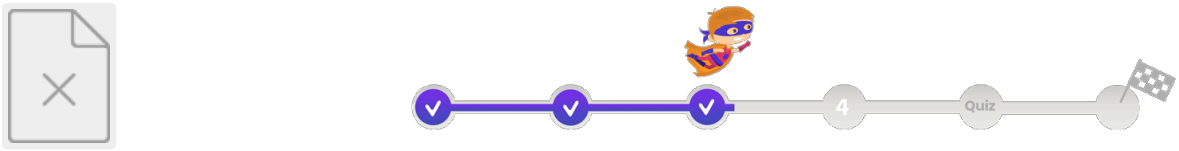
4 - Les services Stateless et Stateful



4. Les services Stateless et Stateful

En matière de service ou d'API, il est primordial de distinguer **deux comportements** fondamentalement opposés.

4.1 Les services dits « stateful »

Pour comprendre le concept de « **stateless** », il faut comprendre le « **stateful** ». Quand on parle de **systèmes informatiques**, un « state », état en français, est simplement **la condition ou la qualité d'une entité à un instant donné.**

Être "**stateful**", c'est **se baser sur des moments** dans le temps et **changer la sortie** en fonction des **entrées** et de l'**état** déterminés.

C'est un concept difficile à maitriser. On peut également le représenter avec **le système binaire.** Il **ne peut pas** être à la fois **1 et 0,** les deux sont **mutuellement exclusifs.**
Maintenant, considérez une situation théorique dans laquelle on vous donne un morceau de papier avec les instructions suivantes : si le nombre est à 0, dites « non », s'il est à 1, dites « oui »
Vous êtes ensuite placé dans une pièce avec un écran qui afficherait un nombre, 0 ou 1 toutes les cinq secondes.
Il s'agit d'un **système d'état.** Votre réponse dépendra entièrement de savoir si cette horloge dit « 0 » ou « 1 ». Vous ne pouvez pas répondre indépendamment de l'état de la grande machine.

C'est l'état d'esprit des services dits « **stateful** », avec **état stocké**.

Dans la réalité, à quoi ressemble un service Web « stateful » ?
Disons que vous vous **connectez à une ressource** et ce faisant, vous passez **votre mot de passe et votre nom d'utilisateur.** Si le **serveur Web stocke** ces données dans **une base de données** et les **utilise pour vous identifier** en tant que client connecté en permanence, **le service est « stateful ».** Gardez à l'esprit qu'il s'agit d'un exemple très spécifique et qu'il existe sous d'autres formes, donc ce qui semble être un état ne l'est pas nécessairement.
Lorsque vous utilisez **le service « stateful »,** tout ce que vous faites est **renvoyé à cet état stocké**.
Lorsque vous demandez un sommaire de compte, le service Web vous demande deux choses :

- Qui fait cette demande ?
- En utilisant l'ID stocké pour qui fait cette demande, à quoi devrait ressembler leur page ?

Dans un **service Web "stateful"** comme celui-ci, la **réponse** formée à partir d'une simple requête **dépend entièrement** de l'**état enregistré** par le serveur. Sans connaissance de cet état, votre demande ne peut pas être retournée correctement.

Un autre excellent **exemple** est le service **FTP**.
Lorsqu'un utilisateur se connecte à un serveur FTP traditionnel, il établit une **connexion active** avec le **serveur.** Chaque **changement d'état** de l'utilisateur, tel que le répertoire actif, est **stocké sur le serveur** en tant qu'**état client**. Chaque modification apportée au serveur est enregistrée comme un **changement d'état**. Lorsque l'utilisateur se déconnecte, son **état** est encore **modifié** pour être déconnecté.

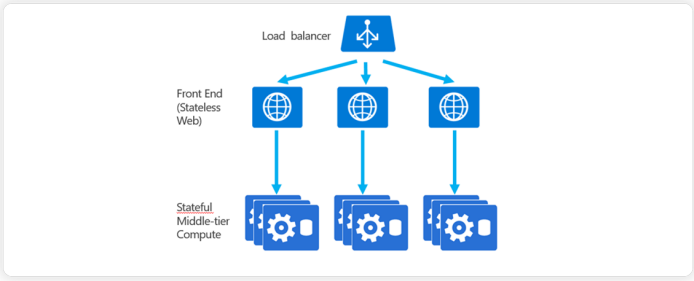


Figure 1 : Application alliant des services « stateless » et « stateful »

La **programmation de service « stateful »** est très bien dans certaines applications très limitées, mais elle a **beaucoup d'inconvénients.** Avant tout, quand on doit faire référence à un état, on s'ouvre à un grand nombre de sessions et de transactions incomplètes. Disons que vous faites un appel pour présenter un élément de données. Dans un système dynamique où l'état est déterminé par le client, combien de temps le système est-il censé laisser cette connexion ouverte ? Comment vérifier si le client s'est écrasé ou déconnecté ? Comment pouvons-nous suivre les actions

de l'utilisateur tout en conservant la capacité de documenter les changements et de revenir en arrière si nécessaire ?

