi les principes majeurs des [architectures orientées service \(SOA\)](#) : le découplage et la possibilité de composition

Les API sont basées sur une architecture REST (avec une approche plus ou moins puriste)

Les APIs REST sont une forme de SOA, dont l'objectif est d'utiliser HTTP comme protocole applicatif => [architectures orientées Web \(WOA\)](#)

REST

- une modélisation orientée ressource (et non opération),
- qui tire partie des forces du standard HTTP et des architectures Web déjà en place et maîtrisées depuis les années 90,
- et qui permet une consommation par tout type de terminal numérique et navigateur internet.

OPEN API = on crée une API sans à priori vis à vis des clients qui vont la consommer.

API FIRST = on raisonne API avant tout. Y compris pour nos propres besoins.

HATEOAS = niveau le + abouti du RestFull. On enrichi la logique d'exposition des ressources en ajoutant les liens pertinents pour chaque ressources (par exemple, si la requête qui est soumise demande une liste d'objet; pour chaque objet que je retourne dans ma réponse j'en rajoute aussi une liste de liens associés, comme le lien vers la fiche de l'objet, son propriétaire etc...)

3 niveaux d'ouverture d'une API :

- Niveau 1 « *Private API* » : l'API est destinée à être consommée par les applications développées au sein de l'entreprise.
- Niveau 2 « *Partners API* » : l'API est destinée à être consommée par les applications développées au sein de l'entreprise ou par des partenaires.
- Niveau 3 « *Open API* » : l'API est destinée à être consommée par tout type de développeurs.

La différence entre chaque niveau réside principalement dans la qualité du **processus d'industrialisation de consommation des services** qu'il convient de mettre en œuvre :

- au niveau 1, un développeur d'application est contraint de consommer l'API et peut – en cas de difficultés – directement contacter l'équipe API.
- au niveau 3, le succès d'une API, qui peut être consommée par des milliers d'utilisateurs sur la toile, dépend de la facilité du processus d'enrôlement et des qualités intrinsèques de l'API et de sa documentation

En ciblant les niveaux 1 ou 2, le risque est de retomber dans une démarche *point à point* en ciblant des cas d'usages spécifiques

Concevoir et développer des API



Guide de bonnes pratiques API de la Dinsic

[Guide de bonnes pratiques pour la conception d'APIs v1-1.pdf](#)

- [Normes du gouvernement du Canada sur les API](#) => Très complet.
- **APIdays**
 - L'ensemble des [vidéos](#) sur youtube
 - Les [slides](#) des présentations
- [API Landscape](#)



API Landscape .pdf

- [Conception technique et fonctionnelle](#), article d'Octo
- [Designer une API, la Ref Card](https://blog.octo.com/designer-une-api-rest/) : <https://blog.octo.com/designer-une-api-rest/>



OCTO-Refcard_AP...sign_EN_3.0.pdf

- [Faire son catalogue d'API](#)
- [Restful foundations](#) (*anglais*)
- [Processus d'ingénierie pour le développement d'API](#) (*anglais*)
- [Versioning et ZDD - Zero Downtime Deployment](#)
- [Versioning](#) (*anglais*)
- Animation Web Service faite par PrO+ : différences entre SOAP et REST ([2016-07-29 Animation WebServices](#))

Les styles d'architecture SOAP et REST s'opposent fondamentalement sur trois niveaux, illustrés par le tableau ci-dessous :

SOAP/RPC	REST
Modélisation par opération	Modélisation par ressources
Cherche à unifier le modèle de programmation distribué et local par l'intermédiaire d'un proxy	Modèle de programmation distribué explicite, et basé sur le WWW
Repose sur un toolkit pour être interprété	Mise sur une bonne expérience développeurs tous niveaux
Consommé que par des serveurs	Consommé par tout type de device, y compris les serveurs

Développement

- [Comprendre HATEOAS](#)
- [PayPal's API Style Guide and Patterns](#) (*anglais*)
- [PUT ou POST ?](#)
- [Formats et méthodes de sérialisation Rest](#)
- Familiarisez-vous avec Markdown : <http://ricostacruz.com/cheatsheets/markdown.html> ! Avec un éditeur, c'est plus facile de s'y retrouver : <http://dillinger.io/>

Sécurité des API

- RefCard d'Octo sur la sécurité des APIs



OCTO-Refcard_AP...Security_BD.pdf

- [Comprendre les mécanismes OAuth2](#)
- [OAuth2 et OpenIdConnect](#)

Ouvrir les données par des API

[Api.gouv.fr](#)

- <https://api.gouv.fr/>
- [GitHub Api.gouv.fr](#)
- [Documentation publication API.gouv.fr](#)
- L'exemple du [travail du projet Arpent](#) pour aller sur [api.gouv.fr](#)

Données publiques

- [Data.gouv.fr](#)
- OpenDataSoft : [Exemple des données de l'enseignement agricole](#)
- The World Bank : [Guide Pratique des Données Ouvertes](#)

État Plateforme

<http://etatplateforme.modernisation.gouv.fr/>

Dernière communication sur la montée de version et support de l'offre APYS

L'équipe PRO+ vous informe que l'offre APYS évolue !

Bonjour,

L'équipe PRO+ vous informe que la version **3.3.1** d'APYS est désormais disponible !