4.2 Les services dits « stateless »

La réponse aux questions précédentes vient avec le service **"stateless"**, littéralement **sans état stocké**.
"Stateless" est **l'opposé** polaire de "stateful", dans lequel **toute réponse** donnée du serveur est **indépendante d'un état**.

Retournons à la théorie de la **salle binaire.** On vous donne la même horloge binaire, seulement cette fois, le papier a simplement un nom – « Bob » - et les instructions sont de répondre quand quelqu'un dit le mot de passe « Alice ». Vous êtes assis à regarder l'horloge changer lentement, et chaque fois que quelqu'un dit le mot de passe spécial, vous dites le nom « Bob ».
Vous êtes **« stateless »,** il n'est même pas nécessaire de faire référence à l'horloge, car **l'information est stockée localement** de telle sorte que les **demandes sont autonomes.** Vous ne **dépendez que des données que vous détenez.** L'orateur pourrait facilement dire le mot secret, vous dire de changer le nom, puis s'éloigner. Il peut alors revenir une heure plus tard, dire le mot de passe secret, et obtenir le nouveau nom - tout est contenu dans la demande, et traité en deux phases distinctes, avec une « demande » et une « réponse ».
Il s'agit d'un système sans état. Votre réponse est indépendante du "0" ou du "1", et chaque demande est autonome.

Le **service "stateless"** est un aspect **fondamental** de l'**Internet moderne** - à tel point que **chaque jour**, vous **utilisez** une variété de services et d'applications "stateless".

Lorsque vous lisez les nouvelles, vous utilisez le protocole HTTP pour vous connecter sans état, en utilisant des messages qui peuvent être analysés et traités séparément les uns des autres et de votre état.

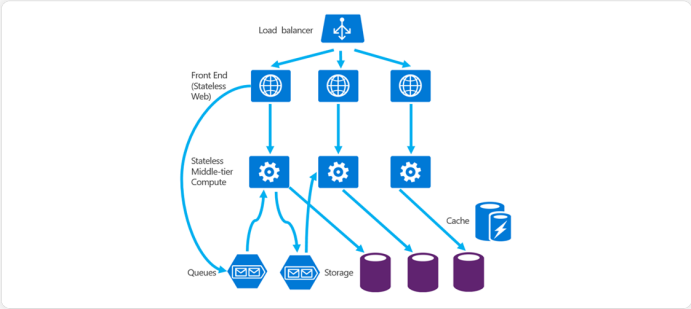


Figure 2 : Application complètement « stateless »

Si vous avez un **compte Twitter**, vous utilisez constamment un service **"stateless"**.

Lorsque le service demande une liste de messages directs récents à l'aide de l'API REST de Twitter, il émet la demande suivante :
GET https://api.twitter.com/1.1/direct_messages.json?since_id=12345
La réponse que vous obtiendrez est **entièrement indépendante du stockage d'état du serveur**, et tout est **stocké du côté du client sous la forme d'un cache.**
Dans l'exemple ci-dessous, nous invoquons une commande « POST », permettant d'envoyer des données sur MonSuperService :

```
POST http://MonSuperService/Entity
Host: MonSuperService
Content-Type: text/xml; charset=utf-8
Content-Length: 123
1
Lorem Ipsum
```

Dans cet exemple, nous créons une entrée, mais cette entrée ne dépend d'aucun état. Gardez à l'esprit qu'il s'agit d'un cas d'utilisation simple, car il ne transmet aucune donnée d'autorisation/authentification, et que l'émission du « POST » ne contient que des données de base.
Même avec toutes ces informations en tête, vous pouvez clairement voir que **faire une émission « POST » de manière « stateless »** signifie que vous **n'avez pas besoin d'attendre la synchronisation du serveur** pour vous assurer que **le processus a été correctement complété,** comme **vous le feriez avec FTP ou d'autres services « stateful ».**
Vous recevez une confirmation, mais cette confirmation est simplement une réponse affirmative, plutôt qu'un état partagé.

En guise de remarque rapide, il faut dire que le **REST** est **spécifiquement conçu** pour être **"stateless"**.
Le concept du **REpresentational State Transfe**r (d'où REST tire son nom) repose sur l'idée de **passer toutes les données** pour **traiter la demande** de manière à **associer** les données au sein de la requête elle-même. Ainsi, REST devrait être considéré comme "stateless" (et en fait, c'est l'une des principales considérations quant à savoir si quelque chose est RESTful ou non selon la thèse originale de Roy Fieldind qui a détaillé le concept.

4.3 Les usages compliqués

Nous devons faire preuve d'une **certaine prudence** lorsque nous parlons des services Web en tant qu'exemples « stateful » ou « stateless », car ce qui semble relever d'une catégorie peut ne pas l'être.

C'est en grande partie parce que le**s services "stateless"** ont réussi à **reflété**r une grande partie du **comportement des services "stateful"** sans franchir techniquement la ligne de démarcation.

Un service de type « stateless » fait référence à une question d'état **autonome** et de **référence** plutôt que de dépendance à une référence externe. La **différence entre l'état « stateless » et l'état « stateful »** dépend uniquement de **l'endroit où l'état est vraiment stocké**. Lorsque nous naviguons sur Internet ou que nous accédons à notre courrier, nous générons un état et cet état doit être stocké.
Lorsque **l'état est stocké par le serveur,** il génère **une session.** On parle alors de **service « stateful ».** Lorsque **l'état est stocké par le client,** dans son navigateur par exemple, il génère une sorte de **pseudo données** qui peut être utilisée pour divers systèmes, alors que nous avons techniquement à faire à un service « stateful » en ce sens qu'il fait référence à un état, l'état est stocké par le client, donc nous il doit être considéré comme « sans état ».
Cela semble déroutant, mais c'est en fait la meilleure façon de contourner les limites du « stateless ». Dans un système purement **« stateless »,** nous sommes essentiellement en interaction avec un système limité.

Si vous deviez **acheter** quelque chose **en ligne** sur ce type de service, le système ne **stockerait pas** votre adresse, vos méthodes de paiement, voir même l'enregistrement de votre commande, il **traiterait simplement** votre paiement et il ne **garderait aucune information**.

Ce n'est évidemment pas le meilleur scénario, il y’a donc des concessions qui ont été faites.
Dans le **cookie** (un petit morceau de vos données, souvent un identifiant unique) qui se trouve sur votre navigateur Web, il est souvent stocké quelques données d'authentification.
Côté serveur, des données clients temporaires sont créées puis référencées dans un stockage de données externe.
Lorsque nous revenons pour effectuer un autre paiement, c'est votre cookie qui établit l'état, et non la session.

4.4 Le problème des sessions ?

En matière de services Web, le paradigme communément accepté est **d'éviter au maximum les sessions.** Bien que cela ne s'applique certainement pas à chaque cas d'utilisation unique, l'utilisation des sessions comme méthode de communication d'état est généralement quelque chose que vous voudrez éviter.

Pour commencer, les sessions **ajoutent** une **grande** quantité de **complexité** avec **très peu de valeur ajoutée**.

Les sessions rendent plus difficile la réplication et la correction des bugs.
Les sessions ne peuvent pas vraiment être "marquées", car toutes les informations sont stockées sur le serveur.

Disons que **vous êtes un joueur d'échecs professionnel**, et que vous aimeriez **jouer avec plusieurs personnes en même temps**. Si vous essayez de vous souvenir de chaque jeu et de votre stratégie, vous atteindrez votre **limite** assez rapidement. Imaginez maintenant que vous ne vous souveniez de rien de ces parties, et que vous ne faisiez que relire l'échiquier à chaque coup. Vous pourriez jouer contre un million de personnes en même temps, sans aucune différence pour vous.

Maintenant, faisons l’analogie avec un serveur. Si votre application reçoit **beaucoup de charges,** vous devrez peut-être la distribuer sur d’autres serveurs. Si vous utilisiez des sessions, vous allez devoir utiliser un système de cache pour vos sessions, comme le fait « Memcached ». Le système deviendrait un peu plus complexe, mais vous perdriez votre système d’état.

Pour faire simple, **les sessions ne font plus ce pour quoi elles ont été initialement conçues** sans introduire de frais supplémentaires (un serveur "Memcached" ou autre à gérer). De plus, leur **comportement** peut **facilement être répliqué** en utilisant les cookies, la mise en cache du client et d'autres solutions de ce type qui stockent les données à la charge du client. Il y a, bien sûr, des situations dans lesquelles les sessions ont un sens, en particulier lorsque les serveurs veulent **stocker l'état** sans avoir la moindre possibilité de modifier les données des clients.

Par exemple, le protocole FTP est **"statefull"** pour une très bonne raison, car il **réplique les changements** à la fois du côté client et du côté serveur tout en offrant une sécurité accrue en raison de la nature de l'accès demandé. Ceci est faisable parce qu'**une seule personne** a besoin d'accéder à un seul serveur pour un seul transfert de données, même si le transfert implique plusieurs dossiers, fichiers et répertoires.
Ce n'est pas le cas avec un service comme une "Dropbox" partagée, dans laquelle les sessions "stateful" ajouteront une complexité sans apporter de valeur. Dans ce cas, le "stateless" serait un bien meilleur choix.

4.5 Conclusion

Vous connaissez la **différence** entre les **architectures "stateful"** et **"stateless"**.
Comprendre ce concept simple est la base sur laquelle la **plupart des architecture**s et des **conceptions actuelles** sont **basées**. Des concepts aussi évolués que le "RESTful design" sont basés sur ces idées. Vous avez maintenant toutes les connaissances vous permettant de **créer** un **socle de conception robuste** pour vos futurs projets basés sur une **architecture en microservices**.

Pour voir la fiche complète et les documents attachés, rendez-vous sur
https://elearning.26academy.com/course/play/5aba7221f2016f84c896ebdc