APYS, cela ne vous dit rien ? Venez voir ce qui se cache derrière ce terme sur [la page MAAFFLUENCE dédiée](#). Pour les initiés, vous pouvez consulter le contenu de la version depuis les [notes de versions](#). **Liens à MAJ**

Nous vous invitons à effectuer les montées de version nécessaires dès que possible, en suivant les [instructions de migration](#). **Liens à MAJ**

En effet, pour chaque nouvelle version, la maintenance corrective pour la correction de bugs évolue. Elle est assurée uniquement pour **la dernière version intermédiaire des deux dernières versions majeures**. Par conséquent, seuls les bugs remontés pour les versions **2.0.0** et **3.3.1** sont actuellement pris en charge dans nos développements.

Le support reste disponible pour l'ensemble des versions sous la forme d'une assistance technique. Pour nous contacter, faire un ticket à assistbmsq.sg@agriculture.gouv.fr.

Merci d'avance pour votre compréhension.

Cordialement,

L'équipe PRO+

Gestion des API : Espace collaboratif

[Lien vers l'espace documentaire "Provisoire"](#)



Questions - Réponses



Articles de conseil,
Comptes rendu, Astuces,

Retours d'expérience

Questions fréquentes

Poser une question

Rubriques :

- [club-api](#)
- [tyk](#)
- [api](#)

Aucune question n'a été posée ici pour le moment. Cliquez sur le bouton ci-dessous pour être le premier à en poser.

Boite à idée collaborative

=> propositions, ateliers, animations,
Retour expérience, partages de
connaissances



[Add Idea](#)

Synthèse d'Ideation

Idées 2 Contributeurs 2 Commentaires 0

Proposed 2

- [Liste](#)
- [Tableau](#)

[Statut: Tous](#)



- **PROPOSED**
- **EVALUATED**
- **ACHIEVED**
- **ARCHIVE**

[Créateurs: Tous](#)

- [APYS, mais qu'est-ce que c'est ?](#) (API-AGRI)
 - [club-api](#)
- [2020-05-07 \[TYK\]\[CNERTA\] Accès WS depuis FREGATA](#) (APPUI SMIB)
 - [meeting-notes](#)
 - [club-api](#)
- [2020-02-27 \[DPRS\] \[BDNU\] Déploiement du Web service courrier BDNU](#) (APPUI SMIB)
 - [meeting-notes](#)
 - [club-api](#)
- [API : informations DINUM](#) (BIALOG)
 - [api](#)
 - [actu_bmsq](#)
 - [club-api](#)
- [2019-11-18 \[DPRS\] \[BDNU\] Web service courrier BDNU TLS 1.2](#) (APPUI SMIB)
 - [meeting-notes](#)
 - [club-api](#)
- [2019-10-17 \[TYK\]\[ADAGE\] Problème dans le dispositif de deploiement de l'api dans TYK](#) (APPUI SMIB)
 - [meeting-notes](#)
 - [club-api](#)
- [2019-08-05 \[API\] \[DAL\]\[PRELEVEMENT-ANALYSE\] Atelier d'architecture sur les solutions d'échange au MAA](#) (APPUI SMIB)
 - [club-api](#)
- [2019-07-30 \[TYK\] \[DPRS\] \[SIRIUS\] Demande évolution API Dashboard TYK](#) (APPUI SMIB)
 - [club-api](#)
 - [tyk](#)
- [2019-04-15 \[TYK\] ECF/INDEXA : Atelier API application AEEA](#) (APPUI SMIB)
 - [meeting-notes](#)
 - [club-api](#)
 - [tyk](#)
-



Utilisateur courant



Pas de créateur spécifique

Période de création

Choisir la période de création



Aujourd'hui Dernière semaine Dernier mois Réinitialiser
De
à

Filtrer Annuler

1



PROPOSED [Retex :Le découpage des API en services](#)

Ajoutée par [Delphine Clapié](#) le Mar 22, 2018

Qui peut le faire? ;-) Le découpage des API en services (Sirius)

- [ideation-blueprint](#)

6 0

[\[TYK\] Interruption de service, instance TYK ORION environnement CTRL, le vendredi 22/03/19 \(12h00-14h00\)](#) (PRO+)

- [club-api](#)



[2019-02-11 \[TYK\] Ouverture de flux sur l'instance TYK ORION CTRL via LS](#) (APPUI SMIB)

- [meeting-notes](#)
- [club-api](#)
- [tyk](#)



[2019-02-05 - DAL / BDNI: Refonte flux SIAL BDNI BOVEX sur SES](#) (APPUI SMIB)

- [meeting-notes](#)
- [club-api](#)
- [tyk](#)



[2019-01-29 - DAL / SI2A: Choix technologique : Flux générique SECOIA / API](#) (APPUI SMIB)

- [meeting-notes](#)
- [club-api](#)
- [tyk](#)



[2018-12-19 ADAGE/PECHE : lancement développement API](#) (APPUI SMIB)

- [meeting-notes](#)
- [club-api](#)
- [tyk](#)



[2018-11-15 CNERTA-DATA API-WS Calcul des bourses](#) (APPUI SMIB)

- [meeting-notes](#)
- [tyk](#)
- [club-api](#)

1



PROPOSED [Trop bonne idée !](#)

Ajoutée par [Anaïs Carme](#) le Feb 16, 2018

génial idée

- [ideation-blueprint](#)

20  0 

